Look-up and Adapt: A One-shot Semantic Parser

Zhichu Lu[*]
Carnegie Mellon University
Pittsburgh, PA
`zhichul@cs.cmu.edu`

Forough Arabshahi[*]
Carnegie Mellon University
Pittsburgh, PA
`farabsha@cs.cmu.edu`

Igor Labutov
LAER AI, Inc.
New York, NY
`igor.labutov@laer.ai`

Tom Mitchell
Carnegie Mellon University
Pittsburgh, PA
`tom.mitchell@cs.cmu.edu`

Abstract

Computing devices have recently become capable of interacting with their end users via natural language. However, they can only operate within a limited "supported" domain of discourse and fail drastically when faced with an out-of-domain utterance, mainly due to the limitations of their semantic parser. In this paper, we propose a semantic parser that generalizes to out-of-domain examples by learning a general strategy for parsing an *unseen* utterance through adapting the logical forms of *seen* utterances, instead of learning to generate a logical form from scratch. Our parser maintains a memory consisting of a representative subset of the seen utterances paired with their logical forms. Given an unseen utterance, our parser works by looking up a similar utterance from the memory and adapting its logical form until it fits the unseen utterance. Moreover, we present a data generation strategy for constructing utterance-logical form pairs from different domains. Our results show an improvement of up to 68.8% on one-shot parsing under two different evaluation settings compared to the baselines.

1 Introduction and Background

Speech recognition technologies are achieving human parity (Xiong et al., 2016). As a result, end users can now access different functionalities of their phones and computers through spoken instructions via a natural language processing interface referred to as a conversational agent. Current commercial conversational agents such as Siri, Alexa or Google Assistant come with a fixed set of simple functions like setting alarms and making reminders, but are often not able to cater to the specific phrasing of a user or the specific action a user needs. However, it has recently been shown that it is possible to add new functionalities to an agent through natural language *instruction* (Azaria et al., 2016; Labutov et al., 2018).

For example, assume that the user wants to add a functionality for resetting an alarm based on the weather forecast for the next day, as demonstrated by the following utterance: "whenever it snows at night, wake me up 30 minutes early". The user can instruct this task to the agent by breaking it down into a sequence of actions that the agent already knows.

- check the weather app,
- see if the forecast is calling for snow,
- if yes, then reset the time of the alarm to 30 minutes earlier.

This set of instructions result in a logical form or a semantic parse for this specific new utterance. However, this approach can be used in practice only if the agent is capable of generalizing from this single new utterance to similar utterances such as "if the weather is rainy, then set an alarm for 1 hour later". We refer to this problem as *one-shot semantic parsing*.

In this paper, we address this one-shot semantic parsing task and present a semantic parser that generalizes to out-of-domain utterances by seeing a single example from that domain. While state of the art neural semantic parsers are flexible to language variations, they need plenty of examples from the new domain to be able to parse an utterance from that domain, which is not possible in our scenario. On the other hand, grammar parsers are not robust to the flexibility of language because of their use of string matching. Therefore, we propose a method that preserves the robustness of neural semantic parsers while addressing

[*] These two authors contributed equally

the data sparsity of the one-shot semantic parsing task. We present a general strategy for "adapting" logical forms rather than constructing them from scratch by "looking up" similar sentences that we know how to parse and changing their logical forms until they fit the new utterances. These logical forms are looked-up from a memory that contains a representative subset of previously seen utterance-logical form pairs. Once this general strategy is learned, the parser can be extended to parse an utterance from a new domain by adding one new example of that domain to the memory.

We propose a dataset generation method that allows us to evaluate the effectiveness of our model in a one-shot setting. We show that we generate reasonably good utterances while creating different experimental setups and scenarios for evaluation.

Summary of Results In this paper we propose a novel neural semantic parser for the task of one-shot semantic parsing. We design two different experiments to evaluate the parsing accuracy. We show that our approach improves the performance of neural semantic parsers by a significant margin across 6 different domains of discourse. Moreover, we present a detailed analysis of our proposed model and the performance of its different components through an oracle study.

2 Problem Definition

In this section, we introduce the basic terminology used in the paper and formally define the task of one-shot semantic parsing.

A semantic parser takes as input an *utterance* and outputs a corresponding *logical form*. An utterance is a sequence of words and a logical form is an s-expression capturing the meaning of the utterance.

For example, "parents of John's friends" is an utterance and `(field parent (field friend John))` is its logical form.

We use a synchronous context free grammar (SCFG), which is a generalization of the context-free grammar (CFG), to generate grammar rules (Chiang, 2006). A rule in an SCFG has a left hand side, which is a non-terminal category, and two right hand sides, referred to as the source and the target. For our semantic parsing task, the source corresponds to an utterance, and the target corresponds to a logical form. We denote a grammar rule with the small letter g. A set of grammar rules, denoted by the capital letter G, span a domain of utterances. In this paper, we define $domain(G)$ as the set of all *utterance-logical form* pairs generated by a set of grammar rules G. A sample SCFG G_{sample} with its domain are provided in Tables 5 of Appendix A and Table 1 below, respectively.

Utterance	Logical Form
John	`john`
Mary	`mary`
parents	`parent`
children	`child`
parents of John	`(field parent john)`
parents of Mary	`(field parent mary)`
children of John	`(field child john)`
children of Mary	`(field child mary)`
John 's parents	`(field parent john)`
Mary 's parents	`(field parent mary)`
John 's children	`(field child john)`
Mary 's children	`(field child mary)`

Table 1: Domain of the sample SCFG G_{sample}. The rules of this SCFG are presented in the Appendix A. The domain of the sample SCFG is the set of all ⟨source, `target`⟩ pairs listed in this table.

In **one-shot semantic parsing** we are given as input a subset of utterance and logical form pairs from $domain(G)$ and a single utterance and logical form pair from $domain(G')$. The goal of one-shot semantic parsing is to parse utterances from $domain(G')$ that are not in the input.

Let us provide an example to make this more clear. Consider that we are given utterance and logical form pairs from $domain(G_{\text{sample}})$ in Table 1 as well as the following utterance and logical form pair as input.

⟨friends of John, `(field friend john)`⟩

The goal of one-shot semantic parsing is to parse examples such as

⟨friends of Mary, `(field friend Mary)`⟩ (1)

⟨parents of Mary's friends, `(field parent(field friend Mary))`⟩ (2)

which are not in $domain(G_{\text{sample}})$.

This is a challenging task for a complex grammar since the domain of a typical grammar consists of hundreds of thousand of examples. Therefore, being able to parse all variations of utterances in the domain by seeing only a single example from it does not have a trivial solution. Grammar-based semantic parsers usually rely on string matching which limits their robustness to natural language variation. Neural semantic parsers are more robust compared to grammar-based parsers. However, their performance significantly drops in such a data-hungry setting. In the next section, we present our look-up adapt semantic parser that addresses the challenges of neural semantic parsers in a data-sparse scenario.

3 Look-up and Adapt

In this section, we propose a novel neural network architecture for one-shot semantic parsing. We are given a set of utterance-logical form pairs as input and our goal is to output a semantic parse for an unseen query utterance. Our model is able to generate a logical form for the utterance by "looking up" a similar utterance from a pool of utterance-logical form pairs and "adapting" its logical form. This pool of utterances is maintained in a "memory" and consist of a representative subset of the input utterance-logical form pairs. We will show that in the data-sparse scenario of one-shot semantic parsing, adapting known logical forms is easier compared to generating a logical form from scratch. We will also discuss that it is important what subset of the data is included in the memory.

Our model consists of two main modules, namely look-up § 3.1 and adapt § 3.2. Figure 1 shows a sketch of our proposed model and the main look-up adapt algorithm is given in Algorithm 1. The look-up module is responsible for retrieving utterance-logical form pairs from the memory, using two Bidirectional LSTM encoders. The adapt module is responsible for adapting the retrieved logical form until it results in the correct semantic parse. The adapt module has two sub-modules, namely the aligner and the discriminator. The aligner is responsible for aligning the logical form with the query utterance and the discriminator decides which parts of the logical form should be swapped with a new one from the memory. In the following sections, we will define each sub-module of our proposed model in detail and propose a loss for training the model in an end-to-end fashion.

Before describing the model components, let us start by introducing the notation we will use in the following sections. Bold capital letters represent matrices, and bold lower case letters represent vectors. $\boldsymbol{X}$ denotes a distributed representation of an utterance, where each column of $\boldsymbol{X}$ is the representation for a word in the utterance. $\boldsymbol{Y}$ denotes a distributed representation of a logical form, where each column of $\boldsymbol{Y}$ is the representation for a predicate in the logical form. T is a tree representation of the logical form. $\boldsymbol{w}$ and $\boldsymbol{w}'$ denote attention weights that sum to at least 0 and at most 1.

Algorithm 1 LookUpAndAdapt

```
1: function LOOKUPANDADAPT(Memory, $\boldsymbol{X}'$, $\boldsymbol{w}'$)
2:     ($\boldsymbol{X}$, $\boldsymbol{Y}$, T) ← LOOKUP(Memory, $\boldsymbol{X}'$, $\boldsymbol{w}'$)
3:     for all t in T.children do
4:         t' ← ADAPT(Memory, $\boldsymbol{X}'$, $\boldsymbol{Y}$, t, $\boldsymbol{w}'$)
5:         T.children.replace(t, t')
6:     return T
```

3.1 Look-up

In this section, we describe our look-up strategy and discuss its sub-modules in detail.

Encoder Bidirectional RNNs have been successfully used to represent sentences in many areas of natural language processing such as question answering, neural machine translation, and semantic parsing (Bengio and LeCun, 2015; Hermann et al., 2015; Dong and Lapata, 2016). We use Bidirectional LSTMs where the forward LSTM reads the input in the utterance word order $(x_1, x_2, ..., x_T)$ and the backward LSTM reads the input in reversed order $(x_T, x_{T-1}, ..., x_1)$. The output for each word x_i is the concatenation of the ith hidden state h_i^{fwd} of the forward LSTM and the $T - i$th hidden state h_{T-i}^{bck} of the backward LSTM. As shown in Figure 1, we use two Bidirectional LSTMs, one to encode the utterance, and one to encode the logical form. The utterance is treated as a sequence of words, where each word is represented as a pretrained GloVe (Pennington et al., 2014) embedding. The logical form is treated as a sequence of predicates and parentheses, each of which is also represented as a pretrained GloVe embedding. The motivation to use GloVe embeddings for both words and predicates is to avoid the problem of unknown words/predicates encountered in the one-shot utterance's domain.

Retrieving from the memory The memory consists of a subset of the utterance-logical form pairs given as input to the algorithm. We refer to the number of pairs in the memory as its size and denote it with n. We would like to note that the algorithm's success depends on having a memory that captures the structure of the domain. In this

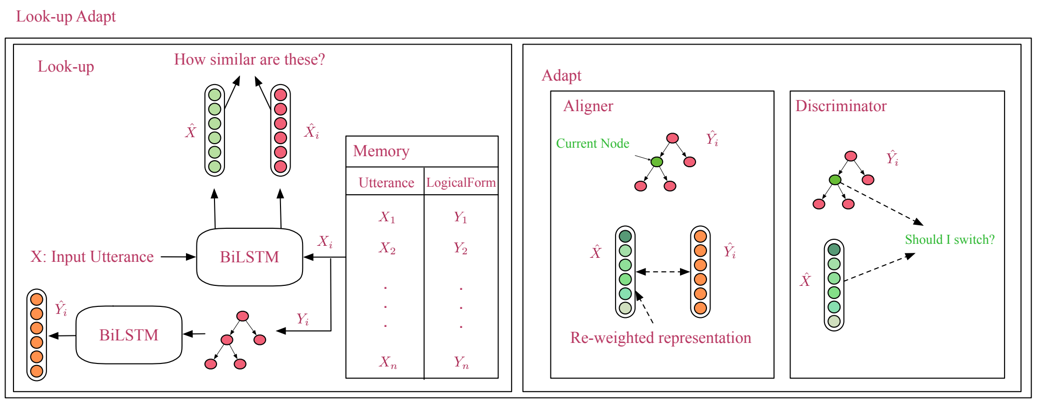

Figure 1: Look-up Adapt semantic parser. Our model consists of two main modules, namely "Look-up" (left box) and "Adapt" (right box). The look-up module is responsible for retrieving utterance-logical form pairs from the memory that are similar to an input query utterance X. The 'Adapt' module is further broken down into two modules "Aligner" and "Discriminator". Given a looked-up utterance and logical form pair, the aligner module traverses the logical form tree and updates the utterance representation accordingly while the discriminator module decides weather to switch the current node with another one from the memory or keep it as is.

paper, we make a simplifying assumption that the memory is given by an oracle, and it consists of one example per grammar rule. Other choices for the memory are also possible but exploring them falls out of the scope of this paper. An interested reader is referred to Appendix B for a more detailed discussion. In the future we wish to automate the acquisition of this memory.

Given an encoded query utterance $\boldsymbol{X}'$, we model the retrieval as a classification over examples in the memory. We compute a fixed size query vector $\boldsymbol{x}' = \boldsymbol{X}' \cdot \boldsymbol{w}'$ which is a weighted average of the column vectors in $\boldsymbol{X}'$. We compute a fixed-size key vector for every example in the memory $\boldsymbol{x} = \sum_{i=1}^{T} \boldsymbol{X}_i$, as the mean of the column vectors of $\boldsymbol{X}$.

Given a query vector $\boldsymbol{x}'$ and vectors $\boldsymbol{x}_i$ from the memory for $i = 1, 2, ..., n$ where n is the memory size, we model the probability of retrieving the ith entry from memory using equation 3

$$P(i|\boldsymbol{x}') = \frac{\exp(e_i)}{\sum_{j=1}^{n} \exp(e_j)} \tag{3}$$

where

$$e_i = f([\boldsymbol{x}', \boldsymbol{x}_i, \boldsymbol{x}' \odot \boldsymbol{x}_i, |\boldsymbol{x}' - \boldsymbol{x}_i|, \boldsymbol{x}'^T W_1 \boldsymbol{x}_i]) \tag{4}$$

where $\odot$ is element-wise product,$[a, b]$ is concatenation of vectors a and b, and f is a Multi-Layer-Perceptron with a single hidden layer and ReLU activation function. This way of featurizing $\boldsymbol{x}'$ and $\boldsymbol{x}$ was adapted from (Kumar et al., 2016). LOOKUP greedily returns the example with the highest probability of being selected.

3.2 Adapt

The algorithm for ADAPT is given in Algorithm 2. The adaptation is two phase. First an alignment from `T` to relevant words in the new utterance $\boldsymbol{X}'$ is computed. Then a decision is made on whether this subtree fails to match the relevant words in the utterance and needs to be replaced. If yes, a recursive call to LOOKUPANDADAPT will be made with an updated attention $\boldsymbol{w}$ to focus on words relevant to the subtree, and the returned value will be propagated and eventually used as a replacement for the current subtree. Otherwise, recursive calls to ADAPT will be made to check the children of the current subtree. For an example see Table 9.

The input `Memory` to the algorithm is used in case a recursive call to LookUpAndAdapt is needed. $\boldsymbol{X}'$ is the representation of the new utterance. $\boldsymbol{Y}$ is the representation of the current logical form. `T` is a subtree of $\boldsymbol{Y}$ that we wish to adapt. $\boldsymbol{w}'$ is the attention weights of the parent of `T`, to be used as a mask when the alignment of `T` is computed. In the following space we describe the aligner module for alignment and the discriminator module for scoring a match between a subtree and part of the new utterance.

Aligner Module The aligner module produces an attention score over the new utterance given a subtree of a logical form. Inspired by the gating attention mechanism in (Kumar et al., 2016), we model attention for every word in the utterance as the maximum attention given by any predicate in the subtree `T`.

Algorithm 2 Adapt

```
1: function ADAPT(Memory, $\boldsymbol{X}'$, $\boldsymbol{Y}$, T, $\boldsymbol{w}'$)
2:     Phase 1: aligning T to relevant words in $\boldsymbol{X}'$
3:     $\boldsymbol{w} \leftarrow$ ALIGN($\boldsymbol{X}', \boldsymbol{Y}$, T, $\boldsymbol{w}'$)
4:     $\boldsymbol{x}' \leftarrow \boldsymbol{X}' \cdot \boldsymbol{w}$
5:     if $\sum_i \boldsymbol{w}_i > 1$ then
6:         $\boldsymbol{x}' \leftarrow \boldsymbol{x}' \div \sum_i \boldsymbol{w}_i$
7:
8:     Phase 2: deciding whether to replace T and then recurse
9:     $P(fail\ to\ match) \leftarrow$ DISCRIMINATE($\boldsymbol{x}', \boldsymbol{Y}$, T)
10:    if $P(fail\ to\ match) > 0.5$ then
11:        return LOOKUPANDADAPT(Memory, $\boldsymbol{X}'$, $\boldsymbol{w}$)
12:    else
13:        for all t in T.children do
14:            t' $\leftarrow$ ADAPT(Memory, $\boldsymbol{X}'$, $\boldsymbol{Y}$, t, $\boldsymbol{w}$)
15:            T.children.replace(t, t')
16:        return T
```

$$\boldsymbol{w}_i = \max_{j \in span(\mathtt{T})} \sigma(s_{ij}) \tag{5}$$

where σ is the sigmoid function and

$$s_{ij} = g([\boldsymbol{X}_i, \boldsymbol{Y}_j, \boldsymbol{Y}_p, \boldsymbol{X}_i \odot \boldsymbol{Y}_j, \boldsymbol{X}_i \odot \boldsymbol{Y}_p, |\boldsymbol{X}_i - \boldsymbol{Y}_j|, |\boldsymbol{X}_i - \boldsymbol{Y}_p|, \boldsymbol{X}_i^T W_2 \boldsymbol{Y}_j, \boldsymbol{X}_i^T W_p \boldsymbol{Y}_p]) \tag{6}$$

where $\odot$ is element-wise product, $[a, b]$ is concatenation of vectors a and b, $\boldsymbol{X}_i$ is the ith column of $\boldsymbol{X}$, $\boldsymbol{Y}_i$ is the ith column of $\boldsymbol{Y}$, and g is a learnable linear transformation. $\boldsymbol{Y}_p$ is the representation of the parent predicate of the subtree `T` (e.g. the parent predicate is `field` for subtree `john` of logical form `(field parent john)`). We found that adding this feature helps the model learn to align arguments of predicates better.

We also found that constraining the attention further by requiring that the attention of a child node $\boldsymbol{w}$ be a refinement of the attention of a parent node $\boldsymbol{w}'$ facilitates good alignment. This is modeled by the update rule

$$\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i \cdot \sqrt{\boldsymbol{w}_i'} \tag{7}$$

Observe that if the attention score of the parent node is very low for some word, the child is not going to be able to look at it. This encourages the parent node to attend to words not just relevant to itself but also the children, and encourages the child to refine rather than drastically change from the alignment of the parent node.

Given the output weights of the aligner module, we compute a fixed-size representation for the relevant words in the utterance to the current subtree `T` by first taking the weighted average $\boldsymbol{x}' = \boldsymbol{X}' \cdot \boldsymbol{w}$ where $\cdot$ is matrix product. If $\sum_i \boldsymbol{w}_i' > 1$, we will divide $\boldsymbol{x}'$ by $\sum_i \boldsymbol{w}_i'$ to normalize it. $\boldsymbol{x}'$ is then compared against the representation of the subtree by the discriminator to produce a confidence score indicating whether the subtree matches the words it aligns to in the utterance.

Discriminator Module The discriminator module learns to tell when a subtree fails to match the meaning of the words in its alignment. Given a subtree `T` of a logical form $\boldsymbol{Y}$, its fixed-size representation is computed as a mean of the representations for each predicate in the subtree using the formula $\boldsymbol{y} = \sum_{i \in span(\mathtt{T})} \boldsymbol{Y}_i$, where $\boldsymbol{Y}_i$ is the ith column of $\boldsymbol{Y}$, and $span(\mathtt{T})$ is the set of indices corresponding to all the predicates in the subtree `T`. For example, the fixed-size representation for the subtree `john` in logical form `(field parent john)` is just $\boldsymbol{Y}_4$ (indexing from 1, and also counting parentheses because they are also encoded in $\boldsymbol{Y}$).

We model the confidence that the subtree representation $\boldsymbol{y}$ fails to match the meaning of the aligned words $\boldsymbol{x}'$ as

$$P(fail\ to\ match|\boldsymbol{x}', \boldsymbol{y}) = \sigma(d) \tag{8}$$

where σ is the sigmoid function and

$$d = h([\boldsymbol{x}', \boldsymbol{y}, \boldsymbol{x}' \odot \boldsymbol{y}, |\boldsymbol{x}' - \boldsymbol{y}|, \boldsymbol{x}'^T W_3 \boldsymbol{y}]) \tag{9}$$

$\odot$ is element-wise product, $[a, b]$ is concatenation of vectors a and b, and h is a learnable linear transformation. This confidence score is used by Algorithm 2 to decide whether to replace the subtree or to keep it.

4 Training

In training, we maximize the log conditional likelihood of the data. Due to our selection of memory (one example for each grammar rule in G), for a given x there is a unique sequence of retrieval and adaptation actions that lead to the production of the correct y. Specifically, letting a_i be the ith action in the correct sequence of l actions, we define the probability of producing y given x as the probability of taking the sequence of actions $a_1, ..., a_l$. We decompose this joint probability into products of conditional probabilities of taking action i given the previous actions and the input x

$$P(y|x) = \prod_{i=1}^{l} P(a_i|a_1, a_2, ..., a_{i-1}, x) \tag{10}$$

If a_i is a retrieval action,

$$P(a_i|a_1, a_2, ..., a_{i-1}, x) = P(i|\boldsymbol{x}') \qquad (11)$$

where $P(i|\boldsymbol{x}')$ is defined in Equation 3. If a_i is an adaptation decision, i.e. to replace a particular subtree

$$P(a_i|a_1, a_2, ..., a_{i-1}, x) = P(fail\,to\,match|\boldsymbol{x}', \boldsymbol{y}) \qquad (12)$$

where $P(fail\ to\ match|\boldsymbol{x}', \boldsymbol{y})$ is defined in Equation 8.

5 Dataset Generation

Using existing semantic parsing datasets to directly evaluate one-shot parsing is difficult, because evaluation of one-shot parsing requires grouping of utterances by structural similarity to construct domains.

Therefore, we generate our own data [1] based on a synchronous context free grammar. We evaluate our approach on the generated dataset. Although our dataset is synthetic, it has the properties needed to evaluate one-shot parsing:

- clear distinction of domains
- plenty of examples for rules in the old domain
- one example for each rule in the new domain
- a separate evaluation set containing variations of the new rules

It is worth noting that although datasets such as OVERNIGHT (Wang et al., 2015) do have clear domain distinction, they do not have the other properties needed for a good evaluation of one-shot parsing. In the future, we are planning to use crowd-sourcing to rephrase the generated utterances in order to make them more natural.

We have two topics of discourse in the SCFG that we use to generate our data. The first is accessing some field of some object, and the second is setting some field of some object to some value or some field of another object. These are general purpose instructions that can express many common intents, such as looking up the location of a restaurant, getting the phone number of a contact, and setting reminders and alarms.

We have six domains of discourse in the SCFG that we use to generate our data – *person*, *restaurant*, *event*, *course*, *animal*, and *vehicle*. Table 2 includes sample sentences and their logical form for every domain of discourse.

[1]Our dataset can be accessed at `https://github.com/zhichul/lookup-and-adapt-parser-data`.

Since one-shot parsing has two phases, our dataset is slightly different from a typical dataset consisting of a single collection of *utterance-logical form* pairs. In the following space we describe the sets of data generated for each domain. In the next section we describe how we can use this dataset to design two different one-shot parsing evaluation setups.

For each domain d, we define a G_d and randomly generate examples from $domain(G_d)$ to construct an "old" set of examples $D_{old,d}$. We also generate a representative subset $M_{old,d}$ to be used as the memory of our model. As described in § 3.1, $M_{old,d}$ contains one example for each grammar rule in G_d.

In addition, we define a set of new rules G'_d disjoint from G_d for each domain and generate one example $e_{i,d}$ for each new rule $g'_{i,d} \in G'_d$, and store them into the memory $M'_{new,d}$. These are the one-shot examples. The extended memory $M_{new,d} = M_{old,d} \cup M'_{new,d}$.

Finally, we generate evaluation sets $E_{old,d} \subset domain(G_d)$ and $E_{new,d} \subset domain(G_d \cup G'_d) - domain(G_d)$ where $-$ denotes set difference. In our experiments we split the evaluation sets randomly into a development set and a test set of the same size.

Some statistics of our generated dataset is presented in Appendix D. In the next section, we describe how we use this data to generate two different experimental scenarios for evaluating one-shot semantic parsing.

6 Results and Evaluation

We evaluate our approach in six domains, namely person, restaurant, event, course, animal, and vehicle, and in two one-shot parsing scenarios, namely *extension* and *transfer*. In the next section we define these two different scenarios. We compare the performance to a sequence-to-sequence parser with attention which is defined in the following paragraphs.

Baseline Our baseline is a sequence-to-sequence parser with attention. We use a 1-layer Bidirectional LSTM as the encoder for the utterance, and a 1-layer LSTM as the decoder for the logical form. We initialize the decoder with a learned projection of the last hidden state of the encoder. The inputs to the encoder are GloVe

	Domain	Utterance & Logical Form
a_1	*person*	the hometown field of john
		(field (relation hometown) (person john))
a_2	*person*	set her 's parents with classmates
		(set (field (relation parent) (person reference)) (person classmate))
b_1	*restaurant*	irish restaurant instances 's price field
		(field (relation price) (restaurant irish))
b_2	*restaurant*	set address field of all irish restaurant as indian restaurant 's price
		(set (field (relation address) (restaurant irish)) (field (relation price) (restaurant indian)))
c_1	*event*	the start time of lectures
		(field (relation start) (event lecture))
c_2	*event*	set the attendants of that event to organizers field of receptions instances
		(set (field (relation attendant) (event reference)) (field (relation organizer) (event reception)))
d_1	*course*	size of my history course instances
		(field (relation size) (course history))
d_2	*course*	set all history course instances 's prerequisite field as physics course
		(set (field (relation prerequisite) (course history)) (course physics))
e_1	*animal*	life span of all lion instances
		(field (relation span) (animal lion))
e_2	*animal*	set fish instances 's family field with dog
		(set (field (relation family) (animal fish)) (animal dog))
f_1	*vehicle*	the source field of all buses
		(field (relation source) (vehicle bus))
f_2	*vehicle*	set operators of all subways as buses instances
		(set (field (relation operator) (vehicle subway)) (vehicle bus))

Table 2: Sample utterance-logical form pairs from each domain of discourse. Sentences a_1 ... f_1 parse to commands that retrieve a field of an object. Sentences a_2 ... f_2 parse to commands which set a field of an object to a constant or a field of another object. The grammar we use do not constrain the type of the set command. As a consequence, some sentences generated are not semantically correct according to a human interpreter but still they preserve the sentence structure. Sentence b_2, e_2, and f_2 are examples of this phenomena.

word embeddings, and the inputs to the decoder are concatenations of embeddings of logical predicates and context vectors. Context vectors are attention weighted averages of projected encoder states. The attention weights over the utterance for each decoding step t is computed by taking the dot product of the hidden state of the decoder at step t and the projected encoder state for each word in the utterance, normalized using the softmax function. The output of a decoding step t is a probability distribution over all the logical predicates. The distribution is modeled as a dot product between a projection of the overall decoding state at time t, and the embedding for each logical predicate, normalized using the softmax function. The overall decoding state at time t is the concatenation of the hidden state of the decoder at time t and the context vector at time t. Decoding of a logical form given an utterance is done as a greedy search over all possible logical forms to output.

Evaluation Metric Our evaluation metric is parsing accuracy. We define parsing accuracy as the ratio of the correctly parsed utterances in the test set. An utterance is parsed correctly if its generated logical form matches the annotated logical form exactly. E.g. `(field (relation parent) (person john))` is a correct parse of "parents of John". We report accuracy percentage in Tables 3 and 4.

6.1 Discussion

We design two different scenarios for one-shot parsing evaluation, namely the *extension* and the *transfer* scenarios. We define and discuss the results of each scenario in the following sections.

Extension This scenario tests one-shot parsing of new rules within the same domain that the model is trained on. In this case, we hold out several rules and their corresponding utterance-logical form pairs for evaluation and train on the remaining ones.

The results of this experiment are presented in Table 3. Each column indicates the evaluation results on each domain. The *full* scores refer to the overall percentage of correct parses on the entire test data, the *d=2* scores refer to the percentage of correct parses on examples with logical form depth 2 and the *d=3* scores refer to the percentage of correct parses on examples with logical form depth 3.

As it can be seen in the table, our model is able to consistently improve the sequence-to-sequence baseline on all the domains by a significant margin of 68.8%.

Transfer This scenario tests one-shot parsing of new rules in a domain different from the one that the model is trained on. For example, in the *transfer* scenario for domain *person* the model is trained on examples from the other domains *restaurant*, *event*, *course*, *animal*, and *vehicle* and tested on samples from domain *person*. This is harder compared to the extension setup since it requires generalization to a completely new domain.

The results of this section are reported in Table 4. As it can be seen, the performance is improved by up to 36.6% compared to the baseline model.

In order to have a better understanding of the model and indicate why it is performing comparable to the baseline on some of the domains setting, we carry out a model analysis in the next section.

Model Analysis We add two variations of our model ORACLE-DISCRIM and PRETRAIN-ENC for the transfer scenario to see the performance improvements gained from replacing different components of our model with an oracle/near oracle. This identifies potential bottlenecks of the model.

In ORACLE-DISCRIM, we load a trained model from LOOKUPADAPT but replace the discriminator with an oracle during evaluation. As the numbers in row 3 of Table 4 show, using an oracle discriminator improves the model performance by more than 10% in the *vehicle* domain and *course* domain, and by smaller amounts in the other domains. On one hand, this suggests that the discriminator has room for improvement. On the other hand, the numbers suggest that there is still a much larger room for improvement for the encoder, aligner, and look-up components. Our hypothesis is that the encoder components are likely the main bottleneck because during evaluation in the *transfer* scenario they have to produce good representations for utterances and logical forms very different from the ones which they have seen during training. We found evidence supporting this hypothesis in the PRETRAIN-ENC variation.

In PRETRAIN-ENC, we pretrain our model on all the domains, which ensures that the two encoders see all domains. We then use the encoder parameters of this all-domain pretrained model to initialize the encoders in the experiments for the *transfer* scenario. We then fix the encoders and train only the aligner, discriminator, and look-up parameters. This results in near perfect generalization to the test domain by the aligner, discriminator, and look-up components, which are trained only on the other domains. This suggests that the encoder is the main bottleneck of our model since using a near-oracle version boosts its performance to perfect. The results for this variation is shown in row 4 of Table 4.

7 Related Work

Semantic parsing is the task of mapping utterances to a formal representation of their meaning. Researchers have used grammar-based methods as well as machine learning-based methods to address this problem. Grammar-based parsers work by having a set of grammar rules that are either learned or hand-written, an algorithm for generating a set of candidate logical forms by recursive application of the grammar rules, and a criterion for picking the best candidate logical form within that set (Liang and Potts, 2015; Zettlemoyer and Collins, 2005, 2007). However, they are brittle to the flexibility of language. To improve this limitation, supervised sequence-based neural semantic parsers have been proposed (Dong and Lapata, 2016). Herzig and Berant (2017) improved the performance of neural semantic parsers by training over multiple knowledge bases and providing the domain encoding at decoding time. In addition to supervised learning, reinforcement learning methods for neural semantic parsing have been explored in (Zhong et al., 2017).

Retrieve-and-edit style semantic parsing is gaining popularity. Hashimoto et al. (2018) proposed a retrieve and edit framework that can efficiently learn to embed utterances in a task-dependent way for easy editing. Our work differs in that we perform hierarchical retrievals and edits, and that we evaluate on cross-domain data and focus on one-shot semantic parsing. It is worth noting that retrieve-and-edit as a general framework is not limited to semantic parsing and is applicable to other areas such as sentence generation. (Guu et al., 2018) and machine translation

		extension					
		person	restaurant	event	course	animal	vehicle
Sequence-To-Sequence	full	20.1	25.1	26.1	19.9	25.0	18.8
	d=2	25.0	18.1	20.0	31.8	26.5	25.9
	d=3	13.9	37.3	42.4	0.0	21.5	5.8
LOOKUPADAPT	full	**45.9**	**61.7**	**85.5**	**79.2**	**63.6**	**87.6**
	d=2	56.7	85.4	94.5	86.7	70.1	93.6
	d=3	27.8	20.0	61.7	66.7	46.5	76.5

Table 3: Test Accuracy on the **extension** Dataset.

		transfer					
		person	restaurant	event	course	animal	vehicle
Sequence-To-Sequence	full	5.2	29.2	5.2	**19.9**	**35.5**	**12.5**
	d=2	6.4	41.7	4.9	27.5	44.9	15.6
	d=3	3.1	8.4	5.7	6.0	13.8	6.3
LOOKUPADAPT	full	**41.8**	**43.8**	**8.4**	16.7	28.3	11.5
	d=2	52.5	53.4	13.2	24.2	37.3	14.1
	d=3	21.4	27.8	0.0	3.0	6.9	6.3
ORACLE-DISCRIM	full	46.9	45.8	12.5	29.2	33.3	25.0
	d=2	55.6	56.7	18.0	41.8	44.8	29.7
	d=3	30.3	27.8	2.9	5.9	6.9	15.6
PRETRAIN-ENC	full	98.0	100.0	100.0	99.0	100.0	96.9
	d=2	100.0	100.0	100.0	100.0	100.0	100.0
	d=3	94.0	100.0	100.0	97.1	100.0	90.1

Table 4: Test Accuracy on the **transfer** Dataset.

Gu et al. (2018).

Another line of research maps semantic parsing under cross-domain setting to a domain adaptation problem (Su and Yan, 2017). In their work, the model is trained on a certain domain and then fine tuned to parse data from another domain. This is in essence different from our work in that we do not adapt the model, rather we adapt seen samples to form parses of new samples. Moreover, We do not fine-tune any part of the model in the new domain and focus on one-shot semantic parsing.

Most of these models need many data-points to train. Therefore, there has been recent attempts at zero-shot semantic parsing. Dadashkarimi et al. (2018) proposed a transfer learning approach where a domain label is predicted first and then the parse. Ferreira et al. (2015) and Herzig and Berant (2018) proposed slot-filling methods for semantic parsing based on general word embeddings. Bapna et al. (2017) focuses on zero-shot frame semantic parsing by leveraging the description of slots to be filled.

8 Conclusion and Future Work

As speech recognition technologies mature, more computing devices support spoken instructions via a conversational agent. However, most agents do not adapt to the phrasing and interest of a specific end user. It has recently been shown that new functionalities can be added to an agent from user instruction (Azaria et al., 2016; Labutov et al., 2018). However, the user instruction only provides one instance of a general instruction template and the agent is challenged to generalize to variations of the instance given during instruction. We define the *one-shot parsing* task for measuring a semantic parser's ability to generalize to new instances of user-taught commands from only one example. We propose a new semantic parser architecture that learns a general strategy of retrieving seen utterances similar to an unseen utterance and adapting the logical forms of seen utterances to fit the unseen utterance. Our results show an improvement of up to 68.8% on one-shot parsing under two different evaluation settings compared to the baselines. We found that the BiLSTM encoders are likely bottlenecks for the model. Some future directions include exploring the effects of contents in the memory, automating memory extraction from dataset, and improving the encoder.

Acknowledgments

This work was supported by AFOSR under grant FA95501710218, NSF under grant IIS1814472, and a Faculty award from J. P. Morgan.The authors would like to sincerely thank Bishan Yang for the initial discussions and ideas related to model architecture, and to Kathryn Mazaitis for the brainstorming sessions on the limitations of the model and future directions.

References

Amos Azaria, Jayant Krishnamurthy, and Tom Mitchell. 2016. Instructable intelligent personal agent. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2681–2689.

Ankur Bapna, Gokhan Tur, Dilek Hakkani-Tur, and Larry Heck. 2017. Towards Zero-Shot Frame Semantic Parsing for Domain Scaling. *arXiv e-prints*, page arXiv:1707.02363.

Yoshua Bengio and Yann LeCun, editors. 2015. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

David Chiang. 2006. An introduction to synchronous grammars.

Javid Dadashkarimi, Alexander Fabbri, Sekhar Tatikonda, and Dragomir R. Radev. 2018. Zero-shot Transfer Learning for Semantic Parsing. *arXiv e-prints*, page arXiv:1808.09889.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

Emmanuel Ferreira, Bassam Jabaian, and Fabrice Lefèvre. 2015. Zero-shot semantic parser for spoken language understanding. In *Sixteenth Annual Conference of the International Speech Communication Association*.

Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor OK Li. 2018. Search engine guided neural machine translation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Kelvin Guu, Tatsunori B Hashimoto, Yonatan Oren, and Percy Liang. 2018. Generating sentences by editing prototypes. *Transactions of the Association for Computational Linguistics*, 6:437–450.

Tatsunori B Hashimoto, Kelvin Guu, Yonatan Oren, and Percy S Liang. 2018. A retrieve-and-edit framework for predicting structured outputs. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10052–10062. Curran Associates, Inc.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1693–1701. Curran Associates, Inc.

Jonathan Herzig and Jonathan Berant. 2017. Neural semantic parsing over multiple knowledge-bases. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 623–628, Vancouver, Canada. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2018. Decoupling structure and lexicon for zero-shot semantic parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1619–1629, Brussels, Belgium. Association for Computational Linguistics.

Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1378–1387, New York, New York, USA. PMLR.

Igor Labutov, Shashank Srivastava, and Tom Mitchell. 2018. LIA: A natural language programmable personal assistant. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 145–150, Brussels, Belgium. Association for Computational Linguistics.

Percy Liang and Christopher Potts. 2015. Bringing machine learning and compositional semantics together. *Annual Review of Linguistics*, 1:355–376.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Yu Su and Xifeng Yan. 2017. Cross-domain semantic parsing via paraphrasing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1235–1246, Copenhagen, Denmark. Association for Computational Linguistics.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. 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)*, pages 1332–1342, Beijing, China. Association for Computational Linguistics.

W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig. 2016. Achieving Human Parity in Conversational Speech Recognition. *arXiv e-prints*, page arXiv:1610.05256.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687, Prague, Czech Republic. Association for Computational Linguistics.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, pages 658–666, Arlington, Virginia, United States. AUAI Press.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *arXiv e-prints*, page arXiv:1709.00103.

A Sample Grammar and Data Generation Grammar

A sample SCFG is given in Table 5 to facilitate the definition of *domain*.

This sample SCFG is not to be confused with the grammar used to generate our dataset. In comparison, the SCFG for data generation contains a larger number of entities, relations, and operations than the sample SCFG. Selected rules from the person domain and restaurant domain are provided below in Table 7 and Table 8.

B Contents of the Memory

In this paper, we assume the memory is provided to the model by a human expert. We picked the strategy of selecting one example for each grammar rule g in the synchronous context free grammar G that generated the data. It makes the formulation of maximum conditional likelihood learning straightforward, because this strategy produces a memory such that for any given utterance there is a unique sequence of *look-up* and *adapt* actions that produce the correct logical form. Recall that an example of a grammar rule g is an *utterance-logical form* pair generated using g and other rules of G. In particular, we choose an example for g such that the top level predicate of the example's logical form matches the top level predicate of the target of g. For example, we will choose to put

⟨ John 's parents, `(field parent john)` ⟩

in memory for the grammar rule

$$g_1 := \textbf{FIELD} \rightarrow \langle \textbf{PSN}_2 \text{ 's } \textbf{PSN\_REL}_1, \texttt{(field } \textbf{PSN\_REL}_1 \ \textbf{PSN}_2\texttt{)} \rangle$$

because `field` is the root predicate of both the example logical form `(field parent john)` and the target of the rule `(field` $\textbf{PSN\_REL}_1$ $\textbf{PSN}_2$`)`. Here's another example. For a terminal rule like

$$g_2 := \textbf{PSN} \rightarrow \langle \textit{John}, \texttt{(person john)} \rangle$$

we put the pair ⟨ John, `(person john)` ⟩ in memory, because the top level predicate of the logical form (`person`) matches that of g_2's target (`person`). We would not put for instance ⟨ John 's children, `(field children john)`⟩ because the top level predicate of the logical form (`field`) does not match that of g_2's target (`person`).

We do note that the requirement of manually selected examples to put in memory is a big limitation of the current approach. In future work, we wish to automate the building of memory.

C The trace of an example run of Look-up and Adapt

A trace of an example run of the LOOKUPANDADAPT algorithm is given in Table 9. The utterance of this run is based on the sample SCFG grammar given in Table 5. Given a memory with 3 examples and a new utterance "John's Parents", the LOOKUPANDADAPT algorithm will look-up an example from the memory that matches the attended parts of the new utterance and recursively adapt the logical form of the retrieved example to produce a correct logical form that fits the new utterance.

D Statistics of the generated data

We provide a breakdown of the number of examples used in training and evaluation for the six domains of discourse at different depths in Table 6. The definition for the different groups of examples and their use is explained in § 5 for dataset generation.

Left-hand-side		Source(Utterance)	Target(Logical Form)
PSN	$\rightarrow$	John	`john`
PSN	$\rightarrow$	Mary	`mary`
PSN_REL	$\rightarrow$	parents	`parent`
PSN_REL	$\rightarrow$	children	`child`
FIELD	$\rightarrow$	**PSN_REL**$_1$ of **PSN**$_2$	`(field` **PSN_REL**$_1$ **PSN**$_2$`)`
FIELD	$\rightarrow$	**PSN**$_2$'s **PSN_REL**$_1$	`(field` **PSN_REL**$_1$ **PSN**$_2$`)`
S	$\rightarrow$	**PSN**$_1$	**PSN**$_1$
S	$\rightarrow$	**PSN_REL**$_1$	**PSN_REL**$_1$
S	$\rightarrow$	**FIELD**$_1$	**FIELD**$_1$

Table 5: Sample SCFG G_{sample}. The subscripts indicate linking of terminals/nonterminals in source and target. A SCFG rule can only be applied to linked terminals/nonterminals together.

		domain of discourse					
		person	restaurant	event	course	animal	vehicle
D_{old}	full	737	810	911	960	801	729
	d=2	417	490	591	640	481	409
	d=3	320	320	320	320	320	320
E_{olddev}	full	96	96	96	96	96	96
	d=2	65	68	67	66	61	64
	d=3	31	28	29	30	35	32
$E_{oldtest}$	full	96	96	96	96	96	96
	d=2	63	60	61	62	67	64
	d=3	33	36	35	34	29	32
E_{newdev}	full	96	96	96	96	96	96
	d=2	68	67	58	68	60	66
	d=3	28	29	38	28	36	30
$E_{newtest}$	full	96	96	96	96	96	96
	d=2	60	61	70	60	68	62
	d=3	36	35	26	36	28	34
M_{old}	full	23	16	16	16	16	15
	d=1	20	13	13	13	13	12
	d=2	2	2	2	2	2	2
	d=3	1	1	1	1	1	1
M_{new}	full	27	20	20	20	20	19
	d=1	24	17	17	17	17	16
	d=2	2	2	2	2	2	2
	d=3	1	1	1	1	1	1

Table 6: Dataset Statistics: sizes of each component with depth breakdown. Please refer to § 5 for the precise definition of each row of the table. In the *extension* scenario, for each domain of discourse d, the model is trained on examples from $D_{old,d}$ using $M_{old,d}$ as memory and is evaluated on examples from $E_{newdev,d}$ and $E_{newtest,d}$ using $M_{new,d}$ as memory. In the *transfer* scenario, for each domain of discourse d, the model is trained on $\cup_{d' \neq d} D_{old,d'}$ using $\cup_{d' \neq d} M_{old,d'}$ as memory, and is evaluated on $E_{olddev,d}$ and $E_{oldtest,d}$ using $\cup_{\forall d'} M_{old,d'}$ as memory.

Left-hand-side		Source(Utterance)	Target(Logical Form)
PSN	$\rightarrow$	john	`(person john)`
PSN	$\rightarrow$	mary	`(person mary)`
PSN	$\rightarrow$	colleagues	`(person colleague)`
PSN	$\rightarrow$	professors	`(person professor)`
PSN_REL	$\rightarrow$	parents	`(relation parent)`
PSN_REL	$\rightarrow$	children	`(relation child)`
PSN_REL	$\rightarrow$	hometown	`(relation hometown)`
PSN_REL	$\rightarrow$	salary	`(relation salary)`
FIELD	$\rightarrow$	$\textbf{PSN\_REL}_1$ of $\textbf{PSN}_2$	`(field` $\textbf{PSN\_REL}_1$ $\textbf{PSN}_2$`)`
FIELD	$\rightarrow$	$\textbf{PSN}_2$'s $\textbf{PSN\_REL}_1$	`(field` $\textbf{PSN\_REL}_1$ $\textbf{PSN}_2$`)`
CMD	$\rightarrow$	set $\textbf{FIELD}_1$ to $\textbf{VALUE}_2$	`(set` $\textbf{FIELD}_1$ $\textbf{VALUE}_2$`)`
VALUE	$\rightarrow$	$\textbf{FIELD}_1$	$\textbf{FIELD}_1$
VALUE	$\rightarrow$	$\textbf{PSN}_1$	$\textbf{PSN}_1$
S	$\rightarrow$	$\textbf{FIELD}_1$	$\textbf{FIELD}_1$
S	$\rightarrow$	$\textbf{CMD}_1$	$\textbf{CMD}_1$

Table 7: This is a selected subset of the grammar used to generate data for the *person* domain. Rules not shown here include those for introducing flexibility in language and rules for additional entities and relations.

Left-hand-side		Source(Utterance)	Target(Logical Form)
RST	$\rightarrow$	chinese restaurant	`(restaurant chinese)`
RST	$\rightarrow$	italian restaurant	`(restaurant italian)`
RST	$\rightarrow$	french restaurant	`(restaurant french)`
RST	$\rightarrow$	german restaurant	`(restaurant german)`
RST_REL	$\rightarrow$	address	`(relation address)`
RST_REL	$\rightarrow$	reviews	`(relation reviews)`
RST_REL	$\rightarrow$	dishes	`(relation dishes)`
RST_REL	$\rightarrow$	price	`(relation price)`
FIELD	$\rightarrow$	$\textbf{RST\_REL}_1$ of $\textbf{RST}_2$	`(field` $\textbf{RST\_REL}_1$ $\textbf{RST}_2$`)`
FIELD	$\rightarrow$	$\textbf{PSN}_2$'s $\textbf{RST\_REL}_1$	`(field` $\textbf{RST\_REL}_1$ $\textbf{RST}_2$`)`
CMD	$\rightarrow$	set $\textbf{FIELD}_1$ to $\textbf{VALUE}_2$	`(set` $\textbf{FIELD}_1$ $\textbf{VALUE}_2$`)`
VALUE	$\rightarrow$	$\textbf{FIELD}_1$	$\textbf{FIELD}_1$
VALUE	$\rightarrow$	$\textbf{RST}_1$	$\textbf{RST}_1$
S	$\rightarrow$	$\textbf{FIELD}_1$	$\textbf{FIELD}_1$
S	$\rightarrow$	$\textbf{CMD}_1$	$\textbf{CMD}_1$

Table 8: This is a selected subset of the grammar used to generate data for the *restaurant* domain. Rules not shown here include those for introducing flexibility in language and rules for additional entities and relations.

Func	Description	New utterance and current logical form	Memory
LkUpNAdpt	Initial call always has uniform attention.	**[John ’s parents]** None	⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary ’s parents , `(field parent mary)`⟩
LookUp	Retrieves entry ⋆ from memory by utterance similarity.	**[John ’s parents]** `(field parent mary)`	⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⋆ ⟨Mary ’s parents , `(field parent mary)`⟩
::Adapt	Adapt first child. `parent` is aligned to **parents**. YES, they match, keep `parent`. There’s no children. Return.	John ’s **[parents]** `(field [parent] mary)`	⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary ’s parents , `(field parent mary)`⟩
::Adapt	Adapt second child. `mary` is aligned to **John**. NO, they don’t match. Find replacement and return it.	**[John]** ’s parents `(field parent [mary])`	⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary ’s parents , `(field parent mary)`⟩
::::LkUpNAdpt	Find replacement for `mary`.	**[John]** ’s parents `(field parent mary)`	⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary ’s parents , `(field parent mary)`⟩
::::LookUp	Retrieves entry ⋆ from memory by utterance similarity.	**[John]** ’s parents `(field parent mary)`	⋆ ⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary ’s parents , `(field parent mary)`⟩
::::LkUpNAdpt	There’s no children. Return `john`	**[John]** ’s parents `(field parent mary)`	⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary ’s parents , `(field parent mary)`⟩
LkUpNAdpt	Replace `mary` with `john`. Exhausted all children, return the current logical form.	**[John ’s parents]** `(field parent john)`	⟨John, `john`⟩ ⟨Mary, `mary`⟩ ⟨Mary ’s parents , `(field parent mary)`⟩

Table 9: Trace of a sample run of the algorithm with sample memory, utterance, and attention. The attention weights (argument $\boldsymbol{w}'$ to LookUpAndAdapt and variable $\boldsymbol{w}$ in Adapt) are visualized as brackets. Bracketed words in the *utterance* receive more attention, un-bracketed words receive less attention. For Adapt calls, bracketed words in the *logical form* indicate the current subtree of the whole logical form tree that the algorithm is looking at (the argument `T` to Adapt). Colons are used to indicate the depth of the call stack.