Inferring Logical Forms From Denotations

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

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

Abstract

A core problem in learning semantic parsers from denotations is picking out consistent logical forms—those that yield the correct denotation—from a combinatorially large space. To control the search space, previous work relied on restricted set of rules, which limits expressivity. In this paper, we consider a much more expressive class of logical forms, and show how to use dynamic programming to efficiently represent the complete set of consistent logical forms. Expressivity also introduces many more spurious logical forms which are consistent with the correct denotation but do not represent the meaning of the utterance. To address this, we generate fictitious worlds and use crowdsourced denotations on these worlds to filter out spurious logical forms. On the WIKITABLEQUESTIONS dataset, we increase the coverage of answerable questions from 53.5% to 76%, and the additional crowdsourced supervision lets us rule out 92.1% of spurious logical forms.

1 Introduction

Consider the task of learning to answer complex natural language questions (e.g., "*Where did the last 1st place finish occur?*") using only question-answer pairs as supervision (Clarke et al., 2010; Liang et al., 2011; Berant et al., 2013; Artzi and Zettlemoyer, 2013). Semantic parsers map the question into a logical form (e.g., $\mathbf{R}[\texttt{Venue}].\texttt{argmax}(\texttt{Position.1st}, \texttt{Index})$) that can be executed on a knowledge source to obtain the answer (denotation). Logical forms are very expressive since they can be recursively composed, but this very expressivity makes it more difficult to search over the space of logical forms. Previous work sidesteps this obstacle by restricting the set of possible logical form compositions, but this is limiting. For instance, for the system in Pasupat and Liang (2015), in only 53.5% of the examples was the correct logical form even in the set of generated logical forms.

The goal of this paper is to solve two main challenges that prevent us from generating more expressive logical forms. The first challenge is computational: the number of logical forms grows exponentially as their size increases. Directly enumerating over all logical forms becomes infeasible, and pruning techniques such as beam search can inadvertently prune out correct logical forms.

The second challenge is the large increase in *spurious* logical forms—those that do not reflect the semantics of the question but coincidentally execute to the correct denotation. For example, while logical forms $z_1, \ldots, z_5$ in Figure 1 are all *consistent* (they execute to the correct answer y), the logical forms z_4 and z_5 are spurious and would give incorrect answers if the table were to change.

We address these two challenges by solving two interconnected tasks. The first task, which addresses the computational challenge, is to enumerate the set Z of all consistent logical forms given a question x, a knowledge source w ("world"), and the target denotation y (Section 4). Observing that the space of possible denotations grows much more slowly than the space of logical forms, we perform *dynamic programming on denotations* (DPD) to make search feasible. Our method is guaranteed to find all consistent logical forms up to some bounded size.

Given the set Z of consistent logical forms, the second task is to filter out spurious logical forms from Z (Section 5). Using the property that spurious logical forms ultimately give a wrong answer when the data in the world w changes, we create

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 23–32,
Berlin, Germany, August 7-12, 2016.

Year	Venue	Position	Event	Time
2001	Hungary	2nd	400m	47.12
2003	Finland	1st	400m	46.69
2005	Germany	11th	400m	46.62
2007	Thailand	1st	relay	182.05
2008	China	7th	relay	180.32

x: "*Where did the last 1st place finish occur?*"
y: Thailand

Consistent

Correct

z_1: $\mathbf{R}$[Venue].argmax(Position.1st, Index)
Among rows with Position = 1st, pick the one with maximum index, then return the Venue of that row.

z_2: $\mathbf{R}$[Venue].Index.max($\mathbf{R}$[Index].Position.1st)
Find the maximum index of rows with Position = 1st, then return the Venue of the row with that index.

z_3: $\mathbf{R}$[Venue].argmax(Position.Number.*1*, $\mathbf{R}[\lambda x.\mathbf{R}$[Date].$\mathbf{R}$[Year].$x]$)
Among rows with Position number 1, pick one with latest date in the Year column and return the Venue.

Spurious

z_4: $\mathbf{R}$[Venue].argmax(Position.Number.*1*, $\mathbf{R}[\lambda x.\mathbf{R}$[Number].$\mathbf{R}$[Time].$x]$)
Among rows with Position number 1, pick the one with maximum Time number. Return the Venue.

z_5: $\mathbf{R}$[Venue].Year.Number.($\mathbf{R}$[Number].$\mathbf{R}$[Year].argmax(Type.Row, Index) $-$ *1*)
Subtract 1 from the Year in the last row, then return the Venue of the row with that Year.

Inconsistent

$\tilde{z}$: $\mathbf{R}$[Venue].argmin(Position.1st, Index)
Among rows with Position = 1st, pick the one with minimum index, then return the Venue. (= Finland)

Figure 1: Six logical forms generated from the question x. The first five are *consistent*: they execute to the correct answer y. Of those, *correct* logical forms z_1, z_2, and z_3 are different ways to represent the semantics of x, while *spurious* logical forms z_4 and z_5 get the right answer y for the wrong reasons.

fictitious worlds to test the denotations of the logical forms in Z. We use crowdsourcing to annotate the correct denotations on a subset of the generated worlds. To reduce the amount of annotation needed, we choose the subset that maximizes the expected information gain. The pruned set of logical forms would provide a stronger supervision signal for training a semantic parser.

We test our methods on the WIKITABLEQUESTIONS dataset of complex questions on Wikipedia tables. We define a simple, general set of deduction rules (Section 3), and use DPD to confirm that the rules generate a correct logical form in

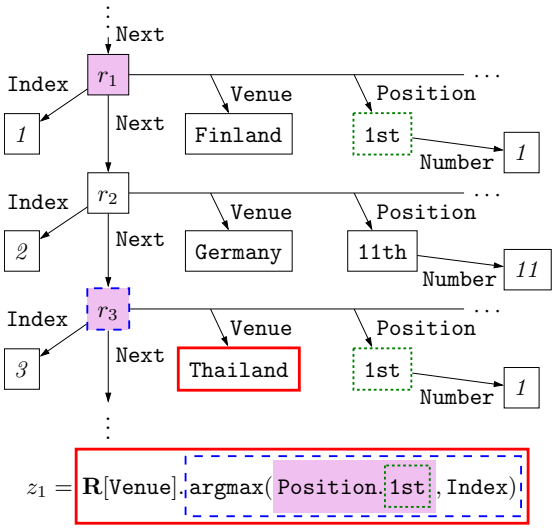

Figure 2: The table in Figure 1 is converted into a graph. The recursive execution of logical form z_1 is shown via the different colors and styles.

76% of the examples, up from the 53.5% in Pasupat and Liang (2015). Moreover, unlike beam search, DPD is guaranteed to find all consistent logical forms up to a bounded size. Finally, by using annotated data on fictitious worlds, we are able to prune out 92.1% of the spurious logical forms.

2 Setup

The overarching motivation of this work is allowing people to ask questions involving computation on semi-structured knowledge sources such as tables from the Web. This section introduces how the knowledge source is represented, how the computation is carried out using *logical forms*, and our task of inferring correct logical forms.

Worlds. We use the term *world* to refer to a collection of entities and relations between entities. One way to represent a world w is as a directed graph with nodes for entities and directed edges for relations. (For example, a world about geography would contain a node `Europe` with an edge `Contains` to another node `Germany`.)

In this paper, we use data tables from the Web as knowledge sources, such as the one in Figure 1. We follow the construction in Pasupat and Liang (2015) for converting a table into a directed graph (see Figure 2). Rows and cells become nodes (e.g., r_0 = first row and `Finland`) while columns become labeled directed edges between them (e.g., `Venue` maps r_1 to `Finland`). The graph is augmented with additional edges `Next` (from each

row to the next) and Index (from each row to its index number). In addition, we add normalization edges to cell nodes, including Number (from the cell to the first number in the cell), Num2 (the second number), Date (interpretation as a date), and Part (each list item if the cell represents a list). For example, a cell with content "*3-4*" has a Number edge to the integer *3*, a Num2 edge to *4*, and a Date edge to *XX-03-04*.

Logical forms. We can perform computation on a world w using a *logical form* z, a small program that can be executed on the world, resulting in a *denotation* $[\![z]\!]_w$.

We use lambda DCS (Liang, 2013) as the language of logical forms. As a demonstration, we will use z_1 in Figure 2 as an example. The smallest units of lambda DCS are entities (e.g., 1st) and relations (e.g., Position). Larger logical forms can be constructed using logical operations, and the denotation of the new logical form can be computed from denotations of its constituents. For example, applying the *join* operation on Position and 1st gives Position.1st, whose denotation is the set of entities with relation Position pointing to 1st. With the world in Figure 2, the denotation is $[\![\texttt{Position.1st}]\!]_w = \{r_1, r_3\}$, which corresponds to the 2nd and 4th rows in the table. The partial logical form Position.1st is then used to construct argmax(Position.1st, Index), the denotation of which can be computed by mapping the entities in $[\![\texttt{Position.1st}]\!]_w = \{r_1, r_3\}$ using the relation Index ($\{r_0 : 0, r_1 : 1, \dots\}$), and then picking the one with the largest mapped value (r_3, which is mapped to *3*). The resulting logical form is finally combined with $\mathbf{R}[\texttt{Venue}]$ with another *join* operation. The relation $\mathbf{R}[\texttt{Venue}]$ is the *reverse* of Venue, which corresponds to traversing Venue edges in the reverse direction.

Semantic parsing. A semantic parser maps a natural language utterance x (e.g., "*Where did the last 1st place finish occur?*") into a logical form z. With denotations as supervision, a semantic parser is trained to put high probability on z's that are *consistent*—logical forms that execute to the correct denotation y (e.g., Thailand). When the space of logical forms is large, searching for consistent logical forms z can become a challenge.

As illustrated in Figure 1, consistent logical forms can be divided into two groups: *correct* logical forms represent valid ways for computing the answer, while *spurious* logical forms accidentally get the right answer for the wrong reasons (e.g., z_4 picks the row with the maximum time but gets the correct answer anyway).

Tasks. Denote by Z and Z_c the sets of all consistent and correct logical forms, respectively. The first task is to efficiently compute Z given an utterance x, a world w, and the correct denotation y (Section 4). With the set Z, the second task is to infer Z_c by pruning spurious logical forms from Z (Section 5).

3 Deduction rules

The space of logical forms given an utterance x and a world w is defined recursively by a set of *deduction rules* (Table 1). In this setting, each constructed logical form belongs to a *category* (*Set*, *Rel*, or *Map*). These categories are used for type checking in a similar fashion to categories in syntactic parsing. Each deduction rule specifies the categories of the arguments, category of the resulting logical form, and how the logical form is constructed from the arguments.

Deduction rules are divided into base rules and compositional rules. A base rule follows one of the following templates:

$$\textit{TokenSpan}[span] \rightarrow c\,[f(span)] \quad (1)$$

$$\emptyset \rightarrow c\,[f()] \quad (2)$$

A rule of Template 1 is triggered by a span of tokens from x (e.g., to construct z_1 in Figure 2 from x in Figure 1, Rule B1 from Table 1 constructs 1st of category *Set* from the phrase "*1st*"). Meanwhile, a rule of Template 2 generates a logical form without any trigger (e.g., Rule B5 generates Position of category *Rel* from the graph edge Position without a specific trigger in x).

Compositional rules then construct larger logical forms from smaller ones:

$$c_1\,[z_1] + c_2\,[z_2] \rightarrow c\,[g(z_1, z_2)] \quad (3)$$

$$c_1\,[z_1] \rightarrow c\,[g(z_1)] \quad (4)$$

A rule of Template 3 combines partial logical forms z_1 and z_2 of categories c_1 and c_2 into $g(z_1, z_2)$ of category c (e.g., Rule C1 uses 1st of category *Set* and Position of category *Rel* to construct Position.1st of category *Set*). Template 4 works similarly.

Most rules construct logical forms without requiring a trigger from the utterance x. This is

	Rule	Semantics
	Base Rules	
B1	*TokenSpan* → *Set*	fuzzymatch($span$)
	(entity fuzzily matching the text: "*chinese*" → `China`)	
B2	*TokenSpan* → *Set*	val($span$)
	(interpreted value: "*march 2015*" → *2015-03-XX*)	
B3	$\emptyset$ → *Set*	`Type.Row`
	(the set of all rows)	
B4	$\emptyset$ → *Set*	$c \in$ ClosedClass
	(any entity from a column with few unique entities)	
	(e.g., `400m` or `relay` from the Event column)	
B5	$\emptyset$ → *Rel*	$r \in$ GraphEdges
	(any relation in the graph: `Venue`, `Next`, `Num2`, ...)	
B6	$\emptyset$ → *Rel*	`!=` | `<` | `<=` | `>` | `>=`
	Compositional Rules	
C1	*Set* + *Rel* → *Set*	$z_2.z_1 \mid \mathbf{R}[z_2].z_1$
	($\mathbf{R}[z]$ is the reverse of z; i.e., flip the arrow direction)	
C2	*Set* → *Set*	$a(z_1)$
	($a \in \{\texttt{count}, \texttt{max}, \texttt{min}, \texttt{sum}, \texttt{avg}\}$)	
C3	*Set* + *Set* → *Set*	$z_1 \sqcap z_2 \mid z_1 \sqcup z_2 \mid z_1 - z_2$
	(subtraction is only allowed on numbers)	
	Compositional Rules with Maps	
	Initialization	
M1	*Set* → *Map*	(z_1, x) (identity map)
	Operations on Map	
M2	*Map* + *Rel* → *Map*	$(u_1, z_2.b_1) \mid (u_1, \mathbf{R}[z_2].b_1)$
M3	*Map* → *Map*	$(u_1, a(b_1))$
	($a \in \{\texttt{count}, \texttt{max}, \texttt{min}, \texttt{sum}, \texttt{avg}\}$)	
M4	*Map* + *Set* → *Map*	$(u_1, b_1 \sqcap z_2) \mid \ldots$
M5	*Map* + *Map* → *Map*	$(u_1, b_1 \sqcap b_2) \mid \ldots$
	(Allowed only when $u_1 = u_2$)	
	(Rules M4 and M5 are repeated for $\sqcup$ and $-$)	
	Finalization	
M6	*Map* → *Set*	$\texttt{argmin}(u_1, \mathbf{R}[\lambda x.b_1])$ | $\texttt{argmax}(u_1, \mathbf{R}[\lambda x.b_1])$

Table 1: Deduction rules define the space of logical forms by specifying how partial logical forms are constructed. The logical form of the i-th argument is denoted by z_i (or (u_i, b_i) if the argument is a *Map*). The set of final logical forms contains any logical form with category *Set*.

crucial for generating implicit relations (e.g., generating `Year` from "*what's the venue in 2000?*" without a trigger "*year*"), and generating operations without a lexicon (e.g., generating `argmax` from "*where's the longest competition*"). However, the downside is that the space of possible logical forms becomes very large.

The *Map* category. The technique in this paper requires execution of partial logical forms. This poses a challenge for `argmin` and `argmax` operations, which take a set and a binary relation as arguments. The binary could be a complex function (e.g., in z_3 from Figure 1). While it is possible to build the binary independently from the set, executing a complex binary is sometimes impossible (e.g., the denotation of $\lambda x.\texttt{count}(x)$ is impossible to write explicitly without knowledge of x).

We address this challenge with the *Map* category. A *Map* is a pair (u, b) of a finite set u (unary) and a binary relation b. The denotation of (u, b) is $([\![u]\!]_w, [\![b]\!]'_w)$ where the binary $[\![b]\!]'_w$ is $[\![b]\!]_w$ with the domain restricted to the set $[\![u]\!]_w$. For example, consider the construction of $\texttt{argmax}(\texttt{Position.1st}, \texttt{Index})$. After constructing `Position.1st` with denotation $\{r_1, r_3\}$, Rule M1 initializes $(\texttt{Position.1st}, x)$ with denotation $(\{r_1, r_3\}, \{r_1 : \{r_1\}, r_3 : \{r_3\}\})$. Rule M2 is then applied to generate $(\texttt{Position.1st}, \mathbf{R}[\texttt{Index}].x)$ with denotation $(\{r_1, r_3\}, \{r_1 : \{1\}, r_3 : \{3\}\})$. Finally, Rule M6 converts the *Map* into the desired `argmax` logical form with denotation $\{r_3\}$.

Generality of deduction rules. Using domain knowledge, previous work restricted the space of logical forms by manually defining the categories c or the semantic functions f and g to fit the domain. For example, the category *Set* might be divided into *Records*, *Values*, and *Atomic* when the knowledge source is a table (Pasupat and Liang, 2015). Another example is when a compositional rule g (e.g., $\texttt{sum}(z_1)$) must be triggered by some phrase in a lexicon (e.g., words like "*total*" that align to `sum` in the training data). Such restrictions make search more tractable but greatly limit the scope of questions that can be answered.

Here, we have increased the coverage of logical forms by making the deduction rules simple and general, essentially following the syntax of lambda DCS. The base rules only generates entities that approximately match the utterance, but all possible relations, and all possible further combinations.

Beam search. Given the deduction rules, an utterance x and a world w, we would like to generate all derived logical forms Z. We first present the floating parser (Pasupat and Liang, 2015), which uses beam search to generate $Z_\text{b} \subseteq Z$, a usually incomplete subset. Intuitively, the algorithm first constructs base logical forms based on spans of the utterance, and then builds larger logical forms of increasing size in a "floating" fashion—without requiring a trigger from the utterance.

Formally, partial logical forms with category c and size s are stored in a *cell* (c, s). The algorithm first generates base logical forms from base deduction rules and store them in cells $(c, 0)$ (e.g., the cell $(\textit{Set}, 0)$ contains `1st`, `Type.Row`, and so on). Then for each size $s = 1, \ldots, s_\text{max}$, we populate

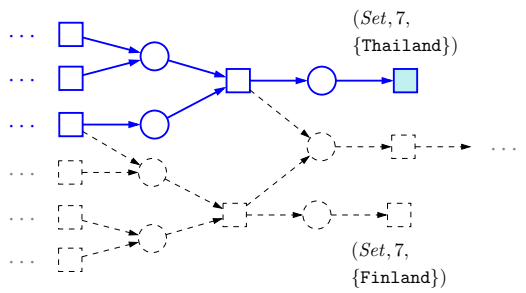

Figure 3: The first pass of DPD constructs cells (c, s, d) (square nodes) using denotationally invariant semantic functions (circle nodes). The second pass enumerates all logical forms along paths that lead to the correct denotation y (solid lines).

the cells (c, s) by applying compositional rules on partial logical forms with size less than s. For instance, when $s = 2$, we can apply Rule C1 on logical forms `Number`.*1* from cell $(\textit{Set}, s_1 = 1)$ and `Position` from cell $(\textit{Rel}, s_2 = 0)$ to create `Position.Number`.*1* in cell $(\textit{Set}, s_0 + s_1 + 1 = 2)$. After populating each cell (c, s), the list of logical forms in the cell is pruned based on the model scores to a fixed beam size in order to control the search space. Finally, the set Z_{b} is formed by collecting logical forms from all cells $(\textit{Set}, s)$ for $s = 1, \ldots, s_{\max}$.

Due to the generality of our deduction rules, the number of logical forms grows quickly as the size s increases. As such, partial logical forms that are essential for building the desired logical forms might fall off the beam early on. In the next section, we present a new search method that compresses the search space using denotations.

4 Dynamic programming on denotations

Our first step toward finding all correct logical forms is to represent all consistent logical forms (those that execute to the correct denotation). Formally, given x, w, and y, we wish to generate the set Z of all logical forms z such that $[\![z]\!]_w = y$.

As mentioned in the previous section, beam search does not recover the full set Z due to pruning. Our key observation is that while the number of logical forms explodes, the number of *distinct denotations* of those logical forms is much more controlled, as multiple logical forms can share the same denotation. So instead of directly enumerating logical forms, we use *dynamic programming on denotations* (DPD), which is inspired by similar methods from program induction (Lau et al., 2003; Liang et al., 2010; Gulwani, 2011).

The main idea of DPD is to collapse logical forms with the same denotation together. Instead of using cells (c, s) as in beam search, we perform dynamic programming using cells (c, s, d) where d is a denotation. For instance, the logical form `Position.Number`.*1* will now be stored in cell $(\textit{Set}, 2, \{r_1, r_3\})$.

For DPD to work, each deduction rule must have a *denotationally invariant* semantic function g, meaning that the denotation of the resulting logical form $g(z_1, z_2)$ only depends on the denotations of z_1 and z_2:

$$[\![z_1]\!]_w = [\![z_1']\!]_w \ \wedge \ [\![z_2]\!]_w = [\![z_2']\!]_w \\ \Rightarrow [\![g(z_1, z_2)]\!]_w = [\![g(z_1', z_2')]\!]_w$$

All of our deduction rules in Table 1 are denotationally invariant, but a rule that, for instance, returns the argument with the larger logical form size would not be. Applying a denotationally invariant deduction rule on any pair of logical forms from (c_1, s_1, d_1) and (c_2, s_2, d_2) always results in a logical form with the same denotation d in the same cell $(c, s_1 + s_2 + 1, d)$.[1] (For example, the cell $(\textit{Set}, 4, \{r_3\})$ contains $z_1 :=$ `argmax(Position.1st, Index)` and $z_1' :=$ `argmin(Event.Relay, Index)`. Combining each of these with `Venue` using Rule C1 gives $\mathbf{R}[\texttt{Venue}].z_1$ and $\mathbf{R}[\texttt{Venue}].z_1'$, which belong to the same cell $(\textit{Set}, 5, \{\texttt{Thailand}\})$).

Algorithm. DPD proceeds in two forward passes. The first pass finds the possible combinations of cells (c, s, d) that lead to the correct denotation y, while the second pass enumerates the logical forms in the cells found in the first pass. Figure 3 illustrates the DPD algorithm.

In the first pass, we are only concerned about finding relevant cell combinations and not the actual logical forms. Therefore, any logical form that belongs to a cell could be used as an argument of a deduction rule to generate further logical forms. Thus, we keep at most one logical form per cell; subsequent logical forms that are generated for that cell are discarded.

After populating all cells up to size $s_{\max}$, we list all cells $(\textit{Set}, s, y)$ with the correct denotation y, and then note all possible rule combinations $(\text{cell}_1, \text{rule})$ or $(\text{cell}_1, \text{cell}_2, \text{rule})$ that lead to those

[1] Semantic functions f with one argument work similarly.

final cells, including the combinations that yielded discarded logical forms.

The second pass retrieves the actual logical forms that yield the correct denotation. To do this, we simply populate the cells (c, s, d) with all logical forms, using only rule combinations that lead to final cells. This elimination of irrelevant rule combinations effectively reduces the search space. (In Section 6.2, we empirically show that the number of cells considered is reduced by 98.7%.)

The parsing chart is represented as a hypergraph as in Figure 3. After eliminating unused rule combinations, each of the remaining hyperpaths from base predicates to the target denotation corresponds to a single logical form. making the remaining parsing chart a compact implicit representation of all consistent logical forms. This representation is guaranteed to cover all possible logical forms under the size limit $s_{\max}$ that can be constructed by the deduction rules.

In our experiments, we apply DPD on the deduction rules in Table 1 and explicitly enumerate the logical forms produced by the second pass. For efficiency, we prune logical forms that are clearly redundant (e.g., applying `max` on a set of size 1). We also restrict a few rules that might otherwise create too many denotations. For example, we restricted the union operation ($\sqcup$) except unions of two entities (e.g., we allow `Germany` $\sqcup$ `Finland` but not `Venue.Hungary` $\sqcup \ldots$), subtraction when building a *Map*, and `count` on a set of size 1.[2]

5 Fictitious worlds

After finding the set Z of all consistent logical forms, we want to filter out spurious logical forms. To do so, we observe that semantically correct logical forms should also give the correct denotation in worlds w' other than than w. In contrast, spurious logical forms will fail to produce the correct denotation on some other world.

Generating fictitious worlds. With the observation above, we generate *fictitious worlds* $w_1, w_2, \ldots$, where each world w_i is a slight alteration of w. As we will be executing logical forms $z \in Z$ on w_i, we should ensure that all entities and relations in $z \in Z$ appear in the fictitious world w_i (e.g., z_1 in Figure 1 would be meaningless if the entity `1st` does not appear in w_i). To this end, we impose that all predicates present in the original world w should also be present in w_i as well.

[2]While we technically can apply `count` on sets of size 1, the number of spurious logical forms explodes as there are too many sets of size 1 generated.

Year	Venue	Position	Event	Time
2001	Finland	7th	relay	46.62
2003	Germany	1st	400m	180.32
2005	China	1st	relay	47.12
2007	Hungary	7th	relay	182.05

Figure 4: From the example in Figure 1, we generate a table for the fictitious world w_1.

	w	w_1	w_2	$\cdots$	
z_1	Thailand	China	Finland	$\cdots$	q_1
z_2	Thailand	China	Finland	$\cdots$	
z_3	Thailand	China	Finland	$\cdots$	
z_4	Thailand	Germany	China	$\cdots$	q_2
z_5	Thailand	China	China	$\cdots$	q_3
z_6	Thailand	China	China	$\cdots$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$		

Figure 5: We execute consistent logical forms $z_i \in Z$ on fictitious worlds to get denotation tuples. Logical forms with the same denotation tuple are grouped into the same equivalence class q_j.

In our case where the world w comes from a data table t, we construct w_i from a new table t_i as follows: we go through each column of t and resample the cells in that column. The cells are sampled using random draws without replacement if the original cells are all distinct, and with replacement otherwise. Sorted columns are kept sorted. To ensure that predicates in w exist in w_i, we use the same set of table columns and enforce that any entity fuzzily matching a span in the question x must be present in t_i (e.g., for the example in Figure 1, the generated t_i must contain "1st"). Figure 4 shows an example fictitious table generated from the table in Figure 1.

Fictitious worlds are similar to test suites for computer programs. However, unlike manually designed test suites, we do not yet know the correct answer for each fictitious world or whether a world is helpful for filtering out spurious logical forms. The next subsections introduce our method for choosing a subset of useful fictitious worlds to be annotated.

Equivalence classes. Let $W = (w_1, \ldots, w_k)$ be the list of all possible fictitious worlds. For each $z \in Z$, we define the denotation tuple $[\![z]\!]_W = ([\![z]\!]_{w_1}, \ldots, [\![z]\!]_{w_k})$. We observe that some logical forms produce the same denotation across all

fictitious worlds. This may be due to an algebraic equivalence in logical forms (e.g., z_1 and z_2 in Figure 1) or due to the constraints in the construction of fictitious worlds (e.g., z_1 and z_3 in Figure 1 are equivalent as long as the Year column is sorted). We group logical forms into equivalence classes based on their denotation tuples, as illustrated in Figure 5. When the question is unambiguous, we expect at most one equivalence class to contain correct logical forms.

Annotation. To pin down the correct equivalence class, we acquire the correct answers to the question x on some subset $W' = (w'_1, \ldots, w'_\ell) \subseteq W$ of ℓ fictitious worlds, as it is impractical to obtain annotations on all fictitious worlds in W. We compile equivalence classes that agree with the annotations into a set Z_c of correct logical forms.

We want to choose W' that gives us the most information about the correct equivalence class as possible. This is analogous to standard practices in active learning (Settles, 2010).[3] Let $\mathcal{Q}$ be the set of all equivalence classes q, and let $[\![q]\!]_{W'}$ be the denotation tuple computed by executing an arbitrary $z \in q$ on W'. The subset W' divides $\mathcal{Q}$ into partitions $F_t = \{q \in \mathcal{Q} : [\![q]\!]_{W'} = t\}$ based on the denotation tuples t (e.g., from Figure 5, if W' contains just w_2, then q_2 and q_3 will be in the same partition $F_{(\text{China})}$). The annotation t^*, which is also a denotation tuple, will mark one of these partitions F_{t^*} as correct. Thus, to prune out many spurious equivalence classes, the partitions should be as numerous and as small as possible.

More formally, we choose a subset W' that maximizes the expected information gain (or equivalently, the reduction in entropy) about the correct equivalence class given the annotation. With random variables $Q \in \mathcal{Q}$ representing the correct equivalence class and $T^*_{W'}$ for the annotation on worlds W', we seek to find $\arg\min_{W'} \mathrm{H}(Q \mid T^*_{W'})$. Assuming a uniform prior on Q ($p(q) = 1/|\mathcal{Q}|$) and accurate annotation ($p(t^* \mid q) = \mathbf{I}[q \in F_{t^*}]$):

$$\begin{aligned}\mathrm{H}(Q \mid T^*_{W'}) &= \sum_{q,t} p(q,t) \log \frac{p(t)}{p(q,t)} \\ &= \frac{1}{|\mathcal{Q}|} \sum_t |F_t| \log |F_t|. \qquad (*)\end{aligned}$$

[3]The difference is that we are obtaining partial information about an individual example rather than partial information about the parameters.

We exhaustively search for W' that minimizes (*). The objective value follows our intuition since $\sum_t |F_t| \log |F_t|$ is small when the terms $|F_t|$ are small and numerous.

In our experiments, we approximate the full set W of fictitious worlds by generating $k = 30$ worlds to compute equivalence classes. We choose a subset of $\ell = 5$ worlds to be annotated.

6 Experiments

For the experiments, we use the training portion of the WIKITABLEQUESTIONS dataset (Pasupat and Liang, 2015), which consists of 14,152 questions on 1,679 Wikipedia tables gathered by crowd workers. Answering these complex questions requires different types of operations. The same operation can be phrased in different ways (e.g., "*best*", "*top ranking*", or "*lowest ranking number*") and the interpretation of some phrases depend on the context (e.g., "*number of*" could be a table lookup or a count operation). The lexical content of the questions is also quite diverse: even excluding numbers and symbols, the 14,152 training examples contain 9,671 unique words, only 10% of which appear more than 10 times.

We attempted to manually annotate the first 300 examples with lambda DCS logical forms. We successfully constructed correct logical forms for 84% of these examples, which is a good number considering the questions were created by humans who could use the table however they wanted. The remaining 16% reflect limitations in our setup—for example, non-canonical table layouts, answers appearing in running text or images, and common sense reasoning (e.g., knowing that "Quarterfinal" is better than "Round of 16").

6.1 Generality of deduction rules

We compare our set of deduction rules with the one given in Pasupat and Liang (2015) (henceforth PL15). PL15 reported generating the annotated logical form in 53.5% of the first 200 examples. With our more general deduction rules, we use DPD to verify that the rules are able to generate the annotated logical form in 76% of the first 300 examples, within the logical form size limit $s_{\max}$ of 7. This is 90.5% of the examples that were successfully annotated. Figure 6 shows some examples of logical forms we cover that PL15 could not. Since DPD is guaranteed to find all consistent logical forms, we can be sure that the logical

"*which opponent has the most wins*" $z = \texttt{argmax}(\mathbf{R}[\texttt{Opponent}].\texttt{Type.Row},$ $\quad \mathbf{R}[\lambda x.\texttt{count}(\texttt{Opponent}.x \sqcap \texttt{Result.Lost}])$
"*how long did ian armstrong serve?*" $z = \mathbf{R}[\texttt{Num2}].\mathbf{R}[\texttt{Term}].\texttt{Member.IanArmstrong}$ $\quad - \mathbf{R}[\texttt{Number}].\mathbf{R}[\texttt{Term}].\texttt{Member.IanArmstrong}$
"*which players came in a place before lukas bauer?*" $z = \mathbf{R}[\texttt{Name}].\texttt{Index.<}.\mathbf{R}[\texttt{Index}].\texttt{Name.LukasBauer}$
"*which players played the same position as ardo kreek?*" $z = \mathbf{R}[\texttt{Player}].\texttt{Position}.\mathbf{R}[\texttt{Position}].\texttt{Player.Ardo}$ $\quad \sqcap \texttt{!=.Ardo}$

Figure 6: Several example logical forms our system can generated that are not covered by the deduction rules from the previous work PL15.

forms not covered are due to limitations of the deduction rules. Indeed, the remaining examples either have logical forms with size larger than 7 or require other operations such as addition, union of arbitrary sets, etc.

6.2 Dynamic programming on denotations

Search space. To demonstrate the savings gained by collapsing logical forms with the same denotation, we track the growth of the number of unique logical forms and denotations as the logical form size increases. The plot in Figure 7 shows that the space of logical forms explodes much more quickly than the space of denotations.

The use of denotations also saves us from considering a significant amount of irrelevant partial logical forms. On average over 14,152 training examples, DPD generates approximately 25,000 consistent logical forms. The first pass of DPD generates $\approx$ 153,000 cells (c, s, d), while the second pass generates only $\approx$ 2,000 cells resulting from $\approx$ 8,000 rule combinations, resulting in a 98.7% reduction in the number of cells that have to be considered.

Comparison with beam search. We compare DPD to beam search on the ability to generate (but not rank) the annotated logical forms. We consider two settings: when the beam search parameters are uninitialized (i.e., the beams are pruned randomly), and when the parameters are trained using the system from PL15 (i.e., the beams are pruned based on model scores). The plot in Figure 8 shows that DPD generates more annotated logical forms (76%) compared to beam search (53.7%), even when beam search is guided heuristically by learned parameters. Note that DPD is an exact algorithm and does not require a heuristic.

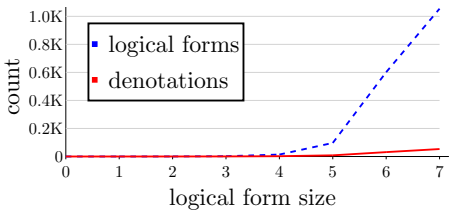

Figure 7: The median of the number of logical forms (dashed) and denotations (solid) as the formula size increases. The space of logical forms grows much faster than the space of denotations.

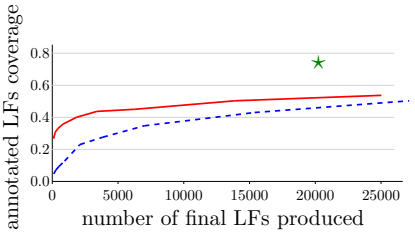

Figure 8: The number of annotated logical forms that can be generated by beam search, both uninitialized (dashed) and initialized (solid), increases with the number of candidates generated (controlled by beam size), but lacks behind DPD (star).

6.3 Fictitious worlds

We now explore how fictitious worlds divide the set of logical forms into equivalence classes, and how the annotated denotations on the chosen worlds help us prune spurious logical forms.

Equivalence classes. Using 30 fictitious worlds per example, we produce an average of 1,237 equivalence classes. One possible concern with using a limited number of fictitious worlds is that we may fail to distinguish some pairs of non-equivalent logical forms. We verify the equivalence classes against the ones computed using 300 fictitious worlds. We found that only 5% of the logical forms are split from the original equivalence classes.

Ideal Annotation. After computing equivalence classes, we choose a subset W' of 5 fictitious worlds to be annotated based on the information-theoretic objective. For each of the 252 examples with an annotated logical form z^*, we use the denotation tuple $t^* = [\![z^*]\!]_{W'}$ as the annotated answers on the chosen fictitious worlds. We are able to rule out 98.7% of the spurious equivalence classes and 98.3% of spurious logical forms. Furthermore, we are able to filter down to just one equivalence class in 32.7% of the examples, and

at most three equivalence classes in 51.3% of the examples. If we choose 5 fictitious worlds randomly instead of maximizing information gain, then the above statistics are 22.6% and 36.5%, respectively. When more than one equivalence classes remain, usually only one class is a dominant class with many equivalent logical forms, while other classes are small and contain logical forms with unusual patterns (e.g., z_5 in Figure 1).

The average size of the correct equivalence class is $\approx$ 3,000 with the standard deviation of $\approx$ 8,000. Because we have an expressive logical language, there are fundamentally many equivalent ways of computing the same quantity.

Crowdsourced Annotation. Data from crowdsourcing is more susceptible to errors. From the 252 annotated examples, we use 177 examples where at least two crowd workers agree on the answer of the original world w. When the crowdsourced data is used to rule out spurious logical forms, the entire set Z of consistent logical forms is pruned out in 11.3% of the examples, and the correct equivalent class is removed in 9% of the examples. These issues are due to annotation errors, inconsistent data (e.g., having date of death before birth date), and different interpretations of the question on the fictitious worlds. For the remaining examples, we are able to prune out 92.1% of spurious logical forms (or 92.6% of spurious equivalence classes).

To prevent the entire Z from being pruned, we can relax our assumption and keep logical forms z that disagree with the annotation in at most 1 fictitious world. The number of times Z is pruned out is reduced to 3%, but the number of spurious logical forms pruned also decreases to 78%.

7 Related Work and Discussion

This work evolved from a long tradition of learning executable semantic parsers, initially from annotated logical forms (Zelle and Mooney, 1996; Kate et al., 2005; Zettlemoyer and Collins, 2005; Zettlemoyer and Collins, 2007; Kwiatkowski et al., 2010), but more recently from denotations (Clarke et al., 2010; Liang et al., 2011; Berant et al., 2013; Kwiatkowski et al., 2013; Pasupat and Liang, 2015). A central challenge in learning from denotations is finding consistent logical forms (those that execute to a given denotation).

As Kwiatkowski et al. (2013) and Berant and Liang (2014) both noted, a chief difficulty with executable semantic parsing is the "schema mismatch"—words in the utterance do not map cleanly onto the predicates in the logical form. This mismatch is especially pronounced in the WIKITABLEQUESTIONS of Pasupat and Liang (2015). In the second example of Figure 6, "*how long*" is realized by a logical form that computes a difference between two dates. The ramification of this mismatch is that finding consistent logical forms cannot solely proceed from the language side. This paper is about using annotated denotations to drive the search over logical forms.

This takes us into the realm of program induction, where the goal is to infer a program (logical form) from input-output pairs (for us, world-denotation pairs). Here, previous work has also leveraged the idea of dynamic programming on denotations (Lau et al., 2003; Liang et al., 2010; Gulwani, 2011), though for more constrained spaces of programs. Continuing the program analogy, generating fictitious worlds is similar in spirit to fuzz testing for generating new test cases (Miller et al., 1990), but the goal there is coverage in a single program rather than identifying the correct (equivalence class of) programs. This connection can potentially improve the flow of ideas between the two fields.

Finally, the effectiveness of dynamic programming on denotations relies on having a manageable set of denotations. For more complex logical forms and larger knowledge graphs, there are many possible angles worth exploring: performing abstract interpretation to collapse denotations into equivalence classes (Cousot and Cousot, 1977), relaxing the notion of getting the correct denotation (Steinhardt and Liang, 2015), or working in a continuous space and relying on gradient descent (Guu et al., 2015; Neelakantan et al., 2016; Yin et al., 2016; Reed and de Freitas, 2016). This paper, by virtue of exact dynamic programming, sets the standard.

Acknowledgments. We gratefully acknowledge the support of the Google Natural Language Understanding Focused Program. In addition, we would like to thank anonymous reviewers for their helpful comments.

Reproducibility. Code and experiments for this paper are available on the CodaLab platform at `https://worksheets.codalab.org/worksheets/0x47cc64d9c8ba4a878807c7c35bb22a42/`.

References

Y. Artzi and L. Zettlemoyer. 2013. UW SPF: The University of Washington semantic parsing framework. *arXiv preprint arXiv:1311.3011*.

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)*.

J. Clarke, D. Goldwasser, M. Chang, and D. Roth. 2010. Driving semantic parsing from the world's response. In *Computational Natural Language Learning (CoNLL)*, pages 18–27.

P. Cousot and R. Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Principles of Programming Languages (POPL)*, pages 238–252.

S. Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *ACM SIGPLAN Notices*, 46(1):317–330.

K. Guu, J. Miller, and P. Liang. 2015. Traversing knowledge graphs in vector space. In *Empirical Methods in Natural Language Processing (EMNLP)*.

R. J. Kate, Y. W. Wong, and R. J. Mooney. 2005. Learning to transform natural to formal languages. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1062–1068.

T. Kwiatkowski, L. Zettlemoyer, S. Goldwater, and M. Steedman. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1223–1233.

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)*.

T. Lau, S. Wolfman, P. Domingos, and D. S. Weld. 2003. Programming by demonstration using version space algebra. *Machine Learning*, 53:111–156.

P. Liang, M. I. Jordan, and D. Klein. 2010. Learning programs: A hierarchical Bayesian approach. In *International Conference on Machine Learning (ICML)*, pages 639–646.

P. Liang, M. I. Jordan, and D. Klein. 2011. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*, pages 590–599.

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

B. P. Miller, L. Fredriksen, and B. So. 1990. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33(12):32–44.

A. Neelakantan, Q. V. Le, and I. Sutskever. 2016. Neural programmer: Inducing latent programs with gradient descent. In *International Conference on Learning Representations (ICLR)*.

P. Pasupat and P. Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Association for Computational Linguistics (ACL)*.

S. Reed and N. de Freitas. 2016. Neural programmer-interpreters. In *International Conference on Learning Representations (ICLR)*.

B. Settles. 2010. Active learning literature survey. Technical report, University of Wisconsin, Madison.

J. Steinhardt and P. Liang. 2015. Learning with relaxed supervision. In *Advances in Neural Information Processing Systems (NIPS)*.

P. Yin, Z. Lu, H. Li, and B. Kao. 2016. Neural enquirer: Learning to query tables with natural language. *arXiv*.

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. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence (UAI)*, pages 658–666.

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.