Semantic Parsing to Probabilistic Programs for Situated Question Answering

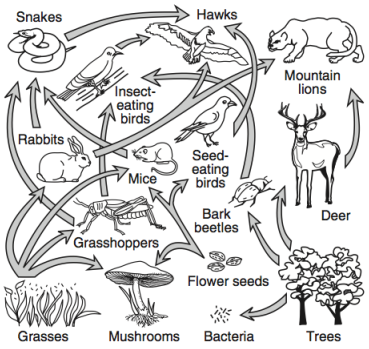
Jayant Krishnamurthy and **Oyvind Tafjord** and **Aniruddha Kembhavi**
Allen Institute for Artificial Intelligence
jayantk,oyvindt,anik@allenai.org

Abstract

Situated question answering is the problem of answering questions about an environment such as an image or diagram. This problem requires jointly interpreting a question and an environment using background knowledge to select the correct answer. We present Parsing to Probabilistic Programs (P^3), a novel situated question answering model that can use background knowledge and global features of the question/environment interpretation while retaining efficient approximate inference. Our key insight is to treat semantic parses as probabilistic programs that execute nondeterministically and whose possible executions represent environmental uncertainty. We evaluate our approach on a new, publicly-released data set of 5000 science diagram questions, outperforming several competitive classical and neural baselines.

1 Introduction

Situated question answering is a challenging problem that requires reasoning about uncertain interpretations of both a question and an environment together with background knowledge to determine the answer. To illustrate these challenges, consider the 8th grade science diagram questions in Figure 1, which are motivated by the Aristo project (Clark and Etzioni, 2016). These questions require both computer vision to interpret the diagram and compositional question understanding. These components, being imperfect, introduce uncertainty that must be jointly reasoned about to avoid implausible interpretations. These uncertain interpretations must further be combined with background knowledge, such as the definition of a "predator," to determine the correct answer.



1. According to the given food chain, what is the number of organisms that eat deer? (A) 3 (B) 2 (C) 4 **(D) 1**

2. Which organism is both predator and prey? (A) Bark Beetles **(B) Insect-eating birds** (C) Deer (D) Hawks

3. Based on the given food web, what would happen if there were no insect-eating birds? **(A) The grasshopper population would increase.** (B) The grasshopper population would decrease. (C) There would be no change in grasshopper number.

Figure 1: Example food web questions. A food web depicts a collection of organisms in an ecosystem with an arrow from organism x to y indicating that y eats x. Questions may require counting (1), knowing animal roles (2) and reasoning about population changes (3).

The challenges of situated question answering have not been completely addressed by prior work. Early "possible worlds" models (Matuszek et al., 2012; Krishnamurthy and Kollar, 2013; Malinowski and Fritz, 2014) were capable of compositional question understanding and using background knowledge, but did not jointly reason about environ-

Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 160–170,
Austin, Texas, November 1-5, 2016.

ment/question uncertainty. These models also used unscalable inference algorithms for reasoning about the environment, despite the lack of joint reasoning. More recent neural models (Antol et al., 2015; Malinowski et al., 2015; Yang et al., 2015) are incapable of using background knowledge and it remains unclear to what extent these models can represent compositionality in language.

We present Parsing to Probabilistic Programs (P^3), a novel approach to situated question answering that addresses these challenges. It is motivated by two observations: (1) situated question answering can be formulated as semantic parsing with an execution model that is a *learned function of the environment*, and (2) *probabilistic programming* is a natural and powerful method for specifying the space of permissible execution models and learning over it. In P^3, we define a domain theory for the task as a probabilistic program, then train a joint loglinear model to semantically parse questions to logical forms in this theory and execute them in an environment. Importantly, the model includes global features over parsing and execution that enable it to avoid unlikely joint configurations. P^3 leverages semantic parsing to represent compositionality in language and probabilistic programming to specify background knowledge and perform linear-time approximate inference over the environment.

We present an experimental evaluation of P^3 on a new data set of 5000 food web diagram questions (Figure 1). We compare our approach to several baselines, including possible worlds and neural network approaches, finding that P^3 outperforms both. An ablation study demonstrates that global features help the model achieve high accuracy. We also demonstrate that P^3 improves accuracy on a previously published data set. Finally, we have released our data and code to facilitate further research.

2 Prior Work

Situated question answering is often formulated in terms of parsing both the question and environment into a common meaning representation where they can be combined to select the answer. This general approach has been implemented using different meaning representations:

Possible world models use a logical meaning representation defined by a knowledge base schema. These models train a semantic parser to map questions to queries and an environment model to map environments to knowledge bases in this schema. Executing the queries against the knowledge bases produces answers. These models assume that the parser and environment model are independent and furthermore that the knowledge base consists of independent predicate instances (Matuszek et al., 2012; Krishnamurthy and Kollar, 2013; Malinowski and Fritz, 2014). Despite these strong independence assumptions, these models have intractable inference. An exception is Seo et al., (2015) who incorporate hard constraints on the joint question/environment interpretation; however, this approach does not generalize to soft constraints or arbitrary logical forms. In some work only the environment model is learned (Kollar et al., 2010; Tellex et al., 2011; Howard et al., 2014b; Howard et al., 2014a; Berant et al., 2014; Krishnamurthy and Mitchell, 2015).

Neural networks use a vector meaning representation that encodes both the question and environment as vectors. These networks have mostly been applied to visual question answering (Antol et al., 2015), where many architectures have been proposed (Malinowski et al., 2015; Yang et al., 2015; Fukui et al., 2016). It is unclear to what extent these networks can represent compositionality in language using their vector encodings. Dynamic Neural Module Networks (Andreas et al., 2016a; Andreas et al., 2016b) are the exception to the above generalization. This approach constructs a neural network to represent the meaning of the question via semantic parsing, then executes this network against the image to produce an answer. Our approach is similar except that we construct and execute a probabilistic program. Advantages of our approach are that it naturally represents the discrete structure of food webs and can use background knowledge.

Preliminaries for our work are semantic parsing and probabilistic programming. Semantic parsing translates natural language questions into executable logical forms and has been used in applications such as question answering against a knowledge base (Zelle and Mooney, 1993; Zettlemoyer and Collins, 2005; Liang et al., 2011; Kwiatkowski et al., 2013; Berant et al., 2013; Reddy et al., 2014; Yih et al., 2015; Xu et al., 2016), direction following

(Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013), and information extraction (Krishnamurthy and Mitchell, 2012; Choi et al., 2015). Semantic parsing alone is insufficient for situated question answering because it does not interpret the environment; many of the above approaches use semantic parsing as a component in a larger model.

Probabilistic programming languages extend programming languages with primitives for nondeterministic choice (McCarthy, 1963; Goodman and Stuhlmüller, 2014). We express logical forms in a probabilistic variant of Scheme similar to Church (Goodman et al., 2008); however, this paper uses Python-like pseudocode for clarity. The language has a single choice primitive called `choose` that nondeterministically returns one of its arguments. For example `choose(1,2,3)` can execute three ways, returning either 1, 2, or 3. Multiple calls to `choose` can be combined. For example, `choose(1,2)+choose(1,2)` adds two nondeterministically chosen values, and therefore has four executions that return 2, 3, 3 and 4. Each execution also has a probability; in our case, these probabilities are assigned by a trained model given the environment and not explicitly specified in the program.

3 Parsing to Probabilistic Programs (P^3)

The P^3 model is motivated by two observations. The first is that situated question answering can be formulated as semantic parsing with an execution model that is a learned function of the environment. Consider the first question in Figure 1. The meaning of this question could be represented by a logical form such as $\text{COUNT}(\lambda x.\text{EATS}(x, \text{DEER}))$, which we could train a semantic parser to predict given a suitable *domain theory* of functions such as COUNT and EATS. However, the information required to execute this logical form and answer the question must be extracted from the diagram. Specifically, $\text{EATS}(x, y)$ depends on whether an arrow is present between x and y, which we must train a vision model to determine. Thus, EATS should be a *learned function of the environment*.

This first observation suggests a need for a formalism for representing uncertainty and performing learning over the domain theory's functions. Our second observation is that probabilistic programming is a natural fit for this task. In this paradigm, the domain theory is a probabilistic program that defines the information to be extracted from the environment by using `choose`. To a first approximation, the diagram question theory includes:

```
def eats(x, y)
  choose(true, false)
```

Logical forms are then probabilistic programs, each of whose possible executions represents a different interpretation of the environment. For example, executing $\text{EATS}(\text{LION}, \text{DEER})$ hits the `choose` in the above definition, resulting in two executions where the lion either eats or does not eat the deer. In the COUNT example above, each execution represents a different set of animals that eat the deer. To learn the correct environment interpretation, we train an execution model to assign a probability to each execution given features of the environment. Using probabilistic programming enables us to combine learned functions, such as EATS, with background knowledge functions, such as COUNT, and also facilitates inference.

According to these observations, applying P^3 has two steps. The first step is to define an appropriate domain theory. This theory is the main design decision in instantiating P^3 and provides a powerful way to encode domain knowledge. The second step is to train a loglinear model consisting of a semantic parser and an execution model. This model learns to semantically parse questions into logical forms in the theory and execute them in the environment to answer questions correctly. We defer discussion of the diagram question domain theory to Section 4 and focus on the loglinear model in this section.

3.1 Model Overview

The input to the P^3 model is a question and an environment and its output is a denotation, which is a formal answer to the question. P^3 is a loglinear model with two factors: a semantic parser and an execution model. The semantic parser scores syntactic parses and logical forms for the question. These logical forms are probabilistic programs with multiple possible executions (specified by the domain theory), each of which may return a different denotation. The execution model assigns a score to each of these executions given the environment. Formally,

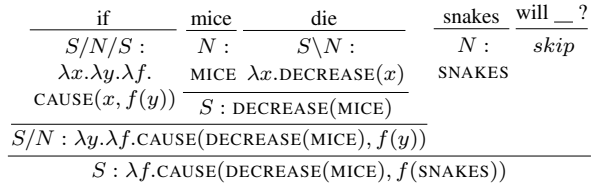


Figure 2: Example CCG parse of a question as predicted by the semantic parser f_{p}. The logical form ℓ for the question is shown on the bottom line.

the model predicts a denotation γ for a question q in an environment v using three latent variables:

$$P(\gamma|v,q;\theta) = \sum_{e,\ell,t} P(e,\ell,t|v,q;\theta)\mathbf{1}(\text{ret}(e)=\gamma)$$

$$P(e,\ell,t|v,q;\theta) = \frac{1}{Z_{q,v}} f_{\text{ex}}(e,\ell,v;\theta_{\text{ex}}) f_{\text{p}}(\ell,t,q;\theta_{\text{p}})$$

The model is composed of two factors. f_{p} represents the semantic parser that scores logical forms ℓ and syntactic parse trees t given question q and parameters θ_{p}. f_{ex} represents the execution model. Given parameters θ_{ex}, this factor assigns a score to a logical form ℓ and its execution e in environment v. The denotation γ, i.e., the formal answer to the question, is simply the value returned by e. $Z_{q,v}$ represents the model's partition function. The following sections describe these factors in more detail.

3.2 Semantic Parser

The factor f_{p} represents a Combinatory Categorial Grammar (CCG) semantic parser (Zettlemoyer and Collins, 2005) that scores logical forms for a question. Given a lexicon[1] mapping words to syntactic categories and logical forms, CCG defines a set of possible syntactic parses t and logical forms ℓ for a question q. Figure 3.2 shows an example CCG parse. f_{p} is a loglinear model over parses (ℓ, t):

$$f_{\text{p}}(\ell,t,q;\theta_{\text{p}}) = \exp\{\theta_{\text{p}}^T \phi(\ell,t,q)\}$$

The function ϕ maps parses to feature vectors. We use a rich set of features similar to those for syntactic CCG parsing (Clark and Curran, 2007); a full description is provided in an online appendix.

3.3 Execution Model

The factor f_{ex} is a loglinear model over the executions of a logical form given an environment. Logical forms in P^3 are probabilistic programs with a *set* of possible executions, where each execution e is a sequence, $e = [e_0, e_1, e_2, ..., e_n]$. e_0 is the program's starting state, e_i represents the state immediately after the ith call to `choose`, and e_n is the state at termination. The score of an execution is:

$$f_{\text{ex}}(e,\ell,v;\theta_{\text{ex}}) = \prod_{i=1}^{n} \exp\{\theta_{\text{ex}}^T \phi(e_{i-1}, e_i, \ell, v)\}$$

In the above equation, θ_{ex} represents the model's parameters and ϕ represents a feature function that produces a feature vector for the difference between sequential program states e_{i-1} and e_i given environment v and logical form ℓ. ϕ can include arbitrary features of the execution, logical form and environment, which is important, for example, to detect cycles in a food web (Section 4.3).

3.4 Inference

P^3 is designed to rely on approximate inference: our goal is to use rich features to accurately make local decisions, as in linear-time parsers (Nivre et al., 2006). We perform approximate inference using a two-stage beam search. Given a question q, the first stage performs a beam search over CCG parses to produce a list of logical forms scored by f_{p}. This step is performed by using a CKY-style chart parsing algorithm then marginalizing out the syntactic parses. The second stage performs a beam search over executions of each logical form. The space of possible executions of a logical form is a tree (Figure 4.2) where each internal node represents a partial execution up to a `choose` call. The search maintains a beam of partial executions at the same depth, and each iteration advances the beam to the next depth, discarding the lowest-scoring executions according to f_{ex} to maintain a fixed size beam. This procedure runs in time linear to the number of `choose` calls. We implement the search by rewriting the probabilistic program into continuation-passing style, which allows `choose` to be implemented as a function that adds multiple continuations to the search queue; we refer the reader to Goodman and Stuhlmüller (2014) for details. Our experiments use a beam size of 100 in the semantic parser, executing each of the 10 highest-scoring logical forms with a beam of 100 executions.

[1] In our experiments, we automatically learn the lexicon in a preprocessing step. See Section 5.2 for details.

3.5 Training

P^3 is trained by maximizing loglikelihood with stochastic gradient ascent. The training data $\{(q^i, v^i, c^i)\}_{i=1}^n$ is a collection of questions q^i and environments v^i paired with *supervision oracles* c^i. $c^i(e) = 1$ for a correct execution e and $c^i(e) = 0$ otherwise. The oracle c^i can implement various kinds of supervision, including: (1) labeled denotations, by verifying the value returned by e and (2) labeled environments, by verifying each choice made by e. The oracle for diagram question answering combines both forms of supervision (Section 4.5).

The objective function O is the loglikelihood of predicting a correct execution:

$$O(\theta) = \sum_{i=1}^{n} \log \sum_{e,l,t} c^i(e) P(e, \ell, t | q^i, v^i; \theta)$$

We optimize this objective function using stochastic gradient ascent, using the approximate inference algorithm from Section 3.4 to estimate the necessary marginals. When computing the marginal distribution over correct executions, we filter each step of the beam search using the supervision oracle c^i to improve the approximation.

4 Diagram Question Answering with P^3

As a case study, we apply P^3 to the task of answering food web diagram questions from an 8th grade science domain. A few steps are required to apply P^3. First, we create a domain theory of food webs that represents extracted information from the diagram and background knowledge for the domain. Second, we define the features of the execution model that are used to learn how programs in the domain theory execute given a diagram. Third, we define a component to select a multiple-choice answer given a denotation. Finally, we define the supervision oracle used for training.

4.1 Food Web Diagram Questions

We consider the task of answering food web diagram questions. The input consists of a diagram depicting a food web, a natural language question and a list of natural language answer options (Figure 1). The goal is to select the correct answer option. This task has many regularities that require global features: for example, food webs are usually acyclic and certain animals usually have certain roles (e.g., mice are herbivores). We have collected and released a data set for this task (Section 5.1).

We preprocess the diagrams in the data set using a computer vision system that identifies candidate diagram elements (Kembhavi et al., 2016). This system extracts a collection of text labels (via OCR), arrows, arrowheads and objects, each with corresponding scores. It also extracts a collection of scored linkages between these elements. These extractions are noisy and contain many discrepancies such as overlapping text labels and spurious linkages. We use these extractions to define a set of candidate organisms (using the text labels), and also to define features of the execution model.

4.2 Domain Theory

The domain theory is a probabilistic program encoding the information to extract from the environment as well as background knowledge about food webs. It represents the structure of a food web using two functions. These functions are predicates that invoke `choose` to return either true or false. The execution model learns to predict which of these values is correct for each set of arguments given the diagram. It furthermore has a collection of deterministic functions that encode domain knowledge, including definitions of animal roles such as HERBIVORE and a model of population change causation.

Figure 4.2 shows pseudocode for a portion of the domain theory. Food webs are represented using two functions over the extracted text labels: ORGANISM(x) indicates whether the label x is an organism (as opposed to, e.g., the diagram title); and EATS(x, y). The definitions of these functions invoke `choose` while remembering previously chosen values to avoid double counting probabilities when executing logical forms such as ORGANISM(DEER) $\wedge$ ORGANISM(DEER). The remembered values are stored in a global variable that is also used to implement the supervision oracle. Deterministic functions such as CAUSE are defined in terms of these learned functions.

The uses of `choose` in the domain theory create a tree of possible executions for every logical form. Figure 4.2 illustrates this tree for the logical form λf.CAUSE(DECREASE(MICE), f(SNAKES)), which

```
# initialize predicate instance variables
# from text labels in environment
world = {"mice": undef,
         ("mice", "snakes"): undef, ...}
def organism(name)
  if (world[name] == undef)
    world[name] = choose(true, false)
  return world[name]

def eats(x, y)
  # same as organism but with pairs.

# entities referenced in the logical form
# must be organisms. choose() represents
# failure; it returns no values.
def getOrganism(x)
  if (organism(x)) return x else choose()

# change events are direction/
# text label tuples
def decrease(x)
  return ("decrease", x)

def cause(e1, e2)
  e12 = eats(e1[1], e2[1])
  e21 = eats(e2[1], e1[1])
  # deterministic model with cases. e.g.
  # if eats(y, x) then (cause (decrease x)
  # (decrease y)) -> true
  return doCause(e1[0], e2[0], e12, e21)
```

Figure 3: Domain theory pseudocode for diagram question answering.

corresponds to the question "what happens to the snakes when the mice decrease?" This logical form is shorthand for the following program:

```
filter(lambda f.cause(
    decrease(getOrganism("mice")),
    f(getOrganism("snakes"))),
  set(decrease, increase, unchanged))
```

Specifically, entities such as MICE are created by calling `getOrganism` and logical forms with functional types implicitly represent filters over the appropriate argument type. Executing this program first applies the filter predicate to `decrease`. Next, it evaluates `getOrganism("mice")`, which calls `organism` and encounters the first call to `choose`. This call is shown as the first branch of the tree in Figure 4.2. The successful branch proceeds to evaluate `getOrganism("snakes")`, shown as the second branch. Finally, the successful branch evaluates `cause`, which calls `eats` twice, resulting in the final two branches. The value returned by each branch is determined by the causation model which performs

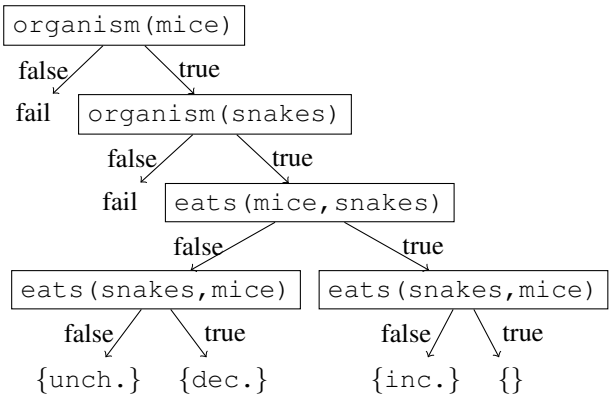


Figure 4: Tree of possible executions for the logical form $\lambda f.\text{CAUSE}(\text{DECREASE}(\text{MICE}), f(\text{SNAKES}))$. Each path from root to leaf represents a single execution that returns the indicated denotation or fails, and each internal node represents a nondeterministic choice made with `choose`.

some deterministic logic on the truth values of the two eats relations.

4.3 Execution Features

The execution model uses three sets of features: instance features, predicate features, and denotation features. Instance features treat each predicate instance independently, while the remainder are global features of multiple predicate instances and the logical form. We provide a complete listing of features in an online appendix.

Instance features fire whenever an execution chooses a truth value for a predicate instance. These features are similar to the per-predicate-instance features used in prior work to produce a distribution over possible worlds. For $\text{ORGANISM}(x)$, our features are the vision model's extraction score for x and indicator features for the number of tokens in x. For $\text{EATS}(x, y)$, our features are various combinations of the vision model's scores for arrows that may connect the text labels x and y.

Predicate features fire based on the global assignment of truth values to all instances of a single predicate. The features for ORGANISM count occurrences of overlapping text labels among true instances. The features for EATS include cycle count features for various cycle lengths and arrow reuse features. The cycle count features help the model learn that food webs are typically, but not always, acyclic and the arrow reuse features aim to prevent the model from predicting two different EATS instances on the basis of a single arrow.

Denotation features fire on the return value of an execution. There are two kinds of denotation features: size features that count the number of entities in denotations of various types and denotation element features for specific logical forms. The second kind of feature can be used to learn that the denotation of $\lambda x.\text{HERBIVORE}(x)$ is likely to contain MOUSE, but unlikely to contain WOLF.

4.4 Answer Selection

P^3 predicts a distribution over denotations for each question, which for our problem must be mapped to a distribution over multiple choice answers. Answer selection performs this task using string match heuristics and an LSTM (Hochreiter and Schmidhuber, 1997). The string match heuristics score each answer option given a denotation then select the highest scoring answer, abstaining in the case of a tie. The score computation depends on the denotation's type. If the denotation is a set of entities, the score is an approximate count of the number of entities in the denotation mentioned in the answer using a fuzzy string match. If the denotation is a set of change events, the score is a fuzzy match of both the change direction and the animal name. If the denotation is a number, string matching is straightforward. Applying these heuristics and marginalizing out denotations yields a distribution over answer options.

A limitation of the above approach is that it does not directly incorporate linguistic prior knowledge about likely answers. For example, "snake" is usually a good answer to "what eats mice?" regardless of the diagram. Such knowledge is known to be essential for visual question answering (Antol et al., 2015; Andreas et al., 2016b) and important in our task as well. We incorporate this knowledge in a standard way, by training a neural network on question/answer pairs (without the diagram) and combining its predictions with the string match heuristics above. The network is a sequence LSTM that is applied to the question concatenated with each answer option a to produce a 50-dimensional vector v_a for each answer. The distribution over answers is the softmax of the inner product of these vectors with a learned parameter vector w. For simplicity, we combine these two components using a 50/50 mix of their answer distributions.

4.5 Supervision Oracle

The supervision oracle for diagram question answering combines supervision of both answers and environment interpretations. We assume that each diagram has been labeled with a food web. An execution is correct if and only if (1) all of the chosen values in the global variable encoding the food web are consistent with the labeled food web, and (2) string match answer selection applied to its denotation chooses the correct answer. The first constraint guarantees that every logical form has at most one correct execution for any given diagram.

5 Evaluation

Our evaluation compares P^3 to both possible worlds and neural network approaches on our data set of food web diagram questions. An ablation study demonstrates that both sets of global features improve accuracy. Finally, we demonstrate P^3's generality by applying it to a previously-published data set, obtaining state-of-the-art results.

Code, data and supplementary material for this paper are available at: `http://www.allenai.org/paper-appendix/emnlp2016-p3`

5.1 FOODWEBS Data Set

FOODWEBS consists of ∼500 food web diagrams and ∼5000 questions designed to imitate actual questions encountered on 8th grade science exams. The train/validation/test sets contain ∼300/100/100 diagrams and their corresponding questions. The data set has three kinds of annotations in addition to the correct answer for each question. First, each diagram is annotated with the food web that it depicts using ORGANISM and EATS. Second, each diagram has predictions from a vision system for various diagram elements such as arrows and text labels (Kembhavi et al., 2016). These are noisy predictions, not ground truth. Finally, each question is annotated by the authors with a logical form (or null if its meaning is not representable in the domain theory). These logical forms are not used to train P^3 but are useful to measure per-component error.

We collected FOODWEBS by using a crowdsourcing process to expand a collection of real exam questions. First, we collected 89 questions from 4th and 8th grade exams and 500 food web diagrams us-

ing an image search engine. Second, we generated questions for these diagrams using Mechanical Turk. Workers were shown a diagram and a real question for inspiration and asked to write a new question and its answer options. We validated each generated question by asking 3 workers to answer it, discarding questions where at least 2 did not choose the correct answer. We also manually corrected any ambiguous (e.g., two answer options are correct) and poorly-formatted (e.g., two answer options have the same letter) questions. The final data set has high quality: a human domain expert correctly answered 95 out of 100 randomly-sampled questions.

5.2 Baseline Comparison

Our first experiment compares P^3 with several baselines for situated question answering. The first baseline, WORLDS, is a possible worlds model based on Malinowski and Fritz (2014). This baseline learns a semantic parser $P(\ell, t|q)$ and a distribution over food webs $P(w|v)$, then evaluates ℓ on w to produce a distribution over denotations. This model is implemented by independently training P^3's CCG parser (on question/answer pairs and labeled food webs) and a possible-worlds execution model (on labeled food webs). The CCG lexicon for both P^3 and WORLDS was generated by applying PAL (Krishnamurthy, 2016) to the same data. Both models select answers as described in Section 4.4.

We also compared P^3 to several neural network baselines. The first baseline, LSTM, is the text-only answer selection model described in Section 4.4. The second baseline, VQA, is a neural network for visual question answering. This model represents each image as a vector by using the final layer of a pre-trained VGG19 model (Simonyan and Zisserman, 2014) and applying a single fully-connected layer. It scores answer options by using the answer selection LSTM to encode question/answer pairs, then computing a dot product between the text and image vectors. This model is somewhat limited because VGG features are unlikely to encode important diagram structure, such as the content of text labels. Our third baseline, DQA, is a neural network that rectifies this limitation (Kembhavi et al., 2016). It encodes the diagram predictions from the vision system as vectors and attends to them using the LSTM-encoded question vector to select an answer. This model is trained with question/answer pairs and diagram parses, which is roughly comparable to the supervision used to train P^3.

Model	Accuracy	Accuracy (Unseen Organisms)
P^3	**69.1**	**57.7**
WORLDS	63.6	50.8
LSTM	60.3	34.7
VQA	56.5	36.8
DQA	59.3	33.0
Random	25.2	25.2

Table 1: Accuracy of P^3 and several baselines on the FOODWEBS test set and a modified test set with unseen organisms.

Model	Accuracy	Δ
P^3	69.1	
-LSTM	59.8	-9.3
-LSTM -denotation	55.8	-13.3
-LSTM -denotation -predicate	52.4	-16.7

Table 2: Test set accuracy of P^3 removing LSTM answer selection (Section 4.4), denotation features and predicate features (Section 4.3).

Table 5.2 compares the accuracy of P^3 to these baselines. Accuracy is the fraction of questions answered correctly. LSTM performs well on this data set, suggesting that many questions can be answered without using the image. This result is consistent with results on visual question answering (Antol et al., 2015). The other neural network models have similar performance to LSTM, whereas both WORLDS and P^3 outperform it. We also find that P^3 outperforms WORLDS likely due to its global features, which we investigate in the next section.

Given these results, we hypothesized that the neural models were largely memorizing common patterns in the text and were not able to interpret the diagram. We tested this hypothesis by running each model on a test set with unseen organisms created by reversing the organism names in every question and diagram (Table 5.2, right column). As expected, the accuracy of LSTM is considerably reduced on this data set. VQA and DQA again perform similarly to LSTM, which is consistent with our hypothesis. In contrast, we find that the accuracies of WORLDS and P^3 are only slightly reduced, which is consistent with superior diagram interpretation abilities but ineffective LSTM answer selection.

5.3 Ablation Study

We performed an ablation study to further understand the impact of LSTM answer selection and global features. Table 5.2 shows the accuracy of P^3 trained without these components. We find that LSTM answer selection improves accuracy by 9 points, as expected due to the importance of linguistic prior knowledge. Global features improve accuracy by 7 points, which is roughly comparable to the delta between P^3 and WORLDS in Table 5.2.

5.4 Component Error Analysis

Our third experiment analyses sources of error by training and evaluating P^3 while providing the gold logical form, food web, or both as input. Table 5.5 shows the accuracy of these three models. The final entry shows the maximum accuracy possible given our domain theory and answer selection. The larger accuracy improvement with gold food webs suggests that the execution model is responsible for more error than semantic parsing, though both components contribute.

5.5 SCENE Experiments

Our final experiment applies P^3 to the SCENE data set of Krishnamurthy and Kollar (2013). In this data set, the input is a natural language expression, such as "blue mug to the left of the monitor," and the output is the set of objects in an image that the expression denotes. The images are annotated with a bounding box for each candidate object. The data set includes a domain theory that was automatically generated by creating a category and/or relation per word based on its part of speech. It also includes a CCG lexicon and image features. We use these resources, adding predicate and denotation features.

Table 5.5 compares P^3 to prior work on SCENE. The evaluation metric is exact match accuracy between the predicted and labeled sets of objects. We consider three supervision conditions: QA trains with question/answer pairs, QA+E further includes labeled environments, and QA+E+LF further includes labeled logical forms. We trained P^3 in the first two conditions, while prior work trained in the first and third conditions. KK2013 is a possible worlds model with a max-margin training objective. P^3 slightly outperforms in the QA condition and P^3 trained with labeled environments outperforms prior work trained with additional logical form labels.

Model	Accuracy	Δ
P^3	69.1	
+ gold logical form	75.1	+6.0
+ gold food web	82.3	+13.2
+ both	91.6	+22.5

Table 3: Accuracy of P^3 when trained and evaluated with labeled logical forms, food webs, or both.

	Supervision		
Model	QA	QA+E	QA+E+LF
P^3	68	75	–
KK2013	67	–	70

Table 4: Accuracy on the SCENE data set. KK2013 results are from Krishnamurthy and Kollar (2013).

6 Conclusion

Parsing to Probabilistic Programs (P^3) is a novel model for situated question answering that jointly reasons about question and environment interpretations using background knowledge to produce answers. P^3 uses a domain theory – a probabilistic program – to define the information to be extracted from the environment and background knowledge. A semantic parser maps questions to logical forms in this theory, which are probabilistic programs whose possible executions represent possible interpretations of the environment. An execution model scores these executions given features of the environment. Both the semantic parser and execution model are jointly trained in a loglinear model, which thereby learns to both parse questions and interpret environments. Importantly, the model includes global features of the logical form and executions, which help the model avoid implausible interpretations. We demonstrate P^3 on a challenging new data set of 5000 science diagram questions, where it outperforms several competitive baselines.

Acknowledgments

We gratefully acknowledge Minjoon Seo, Mike Salvato and Eric Kolve for their implementation help, Isaac Cowhey and Carissa Schoenick for their help with the data, and Oren Etzioni, Peter Clark, Matt Gardner, Hannaneh Hajishirzi, Mike Lewis, and Jonghyun Choi for their comments.

References

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016a. Deep compositional question answering with neural module networks. In *CVPR*.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016b. Learning to compose neural networks for question answering. In *NAACL*.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. 2015. VQA: Visual question answering. In *International Conference on Computer Vision (ICCV)*.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.

Jonathan Berant, Vivek Srikumar, Pei-Chun Chen, Abby Vander Linden, Brittany Harding, Brad Huang, Peter Clark, and Christopher D. Manning. 2014. Modeling biological processes for reading comprehension. In *Proceedings of EMNLP*.

David L. Chen and Raymond J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*.

Eunsol Choi, Tom Kwiatkowski, and Luke Zettlemoyer. 2015. Scalable semantic parsing with partial ontologies. In *Proceedings of the 2015 Association for Computational Linguistics*.

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

Peter Clark and Oren Etzioni. 2016. My computer is an honor student - but how intelligent is it? standardized tests as a measure of ai. *AI Magazine*, 37:5–12.

Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. 2016. Multimodal compact bilinear pooling for visual question answering and visual grounding. *arXiv:1606.01847*.

Noah D Goodman and Andreas Stuhlmüller. 2014. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org. Accessed: 2016-2-25.

Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: A language for generative models. In *Uncertainty in Artificial Intelligence*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Thomas M. Howard, Istvan Chung, Oron Propp, Matthew R. Walter, and Nicholas Roy. 2014a. Efficient natural language interfaces for assistive robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Workshop on Rehabilitation and Assistive Robotics*, September.

Thomas M Howard, Stefanie Tellex, and Nicholas Roy. 2014b. A natural language planner interface for mobile manipulators. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*.

Aniruddha Kembhavi, Mike Salvato, Eric Kolve, Min Joon Seo, Hannaneh Hajishirzi, and Ali Farhadi. 2016. A diagram is worth a dozen images. In *European Conference on Computer Vision (ECCV)*.

Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. 2010. Toward understanding natural language directions. In *Proceedings of the 5th ACM/IEEE International Conference on Human-Robot Interaction*.

Jayant Krishnamurthy and Thomas Kollar. 2013. Jointly learning to parse and perceive: Connecting natural language to the physical world. *Transactions of the Association of Computational Linguistics – Volume 1*.

Jayant Krishnamurthy and Tom M. Mitchell. 2012. Weakly supervised training of semantic parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Jayant Krishnamurthy and Tom M. Mitchell. 2015. Learning a compositional semantics for freebase with an open predicate vocabulary. *Transactions of the Association for Computational Linguistics*, 3:257–270.

Jayant Krishnamurthy. 2016. Probabilistic models for learning a semantic parser lexicon. In *NAACL*.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.

Percy Liang, Michael I. Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the Association for Computational Linguistics*.

Mateusz Malinowski and Mario Fritz. 2014. A multi-world approach to question answering about real-world scenes based on uncertain input. In *Advances in Neural Information Processing Systems*.

Mateusz Malinowski, Marcus Rohrbach, and Mario Fritz. 2015. Ask your neurons: A neural-based approach to

answering questions about images. In *International Conference on Computer Vision*.

Cynthia Matuszek, Nicholas FitzGerald, Luke Zettlemoyer, Liefeng Bo, and Dieter Fox. 2012. A joint model of language and perception for grounded attribute learning. In *Proceedings of the 29th International Conference on Machine Learning*.

John McCarthy. 1963. A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*.

Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*.

Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. Solving geometry problems: Combining text and diagram interpretation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis Banerjee, Seth Teller, and Nicholas Roy. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI Conference on Artificial Intelligence*.

Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. Question Answering on Freebase via Relation Extraction and Textual Evidence. In *Proceedings of the Association for Computational Linguistics (ACL 2016)*.

Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alexander J. Smola. 2015. Stacked attention networks for image question answering. *arXiv preprint arXiv:1511.02274*.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

John M. Zelle and Raymond J. Mooney. 1993. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the 11th National Conference on Artificial Intelligence*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*.