Scaling Semantic Parsers with On-the-fly Ontology Matching

Tom Kwiatkowski **Eunsol Choi** **Yoav Artzi** **Luke Zettlemoyer**
Computer Science & Engineering
University of Washington
Seattle, WA 98195
{tomk,eunsol,yoav,lsz}@cs.washington.edu

Abstract

We consider the challenge of learning semantic parsers that scale to large, open-domain problems, such as question answering with Freebase. In such settings, the sentences cover a wide variety of topics and include many phrases whose meaning is difficult to represent in a fixed target ontology. For example, even simple phrases such as 'daughter' and 'number of people living in' cannot be directly represented in Freebase, whose ontology instead encodes facts about gender, parenthood, and population. In this paper, we introduce a new semantic parsing approach that learns to resolve such ontological mismatches. The parser is learned from question-answer pairs, uses a probabilistic CCG to build linguistically motivated logical-form meaning representations, and includes an ontology matching model that adapts the output logical forms for each target ontology. Experiments demonstrate state-of-the-art performance on two benchmark semantic parsing datasets, including a nine point accuracy improvement on a recent Freebase QA corpus.

1 Introduction

Semantic parsers map sentences to formal representations of their underlying meaning. Recently, algorithms have been developed to learn such parsers for many applications, including question answering (QA) (Kwiatkowski et al., 2011; Liang et al., 2011), relation extraction (Krishnamurthy and Mitchell, 2012), robot control (Matuszek et al., 2012; Krishnamurthy and Kollar, 2013), interpreting instructions (Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013b), and generating programs (Kushman and Barzilay, 2013).

In each case, the parser uses a predefined set of logical constants, or an ontology, to construct meaning representations. In practice, the choice of ontology significantly impacts learning. For example, consider the following questions (Q) and candidate meaning representations (MR):

Q1: What is the population of Seattle?
Q2: How many people live in Seattle?
MR1: $\lambda x.population(Seattle, x)$
MR2: $count(\lambda x.person(x) \wedge live(x, Seattle))$

A semantic parser might aim to construct MR1 for Q1 and MR2 for Q2; these pairings align constants ($count$, $person$, etc.) directly to phrases ('How many,' 'people,' etc.). Unfortunately, few ontologies have sufficient coverage to support both meaning representations, for example many QA databases would only include the population relation required for MR1. Most existing approaches would, given this deficiency, simply aim to produce MR1 for Q2, thereby introducing significant lexical ambiguity that complicates learning. Such ontological mismatches become increasingly common as domain and language complexity increases.

In this paper, we introduce a semantic parsing approach that supports scalable, open-domain ontological reasoning. The parser first constructs a linguistically motivated domain-independent meaning representation. For example, possibly producing MR1 for Q1 and MR2 for Q2 above. It then uses a learned ontology matching model to transform this represen-

x :	How many people visit the public library of New York annually
l_0 :	$\lambda x.eq(x, count(\lambda y.people(y) \wedge \exists e.visit(y, \imath z.public(z) \wedge library(z) \wedge of(z, new\ york), e) \wedge annually(e)))$
y :	λx.library.public_library_system.annual_visits(x, new_york_public_library)
a :	13,554,002

x :	What works did Mozart dedicate to Joseph Haydn
l_0 :	$\lambda x.works(x) \wedge \exists e.dedicate(mozart, x, e) \wedge to(haydn, e)))$
y :	λx.dedicated_work(x) $\wedge$ $\exists$e.dedicated_by(mozart, e) $\wedge$ dedication(x, e) $\wedge$ dedicated_to(haydn, e)))
a :	{ String Quartet No. 19, Haydn Quartets, String Quartet No. 16, String Quartet No. 18, String Quartet No. 17 }

Figure 1: Examples of sentences x, domain-independent underspecified logical forms l_0, fully specified logical forms y, and answers a drawn from the Freebase domain.

tation for the target domain. In our example, producing either MR1, MR2 or another more appropriate option, depending on the QA database schema. This two stage approach enables parsing without any domain-dependent lexicon that pairs words with logical constants. Instead, word meaning is filled in on-the-fly through ontology matching, enabling the parser to infer the meaning of previously unseen words and more easily transfer across domains. Figure 1 shows the desired outputs for two example Freebase sentences.

The first parsing stage uses a probabilistic combinatory categorial grammar (CCG) (Steedman, 2000; Clark and Curran, 2007) to map sentences to new, *underspecified* logical-form meaning representations containing generic logical constants that are not tied to any specific ontology. This approach enables us to share grammar structure across domains, instead of repeatedly re-learning different grammars for each target ontology. The ontology-matching step considers a large number of type-equivalent domain-specific meanings. It enables us to incorporate a number of cues, including the target ontology structure and lexical similarity between the names of the domain-independent and dependent constants, to construct the final logical forms.

During learning, we estimate a linear model over derivations that include all of the CCG parsing decisions and the choices for ontology matching. Following a number of recent approaches (Clarke et al., 2010; Liang et al., 2011), we treat all intermediate decisions as latent and learn from data containing only easily gathered question answer pairs. This approach aligns naturally with our two-stage parsing setup, where the final logical expression can be directly used to provide answers.

We report performance on two benchmark datasets: GeoQuery (Zelle and Mooney, 1996) and Freebase QA (FQ) (Cai and Yates, 2013a). GeoQuery includes a geography database with a small ontology and questions with relatively complex, compositional structure. FQ includes questions to Freebase, a large community-authored database that spans many sub-domains. Experiments demonstrate state-of-the-art performance in both cases, including a nine point improvement in recall for the FQ test.

2 Formal Overview

Task Let an ontology $\mathcal{O}$ be a set of logical constants and a knowledge base $\mathcal{K}$ be a collection of logical statements constructed with constants from $\mathcal{O}$. For example, $\mathcal{K}$ could be facts in Freebase (Bollacker et al., 2008) and $\mathcal{O}$ would define the set of entities and relation types used to encode those facts. Also, let y be a logical expression that can be executed against $\mathcal{K}$ to return an answer $a = \text{EXEC}(y, \mathcal{K})$. Figure 1 shows example queries and answers for Freebase. Our goal is to build a function $y = \text{PARSE}(x, \mathcal{O})$ for mapping a natural language sentence x to a domain-dependent logical form y.

Parsing We use a two-stage approach to define the space of possible parses $\text{GEN}(x, \mathcal{O})$ (Section 5). First, we use a CCG and word-class information from Wiktionary[1] to build domain-independent underspecified logical forms, which closely mirror the linguistic structure of the sentence but do not use constants from $\mathcal{O}$. For example, in Figure 1, l_0 denotes the underspecified logical forms paired with each sentence x. The parser then maps this intermediate representation to a logical form that uses constants from $\mathcal{O}$, such as the y seen in Figure 1.

[1] www.wiktionary.com

Learning We assume access to data containing question-answer pairs $\{(x_i, a_i) : i = 1 \dots n\}$ and a corresponding knowledge base $\mathcal{K}$. The learning algorithm (Section 7.1) estimates the parameters of a linear model for ranking the possible entires in $\text{GEN}(x, \mathcal{O})$. Unlike much previous work (e.g., Zettlemoyer and Collins (2005)), we do not induce a CCG lexicon. The lexicon is open domain, using no symbols from the ontology $\mathcal{O}$ for $\mathcal{K}$. This allows us to write a single set of lexical templates that are reused in every domain (Section 5.1). The burden of learning word meaning is shifted to the second, ontology matching, stage of parsing (Section 5.2), and modeled with a number of new features (Section 7.2) as part of the joint model.

Evaluation We evaluate on held out question-answer pairs in two benchmark domains, Freebase and GeoQuery. Following Cai and Yates (2013a), we also report a cross-domain evaluation where the Freebase data is divided by topics such as sports, film, and business. This condition ensures that the test data has a large percentage of previously unseen words, allowing us to measure the effectiveness of the real time ontology matching.

3 Related Work

Supervised approaches for learning semantic parsers have received significant attention, e.g. (Kate and Mooney, 2006; Wong and Mooney, 2007; Muresan, 2011; Kwiatkowski et al., 2010, 2011, 2012; Jones et al., 2012). However, these techniques require training data with hand-labeled domain-specific logical expressions. Recently, alternative forms of supervision were introduced, including learning from question-answer pairs (Clarke et al., 2010; Liang et al., 2011), from conversational logs (Artzi and Zettlemoyer, 2011), with distant supervision (Krishnamurthy and Mitchell, 2012; Cai and Yates, 2013b), and from sentences paired with system behavior (Goldwasser and Roth, 2011; Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013b). Our work adds to these efforts by demonstrating a new approach for learning with latent meaning representations that scales to large databases like Freebase.

Cai and Yates (2013a) present the most closely related work. They applied schema matching techniques to expand a CCG lexicon learned with the UBL algorithm (Kwiatkowski et al., 2010). This approach was one of the first to scale to Freebase, but required labeled logical forms and did not jointly model semantic parsing and ontological reasoning. This method serves as the state of the art for our comparison in Section 9.

We build on a number of existing algorithmic ideas, including using CCGs to build meaning representations (Zettlemoyer and Collins, 2005, 2007; Kwiatkowski et al., 2010, 2011), building derivations to transform the output of the CCG parser based on context (Zettlemoyer and Collins, 2009), and using weakly supervised margin-sensitive parameter updates (Artzi and Zettlemoyer, 2011, 2013b).[2] However, we introduce the idea of learning an open-domain CCG semantic parser; all previous methods suffered, to various degrees, from the ontological mismatch problem that motivates our work.

The challenge of ontological mismatch has been previously recognized in many settings. Hobbs (1985) describes the need for ontological promiscuity in general language understanding. Many previous hand-engineered natural language understanding systems (Grosz et al., 1987; Alshawi, 1992; Bos, 2008) are designed to build general meaning representations that are adapted for different domains. Recent efforts to build natural language interfaces to large databases, for example DBpedia (Yahya et al., 2012; Unger et al., 2012), have also used hand-engineered ontology matching techniques. Fader et al. (2013) recently presented a scalable approach to learning an open domain QA system, where ontological mismatches are resolved with learned paraphrases. Finally, the databases research community has a long history of developing schema matching techniques (Doan et al., 2004; Euzenat et al., 2007), which has inspired more recent work on distant supervision for relation extraction with Freebase (Zhang et al., 2012).

4 Background

Semantic Modeling We use the typed lambda calculus to build logical forms that represent the meanings of words, phrases and sentences. Logical forms contain constants, variables, lambda abstractions, and literals. In this paper, we use the term literal to

[2] We build on UW SPF (Artzi and Zettlemoyer, 2013a).

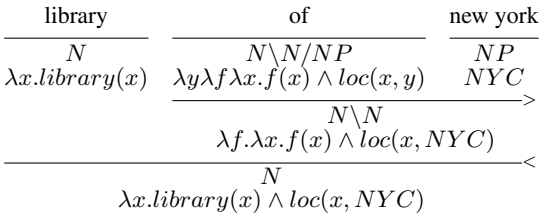


Figure 2: A sample CCG parse.

refer to the application of a constant to a sequence of arguments. We include types for entities e, truth values t, numbers i, events ev, and higher-order functions, such as $\langle e, t\rangle$ and $\langle\langle e, t\rangle, e\rangle$. We use Davidsonian event semantics (Davidson, 1967) to explicitly represent events using event-typed variables and conjunctive modifiers to capture thematic roles.

Combinatory Categorial Grammars (CCG) CCGs are a linguistically-motivated formalism for modeling a wide range of language phenomena (Steedman, 1996, 2000). A CCG is defined by a lexicon and a set of combinators. The lexicon contains entries that pair words or phrases with CCG categories. For example, the lexical entry $\text{library} \vdash N : \lambda x.library(x)$ in Figure 2 pairs the word 'library' with the CCG category that has syntactic category N and meaning $\lambda x.library(x)$. A CCG parse starts from assigning lexical entries to words and phrases. These are then combined using the set of CCG combinators to build a logical form that captures the meaning of the entire sentence. We use the application, composition, and coordination combinators. Figure 2 shows an example parse.

5 Parsing Sentences to Meanings

The function $\text{GEN}(x, \mathcal{O})$ defines the set of possible derivations for an input sentence x. Each derivation $d = \langle \Pi, M\rangle$ builds a logical form y using constants from the ontology $\mathcal{O}$. Π is a CCG parse tree that maps x to an underspecified logical form l_0. M is an ontological match that maps l_0 onto the fully specified logical form y. This section describes, with reference to the example in Figure 3, the operations used by Π and M.

5.1 Domain Independent Parsing

Domain-independent CCG parse trees Π are built using a predefined set of 56 underspecified lexical categories, 49 domain-independent lexical items, and the combinatory rules introduced in Section 4.

An underspecified CCG lexical category has a syntactic category and a logical form containing no constants from the domain ontology $\mathcal{O}$. Instead, the logical form includes underspecified constants that are typed placeholders which will later be replaced during ontology matching. For example, a noun might be assigned the lexical category $\mathsf{N} : \lambda x.p(x)$, where p is an underspecified $\langle e, t\rangle$-type constant.

During parsing, lexical categories are created dynamically. We manually define a set of POS tags for each underspecified lexical category, and use Wiktionary as a tag dictionary to define the possible POS tags for words and phrases. Each phrase is assigned every matching lexical category. For example, the word 'visit' can be either a verb or a noun in Wiktionary. We accordingly assign it all underspecified categories for the classes, including:

$$\mathsf{N} : \lambda x.p(x) \;\;,\;\; \mathsf{S\backslash NP/NP} : \lambda x \lambda y \exists ev.p(y, x, ev)$$

for nouns and transitive verbs respectively.

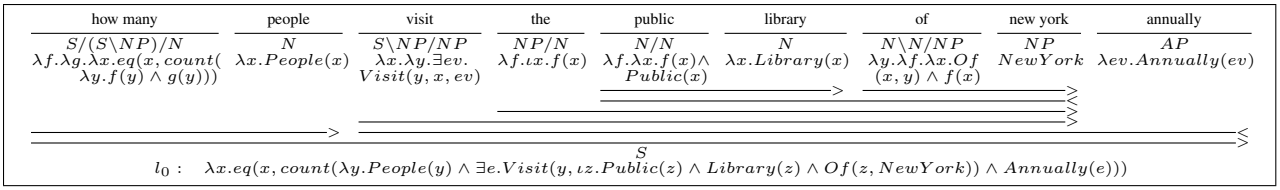
We also define domain-independent lexical items for function words such as 'what,' 'when,' and 'how many,' 'and,' and 'is.' These lexical items pair a word with a lexical category containing only domain-independent constants. For example, $\text{how many} \vdash \mathsf{S/(S\backslash NP)/N} : \lambda f.\lambda g.\lambda x.eq(x, count(\lambda y.f(y) \wedge g(y)))$ contains the function $count$ and the predicate eq.

Figure 3a shows the lexical categories and combinator applications used to construct the underspecified logical form l_0. Underspecified constants in this figure have been labeled with the words that they are associated with for readability.

5.2 Ontological Matching

The second, domain specific, step M maps the underspecified logical form l_0 onto the fully specified logical form y. The mapping from constants in l_0 to constants in y is not one-to-one. For example, in Figure 3, l_0 contains 11 constants but y contains only 2. The ontological match is a sequence of matching operations $M = \langle o_1 \ldots, o_n\rangle$ that can transform the

(a) **Underspecified CCG parse Π:** Map words onto underspecified lexical categories as described in Section 5.1. Use the CCG combinators to combine lexical categories to give the full underpecified logical form l_0.



(b) **Structure Matching Steps in M:** Use the operators described in Section 5.2.1 and Figure 4 to transform l_0. In each step one of the operators is applied to a subexpression of the existing logical form to generate a modified logical form with a new underspecified constant marked in bold.

$$
\begin{aligned}
l_0 &: \lambda x.eq(x, count(\lambda y.People(y) \wedge \exists e.Visit(y, \iota z.Public(z) \wedge Library(z) \wedge Of(z, NewYork), e) \wedge Annually(e))) \\
l_1 &: \lambda x.eq(x, count(\lambda y.People(y) \wedge \exists e.Visit(y, \textbf{PublicLibraryOfNewYork}, e) \wedge Annually(e))) \\
l_2 &: \lambda x.\textbf{HowManyPeopleVisitAnnually}(x, PublicLibraryOfNewYork)))
\end{aligned}
$$

(c) **Constant Matching Steps in M:** Replace all underspecified constants in the transformed logical form with a similarly typed constant from $\mathcal{O}$, as described in Section 5.2.2. The underspecified constant to be replaced is marked in bold and constants from $\mathcal{O}$ are written in typeset.

$$
\begin{aligned}
& \lambda x.HowManyPeopleVisitAnnually(x, \textbf{PublicLibraryOfNewYork}) \\
l_3 : & \qquad \mapsto \lambda x.HowManyPeopleVisitAnnually(x, \texttt{new\_york\_public\_library}) \\
& \lambda x.\textbf{HowManyPeopleVisitAnnually}(x, \texttt{new\_york\_public\_library}) \\
y : & \qquad \mapsto \lambda x.\texttt{public\_library\_system.annual\_visits}(x, \texttt{new\_york\_public\_library})
\end{aligned}
$$

Figure 3: Example derivation for the query 'how many people visit the public library of new york annually.' Underspecified constants are labelled with the words from the query that they are associated with for readability. Constants from $\mathcal{O}$, written in typeset, are introduced in step (c).

	Operator	Definition and Conditions	Example
a.	Collapse Literal to Constant	$P(a_1, \ldots, a_n) \mapsto c$ s.t. $\texttt{type}(P(a_1, \ldots, a_n)) = \texttt{type}(c)$ $\texttt{type}(c) \in \{e, i\}$ $\texttt{freev}(P(a_1, \ldots, a_n)) = \emptyset$	$\iota z.Public(z) \wedge Library(z) \wedge Of(z, NewYork)) \mapsto PublicLibraryOfNewYork$ Input and output have type e. e is allowed in $\mathcal{O}$. Input contains no free variables.
b.	Collapse Literal to Literal	$P(a_1, \ldots, a_n) \mapsto Q(b_1, \ldots, b_m)$ s.t. $\texttt{type}(P(a_1, \ldots, a_n)) = \texttt{type}(Q(b_1, \ldots, b_m))$ $\texttt{type}(Q) \in \{\texttt{type}(c) : c \in \mathcal{O}\}$ $\texttt{freev}(P(a_1, \ldots, a_n)) = \texttt{freev}(Q(b_1, \ldots, b_m))$ $\{b_1, \ldots, b_m\} \in \texttt{subexps}(P(a_1, \ldots, a_n))$	$eq(x, count(\lambda y.People(y) \wedge \exists e.Visit(y, PublicLibraryOfNewYork) \wedge Annually(e))) \mapsto CountPeopleVisitAnnually(x, PublicLibraryOfNewYork)$ Input and output have type t. New constant has type $\langle i, \langle e, t\rangle\rangle$, allowed in $\mathcal{O}$. Input and output contain single free variable x. Arguments of output literal are subexpressions of input.
c.	Split Literal	$P(a_1, \ldots, a_k, x, a_{k+1}, \ldots, a_n) \mapsto Q(b_1, \ldots, x, \ldots b_n) \wedge Q''(c_1, \ldots, x, \ldots c_m)$ s.t. $\texttt{type}(P(\ldots)) = t$ $\{\texttt{type}(Q), \texttt{type}(Q'')\} \in \{\texttt{type}(c) : c \in \mathcal{O}\}$ $\{b_1, \ldots, b_n, c_1, \ldots, c_m\} = \{a_1, \ldots, a_n\}$	$Dedicate(Mozart, Haydn, ev) \mapsto Dedicate(Mozart, ev) \wedge Dedicate''(Haydn, ev)$ Input has type t. This matches output type by definition. New constants have allowed type $\langle e, \langle ev, t\rangle\rangle$. All arguments of input literal are preserved in output.

Figure 4: Definition of the operations used to transform the structure of the underspecified logical form l_0 to match the ontology $\mathcal{O}$. The function $\texttt{type}(c)$ calculates a constant c's type. The function $\texttt{freev}(lf)$ returns the set of variables that are free in lf (not bound by a lambda term or quantifier). The function $\texttt{subexps}(lf)$ generates the set of all subexpressions of the lambda calculus expression lf.

structure of the logical form or replace underspecified constants with constants from $\mathcal{O}$.

5.2.1 Structure Matching

Three structure matching operators, illustrated in Figure 4, are used to collapse or expand literals in l_0. Collapses merge a subexpression from l_0 to create a new underspecified constant, generating a logical form with fewer constants. Expansions split a subexpression from l_0 to generate a new logical form containing one extra constant.

Collapsing Operators The collapsing operator defined in Figure 4a merges all constants in a literal to generate a single constant of the same type. This operator is used to map $\iota z.Public(z) \wedge Library(z) \wedge Of(z,NewYork)$ to $PublicLibraryOfNewYork$ in Figure 3b. Its operation is limited to entity typed expressions that do not contain free variables.

The operator in Figure 4b, in contrast, can be used to collapse the expression $eq(x,count(\lambda y.People(y) \wedge \exists e.Visit(y,PublicLibraryOfNewYork,e)) \wedge Annually(e)))$, which contains free variable x onto a new expression $CountPeopleVisitAnnually(x,PublicLibraryOfNewYork)$. This is only possible when the type of the newly created constant is allowed in $\mathcal{O}$ and the variable x is free in the output expression. Subsets of conjuncts can be collapsed using the operator in Figure 4b by creating ad-hoc conjunctions that encapsulate them. Disjunctions are treated similarly.

Performing collapses on the underspecified logical form allows non-contiguous phrases to be represented in the collapsed form. In this example, the logical form representing the phrase 'how many people visit' has been merged with the logical form representing the non-adjacent adverb 'annually.' This generates a new underspecified constant that can be mapped onto the Freebase relation `public_library_system_annual_visits` that relates to both phrases.

The collapsing operations preserve semantic type, ensuring that all logical forms generated by the derivation sequence are well typed. The full set of allowed collapses of l_0 is given by the transitive closure of the collapsing operations. The size of this set is limited by the number of constants in l_0, since each collapse removes at least one constant. At each step, the number of possible collapses is polynomial in the number of constants in l_0 and exponential in the arity of the most complex type in $\mathcal{O}$. For domains of interest this arity is unlikely to be high and for triple stores such as Freebase it is 2.

Expansion Operators The fully specified logical form y can contain constants relating to multiple words in x. It can also use multiple constants to represent the meaning of a single word. For example, Freebase does not contain a relation representing the concept 'daughter', instead using two relations representing 'female' and 'child'. The expansion operator in Figure 4c allows a single predicate to be split into a pair of conjoined predicates sharing an argument variable. For example, in Figure 1, the constant for 'dedicate' is split in two to match its representation in Freebase. Underspecified constants from l_0 can be split once. For the experiments in Section 8, we constrain the expansion operator to work on event modifiers but the procedure generalizes to all predicates.

5.2.2 Constant Matching

To build an executable logical form y, all underspecified constants must be replaced with constants from $\mathcal{O}$. This is done through a sequence of constant replacement operations, each of which replaces a single underspecified constant with a constant of the same type from $\mathcal{O}$. Two example replacements are shown in Figure 3c. The output from the last replacement operation is a fully specified logical form.

6 Building and Scoring Derivations

This section introduces a dynamic program used to construct derivations and a linear scoring model.

6.1 Building Derivations

The space of derivations is too large to explicitly enumerate. However, each logical form (both final and interim) can be constructed with many different derivations, and we only need to find the highest scoring one. This allows us to develop a simple dynamic program for our two-stage semantic parser.

We use a CKY style chart parser to calculate the k-best logical forms output by parses of x. We then store each interim logical form generated by an operator in M once in a hyper-graph chart structure.

The branching factor of this hypergraph is polynomial in the number of constants in l_0 and linear in the size of $\mathcal{O}$. Subsequently, there are too many possible logical forms to enumerate explicitly; we prune as follows. We allow the top N scoring ontological matches for each original subexpression in l_0 and remove matches that differ from score from the maximum scoring match by more than a threshold τ. When building derivations, we apply constant matching operators as soon as they are applicable to new underspecified constants created by collapses and expansions. This allows the scoring function used by the pruning strategy to take advantage of all features defined in Section 7.2.

6.2 Ranking Derivations

Given feature vector ϕ and weight vector θ, the score of a derivation $d = \langle \Pi, M \rangle$ is a linear function that decomposes over the parse tree Π and the individual ontology-matching steps o.

$$\begin{aligned} \text{SCORE}(d) &= \phi(d)\theta \\ &= \phi(\Pi)\theta + \sum_{o \in M} \phi(o)\theta \end{aligned} \quad (1)$$

The function $\text{PARSE}(x, \mathcal{O})$ introduced as our goal in Section 2 returns the logical form associated with the highest scoring derivation of x:

$$\text{PARSE}(x, \mathcal{O}) = \arg \max_{d \in \text{GEN}(x, \mathcal{O})} (\text{SCORE}(d))$$

The features and learning algorithm used to estimate θ are defined in the next section.

7 Learning

This section describes an online learning algorithm for question-answering data, along with the domain-independent feature set.

7.1 Learning Model Parameters

Our learning algorithm estimates the parameters θ from a set $\{(x_i, a_i) : i = 1 \ldots n\}$ of questions x_i paired with answers a_i from the knowledge base $\mathcal{K}$. Each derivation d generated by the parser is associated with a fully specified logical form $y = \text{YIELD}(d)$ that can be executed in $\mathcal{K}$. A derivation d of x_i is correct if $\text{EXEC}(\text{YIELD}(d), \mathcal{K}) = a_i$. We use a perceptron to estimate a weight vector θ that support a separation of γ between correct and incorrect answers. Figure 5 presents the learning algorithm.

Input: Q/A pairs $\{(x_i, a_i) : i = 1 \ldots n\}$; Knowledge base $\mathcal{K}$; Ontology $\mathcal{O}$; Function $\text{GEN}(x, \mathcal{O})$ that computes derivations of x; Function $\text{YIELD}(d)$that returns logical form yield of derivation d; Function $\text{EXEC}(y, \mathcal{K})$ that calculates execution of y in $\mathcal{K}$; Margin γ; Number of iterations T.

Output: Linear model parameters θ.

Algorithm:

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

$C = \{d : d \in \text{GEN}(x_i, \mathcal{O}); \text{EXEC}(\text{YIELD}(d), \mathcal{K}) = a_i\}$

$W = \{d : d \in \text{GEN}(x_i, \mathcal{O}); \text{EXEC}(\text{YIELD}(d), \mathcal{K}) \neq a_i\}$

$C^* = \arg\max_{d \in C}(\phi(d)\theta)$

$W^* = \{d : d \in W;\ \exists c \in C^*\ s.t.\ \phi(c)\theta - \phi(d)\theta < \gamma)\}$

If $|C^*| > 0 \wedge |W^*| > 0$:

$\theta = \theta + \frac{1}{|C^*|} \sum_{c \in C^*} \phi(c) - \frac{1}{|W^*|} \sum_{e \in W^*} \phi(e)$

Figure 5: Parameter estimation from Q/A pairs.

7.2 Features

The feature vector $\phi(d)$ introduced in Section 6.2 decomposes over each of the derivation steps in d.

CCG Parse Features Each lexical item in Π has three indicator features. The first indicates the number of times each underspecified category is used. For example, the parse in Figure 3a uses the underspecified category $\mathsf{N} : \lambda x.p(x)$ twice. The second feature indicates (word, category) pairings — e.g. that $\mathsf{N} : \lambda x.p(x)$ is paired with 'library' and 'public' once each in Figure 3a. The final lexical feature indicates (part-of-speech, category) pairings for all parts of speech associated with the word.

Structural Features The structure matching operators (Section 5.2.1) in M generate new underspecified constants that define the types of constants in the output logical form y. These operators are scored using features that indicate the type of each complex-typed constant present in y and the identity of domain-independent functional constants in y. The logical form y generated in Figure 3 contains one complex typed constant with type $\langle i, \langle e, t \rangle \rangle$ and no domain-independent functional constants. Structural features allow the model to adapt to different knowledge bases $\mathcal{K}$. They allow it to determine, for example, whether a numeric quantity such as 'population' is likely to be explicitly listed in $\mathcal{K}$ or if it should be computed with the $count$ function.

Lexical Features Each constant replacement operator (Section 5.2.2) in M replaces an underspec-

ified constant c_u with a constant $c_{\mathcal{O}}$ from $\mathcal{O}$. The underspecified constant c_u is associated with the sequence of words $\vec{w}_u$ used in the CCG lexical entries that introduced it in Π. We assume that each of the constants c_O in $\mathcal{O}$ is associated with a string label $\vec{w}_O$. This allows us to introduce five domain-independent features that measure the similarity of $\vec{w}_u$ and $\vec{w}_O$.

The feature $\phi_{np}(c_u, c_O)$ signals the replacement of an entity-typed constant c_u with entity c_O that has label $\vec{w}_u$. For the second example in Figure 1 this feature indicates the replacement of the underspecified constant associated with the word 'mozart' with the Freebase entity `mozart`. Stem and synonymy features $\phi_{stem}(c_u, c_O)$ and $\phi_{syn}(c_u, c_O)$ signal the existence of words $w_u \in \vec{w_u}$ and $w_u \in \vec{w_O}$ that share a stem or synonym respectively. Stems are computed with the Porter stemmer and synonyms are extracted from Wiktionary. A single Freebase specific feature $\phi_{fp:stem}(c_u, c_O)$ indicates a word stem match between $w_u \in \vec{w_u}$ and the word filling the most specific position in $\vec{w_u}$ under Freebase's hierarchical naming schema.

A final feature $\phi_{gl}(c_u, c_O)$ calculates the overlap between Wiktionary definitions for $\vec{w_u}$ and $\vec{w_O}$. Let $\texttt{gl}(w)$ be the Wiktionary definition for w. Then:

$$\phi_{gl}(c_u, c_O) = \sum_{w_u \in \vec{w_u}; w_O \in \vec{w_O}} \frac{2 \cdot |\texttt{gl}(w_O) \cap \texttt{gl}(w_c)|}{|\vec{w_O}| \cdot |\vec{w_u}| \cdot |\texttt{gl}(w_O)| + |\texttt{gl}(w_c)|}$$

Domain-indepedent lexical features allow the model to reason about the meaning of unseen words. In small domains, however, the majority of word usages may be covered by training data. We make use of this fact in the GeoQuery domain with features $\phi_m(c_u, c_O)$ that indicate the pairing of $\vec{w_u}$ with c_O.

Knowledge Base Features Guided by the observation that we generally want to create queries y which have answers in knowledge base $\mathcal{K}$, we define features to signal whether each operation could build a logical form y with an answer in $\mathcal{K}$.

If a predicate-argument relation in y does not exist in $\mathcal{K}$, then the execution of y against $\mathcal{K}$ will not return an answer. Two features indicate whether predicate-argument relations in y exist in $\mathcal{K}$. $\phi_{direct}(y, \mathcal{K})$ indicates predicate-argument applications in y that exists in $\mathcal{K}$. For example, if the application of `dedicated_by` to `mozart` in Figure 1 exists in Freebase, $\phi_{direct}(y, \mathcal{K})$ will fire. $\phi_{join}(y, \mathcal{K})$ indicates entities separated from a predicate by one join in y, such as `mozart` and `dedicated_to` in Figure 1, that exist in the same relationship in $\mathcal{K}$.

If two predicates that share a variable in y do not share an argument in that position in $\mathcal{K}$ then the execution of y against $\mathcal{K}$ will fail. The predicate-predicate $\phi_{pp}(y, \mathcal{K})$ feature indicates pairs of predicates that share a variable in y but cannot occur in this relationship in $\mathcal{K}$. For example, since the subject of the Freebase property `date_of_birth` does not take arguments of type `location`, $\phi_{pp}(y, \mathcal{K})$ will fire if y contains the logical form $\lambda x \lambda y.\texttt{date\_of\_birth}(x, y) \wedge \texttt{location}(x)$.

Both the predicate-argument and predicate-predicate features operate on subexpressions of y. We also define the execution features: $\phi_{emp}(y, \mathcal{K})$ to signal an empty answer for y in $\mathcal{K}$; $\phi_0(y, \mathcal{K})$ to signal a zero-valued answer created by counting over an empty set; and $\phi_1(y, \mathcal{K})$ to signal a one-valued answer created by counting over a singleton set.

As with the lexical cues, we use knowledge base features as soft constraints since it is possible for natural language queries to refer to concepts that do not exist in $\mathcal{K}$.

8 Experimental Setup

Data We evaluate performance on the benchmark GeoQuery dataset (Zelle and Mooney, 1996), and a newly introduced Freebase Query (FQ) dataset (Cai and Yates, 2013a). FQ contains 917 questions labeled with logical form meaning representations for querying Freebase. We gathered question answer labels by executing the logical forms against Freebase, and manually correcting any inconsistencies.

Freebase (Bollacker et al., 2008) is a large, collaboratively authored database containing almost 40 million entities and two billion facts, covering more than 100 domains. We filter Freebase to cover the domains contained in the FQ dataset resulting in a database containing 18 million entities, 2072 relations, 635 types, 135 million facts and 81 domains, including for example film, sports, and business. We use this schema to define our target domain, allowing for a wider variety of queries than could be encoded with the 635 collapsed relations previously used to label the FQ data.

We report two different experiments on the FQ data: test results on the existing 642/275 train/test split and domain adaptation results where the data is split three ways, partitioning the topics so that the logical meaning expressions do not share any symbols across folds. We report on the standard 600/280 training/test split for GeoQuery.

Parameter Initialization and Training We initialize weights for ϕ_{np} and ϕ_{direct} to 10, and weights for ϕ_{stem} and ϕ_{join} to 5. This promotes the use of entities and relations named in sentences. We initialize weights for ϕ_{pp} and ϕ_{emp} to -1 to favour logical forms that have an interpretation in the knowledge base $\mathcal{K}$. All other feature weights are initialized to 0. We run the training algorithm for one iteration on the Freebase data, at which point performance on the development set had converged. This fast convergence is due to the very small number of matching parameters used (5 lexical features and 8 $\mathcal{K}$ features). For GeoQuery, we include the larger domain specific feature set introduced in Section 7.2 and train for 10 iterations. We set the pruning parameters from Section 6.1 as follows: $k = 5$ for Freebase, $k = 30$ for GeoQuery, $N = 50$, $\tau = 10$.

Comparison Systems We compare performance to state-of-the-art systems in both domains. On GeoQuery, we report results from DCS (Liang et al., 2011) without special initialization (DCS) and with an small hand-engineered lexicon (DCS with L^+). We also include results for the FUBL algorithm (Kwiatkowski et al., 2011), the CCG learning approach that is most closely related to our work. On FQ, we compare to Cai and Yates (2013a) (CY13).

Evaluation We evaluate by comparing the produced question answers to the labeled ones, with no partial credit. Because the parser can fail to produce a complete query, we report recall, the percent of total questions answered correctly, and precision, the percentage of produced queries with correct answers. CY13 and FUBL report fully correct logical forms, which is a close proxy to our numbers.

9 Results

Quantitative Analysis For FQ, we report results on the test set and in the cross-domain setting, as defined in Section 8. Figure 6 shows both results. Our approach outperforms the previous state of the art, achieving a nine point improvement in test recall, while not requiring labeled logical forms in training. We also see consistent improvements on both scenarios, indicating that our approach is generalizing well across topic domains. The learned ontology matching model is able to reason about previously unseen ontological subdomains as well as if it was provided explicit, in-domain training data.

Setting	System	R	P	F1
Test	Our Approach	68.0	76.7	72.1
	CY13	59	67	63
Cross Domain	Our Approach	67.9	73.5	71.5
	CY13	60	69	65

Figure 6: Results on the FQ dataset.

	R	P	F1
All Features	68.6	72.0	70.3
Without Wiktionary	67.2	70.7	68.9
Without $\mathcal{K}$ Features	61.8	62.5	62.1

Figure 7: Ablation Results

	Recall
FUBL	88.6
DCS	87.9
DCS with L^+	91.1
Our Approach	89.0

Figure 8: GeoQuery Results

We also performed feature ablations with 5-fold cross validation on the training set, as seen in Figure 7. Both the Wiktionary features and knowledge base features were helpful. Without the Wiktionary features, the model must rely on word stem matches which, in combination with graph constraints, can still recover many of the correct queries. However, without the knowledge base constraints, the model produces many queries that return empty answers, and significantly impacts overall performance.

For GeoQuery, we report test results in Figure 8. Our approach outperforms the most closely related CCG model (FUBL) and DCS without initialization, but falls short of DCS with a small hand-built initial lexicon. Given the small size of the test set, it is fair to say that all algorithms are performing at state-of-the-art levels. This result demonstrates that our approach can handle the high degree of lexical ambi-

Parse Failures (20%)

1.	Query	in what year did motorola have the most revenue
2	Query	on how many projects was james walker a design engineer

Structural Matching Failure (30%)

	Query	how many children does jerry seinfeld have
3.	Labeled	λx.eq(x, count(λy.people.person.children(jerry_seinfeld, y)))
	Predicted	λx.eq(x, count(λy.people.person.children(y, jerry_seinfeld)))

Incomplete Database (10%)

	Query	how many countries participated in the 2006 winter olympics
4.	Labeled	λy.olympics.olympic_games.number_of_countries(2006_winter_olympics, y)
	Predicted	λy.eq(y, count(λy.olympic_participation_country.olympics_participated_in(x, 2006_winter_olympics)))
	Query	what programming languages were used for aol instant messenger
5.	Labeled	λy.computer.software.languages_used(aol_instant_messenger, y)
	Predicted	λy.computer.software.languages_used(aol_instant_messenger, y) $\wedge$ computer.programming_language(y)

Lexical Ambiguity (35%)

	Query	when was the frida kahlo exhibit at the philadelphia art museum
	Labeled	λy.$\exists$x.exhibition_run.exhibition(x, frida_kahlo)$\wedge$
6.		exhibition_venue.exhibitions_at(philadelphia_art_museum, x) $\wedge$ exhibition_run.opened_on(x, y)
	Predicted	λy.$\exists$x.exhibition_run.exhibition(x, frida_kahlo)$\wedge$
		exhibition_venue.exhibitions_at(philadelphia_art_museum, x) $\wedge$ exhibition_run.closed_on(x, y)

Figure 9: Example error cases, with associated frequencies, illustrating system output and gold standard references. 5% of the cases were miscellaneous or otherwise difficult to categorize.

guity in the FQ data, without sacrificing the ability to understanding the rich, compositional phenomena that are common in the GeoQuery data.

Qualitative Analysis We also did a qualitative analysis of errors in the FQ domain. The model learns to correctly produce complex forms that join multiple relations. However, there are a number of systematic error cases, grouped into four categories as seen in Figure 9.

The first and second examples show parse failures, where the underspecified CCG grammar did not have sufficient coverage. The third shows a failed structural match, where all of the correct logical constants are selected, but the argument order is reversed for one of the literals. The fourth and fifth examples demonstrate a failures due to database incompleteness. In both cases, the predicted queries would have returned the same answers as the gold-truth ones if Freebase contained all of the required facts. Developing models that are robust to database incompleteness is a challenging problem for future work. Finally, the last example demonstrates a lexical ambiguity, where the system was unable to determine if the query should include the opening date or the closing date for the exhibit.

10 Conclusion

We considered the problem of learning domain-independent semantic parsers, with application to QA against large knowledge bases. We introduced a new approach for learning a two-stage semantic parser that enables scalable, on-the-fly ontological matching. Experiments demonstrated state-of-the-art performance on benchmark datasets, including effective generalization to previously unseen words.

We would like to investigate more nuanced notions of semantic correctness, for example to support many of the essentially equivalent meaning representations we found in the error analysis. Although we focused exclusively on QA applications, the general two-stage analysis approach should allow for the reuse of learned grammars across a number of different domains, including robotics or dialog applications, where data is more challenging to gather.

11 Acknowledgements

This research was supported in part by DARPA under the DEFT program through the AFRL (FA8750-13-2-0019) and the CSSG (N11AP20020), the ARO (W911NF-12-1-0197), the NSF (IIS-1115966), and by a gift from Google. The authors thank Anthony Fader, Nicholas FitzGerald, Adrienne Wang, Daniel Weld, and the anonymous reviewers for their helpful comments and feedback.

References

Alshawi, H. (1992). *The core language engine*. The MIT Press.

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

Artzi, Y. and Zettlemoyer, L. (2013a). UW SPF: The University of Washington Semantic Parsing Framework.

Artzi, Y. and Zettlemoyer, L. (2013b). Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1(1):49–62.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

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

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

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

Chen, D. and Mooney, R. (2011). Learning to interpret natural language navigation instructions from observations. In *Proceedings of the National Conference on Artificial Intelligence*.

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

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

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

Doan, A., Madhavan, J., Domingos, P., and Halevy, A. (2004). Ontology matching: A machine learning approach. In *Handbook on ontologies*. Springer.

Euzenat, J., Euzenat, J., Shvaiko, P., et al. (2007). *Ontology matching*. Springer.

Fader, A., Zettlemoyer, L., and Etzioni, O. (2013). Paraphrase-driven learning for open question answering. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

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

Grosz, B. J., Appelt, D. E., Martin, P. A., and Pereira, F. (1987). TEAM: An experiment in the design of transportable natural language interfaces. *Artificial Intelligence*, 32(2):173–243.

Hobbs, J. R. (1985). Ontological promiscuity. In *Proceedings of the Annual Meeting on Association for Computational Linguistics*.

Jones, B. K., Johnson, M., and Goldwater, S. (2012). Semantic parsing with bayesian tree transducers. In *Proceedings of the 50th Annual Meeting of the Association of Computational Linguistics*.

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

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

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

Kushman, N. and Barzilay, R. (2013). Using semantic unification to generate regular expressions from natural language. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*.

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

Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., and Steedman, M. (2010). Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., and Steedman, M. (2011). Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

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

Matuszek, C., FitzGerald, N., Zettlemoyer, L., Bo, L., and Fox, D. (2012). A joint model of language and perception for grounded attribute learning. In *Proceedings of the International Conference on Machine Learning*.

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

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

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

Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A., Gerber, D., and Cimiano, P. (2012). Template-based question answering over RDF data. In *Proceedings of the International Conference on World Wide Web*.

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

Yahya, M., Berberich, K., Elbassuoni, S., Ramanath, M., Tresp, V., and Weikum, G. (2012). Natural language questions for the web of data. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

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

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

Zettlemoyer, L. and 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*.

Zhang, C., Hoffmann, R., and Weld, D. S. (2012). Ontological smoothing for relation extraction with minimal supervision. In *Proceeds of the Conference on Artificial Intelligence*.