DeepSym: Deep Symbol Generation and Rule Learning for Planning from Unsupervised Robot Interaction

Alper Ahmetoglu alper.ahmetoglu@boun.edu.tr
M. Yunus Seker yunus.seker1@boun.edu.tr
Department of Computer Engineering, Bogazici University, Istanbul, Turkey

Justus Piater justus.piater@uibk.ac.at
Department of Computer Science, Universität Innsbruck, Austria

Erhan Oztop erhan.oztop@ozyegin.edu.tr
Osaka University, Osaka, Japan / Ozyegin University, Istanbul, Turkey

Emre Ugur emre.ugur@boun.edu.tr
Department of Computer Engineering, Bogazici University, Istanbul, Turkey

Abstract

Autonomous discovery of symbols and rules from the interaction of a robot with its environment is a crucial step for empowering robots with intelligence. Yet, it remains a challenging problem. For robots that are expected to reason and perform in open-ended environments, manual design of symbols must be substituted with automatic symbol formation and rule extraction for scalability, flexibility, and robustness. Towards this goal, we propose a novel general method that finds action-grounded, discrete object and effect categories and builds probabilistic rules over them for non-trivial action planning. Our robot interacts with objects using an initial action repertoire and observes the effects it can create in the environment. To form action-grounded object, effect, and relational categories, we employ a binary bottleneck layer in a predictive, deep encoder-decoder network that takes as input the image of the scene and the action applied as input, and generates the resulting effects in the scene in pixel coordinates. After learning, the binary latent vector represents action-driven object categories based on the interaction experience of the robot. To distill the knowledge represented by the neural network into rules useful for symbolic reasoning, a decision tree is trained to reproduce its decoder function. Probabilistic rules are extracted from the branches of the tree and represented in the Probabilistic Planning Domain Definition Language (PPDDL), allowing off-the-shelf planners to operate on the knowledge extracted from the sensorimotor experience of the robot. To test the proposed approach, a simulated robotic manipulator is used which interacted with the available objects through push and stack actions. The robot learned symbols that can be interpreted as 'rollable', 'insertable', 'larger-than', and generated effective plans to achieve goals such as building towers of desired height from the available objects, demonstrating the effectiveness of our approach.

1. Introduction

Intelligent robotic systems exploit a diverse range of data representations for control, learning, and reasoning. Interaction with the world requires processing low-level continuous sensorimotor representations whereas abstract reasoning requires the use of high-level symbolic representations. The representational gap between the low-level sensorimotor and high-level symbolic representations have been addressed in AI and robotics, often by using manually designed symbols that are grounded in the low-level sensorimotor experience of the robots interacting with their environment (Harnad, 1990; Taniguchi et al., 2019). However, this approach only works in either controlled environments or limited tasks. Yet, truly intelligent robots are expected to form abstractions (Konidaris, 2019) continually from their interaction with the world and use them on-the-fly for complex planning and reasoning in novel environments (Werner and Kaplan, 1963; Callaghan and Corbit, 2015).

In this paper, we address the challenging problem of autonomous discovery of discrete symbols and unsupervised learning of rules from the low-level interaction experience of a self-exploring robot. For this purpose, we propose a novel end-to-end deep neural architecture for symbol formation and rule extraction. At the core of our method, the symbols are discovered in the discrete latent space formed by the bottleneck layer of a predictive, deep encoder-decoder network that takes the image of an object and the action applied as the input, and produces the effect generated by the action as the output. Furthermore, our architecture allows transforming the complete low-level sensorimotor experience into symbolic experience, facilitating direct rule extraction for AI planning. To this end, decision tree models are trained to learn probabilistic rules that are translated to Probabilistic Planning Domain Definition Language (PPDDL; (Younes and Littman, 2004)) operators that are standard in probabilistic planning. Note that the predicates that appear in the PPDDL operators correspond to the discovered symbols.

In order to realize this framework, we created a setup where a simulated robot manipulator interacts with objects, poking them in different directions and stacking them on top of each other to collect interaction experience for object categorization and rule learning. Our system successfully constructs a latent representation through which object and relational categories are discovered, which can be interpreted by humans as 'rollable', 'insertable', 'larger-than'. Contrary to symbols generated by systems that disregard actions and effects, our architecture is shown to generate action-effect-regulated symbols that are more effective in abstract reasoning over the actions of the robot and the consequences in the environment. Furthermore, the number of symbols is determined automatically by optimizing the trade-off between prediction capability and bottleneck size. Finally, the system acquired the capability to generate effective plans to achieve goals such as building towers of desired heights from given cubes, balls, and cups using off-the-shelf probabilistic planners. Our implementation is publicly available[1].

Our primary contribution is a generic end-to-end neural solution for mapping raw sensorimotor experience into the symbolic domain. The same architecture can be used to discover object categories, effect categories, and object-object relational categories. The proposed network further allows progressive learning of increasingly complex abstractions, exploiting previously-learned abstractions as inputs. The learned categories allow abstraction of the interaction of the robot with its environment as a Markov decision process which allows the use of symbolic planning systems for goal satisfaction. In the current study, to show this, we transformed the learned rules into Probabilistic PDDL operators, which allowed probabilistic plan generation and execution achieving goals beyond what was possible with the direct use of the training data.

1. `https://github.com/alper111/DeepSym`

2. Related Work

Bridging the representational gap between the continuous sensorimotor world of a robotic system with the discrete symbols and rules have been a key research goal from the early days of intelligent robotics (Kuipers et al., 2017; Murphy and Murphy, 2000). While grounding predefined symbols in the sensorimotor experience of the robot has been widely used for intelligent robot control (Klingspor et al., 1996; Petrick et al., 2008; Mourao et al., 2008; Wörgötter et al., 2009; Kulick et al., 2013), some argue that symbols "are not formed in isolation", and "they are formed in relation to the experience of agents" (Sun, 2000). We share this viewpoint that has been investigated in a number of studies. Pisokas and Nehmzow (2005) and Ugur et al. (2011) realized systems that clustered low-level sensory experience into categories and performed subsymbolic planning in the continuous perceptual space. While simple planning capability was achieved, the use of continuous prediction and state transition operators limited the use of powerful off-the-shelf symbolic AI planners. In another line of research, Ozturkcu et al. (2020) asked whether there are any symbols formed in a deep RL agent after training the agent for a given task, without imposing any prior on the architecture or the objective.

The bottom-up generation of symbolic structures from continuous interaction experience of a robot has started to draw attention in robotics (Taniguchi et al., 2019; Konidaris, 2019). Konidaris et al. (2014, 2015) studied the construction of symbols that are directly used as preconditions and effects of actions for the generation of deterministic and probabilistic plans in 2d agent settings and Konidaris et al. (2018) extended the framework into a real-world robot setting. However, these studies use a global state representation, and therefore, symbols learned in an environment cannot be used in a novel environment directly. In follow-up work, James et al. (2019) constructs symbols with egocentric representations to allow the transfer of previously learned symbols. These studies train an SVM classifier for each effect cluster to find groundings of precondition symbols. Ugur and Piater (2015a,b) formed symbols used in plan generation in manipulation using a combination of several ad-hoc machine learning techniques such as clustering with X-means and classification with SVMs. Furthermore, they used hand-crafted features to represent scenes and effects. The main difference between our work compared to the previous ones is that a prior separate effect clustering or categorization step is not required in our framework. Rather, we provide a generic and principled symbol formation engine that runs at the pixel level in an end-to-end fashion using deep neural networks. In terms of symbol multiplicity, our approach is more parsimonious, as we do not form symbols for each action as in Ugur and Piater (2015b) and Konidaris et al. (2018); but instead, use a single decoder network that takes the action as part of the input. To be concrete, for n effect categories and k actions, our system generates nk symbols whereas the aforementioned approaches generate n^k symbols. Another big advantage of our model is that it is differentiable and thus can be integrated into gradient-based state-of-the-art machine learning architectures for further tackling more complex problems.

Asai and Fukunaga (2017) realized a similar neural framework where they first train a state autoencoder with discrete latent units, then learn the action precondition-effect mappings. In follow-up work, (Asai and Muise, 2020) combine these two steps and learn the action mapping together with the state auto-encoder. These works are in the visual domain (for example, 2-d puzzles), and achieve visualized plan execution while we focus on robot action planning and execution in the 3D world. Moreover, a critical difference of our method from the aforementioned work is that we learn object symbols taking into account action and the effects in addition to object features, which facilitates the formation of symbols that are likely to capture object affordances (Gibson, 2014; Zech et al., 2017).

3. Problem Formulation

We are interested in learning discrete symbols for object and effect categories by interacting with the environment. The reason to learn such symbols is that it allows one to define the interaction of the robot with the environment as a Markov decision process (MDP). When we have such an MDP, we can solve a task (within the resolution of the MDP) with the state-of-the-art planning systems. For simplification, we make the following assumptions:

- The agent is assumed to have a set of actions such as poking and stacking an object. Such an action repertoire can be autonomously acquired through a developmental progression as in Ugur et al. (2012) or obtained through learning from demonstration and reinforcement learning (e.g. Seker et al., 2019; Akbulut et al., 2020).
- The agent can track objects to observe the generated effects as displacements. In the tabletop setup, we realized this with a simple algorithm. In a real-world scenario, state-of-the-art computer vision techniques can be used to detect and track objects in the 3d world.

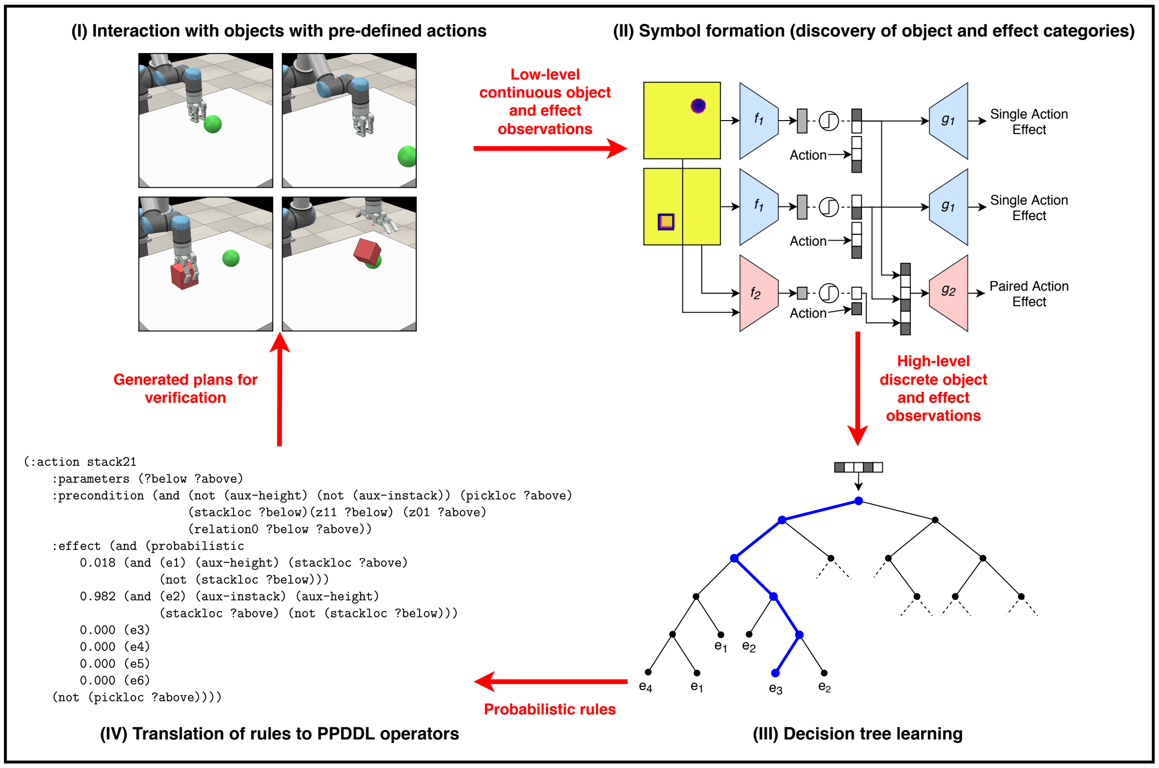
4. Methods

Figure 1 provides the overall learning architecture of our proposed system. In the environment interaction phase (I), the robot chooses an action from its action repertoire $a \in \mathcal{A} = \{a_1, a_2, \ldots, a_k\}$, and applies it to an object o and observes the resulting effect e.

Using the interaction experience, symbol formation is achieved in (II). To this end, a deep neural network model with two parts is trained to learn the conditional distribution $p(e|o,a)$. The first part is an encoder network, $f(z|o)$, which creates a *binary* latent vector z given the depth image of the object. The second part, a decoder network $g(e|z,a)$, predicts the effect when action a is executed on an object o that has the latent representation z. As the network tries to predict effects, symbolic representations are created by the encoder network which can be treated as object categories regulated by the corresponding action-effect experience.

The continuous interaction experience is transformed into the symbolic experience using the discovered categories, and then the symbolic experience is used to distill a decision tree to predict effects given object categories and actions in (III). The reason to use a decision tree is that we can represent any statement in propositional logic with decision trees (Russell and Norvig, 2009, Ch18.3) and we can convert rules of the environment into logical statements that encode pre- and post-conditions of actions on the objects.

Finally, these statements are represented in PPDDL which allows one to make plans in a probabilistic environment in (IV). Lastly, plans are executed to validate the learned symbols and rules. In the following sections, we describe these parts in detail.


(I) Interaction with objects with pre-defined actions
(II) Symbol formation (discovery of object and effect categories)
Low-level continuous object and effect observations
f_1
f_1
f_2
Action
Action
Action
g_1
g_1
g_2
Single Action Effect
Single Action Effect
Paired Action Effect
High-level discrete object and effect observations
Generated plans for verification
e_4 e_1 e_1 e_2 e_3 e_2
Probabilistic rules
(IV) Translation of rules to PPDDL operators
(III) Decision tree learning

```
(:action stack21
    :parameters (?below ?above)
    :precondition (and (not (aux-height) (not (aux-instack)) (pickloc ?above)
                       (stackloc ?below)(z11 ?below) (z01 ?above)
                       (relation0 ?below ?above))
    :effect (and (probabilistic
        0.018 (and (e1) (aux-height) (stackloc ?above)
                   (not (stackloc ?below)))
        0.982 (and (e2) (aux-instack) (aux-height)
                   (stackloc ?above) (not (stackloc ?below)))
        0.000 (e3)
        0.000 (e4)
        0.000 (e5)
        0.000 (e6)
    (not (pickloc ?above))))
```

Figure 1: General system overview of rule generation and refinement.

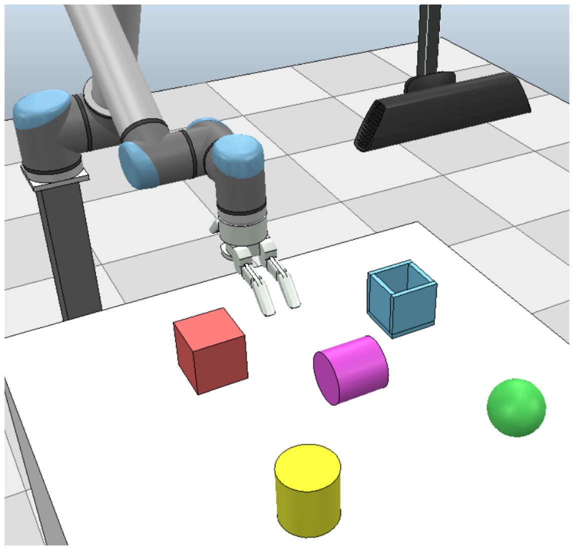
4.1 Exploration with the Environment

A manipulator robot with a gripper and a depth camera is used to explore the environment and monitor the changes (Figure 2). The robot is initialized with a fixed set of actions through which it interacts with the objects in its workspace. Forward, side, and top poking actions are used to poke objects from different sides (Figure 2b, top). The stacking action is used to release one object on top of another object (Figure 2b, bottom). These actions are encoded with one-hot encoding. On the perception side, each detected object is represented with its top-down depth image. The generated change, on the other hand, is represented by the positional offset of the acted object in pixel coordinates together with the force change sensed at the wrist joint of the robot. In single-object interactions, the robot observes and stores the initial state as the object-centered, top-down depth image of the object, and the effect as the change in object position and force sensor readings:

$$e_{\text{single}} = (\Delta x, \Delta y, \Delta d, \Delta F) \tag{1}$$

In paired-object interactions, the robot observes and stores the initial state as the combination of two object-centered depth images (o_1, o_2), and the effect as the change in position of both objects:

$$e_{\text{paired}} = (\Delta x_1, \Delta y_1, \Delta d_1, \Delta x_2, \Delta y_2, \Delta d_2) \tag{2}$$


(a) Experiment setup.

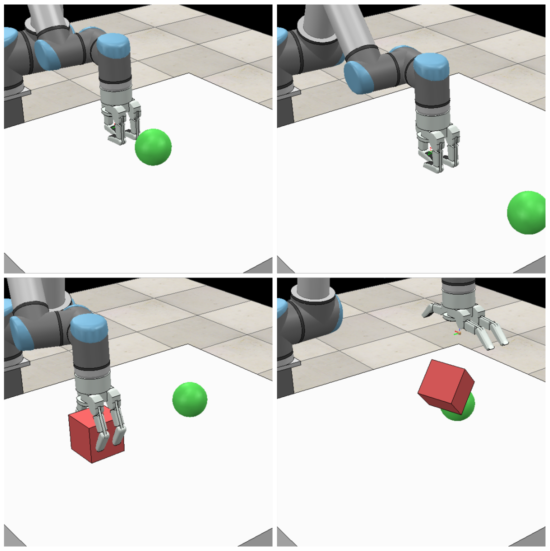

(b) Available actions.

Figure 2: A simulated UR10 robot arm and BarrettHand grasper are used for manipulation; a Kinect sensor is used for perception. Five types of objects are shown on the table.

4.2 Symbol Discovery with Deep Networks

The main objective of the network is to discover symbols, i.e. object and effect categories, that are effective in abstract reasoning about consequences of robot actions. In other words, the object categories together with robot actions should give the ability to predict the effect categories. To achieve this, we propose a special neural network structure which is composed of two parts: an encoder for representing $f(z|o)$, where z corresponds to object category, and a decoder for representing $g(e|z,a)$ (Figure 3, top). As the input is a top-down depth image, the encoder is a convolutional neural network with the $\operatorname{sign}(x)$ function as the last layer activation function (where the error back-propagation is handled with straight-through estimation (STE) (Bengio et al., 2013; Courbariaux et al., 2016)). Via $\operatorname{sign}(x)$ activation of the bottleneck neurons, the continuous object representation is directly transformed to a discrete category. The decoder part is realized as a multi-layer perceptron (MLP). The category z of the object o concatenated with the one-hot vector of action a is given to the decoder as input. The decoder predicts the effect e expected to be observed on object o via action a. The network minimizes the following objective:

$$\mathcal{L} = \sum_{i=1}^{N} \frac{1}{2}\left(\bar{e}^{(i)} - e^{(i)}\right)^2 \tag{3}$$

$$z^{(i)} = \operatorname{sign}(f(o^{(i)})) \quad \bar{e}^{(i)} = g(z^{(i)}, a^{(i)}) \tag{4}$$

This architecture effectively creates high-level symbolic categories of objects that encapsulate the effects of executed actions. One important advantage is that the model does not need hand-engineered object features and object clusters for finding object symbols, contrary to previous studies, since the system learns *discrete* categories in an *end-to-end*

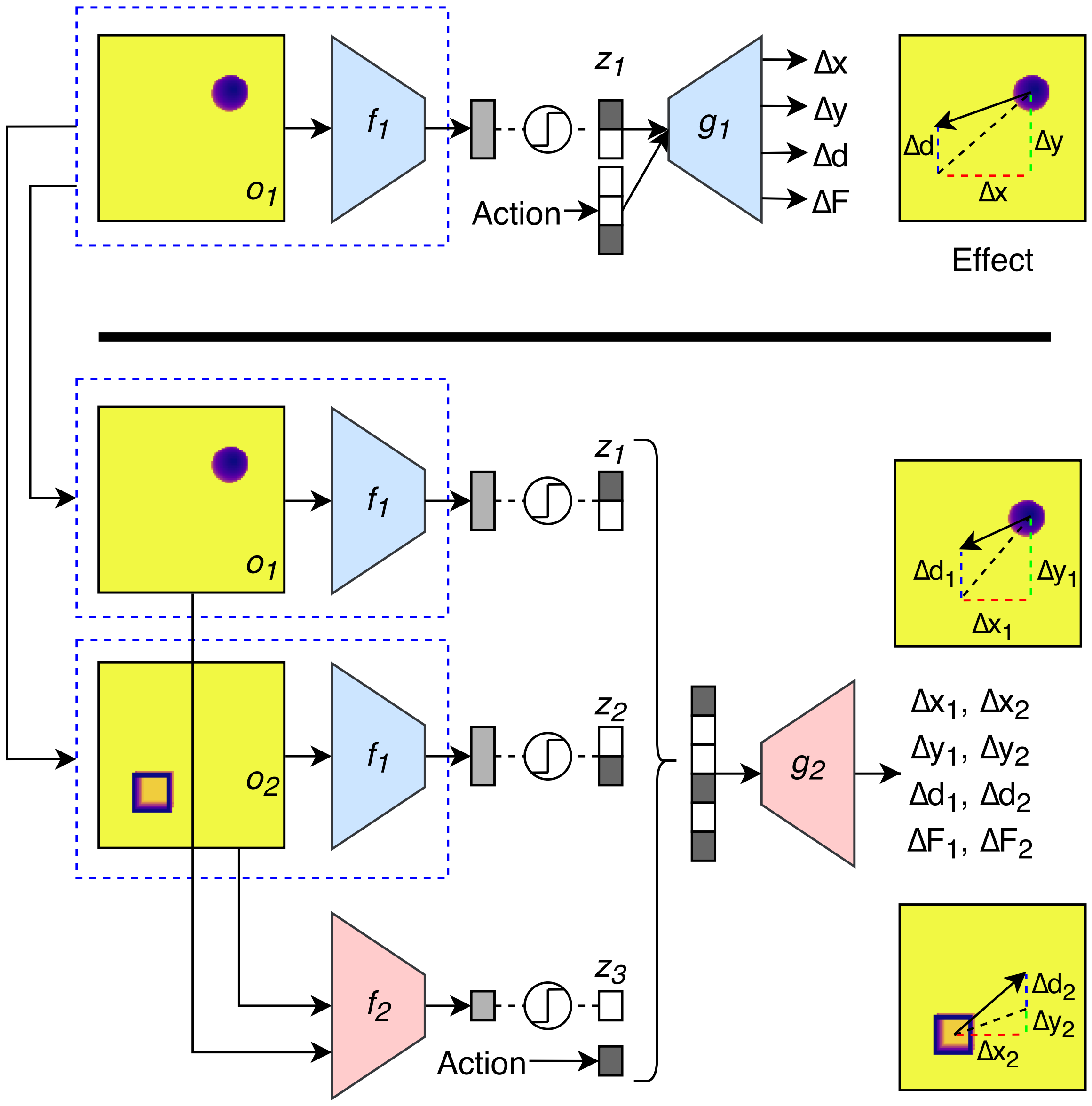


Figure 3: Network architectures for single-object interactions (top) and for paired-object interactions (bottom).

fashion. Moreover, as the bottleneck layer is discrete, the possible decoder outputs form a finite set and thus can be used to represent discrete effects.

The learned object categories also serve as input for the discovery of new categories with new interaction experience such as actions applied on pairs of objects such as stacking an object o_1 on top of another object o_2 (Figure 2b, bottom). The same deep network structure is used to extract the corresponding symbols with a slight modification to incorporate previously learned knowledge (Figure 3, bottom). Here, an encoder f_2 takes the depth images of the objects and produces a binary latent vector z_3. As the important point here, the single object symbols (z_1 and z_2) that are computed by the f_1 encoder are also added to the network as input together with the action information. The idea is that we can use previously-acquired symbols to encode new information more compactly, thus allowing a progressive increment of symbols. Note that the encoder f_1 is frozen at this second stage of the training. Including single object symbols are expected to provide some interaction related information about objects and let the encoder f_2 focus and learn properties and relations *between* the objects.

Number of symbols is automatically set by selecting the number of bottleneck neurons using a hyperparameter search procedure. To limit the number of rules and predicates, this procedure aims to find the minimum number of symbols that provide competitive performance in prediction. Starting from one unit, we record the mean and the standard deviation of mean square errors (MSE) of multiple runs. We increase the number of units until there is no significant drop in the prediction error. MSE curves are reported in Appendix A.

4.3 Extracting Symbolic Rules

In the third part of the pipeline, a decision tree is trained to predict the effect of the stack action given high-level single and paired object categories. Here, the aim is to extract the probabilistic rules of the environment by converting the decision rules on the branches of the tree into logical statements, which ultimately enables probabilistic planning. The effect space induced by the actions is further reduced by an additional effect clustering for the sake of removing near-duplicate effect categories. The decision tree which is trained with discrete input-output pairs is converted into PPDDL descriptions. Each path in the tree corresponds to a rule where the truth values of categories in the intermediate decision nodes are used as precondition predicates and the terminal node corresponds to the effect.

Our motivation to construct PPDDL descriptions is to use probabilistic AI planners to efficiently make plans and execute them. PPDDL is composed of a domain description and a problem definition. In the domain description, there are predicates and actions. Predicates represent boolean values that can be activated or deactivated. Each action has a precondition, which is a set of predicates that needs to be satisfied, and an effect, which activates/deactivates other predicates. The domain description is generated from the list of rules. In the problem definition, the initial state of the world is encoded along with the goal to be satisfied. To encode the initial state, the robot perceives the current environment and sets the truth values of the predicates for the existing categories. The planner finds the sequence of actions to satisfy the predicates given in the goal description starting from the initial state using the actions defined in the domain description.

5. Experiments and Results

Interactions: We adopted the robotic setup, including the action and object sets used, from Ugur and Piater (2015a) who showed effective skill transfer from simulator to real-world, involving actions with 3-fingered prehension. The experiments are performed in CoppeliaSim VREP simulator (Rohmer et al., 2013) where a six degrees-of-freedom UR10 (Universal Robots, 2012) robot arm and a Barrett Hand system (Townsend, 2000) interacts with the objects on the table and a top-down facing Kinect sensor is used for environment perception (Figure 2). The objects used in the experiments include rectangular cups, horizontally and vertically placed cylinders, spheres, and cubes. For each object type, 10 different objects with varying diameters/edge lengths in the range of 10 to 20 cm are included in the object dataset for interaction.

Perception: Before each action execution a top-down depth image (128×128 pixels) of the scene is captured. Objects are placed at different reachable locations on the table during the interactions to ensure the network to be invariant of the perspective. Pixels of the images are normalized globally to increase the convergence speed of stochastic gradient descent (LeCun et al., 2012). Objects in the image are detected with a simple procedure by finding the point with minimum depth and cropping the area of 42×42 pixels centered around it. This procedure yields object-centered representations for the objects used in the current study but preserves the perspective distortion due to varying locations of the objects and fixed sensor position.

Encoder-decoder network: The encoder network (Figure 3) consists of four blocks each containing two convolutional layers that are followed by batch normalization (Ioffe and Szegedy, 2015) and ReLU activation. The number of filters in these blocks are 32, 64, 128, and 256. The last layer consists of two hidden units with a $\mathrm{sign}(x)$ activation function. STE (Bengio et al., 2013) is used for backpropagating gradients. The decoder network is a two-layer MLP with 32 hidden units. Further details of these networks can be found in Appendix A.

5.1 Discovered Object Categories

Based on the hyperparameter optimization procedure, the number of binary activation neurons in the bottleneck layer is automatically set to 2; therefore the system found $2^2 = 4$ object categories. How different object types (unknown to the robot) are represented by the discovered object categories is analyzed and provided in Table 1. We denote the activation (-1, +1) as (0,1) throughout the text for the ease of readability. In general, different types of objects were coded into different categories except that cube and vertical cylinder share the same category even though their depth images differ. This is due to our action and effect regulated categorization: cubes and vertical cylinders behave the same under all available single-object actions of the robot. Although the depth images of the same type of objects with different sizes differ significantly, this information is not reflected in the categories, because the size of the objects does not have a significant influence on the consequences of the current actions. The categories can be interpreted as: ‘rollable in single direction’; ‘rollable in all directions’; ‘pushable’; and ‘pushable and insertable’, respectively. Examples

DeepSym				
Category	(0, 0)	(0, 1)	(1, 0)	(1, 1)
Sphere	**92.9 ± 8.6**	2.5 ± 4.4	3.2 ± 6.8	1.4 ± 2.3
Cube	1.4 ± 3.1	**92.9 ± 10.3**	3.4 ± 5.6	2.3 ± 3.7
Vertical Cylinder	1.9 ± 4.6	**93.6 ± 5.4**	1.8 ± 2.7	2.7 ± 3.1
Horizontal Cylinder	15.8 ± 27.3	7.6 ± 10.7	**74.7 ± 27.0**	2.0 ± 3.8
Cup	0.1 ± 0.2	0.0 ± 0.0	0.0 ± 0.0	**99.9 ± 0.2**
Autoencoder (OBO)				
Category	(0, 0)	(0, 1)	(1, 0)	(1, 1)
Sphere	**85.1 ± 12.1**	12.3 ± 9.6	2.6 ± 4.3	0.0 ± 0.0
Cube	**74.6 ± 17.7**	20.4 ± 12.0	5.0 ± 7.0	0.0 ± 0.0
Vertical Cylinder	**76.2 ± 15.6**	18.3 ± 9.4	5.5 ± 8.5	0.0 ± 0.0
Horizontal Cylinder	**78.1 ± 14.4**	19.2 ± 11.1	2.8 ± 3.8	0.0 ± 0.0
Cup	**92.4 ± 14.1**	6.6 ± 13.5	0.9 ± 2.8	0.1 ± 0.2
Continuous bottleneck + clustering (OCEC)				
Category	(0, 0)	(0, 1)	(1, 0)	(1, 1)
Sphere	**94.2 ± 9.4**	4.0 ± 8.4	1.8 ± 5.5	0.0 ± 0.0
Cube	0.0 ± 0.0	**99.0 ± 3.2**	0.0 ± 0.0	1.0 ± 3.2
Vertical Cylinder	0.0 ± 0.0	**98.5 ± 4.7**	0.0 ± 0.0	1.5 ± 4.7
Horizontal Cylinder	28.6 ± 29.1	0.0 ± 0.0	**63.4 ± 26.1**	8.0 ± 16.9
Cup	0.0 ± 0.0	13.8 ± 32.5	0.0 ± 0.0	**86.2 ± 32.5**

Table 1: The relative assignment frequencies of objects to different symbolic categories. Here, objects vary in their sizes and initial positions. The mean and the standard deviation of 10 runs is reported.

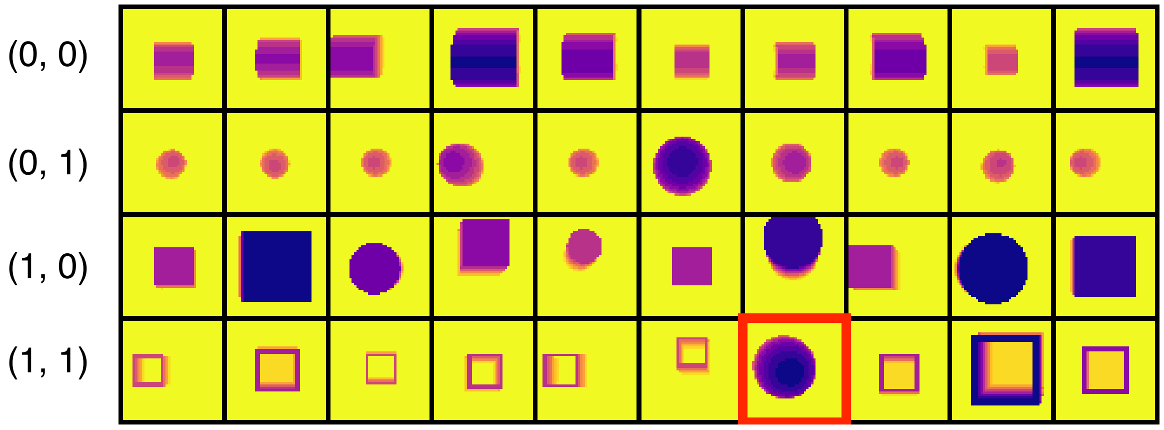


Figure 4: Example depth images as inputs to the encoder network f_1. One sphere categorized incorrectly as hollow by f_1 is indicated by the red frame.

from each category are shown in Figure 4. In the provided dataset, the sphere shown with a red frame is incorrectly encoded in the 'pushable and insertable' category.

As a baseline for comparison, we trained

1. An autoencoder with a binary hidden layer to reconstruct the depth images of objects (inputs to f_1), instead of effects. This approach is similar to Asai and Fukunaga (2017). Let us refer to this approach as Object-Binary-Object (OBO).

2. Our proposed encoder-decoder architecture with the binary bottleneck layer replaced with a usual continuous layer that is applied k-means clustering ($k = 4$) after learning. Let us call this approach Object-Continuous-Effect followed by Clustering (OCEC).

The results are shown in Table 1. For the autoencoder network (i.e. OBO), we see that objects are collapsed primarily into one category. The robot is expected to predict the consequences of its actions using these categories, and as shown, these categories are not distinctive to help such prediction. With this, we verified the advantage of extracting the symbols from the interaction experience of the robot that includes object-action-effect information, i.e. from an object encoder - effect decoder network, rather than searching the symbols in *passively-observed* static features.

OCEC gave better results compared to OBO since the bottleneck layer in OCEC *does* include information from the effect space because of the predictive training similar to our proposed model. However, the latent codes in the bottleneck layer of OCEC might not be distributed locally which would make the clustering harder. When this is the case, we would need more complicated clustering algorithms such as spectral clustering to cluster the latent space accurately. For example, in Table 1, we see that OCEC is more biased to misclassify cups as the stable category, and the horizontal cylinders as spheres. When we take an average over all objects, our method predicts objects in the correct category with 90.8 ± 10.3 % accuracy compared to OCEC.

5.2 Discovered Relational Categories

Stack interaction experience of the robot is used to train the multi-object encoder-decoder network (Figure 3 bottom) transferring the object categories reported above. The number of binary activated bottleneck neurons is automatically set to 1 using the hyperparameter search described in the Methods section.

The response of the bottleneck neuron, i.e. how this neuron categorizes the input object pairs, is analyzed in Figure 5. Given different pairs of objects with different sizes, each image in this figure corresponds to a specific object pair, and each pixel provides the response of the bottleneck neuron (0 or 1) for specific object sizes. In our experiments, the effect of stack action depends on object categories and their relative size. For example, if an object is released on top of a larger cup, the released object drops into the cup. If the released object is larger, it is stacked on top of the walls of the cup. The approximately linear boundaries for some object pairs in Figure 5 (for example, the last column) shows that the bottleneck neuron captured these dynamics and found a symbol that roughly encodes the relative size; the output is 1 when the below cup is larger than the above object. In stacking interactions, the relative size relation only makes sense when the object below is a cup; and our system discovered this relational symbol.

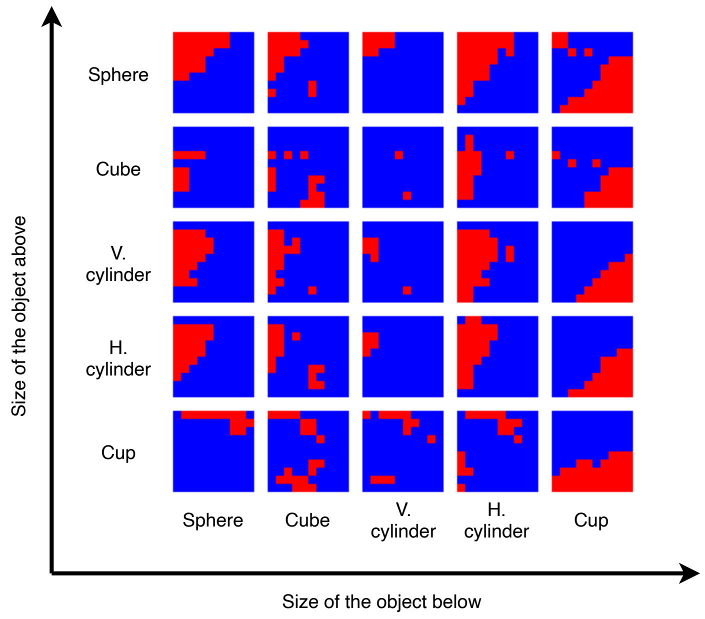


Figure 5: The encoder f_2 activations (blue for 0, red for 1) for paired objects. Here, x and y axes of each of the 5×5 plots represent the sizes of the objects below and above, respectively. Each square represents the relation for a given pair. Note that without any direct supervision, the system discovers approximately linear boundaries (e.g. last column) for some object pairs that would help in effect prediction.

5.3 Discovered Effect Categories

After training, we pass the state space $\mathcal{Z}$ of the bottleneck layer to the decoder to get the effect categories. More specifically:

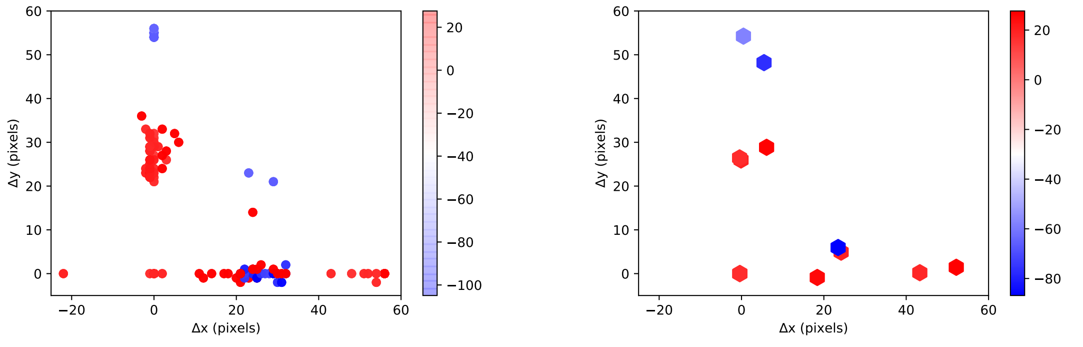
$$\mathcal{C}_{\text{single}} = g_1(\mathcal{Z}_{\text{single}}) \tag{5}$$

$$\mathcal{C}_{\text{paired}} = g_2(\mathcal{Z}_{\text{paired}}) \tag{6}$$

$$\tag{7}$$

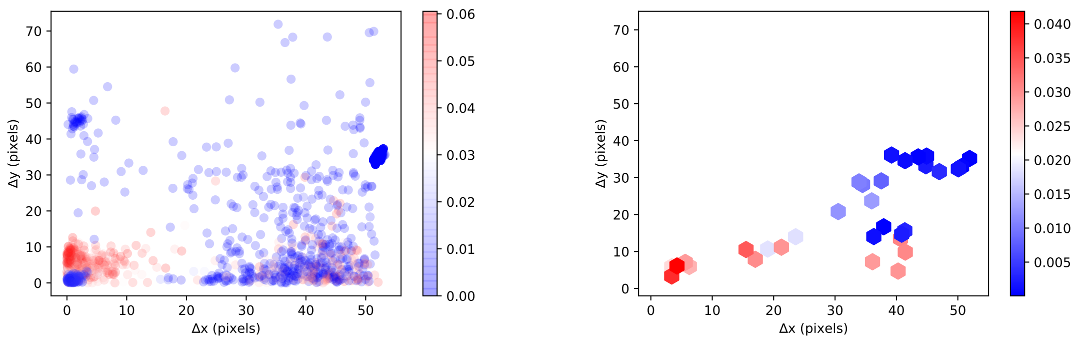
Here, $\mathcal{Z}_{\text{paired}}$ is the Cartesian product of the object category space $\{0,1\}^2$ with the action space $\mathcal{A}_{\text{single}} = \{(0,0,1),(0,1,0),(1,0,0)\}$ resulting in 12 different effect clusters for the single object effects. For the paired object effects, the input consists of two single object categories and one relational object category. Therefore, this number is $\{0,1\}^2 \times \{0,1\}^2 \times \{0,1\} \times \mathcal{A}_{\text{paired}} = 32$. Here, A_{paired} only contains the stack action, therefore $n(\mathcal{A}_{\text{paired}}) = 1$. These effect categories for the single and the paired interactions are shown in Figures 6 and 7, respectively. For visualization purposes, we use colors to represent the third dimension. In Figure 6, the low force values are in blue and the high force values are in red. Likewise in Figure 7, the low depth values are in blue and the high depth values are in red.

In both figures, the found effect categories are indeed representative points in the effect space. This is expected as we train the decoder to predict the generated effects. Yet, we see that some effect categories are close to each other. When we learn a decision tree based on these effect categories, we treat each category as a different class. However, these



(a) Observed effects, $\mathcal{E}_{\text{single}}$. (b) Found effect categories, $\mathcal{C}_{\text{single}}$.

Figure 6: Effect space for the single object interactions. The low force values are in blue, and the high force values are in red. Note that the found effect categories faithfully represent the effect space without any prior clustering.



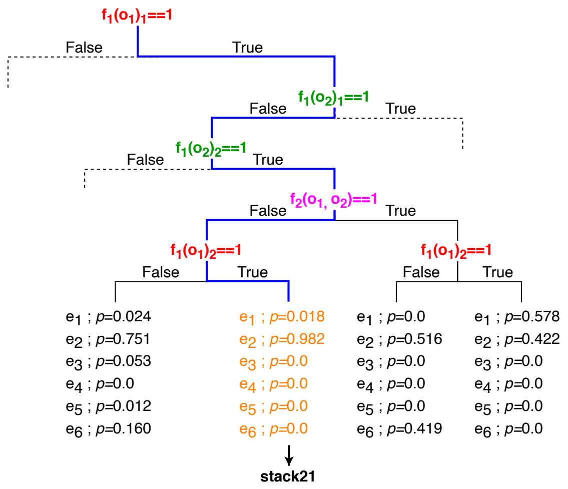
(a) Observed effects, $\mathcal{E}_{\text{paired}}$. (b) Found effect categories, $\mathcal{C}_{\text{paired}}$.

Figure 7: Effect space for the paired object interactions. Effects of the below object $(\Delta x_2, \Delta y_2, \Delta d_2)$ are omitted as they are almost zero. The low depth values are in blue and the high depth values are in red.

classes actually share similarities: in Figures 6b and 7b, some categories are more similar to each other than others. To reflect these similarities on the decision tree learning, we further cluster the found effect categories. We only consider those categories (that have been already found) which are close to each other as one. To be precise, we apply k-means clustering with k=5 to categories in Figures 6b and 7b. When we train the decision tree with clustered and non-clustered effect categories (labels), classification accuracies for predicting the correct effect category are 75.0 ± 4.6 and 42.1 ± 4.2 over 10 runs, respectively.

5.4 Learned Rules and PPDDL Operators

The result of decision tree learning is shown in Figure 8a where only a small number of branches out of 32 is explicitly shown because of the space constraints. Decision rules for



(a) One path of the decision tree is highlighted.

```
(:action stack21
    :parameters (?below ?above)
    :precondition (and (not (aux-height) (not (aux-instack))
                       (pickloc ?above) (stackloc ?below)
                       (f11 ?below) (f01 ?above)
                       (relation0 ?below ?above))
    :effect (and (probabilistic
                  0.018 (and (e1) (aux-height) (stackloc ?above)
                             (not (stackloc ?below)))
                  0.982 (and (e2) (aux-instack) (aux-height)
                             (stackloc ?above) (not (stackloc ?below)))
                  0.000 (e3)
                  0.000 (e4)
                  0.000 (e5)
                  0.000 (e6)
                 (not (pickloc ?above))))

(:action increaseheight1
    :precondition (and (aux-height) (H0))
    :effect (and (not (H0)) (H1) (not (aux-height))))

(:action increasestack1
    :precondition (and (aux-instack) (S0))
    :effect (and (not (S0)) (S1) (not (aux-instack))))

(:action makebase
    :parameters (?obj)
    :precondition (not (base))
    :effect (and (base) (aux-height) (aux-instack) (not (pickloc ?obj))
                 (stackloc ?obj)))
```

(b) Highlighted path is converted into PPDDL.

Figure 8: An example expansion of the decision tree. $(f_1(o_1)_1, f_1(o_1)_2)$ represents the category of the object above where $(f_2(o_2)_1, f_2(o_2)_2 f)$ represents the category of the object below. The relational variable is denoted as $f_2(o_1, o_2)$. On the leaves, each effect e_i is observed with the corresponding probability.

the highlighted branch is $(f_1(o_1)_1, f_1(o_1)_2, f_1(o_2)_1, f_1(o_2)_2, f_2(o_1, o_2)) = (1, 1, 0, 1, 0)$. Here, $(f_1(o_1)_1, f_1(o_1)_2)$ represents the category of the object above, $(f_1(o_2)_1, f_1(o_2)_2)$ represents the category of the object below, and $f_2(o_1, o_2)$ is the symbol for the paired-object relation. A natural-language translation of this branch is as follows: 'If the above object is rollable in all directions (0, 1), and the below object is pushable and insertable (1, 1), and the below object is not larger than the above object, e_2 is observed (which is the stacking effect found with clustering) with 0.982 probability'. PPDDL description corresponding to this branch of the tree is shown in Figure 8b.

For our experiments in the tower building task, we manually introduced some auxiliary predicates, as well as special actions for the domain to be able to chain multiple actions and to count the number of objects in the tower. These are needed to set a goal of constructing a tower with multiple objects which are outside the experience of the robot.

The `aux-instack` predicate is set to true for effect categories that satisfy $|\Delta d_1| > \epsilon_1$ for some small ϵ, and otherwise, `aux-height` predicate is set to true. Actions `increaseheight1` and `increasestack1` are treated as addition operators that increase the height of the tower (H) and the number of the objects in the stack (S), respectively. There are multiple H and S predicates ranging from H1-H7 and S1-S7, and likewise multiple `increaseheight` actions. When the `aux-height` effect is observed, the planner must select the `increaseheight1` action to continue with the plan. Therefore, when a stack effect is observed, the height of the tower (which is represented by H) increases automatically.

These would not be needed if we were able to represent effect clusters with real numbers instead of atomic representations since we could have summed the consecutive effects.

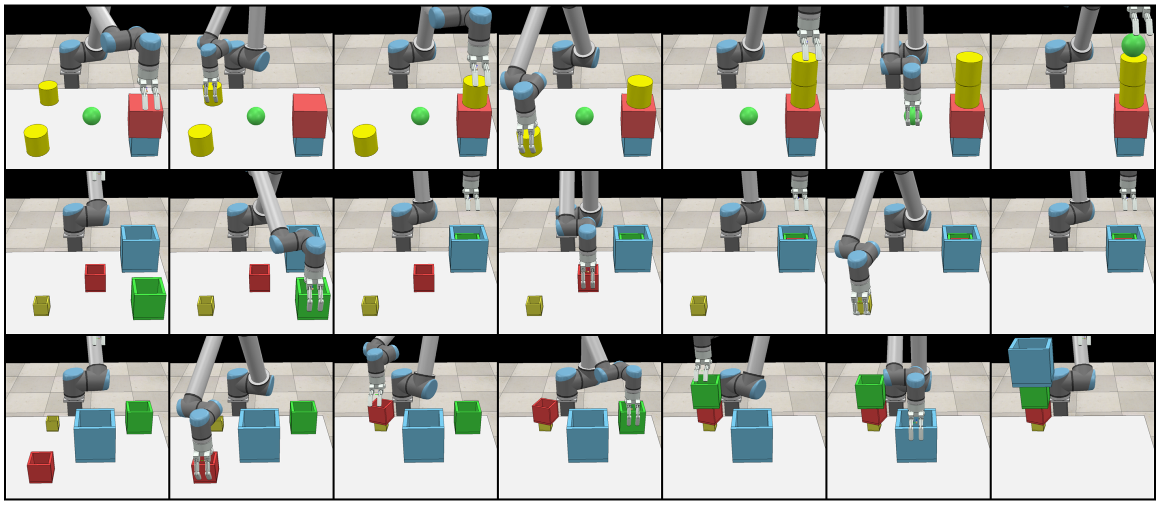


Figure 9: Top row: The objective is to construct a tower of height five using five objects, H5S5. The system assesses the success probability to be 0.67. Middle row: The objective is H1S4 and the system assesses the success probability to be p=0.14. Bottom row: If we change the objective to H4S4 the success probability increases to 0.94.

However, since we adopted mGPT (Bonet and Geffner, 2005) as our planner we went on with the discrete effect representation.

Lastly, the predicate `pickloc` is true for objects that are on the table and available for use for the tower construction; `stackloc` is true for the object that is at the top of the tower. These are shown in Figure 8b.

5.5 Performance of Planning

The autonomously-generated PPDDL description constructed by the discovered symbols and rules were verified by generating plans given a set of goals, executing these plans in the simulator, and assessing the success of the executed action sequences in achieving these goals. To be concrete, we asked the system to generate plans to create towers of desired heights with a given fixed set of objects. The challenge of the task is to place objects on top of each other in the correct order. With five objects, there are 5! plans. For plan generation, the mGPT off-the-shelf probabilistic planner (Bonet and Geffner, 2005) with FF heuristic (Hoffmann and Nebel, 2001) was used.

Since in our experiments, the system is asked to generate plans given a number of objects on the table, we encode the task of, say, “construct a tower with a height of three (H3) using four objects (S4)” as H3S4 (Figure 9).

5.5.1 DeepSym vs. OCEC

We first compare our system with the alternative OCEC system. We train both systems 10 times and select the best performing models (based on the decision tree accuracy). We initialized 20 random problems and asked the planner to solve the task using two different domain descriptions generated from different methods. We run the probabilistic planner

	Estimated prob.	Planning success	Execution success
DeepSym	69.8	95.0	65.0
OCEC	12.1	25.0	15.0

Table 2: Planning results from random 20 configurations.

Task	H4S4	H3S4	H2S4	H1S4
Est. prob.	0.96	0.81	0.63	0.24
Success	0.92	0.40	0.44	0.20
Recognition fail	0.04	0.32	0.04	0.0
Plan fail	0.04	0.16	0.0	0.0
Execution fail	0.0	0.12	0.52	0.80

Table 3: Planning results from 25 executions for each task. To satisfy the H1S4 objective, the robot needs to insert objects inside of each other which is more challenging compared to the other tower tasks, thus success probability is lower.

100 times for each problem and record the number of successes to get an estimate of the success probability of the plans. The results are reported in Table 2. We report two different metrics: (1) planning success shows whether the system generated a feasible plan or not and (2) execution success shows whether the execution is successful or not. The latter concerned with the stochasticity of the environment, not with the feasibility of the plan. We see that the OCEC model performs considerably worse than our approach, mainly caused by the wrong classification of the cup object (see Table 1 in Section 5.1).

5.5.2 DeepSym Performance

Now that we showed the performance gap between the two methods, we want to analyze when our method fails and succeeds. We considered four different goals (towers of heights from 1 to 4) and performed 25 different runs with random initial object configurations for each objective. We configured object types and sizes so that there is at least one feasible solution. For example, for the H1 objective, we make sure that there are at least three cups that can be physically stacked into each other. The initial state and the final state of each problem are provided in Appendix B. The plan execution performance is reported in Table 3. There are four different outcomes: (1) the plan executed successfully, (2) the plan fails due to a recognition error, (3) the plan fails due to incorrectly assessed probabilities of rules, therefore the generated plan is not feasible, (4) the plan fails at execution time due to the stochasticity of the environment.

We see that the robot constructs towers with a height of four successfully. As the height of the tower decreases, the robot needs to insert some of the objects inside other cups. The insertion task is harder than the stacking task due to the stochasticity of the environment, which is also reflected in the estimated probabilities in Figure 8b; even if the below cup is larger than the above sphere, the insertion probability is 0.578. For example, for the challenging objective of creating an H1 tower including all objects, the system estimates the success probability to be 0.24. Accordingly, 20% of executions are successful. This shows that the system can faithfully model the probabilistic nature of the environment. Example executions are shown in Figure 9.

5.5.3 Deterministic vs. Probabilistic Planning

We also experimented with deterministic planning instead of probabilistic one. To do so, while converting rules to PDDL, we take the maximum likely effect as the generated effect. Thus, the deterministic planning eliminates the possibility of other effects, and therefore effectively eliminates possible solutions. In some configurations, deterministic planning cannot even find a feasible plan (10 out of 50 runs) whereas the probabilistic one can. Overall, the adoption of probabilistic planners is almost a must for real-world or highly stochastic environments as the possibility of overlooking feasible plans must be minimized.

6. Conclusion

In this work, we introduced a method that discovers effect and action guided object categories, encodes them as discrete symbols, and learns rules that predict action effects. It sustains a general cognitive development progression where symbols are formed, rules are learned, planning is achieved, and verified in execution. Our system contributes to the state-of-the-art by showing the following desirable properties which have not been achieved/shown simultaneously elsewhere:

- A generic, single pipeline neural solution for mapping raw sensorimotor experience into the symbolic domain.
- The proposed network allows progressive learning of increasingly complex abstractions, exploiting previously-learned abstractions as inputs.
- It is gradient-friendly so can be incorporated in any gradient-based machine learning system for more complex processing.
- When compared with the continuous bottleneck layer version of our system, i.e. OCEC, our system performs better in effect category formation leading to more successful action planning. This suggests that instead of post-training clustering of the continuous unit outputs, employing discrete units from the beginning is crucial.

In future work, we plan to scale up the system by augmenting the perceptual capabilities and the action repertoire of the robot. The ad-hoc perceptual system for determining action effects can be replaced by a state-of-the-art computer vision system. Beyond paired-object relations, graph neural networks can be employed to construct relations between varying numbers of objects. Applying the principles of learning and abstraction of this work to less-constrained scenarios will constitute a major step towards AI-enabled, general-purpose robots.

Appendix A. Network Architecture and Hyperparameters

The network architectures of encoders f_1 and f_2 are shown in Tables 4a and 4b, respectively. Each convolution is followed by a batch normalization layer and ReLU activation after the normalization. The network architectures of decoders g_1 and g_2 are shown in Tables 5a and 5b, respectively. Decoders consist of fully-connected (FC) layers with no batch normalization.

Layer	In ch.	Out ch.	Stride	Pad
Conv3x3	1	32	1	1
Conv3x3	32	32	2	1
Conv3x3	32	64	1	1
Conv3x3	64	64	2	1
Conv3x3	64	128	1	1
Conv3x3	128	128	2	1
Conv3x3	128	256	1	1
Conv3x3	256	256	2	1
Global average pooling over channels				
FC	256	2	-	-
Binary activation				
Number of parameters: 1,174,114				

(a) Encoder f_1.

Layer	In ch.	Out ch.	Stride	Pad
Conv3x3	2	32	1	1
Conv3x3	32	32	2	1
Conv3x3	32	64	1	1
Conv3x3	64	64	2	1
Conv3x3	64	128	1	1
Conv3x3	128	128	2	1
Conv3x3	128	256	1	1
Conv3x3	256	256	2	1
Global average pooling over channels				
FC	256	1	-	-
Binary activation				
Number of parameters: 1,174,145				

(b) Encoder f_2.

Layer	Input units	Output units
FC+ReLU	5	128
FC+ReLU	128	128
FC	128	3
Number of parameters: 17,667		

(a) Decoder g_1.

Layer	Input units	Output units
FC+ReLU	5	128
FC+ReLU	128	128
FC	128	6
Number of parameters: 18,054		

(b) Decoder g_2.

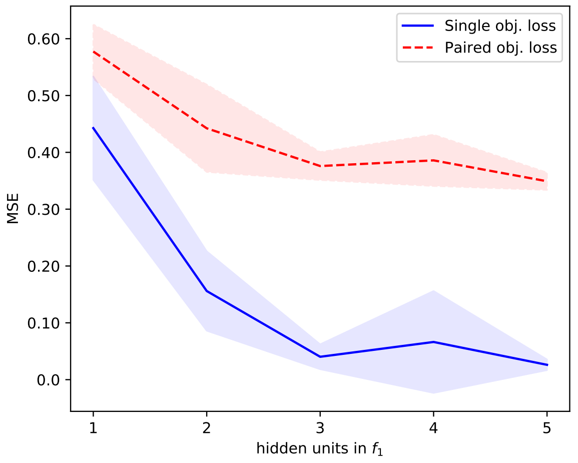
Adam optimizer (Kingma and Ba, 2014) with AMSGrad (Reddi et al., 2019) is used. The learning rate is set to 0.00005 with 128 batch size. Each model is trained for 300 epochs and we select the best model based on the mean square error.

While finding the number of hidden units, we take 5 runs and record the MSE. We increase the number of units if the one-sided Welch's t-test rejects the null hypothesis $\mathcal{H}_0$: "Two numbers result in the same MSE" in favor of $\mathcal{H}_1$: "Increased number results in lower MSE".

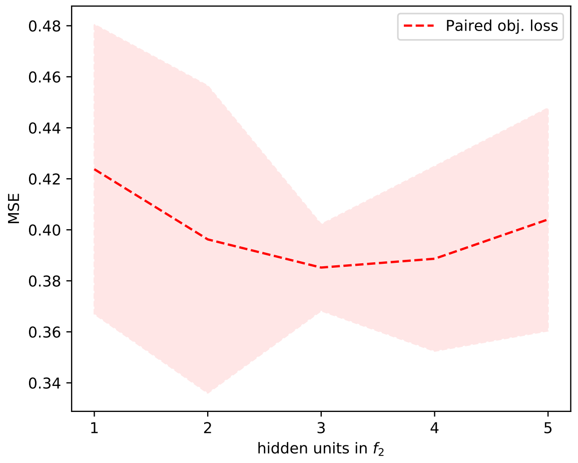
We realize that this requires multiple runs, and in fact, is quite inefficient. It would be better if we had a well-defined metric such as the Bayesian Information Criterion (BIC). We did not use BIC since it does not lead to plausible results with deep neural networks that have large parameter size.

Appendix B. Plans

Generated plans for different goals are shown in Figures 11 and 12. Successes are framed in green and fails are framed in red.



(a) MSE vs the number of units in the bottleneck of f_1.



(b) MSE vs the number of units in the bottleneck of f_2.

Figure 10: The mean square error losses for f_1-g_1 and f_2-g_2 network pairs.

References

Akbulut, T., Oztop, E., Seker, Y., Xue, H., Tekden, A., and Ugur, E. (2020). ACNMP: Flexible skill formation with learning from demonstration and reinforcement learning via representation sharing. In *Conference on Robot Learning (CoRL)*.

Asai, M. and Fukunaga, A. (2017). Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. *arXiv preprint arXiv:1705.00154*.

Asai, M. and Muise, C. (2020). Learning neural-symbolic descriptive planning models via cube-space priors: The voyage home (to strips). *arXiv preprint arXiv:2004.12850*.

Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

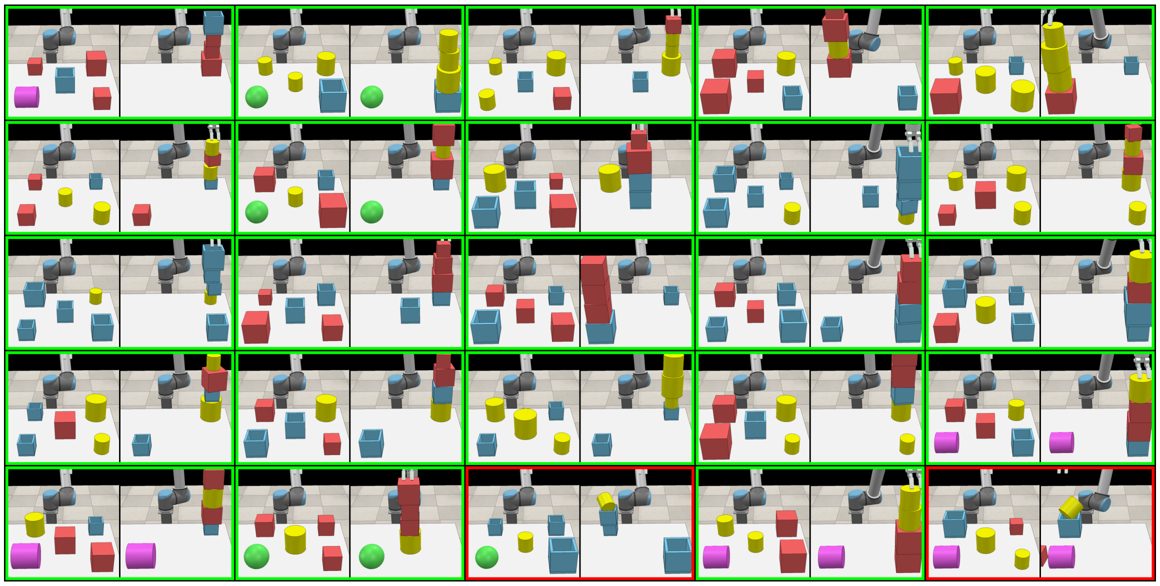
Bonet, B. and Geffner, H. (2005). mgpt: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research*, 24:933–944.

Callaghan, T. and Corbit, J. (2015). The development of symbolic representation. *Handbook of Child Psychology and Developmental Science*, pages 1–46.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*.

Gibson, J. J. (2014). *The ecological approach to visual perception: classic edition*. Psychology Press.

Harnad, S. (1990). The symbol grounding problem. *Physica D*, 42(1-2):335–346.


(a) Tower with height 4 using 4 objects (H4S4).

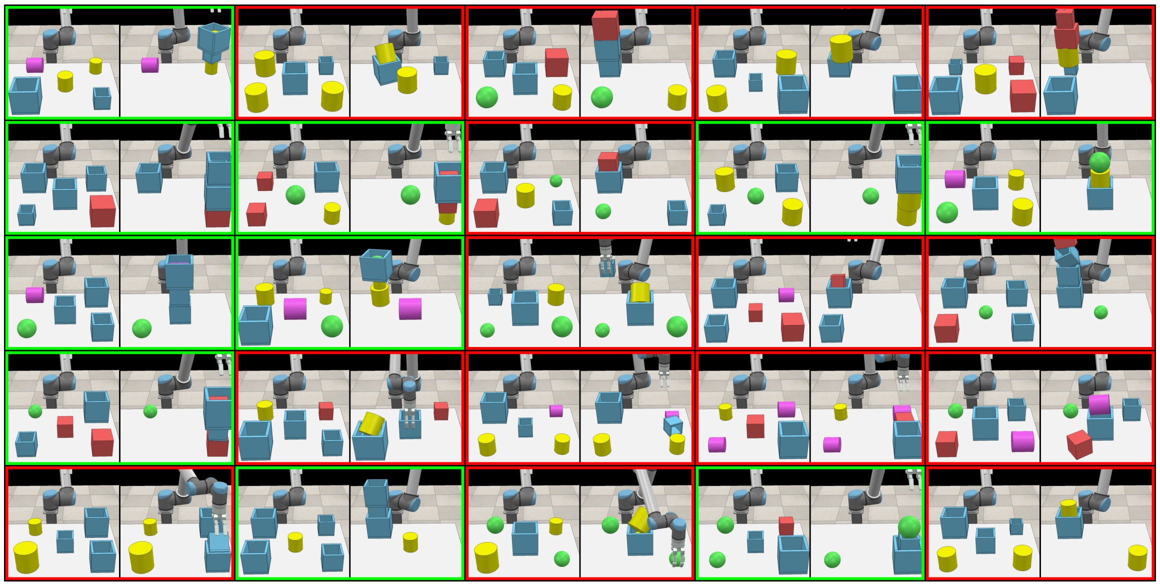
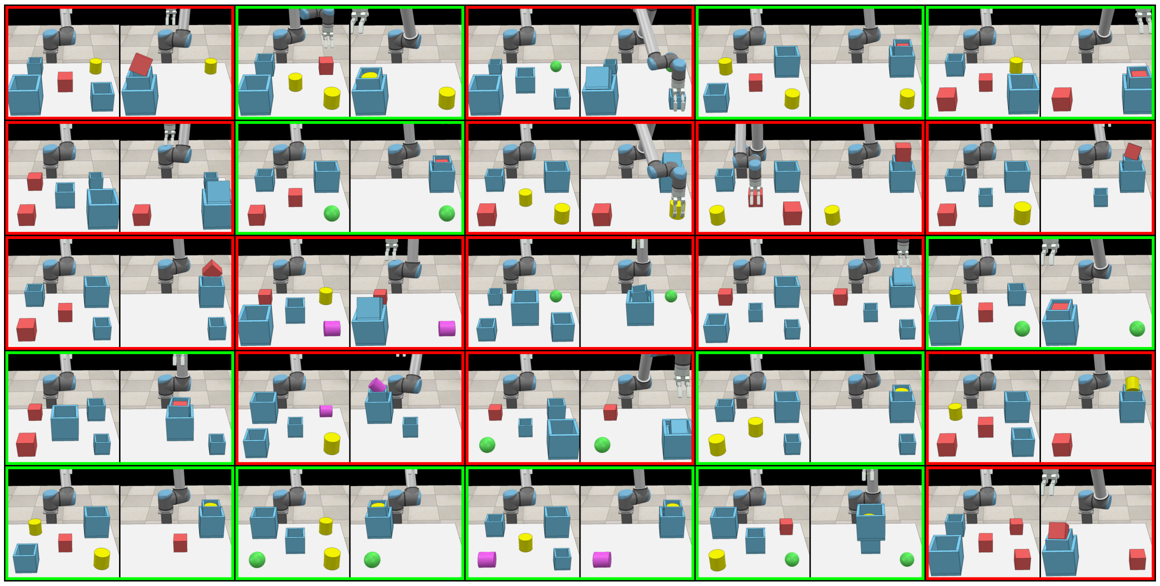

(b) Tower with height 3 using 4 objects (H3S4).

Figure 11: Plan executions.


(a) Tower with height 2 using 4 objects (H2S4).

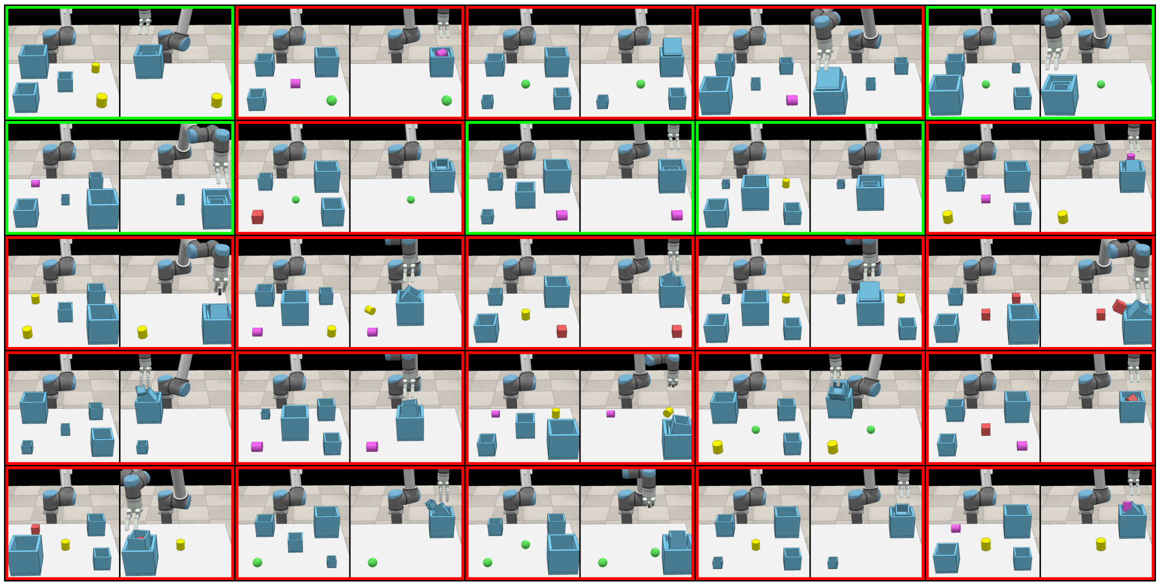

(b) Tower with height 1 using 4 objects (H1S4).

Figure 12: Plan executions.

Hoffmann, J. and Nebel, B. (2001). The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

James, S., Rosman, B., and Konidaris, G. (2019). Learning portable representations for high-level planning. *arXiv preprint arXiv:1905.12006*.

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

Klingspor, V., Morik, K., and Rieger, A. D. (1996). Learning concepts from sensor data of a mobile robot. *Machine Learning*, 23(2-3):305–332.

Konidaris, G. (2019). On the necessity of abstraction. *Current Opinion in Behavioral Sciences*, 29:1–7.

Konidaris, G., Kaelbling, L., and Lozano-Perez, T. (2015). Symbol acquisition for probabilistic high-level planning. In *International Joint Conference on Artificial Intelligence*.

Konidaris, G., Kaelbling, L. P., and Lozano-Perez, T. (2014). Constructing symbolic representations for high-level planning. In *28th AAAI Conf. on AI*.

Konidaris, G., Kaelbling, L. P., and Lozano-Perez, T. (2018). From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289.

Kuipers, B., Feigenbaum, E. A., Hart, P. E., and Nilsson, N. J. (2017). Shakey: From conception to history. *AI Magazine*, 38(1):88–103.

Kulick, J., Toussaint, M., Lang, T., and Lopes, M. (2013). Active learning for teaching a robot grounded relational symbols. In *Proc. 23rd Int. Joint. Conf. AI*, pages 1451–1457. AAAI Press.

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the Trade*, pages 9–48. Springer.

Mourao, K., Petrick, R. P., and Steedman, M. (2008). Using kernel perceptrons to learn action effects for planning. In *CogSys*, pages 45–50.

Murphy, R. and Murphy, R. R. (2000). *Introduction to AI Robotics*. MIT press.

Ozturkcu, O. B., Ugur, E., and Oztop, E. (2020). High-level representations through unconstrained sensorimotor learning. In *International Conference on Development and Learning (ICDL)*.

Petrick, R., Kraft, D., Mourao, K., Pugeault, N., Krüger, N., and Steedman, M. (2008). Representation and integration: Combining robot control, high-level planning, and action learning. In *Proceedings of the 6th International Cognitive Robotics Workshop*, pages 32–41.

Pisokas, J. and Nehmzow, U. (2005). Experiments in subsymbolic action planning with mobile robots. In *Adaptive Agents and Multi-Agent Systems II, Lecture Notes in AI*, pages 80–87. Springer.

Reddi, S. J., Kale, S., and Kumar, S. (2019). On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*.

Rohmer, E., Singh, S. P. N., and Freese, M. (2013). Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. www.coppeliarobotics.com.

Russell, S. J. and Norvig, P. (2009). *Artificial Intelligence: a modern approach*. Pearson, 3 edition.

Seker, M. Y., Imre, M., Piater, J., and Ugur, E. (2019). Conditional neural movement primitives. In *Proceedings of Robotics: Science and Systems (RSS)*, Freiburgim, Germany.

Sun, R. (2000). Symbol grounding: A new look at an old idea. *Philosophical Psychology*, 13(149–172).

Taniguchi, T., Ugur, E., Hoffmann, M., Jamone, L., Nagai, T., Rosman, B., Matsuka, T., Iwahashi, N., Oztop, E., Piater, J., et al. (2019). Symbol emergence in cognitive developmental systems: a survey. *IEEE Transactions on Cognitive and Developmental Systems*.

Townsend, W. (2000). The barretthand grasper–programmably flexible part handling and assembly. *Industrial Robot: An International Journal*.

Ugur, E., Oztop, E., and Sahin, E. (2011). Goal emulation and planning in perceptual space using learned affordances. *Robotics and Autonomous Systems*, 59(7–8):580–595.

Ugur, E. and Piater, J. (2015a). Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2627–2633. IEEE.

Ugur, E. and Piater, J. (2015b). Refining discovered symbols with multi-step interaction experience. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 1007–1012. IEEE.

Ugur, E., Sahin, E., and Oztop, E. (2012). Self-discovery of motor primitives and learning grasp affordances. In *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3260–3267.

Universal Robots (2012). The UR10 collaborative industrial robot. https://www.universal-robots.com/products/ur10-robot. Online; accessed 10 September 2020.

Werner, H. and Kaplan, B. (1963). *Symbol formation*. Wiley.

Wörgötter, F., Agostini, A., Krüger, N., Shylo, N., and Porr, B. (2009). Cognitive agents: A procedural perspective relying on the predictability of Object-Action-Complexes OACs. *Robotics and Autonomous Systems*, 57(4):420–432.

Younes, H. L. and Littman, M. L. (2004). Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*.

Zech, P., Haller, S., Lakani, S. R., Ridge, B., Ugur, E., and Piater, J. (2017). Computational models of affordance in robotics: a taxonomy and systematic classification. *Adaptive Behavior*, 25(5):235–271.