CORE
Metadata, citation and similar papers at core.ac.uk
Provided by DSpace@MIT

Symbol Acquisition for Probabilistic High-Level Planning

George Konidaris[†] **Leslie Pack Kaelbling**[‡] **Tomas Lozano-Perez**[‡]
Duke University[†] MIT CSAIL[‡]
Durham NC 27708 Cambridge MA 02139
gdk@cs.duke.edu {lpk, tlp}@csail.mit.edu

Abstract

We introduce a framework that enables an agent to autonomously learn its own symbolic representation of a low-level, continuous environment. Propositional symbols are formalized as names for probability distributions, providing a natural means of dealing with uncertain representations and probabilistic plans. We determine the symbols that are sufficient for computing the probability with which a plan will succeed, and demonstrate the acquisition of a symbolic representation in a computer game domain.

Introduction

There is a long history of research in intelligent robotics that combines high-level planning with low-level control [Nilsson, 1984; Malcolm and Smithers, 1990; Gat, 1998; Cambon *et al.*, 2009; Choi and Amir, 2009; Dornhege *et al.*, 2009; Wolfe *et al.*, 2010; Kaelbling and Lozano-Pérez, 2011]. Such systems are capable of generating complex, goal-driven behavior but are hard to design because they require a difficult integration of symbolic reasoning and low-level motor control.

Recently, Konidaris *et al.* [2014] showed how to automatically construct a symbolic representation suitable for planning in a high-dimensional, continuous domain. This work modeled the low-level domain as a semi-Markov decision process (SMDP) and formalized a propositional symbol as the name given to a grounding set of low-level states (represented compactly using a classifier). Their key result was that the symbols required to determine the feasibility of a plan are directly determined by characteristics of the actions available to an agent. This close relationship removes the need to hand-design symbolic representations of the world and enables an agent to, in principle, acquire them autonomously.

However, a set-based symbol formulation cannot deal with learned sets that may not be exactly correct, and can only determine whether or not the probability of successfully executing a plan is 1. These restrictions are ill-suited to the real-world, where learning necessarily results in uncertainty and all plans have some probability of failure.

We introduce a probabilistic reformulation of symbolic representations capable of naturally dealing with uncertain representations and probabilistic plans. This is achieved by moving from sets and logical operations to probability distributions and probabilistic operations. We use this framework to design an agent that autonomously learns a completely symbolic representation of a computer game domain, enabling very fast planning using an off-the-shelf probabilistic planner.

Background

Semi-Markov Decision Processes

We assume that the low-level sensor and actuator space of the agent can be described as a fully observable, continuous-state semi-Markov decision process, described by a tuple $M = (S, O, R, P, \gamma)$, where $S \subseteq \mathcal{R}^n$ is the n-dimensional continuous state space; $O(s)$ is a finite set of temporally extended actions, or *options* [Sutton *et al.*, 1999], available in state $s \in S$; $R(s', \tau | s, o)$ is the reward received when executing option $o \in O(s)$ at state $s \in S$ and arriving in state $s' \in S$ after τ time steps; $P(s', \tau | s, o)$ is a PDF describing the probability of arriving in state $s' \in S$, τ time steps after executing option $o \in O(s)$ in state $s \in S$; and $\gamma \in (0, 1]$ is a discount factor.

An option o consists of three components: an *option policy*, π_o, which is executed when the option is invoked; an *initiation set*, $I_o = \{s | o \in O(s)\}$, which describes the states in which the option may be executed; and a *termination condition*, $\beta_o(s) \rightarrow [0, 1]$, which describes the probability that option execution terminates upon reaching state s. The combination of initiation set, reward model, and transition model for an option o is known as o's *option model*. We assume that the agent does not have access to its option models, and can only observe whether its current state is in I_o and the transitions resulting from actually executing o.

Probabilistic High-Level Planning

High-level planning approaches operate using symbolic states and actions. The simplest formalism for high-level planning is the *set-theoretic representation* [Ghallab *et al.*, 2004]. A typical formalization of the probabilistic version of this [Younes and Littman, 2004] describes a planning domain as a set of propositional symbols $\mathcal{P} = \{p_1, ..., p_n\}$ and a set of

actions $\mathcal{A} = \{\alpha_1, ..., \alpha_m\}$. A state $\mathcal{P}_t$ at time t assigns a truth value $\mathcal{P}_t(i)$ to every $p_i \in \mathcal{P}$, and so can be represented by a binary vector.

Each action α_i is a tuple describing a precondition and a set of possible outcomes, along with the probability of each occurring: $\alpha_i = \left(\text{precond}_i, \left\{(\rho_1, \text{effect}^+_{i1}, \text{effect}^-_{i1}), ..., (\rho_k, \text{effect}^+_{ik}, \text{effect}^-_{ik})\right\}\right)$, where $\text{precond}_i \subseteq \mathcal{P}$ lists the propositions that must be true in a state for the action to be applicable, each $\rho_j \in [0, 1]$ is an outcome probability such that $\sum_{j=1}^{k} \rho_j = 1$, and effect^+_{ij} and effect^-_{ij} are the positive (propositions set to be true) and negative (propositions set to be false) effects of outcome j occurring, respectively. All other propositions retain their values. A planning problem is obtained by additionally specifying a start state, s_0, and set of goal states, S_g. The planner is typically tasked with finding a sequence of actions that leads from s_0 to some state in S_g with high probability.

Symbols for Planning

When a high-level planning formalism is used to solve a low-level problem, each proposition can be considered to evaluate to true or false at each low-level state. The propositions can thus be viewed as referring to (or naming) the set of low-level states in which the proposition holds (i.e., evaluates to true). Consequently, Konidaris *et al.* [2014] used the following definition of a symbol:

Definition 1. *A propositional symbol σ_Z is the name associated with a test τ_Z, and the corresponding set of states $Z = \{s \in S \mid \tau_Z(s) = 1\}$.*

The test, or *grounding classifier*, is a compact representation of a set of infinitely many continuous, low-level states. High-level planning can then be performed using set operations over the grounding classifiers (corresponding to logical operations over the symbols). Defining a symbol for each option's initiation set, and the symbols necessary to compute its image (the set of states the agent might be in after executing the option from any of a given set of starting states) are necessary and sufficient for planning [Konidaris *et al.*, 2014]. The feasibility of a plan is evaluated by computing each successive option's image, and then testing whether it is a subset of the next option's initiation set. This process is depicted in Figure 1.

Konidaris *et al.* [2014] defined two option classes (subgoal and abstract subgoal options) for which computing the image simply required one symbol for each option's effects set (the set of states the agent could find itself in after executing the option from any state). The grounding classifiers for the resulting symbols are sufficiently well defined that an agent can gather labeled training data by executing the options and observing the results. In principle, this enables the agent to acquire its own symbolic representation solely through interaction with the environment.

However, the resulting learned classifiers will be difficult to plan with in real domains, for three reasons. First, this formalism cannot account for the uncertainty inherent in *learning* the symbols themselves. Instead, planning proceeds as if the agent's estimate of each of its grounding classifiers is exactly correct. Second, it cannot model the fact that some transitions

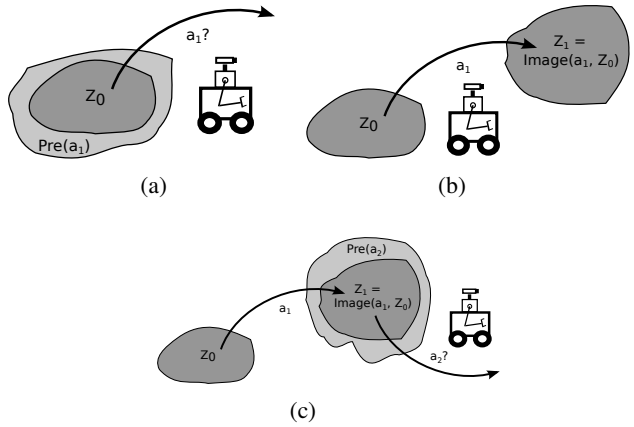


Figure 1: Determining whether a plan consisting of two actions, a_1 and a_2, can be executed from state set Z_0. a_1 can be executed if and only if Z_0 is a subset of a_1's precondition (a). If so, the agent computes Z_1, the set of states it may reach after executing a_1 from some state in Z_0 (b). a_2 can be executed if and only if Z_1 is a subset of a_2's precondition set (c). This procedure can be performed for any plan and starting set.

are more likely than others—effectively assuming that all transitions with a non-zero probability of occurring are equivalently important. Consequently, it cannot be used to reason about expected reward, since expectation requires a probability distribution. Finally, it can only determine whether a plan can certainly be executed, or not—it cannot determine the probability with which it can be executed. These restrictions make it unlikely that a set-based framework can be applied to real problems.

Probabilistic Symbols and Plans

We now describe a probabilistic formulation of symbolic representations that allows an agent to reason about *the probability that a plan can be executed* using *uncertain symbols* (such as are obtained during learning), and compute the *expected reward* of executing a plan. This requires generalizing the idea of a *symbol as a set* to that of *a symbol as a probability distribution*. Instead of referring to a grounding set via a grounding classifier, a symbol will now refer to a *grounding distribution*. There are two senses in which this will be useful, leading to two types of symbols, referred to as types 1 and 2.

In the first sense, a set is a collection of equally likely states, which we can generalize to a probability distribution over states:

Definition 2. *A probabilistic symbol σ_Z of type 1 is the name associated with a distribution, $Z(S)$, over states.*

Symbols of type 1 will be useful in expressing a distribution over start states and for representing an option's image. During learning, the task of estimating the grounding distribution of a type 1 symbol is that of *density estimation*, which is a well-studied unsupervised learning problem.

In the second sense, a set is a collection of states in which some condition holds. We generalize this to a distribution expressing the probability that a condition holds in each state:

Definition 3. *A probabilistic symbol σ_E of type 2 is the name associated with the probability $P(C(s) = 1)$ of some condition C holding at every state $s \in S$.*

Symbols of type 2 are *probabilistic classifiers*, giving the probability that a condition holds for every state in S. Since the agent operates in an SMDP, a condition either holds at each state or it does not—but the agent must necessarily generalize when learning from data, and the probabilistic classifier expresses the agent's uncertainty about the condition in states it has not yet encountered.

We now generalize the notion of a plan from one that starts from a set of states to one that starts from a distribution over states (a probabilistic symbol of type 1):

Definition 4. *A probabilistic plan $p = \{o_1, ..., o_{p_n}\}$ from a start state symbol σ_Z (corresponding to state distribution $Z(S)$) is a sequence of options $o_i \in O$, $1 \leq i \leq p_n$, to be executed from a state drawn from $Z(S)$.*

We can also define the corresponding *plan space*—the set of all plans the agent should be able to evaluate.

Definition 5. *The* probabilistic plan space *for an SMDP is the set of all tuples (σ_Z, p), where σ_Z is a* start symbol *and p is a plan.*

The essential function of a symbolic representation for probabilistic planning is to compute, on demand, the probability that an agent can execute any element of the plan space to completion, and the expected reward received for successfully doing so. We now introduce probabilistic symbols that are provably sufficient for doing so by defining probabilistic versions of the precondition symbol and image operator. The probabilistic precondition symbol expresses the probability that an option can be executed from each state:

Definition 6. *The* probabilistic precondition *is a probabilistic symbol of type 2, defined as Pre(o) = $P(s \in I_o)$.*

Given a distribution over s (a probabilistic symbol of type 1) representing a distribution over start states, we now require a probabilistic image operator that computes an output distribution (also of type 1) over s', representing the distribution over states in which the agent expects to find itself in after option execution.

Definition 7. *Given a start distribution $Z(S)$ and an option o, we define the probabilistic image of o from $Z(S)$ as:*

$$Im(o, Z) = \frac{\int_S P(s'|s, o)Z(s)P(I_o|s)\,\mathrm{d}s}{\int_S Z(s)P(I_o|s)\,\mathrm{d}s},$$

where $P(s'|s, o) = \int P(s', \tau|s, o)\,\mathrm{d}\tau$, since we are not concerned with the time taken to execute o.

These generalizations of the precondition set and image operator allow us to prove the probabilistic generalization of Theorem 1 from Konidaris *et al.* [2014]:

Theorem 1. *Given an SMDP, the ability to represent the probabilistic preconditions of each option and to compute the probabilistic image operator is sufficient to determine the probability of being able to execute any probabilistic plan tuple (σ_Z, p).*

Proof. Consider an arbitrary plan tuple (σ_Z, p), with plan length n. To determine the probability of executing p from σ_Z, we can set $Z_0 = Z$ and repeatedly compute $Z_{j+1} = \text{Im}(p_j, Z_j)$, for $j \in \{1, ..., n\}$. The probability of being able to execute the plan is given by $\Pi_{j=1}^{n} \left[\int_S \text{Pre}(o_j, s)Z_{j-1}(s)\,\mathrm{d}s\right]$. □

Computation proceeds as follows. Starting with an initial distribution over states, the agent repeatedly applies the image operator to obtain the distribution over low-level states it expects to find itself in after executing each option in the plan. It can thereby compute the probability of being able to execute each successive option, and multiply these to compute the probability of successfully executing the entire plan.

To obtain the expected reward of the plan, the agent requires one additional operator:

Definition 8. *Given a start distribution $Z(S)$ and an option o, the reward operator $J(o, Z) = \int_S \int_S \int_{\mathbb{R}^+} P(s', \tau|s, o)R(s', \tau|s, o)Z(s)\,\mathrm{d}\tau\,\mathrm{d}s'\,\mathrm{d}s$.*

The expected reward of a plan tuple (σ_{Z_0}, p) of length n is then $\sum_{i=1}^{n} J(p_i, Z_{i-1})$, where each state distribution $Z_i(S)$ is defined as in Theorem 1. Although the definition of the reward operator involves three integrals, during learning we use the following equation:

$$J(o, s) = \mathbb{E}_{s',\tau}\left[R(s', \tau|s, o)\right],$$

which simply estimates the expected reward for executing an option from each state. The reward obtained after option execution from a state is a sample of the right hand side of this equation, resulting in a standard supervised learning problem. Computing the expected reward now requires integration over the distribution over start states only.

The symbol and operators described above are defined in terms of option models. One approach to planning in SMDPs is to learn the option models themselves, and use them to perform sample-based planning. Representing the operators directly as defined above may allow more efficient planning using analytical methods for computing the image operator. However, when the characteristics of the available options support it, the agent can go further and construct completely symbolic models it can use to plan—after which its grounding distributions are no longer required.

Subgoal Options and Abstract Subgoal Options

As in the deterministic planning case, computing the image operator for a subgoal option is particularly straightforward. A subgoal option in the probabilistic setting satisfies:

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

for all distributions over states $X(S)$. Eff(o) is the *effects distribution*—the distribution over states after executing o from any start distribution. This means that the image distribution is the same regardless of the start distribution—a stronger condition than in the set-based case, where the set of states that the agent has non-zero probability of arriving in cannot depend on the start set, but the probability of arriving in each state can.

This independence assumption models the case where a feedback controller or learned policy guides the agent to a

specific target distribution before completing option execution. It drastically simplifies planning because the image operator always evaluates to the effects distribution, and may be a reasonable approximation in practice even when it does not strictly hold.

If an agent has only subgoal options, a very simple abstract data structure suffices for planning. Given a collection of options and their effects distributions, define an abstract plan graph G as follows. Each option o_i has corresponding vertex v_i, with edges e_{ij} from v_i to v_j with reward $J(o_j, \text{Eff}(o_i))$ and probability of success $\int_S \text{Pre}(o_j, s)\ \text{Eff}(o_i)(s)\,\mathrm{d}s$. After computing G the agent can discard its grounding distributions and evaluate the reward and probability of a plan succeeding by adding the rewards and multiplying the probabilities along the corresponding path in the graph.

Abstract subgoal options model the more general case where the state vector can be partitioned into two parts $s = [a, b]$, such that executing o leaves the agent in state $s' = [a, b']$; the m feature values in $a \in A \subseteq R^m$ are left unchanged, and the state features values in $b \in B \subseteq R^{n-m}$ (o's *mask*) are set to a subgoal distribution $P(b')$ independent of the starting distribution. In this case:

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

where $P(a) = \int_B Z([a, b])\,\mathrm{d}b$ (integrating b out of $Z(S)$) and $\text{Eff}(o)$ is a distribution over the variables in b'.

In some cases an (abstract) subgoal option can be best modeled using multiple *effect outcomes*, where executing the option leads to one of a number of distinct (abstract) subgoals, each with some probability. The effect outcome distributions can be modeled as a single mixture distribution if they all have the same mask; if not, they must be modeled individually. In such cases an option o_i is characterized by a single precondition $\text{Pre}(o_i)$ and a set of effect outcomes $\text{Eff}^j(o_i)$, each with an associated mask and probability ρ_j of occurrence.

The options available to an agent may in some cases not satisfy the (abstract) subgoal property, but can be *partitioned* into a finite number options that do. For example, consider an agent with an option for moving through a corridor. This is not a subgoal option in a building with multiple corridors, but if the agent partitions the building into a finite number of corridors and considers the option executed in each to be distinct, then the subgoal property may hold for each partitioned option. Identifying such partitions is a major challenge when learning a probabilistic symbolic representation in practice. Fortunately, partitioning reflects a statistical measure of independence—a partition should be made when the effect distribution is not independent of the start distribution, but *is* independent conditioned on assignment to a partition.

Generating a Symbolic Domain Description

We now show that an agent can build a probabilistic STRIPS-like representation given a collection of abstract subgoal options. Our target is PPDDL, the *Probabilistic Planning and Domain Definition Language* [Younes and Littman, 2004], which can serve as input to an off-the-shelf planner.

Reasoning about the soundness and completeness of a symbolic representation requires a *grounding scheme* that specifies the semantics of that symbolic representation. Recall that a STRIPS-like representation is based on a collection of propositional symbols $\mathcal{P} = \{p_1, ..., p_n\}$. We choose to use probabilistic symbols of type 1, so that each p_i symbol refers to a distribution over states: p_i has associated grounding distribution $\mathcal{G}(p_i)$. We then define the grounding distribution of an abstract state as the multiplication of the grounding distributions of the propositions set to true in that state:

$$\mathcal{G}(\mathcal{P}_t) = \Pi_{i \in I} \mathcal{G}(p_i),\ I = \{i | \mathcal{P}_t(i) = 1\}.$$

The abstract state $\mathcal{P}_t$ can therefore be interpreted as representing the distribution of states in which the agent expects to find itself in at time t.

Finding Factors To proceed, the agent must identify the appropriate *factors*, similarly to Konidaris *et al.* [2014]—given the function modifies(s_i) (the list of options that modify a low-level SMDP state variable), the agent groups state variables modified by the same list of options together into factors. It can then define the functions factors(o_i) (the list of factors affected by executing o_i) and factors(σ_i) (the list of factors that probability distribution σ_i is defined over).

Building the Symbol Set We define a natural notion of independent factors—those that are statistically independent in the joint distribution $\text{Eff}(o_i)$:

Definition 9. *Factor f_s is independent in effect distribution Eff(o_i) iff* $\text{Eff}(o_i) = \left[\int_{f_s} \text{Eff}(o_i)\mathrm{d}f_s\right] \times \left[\int_{\bar{f}_s} \text{Eff}(o_i)\mathrm{d}\bar{f}_s\right]$, *where $\bar{f}_s = $ factors$(o_i) \setminus f_s$.*

Factor f_s is independent in $\text{Eff}(o_i)$ if the two sets of random variables f_s and $\bar{f}_s$ are statistically independent in the joint distribution $\text{Eff}(o_i)$. When that is the case, $\text{Eff}(o_i)$ can be broken into independent factors with separate propositions for each.

Let $\text{Eff}_r(o_i)$ denote the effect set that remains after integrating out all independent factors from $\text{Eff}(o_i)$, and $\text{factors}_r(o_i)$ denote the remaining factors. Our method requires a separate propositional symbol for $\text{Eff}_r(o_i)$ with each possible subsets of $\text{factors}_r(o_i)$ integrated out. The vocabulary $\mathcal{P}$ thus contains the following symbols:

1. For each option o_i and factor f_s independent in $\text{Eff}(o_i)$, create a propositional symbol with grounding distribution $\int_{\bar{f}_s} \text{Eff}(o_i)\mathrm{d}\bar{f}_s$.
2. For each set of factors $f_r \subseteq \text{factors}_r(o_i)$, create a propositional symbol with grounding distribution $\int_{f_r} \text{Eff}_r(o_i)\mathrm{d}f_r$.

Propositions that are approximate duplicates can be merged.

Constructing Operator Descriptions Given that it can be executed, each option o_i with possible effect outcome $\text{Eff}^j(o_i)$ results in the following effects:

1. All propositional symbols with grounding distributions for each factor independent in $\text{Eff}^j(o_i)$, and an additional proposition with grounding distribution $\text{Eff}^j_r(o_i)$ if necessary, are set to true.

2. All propositional symbols (except the above) σ_j such that factors(σ_j) $\subseteq$ factors(o_i) and $\int_S \mathcal{G}(\sigma_j)(s)\text{Pre}(o_i, s)\mathrm{d}s > 0$, are set to false.
3. All currently true propositional symbols σ_j where $f_{ij} = \text{factors}(\sigma_j) \cap \text{factors}(o_i) \neq \emptyset$ but factors(σ_j) $\not\subseteq$ factors(o_i), and $\int_S \mathcal{G}(\sigma_j)(s)\text{Pre}(o_i, s)\mathrm{d}s > 0$, are set to false. For each such σ_j, the predicate with grounding distribution $\int_{f_{ij}} \mathcal{G}(\sigma_j)\mathrm{d}f_{ij}$ is set to true.

Each such potential outcome is listed with the probability ρ_j of it occurring.

This computation is analogous to the effects computation in Theorem 1, computing $\text{Im}(o_i, Z) = P(a)\text{Eff}(o_i)$, where $P(a) = \int_B Z([a, b])\,\mathrm{d}b$ (integrating out b, the variables in o_i's mask). The first effect in the above list corresponds to Eff(o_i), and the remaining two types of effect model $P(a)$. The second type of effect removes predicates defined entirely using variables within mask(o_i), whose distributions are completely overwritten. The third models the side-effects of the option, where an existing distribution has the variables in mask(o_i) integrated out. The proof that this image computation is correct closely follows the one given by Konidaris *et al.* [2014] and we omit it here.

However, we face a complication present in the probabilistic setting. Given high-level state $\mathcal{P}_t$ and action o, the agent can compute the probability, ρ, that o can be executed from $\mathcal{P}_t$, and the positive and negative effects for that outcome. But that is a probabilistic *precondition*, and PPDDL only allows for probabilistic *outcomes*. This does not occur in the deterministic case, because the subset relationship is always either true or not. For simplicity, we get around this by adding a virtual proposition named `notfailed`, which is set to true during initialization and is a precondition for every operator. An effect of $\neg$`notfailed` is added to each action with probability $(1 - \rho)$.

Symbol Acquisition in the Treasure Game

The Treasure Game features an agent in a 2D, 528×528 pixel video-game like world, whose task is to obtain treasure and return to its starting position on a ladder at the top left of the screen (see Figure 2). The agent's path may be blocked by closed doors. Flipping the direction of either of the two handles switches the status of the two doors on the top left of the screen (flipping one switch also flips the other). The agent must obtain the key and use it in the lock to open the door on the bottom right of the screen to reach the treasure.

The agent can move up, down, left, and right, jump, interact, and perform a no-op. Left and right movement is available when the agent's way is not directly blocked, while up and down are only available when the agent is on or above, or on or below, respectively, a ladder. These actions move the agent between 2 and 4 pixels (chosen uniformly at random). The interact action is available when the agent is standing in front of a handle (flipping the handle's position from right to left, or vice versa, with probability 0.8), or when it possesses the key and is standing in front of the lock (whereupon the agent loses the key). The no-op action lets the game dynamics continue for one time step, and is useful after a jump action or when the agent is falling. Each action has a reward of -1, except for the jump action, which receives a reward of -5. Returning to the top ladder with the treasure ends the episode. The low-level state space is 9-dimensional, featuring the x and y positions of the agent, key, and treasure, the angles of the two handles, and the state of the lock. When the agent has possession of an item (the key or the treasure), it is displayed in the lower-right corner of the screen.

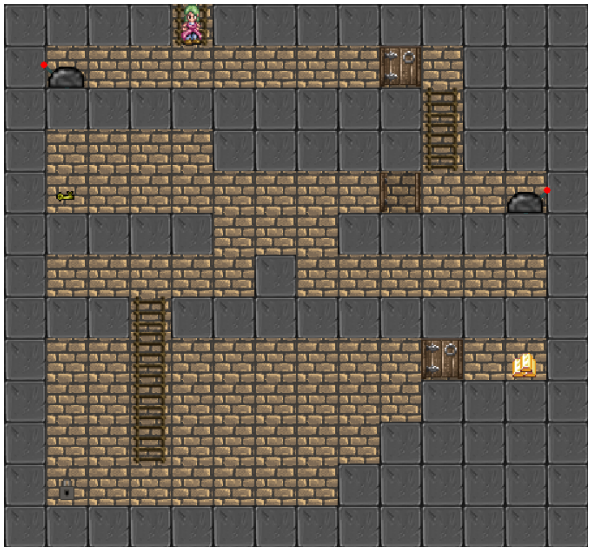


Figure 2: The Treasure Game domain. Sprites courtesy of Hyptosis and `opengameart.org`, Creative Commons license CC-BY 3.0. Although the game screen is drawn using large image tiles, sprite movement is at the pixel level.

The agent has access to the following 9 high-level options, implemented using simple control loops:

- `go-left` and `go-right`, which move the agent continuously left or right, respectively, until it reaches a wall, an edge, an object with which it can interact, or a ladder. These options can only be executed when they would succeed.
- `up-ladder` and `down-ladder`, which cause the agent to ascend or descend a ladder, respectively.
- `down-left` and `down-right`, which cause the agent to execute a controlled fall off an edge onto the nearest solid cell on its left or right, respectively.
- `jump-left` and `jump-right`, which cause the agent to jump and move left, or right, respectively, for about 48 pixels. These options are only available to the agent when the area above its head, and above its head and to the left and right, respectively, are clear.
- `interact`, which executes a primitive interaction.

All options have stochastic termination conditions which, when combined with the stochasticity present in the primitive actions, results in outcome variance ranging from a few pixels (for the `go-left` and `go-right` options) to a much larger amount (e.g., in the case where the `jump-left` option can miss the ledge, causing the agent to fall).

The shortest plan that solves to the Treasure Domain with non-zero probability consists of 42 high-level actions, requiring approximately 3800 low-level actions.

Data was gathered as follows. 100 randomly chosen options were executed sequentially, resulting in one set of data recording whether each option could run at states observed before or after option execution, and another recording the transition data $x_i = (s_i, o_i, r_i, s_i')$ for each executed option. This was repeated 40 times.

Partitioning the Options

First, the options must be partitioned so that the abstract subgoal property approximately holds. This was done using the following procedure for each option o:

1. The mask m_i was computed for each sample transition x_i, and the data was partitioned by mask.
2. For each mask m_j, the effect states $s_i'[m_j]$ were clustered and each cluster was assigned to its own partition. The data was now partitioned into distinct effect distributions, but may have been over-partitioned because distinct effects may occur from the same start state partition.
3. For each pair of partitions, the agent determined whether their start states samples s_i overlapped substantially by clustering the combined start state samples s_i from each partition, and determining whether each resulting cluster contained data from both partitions. If so, the common data was merged into a single partition.
4. When merging, an outcome was created for each effect cluster (which could be distinct due of clustering or due to a different mask) and assigned an outcome probability based on the fraction of the samples assigned to it.

Clustering was performed using the DBSCAN algorithm [Ester *et al.*, 1996] in `scikit-learn` [Pedregosa *et al.*, 2011], with parameters `min_samples = 5` and $\epsilon = 0.4/14$ (for partitioning effects) or $\epsilon = 0.8/14$ (for merging start states). This resulted in 39 partitioned options. Example partitions are visualized in Figures 3 and 4. Note that fuzzy or transparent sprites indicate variance in the distribution.

Creating the Symbolic Vocabulary

The agent created symbols for each of the partitioned options as follows (all parameters set using 3-fold cross-validation):

1. A precondition mask was computed using feature selection with a support vector machine (SVM) [Cortes and Vapnik, 1995].
2. An SVM with Platt scaling [Platt, 1999] was used as a probabilistic precondition classifier, using states assigned to that partition as positive examples and all other states (including those from other partitions of the same option) as negative examples.
3. Kernel density estimation [Rosenblatt, 1956; Parzen, 1962] was used to model each effect distribution.
4. The reward model was learned using support vector regression [Drucker *et al.*, 1997].

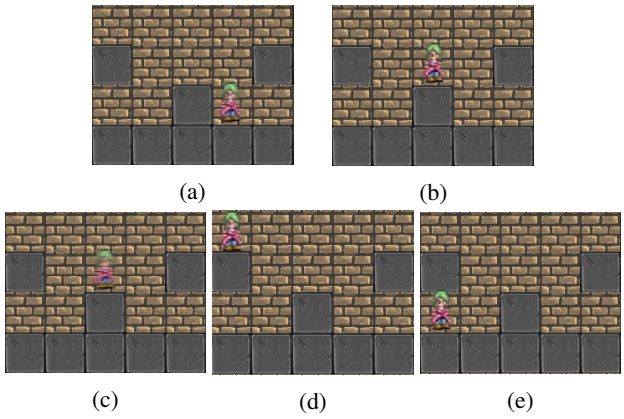


Figure 3: A visualization of both partitions of the `jump-left` option. The first precondition distribution (a) shows the agent standing next to a block. Jumping left leaves agent standing on top of the block (b). The second precondition distribution has the agent standing on the block (c). Jumping left leaves the agent either standing atop the ledge to its left (d, with probability 0.53) or, having missed, on the floor (e, with probability 0.47).

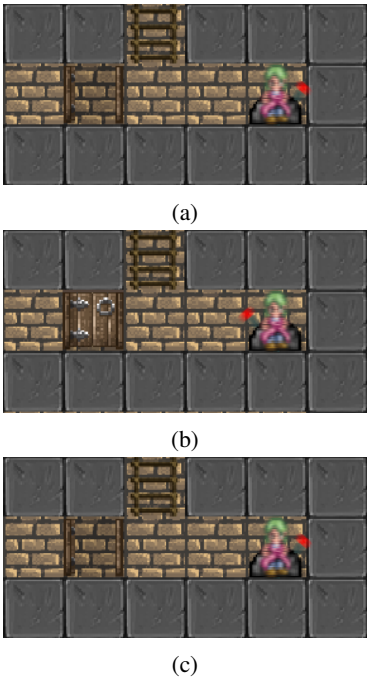


Figure 4: A visualization of the first partition (of five) of the `interact` option. The precondition distribution (a) shows the agent in front of a handle, the handle set to the right, and the door open. Interaction results in an effect distribution where the handle is pushed to the other side and the door is closed (b, with probability 0.795), or where the handle moves only slightly and the door remains open (c, probability 0.204). The two effect distributions have different masks—one changes the angle of one handle, the other changes both.

The agent identified 7 factors from the resulting partitioned probabilistic symbols: (*playerx*; *playery*; *handle1-angle*; *handle2-angle*; *key-x* and *key-y*; *bolt-locked;* and *goldcoin-x* and *goldcoin-y*). The effect distributions were split into 30 distinct type 1 probabilistic symbols (duplicates were detected by a simple coverage interval and mean similarity test). The factors extracted, and the number of effect symbols defined over each factor, are shown in Table 1.

Factor	State Variables	Symbols
1	`playerx`	10
2	`playery`	9
3	`handle1.angle`	2
4	`handle2.angle`	2
5	`key.x, key.y`	3
6	`bolt.locked`	2
7	`goldcoin.x, goldcoin.y`	2

Table 1: Factors identified automatically in the partitioned options extracted from the Treasure Domain, along with the number of probabilistic symbols defined over each.

Constructing a PPDDL Representation

The agent constructed a PPDDL representation by recursing through possible combinations of symbols that overlapped with each partitioned option's precondition mask, and computing the probability of executing that option using Monte Carlo sampling ($m = 100$ samples). Operators estimated to be executable with a probability of less than 5% were discarded, and those with an estimated execution probability of greater than 95% were rounded up to certainty. This resulted in 345 operators; an example operator along with its grounding distributions is given in Figure 5.

Once the PPDDL description was constructed the agent was free to discard its grounding distributions and plan solely using the completely symbolic PPDDL representation. Table 2 shows the time required to compute a policy for the resulting PPDDL problem using the off-the-shelf mGPT planner [Bonet and Geffner, 2005] with the built-in `lrtdp` method and min-min relaxation heuristic. All policies were computed in less than one fifth of a second.

Goal	Min. Depth	Time (ms)
Obtain Key	14	35
Obtain Treasure	26	64
Treasure & Home	42	181

Table 2: Timing results and minimum solution depth (option executions) for example Treasure Game planning problems. Results were obtained on an iMac with a 3.2Ghz Intel Core i5 processor and 16GB of RAM.

Related Work

The most closely related work is that of Jetchev *et al.* [2013], which uses a symbol-as-set definition of a propositional symbol, and then searches for the symbol definitions and a relational, probabilistic STRIPS description of the domain simultaneously. This method is based on a metric that balances predictability and model size, but is hampered by the size of the resulting search. It does not consider uncertainty in the symbol definitions themselves, but is able to find relational operators, which we leave to future work.

Similarly, the early framework described by Huber [2000] can describe transition uncertainty but not uncertainty in the symbols themselves. It is unclear how such uncertainty might be added, since this approach does not refer to low-level states but instead to the discrete states of a set of controllers, which are described in terms of their convergence conditions.

Several researchers have combined a given symbolic vocabulary with learning to obtain symbolic operator models for use in planning [Drescher, 1991; Schmill *et al.*, 2000; Džeroski *et al.*, 2001; Pasula *et al.*, 2007; Amir and Chang, 2008; Kruger *et al.*, 2011; Lang *et al.*, 2012; Mourão *et al.*, 2012]. Our work shows how to construct the symbolic vocabulary itself.

Option discovery is an active topic of research in hierarchical reinforcement learning; see the recent review by Hengst [2012]. This area is largely concerned with discovering an appropriate set of options in an MDP, the presence of which is assumed in our work.

Modayil and Kuipers [2008] use a learned model of the effect of actions on an object to perform high-level planning. However, the learned models are still in the original state space. Later work by Mugan and Kuipers [2012] use *qualitative distinctions* to adaptively discretize a continuous state space to acquire a discrete model suitable for planning; here, discretization is based on the ability to predict the outcome of executing an action.

Other approaches to MDP abstraction have focused on discretizing large continuous MDPs into abstract discrete MDPs [Munos and Moore, 1999] or minimizing the size of a discrete MDP model [Dean and Givan, 1997].

Conclusion

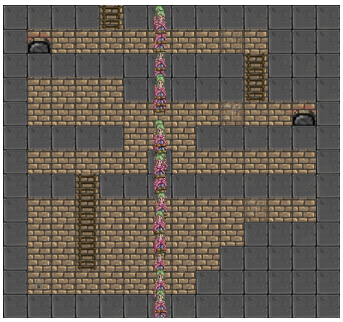
We have specified the symbolic representation required by an agent that wishes to perform probabilistic planning using a set of high-level actions. Our formalization enables an agent to *autonomously learn its own symbolic representations*, and to use those representations to perform efficient high-level probabilistic planning.

Acknowledgements

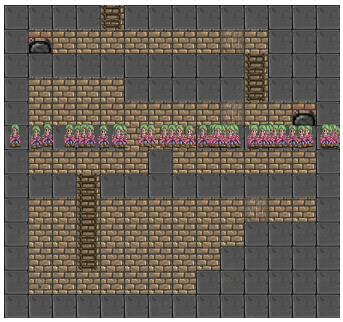
We thank the anonymous reviewers for their thoughtful suggestions. This work was supported in part by the NSF (grant 1420927). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We also gratefully acknowledge support from the ONR (grant N00014-14-1-0486), from the AFOSR (grant FA23861014135), and from the ARO (grant W911NF1410433). GDK was supported in part by an MIT Intelligence Initiative Fellowship.

```
(:action jump_left_option319
 :parameters ()
 :precondition (and (notfailed) (symbol29) (symbol28) )
 :effect (probabilistic  0.4723 (and (symbol17) (symbol1) (not (symbol28)) (not (symbol29))
                                    (decrease (reward) 62.39))
                         0.5277 (and (symbol20) (symbol1) (not (symbol28)) (not (symbol29))
                                    (decrease (reward) 36.32))
         )
   )
```

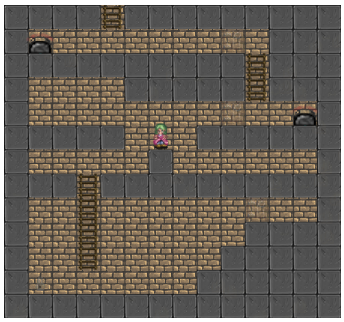
(a) Generated PDDL Operator



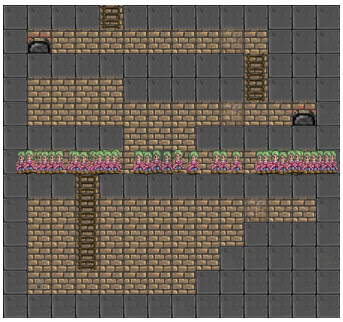
(b) `symbol29`



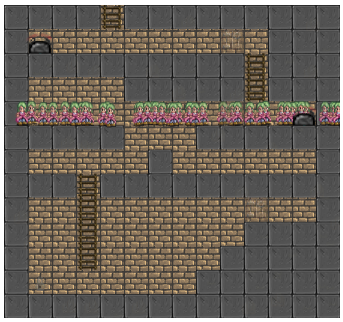
(c) `symbol28`



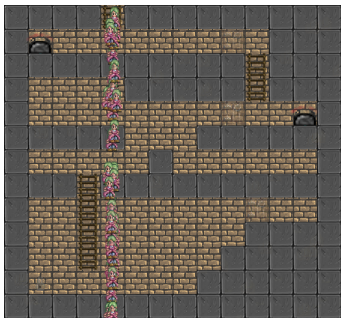
(d) `symbol28&29`



(e) `symbol17`



(f) `symbol20`



(g) `symbol1`

Figure 5: The automatically generated PPDDL operator for one partition of the `jump-left` option (a), together with 50 samples drawn from each symbol's grounding distribution. The precondition distributions (`symbol29` and `symbol28`, b and c) together indicate that the agent should be atop the concrete block in the center of the domain (d). Executing the option results in one of two y coordinate outcomes named by `symbol17` and `symbol20` (e and f). Both outcomes set the agent's x coordinate according to the distribution named by `symbol1` (g). The two outcomes have different rewards—the failed jump costs more because the agent has to wait until it has fallen past the bottom of the ledge before it can finish moving left.

References

[Amir and Chang, 2008] E. Amir and A. Chang. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33:349–402, 2008.

[Bonet and Geffner, 2005] B. Bonet and H. Geffner. mGPT: a probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research*, 24:933–944, 2005.

[Cambon *et al.*, 2009] S. Cambon, R. Alami, and F. Gravot. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research*, 28(1):104–126, 2009.

[Choi and Amir, 2009] J. Choi and E. Amir. Combining planning and motion planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4374–4380, 2009.

[Cortes and Vapnik, 1995] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[Dean and Givan, 1997] T. Dean and R. Givan. Model minimization in Markov decision processes. In *In Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 106–111, 1997.

[Dornhege *et al.*, 2009] Christian Dornhege, Marc Gissler, Matthias Teschner, and Bernhard Nebel. Integrating symbolic and geometric planning for mobile manipulation. In *IEEE International Workshop on Safety, Security and Rescue Robotics*, November 2009.

[Drescher, 1991] G.L. Drescher. *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. MIT Press, 1991.

[Drucker *et al.*, 1997] H. Drucker, C.J.C. Burges, L. Kaufman, A.J. Smola, and V.N. Vapnik. Support vector regression machines.

In *Advances in Neural Information Processing Systems 9*, pages 155–161, 1997.

[Džeroski *et al.*, 2001] S. Džeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine learning*, 43(1):7–52, 2001.

[Ester *et al.*, 1996] M. Ester, H.P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[Gat, 1998] E. Gat. On three-layer architectures. In D. Kortenkamp, R.P. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*. AAAI Press, 1998.

[Ghallab *et al.*, 2004] M. Ghallab, D. Nau, and P. Traverso. *Automated planning: theory and practice*. Morgan Kaufmann, 2004.

[Hengst, 2012] Bernhard Hengst. Hierarchical approaches. In Marco Wiering and Martijn van Otterlo, editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 293–323. Springer Berlin Heidelberg, 2012.

[Huber, 2000] M. Huber. A hybrid architecture for hierarchical reinforcement learning. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pages 3290–3295, 2000.

[Jetchev *et al.*, 2013] N. Jetchev, T. Lang, and M. Toussaint. Learning grounded relational symbols from continuous data for abstract reasoning. In *Proceedings of the 2013 ICRA Workshop on Autonomous Learning*, 2013.

[Kaelbling and Lozano-Pérez, 2011] L. Kaelbling and T. Lozano-Pérez. Hierarchical planning in the Now. In *Proceedings of the IEEE Conference on Robotics and Automation*, 2011.

[Konidaris *et al.*, 2014] G.D. Konidaris, L.P. Kaelbling, and T. Lozano-Perez. Constructing symbolic representations for high-level planning. In *Proceedings of the Twenty-Eighth Conference on Artificial Intelligence*, pages 1932–1940, 2014.

[Kruger *et al.*, 2011] N. Kruger, C. Geib, J. Piater, R. Petrick, M. Steedman, F. Wörgötter, Aleš Ude, T. Asfour, D. Kraft, D. Omrčen, Alejandro Agostini, and Rüdiger Dillmann. Object-action complexes: Grounded abstractions of sensory-motor processes. *Robotics and Autonomous Systems*, 59:740–757, 2011.

[Lang *et al.*, 2012] T. Lang, M. Toussaint, and K. Kersting. Exploration in relational domains for model-based reinforcement learning. *Journal of Machine Learning Research*, 13:3691–3734, 2012.

[Malcolm and Smithers, 1990] C. Malcolm and T. Smithers. Symbol grounding via a hybrid architecture in an autonomous assembly system. *Robotics and Autonomous Systems*, 6(1-2):123–144, 1990.

[Modayil and Kuipers, 2008] J. Modayil and B. Kuipers. The initial development of object knowledge by a learning robot. *Robotics and Autonomous Systems*, 56(11):879–890, 2008.

[Mourão *et al.*, 2012] K. Mourão, L. Zettlemoyer, R.P.A. Patrick, and M. Steedman. Learning STRIPS operators from noisy and incomplete observations. In *Proceedings of Conference on Uncertainty in Articial Intelligence*, 2012.

[Mugan and Kuipers, 2012] J. Mugan and B. Kuipers. Autonomous learning of high-level states and actions in continuous environments. *IEEE Transactions on Autonomous Mental Development*, 4(1):70–86, 2012.

[Munos and Moore, 1999] R. Munos and A. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1348–1355, 1999.

[Nilsson, 1984] N.J. Nilsson. Shakey the robot. Technical report, SRI International, April 1984.

[Parzen, 1962] E. Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065, 1962.

[Pasula *et al.*, 2007] H. Pasula, L.S. Zettlemoyer, and L.P. Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29:309–352, 2007.

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, 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. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Platt, 1999] J.C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

[Rosenblatt, 1956] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832, 1956.

[Schmill *et al.*, 2000] M.D. Schmill, T. Oates, and P.R. Cohen. Learning planning operators in real-world, partially observable environments. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, pages 245–253, 2000.

[Sutton *et al.*, 1999] R.S. Sutton, D. Precup, and S.P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[Wolfe *et al.*, 2010] Jason Wolfe, Bhaskara Marthi, and Stuart J. Russell. Combined Task and Motion Planning for Mobile Manipulation. In *International Conference on Automated Planning and Scheduling*, 2010.

[Younes and Littman, 2004] H.L.S. Younes and M.L. Littman. PPDDL 1.0: an extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University, 2004.