Weakly Supervised Semantic Parsing with Abstract Examples

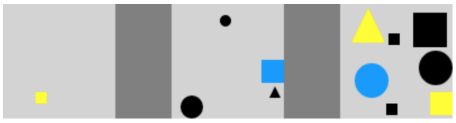
Omer Goldman[*], **Veronica Latcinnik**[*], **Udi Naveh**[*], **Amir Globerson**, **Jonathan Berant**
Tel-Aviv University
{omergoldman@mail,veronical@mail,
ehudnave@mail,gamir@post,joberant@cs}.tau.ac.il

Abstract

Training semantic parsers from weak supervision (denotations) rather than strong supervision (programs) complicates training in two ways. First, a large *search* space of potential programs needs to be explored at training time to find a correct program. Second, *spurious* programs that accidentally lead to a correct denotation add noise to training. In this work we propose that in closed worlds with clear semantic types, one can substantially alleviate these problems by utilizing an abstract representation, where tokens in both the language utterance and program are lifted to an abstract form. We show that these abstractions can be defined with a handful of lexical rules and that they result in sharing between different examples that alleviates the difficulties in training. To test our approach, we develop the first semantic parser for CNLVR, a challenging visual reasoning dataset, where the search space is large and overcoming spuriousness is critical, because denotations are either TRUE or FALSE, and thus random programs are likely to lead to a correct denotation. Our method substantially improves performance, and reaches 82.5% accuracy, a 14.7% absolute accuracy improvement compared to the best reported accuracy so far.

1 Introduction

The goal of semantic parsing is to map language utterances to executable programs. Early work on statistical learning of semantic parsers utilized supervised learning, where training examples included pairs of language utterances and programs (Zelle and Mooney, 1996; Kate et al., 2005; Zettlemoyer and Collins, 2005, 2007). However, collecting such training examples at scale has quickly turned out to be difficult, because expert annotators who are familiar with formal languages are required. This has led to a body of work on weakly-supervised semantic parsing (Clarke et al., 2010; Liang et al., 2011; Krishnamurthy and Mitchell, 2012; Kwiatkowski et al., 2013; Berant et al., 2013; Cai and Yates, 2013; Artzi and Zettlemoyer, 2013). In this setup, training examples correspond to utterance-denotation pairs, where a *denotation* is the result of executing a program against the environment (see Fig. 1). Naturally, collecting denotations is much easier, because it can be performed by non-experts.



k :[[{y_loc: ..., color: 'Black', type: 'square', x_loc: ... size: 20}, ...]]

x :*There is a small yellow item not touching any wall*

y :`True`

z :`Exist(Filter(ALL_ITEMS,` λx`.And(And(IsYellow(`x`), IsSmall(`x`)), Not(IsTouchingWall(`x`, Side.Any))))))`

Figure 1: Overview of our visual reasoning setup for the CNLVR dataset. Given an image rendered from a KB k and an utterance x, our goal is to parse x to a program z that results in the correct denotation y. Our training data includes (x, k, y) triplets.

Training semantic parsers from denotations rather than programs complicates training in two ways: (a) *Search*: The algorithm must learn to search through the huge space of programs at training time, in order to find the correct program. This is a difficult search problem due to the combinatorial nature of the search space. (b) *Spurious-*

[*] Authors equally contributed to this work.

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 1809–1819
Melbourne, Australia, July 15 - 20, 2018.

ness: Incorrect programs can lead to correct denotations, and thus the learner can go astray based on these programs. Of the two mentioned problems, spuriousness has attracted relatively less attention (Pasupat and Liang, 2016; Guu et al., 2017).

Recently, the Cornell Natural Language for Visual Reasoning corpus (CNLVR) was released (Suhr et al., 2017), and has presented an opportunity to better investigate the problem of spuriousness. In this task, an image with boxes that contains objects of various shapes, colors and sizes is shown. Each image is paired with a complex natural language statement, and the goal is to determine whether the statement is true or false (Fig. 1). The task comes in two flavors, where in one the input is the image (pixels), and in the other it is the knowledge-base (KB) from which the image was synthesized. Given the KB, it is easy to view CNLVR as a semantic parsing problem: our goal is to translate language utterances into programs that will be executed against the KB to determine their correctness (Johnson et al., 2017b; Hu et al., 2017). Because there are only two return values, it is easy to generate programs that execute to the right denotation, and thus spuriousness is a major problem compared to previous datasets.

In this paper, we present the first semantic parser for CNLVR. Semantic parsing can be coarsely divided into a *lexical* task (i.e., mapping words and phrases to program constants), and a *structural* task (i.e., mapping language composition to program composition operators). Our core insight is that in closed worlds with clear semantic types, like spatial and visual reasoning, we can manually construct a small lexicon that clusters language tokens and program constants, and create a partially abstract representation for utterances and programs (Table 1) in which the lexical problem is substantially reduced. This scenario is ubiquitous in many semantic parsing applications such as calendar, restaurant reservation systems, housing applications, etc: the formal language has a compact semantic schema and a well-defined typing system, and there are canonical ways to express many program constants.

We show that with abstract representations we can share information across examples and better tackle the search and spuriousness challenges. By pulling together different examples that share the same abstract representation, we can identify programs that obtain high reward across multiple examples, thus reducing the problem of spuriousness. This can also be done at search time, by augmenting the search state with partial programs that have been shown to be useful in earlier iterations. Moreover, we can annotate a small number of abstract utterance-program pairs, and automatically generate training examples, that will be used to warm-start our model to an initialization point in which search is able to find correct programs.

We develop a formal language for visual reasoning, inspired by Johnson et al. (2017b), and train a semantic parser over that language from weak supervision, showing that abstract examples substantially improve parser accuracy. Our parser obtains an accuracy of 82.5%, a 14.7% absolute accuracy improvement compared to state-of-the-art. All our code is publicly available at `https://github.com/udiNaveh/nlvr_tau_nlp_final_proj`.

2 Setup

Problem Statement Given a training set of N examples $\{(x_i, k_i, y_i)\}_{i=1}^{N}$, where x_i is an utterance, k_i is a KB describing objects in an image and $y_i \in \{\text{TRUE}, \text{FALSE}\}$ denotes whether the utterance is true or false in the KB, our goal is to learn a semantic parser that maps a new utterance x to a program z such that when z is executed against the corresponding KB k, it yields the correct denotation y (see Fig. 1).

Programming language The original KBs in CNLVR describe an image as a set of objects, where each object has a color, shape, size and location in absolute coordinates. We define a programming language over the KB that is more amenable to spatial reasoning, inspired by work on the CLEVR dataset (Johnson et al., 2017b). This programming language provides access to functions that allow us to check the size, shape, and color of an object, to check whether it is touching a wall, to obtain sets of items that are above and below a certain set of items, etc.[1] More formally, a program is a sequence of tokens describing a possibly recursive sequence of function applications in prefix notation. Each token is either a function with fixed arity (all functions have either one or two arguments), a constant, a variable or a λ term used to define Boolean functions. Functions, constants and variables have one of the following

[1] We leave the problem of learning the programming language functions from the original KB for future work.

x: *"There are exactly 3 yellow squares touching the wall."*
z: `Equal(3, Count(Filter(ALL_ITEMS, λx. And (And (IsYellow(x), IsSquare(x), IsTouchingWall(x))))))`
$\bar{x}$: *"There are C-QuantMod C-Num C-Color C-Shape touching the wall."*
$\bar{z}$: `C-QuantMod(C-Num, Count(Filter(ALL_ITEMS, λx. And (And (IsC-Color(x), IsC-Shape(x), IsTouchingWall(x))))))`

Table 1: An example for an utterance-program pair (x, z) and its abstract counterpart $(\bar{x}, \bar{z})$

x: *"There is a small yellow item not touching any wall."*
z: `Exist(Filter(ALL_ITEMS, λx.And(And(IsYellow(x), IsSmall(x)), Not(IsTouchingWall(x, Side.Any)))))`
x: *"One tower has a yellow base."*
z: `GreaterEqual(1, Count(Filter(ALL_ITEMS, λx.And(IsYellow(x), IsBottom(x)))))`

Table 2: Examples for utterance-program pairs. Commas and parenthesis provided for readability only.

atomic types: `Int`, `Bool`, `Item`, `Size`, `Shape`, `Color`, `Side` (sides of a box in the image); or a composite type `Set(?)`, and `Func(?,?)`. Valid programs have a return type `Bool`. Tables 1 and 2 provide examples for utterances and their correct programs. The supplementary material provides a full description of all program tokens, their arguments and return types.

Unlike CLEVR, CNLVR requires substantial set-theoretic reasoning (utterances refer to various aspects of sets of items in one of the three boxes in the image), which required extending the language described by Johnson et al. (2017b) to include set operators and lambda abstraction. We manually sampled 100 training examples from the training data and estimate that roughly 95% of the utterances in the training data can be expressed with this programming language.

3 Model

We base our model on the semantic parser of Guu et al. (2017). In their work, they used an encoder-decoder architecture (Sutskever et al., 2014) to define a distribution $p_\theta(z \mid x)$. The utterance x is encoded using a bi-directional LSTM (Hochreiter and Schmidhuber, 1997) that creates a contextualized representation h_i for every utterance token x_i, and the decoder is a feed-forward network combined with an attention mechanism over the encoder outputs (Bahdanau et al., 2015). The feed-forward decoder takes as input the last K tokens that were decoded.

More formally the probability of a program is the product of the probability of its tokens given the history: $p_\theta(z \mid x) = \prod_t p_\theta(z_t \mid x, z_{1:t-1})$, and the probability of a decoded token is computed as follows. First, a Bi-LSTM encoder converts the input sequence of utterance embeddings into a sequence of forward and backward states $h_1^{\{F,B\}}, \ldots, h_{|x|}^{\{F,B\}}$. The utterance representation $\hat{x}$ is $\hat{x} = [h_{|x|}^F; h_1^B]$. Then decoding produces the program token-by-token:

$$q_t = \text{ReLU}(W_q[\hat{x}; \hat{v}; z_{t-K-1:t-1}]),$$
$$\alpha_{t,i} \propto \exp(q_t^\top W_\alpha h_i)\ ,\ c_t = \sum_i \alpha_{t,i} h_i,$$
$$p_\theta(z_t \mid x, z_{1:t-1}) \propto \exp(\phi_{z_t}^\top W_s[q_t; c_t]),$$

where ϕ_z is an embedding for program token z, $\hat{v}$ is a bag-of-words vector for the tokens in x, $z_{i:j} = (z_i, \ldots, z_j)$ is a history vector of size K, the matrices W_q, W_α, W_s are learned parameters (along with the LSTM parameters and embedding matrices), and ';' denotes concatenation.

Search: Searching through the large space of programs is a fundamental challenge in semantic parsing. To combat this challenge we apply several techniques. First, we use beam search at decoding time and when training from weak supervision (see Sec. 4), similar to prior work (Liang et al., 2017; Guu et al., 2017). At each decoding step we maintain a beam B of program prefixes of length n, expand them exhaustively to programs of length $n+1$ and keep the top-$|B|$ program prefixes with highest model probability.

Second, we utilize the semantic typing system to only construct programs that are syntactically valid, and substantially prune the program search space (similar to type constraints in Krishnamurthy et al. (2017); Xiao et al. (2016); Liang et al. (2017)). We maintain a stack that keeps track of the expected semantic type at each decoding step. The stack is initialized with the type `Bool`. Then, at each decoding step, only tokens that return the semantic type at the top of the stack are allowed, the stack is popped, and if the decoded token is a function, the semantic types of its arguments are pushed to the stack. This dramatically reduces the search space and guarantees that only syntactically valid programs will be produced. Fig. 2 illustrates the state of the stack when decoding a program for an input utterance.

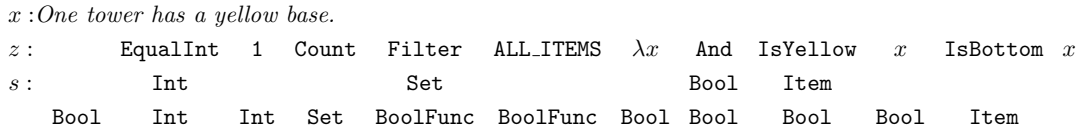


Figure 2: An example for the state of the type stack s while decoding a program z for an utterance x.

Given the constrains on valid programs, our model $p'_\theta(z \mid x)$ is defined as:

$$\prod_t \frac{p_\theta(z_t \mid x, z_{1:t-1}) \cdot \mathbb{1}(z_t \mid z_{1:t-1})}{\sum_{z'} p_\theta(z' \mid x, z_{1:t-1}) \cdot \mathbb{1}(z' \mid z_{1:t-1})},$$

where $\mathbb{1}(z_t \mid z_{1:t-1})$ indicates whether a certain program token is valid given the program prefix.

Discriminative re-ranking: The above model is a locally-normalized model that provides a distribution for every decoded token, and thus might suffer from the label bias problem (Andor et al., 2016; Lafferty et al., 2001). Thus, we add a globally-normalized re-ranker $p_\psi(z \mid x)$ that scores all $|B|$ programs in the final beam produced by $p'_\theta(z \mid x)$. Our globally-normalized model is:

$$p^g_\psi(z \mid x) \propto \exp(s_\psi(x, z)),$$

and is normalized over all programs in the beam. The scoring function $s_\psi(x, z)$ is a neural network with identical architecture to the locally-normalized model, except that (a) it feeds the decoder with the candidate program z and does not generate it. (b) the last hidden state is inserted to a feed-forward network whose output is $s_\psi(x, z)$. Our final ranking score is $p'_\theta(z|x)p^g_\psi(z \mid x)$.

4 Training

We now describe our basic method for training from weak supervision, which we extend upon in Sec. 5 using abstract examples. To use weak supervision, we treat the program z as a latent variable that is approximately marginalized. To describe the objective, define $R(z, k, y) \in \{0, 1\}$ to be one if executing program z on KB k results in denotation y, and zero otherwise. The objective is then to maximize $p(y \mid x)$ given by:

$$\sum_{z \in \mathcal{Z}} p'_\theta(z \mid x) p(y \mid z, k) = \sum_{z \in \mathcal{Z}} p'_\theta(z \mid x) R(z, k, y)$$
$$\approx \sum_{z \in B} p'_\theta(z \mid x) R(z, k, y)$$

where $\mathcal{Z}$ is the space of all programs and $B \subset \mathcal{Z}$ are the programs found by beam search.

In most semantic parsers there will be relatively few z that generate the correct denotation y. However, in CNLVR, y is binary, and so spuriousness is a central problem. To alleviate it, we utilize a property of CNLVR: the same utterance appears 4 times with 4 different images.[2] If a program is spurious it is likely that it will yield the wrong denotation in one of those 4 images.

Thus, we can re-define each training example to be $(x, \{(k_j, y_j)\}_{j=1}^4)$, where each utterance x is paired with 4 different KBs and the denotations of the utterance with respect to these KBs. Then, we maximize $p(\{y_j\}_{j=1}^4 \mid x,)$ by maximizing the objective above, except that $R(z, \{k_j, y_j\}_{j=1}^4) = 1$ iff the denotation of z is correct for **all** four KBs. This dramatically reduces the problem of spuriousness, as the chance of randomly obtaining a correct denotation goes down from $\frac{1}{2}$ to $\frac{1}{16}$. This is reminiscent of Pasupat and Liang (2016), where random permutations of Wikipedia tables were shown to crowdsourcing workers to eliminate spurious programs.

We train the discriminative ranker analogously by maximizing the probability of programs with correct denotation $\sum_{z \in B} p^g_\psi(z \mid x) R(z, k, y)$.

This basic training method fails for CNLVR (see Sec. 6), due to the difficulties of search and spuriousness. Thus, we turn to learning from abstract examples, which substantially reduce these problems.

5 Learning from Abstract Examples

The main premise of this work is that in closed, well-typed domains such as visual reasoning, the main challenge is handling language compositionality, since questions may have a complex and nested structure. Conversely, the problem of mapping lexical items to functions and constants in the programming language can be substantially alleviated by taking advantage of the compact KB schema and typing system, and utilizing a

[2] We used the KBs in CNLVR, for which there are 4 KBs per utterance. When working over pixels there are 24 images per utterance, as 6 images were generated from each KB.

Utterance	Program	Cluster	#
"yellow"	`IsYellow`	`C-Color`	3
"big"	`IsBig`	`C-Size`	3
"square"	`IsSquare`	`C-Shape`	4
"3"	`3`	`C-Num`	2
"exactly"	`EqualInt`	`C-QuantMod`	5
"top"	`Side.Top`	`C-Location`	2
"above"	`GetAbove`	`C-SpaceRel`	6
		Total:	**25**

Table 3: Example mappings from utterance tokens to program tokens for the seven clusters used in the abstract representation. The rightmost column counts the number of mapping in each cluster, resulting in a total of 25 mappings.

small lexicon that maps prevalent lexical items into typed program constants. Thus, if we abstract away from the actual utterance into a partially abstract representation, we can combat the search and spuriousness challenges as we can generalize better across examples in small datasets.

Consider the utterances:

1. *"There are exactly 3 yellow squares touching the wall."*
2. *"There are at least 2 blue circles touching the wall."*

While the surface forms of these utterances are different, at an abstract level they are similar and it would be useful to leverage this similarity.

We therefore define an abstract representation for utterances and logical forms that is suitable for spatial reasoning. We define seven abstract clusters (see Table 3) that correspond to the main semantic types in our domain. Then, we associate each cluster with a small lexicon that contains language-program token pairs associated with this cluster. These mappings represent the canonical ways in which program constants are expressed in natural language. Table 3 shows the seven clusters we use, with an example for an utterance-program token pair from the cluster, and the number of mappings in each cluster. In total, 25 mappings are used to define abstract representations.

As we show next, abstract examples can be used to improve the process of training semantic parsers. Specifically, in sections 5.1-5.3, we use abstract examples in several ways, from generating new training data to improving search accuracy. The combined effect of these approaches is quite dramatic, as our evaluation demonstrates.

5.1 High Coverage via Abstract Examples

We begin by demonstrating that abstraction leads to rather effective coverage of the types of questions asked in a dataset. Namely, that many questions in the data correspond to a small set of abstract examples. We created abstract representations for all 3,163 utterances in the training examples by mapping utterance tokens to their cluster label, and then counted how many distinct abstract utterances exist. We found that as few as 200 abstract utterances cover roughly half of the training examples in the original training set.

The above suggests that knowing how to answer a small set of abstract questions may already yield a reasonable baseline. To test this baseline, we constructured a "rule-based" parser as follows. We manually annotated 106 abstract utterances with their corresponding abstract program (including alignment between abstract tokens in the utterance and program). For example, Table 1 shows the abstract utterance and program for the utterance *"There are exactly 3 yellow squares touching the wall"*. Note that the utterance *"There are at least 2 blue circles touching the wall"* will be mapped to the same abstract utterance and program.

Given this set of manual annotations, our rule-based semantic parser operates as follows: Given an utterance x, create its abstract representation $\bar{x}$. If it exactly matches one of the manually annotated utterances, map it to its corresponding abstract program $\bar{z}$. Replace the abstract program tokens with real program tokens based on the alignment with the utterance tokens, and obtain a final program z. If $\bar{x}$ does not match return TRUE, the majority label. The rule-based parser will fail for examples not covered by the manual annotation. However, it already provides a reasonable baseline (see Table 4). As shown next, manual annotations can also be used for generating new training data.

5.2 Data Augmentation

While the rule-based semantic parser has high precision and gauges the amount of structural variance in the data, it cannot generalize beyond observed examples. However, we can automatically generate non-abstract utterance-program pairs from the manually annotated abstract pairs and train a semantic parser with strong supervision that can potentially generalize better. E.g., consider the utterance *"There are exactly 3 yellow squares touching the wall"*, whose abstract representation is given in Table 1. It is clear that we can use this abstract pair to generate a program for a new utterance *"There are exactly 3 blue squares touching the wall"*. This program will be identical

Algorithm 1 Decoding with an Abstract Cache

1: **procedure** DECODE(x, y, C, D)
2: // C is a map where the key is an abstract utterance and the value is a pair $(Z, \hat{R})$ of a list of abstract programs Z and their average rewards $\hat{R}$. D is an integer.
3: $\bar{x} \leftarrow$ Abstract utterance of x
4: $\mathcal{A} \leftarrow D$ programs in $C[\bar{x}]$ with top reward values
5: $B_1 \leftarrow$ compute beam of programs of length 1
6: **for** $t = 2 \ldots T$ **do** // Decode with cache
7: $B_t \leftarrow$ construct beam from B_{t-1}
8: $\mathcal{A}_t = \text{truncate}(\mathcal{A}, t)$
9: B_t.add(de-abstract($\mathcal{A}_t$))
10: **for** $z \in B_T$ **do** //Update cache
11: Update rewards in $C[\bar{x}]$ using $(\bar{z}, R(z, y))$
12: return $B_T \cup$ de-abstract($\mathcal{A}$).

to the program of the first utterance, with `IsBlue` replacing `IsYellow`.

More generally, we can sample any abstract example and instantiate the abstract clusters that appear in it by sampling pairs of utterance-program tokens for each abstract cluster. Formally, this is equivalent to a synchronous context-free grammar (Chiang, 2005) that has a rule for generating each manually-annotated abstract utterance-program pair, and rules for synchronously generating utterance and program tokens from the seven clusters.

We generated 6,158 (x, z) examples using this method and trained a standard sequence to sequence parser by maximizing $\log p'_\theta(z|x)$ in the model above. Although these are generated from a small set of 106 abstract utterances, they can be used to learn a model with higher coverage and accuracy compared to the rule-based parser, as our evaluation demonstrates.[3]

The resulting parser can be used as a standalone semantic parser. However, it can also be used as an initialization point for the weakly-supervised semantic parser. As we observe in Sec. 6, this results in further improvement in accuracy.

5.3 Caching Abstract Examples

We now describe a caching mechanism that uses abstract examples to combat search and spuriousness when training from weak supervision. As shown in Sec. 5.1, many utterances are identical at the abstract level. Thus, a natural idea is to keep track at training time of abstract utterance-program pairs that resulted in a correct denotation, and use this information to direct the search procedure.

Concretely, we construct a cache C that maps abstract utterances to all abstract programs that were decoded by the model, and tracks the average reward obtained for those programs. For every utterance x, after obtaining the final beam of programs, we add to the cache all abstract utterance-program pairs $(\bar{x}, \bar{z})$, and update their average reward (Alg. 1, line 10). To construct an abstract example $(\bar{x}, \bar{z})$ from an utterance-program pair (x, z) in the beam, we perform the following procedure. First, we create $\bar{x}$ by replacing utterance tokens with their cluster label, as in the rule-based semantic parser. Then, we go over every program token in z, and replace it with an abstract cluster if the utterance contains a token that is mapped to this program token according to the mappings from Table 3. This also provides an alignment from abstract program tokens to abstract utterance tokens that is necessary when utilizing the cache.

We propose two variants for taking advantage of the cache C. Both are shown in Algorithm 1.
1. Full program retrieval (Alg. 1, line 12): Given utterance x, construct an abstract utterance $\bar{x}$, retrieve the top D abstract programs $\mathcal{A}$ from the cache, compute the de-abstracted programs Z using alignments from program tokens to utterance tokens, and add the D programs to the *final* beam.
***2. Program prefix retrieval (Alg. 1, line 9)*:** Here, we additionally consider prefixes of abstract programs to the beam, to further guide the search process. At each step t, let B_t be the beam of decoded programs at step t. For every abstract program $\bar{z} \in \mathcal{A}$ add the de-abstracted prefix $z_{1:t}$ to B_t and expand B_{t+1} accordingly. This allows the parser to potentially construct new programs that are not in the cache already. This approach combats both spuriousness and the search challenge, because we add promising program prefixes to the beam that might have fallen off of it earlier. Fig. 3 visualizes the caching mechanism.

A high-level overview of our entire approach for utilizing abstract examples at training time for both data augmentation and model training is given in Fig. 4.

6 Experimental Evaluation

Model and Training Parameters The Bi-LSTM state dimension is 30. The decoder has one hidden layer of dimension 50, that takes the

[3] Training a parser directly over the 106 abstract examples results in poor performance due to the small number of examples.

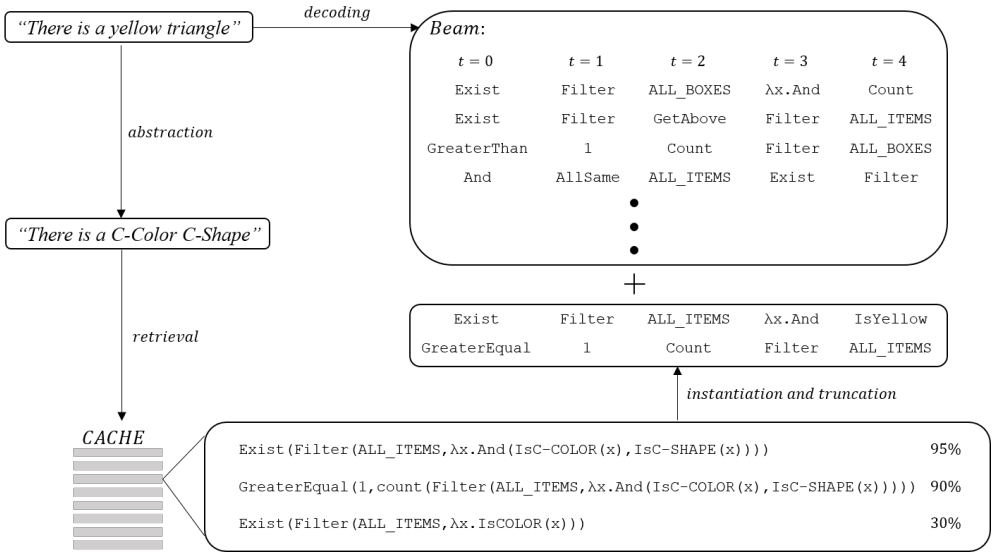


Figure 3: A visualization of the caching mechanism. At each decoding step, prefixes of high-reward abstract programs are added to the beam from the cache.

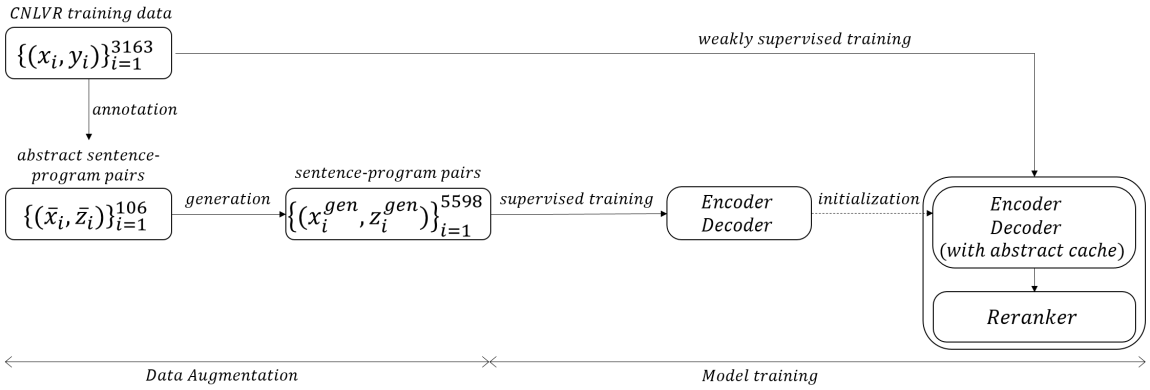


Figure 4: An overview of our approach for utilizing abstract examples for data augmentation and model training.

last 4 decoded tokens as input as well as encoder states. Token embeddings are of dimension 12, beam size is 40 and $D = 10$ programs are used in Algorithm 1. Word embeddings are initialized from CBOW (Mikolov et al., 2013) trained on the training data, and are then optimized end-to-end. In the weakly-supervised parser we encourage exploration with meritocratic gradient updates with $\beta = 0.5$ (Guu et al., 2017). In the weakly-supervised parser we warm-start the parameters with the supervised parser, as mentioned above. For optimization, Adam is used (Kingma and Ba, 2014)), with learning rate of 0.001, and mini-batch size of 8.

Pre-processing Because the number of utterances is relatively small for training a neural model, we take the following steps to reduce sparsity. We lowercase all utterance tokens, and also use their lemmatized form. We also use spelling correction to replace words that contain typos. After pre-processing we replace every word that occurs less than 5 times with an UNK symbol.

Evaluation We evaluate on the public development and test sets of CNLVR as well as on the hidden test set. The standard evaluation metric is accuracy, i.e., how many examples are correctly classified. In addition, we report *consistency*, which is the proportion of utterances for which the decoded program has the correct denotation for all 4 images/KBs. It captures whether a model consistently produces a correct answer.

Baselines We compare our models to the MAJORITY baseline that picks the majority class (TRUE in our case). We also compare to the state-of-the-art model reported by Suhr et al. (2017)

Model	Dev.		Test-P		Test-H	
	Acc.	Con.	Acc.	Con.	Acc.	Con.
MAJORITY	55.3	-	56.2	-	55.4	-
MAXENT	68.0	-	67.7	-	67.8	-
RULE	66.0	29.2	66.3	32.7	-	-
SUP.	67.7	36.7	66.9	38.3	-	-
SUP.+DISC	77.7	52.4	76.6	51.8	-	-
WEAKSUP.	84.3	66.3	81.7	60.1	-	-
W.+DISC	**85.7**	**67.4**	**84.0**	**65.0**	**82.5**	**63.9**

Table 4: Results on the development, public test (Test-P) and hidden test (Test-H) sets. For each model, we report both accuracy and consistency.

Model	Dev.	
	Acc.	Con.
RANDOMER	53.2	7.1
−ABSTRACTION	58.2	17.6
−DATAAUGMENTATION	71.4	41.2
−BEAMCACHE	77.2	56.1
−EVERYSTEPBEAMCACHE	82.3	62.2
ONEEXAMPLEREWARD	58.2	11.2

Table 5: Results of ablations of our main models on the development set. Explanation for the nature of the models is in the body of the paper.

when taking the KB as input, which is a maximum entropy classifier (MAXENT). For our models, we evaluate the following variants of our approach:

- RULE: The rule-based parser from Sec. 5.1.
- SUP.: The supervised semantic parser trained on augmented data as in Sec. 5.2 (5,598 examples for training and 560 for validation).
- WEAKSUP.: Our full weakly-supervised semantic parser that uses abstract examples.
- +DISC: We add a discriminative re-ranker (Sec. 3) for both SUP. and WEAKSUP.

Main results Table 4 describes our main results. Our weakly-supervised semantic parser with re-ranking (W.+DISC) obtains 84.0 accuracy and 65.0 consistency on the public test set and 82.5 accuracy and 63.9 on the hidden one, improving accuracy by 14.7 points compared to state-of-the-art. The accuracy of the rule-based parser (RULE) is less than 2 points below MAXENT, showing that a semantic parsing approach is very suitable for this task. The supervised parser obtains better performance (especially in consistency), and with re-ranking reaches 76.6 accuracy, showing that generalizing from generated examples is better than memorizing manually-defined patterns. Our weakly-supervised parser significantly improves over SUP., reaching an accuracy of 81.7 before re-ranking, and 84.0 after re-ranking (on the public test set). Consistency results show an even crisper trend of improvement across the models.

6.1 Analysis

We analyze our results by running multiple ablations of our best model W.+DISC on the development set.

To examine the overall impact of our procedure, we trained a weakly-supervised parser from scratch without pre-training a supervised parser nor using a cache, which amounts to a re-implementation of the RANDOMER algorithm (Guu et al., 2017). We find that the algorithm is unable to bootstrap in this challenging setup and obtains very low performance. Next, we examined the importance of abstract examples, by pre-training only on examples that were manually annotated (utterances that match the 106 abstract patterns), but with no data augmentation or use of a cache (−ABSTRACTION). This results in performance that is similar to the MAJORITY baseline.

To further examine the importance of abstraction, we decoupled the two contributions, training once with a cache but without data augmentation for pre-training (−DATAAUGMENTATION), and again with pre-training over the augmented data, but without the cache (−BEAMCACHE). We found that the former improves by a few points over the MAXENT baseline, and the latter performs comparably to the supervised parser, that is, we are still unable to improve learning by training from denotations.

Lastly, we use a beam cache without line 9 in Alg. 1 (−EVERYSTEPBEAMCACHE). This already results in good performance, substantially higher than SUP. but is still 3.4 points worse than our best performing model on the development set.

Orthogonally, to analyze the importance of tying the reward of all four examples that share an utterance, we trained a model without this tying, where the reward is 1 iff the denotation is correct (ONEEXAMPLEREWARD). We find that spuriousness becomes a major issue and weakly-supervised learning fails.

Error Analysis We sampled 50 consistent and 50 inconsistent programs from the development set to analyze the weaknesses of our model. By and large, errors correspond to utterances that are more complex syntactically and semantically. In about half of the errors an object was described by two or more modifying clauses: *"there is a box with a yellow circle and three blue items"*; or nesting occurred: *"one of the gray boxes has exactly*

three objects one of which is a circle". In these cases the model either ignored one of the conditions, resulting in a program equivalent to *"there is a box with three blue items"* for the first case, or applied composition operators wrongly, outputting an equivalent to *"one of the gray boxes has exactly three circles"* for the second case. However, in some cases the parser succeeds on such examples and we found that 12% of the sampled utterances that were parsed correctly had a similar complex structure. Other, less frequent reasons for failure were problems with cardinality interpretation, i.e. ,*"there are 2"* parsed as *"exactly 2"* instead of *"at least 2"*; applying conditions to items rather than sets, e.g., *"there are 2 boxes with a triangle closely touching a corner"* parsed as *"there are 2 triangles closely touching a corner"*; and utterances with questionable phrasing, e.g., *"there is a tower that has three the same blocks color"*.

Other insights are that the algorithm tended to give higher probability to the top ranked program when it is correct (average probability 0.18), compared to cases when it is incorrect (average probability 0.08), indicating that probabilities are correlated with confidence. In addition, sentence length is not predictive for whether the model will succeed: average sentence length of an utterance is 10.9 when the model is correct, and 11.1 when it errs.

We also note that the model was successful with sentences that deal with spatial relations, but struggled with sentences that refer to the size of shapes. This is due to the data distribution, which includes many examples of the former case and fewer examples of the latter.

7 Related Work

Training semantic parsers from denotations has been one of the most popular training schemes for scaling semantic parsers since the beginning of the decade. Early work focused on traditional log-linear models (Clarke et al., 2010; Liang et al., 2011; Kwiatkowski et al., 2013), but recently denotations have been used to train neural semantic parsers (Liang et al., 2017; Krishnamurthy et al., 2017; Rabinovich et al., 2017; Cheng et al., 2017).

Visual reasoning has attracted considerable attention, with datasets such as VQA (Antol et al., 2015) and CLEVR (Johnson et al., 2017a). The advantage of CNLVR is that language utterances are both natural and compositional. Treating visual reasoning as an end-to-end semantic parsing problem has been previously done on CLEVR (Hu et al., 2017; Johnson et al., 2017b).

Our method for generating training data resembles data re-combination ideas in Jia and Liang (2016), where examples are generated automatically by replacing entities with their categories.

While spuriousness is central to semantic parsing when denotations are not very informative, there has been relatively little work on explicitly tackling it. Pasupat and Liang (2015) used manual rules to prune unlikely programs on the WIKITABLEQUESTIONS dataset, and then later utilized crowdsourcing (Pasupat and Liang, 2016) to eliminate spurious programs. Guu et al. (2017) proposed RANDOMER, a method for increasing exploration and handling spuriousness by adding randomness to beam search and a proposing a "meritocratic" weighting scheme for gradients. In our work we found that random exploration during beam search did not improve results while meritocratic updates slightly improved performance.

8 Discussion

In this work we presented the first semantic parser for the CNLVR dataset, taking structured representations as input. Our main insight is that in closed, well-typed domains we can generate abstract examples that can help combat the difficulties of training a parser from delayed supervision. First, we use abstract examples to semi-automatically generate utterance-program pairs that help warm-start our parameters, thereby reducing the difficult search challenge of finding correct programs with random parameters. Second, we focus on an abstract representation of examples, which allows us to tackle spuriousness and alleviate search, by sharing information about promising programs between different examples. Our approach dramatically improves performance on CNLVR, establishing a new state-of-the-art.

In this paper, we used a manually-built high-precision lexicon to construct abstract examples. This is suitable for well-typed domains, which are ubiquitous in the virtual assistant use case. In future work we plan to extend this work and automatically learn such a lexicon. This can reduce manual effort and scale to larger domains where there is substantial variability on the language side.

References

D. Andor, C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and M. Collins. 2016. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042* .

S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. 2015. Vqa: Visual question answering. In *International Conference on Computer Vision (ICCV)*. pages 2425–2433.

Y. Artzi and L. Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics (TACL)* 1:49–62.

D. Bahdanau, K. Cho, and Y. Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*.

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

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

J. Cheng, S. Reddy, V. Saraswat, and M. Lapata. 2017. Learning structured natural language representations for semantic parsing. In *Association for Computational Linguistics (ACL)*.

D. Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Association for Computational Linguistics (ACL)*. pages 263–270.

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

K. Guu, P. Pasupat, E. Z. Liu, and P. Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Association for Computational Linguistics (ACL)*.

S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko. 2017. Learning to reason: End-to-end module networks for visual question answering. In *International Conference on Computer Vision (ICCV)*.

R. Jia and P. Liang. 2016. Data recombination for neural semantic parsing. In *Association for Computational Linguistics (ACL)*.

J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick. 2017a. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Computer Vision and Pattern Recognition (CVPR)*.

J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, L. Fei-Fei, C. L. Zitnick, and R. Girshick. 2017b. Inferring and executing programs for visual reasoning. In *International Conference on Computer Vision (ICCV)*.

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

D. Kingma and J. Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

J. Krishnamurthy, P. Dasigi, and M. Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Empirical Methods in Natural Language Processing (EMNLP)*.

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

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

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling data. In *International Conference on Machine Learning (ICML)*. pages 282–289.

C. Liang, J. Berant, Q. Le, K. D. Forbus, and N. Lao. 2017. Neural symbolic machines: Learning semantic parsers on Freebase with weak supervision. In *Association for Computational Linguistics (ACL)*.

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

T. Mikolov, K. Chen, G. Corrado, and Jeffrey. 2013. Efficient estimation of word representations in vector space. *arXiv* .

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

P. Pasupat and P. Liang. 2016. Inferring logical forms from denotations. In *Association for Computational Linguistics (ACL)*.

M. Rabinovich, M. Stern, and D. Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Association for Computational Linguistics (ACL)*.

A. Suhr, M. Lewis, J. Yeh, and Y. Artzi. 2017. A corpus of natural language for visual reasoning. In *Association for Computational Linguistics (ACL)*.

I. Sutskever, O. Vinyals, and Q. V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*. pages 3104–3112.

C. Xiao, M. Dymetman, and C. Gardent. 2016. Sequence-based structured prediction for semantic parsing. In *Association for Computational Linguistics (ACL)*.

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

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

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