Symbolic Manipulation Planning with Discovered Object and Relational Predicates

Alper Ahmetoglu[1], Erhan Oztop[2,3], Emre Ugur[4]

***Abstract*—Discovering the symbols and rules that can be used in long-horizon planning from a robot's unsupervised exploration of its environment and continuous sensorimotor experience is a challenging task. The previous studies proposed learning symbols from single or paired object interactions and planning with these symbols. In this work, we propose a system that learns rules with discovered object and relational symbols that encode an arbitrary number of objects and the relations between them, converts those rules to Planning Domain Description Language (PDDL), and generates plans that involve affordances of the arbitrary number of objects to achieve tasks. We validated our system with box-shaped objects in different sizes and showed that the system can develop a symbolic knowledge of pick-up, carry, and place operations, taking into account object compounds in different configurations, such as boxes would be carried together with a larger box that they are placed on. We also compared our method with other symbol learning methods and showed that planning with the operators defined over relational symbols gives better planning performance compared to the baselines.**

***Index Terms*—Developmental Robotics, Learning Categories and Concepts, Deep Learning Methods**

I. Introduction

THE long-standing challenge for artificial intelligence (AI) research is to build a generalist agent that can parse and understand its environment through its sensors, carry out desired tasks by acting on the environment, and update its knowledge when necessary to adapt to new situations. The difficulty in achieving this stems from the fact that we do not have a well-defined, robust, and generic representation of the environment that applies to a sufficiently large class of tasks. The requirements and the resolution of the representation change drastically when the aimed task properties and/or the capabilities of the agent change. For example, a robotic agent tidying up the table needs a different representation of the environment than a robot tying a rope. It is not even clear whether soft and rigid objects should be represented within a single representational framework. Nevertheless, once a symbolic representation of the environment that is appropriate for the task at hand can be obtained, many AI search techniques become available for efficiently finding a solution to problems such as planning to achieve a goal [1].

An effective approach to build a set of symbols for obtaining the leverage of off-the-shelf symbolic systems is to focus on the preconditions and effects related to the actions of an agent [2], [3], [4]. Learning precondition-effect relations effectively models the environment based on the agent's capabilities. This is preferable for filtering out irrelevant information that would otherwise increase the complexity of the problem. Once the necessary symbols are learned, the environment description can be translated, for example, into Planning Domain Definition Language (PDDL) [5] that allows the use of fast domain-independent planners such Fast Downward [6] or fast-forward [7]. These methods partition the dataset into subgoal options, then learn a symbol pair for the precondition-effect tuple. Alternatively, one can first compress the experience of the agent, i.e., the sensory state-space explored, into a symbolic space using deep neural networks, then encode symbolic transitions either with PDDL or learn a separate module for generating the necessary action from the current state for a given goal state [8], [9], [10]. One advantage of this approach is that it does not require a separate partitioning phase, which can perform poorly in high-dimensional observations such as images.

Our previous work, DeepSym [11], stands in between these two approaches by having a differentiable, deep architecture that can learn symbolic representations for the preconditions of the executed action. DeepSym is composed of an encoder-decoder network with binary bottleneck layers for learning object symbols to predict the effect of the executed action. In a follow-up work [12], the architecture is improved by employing a transformer layer in the decoder, allowing symbols to interact to model multi-object effects. However, the interaction of symbols remains implicit in the weights of the transformer and cannot be translated into PDDL in a straightforward way, thus limiting the domain-independent planning capability. In other words, even though the architecture models relations between objects, these relations cannot be expressed as relational symbols between objects. Our recent work [13] proposes a solution for this problem with an architecture that can learn not only unary object symbols but explicitly encodes relations between objects using binary attention weights. However, symbolic-level transitions from the learned symbols that enable domain-independent planning were not considered.

Manuscript received: July 3, 2024; Revised: October 16, 2024; Accepted: December 20, 2024.

This paper was recommended for publication by Editor Tetsuya Ogata upon evaluation of the Associate Editor and Reviewers' comments. This research was supported by TUBITAK (The Scientific and Technological Research Council of Turkey) ARDEB 1001 program (project number: 120E274). Additional support was given by the Grant-in-Aid for Scientific Research (project no JP23K24926), the project JPNP16007 commissioned by the New Energy and Industrial Technology Development Organization (NEDO), JST, CREST (JPMJCR17A4), and by INVERSE project (no. 101136067) funded by the European Union.

[1]Department of Computer Science, Brown University
Correspondance aahmetog@cs.brown.edu
[2]Department of Computer Science, Ozyegin University
[3]OTRI, SISReC, Osaka University
[4]Department of Computer Engineering, Bogazici University
Digital Object Identifier (DOI): see top of this page.

In the current study, we first learn a set of unary and relational symbols between objects using the Relational DeepSym architecture [13], then we propose a method to build abstract operators defined over these unary and relational symbols that describe the symbolic transition of a state when an action is executed. We translate these operators into PDDL descriptions and show that they can be used for planning with off-the-shelf AI planners in a tabletop object stacking task. We compare our method with [11] and [12] in terms of effect prediction accuracy and planning performance. Our results show that planning with the operators defined over relational symbols performs better than the baselines.

II. Related Work

There is a large body of literature where the learned affordances or effect predictors have been used to make plans in the continuous or sub-symbolic space of the robots [14], [15]. Our work is mainly related to methods that study symbol emergence [16] and that discover symbolic representations of the environment from the continuous experience of the robot through its sensors with the aim of planning with off-the-shelf AI planners.

[2], [3] proved that it is necessary and sufficient to learn symbols for the precondition and the effect set of the agent's action repertoire to enable planning. Based on this observation, they learn a set of symbols that cover actions' pre- and post-conditions and use them to build a PDDL description of the environment. These works use a fixed-sized vector to represent the state of each object with the implicit assumption that the number of objects will be the same across different environment instantiations, restricting the portability of previously learned symbols to new environment settings. In follow-up work, [17] builds upon this framework by using an agent-centric state representation, allowing agent-centric symbols to be shared through different tasks. [4], [18] considers object-centric precondition and effect set to find discrete symbols for producing PDDL descriptions. Similarly, [19] used object-centric representations, which increases the generalization of symbols through objects with the same properties. These works cluster the collected transitions based on actions and effect sets prior to symbol learning procedure to learn a compact symbol for each precondition-effect set. As such, the quality of the learned symbols relies on a successful partitioning of the state-space, which can be non-trivial for high-dimensional spaces. Our work differs from these in that we directly learn symbols—without any clustering—using deep neural networks with binarized bottleneck layers by minimizing the effect prediction error.

In another line work, [20], [21], [22] proposed a bi-level planning schema in which a set of operators and corresponding samplers are learned from previously acquired symbols that allow the agent to make refined plans that can consider the geometric information. [23] learns predicates from demonstrations with a surrogate objective for planning. In a follow-up work, [24] considers a subset of abstract effects to reduce the complexity of the learned operators. These studies follow the task and motion planning (TAMP) formulation [25] in which the problem is not only finding the sequence of tasks but also finding the correct motion parameters. However, these TAMP formulations learn state abstractions on top of high-level predicates such as on(?x, ?y). As our work learns symbols with an encoder-decoder network to minimize effect prediction error, the input modality can be continuous and high-dimensional, as also shown in [11], [12]. Following the TAMP formulation, [26] learns relational state and action abstractions based on critical regions [27], [28], which can be thought as the bottleneck states in the transition history.

[8], [9], [10] use a state autoencoder to compress the state space into a low-dimensional binary vector and an action autoencoder to learn action representations from low-level state transitions. Similarly, [11] uses an encoder-decoder style architecture to generate symbolic representations in the bottleneck layer while minimizing the effect prediction error. In follow-up work, [12] employed transformer layers [29], enabling the flow of information between object symbols to model multi-object effects. Recently, [13] proposed a relational formulation of the encoder-decoder architecture that explicitly models relations between objects using binary attention weights, learning unary and relational symbols simultaneously. Using the learned symbols, the multi-step effect prediction performance is compared with previous works, which can be thought of as a surrogate measure for planning performance. Even though the multi-step effect prediction with the learned symbols enables planning with tree search algorithms in the subsymbolic space, it requires a forward iteration of encoder networks for each node expansion, which can be a computationally heavy process with deep networks. Even though [11] addresses this issue by extracting rules from a decision tree trained with the learned symbols, the input to the tree is a canonical, fixed-size symbolic vector where each dimension denotes the same entity across different samples, which is not the case when there is a varying number objects with no canonical order. In this work, we show how to construct a symbolic transition model in PDDL from a varying number of object symbols and relations, allowing symbolic planning, using the object symbols and relations discovered by the model.

III. Method

A. Problem Definition

This paper deals with the problem of learning (1) a symbolic representation of an environment defined over a finite set of objects, which is sensed as a continuous sensory state, and (2) a set of operators that represent the dynamics of sensory transition dynamics at the symbolic level, which allows domain-independent planning with the learned symbols by the use of off-the-shelf AI planners.

An environment is characterized by a tuple $(\mathcal{X}, \mathcal{A}, P)$ where $\mathcal{X}$ denotes the continuous sensory state of the environment (which is called state-space from now on), $\mathcal{A}$ is a finite set of actions that the agent can execute, and $P(\mathbf{X}' \mid \mathbf{X}, \mathbf{a})$ is the probability that taking action $\mathbf{a} \in \mathcal{A}$ at state $\mathbf{X}$ results in state $\mathbf{X}'$. An environment instance consists of a set of objects $\{o_1, \ldots, o_n\} \in \mathcal{O}$ each having a d_o-dimensional

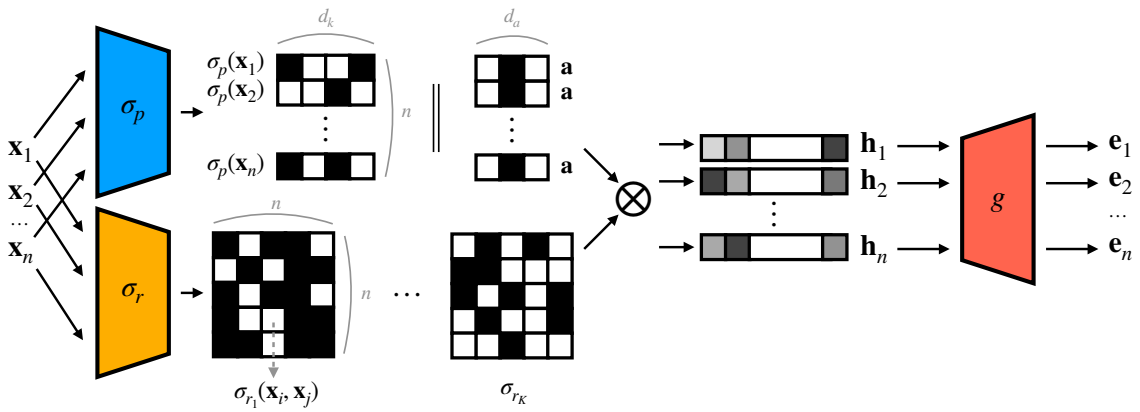

Fig. 1. An overview of the symbol learning architecture. Two encoder networks, σ_p and σ_r, output unary and relational symbols, respectively, by processing each object's input features $\mathbf{x}_i$ in parallel. Both networks are multi-layer perceptrons but one can use other specialized building blocks, such as convolutions to process images. Different from σ_p, there is a self-attention operation at the end of σ_r, which provides an object-object relation. Outputs of these networks are discretized with functions that allow continuous differentiation. Aggregating their symbolic outputs yields a fixed-size, differentiable representation for each object. This representation enables learning unary and relational symbols through tasks like predicting the effect of actions.

continuous-valued feature vector $\mathbf{x}_i \in \mathbb{R}^{d_o}$ defining the state of the environment as an unordered set of feature vectors $\mathbf{X} := \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathcal{X}$. $\mathbf{X}$ is not a fixed-size vector but a variable size depending on the number of objects n in the environment. An action $\mathbf{a}$ is a d_a-dimensional vector representing a high-level parameterized movement primitive that the agent can execute, such as picking an object. This formulation is similar to object-oriented Markov decision processes [30] except, here, we are not interested in maximizing a reward function. Instead, given an initial state $\mathbf{X}_0$ and a goal state $\mathbf{X}_g$, the objective is to find a sequence of actions $(\mathbf{a}_1, \ldots, \mathbf{a}_k)$ that maximizes the probability of reaching the goal state.

We are interested in learning the mapping $f : \mathcal{X} \to \mathcal{P}$ that transforms a state vector $\mathbf{X}$ into a set of predicates (or symbols) $\Sigma := \{\sigma_1(\mathbf{X}), \ldots, \sigma_m(\mathbf{X})\} \in \mathcal{P}$, where $\sigma_i : \mathcal{X} \to \{0,1\}^{d_k}$ is a binary function where d_k is an environment dependent fixed dimension. After symbols are learned, we can find a set of operators (i.e., lifted actions in the symbolic space) $\{\phi_1, \ldots, \phi_k\} \in \Phi$ in which each operator $\phi_i : \Sigma \to \Sigma$ transforms the current symbols into a new set of symbols. Once symbols and operators are learned, we can transform the initial state $\mathbf{X}_0$ and the goal state $\mathbf{X}_g$ into symbolic representations Σ_0 and Σ_g, respectively, and then find a sequence of operators $(\phi_1, \ldots, \phi_k)$ that transforms Σ_0 into Σ_g, and then execute the corresponding sequence of actions $(\mathbf{a}_1, \ldots, \mathbf{a}_k)$ to reach the goal state.

Note that the proposed operator learning system is built on top of our symbol learning network architecture [13]. Thus, to assess the added value of learning explicit symbolic transitions, we compare the performance of the proposed model in effect prediction with DeepSym [11] and Attentive DeepSym [12], and for the planning performance, we only compare with [12] as there is no straightforward way to generate a set of rules with DeepSym when a varying number of objects is affected by the action.

B. Assumptions

We used object pose and type information as input to the network. One can use raw pixel information as input to the network as in [11], [12], which would require convolutional layers, and use slot-attention-based networks to automatically detect objects in the scene [31]. The second assumption is that the robot has a set of high-level actions such as picking and placing. Lastly, we assume that these actions change only a few objects in the environment. This assumption possibly limits the application of actions that change lots of objects, or objects that are in a complex configuration. We discuss and give possible solutions to this limitation in the discussion section.

C. Learning Unary and Relational Symbols

Figure 1 shows an outline of the method. The architecture is composed of four main blocks.

1) Encoder Network: $\sigma_p : \mathcal{X} \to \mathcal{P}$ is a multi-layer perceptron (MLP) with Gumbel-Sigmoid (GS) [32] activation that outputs a binary number for each object in the environment. We treat this as a unary predicate $\sigma_p(\mathbf{x}_i)$ that encodes the property of an object. The number of properties that can be encoded is bounded by the output dimensionality of the MLP—at most 2^{d_k} properties can be encoded with d_k-dimensional outputs.

2) Self-Attention Network: $\sigma_r : \mathcal{X} \to \mathcal{P}$ is a multi-layer perceptron combined with a modified version [13] of the original self-attention layer [29]. Given a state vector $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, the MLP part outputs a set of d-dimensional vectors $\mathbf{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$ and $\mathbf{K} = \{\mathbf{k}_1, \ldots, \mathbf{k}_n\}$ where $\mathbf{q}_i$ and $\mathbf{k}_i$ are the query and key vectors for object o_i, respectively. Unlike the original self-attention layer, we do not define value vectors as we are interested in the attention values. The second important difference is the computation of the attention values:

$$\sigma_r(\mathbf{x}_i, \mathbf{x}_j) = \text{GumbelSigmoid}(\mathbf{q}_i \cdot \mathbf{k}_j) \tag{1}$$

Firstly, using the sigmoid function instead of softmax allows attention values to focus on multiple tokens independently. Secondly, the binarization (due to GS) of attention values allows us to treat them as binary relations between objects while preserving differentiability.

This article has been accepted for publication in IEEE Robotics and Automation Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LRA.2025.3527338

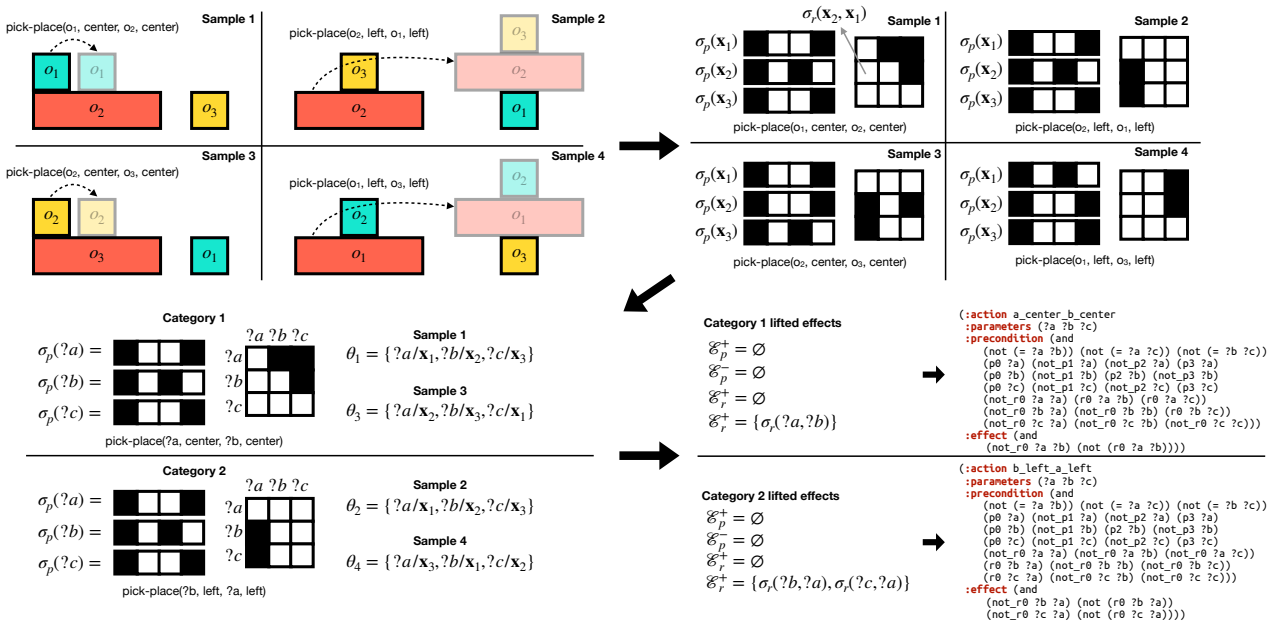

Fig. 2. Top left—Four example transitions are shown in which an object is picked and placed on top of another. In these examples, samples 1 and 3 (and samples 2 and 4) are the same except their object order, and ideally, should be treated as the same. Top right—After training encoder networks σ_p and σ_r using Equation 6, samples represented continuously are converted to unary and relational symbols using the encoders, resulting in a dataset of discrete transition tuples. Bottom left—Samples that can be equated to each other using a substitution are grouped into a category that is order-invariant, ready to be converted into logical rules. Bottom right—Lifted effects are computed for each category and translated into PDDL action schemas.

We define the output of the whole block as a relational predicate $\sigma_r(\mathbf{x}_i, \mathbf{x}_j)$ that encodes the relation between objects o_i and o_j. Relations are directional as $\sigma_r(\mathbf{x}_i, \mathbf{x}_j)$ and $\sigma_r(\mathbf{x}_j, \mathbf{x}_i)$ can have different values due to different query ($\mathbf{q}_i$) and key ($\mathbf{k}_i$) vectors. Note that, without loss of generality, we described the operation with a single attention head; however, in the implementation, we used three attention heads: σ_{r_1}, σ_{r_2}, σ_{r_3}. Through the rest of the text, we denote the number of attention heads—the number of relations—as K.

3) Aggregation Function: fuses unary predicates $\sigma_p(\mathbf{x}_i)$, relational predicates $\sigma_{r_k}(\mathbf{x}_i, \mathbf{x}_j)$, and the action vector $\mathbf{a}$ in a single representation $\mathbf{h}_i$ for each object that is fed into the decoder network for predicting the effect of the executed action. The aggregation function is defined as follows:

$$\mathbf{z}_i = \text{MLP}([\sigma_p(\mathbf{x}_i); \mathbf{a}]) \qquad \forall i \in \{1, \dots, n\} \tag{2}$$

$$\mathbf{h}_i^k = \sum_{j=1}^{n} \sigma_{r_k}(\mathbf{x}_i, \mathbf{x}_j)\mathbf{z}_i \qquad \forall i \in \{1, \dots, n\} \tag{3}$$

$$\forall k \in \{1, \dots, K\} \tag{4}$$

$$\mathbf{h}_i = [\mathbf{h}_i^1; \dots; \mathbf{h}_i^K] \tag{5}$$

where K is the number of relation types. The action vector $\mathbf{a}$ is concatenated with the unary predicate $\sigma_p(\mathbf{x}_i)$, and then fed into an MLP to obtain a representation $\mathbf{z}_i$ that holds action information. Then, for each relation k and object i, intermediate representations $\mathbf{z}_j$ are summed up for indices that satisfy the relation $\sigma_{r_k}(\mathbf{x}_i, \mathbf{x}_j)$, resulting in a fixed-size vector $\mathbf{h}_i^k$ containing information regarding objects that have a relation r_k with object o_i. Lastly, $\{h_i^1, \dots, h_i^K\}$ are concatenated to obtain the aggregated representation $\mathbf{h}_i$ that holds any necessary information about the object o_i, the action $\mathbf{a}$, and the relations between o_i and other objects to predict the effect e_i resulted from the action on object o_i. Equation 3 essentially enables message passing between different object symbols based on the learned relations between objects.

4) Decoder Network: g is an MLP that takes the aggregated representation $\mathbf{h}_i$ as input and outputs the effect $\hat{\mathbf{e}}_i$ of action $\mathbf{a}$ on the object o_i. The predicted effect is used to compute the mean squared error that is backpropagated through the whole network:

$$\mathcal{L} = \sum_{i=1}^{n} \|\hat{\mathbf{e}}_i - \mathbf{e}_i\|^2 \tag{6}$$

where n is the number of objects and the effect vector $\mathbf{e}_i$ is defined as the difference between the current state $\mathbf{x}_i$ and the next state $\mathbf{x}'_i$.

These four blocks create a single differentiable module that can learn unary and relational predicate symbols over object features to minimize the effect prediction error in an end-to-end fashion. To train the network, we execute random actions in the environment and collect a dataset of $(\mathbf{X}, \mathbf{a}, \mathbf{X}')$ tuples. We assume that the number of objects in a single transition tuple $(\mathbf{X}^{(i)}, \mathbf{a}^{(i)}, \mathbf{X}'^{(i)})$ is fixed, but different transitions might contain different number of objects. Then, we train the network to minimize the effect prediction error $\mathcal{L}$ defined in Equation 6.

D. Learning Operators

In this section, we describe how to learn operators that can be used for planning. Throughout this section, we will denote a ground symbol as $\sigma_p(\mathbf{x}_i)$, a lifted symbol as $\sigma_p(?x)$, and a substitution as $\theta = \{?x/\mathbf{x}_i\}$ where $?x$ indicates a free variable that is not bound to any object. The overall procedure is depicted in Figure 2.

This article has been accepted for publication in IEEE Robotics and Automation Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LRA.2025.3527338

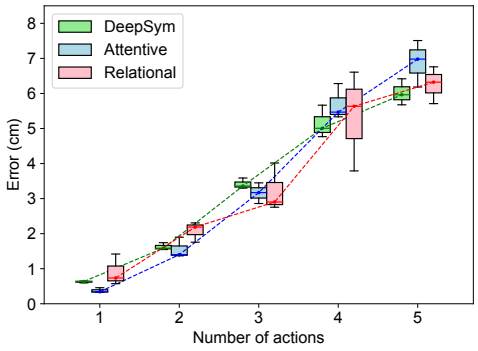

Fig. 3. Effect prediction errors for different numbers of action sequence.

After we train the network, we can transform the dataset $\{(\mathbf{X}^{(i)}, \mathbf{a}^{(i)}, \mathbf{X}'^{(i)})\}_{i=1}^{N}$ into a dataset of propositional symbols $\{(\Sigma_p^{(i)}, \Sigma_r^{(i)}, \mathbf{a}^{(i)}, \Sigma_p'^{(i)}, \Sigma_r'^{(i)})\}_{i=1}^{N}$ where

$$\Sigma_p^{(i)} = \{\sigma_p(\mathbf{x}_1^{(i)}), \dots, \sigma_p(\mathbf{x}_n^{(i)})\}$$
$$\Sigma_r^{(i)} = \{\sigma_r(\mathbf{x}_1, \mathbf{x}_1), \dots, \sigma_r(\mathbf{x}_i, \mathbf{x}_j), \dots, \sigma_r(\mathbf{x}_n, \mathbf{x}_n)\}$$

and $\Sigma_p'^{(i)}$, $\Sigma_r'^{(i)}$ are defined similarly. For notational simplicity, we consider only a single relation type σ_r while the same procedure can be applied to multiple relation types.

Our main goal is to find a set of operators $\Phi = \{\phi_1, \dots, \phi_m\}$ parameterized by lifted actions α that are in the following form:

$$\phi_i(\Sigma_p, \Sigma_r; \alpha_i) = (\Sigma_p', \Sigma_r') \tag{7}$$

modeling the symbolic transition between states. We start by partitioning samples by their actions $\mathbf{a}^{(i)}$ and preconditions $\Sigma_p^{(i)}$, $\Sigma_r^{(i)}$: samples are grouped if their lifted actions and preconditions can be represented by the same substitution θ. For example, consider the following samples:

$$\Sigma_p^{(1)} = \{\sigma_p(\mathbf{x}_1) = 0, \sigma_p(\mathbf{x}_2) = 0, \sigma_p(\mathbf{x}_3) = 1\}$$
$$\mathbf{a}^{(1)} = \text{pick-place}(\mathbf{x}_3, \mathbf{x}_1)$$
$$\Sigma_p^{(2)} = \{\sigma_p(\mathbf{x}_1) = 0, \sigma_p(\mathbf{x}_2) = 1, \sigma_p(\mathbf{x}_3) = 0\}$$
$$\mathbf{a}^{(1)} = \text{pick-place}(\mathbf{x}_2, \mathbf{x}_3)$$

These samples can be grouped into the same category C_1 with substitutions $\theta_1 = \{?a/\mathbf{x}_3, ?b/\mathbf{x}_1, ?c/\mathbf{x}_2\}$ and $\theta_2 = \{?a/\mathbf{x}_2, ?b/\mathbf{x}_3, ?c/\mathbf{x}_1\}$. This procedure gives us a set of groups $\{C_1, \dots, C_k\}$ where each group is defined by lifted preconditions and actions. Next, we compute the lifted effects for each group:

$$\mathcal{E}_p^+ = \{\sigma \mid \sigma \in \Sigma_p'^{(i)}, \sigma \notin \Sigma_p^{(i)}\}$$
$$\mathcal{E}_p^- = \{\sigma \mid \sigma \in \Sigma_p^{(i)}, \sigma \notin \Sigma_p'^{(i)}\}$$

Relational effects $\mathcal{E}_r^+$ and $\mathcal{E}_r^-$ are computed similarly. If lifted effects are not the same for all samples in a group (i.e., a stochastic environment setting), we select the most frequent lifted effect for each group. This completes our operator definition:

$$\phi_i(\Sigma_p, \Sigma_r; \alpha_i) = (\Sigma_p', \Sigma_r')$$
$$\Sigma_p' = \Sigma_p \cup \mathcal{E}_p^+ \setminus \mathcal{E}_p^-$$
$$\Sigma_r' = \Sigma_r \cup \mathcal{E}_r^+ \setminus \mathcal{E}_r^-$$

However, with this strategy, the number of groups increases with the number of objects. On the other hand, most of the time, only a subset of precondition symbols are relevant for a given action. For instance, if the action is to pick and place an object on top of another, then precondition symbols of other objects are irrelevant. Therefore, we only consider a subset of precondition symbols relevant to the action. Although determining which symbols are relevant is difficult to answer in a general sense, a practical and generally valid heuristic is to consider topological neighborhood or contact relations. In our experiments, we define this relevance as objects that are in the action arguments and objects that are in contact with these argument objects. For example, if the action is pick-place($\mathbf{x}_1$, $\mathbf{x}_3$), we consider object and relational symbols that correspond to o_1 and o_3, and those that are in contact with o_1 and o_3. A broader topological relevance alternative can also be to consider objects in the vicinity of action arguments.

E. Translating Operators to PDDL

Each operator ϕ_i is translated into a PDDL action schema where Σ_p and Σ_r are used as preconditions, $\mathcal{E}_p^+$ and $\mathcal{E}_r^+$ are used as effects, free variables that appear in the precondition and/or action are used as parameters, and the action name is defined by the action arguments. In the action schema, each $\sigma_{p_i}(?o)$ appear as `(pi ?o)` or `(not_pi ?o)`, depending on the value of the predicate. We filter out action schemas that are used less than a threshold, which we set to 50 in our experiments. An example action schema is shown in Figure 2, bottom-right. We observed that the most frequently used action schemas are empty actions, such as picking a short cube from the left, which results in no effect. This is due to the exploration process where we execute random actions in the environment that frequently result in no effect. Even though these definitions would not help in the planning process, we choose to keep them as they can be used in later exploration stages to avoid actions that do not have any effect and, thus, are not interesting for the agent.

IV. Experiments

A. Experiment Setup

We conducted our experiments in a tabletop object stacking environment (see top left in Figure 1). There are two object types: 5x5x5cm sized short block and 5x25x5cm sized long block. An environment instance contains two to four objects represented by pose and type. A UR10 robot arm has a four-dimensional action with discrete parameterizations, $\mathbf{a} = (o_i, \delta_i, o_j, \delta_j)$: picking an object o_i from the left, right, or center of the object (δ_i) and releasing it on top, left, or right of (δ_j) another object o_j.

We followed the data collection procedure in [13], where the robot executes random actions in the environment. The difference is that we only record objects that are either action arguments or in contact with them. These object features $\{\mathbf{x}_i\}_{i=1}^{K}$ are used as the state vector. The effect vector $\mathbf{e}_i$ for object o_i is the difference between the next state $\mathbf{x}_i'$ and the current state $\mathbf{x}_i$. We subtract the lateral movement of the arm from the effect vector to remove the effect of the carry

This article has been accepted for publication in IEEE Robotics and Automation Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LRA.2025.3527338

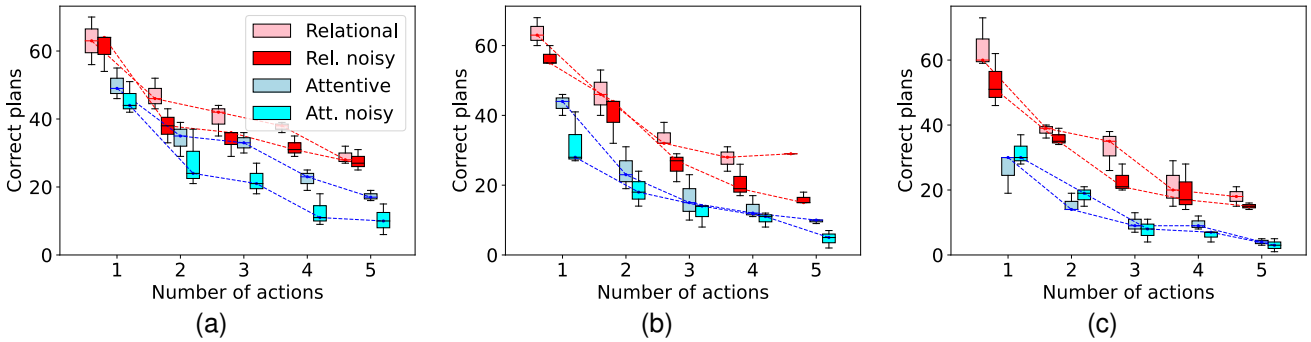

Fig. 4. The planning performance for different numbers of objects over three runs with 100 random problem pairs. (a) 2 objects, (b) 3 objects, (c) 4 objects.

Fig. 5. Given an initial environment configuration in the first column, our model can generate an action sequence reaching the goal state.

action. Otherwise, symbols need to encode global position information of objects, which is not necessary for planning since actions are already parameterized over objects. Note that state vectors can also be selected as raw pixels as in [11], [12]. We collect 200K samples and split them into 160K training, 20K validation, and 20K test samples.

TABLE I
EFFECT PREDICTION RESULTS AVERAGED OVER THREE RUNS. UNITS ARE IN CENTIMETERS.

DeepSym	Attentive	Relational
4.79 ± 0.12	4.47 ± 0.10	**3.21 ± 0.30**

We compare our method with [11] and [12] in terms of effect prediction accuracy, and with [12] for planning performance. The original DeepSym formulation uses a fixed-sized vector to represent the environment's state, making it hard for us to compare the performance of the two methods. However, the comparison with [12] already serves as a valuable measure to assess whether explicitly learning relational symbols helps the planning performance.

We train all methods for 4000 epochs with a batch size of 128 and a learning rate of 0.0001 using Adam optimizer [33]. MLP blocks consist of two layers with 128 hidden units. In Relational DeepSym (ours), we set the number of relation types K to three and the object symbol dimension d to one (i.e., the encoder's output dimensionality). However, for other baselines, we set the object symbol dimension to four since these methods need more representational capacity to encode the state of the environment without relational symbols. Also, the number of attention heads for the attentive formulation [12] is set to four. Following [34], for layers before GS activation, we normalize both the input and the weight vectors to have a norm of three. This prevents the vanishing gradients in the GS function. Lastly, we clip gradients by their norm to 10.

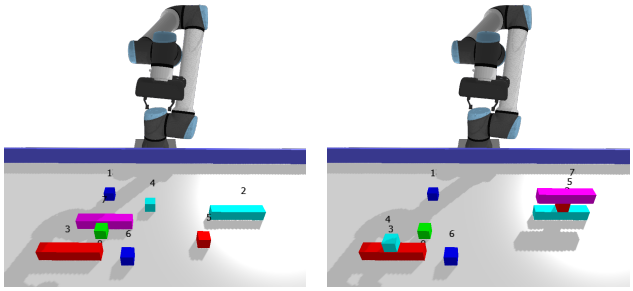

Fig. 6. Given an initial state (a) and a goal state (b), Relational DeepSym can find a plan to achieve the goal even though it is only trained with two to four objects.

B. Effect Prediction Results

We report the test set effect prediction results in Table I. Relational DeepSym performs better than other methods by a small margin, which aligns with the results in [13]. In Figure 3, we compare these methods by their cumulative effect prediction error when predicting a sequence of actions by feeding the prediction back into the state in an autoregressive fashion. Even though we do not see a significant difference between different methods, this does not directly translate to planning performance. Attentive DeepSym uses transformer layers to pass information between object symbols, and there is no straightforward way of translating the relational knowledge embedded in the transformer weights into lifted operators. This is the key advantage of the Relational DeepSym as it directly encodes relational symbols together with object symbols.

C. Planning Performance

In this section, we compare the planning performance of Relational DeepSym with the attentive formulation [12]. We skip the comparison with DeepSym as there is no straightforward way of generating rules for varying number of objects. We generate a random set of problem pairs $\{(\mathbf{X}_0^{(i)}, \mathbf{X}_g^{(i)})\}_{i=1}^{N}$ by executing random actions on the environment. For each problem pair, we convert $\mathbf{X}_0$ and $\mathbf{X}_g$ into their symbolic counterparts Σ_{p_0}, Σ_{r_0} and Σ_{p_g}, Σ_{r_g}, and produce PDDL problem statements. We filter out relations for object pairs that are not in contact in the goal state; otherwise, the planner

This article has been accepted for publication in IEEE Robotics and Automation Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LRA.2025.3527338

might fail to find a plan due to spurious relations. We use the Fast Downward planning system [6] and set a timeout limit to 10 seconds. We automatically check whether a plan is correct by computing cartesian distances of objects from the goal state and accept it as correct if the difference is less than 5cm for all objects.

Figure 4 shows the planning performance for different numbers of objects and actions. We also test each model with a noisy version of the environment where we add a noise of $\epsilon \sim \mathcal{N}(0, 0.01\text{cm})$ (i.e., approximately maximum 3cm) to observations throughout training and testing to simulate a more realistic scenario. Planning with the domain defined over unary and relational symbols generated by Relational DeepSym performs significantly better than the implicit attentive version in which all relational information remains hidden in the network's weights. We also test our method in a real-world experiment with the same environment definition (Figure 5). We detect the locations of objects with an Intel Realsense depth camera by clustering pixels and transform them into the robot frame. We manually define a goal configuration with its corresponding contact graph. The rest of the algorithm works the same as in simulation experiments.

Another advantage of using relational symbols in addition to unary symbols is that we can represent an environment configuration that has many more objects than the ones in the training set since the number of objects does not affect the encoded representation. This is not the case for the attentive formulation [12] as the model is biased towards the number of tokens in the training set, similar to other transformer-based architectures. In Figure 6, we initialize the scene with eight objects and create a goal configuration that was not in the train set. The planner successfully finds an action sequence that reaches the goal state.

D. Interpretation of Learned Symbols

We visualize the groundings of learned unary and relational symbols of a single run in Figure 7. We see that the unary symbol captures the type of the block regardless of its position whereas relational symbols capture the relative position of objects. Namely, Relation 1 models the z-axis differences between objects, Relation 2 captures the misalignment in the x-axis, and Relation 3 seems to be the inverse of Relation 2.

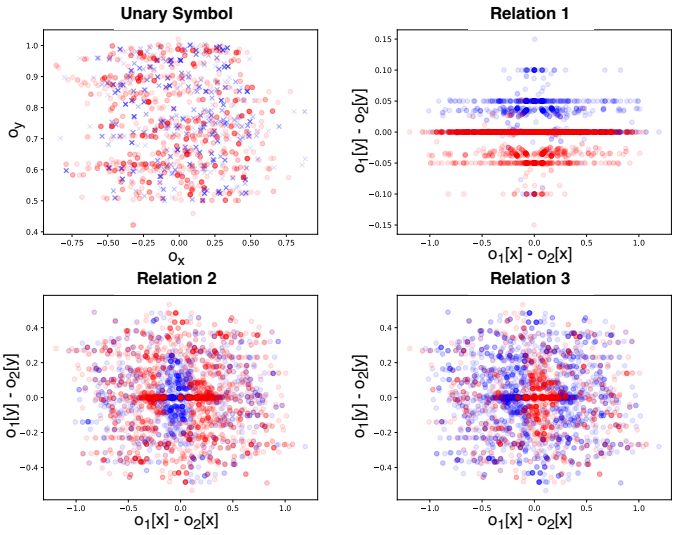

Fig. 7. Visualization of learned unary and relational symbols. Red points correspond to active symbols (i.e., $\sigma(.) = 1$) and blue points correspond to inactive symbols. Top left—Crosses and circles represent small and large blocks, respectively. We see that the unary symbol gets activated for large blocks. Relation 1 catches the relative z-axis position of objects whereas Relation 2 is active when objects are not aligned in the x-axis. Relation 3 seems to be the inverse of Relation 2.

V. Discussion

The learned operators become increasingly complex—and possibly overfit—as the number of objects increases. This is unavoidable if the action indeed changes the states of all objects. However, this is rarely the case in practice as a robot's skill only manipulate a few objects at a time. As such, it is practical to think that operators should model the dynamics of objects that are relevant. In our experiments, we define *relevance* as objects that are in the action arguments and objects that are in contact with these argument objects. A more structured treatment can be done by using deictic references [35], [36], [37], [38]. Deictic references are unique pointers to objects with respect to action arguments and relations to those. 'the object I pick' and 'the object that has an active relation σ_r with the object I pick' are two example deictic references. An alternative relevance might be to consider objects in the vicinity of action arguments. Although this creates a limitation, most of our daily activities can be represented with this relevance. An example exception is controlling a race car with a joystick; learning the relevance between such objects should be treated as a separate problem.

As with any learning algorithm, the learned symbols, relations, and rules depend on the training data. Learning rules that cover the deeper parts of the state graph (e.g., stacking a fourth block on top of three blocks) is naturally harder as the agent cannot collect enough transition data to cover these states. Better exploration strategies that are guided by the learned model in a feedback loop can possibly resolve this issue. In addition, universal quantifiers are needed in the rule learning procedure to learn more generic rules such as 'stacking a block on top of n blocks', instead of learning 'stacking a block on top of two blocks' and 'stacking a block on top of three blocks' separately. This would also reduce the sample complexity as these two cases become equivalent with the use of universal quantifiers (i.e., all blocks that are in the stack). We believe these are the main bottlenecks that will persist regardless of the model used, and therefore, focusing on them would be a promising direction.

In our experiments, we compared the proposed pipeline with our previous symbol learning method [12]. While there are notable works [19], [10], [26], [23] that also learn symbolic representations in the object-centric setting, providing a fair comparison with these works require substantial changes to them. The main differences are a combination of the following: (1) the lack of learning generic relational symbols, (2) the use of pre-defined predicates, (3) no neural network integration. A comparison of these algorithms in simple domains that can be

expressed both with and without relations would be beneficial to understand the added value of relational symbols.

VI. Conclusion

In this study, we proposed and implemented a framework where object and relational predicates are discovered from the continuous sensory experience of the robot, symbolic rules that encode the transition dynamics that involve pick and place actions on arbitrary numbered compound objects are extracted, these rules are automatically transferred to PDDL and symbolic plans are generated and executed to achieve various goals. We showed that the planning performance of our framework significantly outperforms the baseline. The key factor for the performance increase is representing the environment with relational symbols in addition to object symbols. In future work, we plan to use the learned operators to search for novel states and effects, which would drastically increase the sample efficiency and thus, the learning progress.

References

[1] G. Konidaris, "On the necessity of abstraction," *Curr. Opin. Behav. Sci.*, vol. 29, pp. 1–7, 2019.

[2] G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "Constructing symbolic representations for high-level planning," in *Proc. AAAI Conf. Artif. Intell.*, 2014, pp. 1932–1940.

[3] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *J. Artif. Intell. Res.*, vol. 61, pp. 215–289, 2018.

[4] E. Ugur and J. Piater, "Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 2627–2633.

[5] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl— the planning domain definition language," 1998, [Online]. Available: https://api.semanticscholar.org/CorpusID:59656859.

[6] M. Helmert, "The fast downward planning system," *J. Artif. Intell. Res.*, vol. 26, pp. 191–246, 2006.

[7] J. Hoffmann, "FF: The fast-forward planning system," *AI magazine*, vol. 22, no. 3, pp. 57–57, 2001.

[8] M. Asai and A. Fukunaga, "Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 6094–6101.

[9] M. Asai and C. Muise, "Learning neural-symbolic descriptive planning models via cube-space priors: The voyage home (to strips)," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, 2021, pp. 2676–2682.

[10] M. Asai, H. Kajino, A. Fukunaga, and C. Muise, "Classical planning in deep latent space," *J. Artif. Intell. Res.*, vol. 74, pp. 1599–1686, 2022.

[11] A. Ahmetoglu, M. Y. Seker, J. Piater, E. Oztop, and E. Ugur, "Deepsym: Deep symbol generation and rule learning for planning from unsupervised robot interaction," *J. Artif. Intell. Res.*, vol. 75, pp. 709–745, 2022.

[12] A. Ahmetoglu, E. Oztop, and E. Ugur, "Learning multi-object symbols for manipulation with attentive deep effect predictors," *arXiv:2208.01021*, 2022.

[13] A. Ahmetoglu, B. Celik, E. Oztop, and E. Ugur, "Discovering predictive relational object symbols with symbolic attentive layers," *IEEE Robotics and Automation Letters*, vol. 9, pp. 1977–1984, 2024.

[14] E. Ugur, E. Şahin, and E. Oztop, "Unsupervised learning of object affordances for planning in a mobile manipulation platform," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 4312–4317.

[15] P. Zech, S. Haller, S. R. Lakani, B. Ridge, E. Ugur, and J. Piater, "Computational models of affordance in robotics: a taxonomy and systematic classification," *Adaptive Behavior*, vol. 25, no. 5, pp. 235–271, 2017.

[16] T. Taniguchi, T. Nagai, T. Nakamura, N. Iwahashi, T. Ogata, and H. Asoh, "Symbol emergence in robotics: a survey," *Adv. Robot.*, vol. 30, no. 11–12, pp. 706–728, 2016.

[17] S. James, B. Rosman, and G. Konidaris, "Learning portable representations for high-level planning," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 4682–4691.

[18] E. Ugur and J. Piater, "Refining discovered symbols with multi-step interaction experience," in *Proc. IEEE-RAS 15th Int. Conf. Humanoid Robots*, 2015, pp. 1007–1012.

[19] S. James, B. Rosman, and G. Konidaris, "Autonomous learning of object-centric abstractions for high-level planning," in *Proc. Int. Conf. Learn. Representations*, 2022, [Online]. Available: https://openreview.net/forum?id=rrWeE9ZDw_.

[20] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez, "Learning symbolic operators for task and motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 3182–3189.

[21] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Learning neuro-symbolic relational transition models for bilevel planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 4166–4173.

[22] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Learning neuro-symbolic skills for bilevel planning," in *Proc. Conf. Robot Learn.*, 2022, pp. 701–714.

[23] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Pérez, L. Kaelbling, and J. B. Tenenbaum, "Predicate invention for bilevel planning," in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 12 120–12 129.

[24] N. Kumar, W. McClinton, R. Chitnis, T. Silver, T. Lozano-Pérez, and L. P. Kaelbling, "Learning efficient abstract planning models that choose what to predict," in *Proc. Conf. Robot Learn.*, 2023, pp. 2070–2095.

[25] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.

[26] N. Shah, J. Nagpal, P. Verma, and S. Srivastava, "From reals to logic and back: Inventing symbolic vocabularies, actions and models for planning from raw data," *arXiv:2402.11871*, 2024.

[27] D. Molina, K. Kumar, and S. Srivastava, "Learn and link: Learning critical regions for efficient planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 10 605–10 611.

[28] N. Shah and S. Srivastava, "Using deep learning to bootstrap abstractions for hierarchical robot planning," *arXiv:2202.00907*, 2022.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Proc. Adv. Neural Inf. Process. Syst.*, pp. 6000–6010, 2017.

[30] C. Diuk, A. Cohen, and M. L. Littman, "An object-oriented representation for efficient reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2008, pp. 240–247.

[31] F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf, "Object-centric learning with slot attention," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 11 525–11 538, 2020.

[32] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv:1611.00712*, 2016.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015, [Online]. Available: https://arxiv.org/abs/1412.6980.

[34] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," *Proc. Adv. Neural Inf. Process. Syst.*, pp. 901–909, 2016.

[35] P. E. Agre and D. Chapman, "Pengi: An implementation of a theory of activity," in *Proc. 6th National Conf. Artif. Intell.*, 1987, pp. 268–272.

[36] S. Finney, N. H. Gardiol, L. P. Kaelbling, and T. Oates, "Learning with deictic representation," 2002, [Online]. Available: http://www.ai.mit.edu/research/abstracts/abstracts2001/machine-learning/05kaelbling1.pdf.

[37] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling, "Learning symbolic models of stochastic domains," *J. Artif. Intell. Res.*, vol. 29, pp. 309–352, 2007.

[38] O. Marom and B. Rosman, "Zero-shot transfer with deictic object-oriented representation in reinforcement learning," *Proc. Adv. Neural Inf. Process. Syst.*, pp. 2297–2305, 2018.