# Compositional Semantic Parsing on Semi-Structured Tables

**Panupong Pasupat**
Computer Science Department
Stanford University
ppasupat@cs.stanford.edu

**Percy Liang**
Computer Science Department
Stanford University
pliang@cs.stanford.edu

## Abstract

Two important aspects of semantic parsing for question answering are the breadth of the knowledge source and the depth of logical compositionality. While existing work trades off one aspect for another, this paper simultaneously makes progress on both fronts through a new task: answering complex questions on semi-structured tables using question-answer pairs as supervision. The central challenge arises from two compounding factors: the broader domain results in an open-ended set of relations, and the deeper compositionality results in a combinatorial explosion in the space of logical forms. We propose a logical-form driven parsing algorithm guided by strong typing constraints and show that it obtains significant improvements over natural baselines. For evaluation, we created a new dataset of 22,033 complex questions on Wikipedia tables, which is made publicly available.

## 1 Introduction

In semantic parsing for question answering, natural language questions are converted into logical forms, which can be executed on a knowledge source to obtain answer denotations. Early semantic parsing systems were trained to answer highly compositional questions, but the knowledge sources were limited to small closed-domain databases (Zelle and Mooney, 1996; Wong and Mooney, 2007; Zettlemoyer and Collins, 2007; Kwiatkowski et al., 2011). More recent work sacrifices compositionality in favor of using more open-ended knowledge bases such as Freebase (Cai and Yates, 2013; Berant et al., 2013; Fader et al., 2014; Reddy et al., 2014). However, even these broader knowledge sources still define a rigid schema over entities and relation types, thus restricting the scope of answerable questions.

| Year | City | Country | Nations |
|---|---|---|---|
| 1896 | Athens | Greece | 14 |
| 1900 | Paris | France | 24 |
| 1904 | St. Louis | USA | 12 |
| ... | ... | ... | ... |
| 2004 | Athens | Greece | 201 |
| 2008 | Beijing | China | 204 |
| 2012 | London | UK | 204 |

$x_1$: *"Greece held its last Summer Olympics in which year?"*
$y_1$: {2004}

$x_2$: *"In which city's the first time with at least 20 nations?"*
$y_2$: {Paris}

$x_3$: *"Which years have the most participating countries?"*
$y_3$: {2008, 2012}

$x_4$: *"How many events were in Athens, Greece?"*
$y_4$: {2}

$x_5$: *"How many more participants were there in 1900 than in the first year?"*
$y_5$: {10}

Figure 1: Our task is to answer a highly compositional question from an HTML table. We learn a semantic parser from question-table-answer triples $\{(x_i, t_i, y_i)\}$.

To simultaneously increase both the *breadth* of the knowledge source and the *depth* of logical compositionality, we propose a new task (with an associated dataset): answering a question using an HTML table as the knowledge source. Figure 1 shows several question-answer pairs and an accompanying table, which are typical of those in our dataset. Note that the questions are logically quite complex, involving a variety of operations such as comparison ($x_2$), superlatives ($x_3$), aggregation ($x_4$), and arithmetic ($x_5$).

The HTML tables are semi-structured and not normalized. For example, a cell might contain multiple parts (e.g., "*Beijing, China*" or "*200 km*"). Additionally, we mandate that the training and test tables are disjoint, so at test time, we will see relations (column headers; e.g., "*Nations*") and entities (table cells; e.g., "*St. Louis*")

that were not observed during training. This is in contrast to knowledge bases like Freebase, which have a global fixed relation schema with normalized entities and relations.

Our task setting produces two main challenges. Firstly, the increased breadth in the knowledge source requires us to generate logical forms from novel tables with previously unseen relations and entities. We therefore cannot follow the typical semantic parsing strategy of constructing or learning a lexicon that maps phrases to relations ahead of time. Secondly, the increased depth in compositionality and additional logical operations exacerbate the exponential growth of the number of possible logical forms.

We trained a semantic parser for this task from question-answer pairs based on the framework illustrated in Figure 2. First, relations and entities from the semi-structured HTML table are encoded in a graph. Then, the system parses the question into candidate logical forms with a high-coverage grammar, reranks the candidates with a log-linear model, and then executes the highest-scoring logical form to produce the answer denotation. We use beam search with pruning strategies based on type and denotation constraints to control the combinatorial explosion.

To evaluate the system, we created a new dataset, WIKITABLEQUESTIONS, consisting of 2,108 HTML tables from Wikipedia and 22,033 question-answer pairs. When tested on unseen tables, the system achieves an accuracy of 37.1%, which is significantly higher than the information retrieval baseline of 12.7% and a simple semantic parsing baseline of 24.3%.

## 2 Task

Our task is as follows: given a table $t$ and a question $x$ about the table, output a list of values $y$ that answers the question according to the table. Example inputs and outputs are shown in Figure 1. The system has access to a training set $\mathcal{D} = \{(x_i, t_i, y_i)\}_{i=1}^{N}$ of questions, tables, and answers, but the tables in test data do not appear during training.

The only restriction on the question $x$ is that a person must be able to answer it using just the table $t$. Other than that, the question can be of any type, ranging from a simple table lookup question to a more complicated one that involves various logical operations.

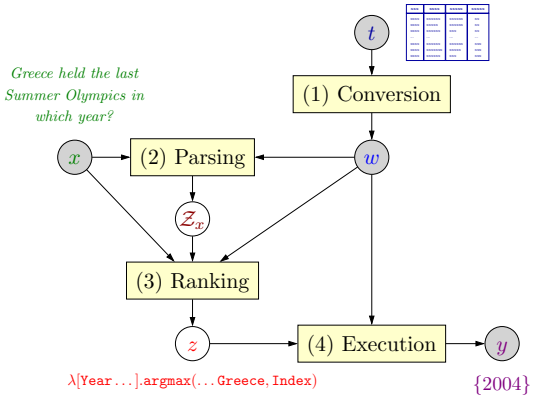


Figure 2: The prediction framework: (1) the table $t$ is deterministically converted into a knowledge graph $w$ as shown in Figure 3; (2) with information from $w$, the question $x$ is parsed into candidate logical forms in $\mathcal{Z}_x$; (3) the highest-scoring candidate $z \in \mathcal{Z}_x$ is chosen; and (4) $z$ is executed on $w$, yielding the answer $y$.

**Dataset.** We created a new dataset, WIKITABLEQUESTIONS, of question-answer pairs on HTML tables as follows. We randomly selected data tables from Wikipedia with at least 8 rows and 5 columns. We then created two Amazon Mechanical Turk tasks. The first task asks workers to write trivia questions about the table. For each question, we put one of the 36 generic prompts such as *"The question should require calculation"* or *"contains the word* 'first' *or its synonym"* to encourage more complex utterances. Next, we submit the resulting questions to the second task where the workers answer each question based on the given table. We only keep the answers that are agreed upon by at least two workers. After this filtering, approximately 69% of the questions remains.

The final dataset contains 22,033 examples on 2,108 tables. We set aside 20% of the tables and their associated questions as the test set and develop on the remaining examples. Simple preprocessing was done on the tables: We omit all non-textual contents of the tables, and if there is a merged cell spanning many rows or columns, we unmerge it and duplicate its content into each unmerged cell. Section 7.2 analyzes various aspects of the dataset and compares it to other datasets.

## 3 Approach

We now describe our semantic parsing framework for answering a given question and for training the model with question-answer pairs.

**Prediction.** Given a table $t$ and a question $x$, we predict an answer $y$ using the framework illustrated in Figure 2. We first convert the table $t$ into a *knowledge graph* $w$ ("world") which encodes different relations in the table (Section 4). Next, we generate a set of candidate logical forms $\mathcal{Z}_x$ by parsing the question $x$ using the information from $w$ (Section 6.1). Each generated logical form $z \in \mathcal{Z}_x$ is a graph query that can be executed on the knowledge graph $w$ to get a *denotation* $[\![z]\!]_w$. We extract a feature vector $\phi(x, w, z)$ for each $z \in \mathcal{Z}_x$ (Section 6.2) and define a log-linear distribution over the candidates:

$$p_\theta(z \mid x, w) \propto \exp\{\theta^\top \phi(x, w, z)\}, \quad (1)$$

where $\theta$ is the parameter vector. Finally, we choose the logical form $z$ with the highest model probability and execute it on $w$ to get the answer denotation $y = [\![z]\!]_w$.

**Training.** Given training examples $\mathcal{D} = \{(x_i, t_i, y_i)\}_{i=1}^N$, we seek a parameter vector $\theta$ that maximizes the regularized log-likelihood of the correct denotation $y_i$ marginalized over logical forms $z$. Formally, we maximize the objective function

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \log p_\theta(y_i \mid x_i, w_i) - \lambda \|\theta\|_1, \quad (2)$$

where $w_i$ is deterministically generated from $t_i$, and

$$p_\theta(y \mid x, w) = \sum_{z \in \mathcal{Z}_x; y = [\![z]\!]_w} p_\theta(z \mid x, w). \quad (3)$$

We optimize $\theta$ using AdaGrad (Duchi et al., 2010), running 3 passes over the data. We use $L_1$ regularization with $\lambda = 3 \times 10^{-5}$ obtained from cross-validation.

The following sections explain individual system components in more detail.

## 4 Knowledge graph

Inspired by the graph representation of knowledge bases, we preprocess the table $t$ by deterministically converting it into a *knowledge graph* $w$ as illustrated in Figure 3. In the most basic form, table rows become row nodes, strings in table cells become entity nodes,[1] and table columns become directed edges from the row nodes to the entity nodes of that column. The column headers are used as edge labels for these row-entity relations.

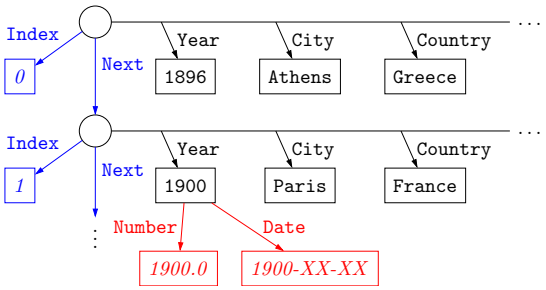


Figure 3: Part of the knowledge graph corresponding to the table in Figure 1. Circular nodes are row nodes. We augment the graph with different entity normalization nodes such as `Number` and `Date` (red) and additional row node relations `Next` and `Index` (blue).

The knowledge graph representation is convenient for three reasons. First, we can encode different forms of entity normalization in the graph. Some entity strings (e.g., *"1900"*) can be interpreted as a number, a date, or a proper name depending on the context, while some other strings (e.g., *"200 km"*) have multiple parts. Instead of committing to one normalization scheme, we introduce edges corresponding to different normalization methods from the entity nodes. For example, the node `1900` will have an edge called `Date` to another node *1900-XX-XX* of type date. Apart from type checking, these normalization nodes also aid learning by providing signals on the appropriate answer type. For instance, we can define a feature that associates the phrase *"how many"* with a logical form that says "traverse a row-entity edge, then a `Number` edge" instead of just "traverse a row-entity edge."

The second benefit of the graph representation is its ability to handle various logical phenomena via graph augmentation. For example, to answer questions of the form *"What is the next ...?"* or *"Who came before ...?"*, we augment each row node with an edge labeled `Next` pointing to the next row node, after which the questions can be answered by traversing the `Next` edge. In this work, we choose to add two special edges on each row node: the `Next` edge mentioned above and an `Index` edge pointing to the row index number $(0, 1, 2, \ldots)$.

Finally, with a graph representation, we can query it directly using a logical formalism for knowledge graphs, which we turn to next.

[1] Two occurrences of the same string constitute one node.

| Name | Example |
|---|---|
| Join | City.Athens<br>(row nodes with a City edge to Athens) |
| Union | City.(Athens ⊔ Beijing) |
| Intersection | City.Athens ⊓ Year.Number.<.*1990* |
| Reverse | **R**[Year].City.Athens<br>(entities where a row in City.Athens has a Year edge to) |
| Aggregation | count(City.Athens)<br>(the number of rows with city Athens) |
| Superlative | argmax(City.Athens, Index)<br>(the last row with city Athens) |
| Arithmetic | sub(*204*, *201*) (= 204 − 201) |
| Lambda | $\lambda x$[Year.Date.$x$]<br>(a binary: composition of two relations) |

Table 1: The lambda DCS operations we use.

| Rule | Semantics | Example |
|---|---|---|
| ***Anchored to the utterance*** | | |
| *TokenSpan* → *Entity* | match($z_1$) | Greece |
| (match($s$) = entity with name $s$) | | anchored to *"Greece"* |
| *TokenSpan* → *Atomic* | val($z_1$) | *2012-07-XX* |
| (val($s$) = interpreted value) | | anchored to *"July 2012"* |
| ***Unanchored (floating)*** | | |
| $\emptyset$ → *Relation* | $r$ | Country |
| ($r$ = row-entity relation) | | |
| $\emptyset$ → *Relation* | $\lambda x$[$r.p.x$] | $\lambda x$[Year.Date.$x$] |
| ($p$ = normalization relation) | | |
| $\emptyset$ → *Records* | Type.Row | (list of all rows) |
| $\emptyset$ → *RecordFn* | Index | (row ← row index) |

Table 2: Base deduction rules. Entities and atomic values (e.g., numbers, dates) are anchored to token spans, while other predicates are kept floating. ($a \leftarrow b$ represents a binary mapping $b$ to $a$.)

## 5 Logical forms

As our language for logical forms, we use lambda dependency-based compositional semantics (Liang, 2013), or lambda DCS, which we briefly describe here. Each lambda DCS logical form is either a *unary* (denoting a list of values) or a *binary* (denoting a list of pairs). The most basic unaries are singletons (e.g., China represents an entity node, and *30* represents a single number), while the most basic binaries are relations (e.g., City maps rows to city entities, Next maps rows to rows, and >= maps numbers to numbers). Logical forms can be combined into larger ones via various operations listed in Table 1. Each operation produces a unary except lambda abstraction: $\lambda x[f(x)]$ is a binary mapping $x$ to $f(x)$.

## 6 Parsing and ranking

Given the knowledge graph $w$, we now describe how to parse the utterance $x$ into a set of candidate logical forms $\mathcal{Z}_x$

### 6.1 Parsing algorithm

We propose a new *floating parser* which is more flexible than a standard chart parser. Both parsers recursively build up derivations and corresponding logical forms by repeatedly applying deduction rules, but the floating parser allows logical form predicates to be generated independently from the utterance.

**Chart parser.** We briefly review the CKY algorithm for chart parsing to introduce notation. Given an utterance with tokens $x_1, \ldots, x_n$, the CKY algorithm applies deduction rules of the following two kinds:

$$(\textit{TokenSpan}, i, j)[s] \rightarrow (c, i, j)[f(s)], \quad (4)$$

$$(c_1, i, k)[z_1] + (c_2, k+1, j)[z_2] \rightarrow (c, i, j)[f(z_1, z_2)]. \quad (5)$$

The first rule is a lexical rule that matches an utterance token span $x_i \cdots x_j$ (e.g., $s =$ *"New York"*) and produces a logical form (e.g., $f(s) =$ NewYorkCity) with category $c$ (e.g., Entity). The second rule takes two adjacent spans giving rise to logical forms $z_1$ and $z_2$ and builds a new logical form $f(z_1, z_2)$. Algorithmically, CKY stores derivations of category $c$ covering the span $x_i \cdots x_j$ in a *cell* $(c, i, j)$. CKY fills in the cells of increasing span lengths, and the logical forms in the top cell $(\textit{ROOT}, 1, n)$ are returned.

**Floating parser.** Chart parsing uses lexical rules (4) to generate relevant logical predicates, but in our setting of semantic parsing on tables, we do not have the luxury of starting with or inducing a full-fledged lexicon. Moreover, there is a mismatch between words in the utterance and predicates in the logical form. For instance, consider the question *"Greece held its last Summer Olympics in which year?"* on the table in Figure 1 and the correct logical form **R**[$\lambda x$[Year.Date.$x$]].argmax(Country.Greece, Index). While the entity Greece can be anchored to the token *"Greece"*, some logical predicates (e.g., Country) cannot be clearly anchored to a token span. We could potentially learn to anchor the logical form Country.Greece to *"Greece"*, but if the relation Country is not seen during training, such a mapping is impossible to learn from the training data. Similarly, some prominent tokens

| Rule | Semantics | Example |
|---|---|---|
| | ***Join + Aggregate*** | |
| *Entity* or *Atomic* → *Values* | $z_1$ | `China` |
| *Atomic* → *Values* | $c.z_1$ | `>=`*.30* (at least 30) |
| ($c \in \{$`<`, `>`, `<=`, `>=`$\}$) | | |
| *Relation* + *Values* → *Records* | $z_1.z_2$ | `Country.China` (events (rows) where the country is China) |
| *Relation* + *Records* → *Values* | $\mathbf{R}[z_1].z_2$ | $\mathbf{R}$[`Year`]`.Country.China` (years of events in China) |
| *Records* → *Records* | `Next`$.z_1$ | `Next.Country.China` (... before China) |
| *Records* → *Records* | $\mathbf{R}$[`Next`]$.z_1$ | $\mathbf{R}$[`Next`]`.Country.China` (... after China) |
| *Values* → *Atomic* | $a(z_1)$ | `count(Country.China)` (How often did China ...) |
| ($a \in \{$`count`, `max`, `min`, `sum`, `avg`$\}$) | | |
| *Values* → *ROOT* | $z_1$ | |
| | ***Superlative*** | |
| *Relation* → *RecordFn* | $z_1$ | $\lambda x$[`Nations.Number`$.x$] (row ← value in Nations column) |
| *Records* + *RecordFn* → *Records* | $s(z_1, z_2)$ | `argmax(Type.Row,` $\lambda x$[`Nations.Number`$.x$]`)` |
| ($s \in \{$`argmax`, `argmin`$\}$) | | (events with the most participating nations) |
| | | `argmin(City.Athens, Index)` (first event in Athens) |
| *Relation* → *ValueFn* | $\mathbf{R}[\lambda x[a(z_1.x)]]$ | $\mathbf{R}$[$\lambda x$[`count(City.`$x$`)`]] (city ← num. of rows with that city) |
| *Relation* + *Relation* → *ValueFn* | $\lambda x[\mathbf{R}[z_1].z_2.x]$ | $\lambda x$[$\mathbf{R}$[`City`]`.Nations.Number.`$x$] |
| | | (city ← value in Nations column) |
| *Values* + *ValueFn* → *Values* | $s(z_1, z_2)$ | `argmax(...,` $\mathbf{R}$[$\lambda x$[`count(City.`$x$`)`]]`)` (most frequent city) |
| | ***Other operations*** | |
| *ValueFn* + *Values* + *Values* → *Values* | $o(\mathbf{R}[z_1].z_2, \mathbf{R}[z_1].z_3)$ | `sub(`$\mathbf{R}$[`Number`]`.`$\mathbf{R}$[`Nations`]`.City.London, ...)` |
| ($o \in \{$`add`, `sub`, `mul`, `div`$\}$) | | (How many more participants were in London than ...) |
| *Entity* + *Entity* → *Values* | $z_1 \sqcup z_2$ | `China` $\sqcup$ `France` (China or France) |
| *Records* + *Records* → *Records* | $z_1 \sqcap z_2$ | `City.Beijing` $\sqcap$ `Country.China` (... in Beijing, China) |

Table 3: Compositional deduction rules. Each rule $c_1, \ldots, c_k \rightarrow c$ takes logical forms $z_1, \ldots, z_k$ constructed over categories $c_1, \ldots, c_k$, respectively, and produces a logical form based on the semantics.

(e.g., *"Olympics"*) are irrelevant and have no predicates anchored to them.

Therefore, instead of anchoring each predicate in the logical form to tokens in the utterance via lexical rules, we propose parsing more freely. We replace the anchored cells $(c, i, j)$ with *floating cells* $(c, s)$ of category $c$ and logical form size $s$. Then we apply rules of the following three kinds:

$$(\textit{TokenSpan}, i, j)[s] \rightarrow (c, 1)[f(s)], \quad (6)$$

$$\emptyset \rightarrow (c, 1)[f()], \quad (7)$$

$$(c_1, s_1)[z_1] + (c_2, s_2)[z_2] \rightarrow (c, s_1 + s_2 + 1)[f(z_1, z_2)]. \quad (8)$$

Note that rules (6) are similar to (4) in chart parsing except that the floating cell $(c, 1)$ only keeps track of the category and its size 1, not the span $(i, j)$. Rules (7) allow us to construct predicates out of thin air. For example, we can construct a logical form representing a table relation `Country` in cell $(\textit{Relation}, 1)$ using the rule $\emptyset \rightarrow \textit{Relation}\,[\texttt{Country}]$ independent of the utterance. Rules (8) perform composition, where the induction is on the size $s$ of the logical form rather than the span length. The algorithm stops when the specified maximum size is reached, after which the logical forms in cells $(\textit{ROOT}, s)$ for any $s$ are included in $\mathcal{Z}_x$. Figure 4 shows an example derivation generated by our floating parser.


Figure 4: A derivation for the utterance *"Greece held its last Summer Olympics in which year?"* Only `Greece` is anchored to a phrase *"Greece"*; `Year` and other predicates are floating.

The floating parser is very flexible: it can skip tokens and combine logical forms in any order. This flexibility might seem too unconstrained, but we can use strong typing constraints to prevent nonsensical derivations from being constructed.

Tables 2 and 3 show the full set of deduction rules we use. We assume that all named entities will explicitly appear in the question $x$, so we an-

*"Greece held its last Summer Olympics in which year?"*
$z = \mathbf{R}[\lambda x[\texttt{Year.Number}.x]].\texttt{argmax}(\texttt{Type.Row}, \texttt{Index})$
$y = \{2012\}$ (type: NUM, column: YEAR)

| Feature Name | Note |
|---|---|
| (*"last"*, predicate = `argmax`) | lex |
| phrase = predicate | unlex (·, *"year"* = `Year`) |
| missing entity | unlex (·, missing *Greece*) |
| denotation type = NUM<br>denotation column = YEAR | |
| (*"which year"*, type = NUM) | lex |
| phrase = column | unlex (·, *"year"* = YEAR) |
| ($Q$ = *"which"*, type = NUM) | lex |
| ($H$ = *"year"*, type = NUM) | lex |
| $H$ = column | unlex (·, *"year"* = YEAR) |

Table 4: Example features that fire for the (incorrect) logical form $z$. All features are binary. (lex = lexicalized)

chor all entity predicates (e.g., `Greece`) to token spans (e.g., *"Greece"*). We also anchor all numerical values (numbers, dates, percentages, etc.) detected by an NER system. In contrast, relations (e.g., `Country`) and operations (e.g., `argmax`) are kept floating since we want to learn how they are expressed in language. Connections between phrases in $x$ and the generated relations and operations in $z$ are established in the ranking model through features.

### 6.2 Features

We define features $\phi(x, w, z)$ for our log-linear model to capture the relationship between the question $x$ and the candidate $z$. Table 4 shows some example features from each feature type. Most features are of the form $(f(x), g(z))$ or $(f(x), h(y))$ where $y = [\![z]\!]_w$ is the denotation, and $f$, $g$, and $h$ extract some information (e.g., identity, POS tags) from $x$, $z$, or $y$, respectively.

**phrase-predicate:** Conjunctions between n-grams $f(x)$ from $x$ and predicates $g(z)$ from $z$. We use both lexicalized features, where all possible pairs $(f(x), g(z))$ form distinct features, and binary unlexicalized features indicating whether $f(x)$ and $g(z)$ have a string match.

**missing-predicate:** Indicators on whether there are entities or relations mentioned in $x$ but not in $z$. These features are unlexicalized.

**denotation:** Size and type of the denotation $y = [\![x]\!]_w$. The type can be either a primitive type (e.g., NUM, DATE, ENTITY) or the name of the column containing the entity in $y$ (e.g., CITY).

**phrase-denotation:** Conjunctions between n-grams from $x$ and the types of $y$. Similar to the phrase-predicate features, we use both lexicalized and unlexicalized features.

**headword-denotation:** Conjunctions between the question word $Q$ (e.g., *what*, *who*, *how many*) or the headword $H$ (the first noun after the question word) with the types of $y$.

### 6.3 Generation and pruning

Due to their recursive nature, the rules allow us to generate highly compositional logical forms. However, the compositionality comes at the cost of generating exponentially many logical forms, most of which are redundant (e.g., logical forms with an `argmax` operation on a set of size 1). We employ several methods to deal with this combinatorial explosion:

**Beam search.** We compute the model probability of each partial logical form based on available features (i.e., features that do not depend on the final denotation) and keep only the $K = 200$ highest-scoring logical forms in each cell.

**Pruning.** We prune partial logical forms that lead to invalid or redundant final logical forms. For example, we eliminate any logical form that does not type check (e.g., `Beijing` $\sqcup$ `Greece`), executes to an empty list (e.g., `Year.Number.`*`24`*), includes an aggregate or superlative on a singleton set (e.g., `argmax(Year.Number.`*`2012`*`, Index)`), or joins two relations that are the reverses of each other (e.g., $\mathbf{R}$`[City].City.Beijing`).

## 7 Experiments

### 7.1 Main evaluation

We evaluate the system on the development sets (three random 80:20 splits of the training data) and the test data. In both settings, the tables we test on do not appear during training.

**Evaluation metrics.** Our main metric is *accuracy*, which is the number of examples $(x, t, y)$ on which the system outputs the correct answer $y$. We also report the *oracle* score, which counts the number of examples where at least one generated candidate $z \in \mathcal{Z}_x$ executes to $y$.

**Baselines.** We compare the system to two baselines. The first baseline (IR), which simulates information retrieval, selects an answer $y$ among the entities in the table using a log-linear model over entities (table cells) rather than logical forms. The features are conjunctions between phrases in $x$ and properties of the answers $y$, which cover all features in our main system that do not involve the logical form. As an upper bound of this baseline,

| | dev | | test | |
|---|---|---|---|---|
| | **acc** | **ora** | **acc** | **ora** |
| IR baseline | 13.4 | 69.1 | 12.7 | 70.6 |
| WQ baseline | 23.6 | 34.4 | 24.3 | 35.6 |
| Our system | 37.0 | 76.7 | 37.1 | 76.6 |

Table 5: Accuracy (acc) and oracle scores (ora) on the development sets (3 random splits of the training data) and the test data.

| | **acc** | **ora** |
|---|---|---|
| **Our system** | 37.0 | 76.7 |
| (a) **Rule Ablation** | | |
| join only | 10.6 | 15.7 |
| join + count (= WQ baseline) | 23.6 | 34.4 |
| join + count + superlative | 30.7 | 68.6 |
| all − {⊓, ⊔} | 34.8 | 75.1 |
| (b) **Feature Ablation** | | |
| all − features involving predicate | 11.8 | 74.5 |
| all − phrase-predicate | 16.9 | 74.5 |
| all − lex phrase-predicate | 17.6 | 75.9 |
| all − unlex phrase-predicate | 34.3 | 76.7 |
| all − missing-predicate | 35.9 | 76.7 |
| all − features involving denotation | 33.5 | 76.8 |
| all − denotation | 34.3 | 76.6 |
| all − phrase-denotation | 35.7 | 76.8 |
| all − headword-denotation | 36.0 | 76.7 |
| (c) **Anchor operations to trigger words** | 37.1 | 59.4 |

Table 6: Average accuracy and oracle scores on development data in various system settings.

69.1% of the development examples have the answer appearing as an entity in the table.

In the second baseline (WQ), we only allow deduction rules that produce join and count logical forms. This rule subset has the same logical coverage as Berant and Liang (2014), which is designed to handle the WEBQUESTIONS (Berant et al., 2013) and FREE917 (Cai and Yates, 2013) datasets.

**Results.** Table 5 shows the results compared to the baselines. Our system gets an accuracy of 37.1% on the test data, which is significantly higher than both baselines, while the oracle is 76.6%. The next subsections analyze the system components in more detail.

### 7.2 Dataset statistics

In this section, we analyze the breadth and depth of the WIKITABLEQUESTIONS dataset, and how the system handles them.

**Number of relations.** With 3,929 unique column headers (relations) among 13,396 columns, the tables in the WIKITABLEQUESTIONS dataset contain many more relations than closed-domain datasets such as Geoquery (Zelle and Mooney, 1996) and ATIS (Price, 1990). Additionally, the logical forms that execute to the correct denotations refer to a total of 2,056 unique column headers, which is greater than the number of relations in the FREE917 dataset (635 Freebase relations).

| **Operation** | **Amount** |
|---|---|
| join (table lookup) | 13.5% |
| + join with `Next` | + 5.5% |
| + aggregate (`count`, `sum`, `max`, ...) | + 15.0% |
| + superlative (`argmax`, `argmin`) | + 24.5% |
| + arithmetic, ⊓, ⊔ | + 20.5% |
| + other phenomena | + 21.0% |

Table 7: The logical operations required to answer the questions in 200 random examples.

**Knowledge coverage.** We sampled 50 examples from the dataset and tried to answer them manually using Freebase. Even though Freebase contains some information extracted from Wikipedia, we can answer only 20% of the questions, indicating that WIKITABLEQUESTIONS contains a broad set of facts beyond Freebase.

**Logical operation coverage.** The dataset covers a wide range of question types and logical operations. Table 6(a) shows the drop in oracle scores when different subsets of rules are used to generate candidates logical forms. The *join only* subset corresponds to simple table lookup, while *join + count* is the WQ baseline for Freebase question answering on the WEBQUESTIONS dataset. Finally, *join + count + superlative* roughly corresponds to the coverage of the Geoquery dataset.

To better understand the distribution of logical operations in the WIKITABLEQUESTIONS dataset, we manually classified 200 examples based on the types of operations required to answer the question. The statistics in Table 7 shows that while a few questions only require simple operations such as table lookup, the majority of the questions demands more advanced operations. Additionally, 21% of the examples cannot be answered using any logical form generated from the current deduction rules; these examples are discussed in Section 7.4.

**Compositionality.** From each example, we compute the logical form size (number of rules applied) of the highest-scoring candidate that executes to the correct denotation. The histogram in Figure 5 shows that a significant number of logical forms are non-trivial.

**Beam size and pruning.** Figure 6 shows the results with and without pruning on various beam

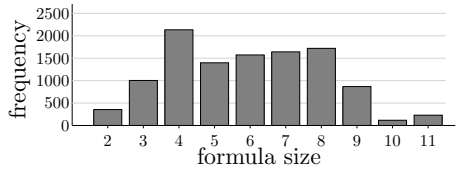


Figure 5: Sizes of the highest-scoring correct candidate logical forms in development examples.

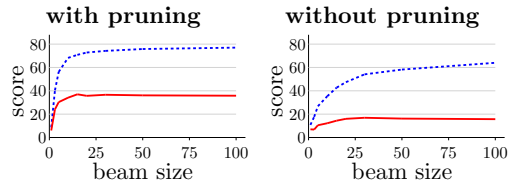


Figure 6: Accuracy (solid red) and oracle (dashed blue) scores with different beam sizes.

sizes. Apart from saving time, pruning also prevents bad logical forms from clogging up the beam which hurts both oracle and accuracy metrics.

### 7.3 Features

**Effect of features.** Table 6(b) shows the accuracy when some feature types are ablated. The most influential features are lexicalized phrase-predicate features, which capture the relationship between phrases and logical operations (e.g., relating *"last"* to `argmax`) as well as between phrases and relations (e.g., relating *"before"* to `<` or `Next`, and relating *"who"* to the relation `Name`).

**Anchoring with trigger words.** In our parsing algorithm, relations and logical operations are not anchored to the utterance. We consider an alternative approach where logical operations are anchored to "trigger" phrases, which are hand-coded based on co-occurrence statistics (e.g., we trigger a `count` logical form with *how*, *many*, and *total*).

Table 6(c) shows that the trigger words do not significantly impact the accuracy, suggesting that the original system is already able to learn the relationship between phrases and operations even without a manual lexicon. As an aside, the huge drop in oracle is because fewer "semantically incorrect" logical forms are generated; we discuss this phenomenon in the next subsection.

### 7.4 Semantically correct logical forms

In our setting, we face a new challenge that arises from learning with denotations: with deeper compositionality, a larger number of nonsensical logical forms can execute to the correct denotation. For example, if the target answer is a small number (say, 2), it is possible to count the number of rows with some random properties and arrive at the correct answer. However, as the system encounters more examples, it can potentially learn to disfavor them by recognizing the characteristics of semantically correct logical forms.

**Generating semantically correct logical forms.** The system can learn the features of semantically correct logical forms only if it can generate them in the first place. To see how well the system can generate correct logical forms, looking at the oracle score is insufficient since bad logical forms can execute to the correct denotations. Instead, we randomly chose 200 examples and manually annotated them with logical forms to see if a trained system can produce the annotated logical form as a candidate.

Out of 200 examples, we find that 79% can be manually annotated. The remaining ones include artifacts such as unhandled question types (e.g., yes-no questions, or questions with phrases *"same"* or *"consecutive"*), table cells that require advanced normalization methods (e.g., cells with comma-separated lists), and incorrect annotations.

The system generates the annotated logical form among the candidates in 53.5% of the examples. The missing examples are mostly caused by anchoring errors due to lexical mismatch (e.g., *"Italian"* $\rightarrow$ `Italy`, or *"no zip code"* $\rightarrow$ an empty cell in the zip code column) or the need to generate complex logical forms from a single phrase (e.g., *"May 2010"* $\rightarrow$ `>=`*.2010-05-01*$\sqcap$`<=`*.2010-05-31*).

### 7.5 Error analysis

The errors on the development data can be divided into four groups. The first two groups are unhandled question types (21%) and the failure to anchor entities (25%) as described in Section 7.4. The third group is normalization and type errors (29%): although we handle some forms of entity normalization, we observe many unhandled string formats such as times (e.g., *3:45.79*) and city-country pairs (e.g., *Beijing, China*), as well as complex calculation such as computing time periods (e.g., *12pm–1am* $\rightarrow$ 1 hour). Finally, we have ranking errors (25%) which mostly occur when the utterance phrase and the relation are obliquely related (e.g., *"airplane"* and `Model`).

## 8 Discussion

Our work simultaneously increases the breadth of knowledge source and the depth of compositionality in semantic parsing. This section explores the connections in both aspects to related work.

**Logical coverage.** Different semantic parsing systems are designed to handle different sets of logical operations and degrees of compositionality. For example, form-filling systems (Wang et al., 2011) usually cover a smaller scope of operations and compositionality, while early statistical semantic parsers for question answering (Wong and Mooney, 2007; Zettlemoyer and Collins, 2007) and high-accuracy natural language interfaces for databases (Androutsopoulos et al., 1995; Popescu et al., 2003) target more compositional utterances with a wide range of logical operations. This work aims to increase the logical coverage even further. For example, compared to the Geoquery dataset, the WIKITABLEQUESTIONS dataset includes a move diverse set of logical operations, and while it does not have extremely compositional questions like in Geoquery (e.g., *"What states border states that border states that border Florida?"*), our dataset contains fairly compositional questions on average.

To parse a compositional utterance, many works rely on a lexicon that translates phrases to entities, relations, and logical operations. A lexicon can be automatically generated (Unger and Cimiano, 2011; Unger et al., 2012), learned from data (Zettlemoyer and Collins, 2007; Kwiatkowski et al., 2011), or extracted from external sources (Cai and Yates, 2013; Berant et al., 2013), but requires some techniques to generalize to unseen data. Our work takes a different approach similar to the logical form growing algorithm in Berant and Liang (2014) by not anchoring relations and operations to the utterance.

**Knowledge domain.** Recent works on semantic parsing for question answering operate on more open and diverse data domains. In particular, large-scale knowledge bases have gained popularity in the semantic parsing community (Cai and Yates, 2013; Berant et al., 2013; Fader et al., 2014). The increasing number of relations and entities motivates new resources and techniques for improving the accuracy, including the use of ontology matching models (Kwiatkowski et al., 2013), paraphrase models (Fader et al., 2013; Berant and Liang, 2014), and unlabeled sentences (Krishnamurthy and Kollar, 2013; Reddy et al., 2014).

Our work leverages open-ended data from the Web through semi-structured tables. There have been several studies on analyzing or inferring the table schemas (Cafarella et al., 2008; Venetis et al., 2011; Syed et al., 2010; Limaye et al., 2010) and answering search queries by joining tables on similar columns (Cafarella et al., 2008; Gonzalez et al., 2010; Pimplikar and Sarawagi, 2012). While the latter is similar to question answering, the queries tend to be keyword lists instead of natural language sentences. In parallel, open information extraction (Wu and Weld, 2010; Masaum et al., 2012) and knowledge base population (Ji and Grishman, 2011) extract information from web pages and compile them into structured data. The resulting knowledge base is systematically organized, but as a trade-off, some knowledge is inevitably lost during extraction and the information is forced to conform to a specific schema. To avoid these issues, we choose to work on HTML tables directly.

In future work, we wish to draw information from other semi-structured formats such as colon-delimited pairs (Wong et al., 2009), bulleted lists (Gupta and Sarawagi, 2009), and top-$k$ lists (Zhang et al., 2013). Pasupat and Liang (2014) used a framework similar to ours to extract entities from web pages, where the "logical forms" were XPath expressions. A natural direction is to combine the logical compositionality of this work with the even broader knowledge source of general web pages.

**Acknowledgements.** We gratefully acknowledge the support of the Google Natural Language Understanding Focused Program and the Defense Advanced Research Projects Agency (DARPA) Deep Exploration and Filtering of Text (DEFT) Program under Air Force Research Laboratory (AFRL) contract no. FA8750-13-2-0040.

**Data and reproducibility.** The WIKITABLEQUESTIONS dataset can be downloaded at `http://nlp.stanford.edu/software/sempre/wikitable/`. Additionally, code, data, and experiments for this paper are available on the CodaLab platform at `https://www.codalab.org/worksheets/0xf26cd79d4d734287868923ad1067cf4c/`.

## References

I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. 1995. Natural language interfaces to databases –

an introduction. *Journal of Natural Language Engineering*, 1:29–81.

J. Berant and P. Liang. 2014. Semantic parsing via paraphrasing. In *Association for Computational Linguistics (ACL)*.

J. Berant, A. Chou, R. Frostig, and P. Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Empirical Methods in Natural Language Processing (EMNLP)*.

M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. 2008. WebTables: exploring the power of tables on the web. In *Very Large Data Bases (VLDB)*, pages 538–549.

Q. Cai and A. Yates. 2013. Large-scale semantic parsing via schema matching and lexicon extension. In *Association for Computational Linguistics (ACL)*.

J. Duchi, E. Hazan, and Y. Singer. 2010. Adaptive subgradient methods for online learning and stochastic optimization. In *Conference on Learning Theory (COLT)*.

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

A. Fader, L. Zettlemoyer, and O. Etzioni. 2014. Open question answering over curated and extracted knowledge bases. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1156–1165.

H. Gonzalez, A. Y. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, W. Shen, and J. Goldberg-Kidon. 2010. Google fusion tables: web-centered data management and collaboration. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1061–1066.

R. Gupta and S. Sarawagi. 2009. Answering table augmentation queries from unstructured lists on the web. In *Very Large Data Bases (VLDB)*, number 1, pages 289–300.

H. Ji and R. Grishman. 2011. Knowledge base population: Successful approaches and challenges. In *Association for Computational Linguistics (ACL)*, pages 1148–1158.

J. Krishnamurthy and T. Kollar. 2013. Jointly learning to parse and perceive: Connecting natural language to the physical world. *Transactions of the Association for Computational Linguistics (TACL)*, 1:193–206.

T. Kwiatkowski, L. Zettlemoyer, S. Goldwater, and M. Steedman. 2011. Lexical generalization in CCG grammar induction for semantic parsing. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1512–1523.

T. Kwiatkowski, E. Choi, Y. Artzi, and L. Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Empirical Methods in Natural Language Processing (EMNLP)*.

P. Liang. 2013. Lambda dependency-based compositional semantics. *arXiv*.

G. Limaye, S. Sarawagi, and S. Chakrabarti. 2010. Annotating and searching web tables using entities, types and relationships. In *Very Large Data Bases (VLDB)*, volume 3, pages 1338–1347.

Masaum, M. Schmitz, R. Bart, S. Soderland, and O. Etzioni. 2012. Open language learning for information extraction. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL)*, pages 523–534.

P. Pasupat and P. Liang. 2014. Zero-shot entity extraction from web pages. In *Association for Computational Linguistics (ACL)*.

R. Pimplikar and S. Sarawagi. 2012. Answering table queries on the web using column keywords. In *Very Large Data Bases (VLDB)*, volume 5, pages 908–919.

A. Popescu, O. Etzioni, and H. Kautz. 2003. Towards a theory of natural language interfaces to databases. In *International Conference on Intelligent User Interfaces (IUI)*, pages 149–157.

P. Price. 1990. Evaluation of spoken language systems: The ATIS domain. In *Proceedings of the Third DARPA Speech and Natural Language Workshop*, pages 91–95.

S. Reddy, M. Lapata, and M. Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics (TACL)*, 2(10):377–392.

Z. Syed, T. Finin, V. Mulwad, and A. Joshi. 2010. Exploiting a web of semantic data for interpreting tables. In *Proceedings of the Second Web Science Conference*.

C. Unger and P. Cimiano. 2011. Pythia: compositional meaning construction for ontology-based question answering on the semantic web. In *Proceedings of the 16th international conference on Natural language processing and information systems*, pages 153–160.

C. Unger, L. Bühmann, J. Lehmann, A. Ngonga, D. Gerber, and P. Cimiano. 2012. Template-based question answering over RDF data. In *World Wide Web (WWW)*, pages 639–648.

P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu. 2011. Recovering semantics of tables on the web. In *Very Large Data Bases (VLDB)*, volume 4, pages 528–538.

Y. Wang, L. Deng, and A. Acero. 2011. Semantic frame-based spoken language understanding. *Spoken Language Understanding: Systems for Extracting Semantic Information from Speech*, pages 41–91.

Y. W. Wong and R. J. Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Association for Computational Linguistics (ACL)*, pages 960–967.

Y. W. Wong, D. Widdows, T. Lokovic, and K. Nigam. 2009. Scalable attribute-value extraction from semi-structured text. In *IEEE International Conference on Data Mining Workshops*, pages 302–307.

F. Wu and D. S. Weld. 2010. Open information extraction using Wikipedia. In *Association for Computational Linguistics (ACL)*, pages 118–127.

M. Zelle and R. J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1050–1055.

L. S. Zettlemoyer and M. Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL)*, pages 678–687.

Z. Zhang, K. Q. Zhu, H. Wang, and H. Li. 2013. Automatic extraction of top-k lists from the web. In *International Conference on Data Engineering*.