# Learning Symbolic Representations for Planning with Parameterized Skills

Barrett Ames[1,3], Allison Thackston[3], George Konidaris[2]

***Abstract*— A critical capability required for generally intelligent robot behavior is the ability to sequence motor skills to reach a goal. This requires a (typically abstract) representation that supports goal-directed planning, which raises the question of how to construct such a representation. Previous work has addressed this question in the context of simple black-box motor skills, which are insufficiently flexible to support the wide range of behavior required of intelligent robots. We now extend that work to include *parametrized motor skills*, where a robot must both select an action to execute and also decide how to parametrize it. We show how to construct a representation suitable for planning with parametrized motor skills, and specify conditions which are sufficient to separate the selection of motor skills from the parametrization of those skills. Our method results in a simple discrete abstract representation for planning followed by a parameter selection process that operates on a fixed plan. We first demonstrate learning this representation in a virtual domain based on Angry Birds and then learn an abstract symbolic representation for a robot manipulation task.**

## I. Introduction

A defining characteristic of intelligent robots is the ability to generate goal-oriented behavior for a wide variety of tasks. One approach to generating such behavior is task-level planning, whereby a robot reasons about sequences of predefined motor skills to find a sequence that reaches a goal with high probability. Task level planning is fast and capable of considering relatively long planning horizons. However, it depends on the availability of a composable set of motor skills, and an abstract representation that supports efficiently reasoning about composing the motor skills.

Previous work [15] has addressed the question of how to construct abstract representations suitable for planning with a given collection of motor skills, providing a principled method for constructing representations that support the operations necessary for probabilistic planning. However, that work assumed that the motor skills are simple black-box controllers. Due to the continuous nature of certain tasks, such controllers are too inflexible to support the range of dexterous behaviors required of intelligent robots that must interact with a complex world.

Existing approaches for generating more flexible behavior find either detailed low-level motion plans [16], which is generally only feasible for short-horizon tasks or use higher-level actions but still reason in detail [10], which performs well but still requires complex planning to find medium-horizon plans. We propose an alternative approach that plans using more flexible *parametrized motor skills* [4] [13] [14] [17] where the behavior of each high-level action is modified by a real-valued parameter vector. Here, the robot must select both the motor skill to execute at a particular time and additionally select appropriate parameters for it.

We extend existing work on constructing abstract representations [15] to support parametrized motor skills and specify conditions which separate the selection of motor skills from the parametrization of those skills. This extension results in a relatively simple discrete abstract representation for planning, which is followed by a fixed parameter selection process.

We demonstrate the effectiveness of our framework by first learning a symbolic representation for a virtual domain based on Angry Birds and then create a symbolic representation for a robot manipulation task.

## II. Background

### A. Parameterized Skills

A parameterized skill is a motor controller which takes as input a parameter vector and generates behavior that varies based on the parameter's value. We describe the robot's task as a parametrized action Markov decision process (PAMDP) [17], described by a tuple:

$$(S, A, R, T, \gamma),$$

where $S \subseteq \mathbb{R}^n$ is a set of states; $A$ is a set of parameterized skills (described next); $R$ is a reward function (equivalently, a cost function) describing the reward received for executing a parametrized action in a particular state; $T$ is a model of the transition dynamics; and $\gamma$ is a discount factor.

Following the *options framework* [23], each parameterized skill is described by a tuple:

$$(\pi_o, \Theta_o, I_o, \beta_o),$$

where $\pi_o(a|s, \theta)$ is a policy that returns the probability of the agent executing action $a$ in state $s$, given input parameter $\theta$; $\Theta_o \subseteq \mathbb{R}^m$ is a range of acceptable policy parameter vectors; $I_o$ is the set of states from which the motor skill can be executed;[1] and $\beta_o(s)$ is a termination condition, which determines the probability with which the motor skill should terminate in a given state $s$.

[1]Duke University Computer Science, cbames at cs.duke.edu
[2]Brown University Computer Science, gdk at cs.brown.edu
[3]Toyota Research Institute, allison.thackston at tri.global

[1]Note that we assume that any parameter value $\theta \in \Theta_o$ can be selected for any state in $I_o$; we leave removing this assumption to future work.

In a MDP, the agent aims to finds a policy, $\pi$, which maps from states to a distribution over actions. In a PAMDP, the agent must find a policy which maps from states to a distribution over, $(a, \theta)$, where $a$ is a parametrized action and $\theta$ is its parametrization.

Masson et al. [17] introduced the use of parametrized action Markov Decision Processes to include parameterized skills in the reinforcement learning setting. They present a reinforcement learning algorithm that learns separate policies for selecting skills and skill parameters, given a state. This method was successfully applied to a simulated robot soccer domain but is unsuitable for use on physical robots as it is model-free, and consequently requires a great deal of experience to learn a good policy.

### B. Planning with Motor Skills

If we are to construct a representation that enables a robot to plan using a set of motor skills, it must support the operations required to compute the probability with which a plan succeeds, and the expected reward (equivalently, cost) received should execution be successful. Prior work [15] has shown that the agent must represent two important pieces of information for each motor controller: the *precondition*, $\text{Pre}(o, s) = P(s \in I_o)$, which estimates the probability that a motor skill $o$ can be run from state $s$, and the *image* $\text{Im}(o, Z)$, which returns a distribution over states, expressing the probability of the agent entering state $s'$ after executing motor skill $o$ from a start state drawn from distribution $Z$. Given these, the robot can compute the probability of a plan succeeding by multiplying the probability of successfully executing each action:

$$p_i = \int \text{Pre}(o_i, s) Z_i(s) ds,$$

where $Z_0$ is the start distribution and each $Z_i = \text{Im}(o_{i-1}, Z_{i-1})$. Similarly, we can compute the expected reward of executing action $o_i$ from distribution $Z_i$ as $r_i = \int R(s, o_i) Z_i(s) ds$.

A symbol is an abstraction applied to preconditions and images. Here we differentiate between two types of symbols. A *distributional symbol*, $\sigma_z$, names a distribution, Z, over states. Distributional symbols are used to estimate the state distributions of images. A *conditional symbol*, $\sigma_E$, is the name associated with a function $P(C(s) = True)$ which returns the probability that condition $C$ holds true at state $s$. Conditional symbols are probabilistic classifiers used to approximate the probability that a state belongs to the precondition of a skill.

A symbolic representation for motor skill planning can be constructed by first creating a pair of symbols for each skill. There is a minimum of at least one conditional symbol to represent the skill's precondition and one distributional symbol to represent the skill's image. These pairs of symbols are used to compute every reachable $p_i$ and $r_i$ value as defined above, and finally used to construct a forward model for obtaining $p_i$ and $r_i$ using only the symbols themselves (and not the classifiers and distributions that they name). However, the ability to do so requires that there be only finitely many such $p_i$ and $r_i$ values, which in turn means that there can be only finitely many reachable image distributions.

This condition is not true in general, but it does hold for some common types of motor skills. The most important one is the *subgoal skill*, where a feedback motor controller drives the state toward some subgoal set of states, and the feedback process eliminates any dependency on the state from which execution started. In that case we can replace any instance of the image operator with an *effect distribution* which does not depend on start state:

$$\text{Im}(o, Z) = \text{Eff}(o).$$

As a result, there are only finitely many reachable image distributions—in fact, exactly as many as there are motor controllers—this allows the construction of a finite representation.

Of course, in many real-life scenarios, the subgoal property does not hold. In practice, we can often *partition* a motor skill's initiation set such that the subgoal property approximately holds when execution only occurs from each of the resulting partitions. In this case, we can build a finite model by considering each partition a distinct motor skill.

## III. Building Symbolic Models for Probabilistic Planning with Parametrized Skills

Due to the uncertain nature of robot sensing and action we are interested in *probabilistic planning*. Probabilistic planning is the process of creating a plan and evaluating its probability of success. Formally, a plan is defined as a sequence of actions intended to accomplish a goal from a particular starting configuration. In order to perform probabilistic planning with parameterized skills, we must extend the previous definition to handle motor skill parameters and thus define a plan as follows:

**Definition 1.** A plan $p = \{o_1(\theta_1)....o_n(\theta_n)\}$ from a start state distribution Z is a sequence of parameterized skills and their parameters $o \in O, \theta \in \Theta_o$, to be executed from a state drawn from Z.

This plan definition maintains the important aspects of the previous definition. The plan is still a sequence of actions and dependent on the start state distribution, however each plan also includes the parameter for each action. It is important to note that each motor skill has a bounded range of parameters from which it can draw. For example, $o_1$ may accept a parameter value between 0 and 5, while $o_2$ may accept a parameter value between 25 and 100. The start state distribution, Z, is required because without a start state the probability of plan completion cannot be evaluated. As before, the probability of plan completion is obtained by computing the probability of successfully completing each successive motor skill execution in the plan, and multiplying these probabilities together.

Konidaris et al. [15] prove that the precondition and image are necessary and sufficient for symbolic planning of unparameterized motor skills. Here we assume that the

preconditions of parameterized motor skills have no dependency on the parameter, and thus are the same as previously defined. However, the image of a parameterized motor skill is dependent on the parameter and must be redefined. We therefore define the *parameterized probabilistic image operator*.

**Definition 2.**

$$\begin{aligned}\text{Image}(Z, o, \theta) = & \quad \frac{\int_S P(s'|s, o, \theta) Z(s) P(s \in I_o) ds}{\int_S Z(s) P(s \in I_o) ds} \\ = & \quad P(s'|o, \theta).\end{aligned}$$

The image is important for calculating the probability of skill $j$ being executable after skill $i$ has completed. The probability of the image of skill $i$ overlapping with the initiation set of skill $j$ is the probability of *skill intersection*:

**Definition 3.**

$$\text{Inter}(o_i, \theta_i, o_j, Z) = \quad \int_S \text{Image}(Z, o_i, \theta_i) P(s \in I_{o_j}) ds. \quad (1)$$

These definitions allow us to now prove that the probability of plan completion can be constructed from these parts.

**Theorem 1.** Given a PAMDP, a model of the probabilistic precondition of each parameterized skill and the parameterized probabilistic image, the probability of completing a plan, ($Z$,p), can be determined.

*Proof*

Consider an arbitrary plan tuple $(Z_0, p)$, with plan length $n$. The probability of successfully executing plan $p$ from starting distribution, $Z_0$, is

$$\prod_{j=0}^{n-1} \text{Inter}(o_j, \theta_j, o_{j+1}, Z_j) \int_S Z_0 P(s \in I_{o_0}) ds.$$

Where $Z_{j+1}$ can be found via $Z_{j+1} = \text{Image}(Z_j, o_j, \theta_j)$, for $j \in \{1...n\}$. □

Starting with a distribution of states, the agent can repeatedly apply the image operator to obtain the distribution of states after taking an action. Using these images the agent can determine the probability of execution of each parameterized skill. It can then multiply all of these probabilities together to obtain the probability of success of a plan. However, we are interested in finding the best plan given the motor skills available to the robot, not only the probability of success of a particular plan. Thus we define the *Skill Optimal Plan*:

**Definition 4.**

$$\arg\max_{\theta_0 ... \theta_n} \prod_{j=0}^{n-1} \text{Inter}(o_j, \theta_j, o_{j+1}, Z_j) \int_S Z_0 P(s \in I_{o_0}) ds. \quad (2)$$

This is the set of motor skill parameters which maximize a given plan's probability of success. In order to maximize the probability of a plan's success, the planner must select a sequence of skills and for each skill the parameter which maximizes the probability of that skill. In Figure 1 the initiation classifier is represented on the left and the parameterized image of the skill is the yellow ribbon on the right. The planner must select a $\theta$ which results in the maximum probability that the image of the skill is in the initiation classifier of the next skill. By selecting the maximum probability parameter setting the representation for a skill optimal plan remains discrete, as we prove next.

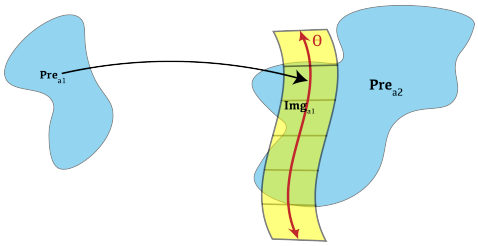



Fig. 1: The yellow ribbon is the set of all image distributions. The image varies with the value of $\theta$ along the red curve. The goal is to select a range of $\theta$ such that the effect distribution lies inside the precondition of the next action with highest probability. Visually, this would be regions where the yellow ribbon is contained by the blue precondition.

**Theorem 2.** Assuming that only the probability of a successful plan execution is of interest and given $n$ parameterized motor skills, the number of symbols required to represent a skill optimal plan is $\mathcal{O}(n^2)$.

*Proof*

For each skill intersection the only parameter setting which needs to be represented by a symbol is one which maximizes the probability of skill intersection. All other parameter settings will result in a lower probability and thus can be ignored without impacting the planner's ability to find a skill optimal plan. This is because a skill optimal plan must be constructed of skills with parameters tuned to maximize the probability of the following skill. This is a consequence of the multiplicative nature of our plan definition. Lastly, there are $n^2$ skill intersections, each with one parameter setting which maximize the probability thus a discrete representation with $\mathcal{O}(n^2)$ symbols can represent all necessary information.

□

Now that the number of symbols has been bounded, a parameter must be selected. In practice we must search for the $\theta$ which maximizes the probability of $\text{Inter}(o_j, \theta_j, o_i, Z)$ for all $o \in O$. This is achieved by uniformly sampling over the range of $\theta$ and estimating the intersection probability for each sample. This approach will always incur some error between the estimated max probability and the true max probability, $|\tilde{m} - m|$. Next we bound this approximation error when the intersection is assumed to be Lipschitz continuous. In order for this assumption to be reasonable the perception

and dynamics of the robot need to be Lipschitz. Many mobile manipulation platforms have dynamics which are Lipschitz continuous. As for perception, in many cases the environment being observed can be treated as Lipschitz by linearly interpolating between any discontinuities.

**Lemma 1.** Given a Lipschitz continuous skill intersection, with Lipschitz constant K, and the distance between parameter sample values, $\Delta$, and $d$ the dimensionality of the parameter space, the approximation error of the maximum is $\frac{\sqrt{d}K\Delta}{2}$.

*Proof*

First, uniformly sample the parameter space of $\theta$ by $\Delta$. For all of these points find the probability of intersection. Next, define $\tilde{m}$ as the max taken over all of the skill intersection samples and let $m$ be the true max. The approximation error is defined as $|\tilde{m} - m|$. Given an $\tilde{m}$, an $m$ can be constructed that does not change the selection of $\tilde{m}$ and maximizes the approximation error. Starting at $\tilde{m}$ increase at a rate of $K$ for $\frac{\sqrt{d}\Delta}{2}$ while heading in the direction of a point that is $\sqrt{d}\Delta$ away. At $\frac{\sqrt{d}\Delta}{2}$ change the rate to $-K$. At $\sqrt{d}\Delta$ from its start point the function will be equal to $\tilde{m}$. The point $\sqrt{d}\Delta$away is the furthest point on the hypercube that is defined by $\Delta$ spaced uniform sampling. Thus the furthest $m$ can be from $\tilde{m}$ is $\frac{\sqrt{d}\Delta}{2}$.

□

Using this approximate max operator we now bound the error of an approximate finite symbolic representation for achieving the skill optimal plan.

**Theorem 3.** Given a finite number of parameterized motor skills with Lipschitz continuous skill intersections, a finite number of bounded parameters for each motor skill, and the approximate max operator, the error of an approximate finite symbolic representation is upper bounded by $\frac{\sqrt{d}K\Delta n}{2}$, where $n$ is the plan length.

*Proof*

This proof is excluded for brevity, but it is an induction over the length of the plan.

The error due to the approximation grows as the plan gets longer, and as the intersection becomes more sensitive to the change in motor skill parameters. In addition, the bound explains how $\Delta$ is related to these two quantities. This relationship shows that we can sample less (i.e. have a large value for $\Delta$) for cases where plans are short, or parameter sensitivity is low.

Unfortunately, even in cases where $\Delta$ can be large, calculating the skill intersection in general is computationally expensive. Konidaris et al. [15] use the subgoal property to approximate the intersection. Here we generalize the subgoal property to include parameter values, we propose the *strong parameterized subgoal property*:

**Definition 5.**

$$\text{Image}(Z, o, \theta) = \text{Effect}_{\theta_l, \theta_u}(o) \quad \forall \theta, \theta_l \leq \theta \leq \theta_u.$$

This property holds when the image and starting distribution are conditionally independent given the parameters of the controller. If this property holds over the entire range of $\theta$, then the motor skill is not parameterized. In general, this property is expected to only hold over segments of the parameter space. When this property does hold, an effect defined over a parameter range can approximate the image in (1). In practice, this property may be overly restrictive, as it requires the image and effect to be the same distribution. Thus we introduce the *weak parameterized subgoal property:*

**Definition 6.**

$$\int_s \text{Image}(Z_i, o_i, \theta_i)(s) P(s \in I_{o_j}) ds = \int_s \text{Effect}_{\theta_l, \theta_u}(o_i)(s) P(s \in I_{o_j}) ds \quad \forall \theta_i, \theta_l \leq \theta_i \leq \theta_u, \forall o_j \in O.$$

This ensures that the probability of executing any following action is necessarily the same, but does not require that the two distributions are the same. Since the effect is a set of images the number of intersections that must be calculated is reduced. In addition, computationally friendly distributions can be chosen as long as they satisfy the subgoal property. The downside of using the effect is that it may not represent the set of images well, unless the weak parameterized subgoal property holds. Thus the combination of Theorem 3 and the weak parameterized subgoal property results in a discrete representation which supports efficient planning and has bounded error from the optimal.

## IV. Experiments

We applied our framework to two different domains: a virtual domain, which demonstrates the compression and expressiveness of the representation, and a robot manipulation task, which demonstrates its performance under sparse and noisy data.

### A. Catapult Domain

The catapult domain is a model of the popular game, Angry Birds (see Figure 4). In this instantiation, three walls can be knocked down, and two obstacles that only fall when their corresponding wall falls. The agent is provided with one parameterized behavior, $shootAtAngle(\theta)$. The parameter, $\theta$, defines the firing angle of the catapult and the goal of the agent is to knock down all the walls.

*1) Data Collection:* An agent uniformly at random selects $\theta$ from 0 to 1.57 until 10,000 parameterized skill executions have been collected. One consequence of the random agent is that in some cases a wall can be perturbed without knocking it over. These samples provide additional information about how the agent can knock down a wall that is not in the canonical start pose. The state space consists of the action

(a) Symbol 0 (b) Symbol 5 (c) Symbol 1 (d) Symbol 6

Fig. 2: Symbols used for constructing the operator in Figure 3. In a) there is a symbol defined over the X position of the first wall, b) a symbol defined over the Y position and $\phi$ of the first wall, c) a symbol defined over the X position of the first wall and has a different mean then (a), d) a symbol defined over the Y position and $\phi$ of the first wall, which has a different mean than (b).

parameter used, $\theta$, and the x,y, and $\phi$ of each wall before and after the action. $\phi$ is the angle, from vertical, of a wall.

*2) Symbol Creation:* In order to prepare the data for symbol creation, the data was clustered using DBSCAN [8, 19] with an $\epsilon = 0.32$ and the minimum number of samples set to 100. Using randomized logistic regression feature selection was performed on each cluster. A feature was kept if it appeared in more than 55% of 500 random re-samplings of the training data. The precondition classifiers were trained on the clustered data using only the selected features. The classifiers used were Probabilistic KSVMs [5].[2] The effect distributions were modeled with conditional kernel density estimators [18].

Each effect distribution is checked against the other effect distributions using a 2-sample KolmogorovSmirnov-test to ensure that the distributions are significantly different. If they are not, their grounding sets are merged into a single symbol.

*3) Operator Creation:* The clustered data was used to create a list of observed cluster transitions. Clusters which occurred before an action require a conditional symbol. Clusters which occurred after an action require a distributional symbol. In this case because there is only one option, all of the operators come from partitioning the initiation set of the option. The probability of the operator follows the methods outlined in Section III.

*4) Planning:* The learned operators are transformed into the Probabilistic Planning Domain Description Language (PPDDL) [25] so that off-the-shelf planners can be used. During this transformation any operator with a probability less than 0.25 was downgraded to impossible and removed. The start state and goal are defined in PPDDL and the problem is posed to the mini-gpt [2] planner, an off-the-shelf MDP planner, with two heuristics applied, zero, and min-min-lrtdp.

*5) Results:* Only 16 symbols were created for the virtual domain. This is a significant reduction in representation size over even a coarse discretization of the combined action and state space.[3] This symbolic space allows very efficient planning, a plan for knocking down all of the walls can be found in 4.5ms.

A common trade-off for increasing performance is to decrease the expressiveness of a representation, however

[2]The radial basis kernel was used.

[3]For example, if each dimension had be discretized into 10 buckets, the resulting matrix would have $10^{10}$ elements.

```
(:action a184
 :precondition (and (symbol_0) (symbol_5))
 :effect (probabilistic 0.96 (and
 (symbol_1)(symbol_6)
 (symbol_9)(symbol_13)
 (not(symbol_2))(not(symbol_8))
 (not(symbol_11))(not(symbol_0))
 (not(symbol_10))(not(symbol_4))
 (not(symbol_7))(not(symbol_15))
 (not(symbol_12))(not(symbol_14))
 (not(symbol_5))(not(symbol_3)))))
```

Fig. 3: A symbolic operator for knocking down wall 1.

for the learned symbols there is enough expressiveness to accomplish multiple tasks. The symbols allow the planner to handle not just narrowly defined start states, but any state which is represented by the symbols. This is demonstrated by Figure 4(a) where the initial start state has been sampled from one of the symbols. Figures 4(b)-(d) demonstrate that the planner is capable of planning from different start states. Similarly, new goal states can be specified without requiring a change in representation. In Figure 4(e) the goal has been changed from knocking down the walls to moving the first wall against the first obstacle. The planner is able to select a $\theta$ which accomplishes this goal even under initial state uncertainty.

A symbolic operator is shown in Figure 3. This is one of the operators which knocks down the first wall. Its preconditions require that the wall is in its canonical start pose. Symbol 0 is a distribution defined over the X-position of the first wall and is visible in Figure 2(a). Symbol 5 is in Figure 2(b) and is a distribution defined over the y and $\phi$ of the wall. Thus the precondition symbols are interpreted as the first wall being upright and in a specific X-position. The effects of the operator are listed after ":effect". The form "probabilistic 0.96" says that with probability p=0.96 the following effects will be set. There are four symbols set to true by this operator, two of which are over the state of the second wall, and two of which are over the first wall. For the sake of clarity we will ignore Symbols 13 and 9, and focus on symbols 1 and 6 which describe the new state of wall 1. Symbol 1 is Figure 2(c), thus the wall will have an X-position before the first obstacle. Symbol 6 is Figure 2(d), which shows the wall will be laying down. The rest of the symbols become false because they conflict with one of the positive symbols. For example, it is impossible for the wall

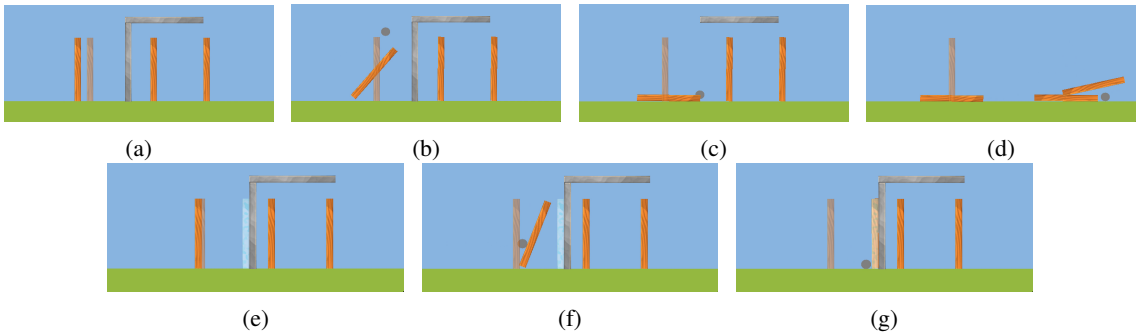



Fig. 4: These images depict the execution of two plans for two different goals from the same set of symbols. The first goal is to knock down all the walls (a-d). a.) The transparent wood wall is the canonical start pose. The first wall's position is now sampled from a start state distribution. b.) Just after impact. c.) After impact. d.) All walls down. The second goal is to move the first wall without knocking it down (e-g). e.) The translucent blue wall is the target location. f.) Moving the wall g.) The wall has been successfully moved from its starting position to its new goal.

to be in the state described by (symbol 0, symbol 5) and the state described by (symbol 1, symbol 6).

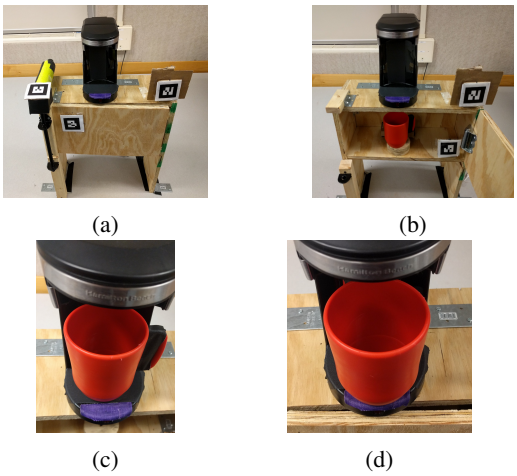



Fig. 5: The experimental setup: a.) the cupboard is closed and unlocked, b.) the cupboard is open c.) the cup is in the correct position, d.) the cup is blocking the button.

### B. Robot Manipulation Task

In this experiment our mobile manipulator, Anathema, was used to perform an idealized form of the "Make Coffee" task (Figure 5). In this instance of the task, the coffee cup is in a locked cabinet. The cabinet can be opened by rotating a handle to a specific angle and pulling up on the handle. In addition, the coffee cup must be placed in the coffee machine within a specific angular range. If the cup is placed in the coffee machine with handle inwards the cup completely obscures the button (Figure 5(d)), preventing the robot from pressing the button. Lastly, the goal is to have the cup in the coffee machine while the button is pressed.[4]

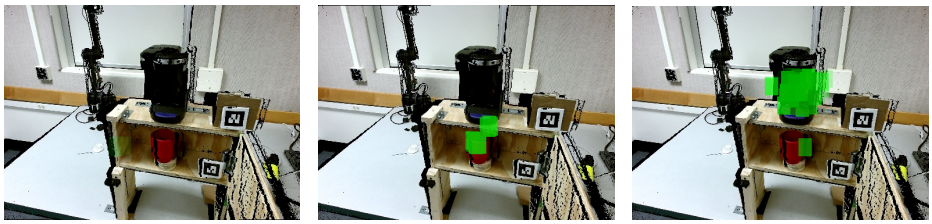
Anathema has four behaviors that can be used for this task $PickPlaceCup(\theta)$, $UnlockCabinet(\phi)$, $OpenCabinet()$ and $PressButton()$. $PickPlaceCup(\theta)$, approaches the cup, picks it up, and then places it in the coffee machine. It accepts one parameter $\theta$, which is the approach angle of the hand relative to the cup. $UnlockCabinet(\phi)$ grasps the cabinet handle, rotates it, and then lets go of the handle. It also accepts one parameter, $\phi$, which is the angle of the handle relative to its starting position. $OpenCabinet()$ grasps the handle and attempts to pull it up. $PressButton()$ approaches the coffee machine from above and attempts to press the coffee machine button.

*1) Methods:* The behaviors in this experiment use MoveIt! [22], AprilTags [24], and the cmvision [3] blob detector. The state space is the position of the cup in image space, the area of the cup, the position of the button in image space, the visible area of the button, the locations of 4 AprilTags, and whether the coffee button is pressed. In order to determine whether the button was pressed, the coffee machine was outfit with a Raspberry Pi and a momentary switch which turned on when the button was depressed. ROS [21] was used to provide communication between all of the various components.

*2) Symbol Creation:* For primitive skills, the state space was clustered using DBSCAN with $\epsilon = 0.9$ and a minimum of 5 samples. For parameterized skills regression clustering was used. Feature selection was performed by univariate feature selection, the highest scoring four features, according to ANOVA F-value, were kept. Conditional symbols were modeled by SVMs with radial basis functions fit using grid search. Distributional symbols were modeled by conditional kernel density estimators. $\theta$ probability clustering was performed using xgboost [6] with 100 trees and $\alpha = 1$. Data was collected by performing 34 trials of $ButtonPress()$, 44 trials of $OpenCabinet()$, 40 trials of $PickPlaceCup(\theta)$, 54 trials of $UnlockCabinet(\phi)$, and 30 no-ops. Lastly, planning with the symbols was performed using mini-gpt, with two heuristics applied, zero, and min-min-lrtdp.

*3) Results:* An example of symbols created for this domain are illustrated in Figure 6. Symbol 8 is a distribution over X and Y positions of the cup which were viable

[4]In order to avoid mixing liquids and robots, no coffee is actually made in this task.



(a) Symbol 7 (b) Symbol 8 (c) Symbol 9

Fig. 6: Symbols learned from robot execution for cup position. a.) Symbol 7 has a low probability of cup visibility. b.) Symbol 8 is a distribution over cup pickup positions. c.) Symbol 9 is the $PressButton()$ precondition distribution.

locations for pickup. This distribution is tight because the hand went to the same position, regardless of actual cup position. Thus if the cup was outside of this distribution the gripper would completely miss it. Symbol 9 is the cup position distribution from which $PressButton()$ could be executed and the goal state reached. The highest density of positions is away from the button, which makes sense because the button must be uncovered in order for it to be pressed. Overall the symbols capture where the cup needs to be for certain operators to take place.

Figure 7 and the included video demonstrate that the robot is capable of using the constructed symbols to create a plan which performs the desired task. The plan was constructed in 0.22s and consisted of four symbolic operators. They were, in order, `a35376`, `a18763`, `a19284`, `a5247`. These symbolic operators resolved to $UnlockCabinet(66 - 72)$, $OpenCabinet()$, $PickPlaceCup(-15 - 12)$, $ButtonPress()$ with their parameters uniformly at random selected from the provided ranges. In order to better understand how the symbols are representing the state space, we will follow how the cup position distribution changes throughout the plan. First, `a35376` requires that the environment be in a configuration that could be sampled from symbol 7, i.e. that no cup is visible. This action has no impact on cup position. Thus the second action, `a18763`, also expects the environment to be sampled from symbol 7. The result of `a18763` is expected to be a cup position that could be sampled from symbol 8. Next, `a19284` expects the cup to be sampled from symbol 8, and places the cup in a position that could be sampled from symbol 9. This enables the last action, `a5247`, which presses the button and completes the task. This symbolic plan demonstrates that a continuous task can be accomplished using parameterized behaviors that were parameterized ahead of time via a discrete planner.

## V. Related Work

Others have addressed the issue of combining low level actions and high level planning. Previous approaches can be split into two main categories. The first category is approaches which stitch together several configuration space representations of tasks and attempt to find a motion plan in this combined space [10][1]. Unfortunately, this may create portions of the combined space which have volume zero and thus are difficult to sample from but necessary to pass through in order to complete the task. In addition, these methods are not able to leverage advances in symbolic planning. The other category uses abstractions of low level actions to search for a solution by leveraging symbolic planners. Within this category there are methods which impose a symbolic representation [7][12][20] and those that construct a symbolic representation from data. Our work is most closely related to the symbol creation methods. Gaudioso et al. [9] use Answer Set Programming to create abstractions but their approach is limited to discrete state and action spaces. The closest to our work is Jetchev et al. [11] they build a symbolic abstraction which maximizes reward for a relational reinforcement learning problem. However, they construct a predicate for each object, whereas our predicate is applied to all of the feature data that is available, nor do they address the complexity of parameterized motor skills.

## VI. Conclusion

In this paper we explored a symbolic representation for parameterized skill planning. First we proved that the probability of a plan can be calculated if it is constructed of parameterized skills that obey the parameterized subgoal property. Next we proved that a discrete representation can always be used to achieve the skill optimal plan. Then the compressive and expressive qualities of the representation were explored in the Catapult domain. We showed that with only 16 symbols the planner was able to solve the original task and other related tasks. Finally, the performance of the representation was evaluated on a robot manipulation task. The planner was able to perform the task and approximate the probability of the plan successfully. The presented representation is a step towards flexible goal-directed abstract planning for robots.

## ACKNOWLEDGMENTS

The authors would like to thank Ron Parr for his thoughtful and incisive discussions. This research was supported in part by DARPA under agreement number D15AP00104 and by ONR under award N00014-17-1-2699. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The content is solely the responsibility of the authors and does not necessarily represent the official views of DARPA. Barrett Ames was supported by a NDSEG Fellowship.

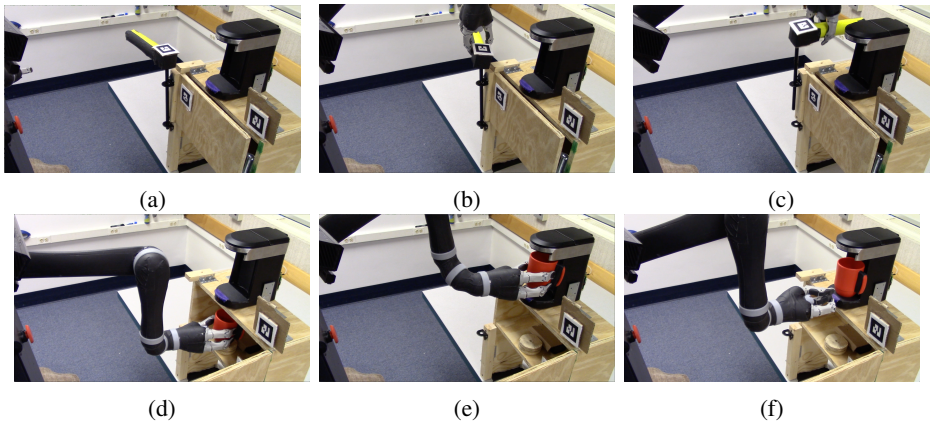


Fig. 7: Execution of the symbolic plan: a.) the environment at start, b.) unlocking the cabinet door, c.) opening the cabinet door, d.) picking up the coffee cup, e.) placing the coffee cup, and f.) pressing the coffee machine button.

## REFERENCES


[1] J. Barry, L. Kaelbling, and T. Lozano-Pérez. A Hierarchical Approach to Manipulation with Diverse Actions. *International Conference on Robotics and Automation*, 2013.

[2] B. Bonet and H. Geffner. MGPT: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research*, 2005.

[3] J. Bruce, T. Balch, and M. Veloso. Fast and Inexpensive Color Image Segmentation for Interactive Robots. *International Conference on Intelligent Robots and Systems*, 2000.

[4] B. Castro, D. Silva, G. Konidaris, and A. Barto. Active Learning of Parameterized Skills. In *International Conference on Machine Learning*, pages 1737–1745, 2014.

[5] C. Chang and C. Lin. LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, 2011.

[6] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[7] N. Dantam, Z. Kingston, S. Chaudhuri, and L. Kavraki. An Incremental Constraint-Based Framework for Task and Motion Planning. *The International Journal of Robotics Research*, 2018.

[8] M. Daszykowski and B. Walczak. Density-Based Clustering Methods. In *Comprehensive Chemometrics*. 2010.

[9] G. Gaudioso, M. Leonetti, and P. Stone. State Aggregation through Reasoning in Answer Set Programming. *IJCAI Workshop on Autonomous Mobile Service Robots*, 2016.

[10] K. Hauser and J. Latombe. Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries. *International Conference on Automated Planning and Scheduling*, 2009.

[11] N. Jetchev, T. Lang, and M. Toussaint. Learning Grounded Relational Symbols from Continuous Data for Abstract Reasoning. *International Conference on Robotics and Automation*, 2013.

[12] L Kaelbling and T Lozano-Pérez. Hierarchical task and motion planning in the now. In *IEEE International Conference on Robotics and Automation*, pages 1470–1477, 2011.

[13] L. Kaelbling and T. Lozano-Pérez. Learning composable models of parameterized skills. In *International Conference on Robotics and Automation*, Singapore, 2017.

[14] J. Kober and J. Peters. Policy Search for Motor Primitives in Robotics. *Machine Learning*, 2011.

[15] G. Konidaris, L. Kaelbling, and T. Lozano-Perez. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

[16] J. Kuffner and S. LaValle. RRT-connect: An efficient approach to single-query path planning. In *International Conference on Robotics and Automation.*, 2000.

[17] W. Masson, P. Ranchod, and G. Konidaris. Reinforcement Learning with Parameterized Actions. *Association for the Advancement of Artificial Intelligence*, 2016.

[18] T. O'Brien, K. Kashinath, N. Cavanaugh, W. Collins, and J. O'Brien. A fast and objective multidimensional kernel density estimation method: FastKDE. *Computational Statistics and Data Analysis*, 2016.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. 2012.

[20] E. Plaku and G. Hager. Sampling-based Motion and Symbolic Action Planning with Geometric and Differential Constraints. *International Conference on Robotics and Automation*, 2010.

[21] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. *International Conference on Robotics and Automation*, 2009.

[22] I. Sucan and S. Chitta. MoveIt! URL `http://moveit.ros.org`.

[23] R. Sutton, D. Precup, and S Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[24] J. Wang and E. Olson. AprilTag 2: Efficient and robust fiducial detection. In *International Conference on Intelligent Robots and Systems*, 2016.

[25] H. Younes and M. Littman. PPDDL1.0: An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects. *Techn. Rep. CMU-CS-04-162*, 2004.