Learning to Map Sentences to Logical Form

by

Luke S. Zettlemoyer

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2009

Author .....
Department of Electrical Engineering and Computer Science
September 1, 2009

Certified by .....
Michael Collins
Associate Professor
Thesis Supervisor

Certified by .....
Leslie Pack Kaelbling
Professor
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Learning to Map Sentences to Logical Form

by

Luke S. Zettlemoyer

Submitted to the Department of Electrical Engineering and Computer Science
on September 1, 2009, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

One of the classical goals of research in artificial intelligence is to construct systems that automatically recover the meaning of natural language text. Machine learning methods hold significant potential for addressing many of the challenges involved with these systems. This thesis presents new techniques for learning to map sentences to logical form — lambda-calculus representations of their meanings.

We first describe an approach to the context-independent learning problem, where sentences are analyzed in isolation. We describe a learning algorithm that takes as input a training set of sentences labeled with expressions in the lambda calculus. The algorithm induces a Combinatory Categorial Grammar (CCG) for the problem, along with a log-linear model that represents a distribution over syntactic and semantic analyses conditioned on the input sentence.

Next, we present an extension that addresses challenges that arise when learning to analyze spontaneous, unedited natural language input, as is commonly seen in natural language interface applications. A key idea is to introduce non-standard CCG combinators that relax certain parts of the grammar — for example allowing flexible word order, or insertion of lexical items — with learned costs. We also present a new, online algorithm for inducing a weighted CCG.

Finally, we describe how to extend this learning approach to the context-dependent analysis setting, where the meaning of a sentence can depend on the context in which it appears. The training examples are sequences of sentences annotated with lambda-calculus meaning representations. We develop an algorithm that maintains explicit, lambda-calculus representations of discourse entities and uses a context-dependent analysis pipeline to recover logical forms. The method uses a hidden-variable variant of the perception algorithm to learn a linear model used to select the best analysis.

Experiments demonstrate that the learning techniques we develop induce accurate models for semantic analysis while requiring less data annotate effort than previous approaches.

Thesis Supervisor: Michael Collins
Title: Associate Professor

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor

Acknowledgments

I owe thanks to many people.

My advisors, Leslie Kaelbling and Michael Collins, have provided unique perspectives and invaluable input for the work described in this thesis and the many other fun projects we have worked on together. My additional committee members, Tomas Lozano-Perez and Fernando Pereira, provided valuable input and criticism.

I have had the pleasure to work with a number of wonderful people, including the many members of the LIS and NLP research groups. In particular, I would like to thank my direct collaborators: Regina Barzilay, Branavan, Harr Chen, Ashwin Deshpande, Michael Haimes, Brian Milch, Hanna Pasula, Krisitan Kersting, and Wei Lu; as well as my office mates: Meg Aycinena, Jenny Barry, Emma Brunskill, Brooke Cowan, Sarah Finney, Natalia Hernandez Gardiol, Kaijen Hsiao, Owen Macindoe, Nick Matsakis, James McLurkin, and Mike Ross.

As an undergraduate, I was first introduced to research by James Lester and Robert St. Amant, for which I am very grateful.

Most importantly, I want to thank my friends, including the many wonderful roommates I have had at the Bishop Allen Drive Cooperative; and family, especially my wife, Paulina.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the classical goals of research in artificial intelligence is to construct systems that automatically recover the meaning of natural language text. Modern linguistic theories for representing and constructing meaning build on the idea of compositional semantics first developed by Montague (1970b, 1970a, 1973). The meaning of a sentence is represented as a logical expression, a logical form, that is constructed from the meanings of the individual words it contains. These logical expressions are commonly written in the lambda calculus, which provides a unified mechanism for both representing the meanings of individual words and combining these meanings to build logical forms for complete sentences.

In this thesis, I develop new techniques for learning to map sentences to logical form — lambda-calculus representations of their meanings. Given a set of training examples containing sentences and their corresponding logical forms, the learning problem is to automatically induce a model that can be used to recover the meaning of new, previously unseen sentences. I describe an approach that incorporates ideas from recent linguistic formalisms that follow the Montague-style compositional semantics tradition (Carpenter, 1997; Steedman, 1996, 2000). I develop learning techniques for both the context-independent case, where sentences are analyzed in isolation, and the context-dependent setting, where we analyze sequences of sentences whose meanings can depend on the context in which they appear.

As an example in the context-dependent setting, consider the following sequence

of sentences annotated with logical forms (LFs). These sentences are typical of an interaction that might be seen in a natural language interface to a flight information database.

Sent. 1:	Show me flights from Boston to Seattle.
LF 1:	$\lambda x.flight(x) \wedge from(x, BOS) \wedge to(x, SEA)$
Sent. 2:	leaving on Friday.
LF 2:	$\lambda x.flight(x) \wedge from(x, BOS) \wedge to(x, SEA) \wedge day(x, FRI)$
Sent. 3:	Which one is the cheapest?
LF 3:	$argmin(\lambda x.flight(x) \wedge from(x, BOS) \wedge to(x, SEA) \wedge day(x, FRI)\ ,$ $\lambda y.cost(y))$

In this sequence, each logical form is a lambda-calculus expression that represents the meaning of the corresponding sentence. For example, LF 1 specifies a set of entities that are flights departing from Boston and arriving in Seattle. LF 3 is a quantified expression that considers a set of entities defined by an embedded lambda-calculus expression, and returns the one with the lowest cost.

Our goal is to learn a model that can be used to automatically convert each sentence to the corresponding logical form.

1.1 Problem Scope

A system that robustly maps sentences to logical form would be useful for a wide range of natural language tasks, such as natural language interfaces to databases or dialog systems. In these settings, the user provides natural language input, for example by asking questions or stating preferences. The system must automatically convert the user's statements into a formal meaning representation that can be used to further the current interaction.

The algorithms developed in this thesis are evaluated on the problem of natural language interfaces to databases (NLIDBs) where the user asks a series of questions that can be answered by listing facts from a database of interest. The sentences above

are a simple example of this type of interaction. However, we develop an approach that we hope with generalize to broader classes of natural language understanding problems.

NLIDBs are a natural application domain that has been studied at least since the early work of Woods (1968). Each sentence has a clearly defined meaning; the logical form defines a query that can be executed to recover the desired facts from the database. Additionally, there is considerable previous work on the problem, as we will discuss in Chapter 3, which provides a baseline for empirically evaluating performance.

Although the work in this thesis is evaluated by considering NLIDBs, we hope that the general approach will be applicable for other semantic analysis tasks. The style of analysis — constructing logical forms with a linguistically plausible grammar formalism — has been used for a number of different semantic analysis tasks. Early examples include the blocks world dialog system of Winograd (1970) and the work on story understanding by Charniak (1972). More recently, there have been efforts to build general purpose understanding systems that recover logical forms, including the core language engine (Alshawi, 1992) and a variety of approaches that use grammars written in logical programming languages such as Prolog (Pereira & Shieber, 1987).

As we will describe in more detail in Section 1.3, one limitation for these more general semantic analysis systems is the engineering effort required to build grammars for specific applications. Machine learning methods, such as those developed in this thesis, are primarily motivated by their ability to reduce the need for expensive manual grammar engineering. Although our focus is on learning grammars for NLIDBs, we expect that the learning techniques should generalize to other grammar formalisms designed for more general semantic analysis tasks.

1.2 Challenges

Building systems that automatically recover logical forms with high accuracy is a challenging task. Such systems must represent knowledge about language at multiple

levels and make decisions about how to best combine this information for each input sentence. There are at least three levels that must be considered:

- **Lexical:** What are the meanings of the individual words? For example, the word "flights" indicates that the predicate *flight* might appear in the logical form while the word "cheapest" is associated with an *argmin* construction that compares the *cost* value for some set of entities.

- **Sentential:** How do we combine the meaning of the words to construct a meaning for a complete sentence? For example, there are many possible ways of combining the lexical meanings of the words in a complex sentence such as "Show me flights from Boston and New York to San Francisco or Oakland that are nonstop." However, the system must select the one interpretation that matches the desired meaning.

- **Contextual:** How does the context in which a sentence appears change its meaning? For example, in a sentence such as "Which one is the cheapest?", the logical form is partially defined by a contextually appropriate set of entities that the word "one" references. A complete approach must be able to introduce contextual dependencies of this type.

There is ambiguity at each level. Words can have multiple meanings, these meanings can be combined in many ways, and there are many possible ways of resolving references to the context. Selecting the correct analysis is a significant challenge.

1.3 Previous Approaches

Traditional approaches for constructing systems that address all of these challenges require significant engineering effort. For example, the ATIS participants (e.g., (Seneff, 1992; Ward & Issar, 1994; Levin et al., 2000)) spent many man-years developing natural language interfaces for a flight reservation database. Other efforts, such as the core language engine (CLE) (Alshawi, 1992), spent years building broad-coverage

grammars for mapping to logical form, that could then be adapted to specific applications such as interfaces to databases. In this section, we discuss one representative ATIS system, the CMU Phoenix understanding module (Ward, 1991), and the CLE. In both cases, we will see that there is a strong need for developing learning techniques to ease the engineering effort required to build complete systems.

The CMU Phoenix system maps sentences to frames that represent their meaning. Each frame represents a possible user request and contains a number of slots that may be specified. For example, one frame might represent a request for a list of flights and include slots that specify desired flight properties, such as the origin and destination cities. Other frames might represent, for example, requests for the cheapest flight, lists of airlines, or airport information. The analysis problem is to select a frame and fill its slot values. The Phoenix system uses a set of hand-engineered finite state machines that perform pattern matching on the input sentence to select phrases that indicate which frame to select and how to fill its slots. Optionally, slot values can be copied from frames recovered from previous sentences, according to a set of hand-specifed rules. This type of approach can work well in practice, but has two main restrictions. First, specifying all of the state machines and rules required to perform complete analyses is very labor-intensive. Second, the range of possible logical forms is restricted by the set of possible frames.

The core language engine (CLE) effort was an attempt to engineer a broad-coverage grammar that can be used to construct unrestricted logical forms that specify a relatively complete meaning representation. For example, the logical forms include representations of tense, aspect, and modals (including knowledge and belief). The CLE includes a context-free grammar that builds logical forms in parallel with syntactic analysis. Lexical knowledge is encoded in a lexicon that specifies the part of speech and semantic content of individual words. Meanings are constructed for complete sentences with a deterministic parsing algorithm that uses hand-specified rules to exclude incorrect syntactic or semantic analyses. Finally, context-dependence is modeled with explicit referential expressions in the logical form that are replaced with semantic content from the context according to a hand-engineered set of reso-

lution rules. The CLE models an impressive range of semantic phenomena but can be difficult to apply to specific domains. The lexicon must be specified, along with domain-dependent rules for parse selection and context resolution.

The machine learning techniques we develop in this thesis are designed to alleviate the knowledge engineering required for these types of systems. The analysis framework we develop is inspired by the CLE-style approach of including a linguistically plausible grammar that can produce unrestricted logical forms. We learn a lexicon in a linguistically-motivated grammatical formalism along with probabilistic models for parsing and context dependence. However, we restricted our attention, for now, to a formalism that only models the types of semantic phenomena needed for natural language interfaces to databases (excluding, for example, verb tense and modals). We expect that the lessons learned for this restricted case will be useful for designed learning algorithms that induce grammars for more general semantic analysis problems.

1.4 Overview of Approach

In this thesis, I develop a machine learning approach for the problem of mapping sentences to logical form. The algorithm learns an extension of Combinatory Categorial Grammar (CCG) (Steedman, 1996, 2000) that includes a model for context-dependent analysis. The result is a unified approach that:

1. represents lexical semantics for individual words,

2. includes a probabilistic parsing model for analyzing individual sentences, and

3. includes a probabilistic model for reasoning about context dependence.

The rest of this section provides an overview of the primary contributions of this thesis. We first describe our approach to the context-independent learning problem, where sentences are analyzed in isolation. Next, we describe an extension that addresses challenges that arise when analyzing spontaneous, unedited input. Finally,

we describe an extension of the learning approach to the context-dependent setting, where the meaning of a sentence can depend on the context in which it appears.

1.4.1 Context-independent Learning

We will first consider the problem of context-independent learning. In this case, each training example is a single sentence paired with a lambda-calculus expression. For instance, one simple training example might be:

Sentence: Show me flights from Boston to Seattle.

Logical Form: $\lambda x.flight(x) \wedge from(x, BOS) \wedge to(x, SEA)$

As before, the lambda-calculus expression is the desired output that represents the underlying meaning of the input sentence.

The first contribution of this thesis is an algorithm for learning to map sentences to logical form. The algorithm takes as input a set of n training examples $\{(w_i, z_i)|i = 1 \ldots n\}$, where each example is a pair (w_i, z_i) containing a sentence w_i and its corresponding logical form z_i. The output is a Probabilistic Combinatory Categorial Grammar (PCCG) (Clark & Curran, 2003) that can be used to map new sentences to logical form.

A PCCG is a probabilistic generalization of Combinatory Categorial Grammars (CCGs) (Steedman, 1996, 2000). The use of PCCGs in our learning framework provides two key advantages. First, CCGs model a wide range of linguistic phenomena; in particular, they provide an integrated treatment of semantics and syntax that makes use of a compositional semantics based on the lambda calculus. Second, PCCGs provide a mechanism for coping with ambiguity in the underlying CCG. They include a log-linear model that defines a distribution over possible logical forms for a given sentence. This distribution gives us a principled method for selecting the best (highest probability) analysis. Chapter 2 includes a more detailed description of PCCGs.

During learning, two parts of the PCCG are induced: a lexicon Λ and a parameter vector θ. The lexicon contains a set of lexical items that pair individual words with CCG categories that define syntactic and semantic information. For example, one

lexical item might pair the word *flights* with the category $N : \lambda x.flight(x)$. This category specifies that the word's syntactic type is N (it is a noun) and that its meaning is the lambda-calculus expression $\lambda x.flight(x)$ (which defines a set of entities that are flights). The parameter vector θ defines the log-linear distribution $p(z|w;\theta)$ over possible logical forms z for each input sentence w. Together, the lexicon and parameter vector represent solutions to the lexical and sentential reasoning challenges described above: the lexicon represents the meanings of individual words and the parameter vector defines how to best combine these meanings to construct complete logical forms.

The learning problem as we have formulated it is particularly challenging because the derivation that maps each sentence to its logical form is not annotated in the training data. We do not label the lexical entries associated with the individual words or how they should be combined. Providing this information would be very labor-intensive; we have deliberately formulated the problem in a way that requires a relatively minimal level of annotation. This choice represents a major departure from previous work on learning CCGs and CRFs. We simultaneously solve both a *structure learning* problem (inducing the CCG lexicon) and a *hidden-variable estimation* problem (learning the parameters θ without access to full derivations).

1.4.2 Learning Relaxed Parsers

Learning a grammar in a detailed grammatical formalism such as CCG has the advantage that it allows a system to handle quite complex semantic effects, such as coordination or scoping phenomena. In particular, it allows us to leverage the considerable body of work on semantics within these formalisms, for example see Carpenter (1997). However, a grammar based on a formalism such as CCG can be somewhat rigid, and this can cause problems when a system is faced with spontaneous, unedited natural language input, as is commonly seen in natural language interface applications. For instance, consider the following example, which is typical of the type of sentences seen in the ATIS travel-planning domain (Dahl et al., 1994):

Sentence: Boston to Seattle the latest on Friday

Logical Form: $argmin(\lambda x.flight(x) \wedge from(x, BOS) \wedge to(x, SEA) \wedge day(x, FRI)$, $\lambda y.time(y))$

This sentence exhibits characteristics which present significant challenges to the previously described approach. For example, it has flexible word order, and includes telegraphic language where words are omitted.

We present a learning algorithm that retains the advantages of using a detailed grammar, but is also effective in dealing with phenomena seen in spontaneous natural language, as exemplified by the ATIS domain. The learning problem is the same as we described in the last section. The input is a training set of sentences paired with logical forms. The outputs are a CCG lexicon Λ and a parameter vector θ. However, we describe two contribution that are required to solve this learning problem when the sentences are spontaneous and unedited.

The first contribution is an extended parsing approach that introduces additional non-standard CCG operators. These operators relax certain parts of the grammar—for example allowing flexible word order, or insertion of lexical items. This approach has the advantage that it can be seamlessly integrated into the CCG learning algorithm described above. In particular, we can learn costs for applying each of the new, relaxed operations, thereby limiting their use only to the cases where they are useful for constructing the correct logical form.

A second contribution is a new, online algorithm for CCG learning. The approach involves perceptron training of a model with hidden variables. However it has the additional twist of also performing grammar induction (lexical learning) in an online (example-by-example) manner. In our experiments, we show that the new algorithm is considerably more efficient than the algorithm described in the previous section; this is important when training on large training sets, for example the ATIS data.

1.4.3 Context-dependent Learning

Finally, we extend our CCG learning approach to the full context-dependent learning problem. In this case, we must learn to interpret sentences whose underlying meanings can depend on the context in which they appear. For example, consider the interaction we saw earlier:

Sent. 1:	Show me flights from Boston to Seattle.
LF 1:	$\lambda x.flight(x) \wedge from(x, BOS) \wedge to(x, SEA)$
Sent. 2:	leaving on Friday.
LF 2:	$\lambda x.flight(x) \wedge from(x, BOS) \wedge to(x, SEA) \wedge day(x, FRI)$
Sent. 3:	Which one is the cheapest?
LF 3:	$argmin(\lambda x.flight(x) \wedge from(x, BOS) \wedge to(x, SEA) \wedge day(x, FRI)$, $\lambda y.cost(y))$

The meanings of the second and third sentences must be determined based on the context established by earlier sentences. For example, the fact that the flights should be from Boston and to Seattle is context-dependent.

Our final contribution is an algorithm for learning context-dependent mappings from sentences to logical form. Here, the learning problem is formulated differently; when analyzing a sentence, we need to consider the meanings of previous sentences. The training set $\{I_i | i = 1 \ldots n\}$ contains n interactions. Each interaction $I_i = \langle (w_{i,1}, z_{i,1}), \ldots, (w_{i,n_i}, z_{i,n_i}) \rangle$ is a sequence of n_i sentences paired with logical forms. For example, the sequence of sentences and logical forms presented above is one possible interaction. The algorithm learns a parameter vector θ that defines a weighted-linear model for the context-dependent analysis problem, as we describe in the rest of this section.

The context-dependent analysis problem is to construct a logical form that can depend on the current interaction. For step j of an interaction i, we define the context to be the sequence $\langle z_{i,1}, \ldots z_{i,j-1} \rangle$ of logical forms for all of the previous sentences. The analysis problem is to map the sentence $w_{i,j}$ and its context to the desired logical form $z_{i,j}$. We present an approach that uses a two-stage pipeline to

construct context-dependent logical forms. The first stage uses a probabilistic CCG to construct a context-independent, underspecified meaning representation. The second stage resolves this underspecified meaning representation by making a sequence of modifications to it that depend on the logical forms in the current context.

In general, there are a large number of possible context-dependent analyses for each sentence. To select the best one, we present a weighted linear model that is used to make a range of parsing and context-resolution decisions. Since the training data contains only the final logical forms, we model these intermediate decisions as hidden variables that must be estimated without explicit supervision. Here, again, we have made a deliberate decision to formulate the problem in a way that minimizes annotation effort but leads to a more challenging learning problem. We show that this model can be effectively trained with a hidden-variable variant of the perceptron algorithm.

1.5 Contributions

This thesis develops learning algorithms for mapping sentences to logical form in both the context-independent and context-dependent settings. The primary contributions include:

- An algorithm for context-independent learning that induces a linguistically motivated grammar, and estimates the parameters of a probabilistic parsing model.
- A relaxation of the CCG formalism for parsing spontaneous, unedited input.
- An online, error-driven learning algorithm that performs lexical induction and parameter estimation in an iterative, example-by-example manner.
- A context-dependent learning algorithm for constructing logical forms that can depend on the context established by the meanings of the previous sentences in the current interaction.

1.6 Thesis Outline

Chapter 2 provides background material, including a review of lambda calculus, CCG, weighted linear models, and the perceptron leaning algorithm. Chapter 3 describes related work for mapping sentences to logical form. Chapters 4-6 present algorithms for the three leaning problems outlined above. Finally, Chapter 7 concludes and describes possible future work.

Chapter 2

Background

This chapter gives background material underlying our learning approach. We first describe the lambda–calculus expressions used to represent logical forms. We then describe combinatory categorial grammar (CCG), and the extension of CCG to probabilistic CCGs (PCCGs) through weighted linear models. Finally, we describe an approach to parameter estimation with the structured perceptron algorithm.

2.1 Semantics

We represent logical forms with a lambda–calculus formalism similar to the one presented by Carpenter (1997). The system has three basic types: e, the type of entities; t, the type of truth values; and r, the type of real numbers. It also allows functional types, for example $\langle e, t \rangle$, which is the type assigned to functions that map from entities to truth values.

Figure 2-1 shows three questions about US geography and their associated logical forms. These sentences are from the Geo880 dataset, which we will describe in Chapter 4. Each logical form is an expression from the lambda calculus. The lambda–calculus expressions we use are formed from the following items:

- **Constants:** Constants can either be entities, numbers or functions. For example, *texas* is an entity (i.e., it is of type e). The constant *state* is a function

a) What states border Texas
$\lambda x.state(x) \wedge borders(x, texas)$

b) What is the largest state
$argmax(\lambda x.state(x), \lambda x.size(x))$

c) What states border the state that borders the most states
$\lambda x.state(x) \wedge borders(x, \arg\max(\lambda y.state(y),$
$\lambda y.count(\lambda z.state(z) \wedge borders(y, z))))$

Figure 2-1: Examples of sentences with their logical forms.

that maps entities to truth values, and is of type $\langle e, t \rangle$. The constant *size* is a function that maps entities to real numbers, and is therefore of type $\langle e, r \rangle$ (in the geography domain, *size(x)* returns the land-area of x).

- **Logical connectors:** The lambda–calculus expressions include conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), and implication ($\rightarrow$).

- **Quantification:** The expressions include universal quantification ($\forall$) and existential quantification ($\exists$). For example, $\exists x.state(x) \wedge borders(x, texas)$ is true if and only if there is at least one state that borders Texas. Expressions involving $\forall$ take a similar form.

- **Lambda expressions:** Lambda expressions represent functions. For example, $\lambda x.borders(x, texas)$ is a function from entities to truth values, which is true of those entities that border Texas.

- **Additional quantifiers:** The expressions involve the additional quantifying terms *count*, *argmax*, and *argmin*. An example of a count expression is $count(\lambda x.state(x))$, which returns the number of entities for which $state(x)$ is true. Argmax expressions are of the form $argmax(\lambda x.state(x), \lambda x.size(x))$. The first argument is a lambda expression denoting some set of entities; the second argument is a function of type $\langle e, r \rangle$. In this case the arg max operator would return the set of items for which $state(x)$ is true, and for which $size(x)$

takes its maximum value. Argmin expressions are defined analogously.

2.2 Combinatory Categorial Grammars

The parsing formalism underlying our approach is combinatory categorial grammar (CCG) (Steedman, 1996, 2000). A CCG specifies one or more logical forms—of the type described in the previous section—for each sentence that can be parsed by the grammar.

The core of any CCG is a lexicon, Λ. In a purely syntactic version of CCG, the entries in Λ consist of a word (lexical item) paired with a syntactic type. A simple example of a CCG lexicon is as follows:

$$\begin{array}{lcl} \text{Utah} & := & NP \\ \text{Idaho} & := & NP \\ \text{borders} & := & (S\backslash NP)/NP \end{array}$$

In this lexicon *Utah* and *Idaho* have the syntactic type NP, and *borders* has the more complex type $(S\backslash NP)/NP$. A syntactic type can be either one of a number of *primitive categories* (in the example, NP or S), or it can be a *complex type* of the form A/B or $A\backslash B$ where both A and B can themselves be a primitive or complex type. We use the primitive categories N, NP, and S, which stand for the linguistic notions of noun, noun-phrase, and sentence. Note that a single word can have more than one syntactic type, and hence more than one entry in the lexicon.

In addition to the lexicon, a CCG has a set of *combinatory rules*, which describe how adjacent syntactic categories in a string can be recursively combined. The simplest such rules are rules of *functional application*, defined as follows:

(1) *The functional application rules:*

a. $A/B \quad B \quad \Rightarrow \quad A \quad (>)$

b. $B \quad A\backslash B \quad \Rightarrow \quad A \quad (<)$

Intuitively, a category of the form A/B denotes a string that is of type A but is missing a string of type B to its right; similarly, $A\backslash B$ denotes a string of type A that

is missing a string of type B to its left.

The first rule says that a string with type A/B can be combined with a right-adjacent string of type B to form a new string of type A. As one example, in our lexicon, *borders*, (which has the type $(S\backslash NP)/NP$) can be combined with *Idaho* (which has the type NP) to form the string *borders Idaho* with type $S\backslash NP$. The second rule is a symmetric rule applying to categories of the form $A\backslash B$. We can use this to combine *Utah* (type NP) with *borders Idaho* (type $S\backslash NP$) to form the string *Utah borders Idaho* with the type S. We can draw a parse tree (or derivation) of *Utah borders Idaho* as follows:

$$\dfrac{\dfrac{\text{Utah}}{NP} \quad \dfrac{\dfrac{\text{borders}}{(S\backslash NP)/NP} \quad \dfrac{\text{Idaho}}{NP}}{(S\backslash NP)}>}{S}<$$

Note that we use the notation $>$ and $<$ to denote application of rules 1(a) and 1(b) respectively.

CCGs typically include a *semantic type*, as well as a syntactic type, for each lexical entry. For example, our lexicon would be extended as follows:

$$\begin{array}{lcl} \text{Utah} & := & NP : utah \\ \text{Idaho} & := & NP : idaho \\ \text{borders} & := & (S\backslash NP)/NP : \lambda x.\lambda y.borders(y, x) \end{array}$$

We use the notation $A : f$ to describe a category with syntactic type A and semantic type f. Thus *Utah* now has syntactic type NP, and semantic type *utah*. The functional application rules are then extended as follows:

(2) *The functional application rules (with semantics):*

a. $A/B : f \quad B : g \quad \Rightarrow \quad A : f(g) \quad (>)$

b. $B : g \quad A\backslash B : f \quad \Rightarrow \quad A : f(g) \quad (<)$

Rule 2(a) now specifies how the semantics of the category A is compositionally built out of the semantics for A/B and B. Our derivations are then extended to

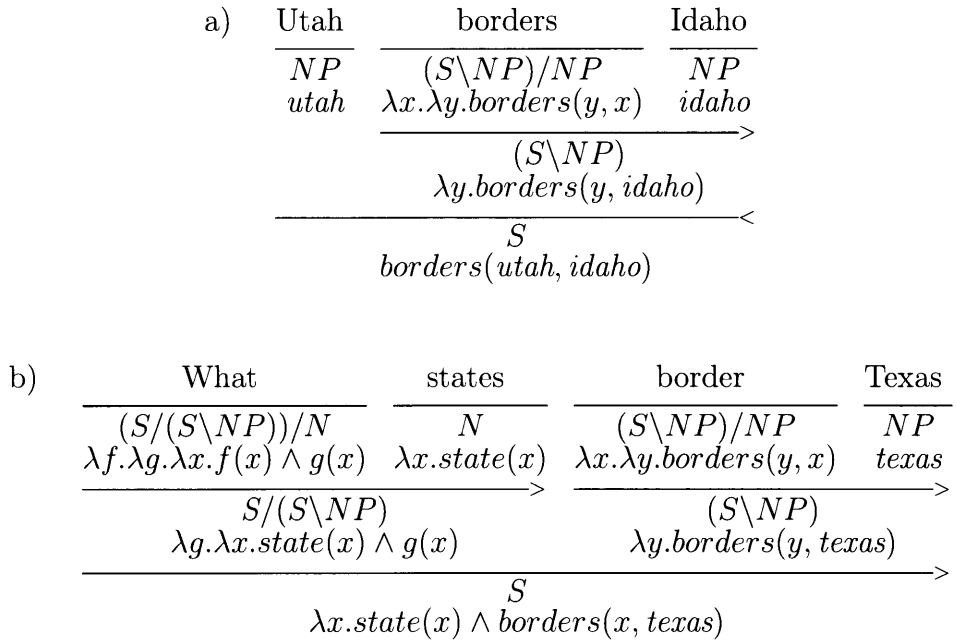


Figure 2-2: Two examples of simple CCG parses.

include a compositional semantics. See Figure 2-2(a) for an example parse. This parse shows that *Utah borders Idaho* has the syntactic type S and the semantics *borders(utah, idaho).*

A second set of combinators in CCG grammars are the rules of *functional composition*:

(3) *The functional composition rules (with semantics):*

a. $A/B : f \quad B/C : g \quad \Rightarrow \quad A/C : \lambda x.f(g(x)) \quad (> \mathbf{B})$

b. $B\backslash C : g \quad A\backslash B : f \quad \Rightarrow \quad A\backslash C : \lambda x.f(g(x)) \quad (< \mathbf{B})$

These rules allow for an unrestricted notion of constituency that is useful for modeling coordination and other linguistic phenomena. As we will see, they also turn out to be useful when modeling constructions with relaxed word order, as seen frequently in domains such as ATIS.

In spite of their relative simplicity, CCGs can capture a wide range of syntactic and semantic phenomena. As one example, see Figure 2-2(b) for a more complex parse. Note that in this case we have an additional primitive category, N (for nouns), and

the final semantics is a lambda expression denoting the set of entities that are states and that border Texas. In this case, the lexical item *what* has a relatively complex category, which leads to the correct analysis of the underlying string. Additionally, the parse in Figure 2-3 uses both functional application and functional composition rules. The backward functional composition rule is used to combine analyses for the substrings *from Dallas* and *to Washington.*[1]

A CCG grammar also includes type-raising and coordination combinators. The type-raising rules are instances of the general forms

(4) *The type-raising rules (with semantics):*

a. $A : g \Rightarrow T/(T\backslash A) : \lambda f.f(g) \quad (> \mathbf{T})$

b. $A : g \Rightarrow T\backslash(T/A) : \lambda f.f(g) \quad (< \mathbf{T})$

where T and A are CCG categories. We use a set of instances that were found to be useful on development sets. Specifically, we restrict T to be NP and let A be S, $S\backslash NP$, S/NP, or $N\backslash N$.

Finally, there is a combinator that models coordination

(5) *The coordination rule (with semantics):*

$$A : g \quad CONJ : b \quad A : g \Rightarrow A : \lambda \ldots b(f \ldots)(g \ldots) \quad (\mathbf{T})$$

This rule is a general template that combines any two categories of the same type to produce a new category with the appropriate conjunction or disjunction b.

These addition rules provide a unified treatment of complex coordination phenomena. For example, Figure 2-4 shows a parse of the phrase *states that border Texas and Idaho* that uses type-raising and coordination rules. In this parse, the lexical categories for the two noun phrases (*Texas* and *Idaho*) are type raised and then coordinated to form a single category. That category then takes the lexical category for the verb *borders* as an argument and produces the desired meaning. The ability to build logical forms for phrases of this type is one of the advantages of adopting the CCG formalism.

[1]Note that in this example a more conventional parse is also possible, where the two prepositional phrases each modify *flight*, and only the functional application rules are used. We have included the given parse to illustrate the use of functional composition rules.

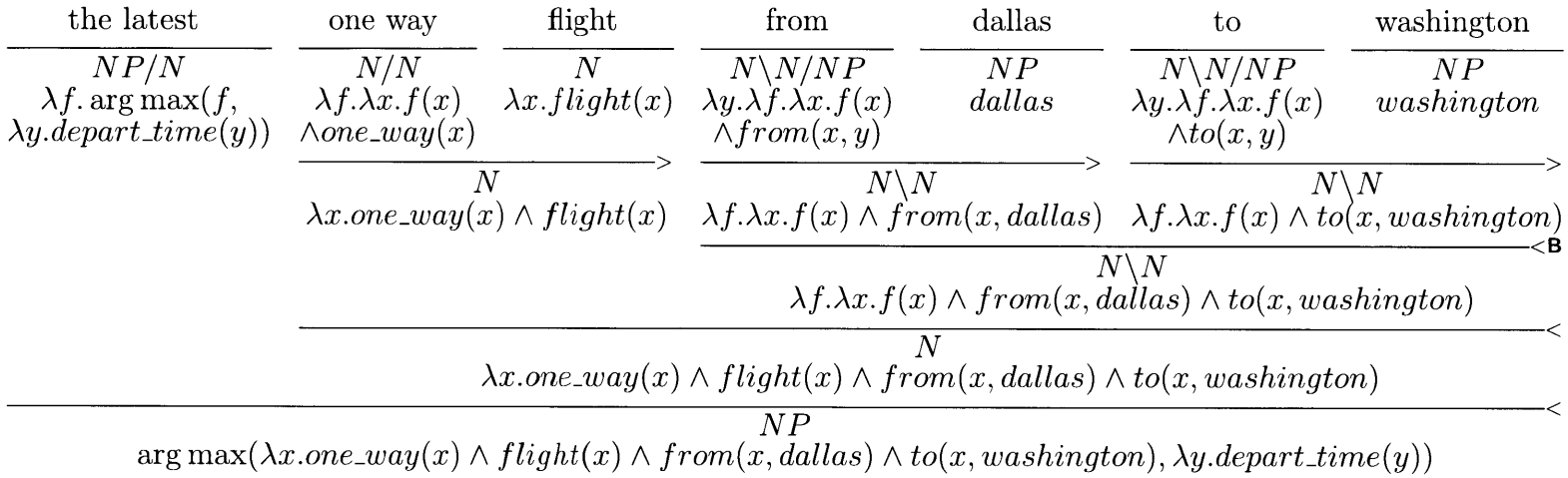


Figure 2-3: A CCG parse built with the application and composition rules.

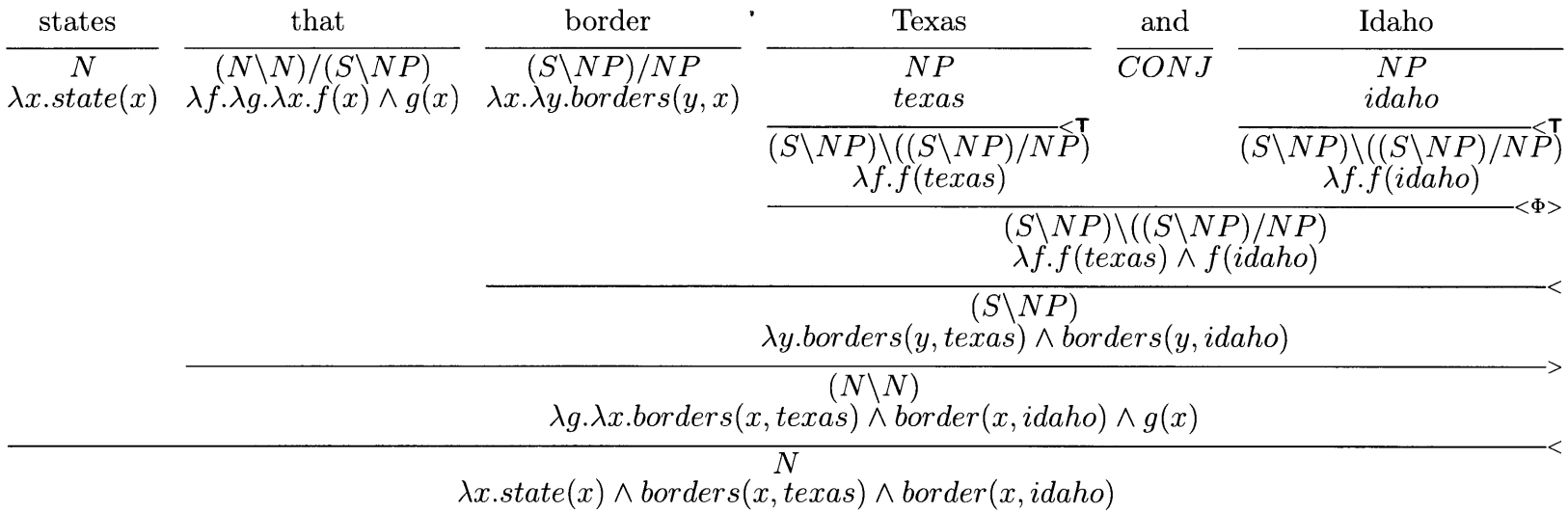


Figure 2-4: A CCG parse with type raising and coordination.

In our work, we make use of the application, composition, coordination, and type-raising rules, as described above. In addition, we allow lexical entries consisting of strings of length greater than one, for example

the Mississippi $:=$ $NP : mississippi\_river$

This leads to a relatively minor change to the formalism, which in practice can be very useful. For example, it is easier to directly represent the fact that *the Mississippi* refers to the Mississippi river with the lexical entry above than it is to try to construct this meaning compositionally from the meanings of the determiner *the* and the word *Mississippi*, which refers to the state of Mississippi when used without the determiner.

2.3 Structured Classification and Weighted Linear Models

In this thesis, we cast the CCG learning problem as a type of structured classification problem. Structured classification tasks involve the prediction of output labels y from inputs x in cases where the output labels have rich internal structure. Previous work in this area has focused on problems such as sequence learning, where y is a sequence of state labels (e.g., see (Lafferty, McCallum, & Pereira, 2001; Taskar, Guestrin, & Koller, 2003)), or natural language parsing, where y is a context-free parse tree for a sentence x (e.g., see Taskar et al. (2004)).

Weighted linear models are a simple and effective way to represent structured classification problems. These models have three main components:

- A generator function GEN(x) that defines the output space for each input x. Each $y \in \text{GEN}(x)$ is a potential output.

- A feature function $\phi(x, y) \in \mathbb{R}^d$ that defines an embedding for the input, output pair (x, y). We will refer to each dimension of $\phi(x, y)$ as a feature and say that the complete model has d individual features.

- A parameter vector $\theta \in \mathbb{R}^d$ that is the same dimension as the feature function.

We will call each entry in this vector a parameter and say that the complete model has d parameters.

Given a weighted linear model, the prediction problem is to select the best output for a given input. This optimal output $y^*(x)$ is defined to be:

$$y^*(x) = \arg \max_{y \in \text{GEN}(x)} \theta \cdot \phi(x, y) \ ,$$

where we simply select the output y that achieves the highest score according to the dot product $\theta \cdot \phi(x, y)$. Intuitively, this means that each parameter is associated with a single feature and the overall score is just the sum of the products of each feature with its associated parameter.

The second major problem for structured classification is how to set the parameters θ given a training set of example input, output pairs. We will describe one approach for this problem in Section 2.4

Weighted Linear CCGs Given the general framework for structured classification, we can now describe a weighted linear model for CCG parsing. This model is similar to several other approaches (Ratnaparkhi et al., 1994; Johnson et al., 1999; Lafferty et al., 2001; Collins, 2004; Taskar et al., 2004).

The input x is a sentence and the output y is a CCG parse for x. The generator function $\text{GEN}(x)$ returns all possible CCG parses for x. In general, the set of parses is determined by the CCG lexicon Λ. We will make this fact explicit by writing $\text{GEN}(x; \Lambda)$ whenever we refer to the generator function. The feature function $\phi(x, y)$ is now a function of a sentence and a CCG parse tree. In principle, ϕ could include features that are sensitive to arbitrary sub-structures within the pair (x, y). Finally, θ is a parameter vector.

In general, the number of possible CCG parses in $\text{GEN}(x; \Lambda)$ that must be considered when computing the best parse $y^*(x)$ is exponential in the sentence length. This is a standard challenge in parsing. Efficient parsing algorithms exist when we restrict the form of the features in $\phi(x, y)$ to only test local aspects of the parse tree.

Examples of local features include ones that count the number of lexical entries of a particular type, or ones that count the number of applications of a particular CCG combinator. In our experiments we will use a parsing algorithm that is similar to a CKY-style parser with dynamic programming. Dynamic programming is used but each entry in the chart maintains a full semantic expression, preventing a polynomial-time algorithm; beam search is used to make the approach tractable. Appendix A provides the complete details of this algorithm.

Training a weighted linear CCG model involves learning the parameters θ and potentially also the lexicon Λ. As we will see, this thesis provides a method for learning a (θ, Λ) pair from a training set of sentences paired with lambda-calculus expressions.

2.4 The Structured Perceptron Algorithm

Given a weighted linear model, the parameter estimation problem is to find a parameter vector θ from a training set $\{(x_i, y_i) : i = 1 \ldots n\}$ containing example inputs x_i and their cooresponding outputs y_i. In this section, we review the structured perceptron algorithm (Collins, 2002), which we will extend later in this thesis. This algorithm provides a simple and effective solution to the parameter estimation problem for weighted linear models.

Figure 2-5 presents the complete structured perceptron algorithm. Learning is online and error-driven. The algorithm considers each training example and computes the best output under the current model. If this output if incorrect, a simple additive update is applied to the parameters. This update will increase the score associated with the correct output, which was not selected by the current model, and decrease the score of the incorrectly selected output. This process continues for T passes over the training data.[2]

This algorithm has a number of theoretical guarantees that apply to its conver-

[2]In practice, T can be a relatively small value that is usually set with experiments on development data.

Inputs:

- Training examples $\{(x_i, y_i) : i = 1 \ldots n\}$.
- Number of training iterations, T.

Definitions:

- The function $\phi(x, y)$ represents the features described in section 2.3.

Initialization:

Set parameters θ to zero.

Algorithm:

- For $t = 1 \ldots T, i = 1 \ldots n$:
  - Let $y' = \arg\max_{y \in \text{GEN}(x_i)} \theta \cdot \phi(x_i, y)$.
  - If $y' \neq y_i$:
    - Set $\theta = \theta + \phi(x_i, y_i) - \phi(x_i, y')$.

Output: Parameters θ.

Figure 2-5: The structured perceptron algorithm.

gence and generalization behavior. These guarantees are based on a notion of *linearly separable* data. The training set is linearly separable when there exists a setting of the parameters θ that correctly predicts all of the labeled outputs — for all i, $y^*(x_i) = y_i$. In this case, the algorithm is guaranteed to find parameter settings that separate the training data. A similar notion of nearly separable data can be defined for cases where there exists a θ that makes only a few errors on the training set. See the discussion by Collins (2002) for the formal details. In practice, even when these conditions do not hold, the algorithm can perform well.

Weighted CCG Learning In this thesis, we will develop a learning approach that incorporates many ideas from the structured perceptron algorithm. However, we will address two new challenges:

- **Lexical Learning:** The structured perceptron algorithm assumes that the function $\text{GEN}(x)$ is fixed and known a priori. However, for a weighted CCG the function $\text{GEN}(x; \Lambda)$ that defines the set possible CCG parse trees depends on the CCG lexicon Λ. We must simultaneously induce the lexicon Λ and estimate the parameters θ.

- **Hidden Variables:** The structured perceptron algorithm assumes that each training example is an (x, y) pair. For weighted CCG learning, y would need to be the entire CCG parse tree. Instead, we will develop an approach in which each training example is a pair (x, z), where z is a logical form. In general, there can be many parse trees that produce the same logical form. We will treat the selection of the correct parse tree as a hidden variable that must be estimated at training time.

We will develop an algorithm that addresses these challenges while maintaining the online, error-driven properties of the original approach.

Chapter 3

Related Work

This chapter describes several lines of related research. First, we review work in the CCG literature. We then describe work that directly addresses the problem of mapping sentences to meaning representations, including both the context-independent and context-dependent cases. Finally, we describe related work that focuses on computational models of child language learning.

3.1 Combinatory Categorial Grammars

We build directly on research from the CCG literature. Steedman (1996, 2000) provides a comprehensive overview of CCG. Although we use a relatively simple CCG setup, the formalism has many extensions that could be incorporated into our approach. For example, Baldridge (2002) describes a multi-modal generalization that allows more fine-grained lexical control of parsing operations and Bozsahin (1998) describes extensions for modeling languages with free word order.

The majority of work on machine learning for CCG has focused on the problem of broad-coverage syntactic parsing. This work requires a training corpus of sentences paired with CCG parse trees. Hockenmaier and Steedman (2007) describe CCGBank, a version of the Penn Treebank (Marcus, Marcinkiewicz, & Santorini, 1993) that is annotated with full CCG syntactic parse trees. This data set has enabled work on probabilistic models for syntactic parsing with CCG (Hockenmaier & Steedman, 2002;

Clark & Curran, 2007). The model developed by Clark and Curran has the same log-linear form as the models we use. However, our focus is on learning to recover logical forms, which they do not consider.

There has also been some work on unsupervised learning of categorial grammars. Watkinson and Manandhar (1999) describe an approach for learning the syntactic categories of words given a corpus of sentences. This approach assumes a fixed set of possible categories and induces a lexicon that can be used to parse all of the training examples. However, it is evaluated on a relatively small set of simple sentences and more work would be required to scale the approach larger data sets. Villavicencio (2001) describes an approach for categorial grammar induction that models child language learning, which we will describe in Section 3.4.

We know of only one approach (Bos, Clark, Steedman, Curran, & Hockenmaier, 2004) for learning CCG grammars that performs broad-coverage semantic analysis. Here the learning is fully supervised. The authors assume access to the CCGBank. They annotate the meaning of a small set of lexical items and demonstrate that the approach achieves high coverage when building logical forms for new sentences. However, there are no gold-standard logical form annotations for the evaluation sentences so there is no way to determine the accuracy of the approach. Scaling our approach to this type of analysis task is an important area for future work.

3.2 Context Independent Analysis

There has been a significant amount of previous work on learning to map isolated sentences to their underlying semantic representations. Most of this work can be divided into two categories, based on the training sets that were used. One line of work has developed techniques that are applied to data sets developed by Raymond Mooney and his students (Zelle & Mooney, 1996; Tang & Mooney, 2001), including the geography and jobs database domains we consider. Another line of work has looked at sentences from the ATIS travel planning domain (Dahl et al., 1994). There is relatively little work that has considered both.

Zelle and Mooney (1996) developed one of the earliest examples of a learning system for semantic analysis. This work made use of a deterministic shift–reduce parser and developed a learning algorithm, called CHILL, based on techniques from Inductive Logic Programming, to learn control rules for parsing. The major limitation of this approach is that it does not learn the lexicon, instead assuming that a lexicon that pairs words with their semantic content (but not syntax) has been created in advance. Later, Thompson and Mooney (2002) developed a system that learns a lexicon for CHILL that performed almost as well as the original system. Tang and Mooney (2001) developed a statistical shift–reduce parser that significantly outperformed these original systems. However, this system, again, does not learn a lexicon.

After this original line of work, a number of different approaches were developed for the problem.

Wong and Mooney (2006, 2007b) developed an approach that incorporates techniques previously used for statistical machine translation. They used word alignment techniques to induce a lexicon and synchronous grammars to represent the correspondence between sentences and logical form. One advantage of this model is that it can be inverted and used for natural language generation (Wong & Mooney, 2007a).

Kate and Mooney (2006, 2007b) describe techniques that incorporate support vector machine learning. They describe a method for training local classifiers that are used to make decisions during semantic parsing. One advantage of this approach is that it is easily generalized to the semi-supervised learning setting by incorporating ideas from work on transductive SVMs (Joachims, 1999).

Ge and Mooney (2005, 2006, 2009) have developed techniques that incorporate ideas from the syntactic parsing literature. They describe a joint model of both Penn-treebank-style syntactic analyses and semantic analyses. In particular, they show that access to a high-quality syntactic parser is useful for learning to perform semantic analysis with fewer annotated training examples.

Lu et al. (2008) present an algorithm for learning a joint generative model of sentences and meaning representations. The model generates hybrid trees that contain both words and meaning representation symbols. The authors develop a learning

algorithm that is a variant of the inside-outside algorithm. They demonstrate empirically that the approach induces hybrid tree distributions with no explicit grammar that can significantly improve recall.

There has also been work on learning to analyze isolated sentences from the ATIS data. Some early approaches applied the IBM translation models to the problem of filling the slots of semantic frames (Papineni, Roukos, & Ward, 1997; Ramaswamy & Kleindienst, 2000).

More recently, He and Young (2005) describe an algorithm that learns a probabilistic push-down automaton that models hierarchical dependencies but can still be trained on a data set that does not have full treebank-style annotations. One advantage of this approach is that it is robust to errors when integrated with a speech recognition system (He & Young, 2006).

3.3 Context Dependent Analysis

There has been a significant amount of work on hand engineering natural language interfaces to databases. There were a large number of successful systems developed for the original ATIS task and other related tasks (e.g., (Carbonell & Hayes, 1983; Seneff, 1992; Ward & Issar, 1994; Levin et al., 2000; Popescu, Armanasu, Etzioni, Ko, & Yates, 2004)). Androutsopoulos, Ritchie, and Thanisch (1995) provide a comprehensive summary of this work.

Recent work in this area has focused on improved parsing techniques and designing grammars that can be ported easily to new domains. Popescu et al. (2004) describe an approach for automatically constructing an analysis system given access only to the underlying database. The resulting system achieves high accuracy on a well defined subset of sentence types but does not attempt to analyze more complex sentences.

We are only aware of one system that learns to construct context-dependent interpretations (Miller, Stallard, Bobrow, & Schwartz, 1996). The Miller et al. (1996) approach is fully supervised and produces a final meaning representation in SQL. It requires complete annotation of all of the syntactic, semantic, and discourse decisions

required to correctly analyze each training example and uses these annotations to learn individual decision trees for each possible choice point.

3.4 Computational Models of Language Acquisition

Our approach learns to recover syntactic knowledge about language given only sentences and representations of their meaning. Given this formulation, it is related to work on computational models of child language learning.

Siskind (1996) presents an algorithm that learns word-to-meaning mappings from child-directed sentences that are paired with a set of possible meaning representations. The central idea is that children observe language that is paired with a set of different possible meanings and must decide which one provides the correct supervision when learning the meaning of individual words. Kate and Mooney (2007a) describe an algorithm for a similar ambiguous learning setup that extends their earlier work on semantic parsing described above.

Villavicencio (2001) describes an approach for modeling child language learning with categorial grammars. This approach uses a universal grammar that includes a relatively small set of parameters that must be set to learn each new language. The semantic learning setting is, again, one with ambiguous supervision. However, the approach also learns syntactic properties of the language represented in the sentences, such as the word order.

Exploring extensions of our CCG learning framework to these types of learning problems is an important area for future work.

Chapter 4

Structured Classification with Probabilistic Categorial Grammars

This chapter addresses the problem of mapping natural language sentences to lambda–calculus encodings of their meaning. We describe a learning algorithm that takes as input a training set of sentences labeled with expressions in the lambda calculus. The algorithm induces a grammar for the problem, along with a log-linear model that represents a distribution over syntactic and semantic analyses conditioned on the input sentence. When applied to the task of learning natural language interfaces to databases, the method learns parsers that outperform previous approches in two benchmark database domains. This chapter is based on work originally described in (Zettlemoyer & Collins, 2005).

4.1 Introduction

Recently, a number of learning algorithms have been proposed for *structured classification* problems. Structured classification tasks involve the prediction of output labels y from inputs x in cases where the output labels have rich internal structure. Previous work in this area has focused on problems such as sequence learning, where y is a sequence of state labels (e.g., see (Lafferty et al., 2001; Collins, 2002; Taskar et al., 2003)), or natural language parsing, where y is a context-free parse tree for a

sentence x (e.g., see Taskar et al. (2004)).

In this chapter we investigate a new type of structured classification problem, where the goal is to learn to map natural language sentences to a lambda-calculus encoding of their semantics. As one example, consider the following sentence paired with a logical form representing its meaning:

Sentence: what states border texas

Logical Form: $\lambda x.state(x) \wedge borders(x, texas)$

The logical form in this case is an expression representing the set of entities that are states, and that also border Texas. The training data in our approach consists of a set of sentences paired with logical forms, as in this example.

This is a particularly challenging problem because the derivation from each sentence to its logical form is not annotated in the training data. For example, there is no direct evidence that the word *states* in the sentence corresponds to the predicate *state* in the logical form; in general there is no direct evidence of the syntactic analysis of the sentence. Annotating entire derivations underlying the mapping from sentences to their semantics is highly labor-intensive. Rather than relying on full syntactic annotations, we have deliberately formulated the problem in a way that requires a relatively minimal level of annotation.

Our algorithm automatically induces a grammar that maps sentences to logical form, along with a probabilistic model that assigns a distribution over parses under the grammar. The grammar formalism we use is combinatory categorial grammar (CCG) (Steedman, 1996, 2000). CCG is a convenient formalism because it has an elegant treatment of a wide range of linguistic phenomena; in particular, CCG has an integrated treatment of semantics and syntax that makes use of a compositional semantics based on the lambda calculus. We use a log-linear model—similar to models used in conditional random fields (CRFs) (Lafferty et al., 2001)—for the probabilistic part of the model. Log-linear models have previously been applied to CCGs by Clark and Curran (2003), but our work represents a major departure from previous work on CCGs and CRFs, in that *structure learning* (inducing an underlying discrete structure,

i.e., the grammar or CCG lexicon) forms a substantial part of our approach.

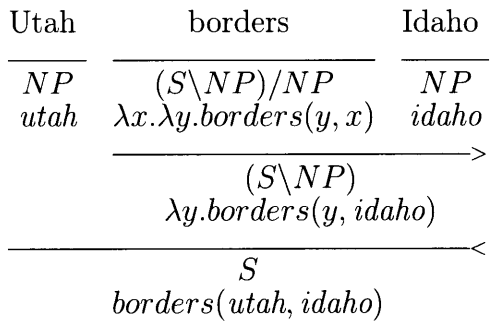
Mapping sentences to logical form is a central problem in designing natural language interfaces. We describe experimental results on two database domains: Geo880, a set of 880 queries to a database of United States geography; and Jobs640, a set of 640 queries to a database of job listings. Tang and Mooney (2001) described previous work on these data sets. Previous work by Thompson and Mooney (2002) and Zelle and Mooney (1996) used a subset of the Geo880 corpus. We evaluated the algorithm's accuracy in returning entirely correct logical forms for each test sentence. Our method achieves over 95% precision on both of these domains, with recall of 79% on each domain. These are highly competitive results when compared to the previous work.

4.2 Background

In this section, we describe a probabilistic extension of CCGs. This approach is closely related to the weighted linear models we saw in Section 2.3, the key distinction being the that score of each parse is exponentiated and normalized to define a valid probability distribution over parse trees.

4.2.1 Probabilistic CCGs

We now describe how to generalize CCGs to probabilistic CCGs (PCCGs). A CCG, as described in Section 2.2, will generate one or more derivations for each sentence w that can be parsed by the grammar. We will describe a derivation as a pair (z, y), where z is the final logical form for the sentence and y is the sequence of steps taken in deriving z. For example, in the following simple derivation:



the final logical form z is $borders(utah, idaho)$ and the derivation y is the CCG parse used to construct it. A PCCG defines a conditional distribution $P(z, y|w)$ over possible (z, y) pairs for a given sentence w.

In general, various sources of ambiguity can lead to a sentence w having more than one valid (z, y) pair. This is the primary motivation for extending CCGs to PCCGs: PCCGs deal with ambiguity by ranking alternative parses for a sentence in order of probability. One source of ambiguity is lexical items having more than one entry in the lexicon. For example, *New York* might have entries $NP : new\_york\_city$ and $NP : new\_york\_state$. Another source of ambiguity is where a single logical form z may be derived by multiple derivations y. This latter form of ambiguity can occur in CCG, and is often referred to as *spurious ambiguity*; the term *spurious* is used because the different syntactic parses lead to identical semantics.

In defining PCCGs, we make use of a conditional log-linear model that is similar to the model form in conditional random fields (CRFs) (Lafferty et al., 2001) or log-linear models applied to parsing (Ratnaparkhi et al., 1994; Johnson et al., 1999; Clark & Curran, 2003). We assume a function ϕ mapping (z, y, w) triples to feature vectors in $\mathbb{R}^d$. This function is defined by d individual features, so that $\phi(z, y, w) = \langle \phi_1(z, y, w), \ldots, \phi_d(z, y, w) \rangle$. Each feature ϕ_j is typically the count of some sub-structure within (z, y, w). The model is parameterized by a vector $\theta \in \mathbb{R}^d$. The probability of a particular (syntax, semantics) pair is defined as

$$P(z, y|w; \theta) = \frac{e^{\phi(z,y,w)\cdot\theta}}{\sum_{(z,y)} e^{\phi(z,y,w)\cdot\theta}} \tag{4.1}$$

The sum in the denominator is over all valid parses for w under the CCG grammar.

4.2.2 Parsing and Parameter Estimation

We now turn to issues of parsing and parameter estimation. Parsing under a PCCG involves computing the most probable logical form z for a sentence w,

$$\arg\max_z P(z|w;\theta) = \arg\max_z \sum_y P(z,y|w;\theta)$$

where the arg max is taken over all logical forms z and the hidden syntax y is marginalized out by summing over all parses that produce z. We use dynamic programming algorithms for this step, which are very similar to CKY–style algorithms for parsing probabilistic context-free grammars (PCFGs).[1] Dynamic programming is feasible within our approach because the feature-vector definitions $\phi(z,y,w)$ involve *local* features that keep track of counts of lexical items in the derivation y.[2] Appendix A provides a detailed discussion of the parsing algorithm.

In parameter estimation, we assume that we have n training examples, $\{(w_i, z_i) : i = 1 \ldots n\}$. Each w_i is a sentence in the training set, and z_i is the lambda-calculus expression associated with that sentence. The task is to estimate the parameter values θ from these examples. Note that the training set does not include derivations y_i, and we therefore view derivations as hidden variables within the approach. The log-likelihood of the training set is given by:

$$\begin{aligned} O(\theta) &= \sum_{i=1}^{n} \log P(z_i|w_i;\theta) \\ &= \sum_{i=1}^{n} \log \left(\sum_y P(z_i,y|w_i;\theta) \right) \end{aligned}$$

[1] CKY–style algorithms for PCFGs (Manning & Schutze, 1999) are related to the Viterbi algorithm for hidden Markov models, or dynamic programming methods for Markov random fields.

[2] We use beam–search during parsing, where low-probability sub-parses are discarded at some points during parsing, in order to improve efficiency.

Differentiating with respect to θ_j yields:

$$\frac{\partial O(\theta)}{\partial \theta_j} = \sum_{i=1}^{n} \sum_{y} \phi_j(z_i, y, w_i) P(y|w_i, z_i; \theta) - \sum_{i=1}^{n} \sum_{z,y} \phi_j(z, y, w_i) P(z, y|w_i; \theta)$$

The two terms in the derivative involve the calculation of expected values of a feature under the distributions $P(y|w_i, z_i; \theta)$ or $P(y, z|w_i; \theta)$. Expectations of this type can again be calculated using dynamic programming, using a variant of the inside-outside algorithm (Baker, 1979), which was originally formulated for probabilistic context-free grammars.

Given this derivative, we can use it directly to maximize the likelihood using a stochastic gradient ascent algorithm (LeCun, Bottou, Bengio, & Haffner, 1998),[3] which takes the following form:

Set θ to some initial value
for $k = 0 \ldots N-1$
  for $i = 1 \ldots n$
    $\theta = \theta + \frac{\alpha_0}{(1+ct)} \frac{\partial \log P(z_i|w_i;\theta)}{\partial \theta}$

where $t = i + k \times n$ is the total number of previous updates and N is a parameter that controls the number of passes over the training data. The learning-rate parameters α_0 and c ensure convergence by decaying the magnitude of the updates over time. They are set empirically based on experiments with development data.

4.3 Learning

In the previous section we saw that a probabilistic Combinatory Categorial Grammar (PCCG) is defined by a lexicon Λ, together with a parameter vector θ. This

[3]The EM algorithm could also be used, but would require some form of gradient ascent for the M-step because it cannot be computed in closed form. Because of this, we found it simpler to use gradient ascent for the entire optimization.

section describes an algorithm that learns a weighted CCG. The algorithm requires two inputs:

- A training set of n examples, $\{(w_i, z_i) : i = 1 \ldots n\}$, where each training example is a sentence w_i paired with a logical form z_i.
- An initial lexicon, Λ_0, as described in Section 4.4.

The training data includes neither direct evidence about the parse trees mapping each w_i to z_i, nor the set of lexical entries which are required for this mapping. We treat the parse trees as a hidden variable within the model. The set of possible parse trees for a sentence depends on the lexicon, which is itself learned from the training examples. Thus, at a high level, learning will involve the following two sub-problems:

- Induction of a lexicon, Λ, which defines a set of parse trees for each training sentence w_i.
- Estimation of parameter values θ, which define a distribution over parse trees for any sentence.

The first problem can be thought of as a form of *structure learning*, and is a major focus of the current section. The second problem is a more conventional *parameter estimation* problem that can be solved with the stochastic gradient ascent algorithm described in the last section.

The remainder of this section describes an overall strategy for these two problems. The approach interleaves a structure-building step, GENLEX, with an online parameter estimation step, in a way that results in a PCCG with a compact lexicon and effective parameter estimates for the weights of the log-linear model.

Section 4.3.1 describes the main structural step, GENLEX(w, z), which generates a set of candidate lexical items that may be useful in deriving a logical form z from a sentence w. Section 5.4 describes the overall learning algorithm, which prunes the lexical entries suggested by GENLEX and estimates the parameters of the log-linear model.

4.3.1 Lexical Learning

This section describes the function GENLEX, which takes a sentence w and a logical form z and generates a set of lexical items. The goal is to define GENLEX(w, z) in such a way that the set of lexical items that it generates allows at least one parse of w that results in z.

As an example, consider the input sentence *Utah borders Idaho* paired with the output logical form $borders(utah, idaho)$. We would like to induce a lexicon that can be used to construct the parse tree:

$$\dfrac{\dfrac{\dfrac{\text{Utah}}{\substack{NP \\ utah}} \quad \dfrac{\text{borders}}{\substack{(S\backslash NP)/NP \\ \lambda x.\lambda y.borders(y,x)}} \quad \dfrac{\text{Idaho}}{\substack{NP \\ idaho}}}{\dfrac{}{\substack{(S\backslash NP) \\ \lambda y.borders(y, idaho)}}>}}{\dfrac{}{\substack{S \\ borders(utah, idaho)}}<}$$

which is a standard CCG analysis that produces the desired logical form.

To achieve this goal, we need GENLEX to produce a lexicon that includes the three lexical items that were used in this parse, namely

$$\begin{aligned} \text{Utah} &:= NP : utah \\ \text{Idaho} &:= NP : idaho \\ \text{borders} &:= (S\backslash NP)/NP : \lambda x.\lambda y.borders(y, x) \end{aligned}$$

where we are modeling the word *borders* as a transitive verb that takes two arguments and the words *Utah* and *Texas* as noun phrases that can be used to satisfy these arguments.

As defined in this section, GENLEX will also produce *spurious* lexical items, such as $borders := NP : idaho$ and $borders\ utah := (S\backslash NP)/NP : \lambda x.\lambda y.borders(y, x)$. Later, we will see how these items can be pruned from the lexicon in a later stage of processing.

To compute GENLEX, we make use of a function, $C(z)$, that maps a logical form to a set of categories (such as $NP : utah$, or $NP : idaho$). GENLEX is then defined as

$$\text{GENLEX}(w, z) = \{x := y \mid x \in W(w), y \in C(z)\}$$

where $W(w)$ is the set of all subsequences of words in w.

The function $C(z)$ is defined through a set of rules that examine z and produce categories based on its structure. Figure 4-1 shows the rules. Each rule consists of a *trigger* that identifies some sub-structure within the logical form z. For each sub-structure in z that matches the trigger, a category is created and added to $C(z)$. As one example, the second row in the table defines a rule that identifies all arity-one predicates p within the logical form as triggers for creating a category $N : \lambda x.p(x)$. Given the logical form $\lambda x.major(x) \wedge city(x)$, which has the arity-one predicates *major* and *city*, this rule would create the categories $N : \lambda x.major(x)$ and $N : \lambda x.city(x)$.

Intuitively, each of the rules in Figure 4-1 corresponds to a different linguistic sub-category such as noun, transitive verb, adjective, and so on. For example, the rule in the first row generates categories that are noun phrases, and the second rule generates nouns. The end result is an efficient way to generate a large set of linguistically plausible categories $C(z)$ that could be used to construct a logical form z.

4.3.2 The Learning Algorithm

Figure 4-2 shows the learning algorithm used within our approach. The output of the algorithm is a PCCG, defined by a lexicon Λ and a parameter vector θ. As input, the algorithm takes a training set of sentences paired with logical forms, together with an initial lexicon, Λ_0, which we will describe in Section 4.4.

At all stages, the algorithm maintains a parameter vector θ which stores a real value associated with every possible lexical item. The set of possible lexical items is

$$\Lambda^* = \Lambda_0 \cup \bigcup_{i=1}^{n} \text{GENLEX}(w_i, z_i)$$

Rules		Example categories produced from the logical form $\arg\max(\lambda x.flight(x) \wedge from(x, boston), \lambda x.cost(x))$
Input Trigger	Output Category	
constant c	$NP : c$	$NP : boston$
arity one predicate p	$N : \lambda x.p(x)$	$N : \lambda x.flight(x)$
arity one predicate p	$S\backslash NP : \lambda x.p(x)$	$S\backslash NP : \lambda x.flight(x)$
arity two predicate p_2	$(S\backslash NP)/NP : \lambda x.\lambda y.p_2(y, x)$	$(S\backslash NP)/NP : \lambda x.\lambda y.from(y, x)$
arity two predicate p_2	$(S\backslash NP)/NP : \lambda x.\lambda y.p_2(x, y)$	$(S\backslash NP)/NP : \lambda x.\lambda y.from(x, y)$
arity one predicate p_1	$N/N : \lambda g.\lambda x.p_1(x) \wedge g(x)$	$N/N : \lambda g.\lambda x.flight(x) \wedge g(x)$
literal with arity two predicate p_2 and constant second argument c	$N/N : \lambda g.\lambda x.p_2(x, c) \wedge g(x)$	$N/N : \lambda g.\lambda x.from(x, boston) \wedge g(x)$
arity two predicate p_2	$(N\backslash N)/NP : \lambda y.\lambda g.\lambda x.p_2(x, y) \wedge g(x)$	$(N\backslash N)/NP : \lambda y.\lambda g.\lambda x.from(x, y) \wedge g(x)$
an arg max / min with second argument arity one function f	$NP/N : \lambda g.\arg\max / \min(g, \lambda x.f(x))$	$NP/N : \lambda g.\arg\max(g, \lambda x.cost(x))$
arity one function f	$S/NP : \lambda x.f(x)$	$S/NP : \lambda x.cost(x)$
arity one function f	$(N\backslash N)/NP : \lambda y.\lambda f.\lambda x.g(x) \wedge f(x) >/< y$	$(N\backslash N)/NP : \lambda y.\lambda f.\lambda x.g(x) \wedge cost(x) > y$
no trigger	$S/NP : \lambda x.x, \quad S/N : \lambda f.\lambda x.f(x)$	$S/NP : \lambda x.x, \quad S/N : \lambda f.\lambda x.f(x)$

Figure 4-1: Rules used in GENLEX. We use the term *predicate* to refer to a function that returns a truth value; *function* to refer to all other functions; and *constant* to refer to constants of type e. Each row represents a rule. The first column lists the triggers that identify some sub-structure within a logical form. The second column lists the category that is created. The third column lists categories that are created when the rule is applied to the logical form at the top of this column.

Inputs:

- Training examples $E = \{(w_i, z_i) : i = 1 \ldots n\}$ where each w_i is a sentence, each z_i is a logical form.
- An initial lexicon Λ_0

Procedures:

- PARSE(w, z, Λ, θ): takes as input a sentence w, a logical form z, a lexicon Λ, and a parameter vector θ. Returns the highest probability parse for w with logical form z, when w is parsed by a PCCG with lexicon Λ and parameters θ. If there is more than one parse with the same highest probability, the entire set of highest probability parses is returned. Dynamic programming methods are used when implementing PARSE, see section 4.2.2.
- ESTIMATE(Λ, E, θ): takes as input a lexicon Λ, a training set E, and a parameter vector θ. Returns parameter values θ that are the output of stochastic gradient descent on the training set E under the grammar defined by Λ. The input θ is the initial setting for the parameters in the stochastic gradient descent algorithm. Dynamic programming methods are used when implementing ESTIMATE, see section 4.2.2.
- GENLEX(w, z): takes as input a sentence w and a logical form z. Returns a set of lexical items. See section 4.3.1 for a description of GENLEX.

Initialization: Define θ to be a real-valued vector of arity $|\Lambda^*|$, where $\Lambda^* = \Lambda_0 \cup \bigcup_{i=1}^{n} \text{GENLEX}(w_i, z_i)$. θ stores a parameter value for each potential lexical item. The initial parameters θ^0 are taken to be 0.1 for any member of Λ_0, and 0.01 for all other lexical items.

Algorithm:

- For $t = 1 \ldots T$

Step 1: (Lexical generation)

- For $i = 1 \ldots n$:
  - Set $\lambda = \Lambda_0 \cup \text{GENLEX}(w_i, z_i)$.
  - Calculate $\pi = \text{PARSE}(w_i, z_i, \lambda, \theta^{t-1})$.
  - Define λ_i to be the set of lexical entries in π.
- Set $\Lambda_t = \Lambda_0 \cup \bigcup_{i=1}^{n} \lambda_i$

Step 2: (Parameter Estimation)

- Set $\theta^t = \text{ESTIMATE}(\Lambda_t, E, \theta^{t-1})$

Output: Lexicon Λ_T together with parameters θ^T.

Figure 4-2: The overall learning algorithm.

The goal of the algorithm is to provide a relatively compact lexicon, which is a small subset of the entire set of possible lexical items. The algorithm achieves this by alternating between two steps. The goal of step 1 is to search for a relatively small number of lexical entries, which are nevertheless sufficient to successfully parse all training examples. Step 2 is then used to re-estimate the parameters of the lexical items that are selected in step 1.

In the t'th application of step 1, each sentence in turn is parsed with the current parameters θ^{t-1} and a special, sentence-specific lexicon which is defined as $\Lambda_0 \cup \text{GENLEX}(w_i, z_i)$. This will result in one or more highest-scoring parses that have the logical form z_i.[4] Lexical items are extracted from these highest-scoring parses alone. The result of this stage is to generate a small subset λ_i of $\text{GENLEX}(w_i, z_i)$ for each training example. The output of step 1, at iteration t, is a subset of Λ^*, defined as $\Lambda_t = \Lambda_0 \cup \bigcup_{i=1}^{n} \lambda_i$.

Step 2 re-estimates the parameters of the members of Λ_t, using stochastic gradient descent. The starting point for gradient descent when estimating θ^t is θ^{t-1}, i.e., the parameter values at the previous iteration. For any lexical item that is not a member of Λ_t, the associated parameter in θ^t is set to be the same as the corresponding parameter in θ^{t-1} (i.e., parameter values are simply copied across from the previous iteration).

The motivation for cycling between steps 1 and 2 is as follows. In step 1, keeping only those lexical items that occur in the highest scoring parse(s) leading to z_i results in a compact lexicon. This step is also guaranteed to produce a lexicon $\Lambda_t \subset \Lambda^*$ such that the accuracy on the training data when running the PCCG $(\Lambda_t, \theta^{t-1})$ is at least as accurate as applying the PCCG $(\Lambda^*, \theta^{t-1})$. In other words, pruning the lexicon in this way cannot hurt parsing performance on training data in comparison to using all possible lexical entries. To see this, note that restricting the lexicon in this way cannot exclude any of the highest-scoring parses for w_i that lead to

[4]Note that this set of highest-scoring parses is identical to the set produced by parsing with Λ^*, rather than the sentence-specific lexicon. This is because $\Lambda_0 \cup \text{GENLEX}(w_i, z_i)$ contains all lexical items that can possibly be used to derive z_i.

z_i. In practice, it may exclude some parses that lead to logical forms for w_i that are incorrect. Because the highest-scoring correct parses are still allowed, parsing performance cannot deteriorate.

Step 2 also has a guarantee, in that the log-likelihood on the training data will improve (assuming that stochastic gradient descent is successful in improving its objective). Step 2 is needed because after each application of step 1, the parameters θ^{t-1} are optimized for Λ_{t-1} rather than Λ_t, the current lexicon. Step 2 derives new parameter values θ_t which are optimized for Λ_t.

In summary, steps 1 and 2 together form a greedy, iterative method for simultaneously finding a compact lexicon and also optimizing the log-likelihood of the model on the training data.

4.4 Experiments

Data We evaluated the learning algorithm on two domains: Geo880, a set of 880 queries to a database of U.S. geography; and Jobs640, a set of 640 queries to a database of job listings. The data were originally annotated with Prolog style semantics which we manually converted to equivalent statements in the lambda calculus.

Initial Lexicon The initial lexicon Λ_0 includes lexical items that are derived directly from the database in the domain; for example, we have a list of entries $\{Utah := NP : utah, Idaho := NP : idaho, Nevada := NP : nevada, \ldots\}$ including every U.S. state in the geography domain.

It also includes lexical items that are domain independent, and easily specified by hand: for example, the definition for "what" in Figure 2-2(b) would be included, as it would be useful across many domains. Appendix C provides more details about the domain-independent lexicon.

Features and Parameter Initialization In these experiments we make use of *lexical features* alone. For each lexical entry in the grammar, we have a feature ϕ_j that counts the number of times that the lexical entry is used in y. For example, in

	Geo880		Jobs640	
	P	R	P	R
Our Method	96.25	79.29	97.36	79.29
COCKTAIL	89.92	79.40	93.25	79.84

Figure 4-3: The results for our method, and the previous work of COCKTAIL, when applied to the two database query domains. P is precision in recovering entire logical forms, R is recall.

the simple grammar with entries for *Utah*, *Idaho* and *borders*, there would be three features of this type. While these features are quite simple, we have found them to be quite successful when applied to the Geo880 and Jobs640 data sets. In later chapters, we will see examples of more complex features.

The initial parameter values were 0.1 for all lexical items in Λ_0, and 0.01 for all other lexical items. These values were chosen through experiments on the development data; they give a small initial bias towards using lexical items from Λ_0 and favor parses that include more lexical items.

Comparison We compare the structured classifier results to the COCKTAIL system (Tang & Mooney, 2001). The COCKTAIL experiments were conducted by performing ten–fold cross validation of the entire data set. We used a slightly different experimental set-up, where we made an explicit split between training and test data sets.[5] The Geo880 data set was divided into 600 training examples and 280 test examples; the Jobs640 set was divided into 500 training and 140 test examples. The parameters of the training algorithm were tuned by cross–validation on the training set. We did two passes of the overall learning loop in Figure 4-2. Each time we used gradient descent to estimate parameters, we performed three passes over the training set with the learning-rate parameters $\alpha_0 = 0.1$ and $c = 0.001$.

We give precision and recall for the different algorithms, defined as

[5]This allowed us to use cross-validation experiments on the *training set* to optimize parameters, and more importantly to develop our algorithms while ensuring that we had not implicitly tuned our approach to the final test set.

$$Precision = \frac{\#\ correct}{total\ \#\ parsed}$$

$$Recall = \frac{\#\ correct}{total\ \#\ examples}$$

where sentences are correct if the parser gives a completely correct semantics.

Results Figure 4.4 shows the results of the experiments. Our approach has higher precision than COCKTAIL on both domains, with a very small reduction in recall. When evaluating these results, it is important to realize that COCKTAIL is provided with a fairly extensive lexicon that pairs words with semantic predicates. For example, the word *borders* would be paired with the predicate $borders(x, y)$. This prior information goes substantially beyond the initial lexicon used in our own experiments.[6]

To better understand these results, we examined performance of our method through cross-validation on the training set. We found that the approach creates a compact lexicon for the training examples that it parses. On the Geo880 domain, the initial number of lexical items created by GENLEX was on average 393.8 per training example. After pruning, on average only 5.1 lexical items per training example remained. The Jobs640 domain showed a reduction from an average of 697.1 lexical items per training example, to 6.6 items.

To investigate the disparity between precision and recall, we examined the behavior of the algorithm when trained in the cross-validation (development) regime. We found that on average, the learner failed to parse 9.3% of the training examples in the Geo880 domain, and 8.7% of training examples in the Jobs640 domain. (Note that sentences which cannot be parsed in step 1 of the training algorithm are excluded from the training set during step 2.) These parse failures were caused by sentences whose semantics could not be built from the lexical items that GENLEX created. For example, the learner failed to parse complex sentences such as *Through which*

[6]Note that the work of Thompson and Mooney (2002) does describe a method which automatically learns a lexicon. However, results for this approach were worse than results for CHILL (Zelle & Mooney, 1996), which in turn were considerably worse than results for COCKTAIL on the Geo250 domain, a subset of the examples in Geo880.

states	:=	$N : \lambda x.state(x)$
major	:=	$N/N : \lambda f.\lambda x.major(x) \wedge f(x)$
population	:=	$N : \lambda x.population(x)$
cities	:=	$N : \lambda x.city(x)$
rivers	:=	$N : \lambda x.river(x)$
run through	:=	$(S\backslash NP)/NP : \lambda x.\lambda y.traverse(y, x)$
the largest	:=	$NP/N : \lambda f.\arg\max(f, \lambda x.size(x))$
river	:=	$N : \lambda x.river(x)$
the highest	:=	$NP/N : \lambda f.\arg\max(f, \lambda x.elev(x))$
the longest	:=	$NP/N : \lambda f.\arg\max(f, \lambda x.len(x))$

Figure 4-4: Ten learned lexical items that had highest associated parameter values from a randomly chosen development run in the Geo880 domain.

states does the Mississippi run because GENLEX does not create lexical entries that allow the verb *run* to find its argument, the preposition *through*, when it has moved to the front of the sentence. This problem is almost certainly a major cause of the lower recall on test examples. Exploring the addition of more rules to GENLEX is an important area for future work.

Figure 4-4 gives a sample of lexical entries that are learned by the approach. These entries are linguistically plausible and should generalize well to unseen data.

4.5 Summary

This chapter presented a learning algorithm that creates accurate structured classifiers for natural language interfaces. We described a procedure, GENLEX, for creating a large set of linguistically-plausible CCG lexical items. We then developed a learning algorithm for probabilistic CCGs that prunes this lexicon while estimating parameters of the log-linear parsing model. Finally, we demonstrated experimentally that this approach is competitive with previous learning methods.

Chapter 5

Learning Relaxed CCGs

In this chapter, we reconsider the problem of learning to map sentences to lambda-calculus representations of their underlying semantics. We focus on the challenges that come with learning to analyze spontaneous, unedited natural language input, as is commonly seen in natural language interface applications. A key idea is to introduce non-standard CCG combinators that relax certain parts of the grammar — for example allowing flexible word order, or insertion of lexical items — with learned costs. We also present a new algorithm for inducing a weighted CCG. This approach uses the GENLEX procedure from the last chapter to induce a lexicon and perceptron-style additive updates to estimate the parameters. The result is an online, error-driven algorithm that considers each training example in turn and makes changes to the model only when the current analysis is incorrect. Experimental results for the approach on ATIS data show 86% F-measure in recovering fully correct semantic analyses and 95.9% F-measure by a partial-match criterion, a more than 5% improvement over the 90.3% partial-match figure reported by He and Young (2006). This chapter is based on work originally described in (Zettlemoyer & Collins, 2007).

5.1 Introduction

This chapter describes an extension to the learning approach from Chapter 4, where we saw how to induce a probabilistic Combinatorial Categorial Grammar (CCG) for

a) on may four atlanta to denver delta flight 257
$\lambda x.month(x, may) \wedge day\_number(x, fourth) \wedge from(x, atlanta) \wedge to(x, denver) \wedge$
$airline(x, delta\_air\_lines) \wedge flight(x) \wedge flight\_number(x, 257)$

b) show me information on american airlines from fort worth texas to philadelphia
$\lambda x.airline(x, american\_airlines) \wedge from(x, fort\_worth) \wedge to(x, philadelphia)$

c) okay that one's great too now we're going to go on april twenty second dallas to washington the latest nighttime departure one way
$argmax(\lambda x.flight(x) \wedge from(x, dallas) \wedge to(x, washington) \wedge month(x, april) \wedge$
$day\_number(x, 22) \wedge during(x, night) \wedge one\_way(x), \lambda y.depart\_time(y))$

Figure 5-1: Three sentences from the ATIS domain.

mapping sentences to logical form. The use of a detailed grammatical formalism such as CCG has the advantage that it allows a system to handle quite complex semantic effects, such as coordination and scoping phenomena. In particular, it allows us to leverage the considerable body of work on semantics within these formalisms, for example see Carpenter (Carpenter, 1997). However, a grammar based on a formalism such as CCG can be somewhat rigid, and this can cause problems when a system is faced with spontaneous, unedited natural language input, as is commonly seen in natural language interface applications. For example, consider the sentences shown in Figure 5-1, which were taken from the ATIS travel-planning domain (Dahl et al., 1994). These sentences exhibit characteristics which present significant challenges to the previously described approach. For example, they have quite flexible word order, and include telegraphic language where some words are effectively omitted.

We describe a learning algorithm that retains the advantages of using a detailed grammar, but is highly effective in dealing with phenomena seen in spontaneous natural language, as exemplified by the ATIS domain. A key idea is to extend the approach by allowing additional non-standard CCG combinators. These combinators relax certain parts of the grammar—for example allowing flexible word order, or insertion of lexical items—with learned costs for the new operations. This approach has the advantage that it can be seamlessly integrated into CCG learning algorithm introduced in the previous chapter.

A second contribution of the work is a new, online algorithm for CCG learning.

The approach involves perceptron training of a model with hidden variables. In this sense it is related to the algorithm of Liang et al. (2006). However it has the additional twist of also performing grammar induction (lexical learning) in an online (example-by-example) manner. In our experiments, we show that the new algorithm is considerably more efficient than the algorithm described in Chapter 4; this is important when training on large training sets, such as the ATIS data used in this chapter.

Results for the approach on ATIS data show 86% F-measure accuracy in recovering fully correct semantic analyses, and 95.9% F-measure by a partial-match criterion described by He and Young (2006). The latter figure contrasts with a figure of 90.3% for the approach reported by He and Young (2006).[1] Results on the Geo880 domain also show an improvement in accuracy, with 88.9% F-measure for the new approach, compared to 87.0% F-measure for the method described in Chapter 4.

In this chapter, we first describe the new parsing rules that relax the CCG grammar. We then describe the online learning algorithm that incorporates these rules. Finally, we present the evaluation.

5.2 Parsing Extensions

This section describes a set of CCG combinators which we add to the conventional CCG combinators described in Section 2.2. These additional combinators are natural extensions of the forward application, forward composition, and type-raising rules seen in CCG. We first describe a set of combinators that allow the parser to significantly relax constraints on word order. We then describe a set of type-raising rules which allow the parser to cope with telegraphic input (in particular, missing function words). In both cases these additional rules lead to significantly more parses for any sentence x given a lexicon Λ. Many of these parses will be suspect from a linguistic perspective; broadening the set of CCG combinators in this way might be considered a dangerous move. However, the learning algorithm in our approach can learn

[1] He and Young (2006) do not give results for recovering fully correct parses.

weights for the new rules, effectively allowing the model to learn to use them only in appropriate contexts; in the experiments we show that the rules are highly effective additions when used within a weighted CCG.

5.2.1 Application and Composition Rules

The first new combinators we consider are the *relaxed functional application* rules:

$$A\backslash B : f \quad B : g \;\Rightarrow\; A : f(g) \quad (\gtrsim)$$
$$B : g \quad A/B : f \;\Rightarrow\; A : f(g) \quad (\lesssim)$$

These are variants of the original application rules, where the slash direction on the principal categories (A/B or $A\backslash B$) is reversed.[2] These rules allow simple reversing of regular word order, for example

$$\dfrac{\dfrac{\text{flights}}{\begin{array}{c}N\\ \lambda x.flight(x)\end{array}} \quad \dfrac{\text{one way}}{\begin{array}{c}N/N\\ \lambda f.\lambda x.f(x) \wedge one\_way(x)\end{array}}}{\begin{array}{c}N\\ \lambda x.flight(x) \wedge one\_way(x)\end{array}}\lesssim$$

Note that we can recover the correct analysis for this fragment, with the same lexical entries as those used for the conventional word order, *one-way flights*.

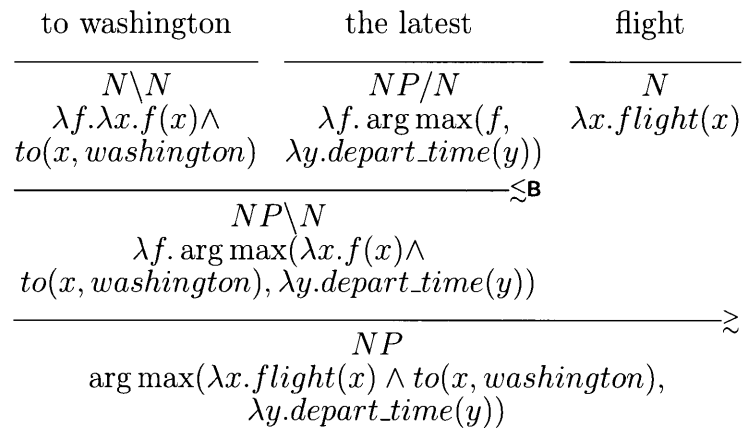
A second set of new combinators are the *relaxed functional composition* rules:

$$A\backslash B : f \quad B/C : g \;\Rightarrow\; A/C : \lambda x.f(g(x)) \quad (\gtrsim \mathbf{B})$$
$$B\backslash C : g \quad A/B : f \;\Rightarrow\; A\backslash C : \lambda x.f(g(x)) \quad (\lesssim \mathbf{B})$$

These rules are variantions of the standard functional composition rules, where the slashes of the principal categories are reversed.

[2]Rules of this type are non-standard in the sense that they violate Steedman's Principle of Consistency (2000); this principle states that rules must be consistent with the slash direction of the principal category. Steedman (2000) only considers rules that do not violate this principle—for example, crossed composition rules, which we consider later, and which Steedman also considers, do not violate this principle.

An important point is that that these new composition and application rules can deal with quite flexible word orders. For example, take the fragment *to washington the latest flight*. In this case the parse is



Note that in this case the substring *the latest* has category NP/N, and this prevents a naive parse where *the latest* first combines with *flight*, and *to washington* then combines with *the latest flight*. The functional composition rules effectively allow *the latest* to take scope over *flight* and *to washington*, in spite of the fact that *the latest* appears between the two other sub-strings. Examples like this are quite frequent in domains such as ATIS.

We add features in the model which track the occurrences of each of these four new combinators. Specifically, we have four new features in the definition of ϕ; each feature tracks the number of times one of the combinators is used in a CCG parse. The model learns parameter values for each of these features, allowing it to learn to penalise these rules to the correct extent.

5.2.2 Additional Rules of Type-Raising

We now describe new CCG operations designed to deal with cases where words are in some sense missing in the input. For example, in the string *flights Boston to New York*, one style of analysis would assume that the preposition *from* had been deleted from the position before *Boston*.

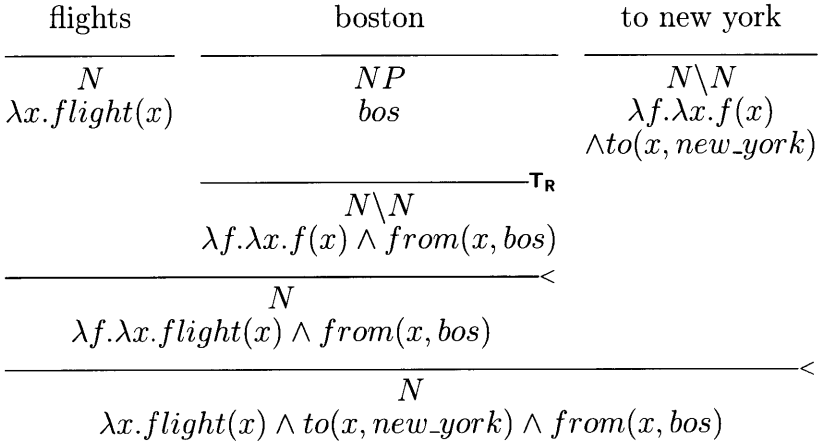
The first set of rules is generated from the following *role-hypothesising type shift-*

ing rules template:

$$NP : c \quad \Rightarrow \quad N\backslash N : \lambda f.\lambda x.f(x) \wedge p(x, c) \quad (\mathbf{T_R})$$

This rule can be applied to any NP with semantics c, and any arity-two function p such that the second argument of p has the same type as c. By "any" arity-two function, we mean any of the arity-two functions seen in training data. We define features within the feature-vector ϕ that are sensitive to the number of times these rules are applied in a parse; a separate feature is defined for each value of p.

In practice, in our experiments most rules of this form have p as the semantics of some preposition, for example *from* or *to*. A typical example of a use of this rule would be the following:



The second rule we consider is the *null-head type shifting* rule:

$$N\backslash N : f \quad \Rightarrow \quad N : f(\lambda x.true) \quad (\mathbf{T_N})$$

This rule allows parses of fragments such as *American Airlines from New York*, where there is again a word that is in some sense missing (it is straightforward to derive a parse for *American Airlines flights from New York*). The analysis would be as follows:

$$\dfrac{\dfrac{\text{American Airlines}}{\begin{array}{c}N/N\\ \lambda f.\lambda x.f(x) \wedge airline(x, aa)\end{array}} \quad \dfrac{\dfrac{\text{from New York}}{\begin{array}{c}N\backslash N\\ \lambda f.\lambda x.f(x) \wedge from(x, new\_york)\end{array}}}{\begin{array}{c}N\\ \lambda x.from(x, new\_york)\end{array}}\,\mathbf{T_N}}{\begin{array}{c}N\\ \lambda x.airline(x, aa) \wedge from(x, new\_york)\end{array}}\,>$$

The new rule effectively allows the prepositional phrase *from New York* to type-shift to an entry with syntactic type N and semantics $\lambda x.from(x, new\_york)$, representing the set of all things from New York.[3]

We introduce a single additional feature that counts the number of times this rule is used.

5.2.3 Crossed Composition Rules

Finally, we include *crossed functional composition* rules:

$$A/B : f \quad B\backslash C : g \quad \Rightarrow \quad A\backslash C : \lambda x.f(g(x)) \quad (>\mathbf{B}_\times)$$
$$B/C : g \quad A\backslash B : f \quad \Rightarrow \quad A/C : \lambda x.f(g(x)) \quad (<\mathbf{B}_\times)$$

These rules are standard CCG operators but they were not used by the parser described in Chapter 4. When used in unrestricted contexts, they can significantly relax word order. Again, we address this problem by introducing features that count the number of times they are used in a parse.[4]

5.2.4 An Example

As a final point, to see how these rules can interact in practice, see figure 5-2. This example demonstrates the use of the relaxed application and composition rules, as

[3]Note that we do not analyze this prepositional phrase as having the semantics $\lambda x.flight(x) \wedge from(x, new\_york)$—although in principle this is possible—as the $flight(x)$ predicate is not necessarily implied by this utterance.

[4]In general, applications of the crossed composition rules can be lexically governed, as described in work on Multi-Modal CCG (Baldridge, 2002). In the future we would like to incorporate more fine-grained lexical distinctions of this type.

well as the new type-raising rules.

5.3 Model and Features

To select the best analysis for each input sentence, we use a weighted CCG, as defined in Section 2.3. The models has three components:

- A parameter vector $\theta \in \mathbb{R}^d$.
- A CCG lexicon Λ.
- A feature function $\phi(w, z)$ that maps a sentence w together with a CCG parse z to a feature vector.

In the next section, we present an algorithm for learning the parameter vector and lexicon. In this section we define the feature function $\phi(w, z)$.

As described in section 5.2, we introduce features for the new CCG combinators. This includes features that track the number of times each of the four flexible application and composition rules are used in parse; a feature that tracks the number of times the *null-head type shifting rule* is used; and finally features that track the number of times the *role-hypothesizing type shifting rule* is used with each possible value for p. In addition, we include features that track the number of times each lexical item in Λ is used, as defined in Section 4.4. For example, we would have one feature tracking the number of times the lexical entry $flights := N : \lambda x.flight(x)$ is used in a parse, and similar features for all other members of Λ.

Finally, we introduce new features that directly consider the semantics of a parse. For each predicate f seen in training data, we introduce a feature that counts the number of times f is conjoined with itself at some level in the logical form. For example, the expression $\lambda x.flight(x) \wedge from(x, new\_york) \wedge from(x, boston)$ would trigger the new feature for the $from$ predicate signaling that the logical-form describes flights with more than one origin city. We introduce similar features that track disjunction as opposed to conjunction.

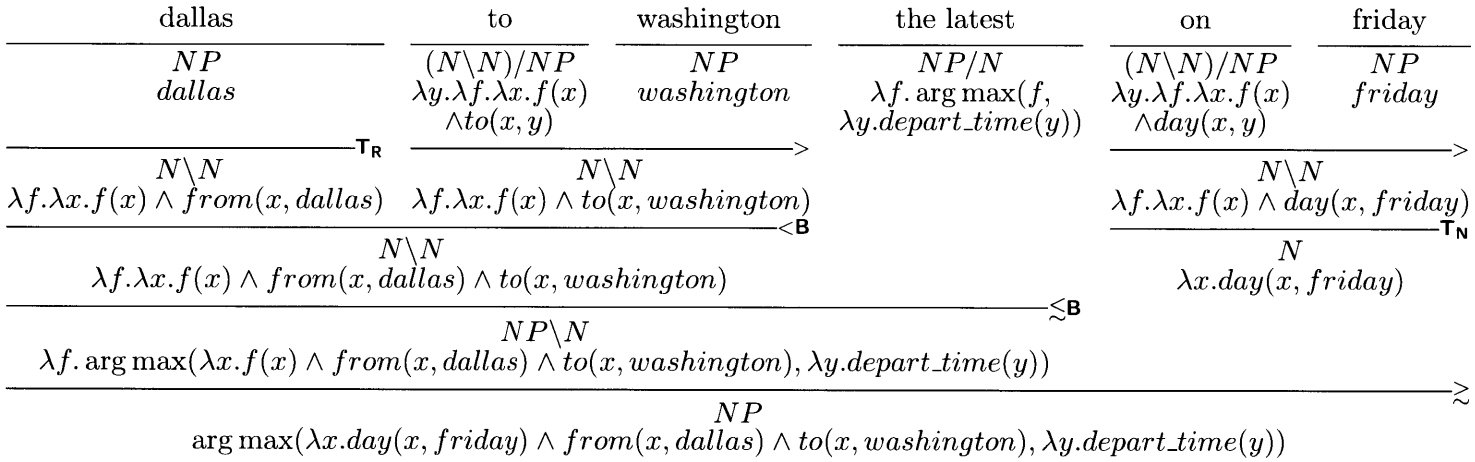


Figure 5-2: A parse with the flexible parser.

5.4 An Online Learning Algorithm

Figure 5-3 shows a learning algorithm that takes a training set of (x_i, z_i) pairs as input, and returns a weighted CCG (i.e., a pair (θ, Λ)) as its output. The algorithm is *online*, in that it visits each example in turn, and updates both θ and Λ if necessary. It repeatedly iterates through the whole training set, performing a three step process for each example. In Step 1, the input x_i is parsed. If it is parsed correctly, the algorithm immediately moves to the next example. In Step 2, the algorithm temporarily introduces all lexical entries seen in GENLEX(x_i, z_i), and finds the highest scoring parse that leads to the correct semantics z_i. A small subset of GENLEX(x_i, z_i)—namely, only those lexical entries that are contained in the highest scoring parse—are added to Λ. In Step 3, a simple perceptron update (Collins, 2002) is performed. The hypothesis is parsed again with the new lexicon, and an update to the parameters θ is made if the resulting parse does not have the correct logical form.

This algorithm differs from the approach describe in Chapter 4 (ZC05) in two important respects. First, the ZC05 algorithm performed learning of the lexicon Λ at each iteration in a batch method, requiring a pass over the entire training set. The new algorithm is fully online, learning both Λ and θ in an example-by-example fashion. This has important consequences for the efficiency of the algorithm. Second, the parameter estimation method in ZC05 was based on stochastic gradient descent on a log-likelihood objective function. The new algorithm makes use of perceptron updates, which are simpler and cheaper to compute.

This algorithm assumes the same initial lexicon Λ_0 that we saw in Chapter 4. There are two types of entries. First, we compile entries such as $Boston := NP : boston$ for entities such as cities, times and month-names that occur in the domain or underlying database. In practice it is easy to compile a list of these atomic entities. Second, the lexicon has entries for some function words such as wh-words, and determiners. These entries are likely to be domain independent, so it is simple enough to compile a list that can be reused in new domains.

Inputs:

- Training examples $\{(w_i, z_i) : i = 1 \ldots n\}$ where each w_i is a sentence, each z_i is a logical form.
- An initial lexicon Λ_0.
- Number of training iterations, T.

Definitions:

- GENLEX(w, z) takes as input a sentence w and a logical form z and returns a set of lexical items as described in Section 4.3.1.
- GEN$(w; \Lambda)$ is the set of all parses for w with lexicon Λ.
- GEN$(w, z; \Lambda)$ is the set of parses for w with lexicon Λ, which have logical form z.
- The function $\phi(w, y)$ represents the features described in section 5.3.
- The function $L(y)$ maps a parse tree y to its associated logical form.

Initialization:

Set parameters θ to initial values described in section 5.5.2. Set $\Lambda = \Lambda_0$.

Algorithm:

- For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Check correctness)

- Let $y^* = \arg\max_{y \in \text{GEN}(x_i; \Lambda)} \theta \cdot \phi(x_i, y)$.
- If $L(y^*) = z_i$, go to the next example.

Step 2: (Lexical generation)

- Set $\lambda = \Lambda \cup \text{GENLEX}(x_i, z_i)$.
- Let $y^* = \arg\max_{y \in \text{GEN}(x_i, z_i; \lambda)} \theta \cdot \phi(x_i, y)$.
- Define λ_i to be the set of lexical entries in y^*.
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$.

Step 3: (Update parameters)

- Let $y' = \arg\max_{y \in \text{GEN}(x_i; \Lambda)} \theta \cdot \phi(x_i, y)$.
- If $L(y') \neq z_i$:
  - Set $\theta = \theta + \phi(x_i, y^*) - \phi(x_i, y')$.

Output: Lexicon Λ together with parameters θ.

Figure 5-3: An online learning algorithm.

5.5 Experiments

The main focus of our experiments is on the ATIS travel planning domain. For development, we used 4978 sentences, split into a training set of 4500 examples, and a development set of 478 examples. For test, we used the ATIS NOV93 test set which contains 448 examples. To create the annotations, we created a script that maps the original SQL annotations provided with the data to lambda-calculus expressions.

He and Young (2006) previously reported results on the ATIS domain, using a learning approach which also takes sentences paired with semantic annotations as input. In their case, the semantic structures resemble context-free parses with semantic (as opposed to syntactic) non-terminal labels. In our experiments we have used the same split into training and test data as He and Young (2006), ensuring that our results are directly comparable.

He and Young (2006) report *partial match* figures for their parser, based on precision and recall in recovering attribute-value pairs. (For example, the sentence *flights to Boston* would have a single attribute-value entry, namely *destination* = *Boston*.) It is simple for us to map from lambda-calculus expressions to attribute-value entries of this form; for example, the expression $to(x, Boston)$ would be mapped to *destination* = *Boston*. He and Young (2006) gave us their data and annotations, so we can directly compare results on the partial-match criterion. We also report accuracy for exact matches of lambda-calculus expressions, which is a stricter criterion.

In addition, we report results for the method on the Geo880 domain. This allows us to compare directly to the approach described in the last chapter, using the same split of the data into training and test sets of sizes 600 and 280 respectively. We use cross-validation of the training set, as opposed to a separate development set, for optimization of parameters.

5.5.1 Improving Recall

The simplest approach to the task is to train the parser and directly apply it to test sentences. In our experiments we will see that this produces results which have

high precision, but somewhat lower recall, due to some test sentences failing to parse (usually due to words in the test set which were never observed in training data). A simple strategy to alleviate this problem is as follows. If the sentence fails to parse, we parse the sentence again, this time allowing parse moves which can delete words at some cost. The cost of this deletion operation is optimized on development data. This approach can significantly improve F-measure on the partial-match criterion in particular. We report results both with and without this second pass strategy.

5.5.2 Parameters in the Approach

The algorithm in figure 6-2 has a number of parameters, the set $\{T, \alpha, \beta, \gamma\}$, which we now describe. The values of these parameters were chosen to optimize the performance on development data. T is the number of passes over the training set, and was set to be 4. Each lexical entry in the initial lexicon Λ_0 has an associated feature which counts the number of times this entry is seen in a parse. The initial parameter value in θ for all features of this form was chosen to be some value α. Each of the new CCG rules—the application, composition, crossed-composition, and type-raising rules described in section 5.2—has an associated parameter. We set all of these parameters to the same initial value β. Finally, when new lexical entries are added to Λ (in step 2 of the algorithm), their initial weight is set to some value γ. Through experiments on development data, we found that the values $\alpha = 0.1$, $\beta = -1.0$, and $\gamma = -0.05$ work well in practice. These settings initially encourage encourage the use of lexical entries from the initial lexicon, discourage adding new entries to the lexicon, and penalize the use of the relaxed CCG parse rules.

5.5.3 Results

Table 5.1 shows accuracy for the method by the exact-match criterion on the ATIS test set. The two pass strategy actually hurts F-measure in this case, although it does improve recall of the method.

Table 5.2 shows results under the partial-match criterion. The results for our

	Precision	Recall	F1
Single-Pass Parsing	90.61	81.92	86.05
Two-Pass Parsing	85.75	84.6	85.16

Table 5.1: Exact-match accuracy on the ATIS test set.

	Precision	Recall	F1
Single-Pass Parsing	96.76	86.89	91.56
Two-Pass Parsing	95.11	96.71	95.9
He and Young (2006)	–	–	90.3

Table 5.2: Partial-credit accuracy on the ATIS test set.

approach are higher than those reported by He and Young (2006) even without the second, high-recall, strategy. With the two-pass strategy our method has more than halved the F-measure error rate, giving improvements from 90.3% F-measure to 95.9% F-measure.

Table 5.3 shows results on the Geo880 domain. The new method gives improvements in performance both with and without the two pass strategy, showing that the new CCG combinators, and the new learning algorithm, give some improvement on even this domain. The improved performance comes from a slight drop in precision which is offset by a large increase in recall.

Table 5.4 shows ablation studies on the ATIS data, where we have selectively removed various aspects of the approach, to measure their impact on performance. It can be seen that accuracy is seriously degraded if the new CCG rules are removed, or if the features associated with these rules (which allow the model to penalize these rules) are removed.

Finally, we report results concerning the efficiency of the new online algorithm as compared to the algorithm from Chapter 4 (ZC05). We compared running times for the new algorithm, and the ZC05 algorithm, on the geography domain, with both methods making 4 passes over the training data. The new algorithm took less than 4 hours, compared to over 12 hours for the ZC05 algorithm. The main explanation for

	Precision	Recall	F1
Single-Pass Parsing	95.49	83.2	88.93
Two-Pass Parsing	91.63	86.07	88.76
ZC05	96.25	79.29	86.95

Table 5.3: Exact-match accuracy on the Geo880 test set.

	Precision	Recall	F1
Full Online Method	87.26	74.44	80.35
Without control features	70.33	42.45	52.95
Without relaxed word order	82.81	63.98	72.19
Without word insertion	77.31	56.94	65.58

Table 5.4: Exact-match accuracy on the ATIS development set for the full algorithm and restricted versions of it. The second row reports results of the approach without the features described in section 5.2 that control the use of the new combinators. The third row presents results without the combinators from section 5.2.1 that relax word order. The fourth row reports experiments without the type-raising combinators presented in section 5.2.2.

this improved performance is that on many training examples,[5] in step 1 of the new algorithm a correct parse is found, and the algorithm immediately moves on to the next example. Thus GENLEX is not required, and in particular parsing the example with the large set of entries generated by GENLEX is not required.

5.6 Summary

We presented a new, online algorithm for learning a combinatory categorial grammar (CCG), together with parameters that define a log-linear parsing model. We showed that the use of non-standard CCG combinators is highly effective for parsing sentences with the types of phenomena seen in spontaneous, unedited natural language. The resulting system achieved significant accuracy improvements in both the ATIS and Geo880 domains.

[5]Measurements on the Geo880 domain showed that in the 4 iterations, 83.3% of all parses were successful at step 1.

Chapter 6

Context-dependent Learning

This chapter considers the problem of learning context-dependent mappings from sentences to logical form. The training examples are sequences of sentences annotated with lambda-calculus meaning representations. We develop an algorithm that maintains explicit, lambda-calculus representations of salient discourse entities and uses a context-dependent analysis pipeline to recover logical forms. The method uses a hidden-variable variant of the perception algorithm to learn a linear model used to select the best analysis. Experiments on context-dependent utterances from the ATIS corpus show that the method recovers fully correct logical forms with 83.7% accuracy. This chapter is based on work originally described in (Zettlemoyer & Collins, 2009).

6.1 Introduction

In the last two chapters we developed approaches for learning to map isolated sentences to logical form. For instance, a training example might be:

Sent. 1: List flights to Boston on Friday night.

LF 1: $\lambda x.flight(x) \wedge to(x, bos) \wedge day(x, fri) \wedge during(x, night)$

Here the logical form (LF) is a lambda-calculus expression defining a set of entities that are flights to Boston departing on Friday night.

In this chapter we develop an approach for the context-dependent learning problem. In this case, the underlying meaning of a sentence can depend on the context in which it appears. For example, consider an interaction where Sent. 1 is followed by the sentence:

Sent. 2: Show me the flights after 3pm.

LF 2: $\lambda x.flight(x) \wedge to(x, bos) \wedge day(x, fri) \wedge depart(x) > 1500$

In this case, the fact that Sent. 2 describes flights to Boston on Friday must be determined based on the context established by the first sentence.

We introduce a supervised, hidden-variable approach for learning to interpret sentences in context. Each training example is a sequence of sentences annotated with logical forms. Figure 6-1 shows excerpts from three training examples in the ATIS corpus (Dahl et al., 1994).

In general, there are a large number of ways that the meaning of sentences can depend on the context in which they appear. Webber (1979) presents a description of more than twelve possibilities, including the use of pronouns (he, she, it), define noun phrases (the flights), and verb phrase references (such as "do so"). In this chapter, we focus on modeling the phenomena that appear in interfaces to databases, such as the ATIS corpus. Although pronouns and definite nouns phrases appear, the most common type of reference is one-anaphora, where sentences refer to descriptions of previous set of flights. For example, they appear in sentences such as "Show me the cheapest ones."

For context-dependent analysis, we develop an approach that maintains explicit, lambda-calculus representations of salient discourse entities and uses a two-stage pipeline to construct context-dependent logical forms. The first stage uses a probabilistic Combinatory Categorial Grammar (CCG) parsing algorithm to produce a context-independent, underspecified meaning representation. The second stage resolves this underspecified meaning representation by making a sequence of modifications to it that depend on the context provided by previous utterances.

In general, there are a large number of possible context-dependent analyses for each sentence. To select the best one, we present a weighted linear model that is used to make a range of parsing and context-resolution decisions. Since the training data contains only the final logical forms, we model these intermediate decisions as hidden variables that must be estimated without explicit supervision. We show that this model can be effectively trained with a hidden-variable variant of the perceptron algorithm.

In experiments on the ATIS DEC94 test set, the approach recovers fully correct logical forms with 83.7% accuracy.

6.2 The Learning Problem

We assume access to a training set that consists of n *interactions* $D = \langle I_1, \ldots, I_n \rangle$. The i'th interaction I_i contains n_i sentences, $w_{i,1}, \ldots, w_{i,n_i}$. Each sentence $w_{i,j}$ is paired with a lambda-calculus expression $z_{i,j}$ specifying the target logical form. Figure 6-1 contains example interactions.

The logical forms in the training set are representations of each sentence's underlying meaning. In most cases, context (the previous utterances and their interpretations) is required to recover the logical form for a sentence. For instance, in Example 1(b) in Figure 6-1, the sentence "show me the ones that leave in the morning" is paired with

$$\lambda x.flight(x) \wedge from(x, bos) \wedge to(x, phi) \wedge during(x, morning)$$

Some parts of this logical form ($from(x, bos)$ and $to(x, phi)$) depend on the context. They have to be recovered from the previous logical forms.

At step j in interaction i, we define the *context* $\langle z_{i,1}, \ldots, z_{i,j-1} \rangle$ to be the $j-1$ preceding logical forms.[1] Now, given the training data, we can create training

[1] In general, the context could also include the previous sentences $w_{i,k}$ for $k < j$. In our data, we never observed any interactions where the choice of the correct logical form $z_{i,j}$ depended on the words in the previous sentences, given the logical forms of the previous sentences.

Example #1:
(a) show me the flights from boston to philly
$\lambda x.flight(x) \wedge from(x, bos) \wedge to(x, phi)$
(b) show me the ones that leave in the morning
$\lambda x.flight(x) \wedge from(x, bos) \wedge to(x, phi) \wedge during(x, morning)$
(c) what kind of plane is used on these flights
$\lambda y.\exists x.flight(x) \wedge from(x, bos) \wedge to(x, phi) \wedge during(x, morning) \wedge aircraft(x) = y$

Example #2:
(a) show me flights from milwaukee to orlando
$\lambda x.flight(x) \wedge from(x, mil) \wedge to(x, orl)$
(b) cheapest
$argmin(\lambda x.flight(x) \wedge from(x, mil) \wedge to(x, orl), \quad \lambda y.fare(y))$
(c) departing wednesday after 5 o'clock
$argmin(\lambda x.flight(x) \wedge from(x, mil) \wedge to(x, orl) \wedge day(x, wed) \wedge depart(x) > 1700$,
$\lambda y.fare(y))$

Example #3:
(a) show me flights from pittsburgh to la thursday evening
$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la) \wedge day(x, thur) \wedge during(x, evening)$
(b) thursday afternoon
$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la) \wedge day(x, thur) \wedge during(x, afternoon)$
(c) thursday after 1700 hours
$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la) \wedge day(x, thur) \wedge depart(x) > 1700$

Figure 6-1: ATIS interaction excerpts.

examples $(x_{i,j}, z_{i,j})$ for $i = 1 \ldots n, j = 1 \ldots n_i$. Each $x_{i,j}$ is a sentence and a context, $x_{i,j} = (w_{i,j}, \langle z_{i,1}, \ldots, z_{i,j-1} \rangle)$. Given this set up, we have a supervised learning problem with input $x_{i,j}$ and output $z_{i,j}$.

6.3 Overview of Approach

In general, the mapping from a sentence and a context to a logical form can be quite complex. In this section, we present an overview of our learning approach. We assume the learning algorithm has access to:

- A training set D, defined in Section 6.2.

- A CCG lexicon. See Section 2.2 for an overview of CCG. Each entry in the lexicon pairs a word (or sequence of words), with a CCG category specifying both the syntax and semantics for that word. One example CCG entry would pair *flights* with the category $N : \lambda x.flight(x)$.

Derivations A *derivation* for the j'th sentence in an interaction takes as input a pair $x = (w_j, C)$, where $C = \langle z_1 \ldots z_{j-1} \rangle$ is the current context. It produces a logical form z. There are two stages:

- First, the sentence w_j is parsed using the CCG lexicon to form an intermediate, context-independent logical form π.
- Second, in a series of steps, π is mapped to z. These steps depend on the current context C.

As one sketch of a derivation, consider how we might analyze Example 1(b) in Figure 6-1. In this case the sentence is "show me the ones that leave in the morning." The CCG parser we will describe in Section 6.4.1 would produce the following context-independent logical form:

$$\lambda x.!\langle e, t\rangle(x) \wedge during(x, morning)$$

The subexpression $!\langle e, t\rangle$ results directly from the referential phrase *the ones*; we discuss this in more detail in Section 6.4.1, but intuitively this subexpression specifies that a lambda-calculus expression of type $\langle e, t\rangle$ (a function from entities to truth values) must be recovered from the context and substituted in its place.

In the second (contextually dependent) stage of the derivation, the expression

$$\lambda x.flight(x) \wedge from(x, bos) \wedge to(x, phi)$$

is recovered from the context, and substituted for the $!\langle e, t\rangle$ subexpression, producing the desired final logical form, seen in Example 1(b).

In addition to substitutions of this type, we will also perform other types of context-dependent resolution steps, as described in Section 6.5.

In general, both of the stages of the derivation involve considerable ambiguity – there will be a large number of possible context-independent logical forms π for w_j and many ways of modifying each π to create a final logical form z_j.

Learning We model the problem of selecting the best derivation as a structured prediction problem (Johnson et al., 1999; Lafferty et al., 2001; Collins, 2002; Taskar et al., 2004). We present a linear model with features for both the parsing and context resolution stages of the derivation. In our setting, the choice of the context-independent logical form π and all of the steps that map π to the output z are hidden variables; these steps are not annotated in the training data. To estimate the parameters of the model, we use a hidden-variable version of the perceptron algorithm. We use an approximate search procedure to find the best derivation both while training the model and while applying it to test examples.

Evaluation We evaluate the approach on sequences of sentences $\langle w_1, \ldots, w_k \rangle$. For each w_j, the algorithm constructs an output logical form z_j which is compared to a gold standard annotation to check correctness. At step j, the context contains the previous z_i, for $i < j$, output by the system.

6.4 Context-independent Parsing

In this section, we describe a set of extensions to the CCG formalism that allow the parser to construct logical forms containing references, such as the $!\langle e, t \rangle$ expression from the example derivation in Section 6.3.

6.4.1 Parsing with References

We use an exclamation point followed by a type expression to specify references in a logical form. For example, $!e$ is a reference to an entity and $!\langle e, t \rangle$ is a reference to

a function. As motivated in Section 6.3, we introduce these expressions so they can later be replaced with appropriate lambda-calculus expressions from the context.

Sometimes references are lexically triggered. For example, consider parsing the phrase "show me the ones that leave in the morning" from Example 1(b) in Figure 6-1. Given the lexical entry:

$$\text{ones} \quad := \quad N : \lambda x.!\langle e,t\rangle(x)$$

a CCG parser could produce the desired context-independent logical form:

$$\lambda x.!\langle e,t\rangle(x) \wedge during(x, morning)$$

Our first extension is to simply introduce lexical items that include references into the CCG lexicon. They describe anaphoric words, for example including "ones," "those," and "it."

In addition, we sometimes need to introduce references when there is no explicit lexical trigger. For instance, Example 2(c) in Figure 6-1 consists of the single word "cheapest." This query has the same meaning as the longer request "show me the cheapest one," but it does not include the lexical reference. We add three CCG type-shifting rules to handle these cases.

The first two new rules are applicable when there is a category that is expecting an argument with type $\langle e,t\rangle$. This argument is replaced with a $!\langle e,t\rangle$ reference:

$$\begin{aligned} A/B : f \quad &\Rightarrow \quad A : f(\lambda x.!\langle e,t\rangle(x)) \\ A\backslash B : f \quad &\Rightarrow \quad A : f(\lambda x.!\langle e,t\rangle(x)) \end{aligned}$$

For example, using the first rule, we could produce the following parse for Example 2(c)

$$\dfrac{\dfrac{\text{cheapest}}{\begin{array}{c} NP/N \\ \lambda g.argmin(\lambda x.g(x), \lambda y.fare(y)) \end{array}}}{\begin{array}{c} NP \\ argmin(\lambda x.!\langle e,t\rangle(x), \lambda y.fare(y)) \end{array}}$$

where the final category has the desired lambda-caculus expression.

The third rule is motivated by examples such as "show me nonstop flights." Consider this sentence being uttered after Example 1(a) in Figure 6-1. Although there is a complete, context-independent meaning, the request actually restricts the salient set of flights to include only the nonstop ones. To achieve this analysis, we introduce the rule:

$$A : f \quad \Rightarrow \quad A : \lambda x.f(x) \ \wedge \ !\langle e,t\rangle(x)$$

where f is a function of type $\langle e,t\rangle$.

With this rule, we can construct the parse

$$\dfrac{\dfrac{\dfrac{\text{nonstop}}{\underset{\lambda f.\lambda x.f(x) \wedge nonstop(x)}{N/N}} \quad \dfrac{\text{flights}}{\underset{\lambda x.flight(x)}{N}}}{\underset{\lambda x.nonstop(x) \wedge flight(x)}{N}}>}{\underset{\lambda x.nonstop(x) \wedge flight(x) \ \wedge \ !\langle e,t\rangle(x)}{N}}$$

where the last parsing step is achieved with the new type-shifting rule.

These three new parsing rules allow significant flexibility when introducing references. Later, we develop an approach that learns when to introduce references and how to best resolve them.

6.5 Contextual Analysis

In this section, we first introduce the general patterns of context-dependent analysis that we consider. We then formally define derivations that model these phenomena.

6.5.1 Overview

This section presents an overview of the ways that the context C is used during the analysis.

References Every reference expression ($!e$ or $!\langle e,t\rangle$) must be replaced with an expression from the context. For example, in Section 6.3, we considered the following logical form:

$$\lambda x.!\langle e,t\rangle(x) \wedge during(x, morning)$$

In this case, we saw that replacing the $!\langle e,t\rangle$ subexpression with the logical form for Example 1(a), which is directly available in C, produces the desired final meaning.

Elaborations Later statements can expand the meaning of previous ones in ways that are difficult to model with references. For example, consider analyzing Example 2(c) in Figure 6-1. Here the phrase "departing wednesday after 5 o'clock" has a context-independent logical form:[2]

$$\lambda x.day(x, wed) \wedge depart(x) > 1700 \tag{6.1}$$

that must be combined with the meaning of the previous sentence from the current context C:

$$argmin(\lambda x.fight(x) \wedge from(x, mil) \wedge to(x, orl), \quad \lambda y.fare(y))$$

to produce the expression

$$argmin(\lambda x.fight(x) \wedge from(x, mil) \wedge to(x, orl) \wedge day(x, wed) \wedge depart(x) > 1700, \\ \lambda y.fare(y))$$

Intuitively, the phrase "departing wednesday after 5 o'clock" is providing new constraints for the set of flights embedded in the *argmin* expression.

We handle examples of this type by constructing *elaboration expressions* from the z_i in C. For example, if we constructed the following function:

[2]Another possible option is the expression $\lambda x.!\langle e,t\rangle \wedge day(x, wed) \wedge depart(x) > 1700$. However, there is no obvious way to resolve the $!\langle e,t\rangle$ expression that would produce the desired final meaning.

$$\lambda f.argmin(\lambda x.fight(x) \wedge from(x, mil) \wedge to(x, orl) \wedge f(x)$$
$$\lambda y.fare(y)) \quad (6.2)$$

we could apply this function to Expression 6.1 and produce the desired result. The introduction of the new variable f provides a mechanism for expanding the embedded subexpression.

References with Deletion When resolving references, we will sometimes need to delete subparts of the expressions that we substitute from the context. For instance, consider Example 3(b) in Figure 6-1. The desired, final logical form is:

$$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la) \wedge day(x, thur) \wedge during(x, afternoon)$$

We need to construct this from the context-independent logical form:

$$\lambda x.!\langle e, t\rangle \wedge day(x, thur) \wedge during(x, afternoon)$$

The reference $!\langle e, t\rangle$ must be resolved. The only expression in the context C is the meaning from the previous sentence, Example 3(a):

$$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la) \wedge day(x, thur) \wedge during(x, evening) \quad (6.3)$$

Substituting this expression directly would produce the following logical form:

$$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la) \wedge day(x, thur) \wedge during(x, evening)$$
$$\wedge\ day(x, thur) \wedge during(x, afternoon)$$

which specifies the day twice and has two different time spans.

We can achieve the desired analysis by deleting parts of expressions before they

are substituted. For example, we could remove the day and time constraints from Expression 6.3 to create:

$$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la)$$

which would produce the desired final meaning when substituted into the original expression.

Elaborations with Deletion We also allow deletions for elaborations. In this case, we delete subexpressions of the elaboration expression that is constructed from the context.

6.5.2 Derivations

We now formally define a *derivation* that maps a sentence w_j and a context $C = \{z_1, \ldots, z_{j-1}\}$ to an output logical form z_j. We first introduce notation for expressions in C that we will use in the derivation steps. We then present a definition of deletion. Finally, we define complete derivations.

Context Sets Given a context C, our algorithm constructs three sets of expressions:

- $R_e(C)$: A set of e-type expressions that can be used to resolve references.
- $R_{\langle e,t \rangle}(C)$: A set of $\langle e, t \rangle$-type expressions that can be used to resolve references.
- $E(C)$: A set of possible elaboration expressions (for example, see Expression 6.2).

We will provide the details of how these sets are defined in Section 6.5.3. As an example, if C contains only the logical form

$$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la)$$

then $R_e(C) = \{pit, la\}$ and $R_{\langle e,t \rangle}(C)$ is a set that contains a single entry, the complete logical form.

Deletion A deletion operator accepts a logical form l and produces a new logical form l'. It constructs l' by removing a single subexpression that appears in a coordination (conjunction or disjunction) in l. For example, if l is

$$\lambda x.flight(x) \wedge from(x, pit) \wedge to(x, la)$$

there are three possible deletion operations, each of which removes a single subexpression.

Derivations We now formally define a *derivation* to be a sequence $d = (\Pi, s_1, \ldots, s_m)$. Π is a CCG parse that constructs a context-independent logical form π with $m - 1$ reference expressions.[3] Each s_i is a function that accepts as input a logical form, makes some change to it, and produces a new logical form that is input to the next function s_{i+1}. The initial s_i for $i < m$ are *reference steps.* The final s_m is an optional *elaboration step.*

- **Reference Steps:** A reference step is a tuple $(l, l', f, r, r_1, \ldots, r_p)$. This operator selects a reference f in the input logical form l and an appropriately typed expression r from either $R_e(C)$ or $R_{\langle e,t \rangle}(C)$. It then applies a sequence of p deletion operators to create new expressions $r_1 \ldots r_p$. Finally, it constructs the output logical form l' by substituting r_p for the selected reference f in l.

- **Elaboration Steps:** An elaboration step is a tuple $(l, l', b, b_1, \ldots, b_q)$. This operator selects an expression b from $E(C)$ and applies q deletions to create new expressions $b_1 \ldots b_q$. The output expression l' is $b_q(l)$.

In general, the space of possible derivations is large. In Section 6.6, we describe a linear model and decoding algorithm that we use to find high scoring derivations.

6.5.3 Context Sets

For a context $C = \{z_1, \ldots, z_{j-1}\}$, we define sets $R_e(C)$, $R_{\langle e,t \rangle}(C)$, and $E(C)$ as follows.

[3] In practice, π rarely contains more than one reference.

e-type Expressions $R_e(z)$ is a set of e-type expressions extracted from a logical form z. We define $R_e(C) = \bigcup_{i=1}^{j-1} R_e(z_i)$.

$R_e(z)$ includes all e-type subexpressions of z.[4] For example, if z is

$$argmin(\lambda x.flight(x) \wedge from(x, mil) \wedge to(x, orl), \quad \lambda y.fare(y))$$

the resulting set is $R_e(z) = \{mil, orl, z\}$, where z is included because the entire $argmin$ expression has type e.

$\langle e,t \rangle$-type Expressions $R_{\langle e,t \rangle}(z)$ is a set of $\langle e,t \rangle$-type expressions extracted from a logical form z. We define $R_{\langle e,t \rangle}(C) = \bigcup_{i=1}^{j-1} R_{\langle e,t \rangle}(z_i)$.

The set $R_{\langle e,t \rangle}(z)$ contains all of the $\langle e,t \rangle$-type subexpressions of z. For each quantified variable x in z, it also contains a function $\lambda x.g$. The expression g contains the subexpressions in the scope of x that do not have free variables. For example, if z is

$$\lambda y.\exists x.flight(x) \wedge from(x, bos) \wedge to(x, phi) \wedge during(x, morning) \wedge aircraft(x) = y$$

$R_{\langle e,t \rangle}(z)$ would contain two functions: the entire expression z and the function

$$\lambda x.flight(x) \wedge from(x, bos) \wedge to(x, phi) \wedge during(x, morning)$$

constructed from the variable x, where the subexpression $aircraft(x) = y$ has been removed because it contains the free variable y.

Elaboration Expressions Finally, $E(z)$ is a set of elaboration expressions constructed from a logical form z. We define $E(C) = \bigcup_{i=1}^{j-1} E(z_i)$.

$E(z)$ is defined by enumerating the places where embedded variables are found in z. For each logical variable x and each coordination (conjunction or disjunction) in the scope of x, a new expression is created by defining a function $\lambda f.z'$ where z' has the function $f(x)$ added to the appropriate coordination. This procedure would

[4] A lambda-calculus expression can be represented as a tree structure with flat branching for coordination (conjunction and disjunction). The subexpressions are the subtrees.

produce the example elaboration Expression 6.2 and elaborations that expand other embedded expressions, such as the quantifier in Example 1(c).

6.6 A Linear Model

In general, there will be many possible derivations d for an input sentence w in the current context C. In this section, we introduce a weighted linear model that scores derivations and a decoding algorithm that finds high scoring analyses.

We define $\text{GEN}(w; C)$ to be the set of possible derivations d for an input sentence w given a context C, as described in Section 6.5.2. Let $\phi(d) \in \mathbb{R}^m$ be an m-dimensional feature representation for a derivation d and $\theta \in \mathbb{R}^m$ be an m-dimensional parameter vector. The optimal derivation for a sentence w given context C and parameters θ is

$$d^*(w; C) = \arg \max_{d \in \text{GEN}(w;C)} \theta \cdot \phi(d)$$

Decoding We now briefly describe an approximate algorithm for computing $d^*(w; C)$. Appendix B provides the full details.

The CCG parser uses a CKY-style chart parsing algorithm that prunes to the top $N = 50$ entries for each span in the chart.

We use a beam search procedure to find the best contextual derivations, with beam size $N = 50$. The beam is initialized to the top N logical forms from the CCG parser. The derivations are extended with reference and elaboration steps. The only complication is selecting the sequence of deletions. For each possible step, we use a greedy search procedure that selects the sequence of deletions that would maximize the score of the derivation after the step is applied.

6.7 Learning

Figure 6-2 details the complete learning algorithm. Training is online and error-driven. Step 1 parses the current sentence in context. If the optimal logical form is

Inputs:

- Training examples $\{I_i | i = 1 \ldots n\}$. Each I_i is a sequence $\{(w_{i,j}, z_{i,j}) : j = 1 \ldots n_i\}$ where $w_{i,j}$ is a sentence and $z_{i,j}$ is a logical form.
- Number of training iterations T.
- Initial parameters θ.

Definitions:

- The function $\phi(d)$ represents the features described in Section 6.8.
- GEN$(w; C)$ is the set of derivations for sentence w in context C.
- GEN$(w, z; C)$ is the set of derivations for sentence w in context C that produce the final logical form z.
- The function $L(d)$ maps a derivation to its associated final logical form.

Algorithm:

- For $t = 1 \ldots T, i = 1 \ldots n$: (Iterate interactions)
  - Set $C = \{\}$. (Reset context)
  - For $j = 1 \ldots n_i$: (Iterate training examples)

    Step 1: (Check correctness)

    - Let $d^* = \arg\max_{d \in \text{GEN}(w_{i,j}; C)} \theta \cdot \phi(d)$.
    - If $L(d^*) = z_{i,j}$, go to Step 3.

    Step 2: (Update parameters)

    - Let $d' = \arg\max_{d \in \text{GEN}(w_{i,j}, z_{i,j}; C)} \theta \cdot \phi(d)$.
    - Set $\theta = \theta + \phi(d') - \phi(d^*)$.

    Step 3: (Update context)

    - Append $z_{i,j}$ to the current context C.

Output: Estimated parameters θ.

Figure 6-2: An online learning algorithm.

not correct, Step 2 finds the best derivation that produces the labeled logical form[5] and does an additive, perceptron-style parameter update. Step 3 updates the context. This algorithm is a direct extension of the one described in Chapter 5. It maintains the context but does not have the lexical induction step that was previously used.

6.8 Features

We now describe the features for both the parsing and context resolution stages of the derivation.

6.8.1 Parsing Features

The parsing features are used to score the context-independent CCG parses during the first stage of analysis. We use the set described in Section 5.3, which includes features that are sensitive to lexical choices and the structure of the logical form that is constructed.

6.8.2 Context Features

The context features are functions of the derivation steps described in Section 6.5.2. In a derivation for sentence j of an interaction, let l be the input logical form when considering a new step s (a reference or elaboration step). Let c be the expression that s selects from a context set $R_e(z_i)$, $R_{\langle e,t\rangle}(z_i)$, or $E(z_i)$, where z_i, $i < j$, is an expression in the current context. Also, let r be a subexpression deleted from c. Finally, let f_1 and f_2 be predicates, for example $from$ or to.

Distance Features The distance features are binary indicators on the distance $j - i$. These features allow the model to, for example, favor resolving references with lambda-calculus expressions recovered from recent sentences.

[5]For this computation, we use a modified version of the beam search algorithm described in Section 6.6, which prunes derivations that could not produce the desired logical form.

	Train	Dev.	Test	All
Interactions	300	99	127	526
Sentences	2956	857	826	4637

Table 6.1: Statistics of the ATIS training, development and test (DEC94) sets, including the total number of interactions and sentences. Each interaction is a sequence of sentences.

Copy Features For each possible f_1 there is a feature that tests if f_1 is present in the context expression c but not in the current expression l. These features allow the model to learn to select expressions from the context that introduce expected predicates. For example, flights usually have a $from$ predicate in the current expression.

Deletion Features For each pair (f_1, f_2) there is a feature that tests if f_1 is in the current expression l and f_2 is in the deleted expression r. For example, if $f_1 = f_2 =$ *days* the model can favor overriding old constraints about the departure day with new ones introduced in the current utterance. When $f_1 =$ *during* and $f_2 =$ *depart_time* the algorithm can learn that specific constraints on the departure time override more general constraints about the period of day.

6.9 Evaluation

Data In this section, we present experiments in the context-dependent ATIS domain (Dahl et al., 1994). Table 6.1 presents statistics for the training, development, and test sets. To facilitate comparison with previous work, we used the standard DEC94 test set. We randomly split the remaining data to make training and development sets. We manually converted the original SQL meaning annotations to lambda-calculus expressions.

Evaluation Metrics Miller et al. (1996) report accuracy rates for recovering correct SQL annotations on the test set. For comparison, we report *exact accuracy* rates for recovering completely correct lambda-calculus expressions.

System	Partial Match			Exact
	Prec.	Rec.	F1	Acc.
Full Method	95.0	96.5	95.7	83.7
Miller et al.	–	–	–	78.4

Table 6.2: Performance on the ATIS DEC94 test set.

We also present precision, recall and F-measure for *partial match* results that test if individual attributes, such as the from and to cities, are correctly assigned. This metric is defined in Section 5.5.

Initialization and Parameters The CCG lexicon is hand engineered. We constructed it by running the algorithm from Chapter 5 to learn a lexicon on the context-independent ATIS data set and making manual corrections to improve performance on the training set. We also added lexical items with reference expressions, as described in Section 6.4.

We ran the learning algorithm for $T = 4$ training iterations. The parsing feature weights were initialized as in Chapter 5. The context distance features were given small negative weights (-0.1) to discourage analyses that use the context. All other feature weights were initially set to zero.

Test Setup During evaluation, the context $C = \{z_1 \dots z_{j-1}\}$ contains the logical forms output by the learned system for the previous sentences. In general, errors made while constructing these expressions can propogate if they are used in derivations for new sentences.

Results Table 6.2 shows performance on the ATIS DEC94 test set. Our approach correctly recovers 83.7% of the logical forms. This result compares favorably to Miller et al.'s fully-supervised approach (Miller et al., 1996) while requiring significantly less annotation effort.

We also evaluated performance when the context is limited to contain only the M most recent logical forms. Table 6.3 shows results on the development set for different

Limited Context	Partial Match			Exact
	Prec.	Rec.	F1	Acc.
$M = 0$	96.2	57.3	71.8	45.4
$M = 1$	94.9	91.6	93.2	79.8
$M = 2$	94.8	93.2	94.0	81.0
$M = 3$	94.5	94.3	94.4	82.1
$M = 4$	94.9	92.9	93.9	81.6
$M = 10$	94.2	94.0	94.1	81.4

Table 6.3: Performance on the ATIS development set for varying context window lengths M.

values of M. The poor performance with no context ($M = 0$) demonstrates the need for context-dependent analysis. Limiting the context to the most recent statement ($M = 1$) significantly improves performance while using the last three utterances ($M = 3$) provides the best results.

Finally, we evaluated a variation where the context contains gold-standard logical forms during evaluation instead of the output of the learned model. On the development set, this approach achieved 85.5% exact-match accuracy, an improvement of approximately 3% over the standard approach. This result suggests that incorrect logical forms in the context have a relatively limited impact on overall performance.

6.10 Summary

In this chapter, we addressed the problem of learning context-dependent mappings from sentences to logical form. We developed a context-dependent analysis model and showed that it can be effectively trained with a hidden-variable variant of the perceptron algorithm. In the experiments, we showed that the approach recovers fully correct logical forms with 83.7% accuracy.

Chapter 7

Conclusion

This thesis presented algorithms for learning to map sentences to logical form. We described how to learn an extension of Combinatory Categorial Grammar (CCG) (Steedman, 1996, 2000) that is augmented with a model for context-dependent analysis. The resulted was a unified approach that (1) represents lexical semantics for individual words, (2) includes a probabilistic parsing model for analyzing individual sentences, and (3) includes a probabilistic model for reasoning about context dependence.

Chapter 4 presented an initial approach for the context-independent learning problem, where sentences are analyzed in isolation. We described a procedure, GENLEX, for creating a large set of linguistically-plausible CCG lexical items. We then developed a learning algorithm for probabilistic CCGs that prunes this lexicon while estimating parameters of the log-linear parsing model. Finally, we demonstrated experimentally that this approach is competitive with previous learning methods.

In Chapter 5, we presented a new, online algorithm for learning a CCG, together with parameters that define a linear parsing model. We showed that the use of non-standard CCG combinators is highly effective for parsing sentences with the types of phenomena seen in spontaneous, unedited natural language. The resulting system achieved significant accuracy improvements in both the ATIS and Geo880 domains.

Finally, Chapter 6 addressed the problem of learning context-dependent mappings from sentences to logical form. We developed a context-dependent analysis model

and showed that it can be effectively trained with a hidden-variable variant of the perceptron algorithm. In the experiments, we showed that the approach recovers fully correct logical forms with 83.7% accuracy.

7.1 Future Work

There are a number of potential areas for future work. The possibilities include extensions to the learning approach and applications to other semantic analysis problems.

Algorithmic Extensions The CCG grammar induction techniques we developed required two types of hand-engineered knowledge: the GENLEX rules and the initial lexicon. In future work, both could be automatically induced. One possibility is to develop an approach based on higher-order unification. We might first induce the meaning of simple phrases, such as noun phrases. Then, we could use an unification algorithm to find more complex categories that combine the simple ones and produce the desired logical forms. Automatically inducing a complete grammar would make the approach applicable to languages other than English, with no additional engineering effort.

Broad Coverage Semantic Analysis Another direction is to scale the approach towards broad-coverage semantic analysis. This would require us to model and recover many additional types of semantic phenomena. For example, we would need to model events occurring in time and understand a range of complex verb tenses that rarely occur in existing natural language interfaces to databases. Annotating the logical forms of the training text will be a major challenge; there is no current consensus for how to best represent the meaning of open-domain text. One important direction will be to develop semisupervised and unsupervised algorithms to minimize the data annotation requirements.

Dialog Systems The context-dependent analysis problem in Chapter 6 is a simple dialog problem, where the user has complete control of the conversation and the sys-

tem passively answers questions. More general dialog systems are also possible. They would require an integrated approach for recovering logical forms and performing decision-theoretic reasoning about what to say to future the current interaction. One possibility would be to develop first-order partially observable Markov decision processes that represent the user's goals as hidden state. This would provided a unified mechanisms for probabilistic pragmatic reasoning in conversations where the user's statements can have complex meanings.

Appendix A

A CCG Parsing Algorithm

In this section, we describe a chart parsing algorithm for weighted Combinatory Categorial Grammars (CCGs).

As described in Section 2.2, a CCG is defined by:

- A lexicon Λ. Each lexical item in Λ pairs a sequence of words with a CCG category. For example, one lexical item might contain the words *the Mississippi* and the category $NP : mississippi\_river$.

- A set of combinators, including both unary type-raising rules and binary parsing rules. A unary rule $t(c)$ accepts a category c and produces a new category. A binary rule $b(c_1, c_2)$ accepts two categories and produces a new category. The category c_1 is the left category, which precedes the right category c_2. Both types of rules return *null* if they are not applicable to input categories. These rules are used to construct intermediate categories during parsing, as described in Section 2.2.

The lexicon and combinators define the space of possible parse trees for an input sentence w.

We describe a parsing algorithm for weighted CCGs, which additionally include:

- A parameter vector $\theta \in \mathbb{R}^d$.

- A feature representation $\phi(w, y) \in \mathbb{R}^d$ that is a function of a sentence w and a parse tree y.

Given a weighted CCG, we define the score of a sentence and a parse as the dot product $\theta \cdot \phi(w, y)$.

We describe two instances of the parsing problem. In the first, we are given a sentence w and must find the highest scoring parse y for w. In the second, we are given a sentence w and a logical form z and must find the highest scoring parse y that has the final logical form z.

In both cases, we use a CKY-style parsing algorithm to construct a *parse chart* that compactly represents a large set of possible parse trees. In practice, it is not possible to represent every possible parse. Instead, we prune the chart given the information available. Given only the sentence w, we prune low scoring intermediate parses. When also given the final logical form z, we use a simple test to additionally prune a large set of parses that could not produce z. The resulting approach is efficient and considers a large subset of the possible parse trees.

A.1 The Algorithm

Given a sentence w containing n words and a weighted CCG, we now describe how to construct a *parse chart*.

The chart contains *edges* that are organized into *spans*. Each span $C[i, j]$ for $0 \leq i \leq j < n$ contains a set of edges that define the possible root categories for CCG parse trees that span the words in w from index i to index j. Each edge $e = (c, p, s)$ is a tuple containing a CCG category c, the CCG combinator p used to construct c, and a real-valued score s. Parse trees are extracted from a chart by selecting an appropriate set of edges.

To allow for dynamically programming, we restrict the features in ϕ to be a sum of features defined on chart edges. For a CCG parse y defined by the set of edges E:

$$\phi(y) = \sum_{e \in E} \phi(e)$$

```
parse(){
  add_lexical_items();
  // n is the number of words in the sentence
  for(span=2; span<n; span++){
    for(start=0; span<n-span+1; start++){
       end = start + span - 1;
       apply_binary_combinators(start,end);
       apply_type_raising(start,end);
       prune(start,end);
    }
  }
}
```

Figure A-1: A CCG parsing algorithm.

```
add_lexical_items(){
  for(span=2; span<n; span++){
    for(start=0; span<n-span+1; start++){
       end = start + span - 1;
       foreach lexical item with words w[start,end] and CCG category c {
          add_edge(start,end,c,null,{});
       }
       apply_type_raising(start,end);
    }
  }
}
```

Figure A-2: Initializing the parse chart.

This decomposition allows us to recursively compute the max score over all possible subtrees that produce the same edge in the same span.

Figure A-1 contains the procedure `parse()` that constructs the parse chart. This procedure first adds edges for all of the lexical items that match the words in the input sentence w, using the subprocedure `add_lexical_items()` in Figure A-2. It then constructs all of the chart spans, starting from those of length two and increasing to length n.

The edges in each span $C[i, j]$ are constructed with a three step process. First,

we add all possible combinations of two contiguous subspans, using the procedure `apply_binary_combinators(i,j)` in Figure A-3. Next, we apply type raising operators to all of these new edges, using the procedure `apply_type_raising(i,j)` in Figure A-3. Finally, we prune the span $C[i,j]$ to limit the number of subedges that must be considered when constructing longer edges. This is done with the `prune(i,j)` procedure, which we present in the next section.

The `add_edge(i,j,c,o,B)` procedure adds an edge e to the chart span $C[i,j]$ for a category c created by a CCG combinator o from the edges represented in the set B. This procedure computes the score $s(e)$ for the new edge e. For example, if o is a binary operator, then the score is:

$$s(e) = \max_{e_1,e_2} \phi(e) \cdot \theta + s(e_1) + s(e_2)$$

where the max is over all pairs of edges (e_1 and e_2) that can be combined with operator o to create a new edge with category c for the span $C[i,j]$.

Given a parse chart, we can find the highest scoring parse by selecting the edge in the span $C[0, n-1]$ with the highest score and recursively tracing back the edges in the subspans that were used to construct it.

A.1.1 Pruning

In general, there will be a large number of possible edges for each chart span $C[i,j]$. We use the procedure `prune(i,j)` to prune this set while retaining high scoring entries. We consider both the case where we are only given an input sentence w and the case where we are additionally given a desired logical form z.

Given only w, we remove all but the N highest scoring edges in $C[i,j]$. This type of *beam pruning* is a commonly used method in chart parsing algorithms that can work well in practice.

When we are also given a target logical form z, we would like to prune edges that could not possibly be used in a parse that would produce z. We use the following strategy. Let f be a constant in the logical language, for example *from* or *Texas*.

```
apply_type_raising(start,end){
  foreach edge e in C[start,end]{
     if e was created by type-raising, continue with next edge;
     let c be the CCG category in e;
     for each CCG type raising operator t {
       let c_t = t(c);
       if (c_t != null){
         add_edge(start,end,c_t,t,{e});
       }
    }
  }
}

apply_binary_combinators(start,end){
  for (split = start; split<end-1; split++){
    foreach edge e1 in C[start,split] with CCG category c1 {
      foreach edge e2 in C[split+1,end] with CCG category c2 {
        foreach binary CCG Combinator b {
          let c_b = b(c1,c2);
          if (c_b != null){
            add_edge(start,end,c_b,b,{e1,e2});
          }
        }
      }
    }
  }
}

add_edge(start,end,c,o,B){
  create new edge e with category c and operator o;
  calculate score s for e as local score
        plus the sum of scores of edges in B;
  foreach edge e_p with CCG Category c_p
        and score s_p in chart span C[start,end] {
    if (c equals c_p && s>s_p){
      replace e_p with e;
      return;
    }
    add e to the C[start,end];
  }
}
```

Figure A-3: Procedures for adding edges to the parse chart.

Also, define $N(f, z)$ to be the count of the number of times the constant f appears in the logical expression z. For each edge $e \in C[i, j]$, let $l(e)$ be the lambda-calculus expression in its CCG category. We prune all edges e where there exists a constant f such that $N(f, l(e)) > N(f, z)$. We can safely prune all edges whose meanings have more instances of some constant than the desired logical form. There are no parsing operations that remove constants from the semantics.

A.1.2 Parsing with PCCGs

The procedure `parse()` in Figure A-1 can also be used for parsing with probabilistic CCGs (PCCGs), which were defined in Section 4.2.1. The only difference is the way that the score is computed for each edge. For PCCGs, the score includes a the sum over the scores of all possible subtrees rooted with the edge, instead of a max.

We modify the `add_edge` procedure to sum over edge scores while compute the score $s(e)$ for a new edge e in chart span $C[i, j]$. We assume that e has CCG category c, which was created with a CCG operator o. For example, if o is a binary operator, then the score is:

$$s(e) = \sum_{e_1, e_2} e^{\phi(e) \cdot \theta} \cdot s(e_1) \cdot s(e_2)$$

where the sum is over all pairs of edges (e_1 and e_2) that can be combined with operator o to create a new edge with category c for the span $C[i, j]$. This sum is incrementally computed as each new edge is added to the chart.

Appendix B

A Beam Decoding Algorithm for Context-dependent Analysis

In this chapter, we present an algorithm for finding high scoring derivations for the context-dependent analysis problem defined in Chapter 6. A *derivation* maps a sentence w_j and a context $C = \{z_1, \ldots, z_{j-1}\}$ to an output logical form z_j. Each derivation is a sequence $d = (\Pi, s_1, \ldots, s_m)$. Π is a CCG parse that constructs a context-independent logical form π with $m - 1$ reference expressions. Each s_i is a function that accepts as input a logical form, makes some change to it, and produces a new logical form that is input to the next function s_{i+1}. The final expression is the output logical form z_j.

As described in Section 6.6, we define the score of a derivation d to be the dot product $\theta \cdot \phi(d)$, given a parameter vector θ and a feature function ϕ. We consider two decoding problems. The first is to find the highest scoring derivation for a sentence w_j and a context C. The second problem is to find the highest scoring derivation for w_j and C that constructs a specific logical form z_j.

Figure B-1 present a simple beam search algorithm `decode()` that applies to both cases by incrementally constructing a list of the N highest scoring derivations. First, we use the CCG parsing algorithm described in Appendix A to find the top N CCG parses, and create a derivation for each. Next, we repeatedly expand the derivations by appending derivation steps. Section 6.5.2 defines the two types of derivation steps

```
decode(){
  call parse();
  make a derivation for each of the N highest scoring parses;

  while exists a derivation d with an unresolved reference {
      remove d from the beam;
      for each possible reference operator r {
        create a new derivation d' by appending r to d;
        add d' to beam;
      }
      prune();
  }

  foreach derivation d in the beam {
    foreach elaboration operator e {
        create a new derivation d' by appending e to d;
        add d' to beam;
    }
    prune();
  }
}
```

Figure B-1: A beam decoding algorithm for context-dependent analysis.

(reference and elaboration) and describes the space of possible steps at each point in a derivation. After each expansion, the `prune()` procedure prunes the set of derivations to include only the N highest scoring options.

In the rest of this chapter, we describe the details of the `decode()` algorithm. First, we describe how to prune the beam. Then we present a method for performing deletions, which must be done each time a derivation step is added.

B.1 Pruning

The `prune()` procedure ensures that the list of possible derivations contains no more than N entries. We prune differently depending on the decoding problem — whether we are given only a sentence w_j and context C, or we are additionally given a target logical form z_j.

Given only w_j and C, we simply remove all but the N highest scoring derivations.

When we are given a target logical form z_j, we also prune derivations that could not possibly produce z_j. We use the same simple strategy as in Appendix A based on counting the constants in the logical form. Let f be a constant in the logical language, for example *from* or *Texas*. Also, define $N(f, z)$ to be the count of the number of times the constant f appears in the logical expression z. For each derivation d, let $l(d)$ be the output lambda-calculus expression. We prune all derivations d where there exists a constant f such that $N(f, l(d)) > N(f, z_j)$. If the meaning associated with the derivation has more instances of a constant than the desired logical form, we can safely prune it. There are no derivation steps that remove constants from the output semantics.

B.2 Deletion

In a derivation d, each new step s_i involves using the context C to construct a logical expression l that is combined with the output logical form defined by d. Section 6.5.2 defines the options for constructing l and integrating it into the analysis. One important subproblem is selecting a sequence of *deletion operators* to apply to l. We describe an approach in this section.

A deletion operator accepts a logical form l and produces a new logical form l'. It constructs l' by removing a single subexpression that appears in a coordination (conjunction or disjunction) in l.

We use a greedy search procedure to select the sequence of deletions. Given the initial logical expression l, we repeated consider each possible deletion. We compute the score of the new derivation d' that would result if the operator were applied and select the operator that produces the highest score. This operator is applied and the process repeats until we can no longer improve the score of the derivation.

Appendix C

The Domain Independent Lexicon

Figure C-1 shows example lexical entries from the domain-independent initial lexicon. Each row includes a set of phrases that are assigned the same category. A vertical slash (|) is a compact notation for either a forward slash (/) or back slash (\). The complete fixed lexicon contains 185 entries. They include standard CCG lexical entries for closed class words such as determiners, conjunctions, and quantifiers.

{and, but}	:=	$CONJ : \wedge$
{or, and}	:=	$CONJ : \vee$
{is, are, does, ...}	:=	$(S\|NP)/(S\|NP) : \lambda g.g$
{the, a, an,...}	:=	$NP/N : \lambda f.f$
{the number of, ...}	:=	$NP/N : \lambda f.count(\lambda x.f(x))$
{how many, ...}	:=	$(S/(S\|NP))/N : \lambda g.\lambda f.count(\lambda x.f(x) \wedge g(x))$
{with the largest, ...}	:=	$(NP\backslash N)/N : \lambda g.\lambda f.argmax(\lambda x.f(x), \lambda x.g(x))$
{with the smallest, ...}	:=	$(NP\backslash N)/N : \lambda g.\lambda f.argmin(\lambda x.f(x), \lambda x.g(x))$
{do not, are not ...}	:=	$(S\backslash NP)/(S\backslash NP) : \lambda f.\lambda x.\neg f(x)$
{and also, and their, ...}	:=	$(N\backslash N)/N : \lambda g.\lambda f.\lambda x.\lambda y.f(x) \wedge g(x) = y$
{for, on, of, ...}	:=	$(N\backslash N)/N : \lambda g.\lambda f.\lambda y.\exists x.f(x) \wedge g(x) = y$
{every, all, ...}	:=	$(S/(S\|NP))/N : \lambda f.\lambda g.\forall x.f(x) \rightarrow g(x)$
{some, ...}	:=	$(S/(S\|NP))/N : \lambda f.\lambda g.\exists x.f(x) \wedge g(x)$
{no, ...}	:=	$(S/(S\|NP))/N : \lambda f.\lambda g.\neg\exists x.f(x) \wedge g(x)$
{a, an, ...}	:=	$((S\backslash NP)\backslash(S\backslash NP/NP))/N : \lambda f.\lambda g\lambda y.\exists x.f(x) \wedge g(x,y)$
{no, ...}	:=	$((S\backslash NP)\backslash(S\backslash NP/NP))/N : \lambda f.\lambda g\lambda y.\neg\exists x.f(x) \wedge g(x,y)$

Figure C-1: Entries from the domain-independent fixed lexicon.

Bibliography

Alshawi, H. (1992). *The Core Language Engine.* The MIT Press.

Androutsopoulos, I., Ritchie, G., & Thanisch, P. (1995). Natural language interfaces to databases–an introduction. *Journal of Language Engineering, 1*(1), 29–81.

Baker, J. K. (1979). Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America.*

Baldridge, J. (2002). *Lexically Specified Derivational Control in Combinatory Categorial Grammar.* Ph.D. thesis, University of Edinburgh.

Bos, J., Clark, S., Steedman, M., Curran, J. R., & Hockenmaier, J. (2004). Wide-coverage semantic representations from a CCG parser. In *Proceedings of the International Conference on Computational Linguistics.*

Bozsahin, C. (1998). Deriving the predicate-argument structure for a free word order language. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics.*

Carbonell, J. G., & Hayes, P. J. (1983). Recovery strategies for parsing extragrammatical language. *American Journal of Computational Linguistics, 9.*

Carpenter, B. (1997). *Type-Logical Semantics.* The MIT Press.

Charniak, E. (1972). *Toward a model of children's story comprehension.* Ph.D. thesis, Massachusetts Institute of Technology.

Clark, S., & Curran, J. R. (2003). Log-linear models for wide-coverage CCG parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.*

Clark, S., & Curran, J. R. (2007). Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, *33*(4), 493–552.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.*

Collins, M. (2004). Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In Harry Bunt, John Carroll and Giorgio Satta, editors, *New Developments in Parsing Technology.* Kluwer.

Dahl, D. A., Bates, M., Brown, M., Fisher, W., Hunicke-Smith, K., Pallett, D., Pao, C., Rudnicky, A., & Shriberg, E. (1994). Expanding the scope of the ATIS task: the ATIS-3 corpus. In *ARPA HLT Workshop.*

Ge, R., & Mooney, R. J. (2005). A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Conference on Computational Natural Language Learning.*

Ge, R., & Mooney, R. J. (2006). Discriminative reranking for semantic parsing. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions.*

Ge, R., & Mooney, R. J. (2009). Learning a compositional semantic parser using an existing syntactic parser. In *Joint conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing.*

He, Y., & Young, S. (2005). Semantic processing using the hidden vector state model. *Computer Speech and Language.*

He, Y., & Young, S. (2006). Spoken language understanding using the hidden vector state model. *Speech Communication*, *48*(3-4).

Hockenmaier, J., & Steedman, M. (2002). Generative models for statistical parsing with combinatory categorial grammar. In *Annual Meeting of the Association for Computational Linguistics.*

Hockenmaier, J., & Steedman, M. (2007). CCGbank: a corpus of CCG derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, *33*(3), 355–396.

Joachims, T. (1999). Transductive inference for text classification using support vector machines. In *Proceedings of the International Conference on Machine Learning.*

Johnson, M., Geman, S., Canon, S., Chi, Z., & Riezler, S. (1999). Estimators for stochastic "unification-based" grammars. In *Proc. of the Association for Computational Linguistics.*

Kate, R. J., & Mooney, R. J. (2006). Using string-kernels for learning semantic parsers. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics.*

Kate, R. J., & Mooney, R. J. (2007a). Learning language semantics from ambiguous supervision. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence.*

Kate, R. J., & Mooney, R. J. (2007b). Semi-supervised learning for semantic parsing using support vector machines. In *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics.*

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning.*

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Levin, E., Narayanan, S., Pieraccini, R., Biatov, K., Bocchieri, E., Fabbrizio, G. D., Eckert, W., Lee, S., Pokrovsky, A., Rahim, M., Ruscitti, P., & Walker, M. (2000). The AT&T darpa communicator mixed-initiative spoken dialogue system. In *Proceedings of the International Conference on Spoken Language Processing.*

Liang, P., Bouchard-Côté, A., Klein, D., & Taskar, B. (2006). An end-to-end discriminative approach to machine translation. In *Proc. of the Annual Meeting of the Association for Computational Linguistics.*

Lu, W., Ng, H. T., Lee, W. S., & Zettlemoyer, L. S. (2008). A generative model for parsing natural language to meaning representations. In *Proceedings of The Conference on Empirical Methods in Natural Language Processing.*

Manning, C. D., & Schutze, H. (1999). *Foundations of statistical natural language processing.* The MIT Press.

Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational Linguistics, 19*(2), 313–330.

Miller, S., Stallard, D., Bobrow, R. J., & Schwartz, R. L. (1996). A fully statistical approach to natural language interfaces. In *Proc. of the Association for Computational Linguistics.*

Montague, R. (1970a). English as a formal language. In *Linguaggi nella societá e nella tecnica*, pp. 189–223.

Montague, R. (1970b). Universal grammar. *Theoria, 36*, 373–398.

Montague, R. (1973). The proper treatment of quantification in ordinary english. In *Approaches to Natural Language*, pp. 221–242.

Papineni, K. A., Roukos, S., & Ward, T. R. (1997). Feature-based language understanding. In *Proceedings of European Conference on Speech Communication and Technology.*

Pereira, F. C. N., & Shieber, S. M. (1987). *Prolog and natural-language analysis.* Center for the Study of Language and Information.

Popescu, A.-M., Armanasu, A., Etzioni, O., Ko, D., & Yates, A. (2004). Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of the International Conference on Computational Linguistics.*

Ramaswamy, G. N., & Kleindienst, J. (2000). Hierarchical feature-based translation for scalable natural language understanding. In *Proceedings of International Conference on Spoken Language Processing.*

Ratnaparkhi, A., Roukos, S., & Ward, R. T. (1994). A maximum entropy model for parsing. In *Proceedings of the International Conference on Spoken Language Processing.*

Seneff, S. (1992). Robust parsing for spoken language systems. In *Proc. of the IEEE Conference on Acoustics, Speech, and Signal Processing.*

Siskind, J. M. (1996). A computational study of cross-situational techniques for learning word-to-meaning mappings. *Cognition, 61*(2-3).

Steedman, M. (1996). *Surface Structure and Interpretation.* The MIT Press.

Steedman, M. (2000). *The Syntactic Process.* The MIT Press.

Tang, L. R., & Mooney, R. J. (2001). Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the European Conference on Machine Learning.*

Taskar, B., Guestrin, C., & Koller, D. (2003). Max-margin markov networks. In *Neural Information Processing Systems.*

Taskar, B., Klein, D., Collins, M., Koller, D., & Manning, C. (2004). Max-margin parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.*

Thompson, C. A., & Mooney, R. J. (2002). Acquiring word-meaning mappings for natural language interfaces. *Journal of Artificial Intelligence Research, 18.*

Villavicencio, A. (2001). *The acquisition of a unification-based generalised categorial grammar.* Ph.D. thesis, University of Cambridge.

Ward, W. (1991). Understanding spontaneous speech: the phoenix system. In *Proceedings of the Conference on Acoustics, Speech, and Signal Processing.*

Ward, W., & Issar, S. (1994). Recent improvements in the CMU spoken language understanding system. In *Proceedings of the workshop on Human Language Technology*.

Watkinson, S., & Manandhar, S. (1999). Unsupervised lexical learning with categorial grammars using the LLL corpus. In *Proceedings of the 1st Workshop on Learning Language in Logic*.

Webber, B. (1979). *A Formal Approach to Discourse Anaphora*. Garland Publishing.

Winograd, T. (1970). *Procedures as a representation for data in a computer program for understanding natural language*. Ph.D. thesis, Massachusetts Institute of Technology.

Wong, Y. W., & Mooney, R. (2006). Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL*.

Wong, Y. W., & Mooney, R. (2007a). Generation by inverting a semantic parser that uses statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*.

Wong, Y. W., & Mooney, R. (2007b). Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the Association for Computational Linguistics*.

Woods, W. A. (1968). Procedural semantics for a question-answering machine. In *AFIPS '68 (Fall, part I): Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pp. 457–471.

Zelle, J. M., & Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*.

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

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

Zettlemoyer, L. S., & Collins, M. (2009). 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 Processing.*