# Online Learning of Relaxed CCG Grammars for Parsing to Logical Form

**Luke S. Zettlemoyer** and **Michael Collins**
MIT CSAIL
lsz@csail.mit.edu,mcollins@csail.mit.edu

## Abstract

We consider the problem of learning to parse sentences to lambda-calculus representations of their underlying semantics and present an algorithm that learns a weighted combinatory categorial grammar (CCG). 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. 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).

## 1 Introduction

Recent work (Mooney, 2007; He and Young, 2006; Zettlemoyer and Collins, 2005) has developed learning algorithms for the problem of mapping sentences to underlying semantic representations. In one such approach (Zettlemoyer and Collins, 2005) (ZC05), the input to the learning algorithm is a training set consisting of sentences paired with lambda-calculus expressions. For instance, the training data might contain the following example:

Sentence: list flights to boston
Logical Form: $\lambda x.flight(x) \wedge to(x, boston)$

In this case the lambda-calculus expression denotes the set of all flights that land in Boston. In ZC05 it is assumed that training examples do not include additional information, for example parse trees or other derivations. The output from the learning algorithm is a combinatory categorial grammar (CCG), together with parameters that define a log-linear distribution over parses under the grammar. Experiments show that the approach gives high accuracy on two database-query problems, introduced by Zelle and Mooney (1996) and Tang and Mooney (2000).

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 1: Three sentences from the ATIS domain.

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 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 example, consider the sentences shown in figure 1, which were taken from the ATIS travel-planning domain (Dahl et al., 1994). These sentences exhibit characteristics which present significant challenges to the approach of ZC05. For ex-



*Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 678–687, Prague, June 2007. 

ample, the sentences have quite flexible word order, and include telegraphic language where some words are effectively omitted.

In this paper 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 of ZC05 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 algorithms such as the algorithm described in ZC05.

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 manner. In our experiments, we show that the new algorithm is considerably more efficient than the ZC05 algorithm; this is important when training on large training sets, for example the ATIS data used in this paper.

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 in ZC05.

# 2 Background

## 2.1 Semantics

Training examples in our approach consist of sentences paired with lambda-calculus expressions. We use a version of the lambda calculus that is closely related to the one presented by Carpenter (1997). There are three basic types: $t$, the type of truth values; $e$, the type for entities; and $r$, the type for real numbers. Functional types are defined by specifying their input and output types, for example $\langle e, t\rangle$ is the type of a function from entities to truth values. In general, declarative sentences have a logical form of type $t$. Question sentences generally have functional types.[2] Each expression is constructed from constants, logical connectors, quantifiers and lambda functions.

## 2.2 Combinatory Categorial Grammars

Combinatory categorial grammar (CCG) is a syntactic theory that models a wide range of linguistic phenomena (Steedman, 1996; Steedman, 2000). The core of a CCG grammar is a lexicon $\Lambda$. For example, consider the lexicon

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

Each entry in the lexicon is a pair consisting of a word and an associated category. The category contains both syntactic and semantic information. For example, the first entry states that the word *flights* can have the category $N : \lambda x.flight(x)$. This category consists of a syntactic type $N$, together with the semantics $\lambda x.flight(x)$. In general, the semantic entries for words in the lexicon can consist of any lambda-calculus expression. Syntactic types can either be simple types such as $N$, $NP$, or $S$, or can be more complex types that make use of slash notation, for example $(N\backslash N)/NP$.

CCG makes use of a set of *combinators* which are used to combine categories to form larger pieces of syntactic and semantic structure. The simplest such rules are the *functional application* rules:

$$\begin{array}{lllll} A/B : f & B : g & \Rightarrow & A : f(g) & (>) \\ B : g & A\backslash B : f & \Rightarrow & A : f(g) & (<) \end{array}$$

The first rule states that a category with syntactic type $A/B$ can be combined with a category to the right of syntactic type $B$ to create a new category of type $A$. It also states that the new semantics will be formed by applying the function $f$ to the expression $g$. The second rule handles arguments to the left. Using these rules, we can parse the

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

[2] For example, many question sentences have semantics of type $\langle e, t\rangle$, as in $\lambda x.flight(x) \wedge to(x, boston)$.

following phrase to create a new category of type $N$:

$$
\begin{array}{ccc}
\text{flights} & \text{to} & \text{boston} \\
\hline
N & (N\backslash N)/NP & NP \\
\lambda x.flight(x) & \lambda y.\lambda f.\lambda x.f(x) \wedge to(x,y) & boston \\
\cline{2-3}
& \multicolumn{2}{c}{(N\backslash N) \quad >} \\
& \multicolumn{2}{c}{\lambda f.\lambda x.f(x) \wedge to(x, boston)} \\
\hline
\multicolumn{3}{c}{N \quad <} \\
\multicolumn{3}{c}{\lambda x.flight(x) \wedge to(x, boston)}
\end{array}
$$

The top-most parse operations pair each word with a corresponding category from the lexicon. The later steps are labeled $->$ (for each instance of forward application) or $-<$ (for backward application).

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

$$
\begin{array}{llll}
A/B : f & B/C : g & \Rightarrow & A/C : \lambda x.f(g(x)) \quad (> \mathbf{B}) \\
B\backslash C : g & A\backslash B : f & \Rightarrow & A\backslash C : \lambda x.f(g(x)) \quad (< \mathbf{B})
\end{array}
$$

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 addition to the application and composition rules, we will also make use of type raising and coordination combinators. A full description of these combinators goes beyond the scope of this paper. Steedman (1996; 2000) presents a detailed description of CCG.

### 2.3 Log-Linear CCGs

We can generalize CCGs to weighted, or probabilistic, models as follows. Our models are similar to several other approaches (Ratnaparkhi et al., 1994; Johnson et al., 1999; Lafferty et al., 2001; Collins, 2004; Taskar et al., 2004). We will write $x$ to denote a sentence, and $y$ to denote a CCG parse for a sentence. We use GEN$(x; \Lambda)$ to refer to all possible CCG parses for $x$ under some CCG lexicon $\Lambda$. We will define $\mathbf{f}(x, y) \in \mathbb{R}^d$ to be a $d$-dimensional *feature–vector* that represents a parse tree $y$ paired with an input sentence $x$. In principle, $\mathbf{f}$ could include features that are sensitive to arbitrary substructures within the pair $(x, y)$. We will define $\mathbf{w} \in \mathbb{R}^d$ to be a parameter vector. The optimal parse for a sentence $x$ under parameters $\mathbf{w}$ and lexicon $\Lambda$ is then defined as

$$y^*(x) = \arg \max_{y \in \text{GEN}(x;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x, y) \ .$$

Assuming sufficiently local features[3] in $\mathbf{f}$, search for $y^*$ can be achieved using dynamic-programming-style algorithms, typically with some form of beam search.[4] Training a model of this form involves learning the parameters $\mathbf{w}$ and potentially also the lexicon $\Lambda$. This paper focuses on a method for learning a $(\mathbf{w}, \Lambda)$ pair from a training set of sentences paired with lambda-calculus expressions.

### 2.4 Zettlemoyer and Collins 2005

We now give a description of the approach of Zettlemoyer and Collins (2005). This method will form the basis for our approach, and will be one of the baseline models for the experimental comparisons.

The input to the ZC05 algorithm is a set of training examples $(x_i, z_i)$ for $i = 1 \dots n$. Each $x_i$ is a sentence, and each $z_i$ is a corresponding lambda-expression. The output from the algorithm is a pair $(\mathbf{w}, \Lambda)$ specifying a set of parameter values, and a CCG lexicon. Note that for a given training example $(x_i, z_i)$, there may be many possible parses $y$ which lead to the correct semantics $z_i$.[5] For this reason the training problem is a *hidden-variable* problem, where the training examples contain only partial information, and the CCG lexicon and parse derivations must be learned without direct supervision.

A central part of the ZC05 approach is a function GENLEX$(x, z)$ which maps a sentence $x$ together with semantics $z$ to a set of potential lexical entries. The function GENLEX is defined through a set of rules—see figure 2—that consider the expression $z$, and generate a set of categories that may help in building the target semantics $z$. An exhaustive set of lexical entries is then generated by taking all categories generated by the GENLEX rules, and pairing them with all possible sub-strings of the sentence $x$. Note that our lexicon can contain multi-word entries, where a multi-word string such as *New York* can be paired with a CCG category. The final out-

[3]For example, features which count the number of lexical entries of a particular type, or features that count the number of applications of a particular CCG combinator.

[4]In our experiments we 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.

[5]This problem is compounded by the fact that the lexicon is unknown, so that many of the possible hidden derivations involve completely spurious lexical entries.

| 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 2: Rules used in GENLEX. 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. We use the 10 rules described in ZC05 and add two new rules, listed in the last two rows above. This first new rule is instantiated for greater than ($>$) and less than ($<$) comparisions. The second new rule has no trigger; it is always applied. It generates categories that are used to learn lexical entries for semantically vacuous sentence prefixes such as the phrase *show me information on* in the example in figure 1(b).

put from GENLEX$(x, z)$ is a large set of potential lexical entries, with the vast majority of those entries being spurious. The algorithm in ZC05 embeds GENLEX within an overall learning approach that simultaneously selects a small subset of all entries generated by GENLEX and estimates parameter values $\mathbf{w}$. Zettlemoyer and Collins (2005) present more complete details. In section 4.2 we describe a new, online algorithm that uses GENLEX.

## 3 Parsing Extensions: Combinators

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 $\Lambda$. 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 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.

### 3.1 Application and Composition Rules

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

$$\begin{array}{llcll} A\backslash B : f & B : g & \Rightarrow & A : f(g) & (\gtrsim) \\ B : g & A/B : f & \Rightarrow & A : f(g) & (\lesssim) \end{array}$$

These are variants of the original application rules, where the slash direction on the principal categories ($A/B$ or $A\backslash B$) is reversed.[6] These rules allow simple reversing of regular word order, for example

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

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:

$$\begin{array}{llcll} A\backslash B : f & B/C : g & \Rightarrow & A/C : \lambda x.f(g(x)) & (\gtrsim \mathbf{B}) \\ B\backslash C : g & A/B : f & \Rightarrow & A\backslash C : \lambda x.f(g(x)) & (\lesssim \mathbf{B}) \end{array}$$

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

[6]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

$$
\begin{array}{ccc}
\text{to washington} & \text{the latest} & \text{flight} \\
\hline
N\backslash N & NP/N & N \\
\lambda f.\lambda x.f(x)\wedge & \lambda f.\arg\max(f, & \lambda x.flight(x) \\
to(x, washington) & \lambda y.depart\_time(y)) & \\
\hline
\multicolumn{2}{c}{\lesssim_{\mathbf{B}}} & \\
\multicolumn{2}{c}{NP\backslash N} & \\
\multicolumn{2}{c}{\lambda f.\arg\max(\lambda x.f(x)\wedge} & \\
\multicolumn{2}{c}{to(x, washington), \lambda y.depart\_time(y))} & \\
\hline
\multicolumn{3}{c}{\gtrsim} \\
\multicolumn{3}{c}{NP} \\
\multicolumn{3}{c}{\arg\max(\lambda x.flight(x)\wedge to(x, washington),} \\
\multicolumn{3}{c}{\lambda y.depart\_time(y))}
\end{array}
$$

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 **f**; 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.

### 3.2 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 shifting* rules template:

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

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 **f** 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:

$$
\begin{array}{ccc}
\text{flights} & \text{boston} & \text{to new york} \\
\hline
N & NP & N\backslash N \\
\lambda x.flight(x) & bos & \lambda f.\lambda x.f(x) \\
 & & \wedge to(x, new\_york) \\
 & \overline{\phantom{xxxxxxxx}}\,\mathbf{T_R} & \\
 & N\backslash N & \\
 & \lambda f.\lambda x.f(x)\wedge from(x, bos) & \\
\hline
\multicolumn{2}{c}{<} & \\
\multicolumn{2}{c}{N} & \\
\multicolumn{2}{c}{\lambda f.\lambda x.flight(x)\wedge from(x, bos)} & \\
\hline
\multicolumn{3}{c}{<} \\
\multicolumn{3}{c}{N} \\
\multicolumn{3}{c}{\lambda x.flight(x)\wedge to(x, new\_york)\wedge from(x, bos)}
\end{array}
$$

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

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

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:

$$
\begin{array}{cc}
\text{American Airlines} & \text{from New York} \\
\hline
N/N & N\backslash N \\
\lambda f.\lambda x.f(x)\wedge airline(x, aa) & \lambda f.\lambda x.f(x)\wedge from(x, new\_york) \\
 & \overline{\phantom{xxxxxxxx}}\,\mathbf{T_N} \\
 & N \\
 & \lambda x.from(x, new\_york) \\
\hline
\multicolumn{2}{c}{>} \\
\multicolumn{2}{c}{N} \\
\multicolumn{2}{c}{\lambda x.airline(x, aa)\wedge from(x, new\_york)}
\end{array}
$$

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.[7]

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

### 3.3 Crossed Composition Rules

Finally, we include *crossed functional composition* rules:

$$
\begin{array}{llcll}
A/B : f & B\backslash C : g & \Rightarrow & A\backslash C : \lambda x.f(g(x)) & (>\mathbf{B}_\times) \\
B/C : g & A\backslash B : f & \Rightarrow & A/C : \lambda x.f(g(x)) & (<\mathbf{B}_\times)
\end{array}
$$

These rules are standard CCG operators but they were not used by the parser described in ZC05. When used in unrestricted contexts, they can significantly relax word order. Again, we address this

[7]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.

$$
\begin{array}{cccccc}
\text{dallas} & \text{to} & \text{washington} & \text{the latest} & \text{on} & \text{friday} \\
\hline
NP & (N\backslash N)/NP & NP & NP/N & (N\backslash N)/NP & NP \\
dallas & \lambda y.\lambda f.\lambda x.f(x) \wedge to(x,y) & washington & \lambda f.\arg\max(f, \lambda y.depart\_time(y)) & \lambda y.\lambda f.\lambda x.f(x) \wedge day(x,y) & friday
\end{array}
$$

$$
\frac{NP : dallas}{N\backslash N : \lambda f.\lambda x.f(x) \wedge from(x, dallas)}\ \mathsf{T_R}
\qquad
\frac{(N\backslash N)/NP \quad NP}{N\backslash N : \lambda f.\lambda x.f(x) \wedge to(x, washington)}\ >
\qquad
\frac{(N\backslash N)/NP \quad NP}{N\backslash N : \lambda f.\lambda x.f(x) \wedge day(x, friday)}\ >
$$

$$
\frac{N\backslash N \quad N\backslash N}{N\backslash N : \lambda f.\lambda x.f(x) \wedge from(x, dallas) \wedge to(x, washington)}\ <\mathsf{B}
\qquad
\frac{N\backslash N : \lambda f.\lambda x.f(x) \wedge day(x, friday)}{N : \lambda x.day(x, friday)}\ \mathsf{T_N}
$$

$$
\frac{N\backslash N \quad NP/N}{NP\backslash N : \lambda f.\arg\max(\lambda x.f(x) \wedge from(x, dallas) \wedge to(x, washington), \lambda y.depart\_time(y))}\ \lesssim\mathsf{B}
$$

$$
\frac{NP\backslash N \quad N}{NP : \arg\max(\lambda x.day(x, friday) \wedge from(x, dallas) \wedge to(x, washington), \lambda y.depart\_time(y))}\ \gtrsim
$$

Figure 3: A parse with the flexible parser.

problem by introducing features that count the number of times they are used in a parse.[8]

### 3.4 An Example

As a final point, to see how these rules can interact in practice, see figure 3. This example demonstrates the use of the relaxed application and composition rules, as well as the new type-raising rules.

## 4 Learning

This section describes an approach to learning in our model. We first define the features used and then describe a new online learning algorithm for the task.

### 4.1 Features in the Model

Section 2.3 described the use of a function $\mathbf{f}(x, y)$ which maps a sentence $x$ together with a CCG parse $y$ to a feature vector. As described in section 3, we introduce features for the new CCG combinators. In addition, we follow ZC05 in defining features which track the number of times each lexical item in $\Lambda$ is used. For example, we would have one feature tracking the number of times the lexical entry $flights := N : \lambda x.flights(x)$ is used in a parse, and similar features for all other members of $\Lambda$.

Finally, we introduce new features which 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 which track disjunction as opposed to conjunction.

[8]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.

### 4.2 An Online Learning Algorithm

Figure 4 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 $(\mathbf{w}, \Lambda)$) as its output. The algorithm is *online*, in that it visits each example in turn, and updates both $\mathbf{w}$ and $\Lambda$ if necessary. In Step 1 on each example, 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 $\Lambda$. 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 $\mathbf{w}$ is made if the resulting parse does not have the correct logical form.

This algorithm differs from the approach in ZC05 in a couple of important respects. First, the ZC05 algorithm performed learning of the lexicon $\Lambda$ at each iteration in a batch method, requiring a pass over the entire training set. The new algorithm is fully online, learning both $\Lambda$ and $\mathbf{w}$ 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

**Inputs:** Training examples $\{(x_i, z_i) : i = 1 \ldots n\}$ where each $x_i$ is a sentence, each $z_i$ is a logical form. An initial lexicon $\Lambda_0$. Number of training iterations, $T$.

**Definitions:** GENLEX$(x, z)$ takes as input a sentence $x$ and a logical form $z$ and returns a set of lexical items as described in section 2.4. GEN$(x; \Lambda)$ is the set of all parses for $x$ with lexicon $\Lambda$. GEN$(x, z; \Lambda)$ is the set of all parses for $x$ with lexicon $\Lambda$, which have logical form $z$. The function $\mathbf{f}(x, y)$ represents the features described in section 4.1. The function $L(y)$ maps a parse tree $y$ to its associated logical form.

**Initialization:** Set parameters $\mathbf{w}$ to initial values described in section 6.2. Set $\Lambda = \Lambda_0$.

**Algorithm:**

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

**Step 1:** (Check correctness)

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

**Step 3:** (Update parameters)

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

**Output:** Lexicon $\Lambda$ together with parameters $\mathbf{w}$.

Figure 4: An online learning algorithm.

updates, which are simpler and cheaper to compute.

As in ZC05, the algorithm assumes an initial lexicon $\Lambda_0$ that contains 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.[9]

## 5 Related Work

There has been a significant amount of previous work on learning to map sentences to underlying semantic representations. A wide variety of techniques have been considered including approaches based on machine translation techniques (Papineni et al., 1997; Ramaswamy and Kleindienst, 2000; Wong and Mooney, 2006), parsing techniques (Miller et al., 1996; Ge and Mooney, 2006), techniques that use inductive logic programming (Zelle and Mooney, 1996; Thompson and Mooney, 2002; Tang and Mooney, 2000; Kate et al., 2005), and ideas from string kernels and support vector machines (Kate and Mooney, 2006; Nguyen et al., 2006). In our experiments we compare to He and Young (2006) on the ATIS domain and Zettlemoyer and Collins (2005) on the Geo880 domain, because these systems currently achieve the best performance on these problems.

The approach of Zettlemoyer and Collins (2005) was presented in section 2.4. 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. This approach has been integrated with a speech recognizer and shown to be robust to recognition errors (He and Young, 2006).

There is also related work in the CCG literature. Clark and Curran (2003) present a method for learning the parameters of a log-linear CCG parsing model from fully annotated normal–form parse trees. Watkinson and Manandhar (1999) present an unsupervised approach for learning CCG lexicons that does not represent the semantics of the training sentences. Bos et al. (2004) present an algorithm that learns CCG lexicons with semantics but requires fully–specified CCG derivations in the training data. Bozsahin (1998) presents work on using CCG to model languages with free word order.

In addition, there is related work that focuses on modeling child language learning. Siskind (1996) presents an algorithm that learns word-to-meaning mappings from sentences that are paired with a set of possible meaning representations. Villavicencio (2001) describes an approach that learns a categorial grammar with syntactic and semantic information. Both of these approaches use sentences from child-directed speech, which differ significantly from the natural language interface queries we consider.

Finally, there is work on manually developing parsing techniques to improve robustness (Carbonell

[9]Our assumption is that these entries are likely to be domain independent, so it is simple enough to compile a list that can be reused in new domains. Another approach, which we may consider in the future, would be to annotate a small subset of the training examples with full CCG derivations, from which these frequently occurring entries could be learned.

and Hayes, 1983; Seneff, 1992). In contrast, our approach is integrated into a learning framework.

## 6 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 previous work of Zettlemoyer and Collins (2005), 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.

### 6.1 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.

### 6.2 Parameters in the Approach

The algorithm in figure 4 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 $\Lambda_0$ has an associated feature which counts the number of times this entry is seen in a parse. The initial parameter value in $\mathbf{w}$ for all features of this form was chosen to be some value $\alpha$. Each of the new CCG rules—the application, composition, crossed-composition, and type-raising rules described in section 3—has an associated parameter. We set all of these parameters to the same initial value $\beta$. Finally, when new lexical entries are added to $\Lambda$ (in step 2 of the algorithm), their initial weight is set to some value $\gamma$. In practice, optimization on development data led to a positive value for $\alpha$, and negative values for $\beta$ and $\gamma$.

### 6.3 Results

Table 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 2 shows results under the partial-match criterion. The results for our 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 3 shows results on the Geo880 domain. The

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

Table 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 2: Partial-credit accuracy on the ATIS test set.

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 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 ZC05 algorithm. 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 this improved performance is that on many training examples,[10] 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.

## 7 Discussion

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.

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

| | 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 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 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 3 that control the use of the new combinators. The third row presents results without the combinators from section 3.1 that relax word order. The fourth row reports experiments without the type-raising combinators presented in section 3.2.


### Acknowledgements

We would like to thank Yulan He and Steve Young for their help with obtaining the ATIS data set. We also acknowledge the support for this research. Luke Zettlemoyer was funded by a Microsoft graduate research fellowship and Michael Collins was supported by the National Science Foundation under grants 0347631 and DMS-0434222.


## References


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

Johan Bos, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. 2004. Wide-coverage semantic representations from a CCG parser. In *Proceedings of the 20th International Conference on Computational Linguistics*.

Cem Bozsahin. 1998. Deriving the predicate-argument structure for a free word order language. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics*.

Jaime G. Carbonell and Philip J. Hayes. 1983. Recovery strategies for parsing extragrammatical language. *American Journal of Computational Linguistics*, 9.

Bob Carpenter. 1997. *Type-Logical Semantics*. The MIT Press.

Stephen Clark and James R. Curran. 2003. Log-linear models for wide-coverage CCG parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

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

Michael Collins. 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.

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

Ruifang Ge and Raymond J. Mooney. 2006. Discriminative reranking for semantic parsing. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*.

Yulan He and Steve Young. 2005. Semantic processing using the hidden vector state model. *Computer Speech and Language*.

Yulan He and Steve Young. 2006. Spoken language understanding using the hidden vector state model. *Speech Communication Special Issue on Spoken Language Understanding for Conversational Systems*.

Mark Johnson, Stuart Geman, Steven Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the Association for Computational Linguistics*.

Rohit J. Kate and Raymond J. Mooney. 2006. Using string-kernels for learning semantic parsers. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics*.

Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of the 20th National Conference on Artificial Intelligence*.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*.

Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. 2006. An end-to-end discriminative approach to machine translation. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics*.

Scott Miller, David Stallard, Robert J. Bobrow, and Richard L. Schwartz. 1996. A fully statistical approach to natural language interfaces. In *Proceedings of the Association for Computational Linguistics*.

Raymond J. Mooney. 2007. Learning for semantic parsing. In *Computational Linguistics and Intelligent Text Processing: Proceedings of the 8th International Conference*.

Le-Minh Nguyen, Akira Shimazu, and Xuan-Hieu Phan. 2006. Semantic parsing with structured SVM ensemble classification models. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*.

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

Ganesh N. Ramaswamy and Jan Kleindienst. 2000. Hierarchical feature-based translation for scalable natural language understanding. In *Proceedings of 6th International Conference on Spoken Language Processing*.

Adwait Ratnaparkhi, Salim Roukos, and R. Todd Ward. 1994. A maximum entropy model for parsing. In *Proceedings of the International Conference on Spoken Language Processing*.

Stephanie Seneff. 1992. Robust parsing for spoken language systems. In *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing*.

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

Mark Steedman. 1996. *Surface Structure and Interpretation*. The MIT Press.

Mark Steedman. 2000. *The Syntactic Process*. The MIT Press.

Lappoon R. Tang and Raymond J. Mooney. 2000. Automated construction of database interfaces: Integrating statistical and relational learning for semantic parsing. In *Joint Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Christopher Manning. 2004. Max-margin parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Cynthia A. Thompson and Raymond J. Mooney. 2002. Acquiring word-meaning mappings for natural language interfaces. *Journal of Artificial Intelligence Research*, 18.

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

Stephen Watkinson and Suresh Manandhar. 1999. Unsupervised lexical learning with categorial grammars using the LLL corpus. In *Proceedings of the 1st Workshop on Learning Language in Logic*.

Yuk Wah Wong and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL*.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the 14th National Conference on Artificial Intelligence*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*.