Lexical Generalization in CCG Grammar Induction for Semantic Parsing

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

We consider the problem of learning factored probabilistic CCG grammars for semantic parsing from data containing sentences paired with logical-form meaning representations. Traditional CCG lexicons list lexical items that pair words and phrases with syntactic and semantic content. Such lexicons can be inefficient when words appear repeatedly with closely related lexical content. In this paper, we introduce factored lexicons, which include both *lexemes* to model word meaning and *templates* to model systematic variation in word usage. We also present an algorithm for learning factored CCG lexicons, along with a probabilistic parse-selection model. Evaluations on benchmark datasets demonstrate that the approach learns highly accurate parsers, whose generalization performance benefits greatly from the lexical factoring.

1 Introduction

Semantic parsers automatically recover representations of meaning from natural language sentences. Recent work has focused on learning such parsers directly from corpora made up of sentences paired with logical meaning representations (Kate et al., 2005; Kate and Mooney, 2006; Wong and Mooney, 2006, 2007; Zettlemoyer and Collins, 2005, 2007; Lu et al., 2008; Kwiatkowski et al., 2010).

For example, in a flight booking domain we might have access to training examples such as:

Sentence: I want flights from Boston
Meaning: $\lambda x.flight(x) \wedge from(x, bos)$

and the goal is to learn a grammar that can map new, unseen, sentences onto their corresponding meanings, or logical forms.

One approach to this problem has developed algorithms for leaning probabilistic CCG grammars (Zettlemoyer and Collins, 2005, 2007; Kwiatkowski et al., 2010). These grammars are well-suited to the task of semantic parsing, as they closely link syntax and semantics. They can be used to model a wide range of complex linguistic phenomena and are strongly lexicalized, storing all language-specific grammatical information directly with the words in the lexicon. For example, a typical learned lexicon might include entries such as:

(1) $flight \vdash N : \lambda x.flight(x)$
(2) $flight \vdash N/(S|NP) : \lambda f \lambda x.flight(x) \wedge f(x)$
(3) $flight \vdash N\backslash N : \lambda f \lambda x.flight(x) \wedge f(x)$
(4) $fare \vdash N : \lambda x.cost(x)$
(5) $fare \vdash N/(S|NP) : \lambda f \lambda x.cost(x) \wedge f(x)$
(6) $fare \vdash N\backslash N : \lambda f \lambda x.cost(x) \wedge f(x)$
(7) $Boston \vdash NP : bos$
(8) $Boston \vdash N\backslash N : \lambda f \lambda x.from(x, bos) \wedge f(x)$
(9) $New\ York \vdash NP : nyc$
(10) $New\ York \vdash N\backslash N : \lambda f \lambda x.from(x, nyc) \wedge f(x)$

Although lexicalization of this kind is useful for learning, as we will see, these grammars can also suffer from sparsity in the training data, since closely related entries must be repeatedly learned for all members of a certain class of words. For example, the list above shows a selection of lexical items that would have to be learned separately.

In this list, the word "flight" is paired with the predicate *flight* in three separate lexical items which are required for different syntactic contexts. Item

Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, pages 1512–1523,
Edinburgh, Scotland, UK, July 27–31, 2011.

(1) has the standard *N* category for entries of this type, item (2) allows the use of the word "flight" with that-less relative clauses such as "flight departing Boston", and item (3) is useful for phrases with unconventional word order such as "from Boston flight to New York". Representing these three lexical items separately is inefficient, since each word of this class (such as "fare") will require three similarly structured lexical entries differing only in predicate name. There may also be systemtatic semantic variation between entries for a certain class of words. For example, in (6) "Boston" is paired with the constant *bos* that represents its meaning. However, item (7) also adds the predicate *from* to the logical form. This might be used to analyse somewhat elliptical, unedited sentences such as "Show me flights Boston to New York," which can be challenging for semantic parsers (Zettlemoyer and Collins, 2007).

This paper builds upon the insight that a large proportion of the variation between lexical items for a given class of words is systematic. Therefore it should be represented once and applied to a small set of basic lexical units. [1] We develop a *factored lexicon* that captures this insight by distinguishing *lexemes*, which pair words with logical constants, from *lexical templates*, which map lexemes to full lexical items. As we will see, this can lead to a significantly more compact lexicon that can be learned from less data. Each word or phrase will be associated with a few lexemes that can be combined with a shared set of general templates.

We develop an approach to learning factored, probabilistic CCG grammars for semantic parsing. Following previous work (Kwiatkowski et al., 2010), we make use of a higher-order unification learning scheme that defines a space of CCG grammars consistent with the (sentence, logical form) training pairs. However, instead of constructing fully specified lexical items for the learned grammar, we automatically generate sets of lexemes and lexical templates to model each example. This is a difficult learning problem, since the CCG analyses that are required to construct the final meaning representations are not explicitly labeled in the training data. Instead, we model them with hidden variables and develop an online learning approach that simultaneously estimates the parameters of a log-linear parsing model, while inducing the factored lexicon.

We evaluate the approach on the benchmark Atis and GeoQuery domains. This is a challenging setup, since the GeoQuery data has complex meaning representations and sentences in multiple languages, while the Atis data contains spontaneous, unedited text that can be difficult to analyze with a formal grammar representation. Our approach achieves at or near state-of-the-art recall across all conditions, despite having no English or domain-specific information built in. We believe that ours is the only system of sufficient generality to run with this degree of success on all of these datasets.

2 Related work

There has been significant previous work on learning semantic parsers from training sentences labelled with logical form meaning representations.

We extend a line of research that has addressed this problem by developing CCG grammar induction techniques. Zettlemoyer and Collins (2005, 2007) presented approaches that use hand generated, English-language specific rules to generate lexical items from logical forms as well as English specific type-shifting rules and relaxations of the CCG combinators to model spontaneous, unedited sentences. Zettlemoyer and Collins (2009) extends this work to the case of learning in context dependent environments. Kwiatkowski et al. (2010) described an approach for language-independent learning that replaces the hand-specified templates with a higher-order-unification-based lexical induction method, but their approach does not scale well to challenging, unedited sentences. The learning approach we develop for inducing factored lexicons is also language independent, but scales well to these challenging sentences.

There have been a number of other approaches for learning semantic parsers, including ones based on machine translation techniques (Papineni et al., 1997; Ramaswamy and Kleindienst, 2000; Wong and Mooney, 2006), parsing models (Miller et al., 1996; Ge and Mooney, 2006; Lu et al., 2008), in-

[1] A related tactic is commonly used in wide-coverage CCG parsers derived from treebanks, such as work by Hockenmaier and Steedman (2002) and Clark and Curran (2007). These parsers make extensive use of category-changing unary rules, to avoid data sparsity for systematically related categories (such as those related by type-raising). We will automatically learn to represent these types of generalizations in the factored lexicon.

ductive logic programming algorithms (Zelle and Mooney, 1996; Thompson and Mooney, 2002; Tang and Mooney, 2000), probabilistic automata (He and Young, 2005, 2006), and ideas from string kernels and support vector machines (Kate and Mooney, 2006; Nguyen et al., 2006).

More recent work has focused on training semantic parsers without supervision in the form of logical-form annotations. Clarke et al. (2010) and Liang et al. (2011) replace semantic annotations in the training set with target answers which are more easily available. Goldwasser et al. (2011) present work on unsupervised learning of logical form structure. However, all of these systems require significantly more domain and language specific initialization than the approach presented here.

Other work has learnt semantic analyses from text in the context of interactions in computational environments (Branavan et al. (2010), Vogel and Jurafsky (2010)); text grounded in partial observations of a world state (Liang et al., 2009); and from raw text alone (Poon and Domingos, 2009, 2010).

There is also related work that uses the CCG grammar formalism. 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) describe an unsupervised approach for learning syntactic CCG lexicons. Bos et al. (2004) present an algorithm for building semantic representations from CCG parses but requires fully–specified CCG derivations in the training data.

3 Overview of the Approach

Here we give a formal definition of the problem and an overview of the learning approach.

Problem We will learn a semantic parser that takes a sentences x and returns a logical form z representing its underlying meaning. We assume we have input data $\{(x_i, z_i) | i = 1 \ldots n\}$ containing sentences x_i and logical forms z_i, for example $x_i =$"Show me flights to Boston" and $z_i = \lambda x.flight(x) \wedge to(x, bos)$.

Model We will represent the parser as a factored, probabilistic CCG (PCCG) grammar. A traditional CCG lexical item would fully specify the syntax and semantics for a word (reviewed in Section 4). For example, Boston $\vdash NP : bos$ represents the entry for the word "Boston" with syntactic category NP and meaning represented by the constant bos. Where a lexicon would usually list lexical items such as this, we instead use a factored lexicon (L, T) containing:

- A list of lexemes L. Each lexeme pairs a word or phrase with a list of logical constants that can be used to construct its meaning. For example, one lexeme might be $(\text{Boston}, [bos])$.
- A list of lexical templates T. Each template takes a lexeme and maps it on to a full lexical item. For example, there is a single template that can map the lexeme above to the final lexical entry Boston $\vdash NP : bos$.

We will make central use of this factored representation to provide a more compact representation of the lexicon that can be learned efficiently.

The factored PCCG will also contain a parameter vector, θ, that defines a log-linear distribution over the possible parses y, conditioned on the sentence x.

Learning Our approach for learning factored PCCGs extends the work of Kwiatkowski et al. (2010), as reviewed in Section 7. Specifically, we modify the lexical learning, to produce lexemes and templates, as well as the feature space of the model, but reuse the existing parameter estimation techniques and overall learning cycle, as described in Section 7.

We present the complete approach in three parts by describing the factored representation of the lexicon (Section 5), techniques for proposing potential new lexemes and templates (Section 6), and finally a complete learning algorithm (Section 7). However, the next section first reviews the required background on semantic parsing with CCG.

4 Background

4.1 Lambda Calculus

We represent the meanings of sentences, words and phrases with logical expressions that can contain constants, quantifiers, logical connectors and lambda abstractions. We construct the meanings of sentences from the meanings of words and phrases using lambda-calculus operations. We use a version of the typed lambda calculus (Carpenter, 1997), in which the basic types include e, for entities; t, for truth values; and i for numbers. We also have function types that are assigned to lambda expressions.

The expression $\lambda x.flight(x)$ takes an entity and returns a truth value, and has the function type $\langle e,t \rangle$.

4.2 Combinatory Categorial Grammar

CCG (Steedman, 1996, 2000) is a linguistic formalism that tightly couples syntax and semantics, and can be used to model a wide range of language phenomena. A traditional CCG grammar includes a lexicon Λ with entries like the following:

$$\begin{aligned} &flights \vdash N : \lambda x.flight(x) \\ &\quad to \vdash (N\backslash N)/NP : \lambda y.\lambda f.\lambda x.f(x) \wedge to(x,y) \\ &Boston \vdash NP : bos \end{aligned}$$

where each lexical item w$\vdash X : h$ has words w, a syntactic category X, and a logical form h. For the first example, these are "flights," N, and $\lambda x.flight(x)$. In this paper, we introduce a new way of representing lexical items as (*lexeme*, *template*) pairs, as described in section 5.

CCG syntactic categories may be atomic (such as S or NP) or complex (such as $(N\backslash N)/NP$) where the slash combinators encode word order information. CCG uses a small set of *combinatory rules* to build syntactic parses and semantic representations *concurrently*. Two example combinatory rules are forward ($>$) and backward ($<$) *application*:

$$\begin{array}{llll} X/Y : f & Y : g & \Rightarrow & X : f(g) \quad (>) \\ Y : g & X\backslash Y : f & \Rightarrow & X : f(g) \quad (<) \end{array}$$

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 phrase "flights to Boston" can be parsed to produce:

$$\dfrac{\dfrac{\text{flights}}{\begin{array}{c}N \\ \lambda x.flight(x)\end{array}} \quad \dfrac{\dfrac{\text{to}}{\begin{array}{c}(N\backslash N)/NP \\ \lambda y\lambda f\lambda x.f(x) \wedge to(x,y)\end{array}} \quad \dfrac{\text{Boston}}{\begin{array}{c}NP \\ bos\end{array}}}{\begin{array}{c}(N\backslash N) \\ \lambda f\lambda x.f(x) \wedge to(x,bos)\end{array}}>}{\begin{array}{c}N \\ \lambda x.flight(x) \wedge to(x,bos)\end{array}}<$$

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*:

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

These rules allow a relaxed notion of constituency which helps limit the number of distinct CCG lexical items required.

To the standard forward and backward slashes of CCG we also add a vertical slash for which the direction of application is underspecified. We shall see examples of this in Section 10.

4.3 Probabilistic CCGs

Due to ambiguity in both the CCG lexicon and the order in which combinators are applied, there will be many parses for each sentence. We discriminate between competing parses using a log-linear model which has a feature vector ϕ and a parameter vector θ. The probability of a parse y that returns logical form z, given a sentence x is defined as:

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

Section 8 fully defines the set of features used in the system presented. The most important of these control the generation of lexical items from (lexeme, template) pairs. Each (lexeme, template) pair used in a parse fires three features as we will see in more detail later.

The parsing, or inference, problem done at test time requires us to find the most likely logical form z given a sentence x, assuming the parameters θ and lexicon Λ are known:

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

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

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

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. For a sentence of length m, each parse chart span is pruned using a beam width proportional to $m^{\frac{2}{3}}$, to allow larger beams for shorter sentences.

5 Factored Lexicons

A factored lexicon includes a set L of lexemes and a set T of lexical templates. In this section, we formally define these sets, and describe how they are used to build CCG parses. We will use a set of lexical items from our running example to discuss the details of how the following lexical items:

(1) $flight \vdash N : \lambda x.flight(x)$
(2) $flight \vdash N/(S|NP) : \lambda f \lambda x.flight(x) \wedge f(x)$
...
(6) $Boston \vdash NP : bos$
(7) $Boston \vdash N\backslash N : \lambda f \lambda x.from(x,bos) \wedge f(x)$

are constructed from specific lexemes and templates.

5.1 Lexemes

A lexeme $(\mathrm{w}, \vec{c})$ pairs a word sequence w with an ordered list of logical constants $\vec{c} = [c_1 \ldots c_m]$. For example, item (1) and (2) above would come from a single lexeme $(\text{flight}, [flight])$. Similar lexemes would be represented for other predicates, for example $(\text{fare}, [cost])$. Lexemes also can contain multiple constants, for example $(\text{cheapest}, [argmin, cost])$, which we will see more examples of later.

5.2 Lexical Templates

A lexical template takes a lexeme and produces a lexical item. Templates have the general form

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

where $h_{\vec{v}}$ is a logical expression that contains variables from the list $\vec{v}$. Applying this template to the input lexeme $(\mathrm{w}, \vec{c})$ gives the full lexical item $\mathrm{w} \vdash X : h$ where the variable ω has been replaced with the wordspan w and the logical form h has been created by replacing each of the variables in $\vec{v}$ with the counterpart constant from $\vec{c}$. For example, the lexical item (6) above would be constructed from the lexeme $(\text{Boston}, [bos])$ using the template $\lambda(\omega, \vec{v}).[\omega \vdash NP : v_1]$. Items (1) and (2) would both be constructed from the single lexeme $(\text{flight}, [flight])$ with the two different templates $\lambda(\omega, \vec{v}).[\omega \vdash N : \lambda x.v_1(x)]$ and $\lambda(\omega, \vec{v}).[\omega \vdash N/(S|NP) : \lambda f \lambda x.v_1(x) \wedge f(x)]$

5.3 Parsing with a Factored Lexicon

In general, there can by many different (lexeme, template) pairs that produce the same lexical item. For example, lexical item (7) in our running example above can be constructed from the lexemes $(\text{Boston}, [bos])$ and $(\text{Boston}, [from, bos])$, given appropriate templates.

To model this ambiguity, we include the selection of a (lexeme, template) pair as a decision to be made while constructing a CCG parse tree. Given the lexical item produced by the chosen lexeme and template, parsing continues with the traditional combinators, as reviewed in Section 4.2. This direct integration allows for features that signal which lexemes and templates have been used while also allowing for well defined marginal probabilities, by summing over all ways of deriving a specific lexical item.

6 Learning Factored Lexicons

To induce factored lexicons, we will make use of two procedures, presented in this section, that factor lexical items into lexemes and templates. Section 7 will describe how this factoring operation is integrated into the complete learning algorithm.

6.1 Maximal Factorings

Given a lexical item l of the form $w \vdash X : h$ with words w, a syntactic category X, and a logical form h, we define the *maximal factoring* to be the unique (lexeme, template) pair that can be used to reconstruct l and includes all of the constants of h in the lexeme (listed in a fixed order based on an ordered traversal of h). For example, the maximal factoring for the lexical item $Boston \vdash NP : bos$ is the pair we saw before: $(\text{Boston}, [bos])$ and $\lambda(\omega, \vec{v}).[\omega \vdash NP : v_1]$. Similarly, the lexical item $Boston \vdash N\backslash N : \lambda f.\lambda x.f(x) \wedge from(x,bos)$ would be factored to produce $(\text{Boston}, [from, bos])$ and $\lambda(\omega, \vec{v}).[\omega \vdash N\backslash N : \lambda f.\lambda x.f(x) \wedge v_1(x, v_2)]$.

As we will see in Section 7, this notion of factor-

ing can be directly incorporated into existing algorithms that learn CCG lexicons. When the original algorithm would have added an entry l to the lexicon, we can instead compute the factoring of l and add the corresponding lexeme and template to the factored lexicon.

6.2 Introducing Templates with Content

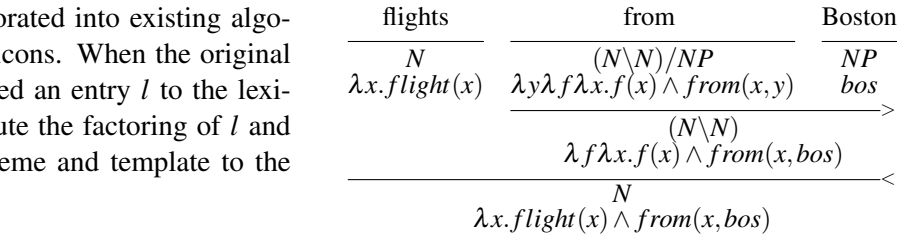
Maximal factorings, as just described, provide for significant lexical generalization but do not handle all of the cases needed to learn effectively. For instance, the maximal split for the item $Boston \vdash N\backslash N : \lambda f.\lambda x.f(x) \wedge from(x,bos)$ would introduce the lexeme $(\text{Boston}, [from, bos])$, which is suboptimal since each possible city would need a lexeme of this type, with the additional *from* constant included. Instead, we would ideally like to learn the lexeme $(\text{Boston}, [bos])$ and have a template that introduces the *from* constant. This would model the desired generalization with a single lexeme per city.

In order to permit the introduction of extra constants into lexical items, we allow the creation of templates that contain logical constants through *partial factorings*. For instance, the template below can introduce the predicate *from*

$$\lambda(\omega, \vec{v}).[\omega \vdash N\backslash N : \lambda f.\lambda x.f(x) \wedge from(x, v_1)]$$

The use of templates to introduce extra semantic constants into a lexical item is similar to, but more general than, the English-specific *type-shifting* rules used in Zettlemoyer and Collins (2007), which were introduced to model spontaneous, unedited text. They are useful, as we will see, in learning to recover semantic content that is implied, but not explicitly stated, such as our original motivating phrase "flights Boston to New York."

To propose templates which introduce semantic content, during learning, we build on the intuition that we need to recover from missing words, such as in the example above. In this scenario, there should also be other sentences that actually include the word, in our example this would be something like "flights from Boston." We will also assume that we have learned a good factored lexicon for the complete example that could produce the parse:



Given analyses of this form, we introduce new templates that will allow us to recover from missing words, for example if "from" was dropped. We identify commonly occurring nodes in the best parse trees found during training, in this case the nonterminal spanning "from Boston," and introduce templates that can produce the nonterminal, even if one of the words is missing. Here, this approach would introduce the desired template $\lambda(\omega, \vec{v}).[\omega \vdash N\backslash N : \lambda f.\lambda x.f(x) \wedge from(x, v_1)]$ for mapping the lexeme $(\text{Boston}, [bos])$ directly to the intermediate structure.

Not all templates introduced this way will model valid generalizations. However, we will incorporate them into a learning algorithm with indicator features that can be weighted to control their use. The next section presents the complete approach.

7 Learning Factored PCCGs

Our Factored Unification Based Learning (FUBL) method extends the UBL algorithm (Kwiatkowski et al., 2010) to induce factored lexicons, while also simultanously estimating the parameters of a log-linear CCG parsing model. In this section, we first review the NEW-LEX lexical induction procedure from UBL, and then present the FUBL algorithm.

7.1 Background: NEW-LEX

NEW-LEX generates lexical items by *splitting* and *merging* nodes in the best parse tree of each training example. Each parse node has a CCG category $X : h$ and a sequence of words w that it spans. We will present an overview of the approach using the running example with the phrase $w =$"in Boston" and the category $X : h = S\backslash NP : \lambda x.loc(x, bos)$, which is of the type commonly seen during learning. The splitting procedure is a two step process that first splits the logical form h, then splits the CCG syntactic category X and finally splits the string w.

The first step enumerates all possible splits of the logical form h into a pair of new expressions

(f, g) that can be used to reconstruct h by either function application ($h = f(g)$) or composition ($h = \lambda x.f(g(x))$). For example, one possible split is:

$$(f = \lambda y.\lambda x.loc(x,y)\ ,\ g = bos)$$

which corresponds to the function application case.

The next two steps enumerate all ways of splitting the syntactic category X and words w to introduce two new lexical items which can be recombined with CCG combinators (application or composition) to recreate the original parse node $X : h$ spanning w. In our example, one possibility would be:

$$(\text{in} \vdash (S\backslash NP)/NP : \lambda y.\lambda x.loc(x,y)\ ,\ \text{Boston} \vdash NP : bos)$$

which could be recombined with the forward application combinator from Section 4.2.

To assign categories while splitting, the grammar used by NEW-LEX only uses two atomic syntactic categories S and NP. This allows NEW-LEX to make use of a direct mapping from semantic type to syntactic category when proposing syntactic categories. In this schema, the standard syntactic category N is replaced by the category $S|NP$ which matches the type $\langle e,t \rangle$ and uses the vertical slash introduced in Section 4.2. We will see categories such as this in the evaluation.

7.2 The FUBL Algorithm

Figure 1 shows the FUBL learning algorithm. We assume training data $\{(x_i, z_i) : i = 1 \ldots n\}$ where each example is a sentence x_i paired with a logical form z_i. The algorithm induces a factored PCCG, including the lexemes L, templates T, and parameters θ.

The algorithm is online, repeatedly performing both lexical expansion (Step 1) and a parameter update (Step 2) for each training example. The overall approach is closely related to the UBL algorithm (Kwiatkowski et al., 2010), but includes extensions for updating the factored lexicon, as motivated in Section 6.

Initialization The model is initialized with a factored lexicon as follows. MAX-FAC is a function that takes a lexical item l and returns the maximal factoring of it, that is the unique, maximal (lexeme, template) pair that can be combined to construct l, as described in Section 6.1. We apply MAX-FAC to each of the training examples (x_i, z_i), creating a single way of producing the desired meaning z_i from a lexeme containing all of the words in x_i. The lexemes and templates created in this way provide the initial factored lexicon.

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 entity name lexemes L_e. Number of iterations J. Learning rate parameter α_0 and cooling rate parameter c. Empty lexeme set L. Empty template set T.

Definitions: NEW-LEX(y) returns a set of new lexical items from a parse y as described in Section 7.1. MAX-FAC(l) generates a (lexeme, template) pair from a lexical item l. PART-FAC(y) generates a set of templates from parse y. Both of these are described in Section 7.2. The distributions $p(y|x,z;\theta,(L,T))$ and $p(y,z|x;\theta,(L,T))$ are defined by the log-linear model described in Section 4.3.

Initialization:

- For $i = 1 \ldots n$
  - $(\psi, \pi) = \text{MAX-FAC}(x_i \vdash S : z_i)$
  - $L = L \cup \psi\ ,\ T = T \cup \pi$
- Set $L = L \cup L_e$.
- Initialize θ using coocurrence statistics, as described in Section 8.

Algorithm:

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

Step 1: (Add Lexemes and Templates)

- Let $y^* = \arg\max_y p(y|x_i, z_i; \theta, (L,T))$
- For $l \in \text{NEW-LEX}(y^*)$
  - $(\psi, \pi) = \text{MAX-FAC}(l)$
  - $L = L \cup \psi\ ,\ T = T \cup \pi$
- $\Pi = \text{PART-FAC}(y^*)\ ,\ T = T \cup \Pi$

Step 2: (Update Parameters)

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

Output: Lexemes L, templates T, and parameters θ.

Figure 1: The FUBL learning algorithm.

Step 1 The first step of the learning algorithm in Figure 1 adds lexemes and templates to the factored model given by performing manipulations on the highest scoring correct parse y^* of the current training example (x_i, z_i). First the NEW-LEX procedure is run on y^* as described in Section 6.1 to

generate new lexical items. We then use the function MAX-FAC to create the maximal factorings of each of these new lexical items as described in Section 6 and these are added to the factored representation of the lexicon. New templates can also be introduced through partial factorings of internal parse nodes as described in Section 6.2. These templates are generated by using the function PART-FAC to abstract over the wordspan and a subset of the constants contained in the internal parse nodes of y^*. This step allows for templates that introduce new semantic content to model elliptical language, as described in Section 6.2.

Step 2 The second step does a stochastic gradient descent update on the parameters θ used in the parsing model. This update is described in Section 4.3

Discussion The FUBL algorithm makes use of a direct online approach, where lexemes and templates are introduced in place while analyzing specific sentences. In general, this will overgeneralize; not all ways of combining lexemes and templates will produce high quality lexical items. However, the overall approach includes features, presented in Section 8, that can be used to learn which ones are best in practice. The complete algorithm iterates between adding new lexical content and updating the parameters of the parsing model with each procedure guiding the other.

8 Experimental setup

Data Sets We evaluate on two benchmark semantic parsing datasets: GeoQuery, which is made up of natural language queries to a database of geographical information; and Atis, which contains natural language queries to a flight booking system. The Geo880 dataset has 880 (English-sentence, logical-form) pairs split into a training set of 600 pairs and a test set of 280. The Geo250 data is a subset of the Geo880 sentences that have been translated into Japanese, Spanish and Turkish as well as the original English. We follow the standard evaluation procedure for Geo250, using 10-fold cross validation experiments with the same splits of the data as Wong and Mooney (2007). The Atis dataset contains 5410 (sentence, logical-form) pairs split into a 4480 example training set, a 480 example development set and a 450 example test set.

Evaluation Metrics We report exact match *Recall* (percentage of sentences for which the correct logical-form was returned), *Precision* (percentage of returned logical-forms that are correct) and *F1* (harmonic mean of Precision and Recall). For Atis we also report partial match Recall (percentage of correct literals returned), Precision (percentage of returned literals that are correct) and F1, computed as described by Zettlemoyer and Collins (2007).

Features We introduce two types of features to discriminate between parses: *lexical features* and *logical-form features*.

Lexical features fire on the lexemes and templates used to build the lexical items used in a parse. For each (lexeme,template) pair used to create a lexical item we have indicator features ϕ_l for the lexeme used, ϕ_t for the template used, and $\phi_{(l,t)}$ for the pair that was used. We assign the features on lexical templates a weight of 0.1 to prevent them from swamping the far less frequent but equally informative lexeme features.

Logical-form features are computed on the lambda-calculus expression z returned at the root of the parse. Each time a predicate p in z takes an argument a with type $Ty(a)$ in position i, it triggers two binary indicator features: $\phi_{(p,a,i)}$ for the predicate-argument relation; and $\phi_{(p,Ty(a),i)}$ for the predicate argument-type relation. Boolean operator features look at predicates that occurr together in conjunctions and disjunctions. For each variable v_i that fills argument slot i in two conjoined predicates p_1 and p_2 we introduce a binary indicator feature $\phi_{conj(i,p_1,p_2)}$. We introduce similar features $\phi_{disj(i,p_1,p_2)}$ for variables v_i that are shared by predicates in a disjunction.

Initialization The weights for lexeme features are initialized according to coocurrance statistics between words and logical constants. These are estimated with the Giza++ (Och and Ney, 2003) implementation of IBM Model 1. The initial weights for templates are set by adding -0.1 for each slash in the syntactic category and -2 if the template contains logical constants. Features on lexeme-template pairs and all parse features are initialized to zero.

Systems We compare performance to all recently-published, directly-comparable results. For GeoQuery, this includes the ZC05, ZC07 (Zettlemoyer

System	Exact Match		
	Rec.	Pre.	F1
ZC07	74.4	**87.3**	80.4
UBL	65.6	67.1	66.3
FUBL	**81.9**	82.1	**82.0**

Table 1: Performance on the Atis development set.

System	Exact Match			Partial Match		
	Rec.	Pre.	F1.	Rec.	Pre.	F1
ZC07	**84.6**	**85.8**	**85.2**	**96.7**	**95.1**	**95.9**
HY06	-	-	-	-	-	90.3
UBL	71.4	72.1	71.7	78.2	98.2	87.1
FUBL	82.8	82.8	82.8	95.2	93.6	94.6

Table 2: Performance on the Atis test set.

and Collins, 2005, 2007), λ-WASP (Wong and Mooney, 2007), UBL (Kwiatkowski et al., 2010) systems and DCS (Liang et al., 2011). For Atis, we report results from HY06 (He and Young, 2006), ZC07, and UBL.

9 Results

Tables 1-4 present the results on the Atis and Geoquery domains. In all cases, FUBL achieves at or near state-of-the-art recall (overall number of correct parses) when compared to directly comparable systems and it significantly outperforms UBL on Atis.

On Geo880 the only higher recall is achieved by DCS with prototypes - which uses significant English-specific resources, including manually specified lexical content, but does not require training sentences annotated with logical-forms. On Geo250, FUBL achieves the highest recall across languages. Each individual result should be interpreted with care, as a single percentage point corresponds to 2-3 sentences, but the overall trend is encouraging.

On the Atis development set, FUBL outperforms ZC07 by 7.5% of recall but on the Atis test set FUBL lags ZC07 by 2%. The reasons for this discrepancy are not clear, however, it is possible that the syntactic constructions found in the Atis test set do not exhibit the same degree of variation as those seen in the development set. This would negate the need for the very general lexicon learnt by FUBL.

Across the evaluations, despite achieving high recall, FUBL achieves significantly lower precision than ZC07 and λ-WASP. This illustrates the trade-off from having a very general model of proposing lexical structure. With the ability to skip unseen words, FUBL returns a parse for all of the Atis test sentences, since the factored lexicons we are learning can produce a very large number of lexical items. These parses are, however, not always correct.

System	Rec.	Pre.	F1
Labelled Logical Forms			
ZC05	79.3	**96.3**	87.0
ZC07	86.1	91.6	**88.8**
UBL	87.9	88.5	88.2
FUBL	**88.6**	88.6	88.6
Labelled Question Answers			
DCS	**91.1**	-	-

Table 3: Exact match accuracy on the Geo880 test set.

System	English			Spanish		
	Rec.	Pre.	F1	Rec.	Pre.	F1
λ-WASP	75.6	**91.8**	82.9	80.0	**92.5**	**85.8**
UBL	81.8	83.5	82.6	81.4	83.4	82.4
FUBL	**83.7**	83.7	**83.7**	**85.6**	85.8	85.7
System	Japanese			Turkish		
	Rec.	Pre.	F1	Rec.	Pre.	F1
λ-WASP	81.2	**90.1**	**85.8**	68.8	**90.4**	**78.1**
UBL	83.0	83.2	83.1	71.8	77.8	74.6
FUBL	**83.2**	83.8	83.5	**72.5**	73.7	73.1

Table 4: Exact-match accuracy on the Geo250 data set.

10 Analysis

The Atis results in Tables 1 and 2 highlight the advantages of factored lexicons. FUBL outperforms the UBL baseline by 16 and 11 points respectively in exact-match recall. Without making any modification to the CCG grammars or parsing combinators, we are able to induce a lexicon that is general enough model the natural occurring variations in the data, for example due to sloppy, unedited sentences.

Figure 2 shows a parse returned by FUBL for a sentence on which UBL failed. While the word "cheapest" is seen 208 times in the training data, in only a handful of these instances is it seen in the middle of an utterance. For this reason, UBL never proposes the lexical item, cheapest $\vdash NP\backslash(S|NP)/(S|NP) : \lambda f\lambda g.argmin(\lambda x.f(x) \wedge g(x), \lambda y.cost(y))$, which is used to parse the sentence in Figure 2. In contrast, FUBL uses a lexeme learned from the same word in different contexts, along with a template learnt from similar words in a similar context, to learn to per-

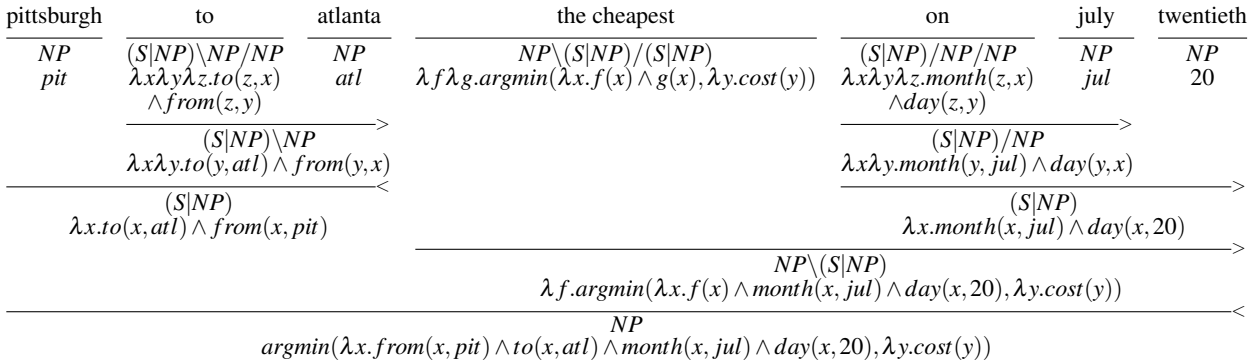


Figure 2: An example learned parse. FUBL can learn this type of analysis with novel combinations of lexemes and templates at test time, even if the individual words, like "cheapest," were never seen in similar syntactic constructions during training, as described in Section 10.

form the desired analysis.

As well as providing a new way to search the lexicon during training, the factored lexicon provides a way of proposing new, unseen, lexical items at test time. We find that new, non-*NP*, lexical items are used in 6% of the development set parses.

Interestingly, the addition of templates that introduce semantic content (as described in Section 6.2) account for only 1.2% of recall on the Atis development set. This is suprising as elliptical constructions are found in a much larger proportion of the sentences than this. In practice, FUBL learns to model many elliptical constructions with lexemes and templates introduced through maximal factorings. For example, the lexeme $(\text{to}, [from, to])$ can be used with the correct lexical template to deal with our motivating example "flights Boston to New York". Templates that introduce content are therefore only used in truly novel elliptical constructions for which an alternative analysis could not be learned.

Table 5 shows a selection of lexemes and templates learned for Atis. Examples 2 and 3 show that morphological variants of the same word must still be stored in separate lexemes. However, as these lexemes now share templates, the total number of lexical variants that must be learned is reduced.

Most common lexemes by type of constants in $\vec{c}$.			
1	e	(Boston, $[bos]$)	(Denver, $[den]$)
2	$\langle e,t \rangle$	(flight, $[flight]$)	(flights, $[flight]$)
3	$\langle e,i \rangle$	(fare, $[cost]$)	(fares, $[cost]$)
4	$\langle e, \langle e,t \rangle \rangle$	(from, $[from]$)	(to, $[to]$)
5	$\langle e,i \rangle, \langle e,t \rangle$	(cheapest, $[argmin, cost]$)	(earliest, $[argmin, dep\_time]$)
6	$\langle i, \langle i,t \rangle \rangle$, $\langle e,i \rangle$	(after, $[>, dep\_time]$)	(before, $[<, dep\_time]$)
Most common templates matching lexemes above.			
1	$\lambda(\omega, \vec{v}).\omega \vdash NP : v_1$		
2	$\lambda(\omega, \vec{v}).\omega \vdash S\|NP : \lambda x.v_1(x)$		
3	$\lambda(\omega, \vec{v}).\omega \vdash NP\|NP : \lambda x.v_1(x)$		
4	$\lambda(\omega, \vec{v}).\omega \vdash S\|NP/NP\backslash(S\|NP) : \lambda x\lambda y.v_1(x,y)$		
5	$\lambda(\omega, \vec{v}).\omega \vdash NP/(S\|NP) : \lambda f.v_1(\lambda x.f(x), \lambda y, v_2(y))$		
6	$\lambda(\omega, \vec{v}).\omega \vdash S\|NP\backslash(S\|NP)/NP : \lambda x\lambda y\lambda z.v_1(v_2(z),x) \wedge y(x)$		

Table 5: Example lexemes and templates learned from the Atis development set.

11 Discussion

We argued that factored CCG lexicons, which include both lexemes and lexical templates, provide a compact representation of lexical knowledge that can have advantages for learning. We also described a complete approach for inducing factored, probabilistic CCGs for semantic parsing, and demonstrated strong performance across a wider range of benchmark datasets that any previous approach.

In the future, it will also be important to explore morphological models, to better model variation within the existing lexemes. The factored lexical representation also has significant potential for lexical transfer learning, where we would need to learn new lexemes for each target application, but much of the information in the templates could, potentially, be ported across domains.

Acknowledgements

The work was supported in part by EU ERC Advanced Fellowship 249520 GRAMPLUS, and an ESPRC PhD studentship. We would like to thank Yoav Artzi for helpful discussions.

References

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

Branavan, S.R.K., Luke Zettlemoyer, and Regina Barzilay. 2010. Reading between the lines: Learning to map high-level instructions to commands. In *Association for Computational Linguistics (ACL)*.

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

Clark, Stephen 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*.

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

Clarke, James, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning (CoNLL-2010)*. Uppsala, Sweden, pages 18–27.

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

Goldwasser, Dan, Roi Reichart, James Clarke, and Dan Roth. 2011. Confidence driven unsupervised semantic parsing. In *Association for Computational Linguistics (ACL)*.

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

He, Yulan and Steve Young. 2006. Spoken language understanding using the hidden vector state model. *Speech Communication* 48(3-4).

Hockenmaier, Julia and Mark Steedman. 2002. Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Meeting of the ACL*. Philadelphia, PA, pages 335–342.

Kate, Rohit J. 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*.

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

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

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

Liang, P., M. I. Jordan, and D. Klein. 2009. Learning semantic correspondences with less supervision. In *Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*.

Liang, P., M. I. Jordan, and D. Klein. 2011. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*.

Lu, Wei, Hwee Tou Ng, Wee Sun Lee, and Luke S. Zettlemoyer. 2008. A generative model for parsing natural language to meaning representations. In *Proceedings of The Conference on Empirical Methods in Natural Language Processing*.

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

Nguyen, Le-Minh, 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*.

Och, Franz Josef and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics* 29(1):19–51.

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

Poon, Hoifung and Pedro Domingos. 2009. Unsupervised semantic parsing. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Poon, Hoifung and Pedro Domingos. 2010. Unsupervised ontology induction from text. In *Association for Computational Linguistics (ACL)*.

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

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

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

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

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

Vogel, Adam and Dan Jurafsky. 2010. Learning to follow navigational directions. In *Association for Computational Linguistics (ACL)*.

Watkinson, Stephen 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*.

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

Wong, Yuk Wah and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the Association for Computational Linguistics*.

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

Zettlemoyer, Luke S. and Michael Collins. 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, Luke S. and Michael Collins. 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, Luke S. and Michael Collins. 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*.