Learning Executable Semantic Parsers for Natural Language Understanding

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

ABSTRACT

For building question answering systems and natural language interfaces, semantic parsing has emerged as an important and powerful paradigm. Semantic parsers map natural language into logical forms, the classic representation for many important linguistic phenomena. The modern twist is that we are interested in learning semantic parsers from data, which introduces a new layer of statistical and computational issues. This article lays out the components of a statistical semantic parser, highlighting the key challenges. We will see that semantic parsing is a rich fusion of the logical and the statistical world, and that this fusion will play an integral role in the future of natural language understanding systems.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Language parsing and understanding*

1. INTRODUCTION

A long-standing goal of artificial intelligence (AI) is to build systems capable of understanding natural language. To focus the notion of "understanding" a bit, let us say that the system must produce an appropriate action upon receiving an input utterance from a human. For example:

Context:	knowledge of mathematics
Utterance:	*What is the largest prime less than 10?*
Action:	7
Context:	knowledge of geography
Utterance:	*What is the tallest mountain in Europe?*
Action:	Mt. Elbrus
Context:	user's calendar
Utterance:	*Cancel all my meetings after 4pm tomorrow.*
Action:	(removes meetings from calendar)

We are interested in utterances such as the ones above, which require deep understanding and reasoning. This article focuses on *semantic parsing*, an area within the field of natural language processing (NLP), which has been growing over the last decade. Semantic parsers map input utterances into semantic representations called *logical forms* that support this form of reasoning. For example, the first utterance above would map onto the logical form $\max(\texttt{primes} \cap (-\infty, 10))$. We can think of the logical form as a program that is *executed* to yield the desired behavior (e.g., answering 7). The second utterance would map onto a database query; the third, onto an invocation of a calendar API.

Semantic parsing is rooted in formal semantics, pioneered by logician Richard Montague [25], who famously argued that there is "no important theoretical difference between natural languages and the artificial languages of logicians." Semantic parsing, by residing in the practical realm, is more exposed to the differences between natural language and logic, but it inherits two general insights from formal semantics: The first idea is *model theory*, which states that expressions (e.g., `primes`) are mere symbols which only obtain their meaning or denotation (e.g., $\{2, 3, 5, \dots\}$) by executing the expression with respect to a model, or in our terminology, a context. This property allows us to factor out the understanding of language (semantic parsing) from world knowledge (execution). Indeed, one can understand the utterance "*What is the largest prime less than 10?*" without actually computing the answer. The second idea is *compositionality*, a principle often attributed to Gottlob Frege, which states that the denotation of an expression is defined recursively in terms of the denotations of its subexpressions. For example, `primes` denotes the set of primes, $(-\infty, 10)$ denotes the set of numbers smaller than 10, and so $\texttt{primes} \cap (-\infty, 10)$ denotes the intersection of those two sets. This compositionality is what allows us to have a succinct characterization of meaning for a combinatorial range of possible utterances.

Early systems. Logical forms have played a foundational role in natural language understanding systems since their genesis in the 1960s. Early examples included LUNAR, a natural language interface into a database about moon rocks [34], and SHRDLU, a system that could both answer questions and perform actions in a toy blocks world environment [32]. For their time, these systems were significant achievements. They were able to handle fairly complex linguistic phenomena and integrate syntax, semantics, and reasoning in an end-to-end application. For example, SHRDLU was able to process "*Find a block which is taller than the one you are holding and put it into the box.*" However, as the systems were based on hand-crafted rules, it became increasingly dif-

ficult to generalize beyond the narrow domains and handle the intricacies of general language.

Rise of machine learning. In the early 1990s, influenced by the successes of statistical techniques in the neighboring speech recognition community, the field of NLP underwent a statistical revolution. Machine learning offered a new paradigm: Collect *examples* of the desired input-output behavior and then fit a statistical model to these examples. The simplicity of this paradigm coupled with the increase in data and computation allowed machine learning to prevail.

What fell out of favor was not only rule-based methods, but also the natural language understanding problems. In the statistical NLP era, much of the community's attention turned to tasks—documentation classification, part-of-speech tagging, and syntactic parsing—which fell short of full end-to-end understanding. Even question answering systems relied less on understanding and more on a shallower analysis coupled with a large collection of unstructured text documents [10], typified by the TREC competitions.

Statistical semantic parsing. The spirit of deep understanding was kept alive by researchers in statistical semantic parsing [36, 24, 33, 37, 19]. A variety of different semantic representations and learning algorithms were employed, but all of these approaches relied on having a labeled dataset of natural language utterances paired with annotated logical forms, for example:

Utterance: *What is the largest prime less than 10?*
Logical form: $\max(\texttt{primes} \cap (-\infty, 10))$

Weak supervision. Over the last few years, two exciting developments have really spurred interest in semantic parsing. The first is reducing the amount of supervision from annotated logical forms to answers [13, 21]:

Utterance: *What is the largest prime less than 10?*
Action: 7

This form of supervision is much easier to obtain via crowdsourcing. Although the logical forms are not observed, they are still modeled as latent variables, which must be inferred from the answer. This results in a more difficult learning problem, but [21] showed that it is possible to solve it without degrading accuracy.

Scaling up. The second development is the scaling up of semantic parsers to more complex domains. Previous semantic parsers had only been trained on limited domains such as US geography, but the creation of broad-coverage knowledge bases such as Freebase [8] set the stage for a new generation of semantic parsers for question answering. Initial systems required annotated logical forms [11], but soon, systems became trainable from answers [4, 18, 5]. Semantic parsers have even been extended beyond fixed knowledge bases to semi-structured tables [26]. With the ability to learn semantic parsers from question-answer pairs, it is easy to collect datasets via crowdsourcing. As a result, semantic parsing datasets have grown by an order of magnitude.

In addition, semantic parsers have been applied to a number of applications outside question answering: robot navigation

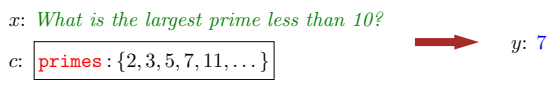

Figure 1: A natural language understanding problem where the goal is to map an utterance x in a context c to an action y.

[29, 2], identifying objects in a scene [23, 15], converting natural language to regular expressions [17], and many others.

Outlook. Today, the semantic parsing community is a vibrant field, but it is still young and grappling with the complexities of the natural language understanding problem. Semantic parsing poses three types of challenges:

- Linguistic: How should we represent the semantics of natural language and construct it compositionally from the natural language?
- Statistical: How can we learn semantic parsers from weak supervision and *generalize* well to new examples?
- Computational: How do we efficiently search over the combinatorially large space of possible logical forms?

The rest of the paper is structured as follows: We first present a general framework for semantic parsing, introducing the key components (Section 2). The framework is pleasantly modular, with different choices of the components corresponding to existing semantic parsers in the literature (Section 3). We describe the datasets (Section 4) and then conclude (Section 5).

2. FRAMEWORK

Natural language understanding problem. In this article, we focus on the following natural language understanding problem: Given an *utterance* x in a *context* c, output the desired *action* y. Figure 1 shows the setup for a question answering application, in which case x is a question, c is a knowledge base, and y is the answer. In a robotics application, x is a command, c represents the robot's environment, and y is the desired sequence of actions to be carried by the robot [29]. To build such a system, assume that we are given a set of n examples $\{(x_i, c_i, y_i)\}_{i=1}^n$. We would like to use these examples to train a model that can generalize to new unseen utterances and contexts.

Semantic parsing components. This article focuses on a statistical semantic parsing approach to the above problem, where the key is to posit an intermediate *logical form* z that connects x and y. Specifically, z captures the semantics of the utterance x, and it also executes to the action y (in the context of c). In our running example, z would be $\texttt{max}(\texttt{primes} \cap (-\infty, 10))$. Our semantic parsing framework consists of the following five components (see Figure 2):

1. **Executor**: computes the denotation (action) $y = [\![z]\!]_c$ given a logical form z and context c. This defines the semantic representation (logical forms along with their denotations).

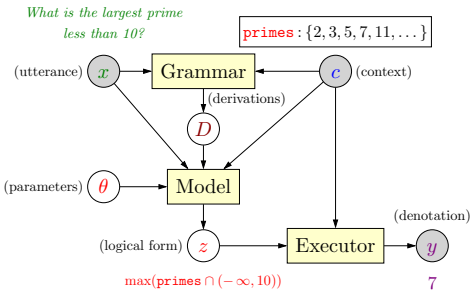

Figure 2: Semantic parsing framework depicting the executor, grammar, and model. The parser and learner are algorithmic components that are responsible for generating the logical form z and parameters θ, respectively.

2. **Grammar**: a set of rules G that produces $D(x, c)$, a set of candidate derivations of logical forms.
3. **Model**: specifies a distribution $p_\theta(d \mid x, c)$ over derivations d parameterized by θ.
4. **Parser**: searches for high probability derivations d under the model p_θ.
5. **Learner**: estimates the parameters θ (and possibly rules in G) given training examples $\{(x_i, c_i, y_i)\}_{i=1}^n$.

We now instantiate each of these components for our running example: "*What is the largest prime less than 10?*"

Executor. Let the semantic representation be the language of mathematics, and the executor is the standard interpretation, where the interpretations of predicates (e.g., `primes`) are given by c. With $c(\texttt{primes}) = \{2, 3, 5, 7, 11, \ldots, \}$, the denotation is $[\![\texttt{primes} \cap (-\infty, 10)]\!]_c = \{2, 3, 5, 7\}$.

Grammar. The grammar G connects utterances to possible *derivations* of logical forms. Formally, the grammar is a set of rules of the form $\alpha \Rightarrow \beta$.[1] Here is a simple grammar for our running example:

(R1)	*prime*	$\Rightarrow$	NP[`primes`]
(R2)	*10*	$\Rightarrow$	NP[10]
(R3)	*less than* NP[z]	$\Rightarrow$	QP[$(-\infty, z)$]
(R4)	NP[z_1] QP[z_2]	$\Rightarrow$	NP[$z_1 \cap z_2$]
(R5)	*largest* NP[z]	$\Rightarrow$	NP[max(z)]
(R6)	*largest* NP[z]	$\Rightarrow$	NP[min(z)]
(R7)	*What is the* NP[z]?	$\Rightarrow$	ROOT[z]

We start with the input utterance and repeatedly apply rules in G. A rule $\alpha \Rightarrow \beta$ can be applied if some span of the utterance matches α, in which case a derivation over the same span with a new syntactic category and logical form according to β is produced. Here is one possible derivation (call it d_1) for our running example:

[1]The standard way context-free grammar rules are written is $\beta \rightarrow \alpha$. Because our rules build logical forms, reversing the arrow is more natural.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\textit{What is the}}{}\quad \dfrac{\textit{largest}\quad \textit{prime}}{\text{NP}[\texttt{primes}]}\text{(R1)}\quad \textit{less than}\quad \dfrac{\textit{10}}{\text{NP}[10]}\text{(R2)}\quad ?}{\text{QP}[(-\infty, 10)]}\text{(R3)}}{\text{NP}[\texttt{primes} \cap (-\infty, 10)]}\text{(R4})}{\text{NP}[\max(\texttt{primes} \cap (-\infty, 10))]}\text{(R5)}}{\text{ROOT}[\max(\texttt{primes} \cap (-\infty, 10))]}\text{(R7)} \tag{1}$$

For example, applying (R3) produces category QP and logical form $(-\infty, 10)$ over span $[5\!:\!7]$ corresponding to "*less than 10*". We stop when we produce the designated ROOT category over the entire utterance. Note that we could have also applied (R6) instead of (R5) to generate the (incorrect) logical form $\min(\texttt{primes} \cap (-\infty, 10))$; let this derivation be d_2. We have $D(x, c) = \{d_1, d_2\}$ here, but in general, there could be exponentially many derivations, and multiple derivations can generate the same logical form. In general, the grammar might contain nonsense rules (R6) that do not reflect ambiguity in language but are rather due to model uncertainty prior to learning.

Model. The model scores the set of candidate derivations generated by the grammar. A common choice used by virtually all existing semantic parsers are log-linear models (generalizations of logistic regressions). In a log-linear model, define a *feature vector* $\phi(x, c, d) \in \mathbb{R}^F$ for each possible derivation d. We can think of each feature as casting a vote for various derivations d based on some coarse property of the derivation. For example, define $F = 7$ features, each counting the number of times a given grammar rule is invoked in d, so that $\phi(x, c, d_1) = [1, 1, 1, 1, 1, 0, 1]$ and $\phi(x, c, d_2) = [1, 1, 1, 1, 0, 1, 1]$.

Next, let $\theta \in \mathbb{R}^F$ denote the *parameter vector*, which defines a weight for each feature representing how reliable that feature is. Their weighted combination $\text{score}(x, c, d) = \phi(x, c, d) \cdot \theta$ represents how good the derivation is. We can exponentiate and normalize these scores to obtain a distribution over derivations:

$$p_\theta(d \mid x, c) = \frac{\exp(\text{score}(x, c, d))}{\sum_{d' \in D(x,c)} \exp(\text{score}(x, c, d'))}. \tag{2}$$

If $\theta = [0, 0, 0, 0, +1, -1, 0]$, then p_θ would assign probability $\frac{\exp(1)}{\exp(1)+\exp(-1)} \approx 0.88$ to d_1 and ≈ 0.12 to d_2.

Parser. Given a trained model p_θ, the parser (approximately) computes the highest probability derivation(s) for an utterance x under p_θ. Assume the utterance x is represented as a sequence of tokens (words). A standard approach is to use a *chart parser*, which recursively builds derivations for each span of the utterance. Specifically, for each category A and span $[i\!:\!j]$ (where $0 \le i < j \le \text{length}(x)$), we loop over the applicable rules in the grammar G and apply each one to build new derivations of category A over $[i\!:\!j]$. For binary rules—those of the form $B\,C \Rightarrow A$ such as (R4), we loop over split points k (where $i < k \le j$), recursively compute derivations $B[z_1]$ over $[i\!:\!k]$ and $C[z_2]$ over $[k\!:\!j]$, and

combine them into a new derivation $A[z]$ over $[i:j]$, where z is determined by the rule; for example, $z = z_1 \cap z_2$ for (R4). The final derivations for the utterance are collected in the ROOT category over span $[0:\text{length}(x)]$.

The above procedure would generate all derivations, which could be exponentially large. Generally, we only wish to compute compute the derivations with high probability under our model p_θ. If the features of p_θ were to *decompose* as a sum over the rule applications in d—that is, $\phi(x, c, d) = \sum_{(r,i,j)\in d} \phi_{\text{rule}}(x, c, r, i, j)$, then we could use dynamic programming: For each category A over $[i:j]$, compute the highest probability derivation. However, in executable semantic parsing, feature decomposition isn't sufficient, since during learning, we also need to incorporate the constraint that the logical form executes to the true denotation ($\mathbb{I}[[\![d.z]\!]_c = y]$); see (6) below. To maintain exact computation in this setting, the dynamic programming state would need to include the entire logical form $d.z$, which is infeasible, since there are exponentially many logical forms. Therefore, *beam search* is generally employed, where we keep only the K sub-derivations with the highest model score based on only features of the sub-derivations. Beam search is not guaranteed to return the K highest scoring derivations, but it is often an effective heuristic.

Learner. While the parser turns parameters into derivations, the learner solves the inverse problem. The dominant paradigm in machine learning is to set up an objective function and optimize it. A standard principle is to maximize the likelihood of the training data $\{(x_i, c_i, y_i)\}_{i=1}^n$. An important point is that we don't observe the correct derivation for each example, but only the action y_i, so we must consider all derivations d whose logical form $d.z$ satisfy $[[\![d.z]\!]_{c_i} = y_i$. This results in the log-likelihood of the observed action y_i:

$$\mathcal{O}_i(\theta) \stackrel{\text{def}}{=} \log \sum_{\substack{d\in D(x_i,c_i) \\ [[\![d.z]\!]_{c_i}=y_i}} p_\theta(d \mid x_i, c_i). \quad (3)$$

The final objective is then simply the sum across all n training examples:

$$\mathcal{O}(\theta) \stackrel{\text{def}}{=} \sum_{i=1}^n \mathcal{O}_i(\theta), \quad (4)$$

The simplest approach to maximize $\mathcal{O}(\theta)$ is to use *stochastic gradient descent* (SGD), an iterative algorithm that takes multiple passes (e.g., say 5) over the training data and makes the following update on example i:

$$\theta \leftarrow \theta + \eta \nabla \mathcal{O}_i(\theta), \quad (5)$$

where η is a step size that governs how aggressively we want to update parameters (e.g., $\eta = 0.1$). In the case of log-linear models, the gradient has a nice interpretable form:

$$\nabla \mathcal{O}_i(\theta) = \sum_{d\in D(x_i,c_i)} (q(d) - p_\theta(d \mid x_i, c_i))\phi(x_i, c_i, d), \quad (6)$$

where $q(d) \propto p_\theta(d \mid x_i, c_i)\mathbb{I}[[\![d.z]\!]_{c_i} = y_i]$ is the model distribution p_θ over derivations d, but restricted to ones consistent with y_i. The gradient pushes θ to put more probability mass on q and less on p_θ. For example, if p_θ assigns probabilities $[0.2, 0.4, 0.1, 0.3]$ to four derivations and the middle two derivations are consistent, then q assigns probabilities $[0, 0.8, 0.2, 0]$.

The objective function $\mathcal{O}(\theta)$ is not concave, so SGD is at best guaranteed to converge to a local optimum, not a global one. Another problem is that we cannot enumerate all derivations $D(x_i, c_i)$ generated by the grammar, so we approximate this set with the result of beam search, which yields K candidates (typically $K = 200$); p_θ is normalized over this set. Note that this candidate set depends on the current parameters θ, resulting a heuristic approximation of the gradient $\nabla \mathcal{O}_i$.

Summary. We have covered the components of a semantic parsing system. Observe that the components are relatively loosely coupled: The executor is concerned purely with what we want to express independent of how it would be expressed in natural language. The grammar describes how candidate logical forms are constructed from the utterance but does not provide algorithmic guidance nor specify a way to score the candidates. The model focuses on a particular derivation and defines features that could be helpful for predicting accurately. The parser and the learner provide algorithms largely independent of semantic representations. This modularity allows us to improve each component in isolation.

3. REFINING THE COMPONENTS

Having toured the components of a semantic parsing system, we now return to each component and discuss the key design decisions and possibilities for improvement.

3.1 Executor

By describing an executor, we are really describing the language of the logical form. A basic textbook representation of language is *first-order logic*, which can be used to make quantified statements about relations between objects. For example, "*Every prime greater than two is odd.*" would be expressed in first-order logic as $\forall x.\texttt{prime}(x) \wedge \texttt{more}(x, 2) \rightarrow \texttt{odd}(x)$. Here, the context c is a model (in the model theory sense), which maps predicates to sets of objects or object pairs. The execution of the above logical form with respect to the standard mathematical context would be `true`. [7] gives a detailed account on how first-order logic is used for natural language semantics.

First-order logic is reasonably powerful, but it fails to capture some common phenomena in language. For example, "*How many primes are less than 10?*" requires constructing a set and manipulating it and thus goes beyond the power of first-order logic. We can instead augment first-order logic with constructs from *lambda calculus*. The logical form corresponding to the above question would be $\texttt{count}(\lambda x.\texttt{prime}(x) \wedge \texttt{less}(x, 10))$, where the λ operator can be thought of as constructing a set of all x that satisfy the condition; in symbols, $[[\![\lambda x.f(x)]\!]_c = \{x : [[\![f(x)]\!]_c = \texttt{true}\}$. Note that `count` is a higher-order functions that takes a function as an argument.

Another logical language, which can be viewed as syntactic sugar for lambda calculus, is *lambda dependency-based semantics (DCS)* [20]. In lambda DCS, the above logical form would be $\texttt{count}(\texttt{prime} \sqcap (\texttt{less}.10))$, where the constant 10 represent $\lambda x.(x = 10)$, the intersection operator $z_1 \sqcap z_2$ represents $\lambda x.z_1(x) \wedge z_2(x)$, and the join operator $r.z$ represents $\lambda x.\exists y.r(x, y) \wedge z(y)$.

Lambda DCS is "lifted" in the sense that operations combine functions from objects to truth values (think sets) rather

than truth values. As a result, lambda DCS logical forms partially eliminate the need for variables. Noun phrases in natural language (e.g., "*prime less than 10*") also denote sets. Thus, lambda DCS arguably provides a transparent interface with natural language.

From a linguistic point of view, the logical language seeks primarily to model natural language phenomena. From an application point of view, the logical language dictates what actions we support. It is thus common to use application-specific logical forms, for example, regular expressions [17]. Note that the other components of the framework are largely independent of the exact logical language used.

3.2 Grammar

Recall that the goal of the grammar in this article is just to define a set of candidate derivations for each utterance and context. Note that this is in contrast to a conventional notion of a grammar in linguistics, where the goal is to precisely characterize the set of valid sentences and interpretations. This divergence is due to two factors: First, we will learn a statistical model over the derivations generated by the grammar anyway, so the grammar can be simple and coarse. Second, we might be interested in application-specific logical forms. In a flight reservation domain, the logical form we wish to extract from "*I'm in Boston and would like to go to Portland*" is $\texttt{flight} \sqcap \texttt{from.Boston} \sqcap \texttt{to.Portland}$, which is certainly not the full linguistic meaning of the utterance, but suffices for the task at hand. Note that the connection here between language and logic is less direct compared to "*prime less than 10*" $\Rightarrow \texttt{prime} \sqcap (\texttt{less}.10)$.

CCG. One common approach to the grammar in semantic parsing is *combinatory categorial grammar* (CCG) [28], which had been developed extensively in linguistics before it was first used for semantic parsing [37]. CCG is typically coupled with logical forms in lambda calculus. Here is an example of a CCG derivation:

$$\dfrac{\dfrac{\dfrac{\textit{prime}}{\mathrm{N}[\lambda x.\texttt{prime}(x)]} \quad \dfrac{\dfrac{\textit{less than}}{(\mathrm{N}\backslash\mathrm{N})/\mathrm{NP}[\lambda y.\lambda f.\lambda x.f(x) \wedge \texttt{less}(x,y)} \quad \dfrac{\textit{10}}{\mathrm{NP}[10]}}{\mathrm{N}\backslash\mathrm{N}[\lambda f.\lambda x.f(x) \wedge \texttt{less}(x,10)]}\,(>)}{\mathrm{N}[\lambda x.\texttt{prime}(x) \wedge \texttt{less}(x,10)]}\,(<)} \tag{7}$$

Syntactic categories in CCG include nouns (N) denoting sets and noun phrases (NP) denoting objects. Composite categories ((N\N)/NP) represent functions of multiple arguments, but where the directionality of the slashes indicate the location of the arguments. Most rules in CCG are lexical (e.g., $[\textit{prime} \Rightarrow \mathrm{N}[\lambda x.\texttt{prime}(x)]$), which mention particular words. The rest of the rules are glue; for example, we have a forward application (>) and backward application (<) rule:

(>)	$A/B[f]$	$B[x]$	$\Rightarrow$	$A[f(x)]$
(<)	$B[x]$	$A\backslash B[f]$	$\Rightarrow$	$A[f(x)]$

It is common to use other combinators which both handle more complex linguistic phenomena such as NP coordination "*integers that divide 2 or 3*" as well as ungrammatical language [38], although these issues can also be handled by having a more expansive lexicon [19].

Crude rules. In CCG, the lexical rules can become quite complicated. An alternative approach that appears mostly in work on lambda DCS is to adopt a much cruder grammar whose rules correspond directly to the lambda DCS constructions, join and intersect. This results in the following derivation:

$$\dfrac{\dfrac{\dfrac{\textit{prime}}{\mathrm{N}[\texttt{prime}]} \quad \dfrac{\dfrac{\textit{less than}}{\mathrm{N}|\mathrm{N}[\texttt{less}]} \quad \dfrac{\textit{10}}{\mathrm{N}[10]}}{\mathrm{N}[\texttt{less}.10]}\,(\text{join})}{\mathrm{N}[\texttt{prime} \sqcap \texttt{less}.10]}\,(\text{intersect})} \tag{8}$$

(join)	$\mathrm{N}\|\mathrm{N}[r]$	$\mathrm{N}[z]$	$\Rightarrow$	$\mathrm{N}[r.z]$
(intersect)	$\mathrm{N}[z_1]$	$\mathrm{N}[z_2]$	$\Rightarrow$	$\mathrm{N}[z_1 \sqcap z_2]$

The lack of constraints permits the derivation of other logical forms such as $\texttt{10} \sqcap \texttt{less.prime}$, but these incorrect logical forms can be ruled out statistically via the model. The principle is to keep the lexicon simple and lean more heavily on features (whose weights are learned from data) to derive drive the selection of logical forms.

Floating rules. While this new lexicon is simpler, the bulk of the complexity comes from having it in the first place. Where does the rules $[\textit{prime} \Rightarrow \mathrm{N}|\mathrm{N} : \texttt{prime}]$ and $[\textit{less than} \Rightarrow \mathrm{N}|\mathrm{N} : \texttt{less}]$ come from? We can reduce the lexicon even further by introducing *floating* rules $[\emptyset \Rightarrow \mathrm{N}|\mathrm{N} : \texttt{prime}]$ and $[\emptyset \Rightarrow \mathrm{N}|\mathrm{N} : \texttt{less}]$, which can be applied in the absence of any lexical trigger. This way, the utterance "*primes smaller than 10*" would also be able to generate $\texttt{prime} \sqcap \texttt{less}.10$, where the predicates `prime` and `less` are not anchored to any specific words.

While this relaxation may seem linguistically blasphemous, recall that the purpose of the grammar is to merely deliver a set of logical forms, so floating rules are quite sensible provided we can keep the set of logical forms under control. Of course, since the grammar is so flexible, even more nonsensical logical forms are generated, so we must lean heavily on the features to score the derivations properly. When the logical forms are simple and we have strong type constraints, this strategy can be quite effective [5, 30, 26].

The choice of grammar is arguably the most important component of a semantic parser, as it most directly governs the tradeoff between expressivity and statistical/computational complexity. We have explored three grammars, from a tightly-constrained CCG to a very laissez faire floating grammar. CCG is natural for capturing complex linguistic phenomena in well-structured sentences, whereas for applications where the utterances are noisy but the logical forms are simple, more flexible grammars are appropriate. In practice, one would probably want to mix and match depending on the domain.

3.3 Model

At the core of a statistical model is a function $\text{score}(x, c, d)$ that judges how good a derivation d is with respect to the

utterance x and context c. In Section 2, we described a simple log-linear model (2) in which $\text{score}(x, c, d) = \phi(x, c, d) \cdot \theta$, and with simple rule features. There are two ways to improve the model: use more expressive features and and use a non-linear scoring function.

Most existing semantic parsers stick with a linear model and leverage more targeted features. One simple class of features $\{\phi_{a,b}\}$ is equal to the number of times word a appears in the utterance and predicate b occurs in the logical form; if the predicate b is generated by a floating rule, then a is allowed to range over the entire utterance; otherwise, it must appear in the span of b.

The above features are quite numerous, which provides flexibility but require a lot of data. Note that logical predicates (e.g., `birthplace`) and the natural language (e.g., *born in*) often share a common vocabulary (English words). One can leverage this structure by defining a few features based on statistics from large corpora, external lexical resources, or simply string overlap. For example, ϕ_{match} might be the number of word-predicate pairs with that match almost exactly. This way, the lexical mapping need not be learned from scratch using only the training data [4, 18].

A second class of useful features are based on the denotation $[\![d.z]\!]_c$ of the predicted logical form. For example, in a robot control setting, a feature would encode whether the predicted actions are valid in the environment c (picking up a cup requires the cup to be present). These features make it amply clear that semantic parsing is not simply a natural language problem but one that involves jointly reasoning about the non-linguistic world. In addition, we typically include type constraints (features with weight $-\infty$) to prune out ill-formed logical forms such as `birthplace`.4 that are licensed the grammar.

One could also adopt a non-linear scoring function, which does not require any domain knowledge about semantic parsing. For example, one could use a simple one-layer neural network, which takes a weighted combination of m nonlinear basis functions: $\text{score}(x, c, d) = \sum_{i=1}^{m} \alpha_i \tanh(\phi(x, c, d) \cdot w_i)$. The parameters of the model that we would learn are then the m top-level weights $\{\alpha_i\}$ and the mF bottom-level weights $\{w_{ij}\}$; recall that F is the number of features ($\phi(x, c, d) \in \mathbb{R}^F$). With more parameters, the model becomes more powerful, but also requires more data to learn.

3.4 Parsing

Most semantic parsing algorithms are based on chart parsing, where we recursively generate a set of candidate derivations for each syntactic category (e.g., NP) and span (e.g., [3:5]). There are two disadvantages of chart parsing: First, it does not easily support incremental contextual interpretation: The features of a span $[i : j]$ can only depend on the sub-derivations in that span, not on the derivations constructed before i. This makes anaphora (resolution of "*it*") difficult to model. A solution is to use shift-reduce parsing rather than chart parsing [40]. Here, one parses the utterance from left to right, and new sub-derivations can depend arbitrarily on the sub-derivations constructed thus far.

A second problem is that the chart parsing generally builds derivations in some fixed order—e.g., of increasing span size. This causes the parser to waste equal resources on non-promising spans which are unlikely to participate in the final derivation that is chosen. This motivates agenda-based parsing, in which derivations that are deemed more promising by the model are built first [6].

3.5 Learner

In learning, we are given examples of utterance-context-response triples (x, c, y). There are two aspects of learning: inducing the grammar rules and estimating the model parameters. It is important to remember that practical semantic parsers do not do everything from scratch, and often the hard-coded grammar rules are as important as the training examples. First, some lexical rules that map named entities (e.g., [*paris* ⇒ `ParisFrance`]), dates, and numbers are generally assumed to be given [37], though we need not assume that these rules are perfect [21]. These rules are also often represented implicitly [21, 4].

How the rest of the grammar is handled varies across approaches. In CCG-style approach, inducing lexical rules is an important part of learning. In [37], a procedure called GENLEX is used to generate candidate lexical rules from a utterance-logical form pair (x, z). A more generic induction algorithm based on higher-order unification does not require any initial grammar [19]. [33] use machine translation ideas to induce a synchronous grammar (which can also be used to generate utterances from logical forms). However, all these lexicon induction methods require annotated logical forms z. In approaches that learn from denotations y [21, 4], an initial crude grammar is used to generate candidate logical forms, and rest of the work is done by the features.

As we discussed earlier, parameter estimation can be performed by stochastic gradient descent on the log-likelihood; similar objectives based on max-margin are also possible [22]. It can be helpful to also add an L_1 regularization term $\lambda\|\theta\|_1$, which encourages feature weights to be zero, which produces a more compact model that generalizes better [5]. In addition, one can use AdaGrad [14], which maintains a separate step size for each feature. This can improve stability and convergence.

4. DATASETS AND RESULTS

In a strong sense, datasets are the main driver of progress for statistical approaches. We will now survey some of the existing datasets, describe their properties, and discuss the state-of-the-art.

The Geo880 dataset [36] drove nearly a decade of semantic parsing research. This dataset consists of 880 questions and database queries about US geography (e.g., "*what is the highest point in the largest state?*"). The utterances are compositional, but the language is clean and the domain is limited. On this dataset, learning from logical forms [18] and answers [21] both achieve around 90% accuracy.

The ATIS-3 dataset [38] consists of 5418 utterances paired with logical forms (e.g., "*show me information on american airlines from fort worth texas to philadelphia*"). The utterances contain more disfluencies and flexible word order compared with Geo880, but they are logically simpler. As a result, slot filling methods have been a successful paradigm in the spoken language understanding community for this

domain since the 1990s. The best reported result is based on semantic parsing and obtains 84.6% accuracy [38].

The Regexp824 dataset [17] consists of 824 natural language and regular expression pairs (e.g., "*three letter word starting with 'X'*"). The main challenge here is that there are many logically equivalent regular expressions, some aligning better to the natural language than others. [17] uses semantic unification to test for logical form equivalence and obtains 65.6% accuracy.

The Free917 dataset [11] consists of 917 examples of question-logical form pairs that can be answered via Freebase, e.g. "*how many works did mozart dedicate to joseph haydn?*" The questions are logically less complex than those in the semantic parsing datasets above, but introduces the new challenge of scaling up to many more predicates (but is in practice manageable by assuming perfect named entity resolution and leveraging the strong type constraints in Freebase). The state-of-the-art accuracy is 68% accuracy [18].

WebQuestions [4] is another dataset on Freebase consisting of 5810 question-answer pairs (no logical forms) such as "*what do australia call their money?*" Like Free917, the questions are not very compositional, but unlike Free917, they are real questions asked by people on the Web independent from Freebase, so they are more realistic and more varied. Because the answers are required to come from a single Freebase page, a noticeable fraction of the answers are imperfect. The current state-of-the-art is 52.5% [35].

The goal of WikiTableQuestions [26] is to extend question answering beyond Freebase to HTML tables on Wikipedia, which are semi-structured. The dataset consists of 22033 question-table-answer triples (e.g., "*how many runners took 2 minutes at the most to run 1500 meters?*"), where each question can be answered by aggregating information across the table. At test time, we are given new tables, so methods must learn how to generalize to new predicates. The result on this new dataset is 37.1%.

[30] proposed a new recipe for quickly using crowdsourcing to generate new compositional semantic parsing datasets consisting of question-logical form pairs. Using this recipe, they created eight new datasets in small domains consisting of 12602 total question-answer pairs, and achieved an average accuracy on across datasets of 58.8%.

[12] introduced a dataset of 706 navigation instructions (e.g., "*facing the lamp go until you reach a chair*") in a simple grid world. Each instruction sequence contains multiple sentences with various imperative and context-dependent constructions not found in previous datasets. [2] obtained 65.3% sentence-level accuracy.

5. DISCUSSION

We have presented a semantic parsing framework for the problem of natural language understanding. Going forward, the two big questions are (i) how to represent the semantics of language and (ii) what supervision to use to learn the semantics from.

Alternative semantic representations. One of the main difficulties with semantic parsing is the divergence between the structure of the natural language and the logical forms—purely compositional semantics will not work. This has led to some efforts to introduce an intermediate layer between utterances and logical forms. One idea is to use general paraphrasing models to map input utterances to the "canonical utterances" of logical forms [5, 30]. This reduces semantic parsing to a text-only problem for which there is much more data and resources.

One could also use domain-general logical forms that capture the basic predicate-argument structures of sentences [18]. Abstract meaning representation (AMR) [3] is one popular representation backed by an extension linguistic annotation effort. Multiple AMR parsers have been developed, including one based on CCG [1]. While these representations offer broad coverage, solving downstream understanding tasks still require additional work (e.g., inferring that "*population of X*" is synonymous with "*number of people living in X*"). In contrast, *executable* semantic parsing operates on the full pipeline from utterance to task output, but compromises on domain-generality. This is partially inevitable, as any understanding task requires some domain-specific knowledge or grounding. Designing the best general representation that supports many downstream tasks remains an open challenge.

Alternative supervision. Much of the progress in semantic parsing has been due to being able to learn from weaker supervision. In the framework we presented, this supervision are the desired actions y (e.g., answers to questions). One can use a large corpus of text to exploit even weaker supervision [16, 27]. More generally, one can think about language interpretation in a reinforcement learning setting [9], where an agent who presented with an utterance in some context performs some action, and receives a corresponding reward signal. This framework highlights the importance of context-dependence in language interpretation [39, 2].

Due to their empirical success, there has been a recent surge of interest in using recurrent neural networks and their extensions for solving NLP tasks such as machine translation and question answering [35, 31]. These methods share the same spirit of end-to-end utterance-to-behavior learning as executable semantic parsing, but they do not explicitly separate parsing from execution. This makes them architecturally simpler than semantic parsers, but they are more data hungry and it is unclear whether they can learn to perform complex logical reasoning in a generalizable way. Nonetheless, it is quite likely that these methods will play an important role in the story of language understanding.

Outlook. This is an exciting time for semantic parsing and natural language understanding. As natural language interfaces (e.g., Siri) become more ubiquitous, the demand for deep understanding will continue to grow. At the same time, there is a fresh wave of ambition pushing the limits of what can be machine learnable. The confluence of these two factors will likely generate both end-user impact as well as new insights into the nature of language and learning.

6. REFERENCES

[1] Y. Artzi and K. L. L. Zettlemoyer. Broad-coverage CCG semantic parsing with AMR. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

[2] Y. Artzi and L. Zettlemoyer. Weakly supervised learning of

semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics (TACL)*, 1:49–62, 2013.

[3] L. BanaRescu, C. B. S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider. Abstract meaning representation for sembanking. In *7th Linguistic Annotation Workshop and Interoperability with Discourse*, 2013.

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

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

[6] J. Berant and P. Liang. Imitation learning of agenda-based semantic parsers. *Transactions of the Association for Computational Linguistics (TACL)*, 0, 2015.

[7] P. Blackburn and J. Bos. *Representation and Inference for Natural Language: A First Course in Computational Semantics*. CSLI Publishers, 2005.

[8] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *International Conference on Management of Data (SIGMOD)*, pages 1247–1250, 2008.

[9] S. Branavan, H. Chen, L. S. Zettlemoyer, and R. Barzilay. Reinforcement learning for mapping instructions to actions. In *Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 82–90, 2009.

[10] E. Brill, S. Dumais, and M. Banko. An analysis of the AskMSR question-answering system. In *Association for Computational Linguistics (ACL)*, pages 257–264, 2002.

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

[12] D. L. Chen and R. J. Mooney. Learning to interpret natural language navigation instructions from observations. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 859–865, 2011.

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

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

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

[16] J. Krishnamurthy and T. Mitchell. Weakly supervised training of semantic parsers. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL)*, pages 754–765, 2012.

[17] N. Kushman and R. Barzilay. Using semantic unification to generate regular expressions from natural language. In *Human Language Technology and North American Association for Computational Linguistics (HLT/NAACL)*, pages 826–836, 2013.

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

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

[20] P. Liang. Lambda dependency-based compositional semantics. *arXiv*, 2013.

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

[22] P. Liang and C. Potts. Bringing machine learning and compositional semantics together. *Annual Reviews of Linguistics*, 1(1):355–376, 2015.

[23] C. Matuszek, N. FitzGerald, L. Zettlemoyer, L. Bo, and D. Fox. A joint model of language and perception for grounded attribute learning. In *International Conference on Machine Learning (ICML)*, pages 1671–1678, 2012.

[24] S. Miller, D. Stallard, R. Bobrow, and R. Schwartz. A fully statistical approach to natural language interfaces. In *Association for Computational Linguistics (ACL)*, pages 55–61, 1996.

[25] R. Montague. The proper treatment of quantification in ordinary English. In *Approaches to Natural Language*, pages 221–242, 1973.

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

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

[28] M. Steedman. *The Syntactic Process*. MIT Press, 2000.

[29] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2011.

[30] Y. Wang, J. Berant, and P. Liang. Building a semantic parser overnight. In *Association for Computational Linguistics (ACL)*, 2015.

[31] J. Weston, S. Chopra, and A. Bordes. Memory networks. In *International Conference on Learning Representations (ICLR)*, 2015.

[32] T. Winograd. *Understanding Natural Language*. Academic Press, 1972.

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

[34] W. A. Woods, R. M. Kaplan, and B. N. Webber. The lunar sciences natural language information system: Final report. Technical report, BBN Report 2378, Bolt Beranek and Newman Inc., 1972.

[35] W. Yih, M. Chang, X. He, and J. Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Association for Computational Linguistics (ACL)*, 2015.

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

[37] L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence (UAI)*, pages 658–666, 2005.

[38] L. S. Zettlemoyer and M. Collins. 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, 2007.

[39] L. S. Zettlemoyer and M. Collins. Learning context-dependent mappings from sentences to logical form. In *Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, 2009.

[40] K. Zhao and L. Huang. Type-driven incremental semantic parsing with polymorphism. In *North American Association for Computational Linguistics (NAACL)*, 2015.