Generating Programmatic Referring Expressions via Program Synthesis

Abstract

Incorporating symbolic reasoning into machine learning algorithms is a promising approach to improve performance on tasks that require logical reasoning. We study the problem of generating a programmatic variant of referring expressions: given a symbolic representation of an image and a target object in that image, generate a relational program that uniquely identifies the target object. We propose a program synthesis algorithm that uses reinforcement learning to generate such programs. We demonstrate that our approach significantly outperforms several baselines on challenging benchmarks based on the CLEVR dataset.

1. Introduction

Incorporating symbolic reasoning with deep neural networks (DNNs) is an important challenge in machine learning. Intuitively, DNNs are promising techniques for processing perceptual information; then, symbolic reasoning should be able to operate over the outputs of the DNNs to accomplish more abstract tasks. Recent work has successfully applied this approach to question-answering tasks, showing that leveraging programmatic representations can substantially improve performance—in particular, in visual question answering, by building a programmatic representation of the question and a symbolic representation of the image, we can evaluate the question representation in the context of the image representation to compute the answer (Yi et al., 2018; Mao et al., 2019; Ma et al., 2019).

A natural question is whether incorporating symbolic reasoning with DNNs can be useful for tasks beyond question answering. In particular, consider the problem of generating a *referring expression*—i.e., an image caption that uniquely identifies a given *target object* in a given image (Golland et al., 2010; Kazemzadeh et al., 2014). In contrast to visual question answering, where we translate a question to a program and then execute that program, in this case, we want to *synthesize* a program identifying the target object and then translate this program into a caption.

We take a first step towards realizing this approach. We study the problem of generating a programmatic variant of referring expressions that we call *referring relational programs*. We assume we are given a symbolic representation of an image—such a representation can be easily constructed using state-of-the-art deep learning algorithms (Redmon et al., 2016; Krishna et al., 2017; Yi et al., 2018; Mao et al., 2019)—together with a target object in that image. Then, our goal is to synthesize a relational program that uniquely identifies the target object in terms of its attributes and its relations to other objects in the image. Figure 1 (left) shows an example of an image from the CLEVR dataset, and two referring relational programs for object G in this image. The task for this image is challenging since G has identical attributes as H, which means that the program must use spatial relationships to distinguish them.

Then, we propose an algorithm for synthesizing referring relational programs given the symbolic representation of the image. The objective is to search over the space of possible programs to find one that achieves the given goal—in our case, find a relational program that uniquely identifies the target object when evaluated against the symbolic representation of the image. To account for the combinatorial size of the search space, our algorithm builds on recent techniques including *execution-guided synthesis* (Chen et al., 2018), *hierarchical synthesis* (Nye et al., 2019), and a simple *meta-learning* approach (Si et al., 2018).

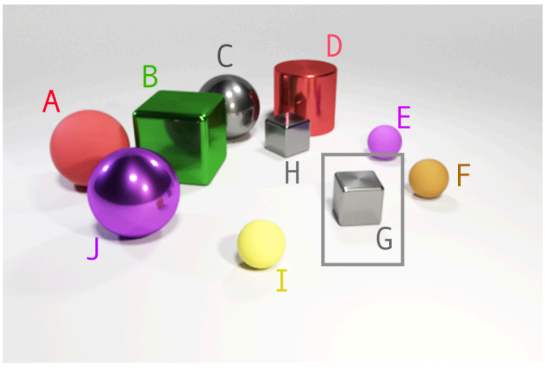
We evaluate our approach on the CLEVR dataset (Johnson et al., 2017), a synthetic dataset of objects with different attributes and spatial relationships. We leverage control over the data generation process to generate problem instances that are particularly challenging—i.e., where there are multiple objects with the same attributes in each scene. By doing so, a valid relational referring program must include complex spatial relationships to successfully identify the target object. Our approach outperforms several baselines.

2. Referring Relational Programs

Scene graph representation of images. We represent images via *scene graphs* $G \in \mathcal{G}$. Vertices in G are objects in the image, and edges encode relations between objects. Unary relations represent attributes such as color and shape, and binary relations capture spatial information between objects—above, left, right, and the like. Abstractly, we think of G as a set of relations over objects, where $\rho_i \in \mathcal{R}$:

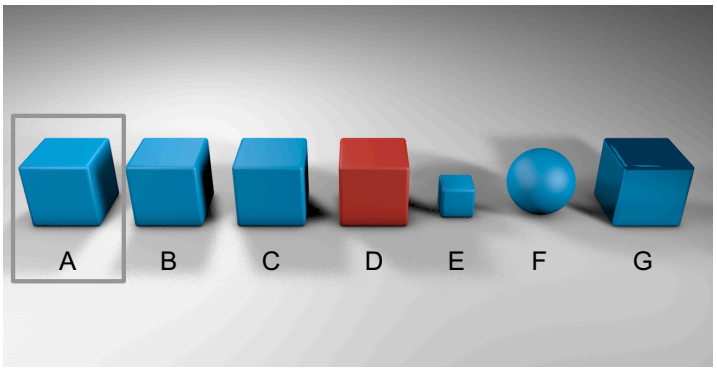
$$G = \left\{\rho_i(o_1^i, \ldots, o_{n_i}^i)\right\}_{i=1}^{n}.$$

Relational programs. Our search space consists of *relational programs*, which we view as sets of relations over



Program 1:
```
color(var0, gray)
/\ front(var0, var1)
/\ color(var1, brown)
```
Output:
{var0 = **G**, var1 = **F**}

Program 2:
```
color(var0, gray)
/\ front(var0, var1)
/\ shape(var1, cube)
```
Output:
{var0 = **G**, var1 = **C**}
{var0 = **G**, var1 = **B**}



Program:
```
color(var0, blue) /\ shape(var0, cube) /\ size(var0, large)
/\ left(var0, var1) /\ left(var1, var2)
/\ left(var0, var2) /\ color(var2, blue) /\ shape(var2, cube)
/\ size(var2, large) /\ material(var2, rubber)
```
Output:
{var0 = **A**, var1 = **B**, var2 = **C**}

Figure 1. Left: An example image from the CLEVR dataset, and two referring relational programs that identify the target object G. The challenge is distinguishing G from the second gray cube H. The first program identifies the target object as the "gray object in front of the brown object", where the brown object is the sphere F. The second program identifies the target object as the "gray object in front of the cube". In this case, the cube can be either B or H, but either of these choices uniquely identifies G. Right: A challenging problem instance that requires several relations to solve (especially when restricted to three free variables—i.e., $|\mathcal{Z}| = 3$). The program shown is generated by our algorithm. It identifies the target object as "the large blue cube to the left of the object to the left of a large blue rubber cube".

variables. More precisely, let $\mathcal{Z}$ be a finite set of variables, with $z_t \in \mathcal{Z}$ a *target variable* representing the object being referred to. A relational program is a set of predicates:

$$P = \{\rho_i(z_1^i, ..., z_{n_i}^i)\}_{i=1}^m.$$

A *valuation* $v \in \mathcal{V}$ is a function $v : \mathcal{Z} \to \mathcal{O}$ that maps each variable to an object in the scene. Given a valuation, we can *ground* the variables in a program using $[\![\cdot]\!]$:

$$[\![P]\!]_v = \left\{\rho_i(v(z_1^i), ..., v(z_{n_i}^i))\right\}_{i=1}^m.$$

That is, $[\![\cdot]\!]$ converts P into a set of predicates over objects. Then we can treat $[\![P]\!]_v : \mathcal{G} \to \mathbb{B}$, where $\mathbb{B} = \{\text{true}, \text{false}\}$, as a Boolean function over scene graphs defined so that

$$[\![P]\!]_v (G) = ([\![P]\!]_v \subseteq G),$$

i.e., $[\![P]\!]_v (G)$ is true if and only if all of the relationships in $[\![P]\!]_v$ are also contained in G.

Definition 2.1 A valuation $v \in \mathcal{V}$ is *valid* for relational program P and scene graph G iff $[\![P]\!]_v (G) = \text{true}$.

We denote the set of all valid valuations for P in G by

$$[\![P]\!]_G = \{v \in \mathcal{V} \mid [\![P]\!]_v (G)\}.$$

Referring relational programs. Our goal is to generate a relational program that satisfies the properties of a referring expression (Golland et al., 2010; Kazemzadeh et al., 2014). Given a scene and an object o_t in that scene, a referring expression is a natural language caption that uniquely identifies o_t. Figure 1 shows an example of an image together with referring relational programs that identify the target object in that image, and Figure 5 shows an example of a scene graph (ignoring the gray variable nodes).

We study a *symbolic variant* of this problem—i.e., (i) we assume the image is given as a scene graph G (e.g., these can be constructed using deep learning (Redmon et al., 2016; Krishna et al., 2017; Yi et al., 2018; Mao et al., 2019)), and (ii) our referring expressions are relational programs that uniquely identify o_t. More precisely, given a scene graph G and an object o_t in G, we want to construct a relational program P such that z_t must refer to o_t in the context of G.

Definition 2.2 Given scene graph G and target object o_t in G, P is a *referring relational program* for o_t in G if (i) $[\![P]\!]_G \neq \varnothing$, and (ii) for all $v \in [\![P]\!]_G$, $v(z_t) = o_t$.

Intuitively, a referring relational program must (i) have at least one interpretation, and (ii) all interpretations must refer to the target object. We assume o_t is encoded in G via a unary *target* relation, and use the predicate $\phi_G(P)$ to indicate P is a referring relational program for o_t in G.

3. Program Synthesis Algorithm

Next, we describe our algorithm that, given a scene graph G, synthesizes a referring relational program P for G. At a

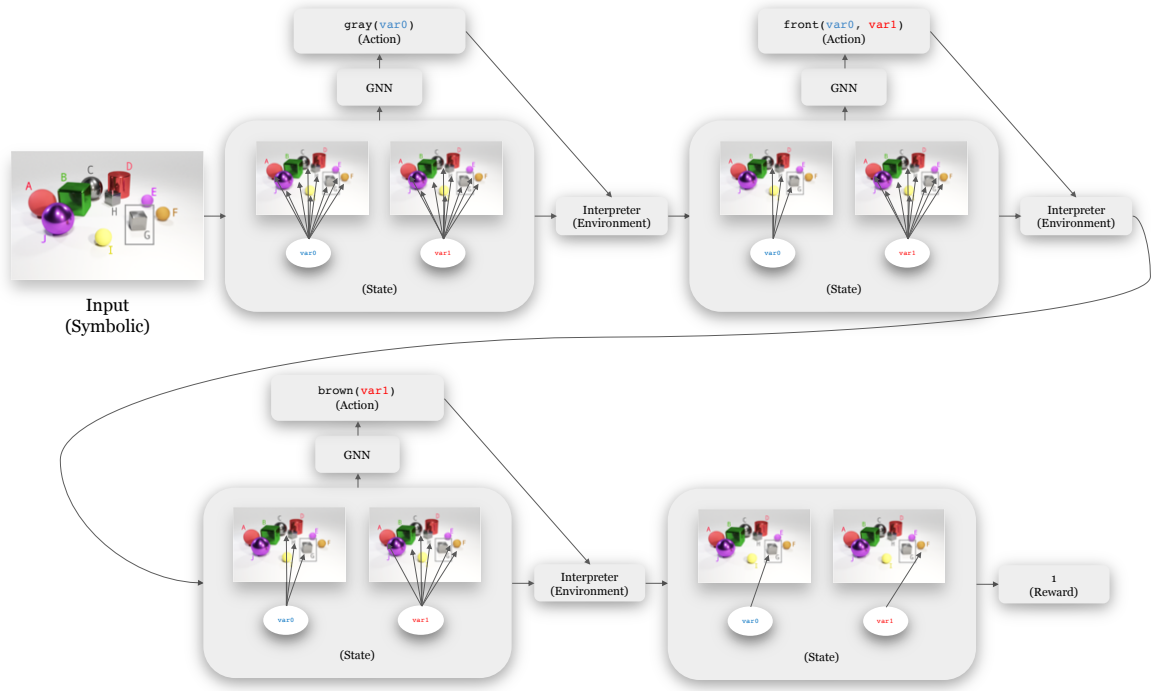


Figure 2. Example rollout according to our MDP. The input is a symbolic representation of the image as a graph. The states encode possible assignments of variables to objects in the scene; these are represented as graphs such as the one shown in Figure 5. The actions are clauses $\rho(z_1, ..., z_n)$; an action is chosen according to the Q-values predicted by the GNN Q-network. The interpreter, which serves as the "environment", removes the variables assignments that are no longer permitted by the newly chosen clause.

high level, we formulate the synthesis problem as a Markov decision process (MDP), visualized in Figure 4. We use reinforcement learning to learn a good policy π for this MDP on a training benchmark. Then, given a new test graph G, we continue to fine-tune π holding G fixed, and return once we find a referring relational program P for G.

Our MDP builds on *execution-guided synthesis* (Chen et al., 2018), where the states are the outputs produced by executing programs P. Intuitively, our goal is to compute a program P such that all consistent valuations uniquely identify the target object—i.e., $v(z_t) = o_t$. Thus, given $G \in \mathcal{G}$ for the current image, we consider the output of P to be the set of valuations $v \in \mathcal{V}$ consistent with G.

In particular, the states $s \in S$ in our MDP are $s = (G, V)$, where G is a scene graph and $V \subseteq \mathcal{V}$ is a subset of valuations. Given a graph G, the initial state is $s_0 = (G, \mathcal{V})$; this choice corresponds to the empty program $P_0 = \text{true}$ (so $[\![P_0]\!]_G = \mathcal{V}$). Next, the actions $a \in A$ in our MDP are $a = (\rho, z_1, ..., z_n) \in \mathcal{R} \times \mathcal{Z}^*$, where ρ is an n-ary relationship. Then, the (deterministic) transitions are

$$(G, V') = T((G, V), (\rho, z_1, ..., z_n))$$
$$V' = \{v \in V \mid [\![\rho(z_1, ..., z_n)]\!]_v(G)\}.$$

That is, V' is the set of all valuations that are consistent with G given the additional predicate $\rho(z_1, ..., z_n)$. Finally, we use a sparse reward function

$$R((G, V)) = \mathbb{1}(\forall v \in V.\ v(z_t) = o_t).$$

That is, $R((G, V)) = 1$ if and only if the program P corresponding to the sequence of actions taken is a referring relational program for G. Thus, a policy that achieves good reward on this MDP should quickly identify a valid referring relational program for a given graph G.

We use the deep Q-learning algorithm with a replay buffer to perform reinforcement learning—surprisingly, we found this approach outperformed policy gradient and actor-critic approaches. We believe it works well since the states in our formulation capture a lot of information about the progress of the policy. Given the deep Q-network $Q_\theta(s, a)$, the corresponding policy π is to use $Q_\theta(s, a)$ with ϵ-greedy exploration—i.e., $\pi(s) = \arg\max_{a \in A} Q_\theta(s, a)$ with probability $1 - \epsilon$, and $\pi(s) \sim \text{Uniform}(A)$ with probability ϵ.

We give details of our algorithm in Appendix A. Our approach can also be extended to handling cases where the scene graph contains uncertain relationships (e.g., to capture cases where the CNN used to construct the scene graph is uncertain about) a prediction; see Appendix B for details.

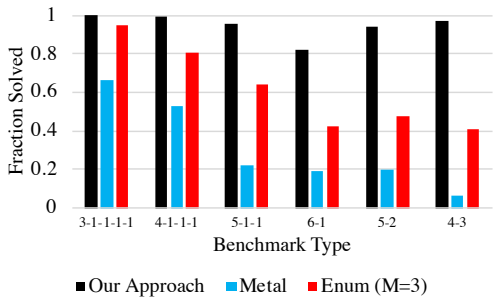


Figure 3. Fraction of problem instances solved in a variety of benchmarks by different algorithms. Comparing our algorithm (black) to baselines neurosymbolic synthesis (blue) and enumerative synthesis with $M = 3$ (red).

4. Experiments

We evaluate our approach on the CLEVR dataset, both (i) on purely synthetic graphs that we generated, and (ii) on graphs constructed using a CNN based on the original CLEVR images. We use synthetic data since it allows us to generate challenging problem instances that require the use of relationships involving multiple objects. We demonstrate that our approach outperforms several baselines on the purely synthetic graphs; we include comparisons to ablations and results on CNN graphs in Appendix C.

4.1. Experimental Setup

Dataset. Our dataset is a set of synthetic scene graphs that include objects and relations between these objects, including unary ones (called *attributes*), namely shape, color, and material, as well as binary ones that encode spatial relations, namely front/behind and left/right.

Using this approach, we can create challenging problem instances (i.e., a scene graph and a target object) to evaluate our algorithm. Our primary goal is to create problem instances where the referring relational program has to include spatial relationships to identify the target object. These instances are challenging since multiple relations are needed to distinguish two identical objects—e.g., in Figure 1 (left), at least two relations are needed to distinguish G from H, and more are needed in Figure 1 (right).

To this end, we create graphs with multiple identical objects in the scene. We classify these datasets by the set of counts of identical objects (in terms of attributes). For instance, the dataset CLEVR-4-3 consists of 7 objects total, the first 4 and last 3 of which have identical attributes—e.g., it might contain 4 gray metal cubes and 3 red metal spheres.

For simplicity, we directly generate scene graphs; thus, they do not contain any uncertainty. We impose constraints on the graphs to ensure they can be rendered into actual CLEVR images if desired. We consider the following datasets: 3-1-1-1-1, 4-1-1-1-1, 5-1-1, 6-1, 5-2, and 4-3. For each dataset, we use 7 total objects. Each dataset has 30 scene graphs for training (a total of 210 problem instances), and 500 scene graphs for testing (a total of 3500 problem instances).

Our algorithm. We search for programs of length at most $M = 8$, using $K = 2$ in hierarchical synthesis. We consider three variables—i.e., $|\mathcal{Z}| = 3$, including z_t. We use $N = 200$ rollouts during reinforcement learning. We pretrain our Q-network on the training set corresponding to each dataset, using $N = 10000$ gradient steps with a batch size of 5.

4.2. Comparison to Baselines

We use each algorithm on our synthetic graphs dataset; in Figure 6 (left), we report what fraction of each kind of problem instance that is solved by each one.

Neurosymbolic synthesis. We compare to a state-of-the-art synthesizer called Metal (Si et al., 2018). This approach uses reinforcement learning to find a program that satisfies the given specification; in addition, they use the same simple meta-learning approach as ours. As can be seen in Figure 6, our approach substantially outperforms this baseline by using hierarchical synthesis and execution-guided synthesis. For instance, on 6-1, our approach solves 82%; in contrast, Metal solves just 19%. Similarly, on 4-3, our approach solves 97% whereas Metal solves just 6%. We believe Metal works poorly in our setting due to the lack of intermediate feedback in our setting.

Enumerative synthesis. We compare to a synthesis algorithm that enumerates programs to find one that solves the task (Alur et al., 2013). This approach does not use machine learning to guide its search, making it challenging to scale to large programs (i.e., large M) due to the combinatorial blowup in the search space; thus, we consider $M = 3$. As can be seen in Figure 6, our approach substantially outperforms enumerative synthesis. For instance, on 6-1, our approach solves 82%, whereas enumerative synthesis solves 42%, and on 4-3, our approach solves 97%, whereas enumerative synthesis solves 41%.

5. Conclusions

We have proposed an approach to solving a symbolic variant of referring expressions using program synthesis. Our work is a first step towards incorporating symbolic reasoning into image captioning tasks. Future work includes leveraging our approach to generate natural language referring expressions for real-world images—i.e., by synthesizing a referring relational program and translating it to natural language. In addition, we have ignored the problem of *naturalness* for the programs we generate (i.e., how easy it is for a human to understand the program), which is important to address.

References

Alur, R., Bodik, R., Juniwal, G., Martin, M. M., Raghothaman, M., Seshia, S. A., Singh, R., Solar-Lezama, A., Torlak, E., and Udupa, A. *Syntax-guided synthesis*. IEEE, 2013.

Chen, X., Liu, C., and Song, D. Execution-guided neural program synthesis. In *ICLR*, 2018.

Golland, D., Liang, P., and Klein, D. A game-theoretic approach to generating spatial descriptions. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pp. 410–419. Association for Computational Linguistics, 2010.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Lawrence Zitnick, C., and Girshick, R. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2901–2910, 2017.

Kazemzadeh, S., Ordonez, V., Matten, M., and Berg, T. Referitgame: Referring to objects in photographs of natural scenes. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 787–798, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.-J., Shamma, D. A., et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 123(1):32–73, 2017.

Ma, K., Francis, J., Lu, Q., Nyberg, E., and Oltramari, A. Towards generalizable neuro-symbolic systems for commonsense question answering. *arXiv preprint arXiv:1910.14087*, 2019.

Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., and Wu, J. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *ICLR*, 2019.

Nye, M., Hewitt, L., Tenenbaum, J., and Solar-Lezama, A. Learning to infer program sketches. In *ICML*, 2019.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

Si, X., Yang, Y., Dai, H., Naik, M., and Song, L. Learning a meta-solver for syntax-guided program synthesis. In *ICLR*, 2018.

Yi, K., Wu, J., Gan, C., Torralba, A., Kohli, P., and Tenenbaum, J. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *Advances in Neural Information Processing Systems*, pp. 1031–1042, 2018.

A. Program Synthesis Algorithm Details

We give details on our program synthesis algorithm; our algorithm is summarized in Algorithm 1.

State encoding. A key challenge is designing a neural network architecture for predicting $Q_\theta(s, a)$. Our approach is based on encoding $s = (G, V)$ as a graph data structure, and then choosing $Q_\theta(s, a)$ to be a graph neural network (GNN). Our graph encoding of (G, V) has three main kinds of vertices: (i) objects o in G, (ii) relationships $\rho \in \mathcal{R}$, and (iii) variables $z \in \mathcal{Z}$, as well as a few auxiliary kinds of vertices to support the graph encoding. In Figure 5, we show an example of a graph encoding of a state in our MDP.

First, each object o is represented by exactly one vertex in the graph; each relationship $\rho \in \mathcal{R}$ is represented by exactly one vertex in the graph; and each variable $z \in \mathcal{Z}$ is represented by exactly one vertex in the graph.

Second, for each relationship $\rho_i(o_{i,1}, ..., o_{i,n_i}) \in G$, we introduce $n + 1$ new vertices $\{(i, \rho_i), (i, 1), ..., (i, n_i)\}$ into the graph, along with the edges

$$(i, \rho_i) \to (i, 1) \to o_1, ..., (i, \rho_i) \to (i, n_i) \to o_{n_i}$$

as well as the edge $\rho_i \to (i, \rho_i)$. This approach serves two purposes. The first purpose is that the intermediate vertex (i, ρ_i) distinguishes different relationships in G with the same type $\rho_i \in \mathcal{R}$. In addition, the edges $\rho_i \to (i, \rho_i)$ connects all relationship of the same type, which allows information to flow between these parts of the graph—for example, these edges could help the GNN count how many occurrences of the relationship "red" are in G. The second purpose is that the intermediate vertices (i, j) preserve information about the ordering of the objects in the relationship—e.g., in front(o, o'), the edge $(i, 0) \to o$ indicates that o is in front, and $(i, 1) \to o'$ indicates that o' is behind.

Third, to encode the valuations $v \in V$, we include the following edges in the graph:

$$\bigcup_{v \in V} \{z \to v(z) \mid z \in \mathcal{Z}\}.$$

Intuitively, these edges capture all possible assignments of objects o to variables z that are allowed by V. For instance, in the initial state $S_0 = (G, \mathcal{V})$, these edges are $z \to o$ for every $z \in \mathcal{Z}$ and o in G. This encoding loses information about V, since an assignment o to z may only be allowed for a subset of $v \in V$. However, V is combinatorial in size, so encoding the entire structure of V yields too large a graph.

To ensure that information can flow both ways, all edges in our encoding described above are bidirectional.

Neural network architecture. As mentioned above, $Q_\theta(s, a)$ is based on a graph convolutional network (GCN) (Kipf & Welling, 2017). We use $\psi(s)$ to denote the graph encoding of $s = (G, V)$ described above, where in addition each node is represented by a fixed embedding vector depending on its node name. The relationship vertices ρ and (i, ρ_i) have an embedding vector x_ρ. The positional vertices (i, j) encoding object ordering use an single embedding $x_{\rho_i, j}$ specific to both the corresponding relationship ρ_i and the object position j within the relationship.

Now, Q_θ applies a sequence of graph convolutions to $\psi(s)$:

$$\psi^{(0)} = \psi(s)$$
$$\psi^{(t+1)} = f_\theta^{(t)}(\psi^{(t)}) \quad (\forall t \in \{0, 1, ..., m-1\}.$$

Each $\psi^{(t)}$ has the same graph structure as $\psi(s)$, but the embedding vectors $x_k^{(t)}$ associated with each vertex k are different (i.e., computed as a function of the embeddings in the previous layer and of the GCN parameters).

Finally, at the output layer, Q_θ decodes the Q-values for each action $\rho(z_1, ..., z_n)$ based on the embedding vectors of the corresponding vertices $\rho, z_1, ..., z_n$:

$$Q_\theta(s, \rho(z_1, ..., z_n)) = g_\theta(x_\rho^{(m)}, x_{z_1}^{(m)}, ..., x_{z_n}^{(m)})$$

The architecture of g_θ can be any aggregation method from vertex level to action level. Two example strategies are LSTM structure and concatenation.

Hierarchical synthesis. We adopt an approach based on *hierarchical synthesis* (Nye et al., 2019). The idea is to combine a neurosymbolic synthesizer with a traditional one based on enumerative search. Intuitively, the neurosymbolic synthesizer can determine the majority of the program, after which the enumerative synthesizer can be used to complete the program into one that satisfies $\phi_G(P)$.

More precisely, in the first phase, we run the neurosymbolic synthesizer for a fixed number N of steps. At each step in this phase, we generate a program P of length M; if we find one that satisfies $\phi_G(P)$, then we return it. Otherwise,

Algorithm 1 Our algorithm for synthesizing referring relational programs. Hyperparameters are $N, M, K \in \mathbb{N}$.

function SynthesizeProgram(G)
  Initialize Q-network Q_θ with pretrained parameters θ^0
  for $i \in \{1, ..., N\}$ **do**
    Sample program P of length M according to Q_θ
    if $\phi_G(P)$ **then**
      return P
    Update Q_θ using deep Q learning
  Get best length $M - K$ program P^0 according to Q_θ
  for Programs P of length K **do**
    if $\phi_G(P^0 \wedge P)$ **then**
      return $P^0 \wedge P$
  return $\varnothing$

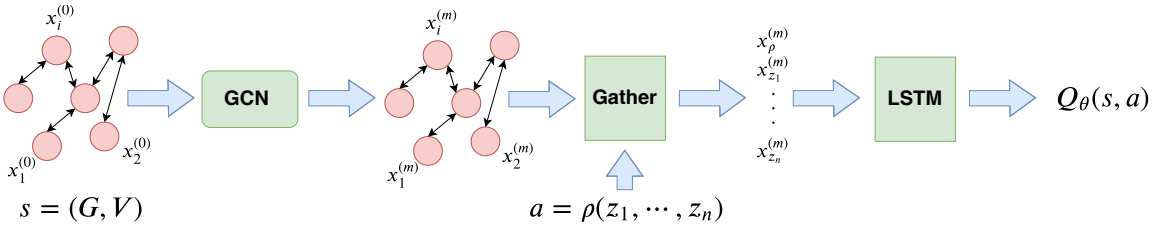


Figure 4. Our Q-network architecture. It takes as input an encoding of the state as a graph, and produces a vector embedding for each node using a GCN. Then, it predicts $Q(s, a)$ based on the vector embeddings for the nodes in the graph relevant to the action a.

we continue to the second phase. In this phase, we begin by constructing the best program P^0 of length $M - K$ according to Q_θ (i.e., use zero exploration $\epsilon = 0$), where $K \in \mathbb{N}$ is a hyperparameter of our algorithm. Then, we perform an exhaustive search over programs P of length K, checking if the combined program $P' = P^0 \wedge P$ satisfies $\phi_G(P')$. If we find such a program, then we return it. Finally, we return $\varnothing$ if we do not find a valid program, indicating failure.

Meta-learning. Finally, the algorithm we have described so far is for synthesizing a single referring relational program from scratch for a given scene graph G. We use a simple meta-learning approach where we pretrain Q_θ on a training benchmark of similar synthesis problems. In particular, we assume given a training set $\mathcal{G}_{\text{train}} \subseteq \mathcal{G}$ of scene graphs; then, we use deep Q-learning to train a neural network Q_{θ^0} that achieves high reward on average for random

$$\theta^0 = \arg\max_\theta \mathop{\mathbb{E}}_{G \sim \text{Uniform}(\mathcal{G}_{\text{train}})} [J(\theta; G)],$$

where $J(\theta; G)$ is the standard Q-learning objective for the MDP constructed for scene graph G.

Overall algorithm. Our overall algorithm is summarized in Algorithm 1. It takes as input a scene graph G, and outputs a relational referring program P (i.e., that satisfies $\phi_G(P)$), or $\varnothing$ if it fails to find such a program. The first step initializes Q_θ with the pretrained parameters θ^0. Then, the first phase uses deep Q-learning to tune θ based on programs P of length M sampled from the MDP for G. If no valid program is found, then it proceeds to the second phase, where it performs an exhaustive enumerative search over programs $P^0 \wedge P$, where P^0 is the optimal program of length $M - K$ according to Q_θ. If again no valid program is found, then it returns $\varnothing$ to indicate failure.

B. Handling Uncertain Relations

Our approach can be extended to handle cases when the relations $\rho(o_1, ..., o_n)$ in the scene graph G are uncertain—e.g., to handle relations where the CNN used to predict the scene graph has low confidence. In particular, relations in G can be certain (guaranteed to be in the graph), uncertain (may or may not be in the graph), or absent (guaranteed to not be in the graph). We represent this decomposition by writing $G = G_+ \sqcup G_?$ as the *disjoint union* of the certain relations G_+ and the uncertain relations $G_?$; absent relations are omitted.

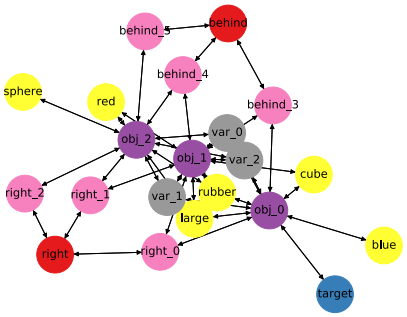


Figure 5. Example of a graph encoding of a state. Variables are shown in gray and objects are shown in purple. Binary relationships are shown in red (for a vertex ρ) and pink (for a vertex (i, ρ_i)). Unary relationships are shown in yellow; these relationships only have a single object, so we do not need a separate vertex for each relationship in (i, ρ_i). The target relationship is shown in blue.

Definition B.1 Given scene graph G and target object o_t in G, P is a *referring relational program* for o_t in G if (i) $[\![P]\!]_{G_+} \neq \varnothing$, and (ii) for all $v \in [\![P]\!]_G$, $v(z_t) = o_t$.

Intuitively, a referring relational program must (i) have at least one certain interpretation, and (ii) all interpretations must refer to the target object, *regardless of the value of uncertain relations*.

Next, we modify our MDP to handle uncertain relationships. In particular, it keeps track of both certain and uncertain relationships—i.e., the initial state is $s_0 = (G, \mathcal{V}, \varnothing)$, and the transitions are

$$(G, V'_+, V'_?) = T((G, V), (\rho, z_1, ..., z_n))$$

where

$$\begin{aligned} V'_+ &= \{v \in V_+ \mid [\![\rho(z_1, ..., z_n)]\!]_v (G_+)\} \\ V'_? &= \{v \in V_? \mid [\![\rho(z_1, ..., z_n)]\!]_v (G)\} \\ &\quad \cup \{v \in V_+ \mid [\![\rho(z_1, ..., z_n)]\!]_v (G_?)\}. \end{aligned}$$

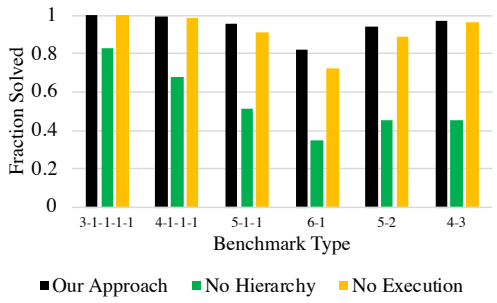


Figure 6. Fraction of problem instances solved in a variety of benchmarks by different algorithms. Comparing our algorithm (black) to ablations without hierarchical synthesis (green) and without execution guidance (yellow).

The rewards are as before—i.e.,

$$R((G, V_+, V_?)) = \begin{cases} 1 & \text{if } \forall v \in V_+ \cup V_?.\ v(z_t) = o_t \\ 0 & \text{otherwise.} \end{cases}$$

Finally, we modify need to modify how our MDP state is encoded by the neural network. In particular, it encodes whether the relationship is certain as an edge type $\rho \xrightarrow{+} (i, \rho_i)$ for certain relationships in G_+ and $\rho \xrightarrow{?} (i, \rho_i)$ for uncertain relationships in $G_?$. Similarly, we use $z \xrightarrow{+} v(z)$ for certain valuations $v \in V_+$ and $z \xrightarrow{?} v(z)$ for uncertain valuations $v \in V_?$.

C. Additional Experiments

We describe two additional experiments: (i) comparisons to ablations of our algorithm, and (ii) experiments on rendered CLEVR images where a CNN is used to predict the scene graph (which may lead to errors). For the latter, we include uncertain relationships in the scene graph G, which are handled by our algorithm as described in Appendix B.

C.1. Comparison to Ablations

We compare to two ablations on the synthetic graph datasets; in Figure 6 (right), we report what fraction of the benchmarks in each category are solved.

Hierarchical synthesis. Next, we compare to an ablation that does not use hierarchical synthesis—i.e., it only count the program generated by the neural symbolic synthesizer, but no enumerative search to correct the generated program. As can be seen, our approach substantially outperforms this ablation—e.g., on 6-1, hierarchical synthesis improves performance from 35% to 82%, and on 4-3, it improves performance from 46% to 97%. Intuitively, hierarchical synthesis improves performance by using reinforcement learning to find the larger but more straightforward parts of the program, whereas the enumerative synthesizer can find the more challenging parts using brute force.

Execution guidance. We compare to an ablation where we do not use the interpreter to guide RL; instead, the states are partial programs P. In particular, this ablation does not have feedback from the interpreter until the sparse reward at the very end of a rollout. As can be seen from Figure 6, using execution guidance improves our performance, especially on harder benchmarks—e.g., for the 6-1 benchmark, it improves performance from 72% to 82%.

C.2. Rendered CLEVR Images

We also evaluate based on a dataset of images from the original CEVR dataset. These images have the same kinds of relations as our generated scene graphs.

We use a convolutional neural network (CNN) to construct the scene graph (Yi et al., 2018). For simplicity, this CNN predicts both object attributes and positions. The object attributes are predicted independently—i.e., it could predict that object J is both red with probability 0.75 and purple with probability 0.75. We consider a relation to be absent if $p = p(\rho(o_1, ..., o_n)) < 1/2$; for relations with $p \geq 1/2$, we consider them to be uncertain if there are multiple such attributes of the type (e.g., object J is predicted to be both red and purple with probability $\geq 1/2$), and certain otherwise. The spatial relationships are inferred based on the object positions; we consider it to be uncertain if the objects are very close together along some dimension.

Out of 6487 tasks, our approach solved all but 25 according to the ground truth relations—i.e., the ground truth in the CLEVR dataset, not the predicted ones seen by our interpreter. Thus, our approach works well even when there is uncertainty in the scene graph predicted using a CNN.