Inducing Probabilistic CCG Grammars from Logical Form with Higher-Order Unification

Tom Kwiatkowski* t.m.kwiatkowksi@sms.ed.ac.uk
Luke Zettlemoyer† lsz@cs.washington.edu
Sharon Goldwater* sgwater@inf.ed.ac.uk
Mark Steedman* steedman@inf.ed.ac.uk

*School of Informatics
University of Edinburgh
Edinburgh, EH8 9AB, UK

†Computer Science & Engineering
University of Washington
Seattle, WA 98195

Abstract

This paper addresses the problem of learning to map sentences to logical form, given training data consisting of natural language sentences paired with logical representations of their meaning. Previous approaches have been designed for particular natural languages or specific meaning representations; here we present a more general method. The approach induces a probabilistic CCG grammar that represents the meaning of individual words and defines how these meanings can be combined to analyze complete sentences. We use higher-order unification to define a hypothesis space containing all grammars consistent with the training data, and develop an online learning algorithm that efficiently searches this space while simultaneously estimating the parameters of a log-linear parsing model. Experiments demonstrate high accuracy on benchmark data sets in four languages with two different meaning representations.

1 Introduction

A key aim in natural language processing is to learn a mapping from natural language sentences to formal representations of their meaning. Recent work has addressed this problem by learning semantic parsers given sentences paired with logical meaning representations (Thompson & Mooney, 2002; Kate et al., 2005; Kate & Mooney, 2006; Wong & Mooney, 2006, 2007; Zettlemoyer & Collins, 2005, 2007; Lu et al., 2008). For example, the training data might consist of English sentences paired with lambda-calculus meaning representations:

Sentence: which states border texas
Meaning: $\lambda x.state(x) \wedge next\_to(x, tex)$

Given pairs like this, the goal is to learn to map new, unseen, sentences to their corresponding meaning.

Previous approaches to this problem have been tailored to specific natural languages, specific meaning representations, or both. Here, we develop an approach that can learn to map any natural language to a wide variety of logical representations of linguistic meaning. In addition to data like the above, this approach can also learn from examples such as:

Sentence: hangi eyaletin texas ye siniri vardir
Meaning: $answer(state(borders(tex)))$

where the sentence is in Turkish and the meaning representation is a variable-free logical expression of the type that has been used in recent work (Kate et al., 2005; Kate & Mooney, 2006; Wong & Mooney, 2006; Lu et al., 2008).

The reason for generalizing to multiple languages is obvious. The need to learn over multiple representations arises from the fact that there is no standard representation for logical form for natural language. Instead, existing representations are ad hoc, tailored to the application of interest. For example, the variable-free representation above was designed for building natural language interfaces to databases.

Our approach works by inducing a combinatory categorial grammar (CCG) (Steedman, 1996, 2000). A CCG grammar consists of a language-specific lexicon, whose entries pair individual words and phrases with both syntactic and semantic information, and a universal set of combinatory rules that

project that lexicon onto the sentences and meanings of the language via syntactic derivations. The learning process starts by postulating, for each sentence in the training data, a single multi-word lexical item pairing that sentence with its complete logical form. These entries are iteratively refined with a restricted higher-order unification procedure (Huet, 1975) that defines all possible ways to subdivide them, consistent with the requirement that each training sentence can still be parsed to yield its labeled meaning.

For the data sets we consider, the space of possible grammars is too large to explicitly enumerate. The induced grammar is also typically highly ambiguous, producing a large number of possible analyses for each sentence. Our approach discriminates between analyses using a log-linear CCG parsing model, similar to those used in previous work (Clark & Curran, 2003, 2007), but differing in that the syntactic parses are treated as a hidden variable during training, following the approach of Zettlemoyer & Collins (2005, 2007). We present an algorithm that incrementally learns the parameters of this model while simultaneously exploring the space of possible grammars. The model is used to guide the process of grammar refinement during training as well as providing a metric for selecting the best analysis for each new sentence.

We evaluate the approach on benchmark datasets from a natural language interface to a database of US Geography (Zelle & Mooney, 1996). We show that accurate models can be learned for multiple languages with both the variable-free and lambda-calculus meaning representations introduced above. We also compare performance to previous methods (Kate & Mooney, 2006; Wong & Mooney, 2006, 2007; Zettlemoyer & Collins, 2005, 2007; Lu et al., 2008), which are designed with either language- or representation- specific constraints that limit generalization, as discussed in more detail in Section 6. Despite being the only approach that is general enough to run on all of the data sets, our algorithm achieves similar performance to the others, even outperforming them in several cases.

2 Overview of the Approach

The goal of our algorithm is to find a function $f : x \rightarrow z$ that maps sentences x to logical expressions z. We learn this function by inducing a probabilistic CCG (PCCG) grammar from a training set $\{(x_i, z_i)|i = 1 \ldots n\}$ containing example (sentence, logical-form) pairs such as ("New York borders Vermont", $next\_to(ny, vt)$). The induced grammar consists of two components which the algorithm must learn:

- A CCG lexicon, Λ, containing lexical items that define the space of possible parses y for an input sentence x. Each parse contains both syntactic and semantic information, and defines the output logical form z.

- A parameter vector, θ, that defines a distribution over the possible parses y, conditioned on the sentence x.

We will present the approach in two parts. The lexical induction process (Section 4) uses a restricted form of higher order unification along with the CCG combinatory rules to propose new entries for Λ. The complete learning algorithm (Section 5) integrates this lexical induction with a parameter estimation scheme that learns θ. Before presenting the details, we first review necessary background.

3 Background

This section provides an introduction to the ways in which we will use lambda calculus and higher-order unification to construct meaning representations. It also reviews the CCG grammar formalism and probabilistic extensions to it, including existing parsing and parameter estimation techniques.

3.1 Lambda Calculus and Higher-Order Unification

We assume that sentence meanings are represented as logical expressions, which we will construct from the meaning of individual words by using the operations defined in the lambda calculus. We use a version of the typed lambda calculus (cf. Carpenter (1997)), in which the basic types include e, for entities; t, for truth values; and i for numbers. There are also function types of the form $\langle e, t \rangle$ that are assigned to lambda expressions, such as $\lambda x.state(x)$, which take entities and return truth values. We represent the meaning of words and phrases using

lambda-calculus expressions that can contain constants, quantifiers, logical connectors, and lambda abstractions.

The advantage of using the lambda calculus lies in its generality. The meanings of individual words and phrases can be arbitrary lambda expressions, while the final meaning for a sentence can take different forms. It can be a full lambda-calculus expression, a variable-free expression such as $answer(state(borders(tex)))$, or any other logical expression that can be built from the primitive meanings via function application and composition.

The higher-order unification problem (Huet, 1975) involves finding a substitution for the free variables in a pair of lambda-calculus expressions that, when applied, makes the expressions equal each other. This problem is notoriously complex; in the unrestricted form (Huet, 1973), it is undecidable. In this paper, we will guide the grammar induction process using a restricted version of higher-order unification that is tractable. For a given expression h, we will need to find expressions for f and g such that either $h = f(g)$ or $h = \lambda x.f(g(x))$. This limited form of the unification problem will allow us to define the ways to split h into subparts that can be recombined with CCG parsing operations, which we will define in the next section, to reconstruct h.

3.2 Combinatory Categorial Grammar

CCG (Steedman, 2000) is a linguistic formalism that tightly couples syntax and semantics, and can be used to model a wide range of language phenomena. For present purposes a CCG grammar includes a lexicon Λ with entries like the following:

$$\text{New York} \vdash NP : ny$$
$$\text{borders} \vdash S\backslash NP/NP : \lambda x\lambda y.next\_to(y, x)$$
$$\text{Vermont} \vdash NP : vt$$

where each lexical item $w \vdash X : h$ has words w, a syntactic category X, and a logical form h expressed as a lambda-calculus expression. For the first example, these are "New York," NP, and ny. CCG syntactic categories may be atomic (such as S, NP) or complex (such as $S\backslash NP/NP$).

CCG combines categories using a set of *combinatory rules*. For example, the forward ($>$) and backward ($<$) *application* rules are:

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

These rules apply to build syntactic and semantic derivations under the control of the word order information encoded in the slash directions of the lexical entries. For example, given the lexicon above, the sentence *New York borders Vermont* can be parsed to produce:

$$\dfrac{\dfrac{\begin{array}{c}\text{New York}\\ \hline NP\\ ny\end{array} \quad \dfrac{\begin{array}{c}\text{borders}\\ \hline (S\backslash NP)/NP\\ \lambda x\lambda y.next\_to(y,x)\end{array} \quad \begin{array}{c}\text{Vermont}\\ \hline NP\\ vt\end{array}}{\begin{array}{c}(S\backslash NP)\\ \lambda y.next\_to(y, vt)\end{array}}>}{\begin{array}{c}S\\ next\_to(ny, vt)\end{array}}<$$

where each step in the parse is labeled with the combinatory rule ($->$ or $-<$) that was used.

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

$$X/Y : f \quad Y/Z : g \Rightarrow X/Z : \lambda x.f(g(x)) \qquad (> \mathbf{B})$$
$$Y\backslash Z : g \quad X\backslash Y : f \Rightarrow X\backslash Z : \lambda x.f(g(x)) \qquad (< \mathbf{B})$$

These rules provide for a relaxed notion of constituency which will be useful during learning as we reason about possible refinements of the grammar.

We also allow vertical slashes in CCG categories, which act as wild cards. For example, with this extension the forward application combinator ($>$) could be used to combine the category $S/(S|NP)$ with any of $S\backslash NP$, S/NP, or $S|NP$. Figure 1 shows two parses where the composition combinators and vertical slashes are used. These parses closely resemble the types of analyses that will be possible under the grammars we learn in the experiments described in Section 8.

3.3 Probabilistic CCGs

Given a CCG lexicon Λ, there will, in general, be many possible parses for each sentence. We select the most likely alternative using a log-linear model, which consists of a feature vector ϕ and a parameter vector θ. The joint probability of a logical form z constructed with a parse y, given a sentence x is

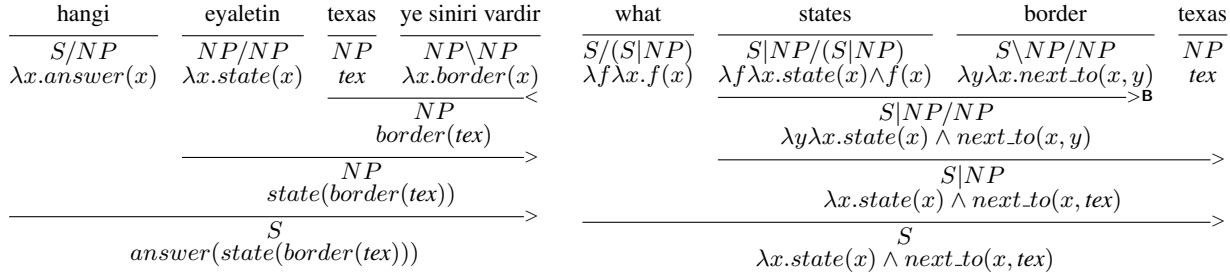

Figure 1: Two examples of CCG parses with different logical form representations.

defined as:

$$P(y, z|x; \theta, \Lambda) = \frac{e^{\theta \cdot \phi(x,y,z)}}{\sum_{(y',z')} e^{\theta \cdot \phi(x,y',z')}} \quad (1)$$

Section 7 defines the features used in the experiments, which include, for example, lexical features that indicate when specific lexical items in Λ are used in the parse y. For parsing and parameter estimation, we use standard algorithms (Clark & Curran, 2007), as described below.

The parsing, or inference, problem is to find the most likely logical form z given a sentence x, assuming the parameters θ and lexicon Λ are known:

$$f(x) = \arg\max_z p(z|x; \theta, \Lambda) \quad (2)$$

where the probability of the logical form is found by summing over all parses that produce it:

$$p(z|x; \theta, \Lambda) = \sum_y p(y, z|x; \theta, \Lambda) \quad (3)$$

In this approach the distribution over parse trees y is modeled as a hidden variable. The sum over parses in Eq. 3 can be calculated efficiently using the inside-outside algorithm with a CKY-style parsing algorithm.

To estimate the parameters themselves, we use stochastic gradient updates (LeCun et al., 1998). Given a set of n sentence-meaning pairs $\{(x_i, z_i) : i = 1...n\}$, we update the parameters θ iteratively, for each example i, by following the local gradient of the conditional log-likelihood objective $O_i = \log P(z_i|x_i; \theta, \Lambda)$. The local gradient of the individual parameter θ_j associated with feature ϕ_j and training instance (x_i, z_i) is given by:

$$\frac{\partial O_i}{\partial \theta_j} = E_{p(y|x_i,z_i;\theta,\Lambda)}[\phi_j(x_i, y, z_i)] - E_{p(y,z|x_i;\theta,\Lambda)}[\phi_j(x_i, y, z)] \quad (4)$$

As with Eq. 3, all of the expectations in Eq. 4 are calculated through the use of the inside-outside algorithm on a pruned parse chart. In the experiments, each chart cell was pruned to the top 200 entries.

4 Splitting Lexical Items

Before presenting a complete learning algorithm, we first describe how to use higher-order unification to define a procedure for splitting CCG lexical entries. This splitting process is used to expand the lexicon during learning. We seed the lexical induction with a multi-word lexical item $x_i \vdash S{:}z_i$ for each training example (x_i, z_i), consisting of the entire sentence x_i and its associated meaning representation z_i. For example, one initial lexical item might be:

$$\text{New York borders Vermont} \vdash S{:}next\_to(ny, vt) \quad (5)$$

Although these initial, sentential lexical items can parse the training data, they will not generalize well to unseen data. To learn effectively, we will need to split overly specific entries of this type into pairs of new, smaller, entries that generalize better. For example, one possible split of the lexical entry given in (5) would be the pair:

$$\text{New York borders} \vdash S/NP : \lambda x.next\_to(ny, x),$$
$$\text{Vermont} \vdash NP : vt$$

where we broke the original logical expression into two new ones $\lambda x.next\_to(ny, x)$ and vt, and paired them with syntactic categories that allow the new lexical entries to be recombined to produce the original analysis. The next three subsections define the set of possible splits for any given lexical item. The process is driven by solving a higher-order unification problem that defines all of the ways of splitting the logical expression into two parts, as described in Section 4.1. Section 4.2 describes how to construct

syntactic categories that are consistent with the two new fragments of logical form and which will allow the new lexical items to recombine. Finally, Section 4.3 defines the full set of lexical entry pairs that can be created by splitting a lexical entry.

As we will see, this splitting process is overly prolific for any single language and will yield many lexical items that do not generalize well. For example, there is nothing in our original lexical entry above that provides evidence that the split should pair "Vermont" with the constant vt and not $\lambda x.next\_to(ny, x)$. Section 5 describes how we estimate the parameters of a probabilistic parsing model and how this parsing model can be used to guide the selection of items to add to the lexicon.

4.1 Restricted Higher-Order Unification

The set of possible splits for a logical expression h is defined as the solution to a pair of higher-order unification problems. We find pairs of logical expressions (f, g) such that either $f(g) = h$ or $\lambda x.f(g(x)) = h$. Solving these problems creates new expressions f and g that can be recombined according to the CCG combinators, as defined in Section 3.2, to produce h.

In the unrestricted case, there can be infinitely many solution pairs (f, g) for a given expression h. For example, when $h = tex$ and $f = \lambda x.tex$, the expression g can be anything. Although it would be simple enough to forbid vacuous variables in f and g, the number of solutions would still be exponential in the size of h. For example, when h contains a conjunction, such as $h = \lambda x.city(x) \wedge major(x) \wedge in(x, tex)$, any subset of the expressions in the conjunction can be assigned to f (or g).

To limit the number of possible splits, we enforce the following restrictions on the possible higher-order solutions that will be used during learning:

- **No Vacuous Variables:** Neither g or f can be a function of the form $\lambda x.e$ where the expression e does not contain the variable x. This rules out functions such as $\lambda x.tex$.
- **Limited Coordination Extraction:** The expression g cannot contain more than N of the conjuncts that appear in any coordination in h. For example, with $N = 1$ the expression $g = \lambda x.city(x) \wedge major(x)$ could not be used as a solution given the h conjuction above. We use $N = 4$ in our experimental evaluation.
- **Limited Application:** The function f cannot contain new variables applied to any non-variable subexpressions from h. For example, if $h = \lambda x.in(x, tex)$, the pair $f = \lambda q.q(tex)$ and $g = \lambda y \lambda x.in(x, y)$ is forbidden.

Together, these three restrictions guarantee that the number of splits is, in the worst case, an N-degree polynomial of the number of constants in h. The constraints were designed to increase the efficiency of the splitting algorithm without impacting performance on the development data.

4.2 Splitting Categories

We define the set of possible splits for a category $X{:}h$ with syntax X and logical form h by enumerating the solution pairs (f, g) to the higher-order unification problems defined above and creating syntactic categories for the resulting expressions. For example, given $X{:}h = S\backslash NP{:}\lambda x.in(x, tex)$, $f = \lambda y \lambda x.in(x, y)$, and $g = tex$, we would produce the following two pairs of new categories:

$$(\ S\backslash NP/NP{:}\lambda y \lambda x.in(x, y)\ ,\ NP{:}tex\ )$$
$$(\ NP{:}tex\ ,\ S\backslash NP\backslash NP{:}\lambda y \lambda x.in(x, y)\ )$$

which were constructed by first choosing the syntactic category for g, in this case NP, and then enumerating the possible directions for the new slash in the category containing f. We consider each of these two steps in more detail below.

The new syntactic category for g is determined based on its type, $T(g)$. For example, $T(tex) = e$ and $T(\lambda x.state(x)) = \langle e, t \rangle$. Then, the function $C(T)$ takes an input type T and returns the syntactic category of T as follows:

$$C(T) = \begin{cases} NP & \text{if } T = e \\ S & \text{if } T = t \\ C(T_2)|C(T_1) & \text{when } T = \langle T_1, T_2 \rangle \end{cases}$$

The basic types e and t are assigned syntactic categories NP and S, and all functional types are assigned categories recursively. For example $C(\langle e, t \rangle) = S|NP$ and $C(\langle e, \langle e, t \rangle \rangle) = S|NP|NP$. This definition of CCG categories is unconventional in that it never assigns atomic categories to functional types. For example, there is no

distinct syntactic category N for nouns (which have semantic type $\langle e, t \rangle$). Instead, the more complex category $S|NP$ is used.

Now, we are ready to define the set of all category splits. For a category $A = X{:}h$ we can define

$$\mathrm{S}_C(A) = \{\mathrm{FA}(A) \cup \mathrm{BA}(A) \cup \mathrm{FC}(A) \cup \mathrm{BC}(A)\}$$

which is a union of sets, each of which includes splits for a single CCG operator. For example, $\mathrm{FA}(X{:}h)$ is the set of category pairs

$$\mathrm{FA}(X{:}h) = \{(X/Y{:}f, Y{:}g) \mid h{=}f(g) \wedge Y{=}C(T(g))\}$$

where each pair can be combined with the forward application combinator, described in Section 3.2, to reconstruct $X{:}h$.

The remaining three sets are defined similarly, and are associated with the backward application and forward and backward composition operators, respectively:

$$\mathrm{BA}(X{:}h) = \{(Y{:}g, X\backslash Y{:}f) \mid h{=}f(g) \wedge Y{=}C(T(g))\}$$

$$\mathrm{FC}(X/Y{:}h) = \{(X/W{:}f, W/Y{:}g) \mid h{=}\lambda x.f(g(x)) \wedge W{=}C(T(g(x)))\}$$

$$\mathrm{BC}(X\backslash Y{:}h) = \{(W\backslash Y{:}g, X\backslash W{:}f) \mid h{=}\lambda x.f(g(x)) \wedge W{=}C(T(g(x)))\}$$

where the composition sets FC and BC only accept input categories with the appropriate outermost slash direction, for example $\mathrm{FC}(X/Y{:}h)$.

4.3 Splitting Lexical Items

We can now define the lexical splits that will be used during learning. For lexical entry $w_{0:n} \vdash A$, with word sequence $w_{0:n} = \langle w_0, \ldots, w_n \rangle$ and CCG category A, define the set S_L of splits to be:

$$\mathrm{S}_L(w_{0:n}{\vdash}A) = \{(w_{0:i}{\vdash}B, w_{i+1:n}{\vdash}C) \mid 0 \leq i < n \wedge (B, C) \in \mathrm{S}_C(A)\}$$

where we enumerate all ways of splitting the words sequence $w_{0:n}$ and aligning the subsequences with categories in $\mathrm{S}_C(A)$, as defined in the last section.

5 Learning Algorithm

The previous section described how a splitting procedure can be used to break apart overly specific lexical items into smaller ones that may generalize better to unseen data. The space of possible lexical items supported by this splitting procedure is too large to explicitly enumerate. Instead, we learn the parameters of a PCCG, which is used both to guide the splitting process, and also to select the best parse, given a learned lexicon.

Figure 2 presents the unification-based learning algorithm, UBL. This algorithm steps through the data incrementally and performs two steps for each training example. First, new lexical items are induced for the training instance by splitting and merging nodes in the best correct parse, given the current parameters. Next, the parameters of the PCCG are updated by making a stochastic gradient update on the marginal likelihood, given the updated lexicon.

Inputs and Initialization The algorithm takes as input the training set of n (sentence, logical form) pairs $\{(x_i, z_i) : i = 1...n\}$ along with an NP list, Λ_{NP}, of proper noun lexical items such as $\mathrm{Texas} \vdash NP{:}tex$. The lexicon, Λ, is initialized with a single lexical item $x_i \vdash S{:}z_i$ for each of the training pairs along with the contents of the NP list. It is possible to run the algorithm without the initial NP list; we include it to allow direct comparisons with previous approaches, which also included NP lists. Features and initial feature weights are described in Section 7.

Step 1: Updating the Lexicon In the lexical update step the algorithm first computes the best correct parse tree y^* for the current training example and then uses y^* as input to the procedure NEW-LEX, which determines which (if any) new lexical items to add to Λ. NEW-LEX begins by enumerating all pairs $(C, w_{i:j})$, for $i < j$, where C is a category occurring at a node in y^* and $w_{i:j}$ are the (two or more) words it spans. For example, in the left parse in Figure 1, there would be four pairs: one with the category $C = NP\backslash NP{:}\lambda x.border(x)$ and the phrase $w_{i:j}$ ="ye siniri vardir", and one for each non-leaf node in the tree.

For each pair $(C, w_{i:j})$, NEW-LEX considers introducing a new lexical item $w_{i:j}{\vdash}C$, which allows for the possibility of a parse where the subtree rooted at C is replaced with this new entry. (If C is a leaf node, this item will already exist.) NEW-LEX also considers adding each pair of new lexical items that is obtained by splitting $w_{i:j}{\vdash}C$ as described in Section 4, thereby considering many different ways of reanalyzing the node. This process creates a set of possible new lexicons, where each lexicon expands

Λ in a different way by adding the items from either a single split or a single merge of a node in y^*.

For each potential new lexicon Λ', NEW-LEX computes the probability $p(y^*|x_i, z_i; \theta', \Lambda')$ of the original parse y^* under Λ' and parameters θ' that are the same as θ but have weights for the new lexical items, as described in Section 7. It also finds the best new parse $y' = \arg\max_y p(y|x_i, z_i; \theta', \Lambda')$.[1] Finally, NEW-LEX selects the Λ' with the largest difference in log probability between y' and y^*, and returns the new entries in Λ'. If y^* is the best parse for every Λ', NEW-LEX returns the empty set; the lexicon will not change.

Step 2: Parameter Updates For each training example we update the parameters θ using the stochastic gradient updates given by Eq. 4.

Discussion The alternation between refining the lexicon and updating the parameters drives the learning process. The initial model assigns a conditional likelihood of one to each training example (there is a single lexical item for each sentence x_i, and it contains the labeled logical form z_i). Although the splitting step often decreases the probability of the data, the new entries it produces are less specific and should generalize better. Since we initially assign positive weights to the parameters for new lexical items, the overall approach prefers splitting; trees with many lexical items will initially be much more likely. However, if the learned lexical items are used in too many incorrect parses, the stochastic gradient updates will down weight them to the point where the lexical induction step can merge or re-split nodes in the trees that contain them. This allows the approach to correct the lexicon and, hopefully, improve future performance.

6 Related Work

Previous work has focused on a variety of different meaning representations. Several approaches have been designed for the variable-free logical representations shown in examples throughout this paper. For example, Kate & Mooney (2006) present a method (KRISP) that extends an existing SVM learning algorithm to recover logical representations. The WASP system (Wong & Mooney, 2006) uses statistical machine translation techniques to learn synchronous context free grammars containing both words and logic. Lu et al. (2008) (Lu08) developed a generative model that builds a single hybrid tree of words, syntax and meaning representation. These algorithms are all language independent but representation specific.

Other algorithms have been designed to recover lambda-calculus representations. For example, Wong & Mooney (2007) developed a variant of WASP (λ-WASP) specifically designed for this alternate representation. Zettlemoyer & Collins (2005, 2007) developed CCG grammar induction techniques where lexical items are proposed according to a set of hand-engineered lexical templates.

[1] This computation can be performed efficiently by incrementally updating the parse chart used to find y^*.

Inputs: Training set $\{(x_i, z_i) : i = 1 \ldots n\}$ where each example is a sentence x_i paired with a logical form z_i. Set of NP lexical items Λ_{NP}. Number of iterations T. Learning rate parameter α_0 and cooling rate parameter c.

Definitions: The function NEW-LEX(y) takes a parse y and returns a set of new lexical items found by splitting and merging categories in y, as described in Section 5. The distributions $p(y|x, z; \theta, \Lambda)$ and $p(y, z|x; \theta, \Lambda)$ are defined by the log-linear model, as described in Section 3.3.

Initialization:

- Set $\Lambda = \{x_i \vdash S : z_i\}$ for all $i = 1 \ldots n$.
- Set $\Lambda = \Lambda \cup \Lambda_{NP}$
- Initialize θ using coocurrence statistics, as described in Section 7.

Algorithm:

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

Step 1: (Update Lexicon)

- Let $y^* = \arg\max_y p(y|x_i, z_i; \theta, \Lambda)$
- Set $\Lambda = \Lambda \cup$ NEW-LEX(y^*) and expand the parameter vector θ to contain entries for the new lexical items, as described in Section 7.

Step 2: (Update Parameters)

- Let $\gamma = \frac{\alpha_0}{1+c\times k}$ where $k = i + t \times n$.
- Let $\Delta = E_{p(y|x_i,z_i;\theta,\Lambda)}[\phi(x_i, y, z_i)] - E_{p(y,z|x_i;\theta,\Lambda)}[\phi(x_i, y, z)]$
- Set $\theta = \theta + \gamma\Delta$

Output: Lexicon Λ and parameters θ.

Figure 2: The UBL learning algorithm.

Our approach eliminates this need for manual effort.

Another line of work has focused on recovering meaning representations that are not based on logic. Examples include an early statistical method for learning to fill slot-value representations (Miller et al., 1996) and a more recent approach for recovering semantic parse trees (Ge & Mooney, 2006). Exploring the extent to which these representations are compatible with the logic-based learning approach we developed is an important area for future work.

Finally, there is work on using categorial grammars to solve other, related learning problems. For example, Buszkowski & Penn (1990) describe a unification-based approach for grammar discovery from bracketed natural language sentences and Villavicencio (2002) developed an approach for modeling child language acquisition. Additionally, Bos et al. (2004) consider the challenging problem of constructing broad-coverage semantic representations with CCG, but do not learn the lexicon.

7 Experimental Setup

Features We use two types of features in our model. First, we include a set of *lexical features*: For each lexical item $L \in \Lambda$, we include a feature ϕ_L that fires when L is used. Second, we include *semantic features* that are functions of the output logical expression z. Each time a predicate p in z takes an argument a with type $T(a)$ in position i it triggers two binary indicator features: $\phi_{(p,a,i)}$ for the predicate-argument relation; and $\phi_{(p,T(a),i)}$ for the predicate argument-type relation.

Initialization The weights for the semantic features are initialized to zero. The weights for the lexical features are initialized according to coocurrance statistics estimated with the Giza++ (Och & Ney, 2003) implementation of IBM Model 1. We compute translation scores for (word, constant) pairs that cooccur in examples in the training data. The initial weight for each ϕ_L is set to ten times the average score over the (word, constant) pairs in L, except for the weights of seed lexical entries in Λ_{NP} which are set to 10 (equivalent to the highest possible coocurrence score). We used the learning rate $\alpha_0 = 1.0$ and cooling rate $c = 10^{-5}$ in all training scenarios, and ran the algorithm for $T = 20$ iterations. These values were selected with cross validation on the Geo880 development set, described below.

Data and Evaluation We evaluate our system on the GeoQuery datasets, which contain natural-language queries of a geographical database paired with logical representations of each query's meaning. The full Geo880 dataset contains 880 (English-sentence, logical-form) pairs, which we split into a development set of 600 pairs and a test set of 280 pairs, following Zettlemoyer & Collins (2005). The Geo250 dataset is a subset of Geo880 containing 250 sentences that have been translated into Turkish, Spanish and Japanese as well as the original English. Due to the small size of this dataset we use 10-fold cross validation for evaluation. We use the same folds as Wong & Mooney (2006, 2007) and Lu et al. (2008), allowing a direct comparison.

The GeoQuery data is annotated with both lambda-calculus and variable-free meaning representations, which we have seen examples of throughout the paper. We report results for both representations, using the standard measures of *Recall* (percentage of test sentences assigned correct logical forms), *Precision* (percentage of logical forms returned that are correct) and *F1* (the harmonic mean of *Precision* and *Recall*).

Two-Pass Parsing To investigate the trade-off between precision and recall, we report results with a two-pass parsing strategy. When the parser fails to return an analysis for a test sentence due to novel words or usage, we reparse the sentence and allow the parser to skip words, with a fixed cost. Skipping words can potentially increase recall, if the ignored word is an unknown function word that does not contribute semantic content.

8 Results and Discussion

Tables 1, 2, and 3 present the results for all of the experiments. In aggregate, they demonstrate that our algorithm, UBL, learns accurate models across languages and for both meaning representations. This is a new result; no previous system is as general.

We also see the expected tradeoff between precision and recall that comes from the two-pass parsing approach, which is labeled UBL-s. With the ability to skip words, UBL-s achieves the highest recall of all reported systems for all evaluation conditions.

System	English			Spanish		
	Rec.	Pre.	F1	Rec.	Pre.	F1
WASP	70.0	**95.4**	80.8	72.4	91.2	81.0
Lu08	72.8	91.5	81.1	79.2	**95.2**	**86.5**
UBL	78.1	88.2	**82.7**	76.8	86.8	81.4
UBL-s	**80.4**	80.8	80.6	**79.7**	80.6	80.1
System	Japanese			Turkish		
	Rec.	Pre.	F1	Rec.	Pre.	F1
WASP	74.4	**92.0**	**82.9**	62.4	**97.0**	75.9
Lu08	76.0	87.6	81.4	66.8	93.8	78.0
UBL	78.5	85.5	81.8	70.4	89.4	**78.6**
UBL-s	**80.5**	80.6	80.6	**74.2**	75.6	74.9

Table 1: Performance across languages on Geo250 with variable-free meaning representations.

System	English			Spanish		
	Rec.	Pre.	F1	Rec.	Pre.	F1
λ-WASP	75.6	91.8	82.9	80.0	92.5	**85.8**
UBL	78.0	**93.2**	**84.7**	75.9	**93.4**	83.6
UBL-s	**81.8**	83.5	82.6	**81.4**	83.4	82.4
System	Japanese			Turkish		
	Rec.	Pre.	F1	Rec.	Pre.	F1
λ-WASP	81.2	90.1	**85.8**	68.8	90.4	**78.1**
UBL	78.9	**90.9**	84.4	67.4	**93.4**	**78.1**
UBL-s	**83.0**	83.2	83.1	**71.8**	77.8	74.6

Table 2: Performance across languages on Geo250 with lambda-calculus meaning representations.

However, UBL achieves much higher precision and better overall F1 scores, which are generally comparable to the best performing systems.

The comparison to the CCG induction techniques of ZC05 and ZC07 (Table 3) is particularly striking. These approaches used language-specific templates to propose new lexical items and also required as input a set of hand-engineered lexical entries to model phenomena such as quantification and determiners. However, the use of higher-order unification allows UBL to achieve comparable performance while automatically inducing these types of entries.

For a more qualitative evaluation, Table 4 shows a selection of lexical items learned with high weights for the lambda-calculus meaning representations. Nouns such as "state" or "estado" are consistently learned across languages with the category $S|NP$, which stands in for the more conventional N. The algorithm also learns language-specific constructions such as the Japanese case markers "no" and "wa", which are treated as modifiers that do not add semantic content. Language-specific word order is also encoded, using the slash directions of the CCG categories. For example, "what" and "que" take their arguments to the right in the wh-initial English and Spanish. However, the Turkish wh-word "nelerdir" and the Japanese question marker "nan desu ka" are sentence final, and therefore take their arguments to the left. Learning regularities of this type allows UBL to generalize well to unseen data.

System	Variable Free			Lambda Calculus		
	Rec.	Pre.	F1	Rec.	Pre.	F1
Cross Validation Results						
KRISP	71.7	**93.3**	81.1	–	–	–
WASP	74.8	87.2	80.5	–	–	–
Lu08	81.5	89.3	**85.2**	–	–	–
λ-WASP	–	–	–	86.6	92.0	89.2
Independent Test Set						
ZC05	–	–	–	79.3	**96.3**	87.0
ZC07	–	–	–	86.1	91.6	88.8
UBL	81.4	89.4	**85.2**	85.0	94.1	**89.3**
UBL-s	**84.3**	85.2	84.7	**87.9**	88.5	88.2

Table 3: Performance on the Geo880 data set, with varied meaning representations.

There is less variation and complexity in the learned lexical items for the variable-free representation. The fact that the meaning representation is deeply nested influences the form of the induced grammar. For example, recall that the sentence "what states border texas" would be paired with the meaning $answer(state(borders(tex)))$. For this representation, lexical items such as:

$$\text{what} \vdash S/NP : \lambda x.answer(x)$$
$$\text{states} \vdash NP/NP : \lambda x.state(x)$$
$$\text{border} \vdash NP/NP : \lambda x.borders(x)$$
$$\text{texas} \vdash NP : tex$$

can be used to construct the desired output. In practice, UBL often learns entries with only a single slash, like those above, varying only in the direction, as required for the language. Even the more complex items, such as those for quantifiers, are consistently simpler than those induced from the lambda-calculus meaning representations. For example, one of the most complex entries learned in the experiments for English is $\text{the smallest} \vdash NP\backslash NP/(NP|NP)$:$\lambda f \lambda x.smallest\_one(f(x))$.

There are also differences in the aggregate statistics of the learned lexicons. For example, the average length of a learned lexical item for the (lambda-

calculus, variable-free) meaning representations is: (1.21,1.08) for Turkish, (1.34,1.19) for English, (1.43,1.25) for Spanish and (1.63,1.42) for Japanese. For both meaning representations the model learns significantly more multiword lexical items for the somewhat analytic Japanese than the agglutinative Turkish. There are also variations in the average number of learned lexical items in the best parses during the final pass of training: 192 for Japanese, 206 for Spanish, 188 for English and 295 for Turkish. As compared to the other languages, the morpologically rich Turkish requires significantly more lexical variation to explain the data.

Finally, there are a number of cases where the UBL algorithm could be improved in future work. In cases where there are multiple allowable word orders, the UBL algorithm must learn individual entries for each possibility. For example, the following two categories are often learned with high weight for the Japanese word "chiisai":

$NP/(S|NP)\backslash(NP|NP)$:$\lambda f\lambda g.argmin(x, g(x), f(x))$
$NP|(S|NP)/(NP|NP)$:$\lambda f\lambda g.argmin(x, g(x), f(x))$

and are treated as distinct entries in the lexicon. Similarly, the approach presented here does not model morphology, and must repeatedly learn the correct categories for the Turkish words "nehri," "nehir," "nehirler," and "nehirlerin", all of which correspond to the logical form $\lambda x.river(x)$.

English
population of $\vdash NP/NP : \lambda x.population(x)$
smallest $\vdash NP/(S\|NP) : \lambda f.arg\ min(y, f(y), size(y))$
what $\vdash S\|NP/(S\|NP) : \lambda f\lambda x.f(x)$
border $\vdash S\|NP/NP : \lambda x\lambda y.next\_to(y, x)$
state $\vdash S\|NP : \lambda x.state(x)$
most $\vdash NP/(S\|NP)\backslash(S\|NP)\backslash(S\|NP\|NP) :$ $\lambda f\lambda g\lambda h\lambda x.argmax(y, g(y), count(z, f(z, y) \wedge h(z)))$
Japanese
no $\vdash NP\|NP/(NP\|NP) : \lambda f\lambda x.f(x)$
shuu $\vdash S\|NP : \lambda x.state(x)$
nan desu ka $\vdash S\backslash NP\backslash(NP\|NP) : \lambda f\lambda x.f(x)$
wa $\vdash NP\|NP\backslash(NP\|NP) : \lambda f\lambda x.f(x)$
ikutsu $\vdash NP\|(S\|NP)\backslash(S\|NP\|(S\|NP)) :$ $\lambda f\lambda g.count(x, f(g(x)))$
chiiki $\vdash NP\backslash NP$:$\lambda x.area(x)$
Turkish
nedir $\vdash S\backslash NP\backslash(NP\|NP) : \lambda f\lambda x.f(x)$
sehir $\vdash S\|NP : \lambda x.city(x)$
nufus yogunlugu $\vdash NP\|NP : \lambda x.density(x)$
siniri$\vdash S\|NP/NP : \lambda x\lambda y.next_to(y, x)$
kac tane $\vdash S\backslash NP/(S\|NP\|NP)\backslash(S\|NP) :$ $\lambda f\lambda g\lambda x.count(y, f(y) \wedge g(y, x))$
ya siniri $\vdash S\|NP\backslash NP : \lambda x\lambda y.next\_to(y, x)$
Spanish
en $\vdash S\|NP/NP$: $\lambda x\lambda y.loc(y, x)$
que es la $\vdash S/NP/(NP\|NP)$: $\lambda f\lambda x.f(x)$
pequena $\vdash NP\backslash(S\|NP)\backslash(NP\|NP) :$ $\lambda g\lambda f.arg\ min(y, f(y), g(y))$
estado $\vdash S\|NP : \lambda x.state(x)$
mas $\vdash S\backslash(S\|NP)/(S\|NP)\backslash(NP\|NP\|(S\|NP)) :$ $\lambda f\lambda g\lambda h.argmax(x, h(x), f(g, x))$
mayores $\vdash S\|NP\backslash(S\|NP) :\lambda f\lambda x.f(x) \wedge major(x)$

Table 4: Example learned lexical items for each language on the Geo250 lambda-calculus data sets.

9 Conclusions and Future Work

This paper has presented a method for inducing probabilistic CCGs from sentences paired with logical forms. The approach uses higher-order unification to define the space of possible grammars in a language- and representation-independent manner, paired with an algorithm that learns a probabilistic parsing model. We evaluated the approach on four languages with two meaning representations each, achieving high accuracy across all scenarios.

For future work, we are interested in exploring the generality of the approach while extending it to new understanding problems. One potential limitation is in the constraints we introduced to ensure the tractability of the higher-order unification procedure. These restrictions will not allow the approach to induce lexical items that would be used with, among other things, many of the type-raised combinators commonly employed in CCG grammars. We are also interested in developing similar grammar induction techniques for context-dependent understanding problems, such as the one considered by Zettlemoyer & Collins (2009). Such an approach would complement ideas for using high-order unification to model a wider range of language phenomena, such as VP ellipsis (Dalrymple et al., 1991).

Acknowledgements

We thank the reviewers for useful feedback. This work was supported by the EU under IST Cognitive Systems grant IP FP6-2004-IST-4-27657 "Paco-Plus" and ERC Advanced Fellowship 249520 "GRAMPLUS". Kwiatkowski was supported by an EPRSC studentship. Zettlemoyer was supported by a US NSF International Research Fellowship.

References

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*.

Buszkowski, W. & Penn, G. (1990). Categorial grammars determined from linguistic data by unification. *Studia Logica*, *49*, 431–454.

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

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.

Dalrymple, M., Shieber, S., & Pereira, F. (1991). Ellipsis and higher-order unification. *Linguistics and Philosophy*, *14*, 399–452.

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

Huet, G. (1975). A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, *1*, 27–57.

Huet, G. P. (1973). The undecidability of unification in third order logic. *Information and Control*, *22*(3), 257–267.

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., Wong, Y. W., & Mooney, R. J. (2005). Learning to transform natural to formal languages. In *Proceedings of the National Conference on Artificial Intelligence*.

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

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*.

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*.

Och, F. J. & Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, *29*(1), 19–51.

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

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

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

Villavicencio, A. (2002). The acquisition of a unification-based generalised categorial grammar. Ph.D. thesis, University of Cambridge.

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. (2007). Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the Association for Computational Linguistics*.

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*.