# Learning to Plan with Portable Symbols

**Steven James** [1] **Benjamin Rosman** [1 2] **George Konidaris** [3]

## Abstract

We present a framework for autonomously learning a portable symbolic representation that describes a collection of low-level continuous environments. We show that abstract representations can be learned in a task-independent space specific to the agent that, when combined with problem-specific information, can be used for planning. We demonstrate knowledge transfer in a video game domain where an agent learns portable, task-independent symbolic rules, and then learns instantiations of these rules on a per-task basis, reducing the number of samples required to learn a representation of a new task.

## 1. Introduction

On the surface, the learning and planning communities operate in very different paradigms. Agents in reinforcement learning (RL) interact directly with the environment in order to learn either an optimal behaviour or the dynamics of the environment (Sutton & Barto, 1998). The latter produces a learned forward model of the transition dynamics, which can then be used in some form of tree search to compute optimal actions (Coulom, 2006; Kocsis & Szepesvári, 2006).

Unfortunately, this approach founders when confronted with low-level, high-dimensional and continuous state and action spaces. The innate action space of a robot, for instance, involves directly actuating motors at a high frequency, but it would take thousands of such actuations to accomplish most useful goals. Thus planning is simply infeasible, even with a perfect model.

Approaches such as hierarchical reinforcement learning (Barto & Mahadevan, 2003) tackle this problem by abstracting away the low-level action space using higher-level *skills*, which can accelerate learning and planning. While skills alleviate the problem of reasoning over low-level actions, the state space remains complex and planning continues to be challenging.

On the other hand, the classical planning approach is to represent the world using abstract symbols, with actions represented as operators that manipulate these symbols (Ghallab et al., 2004). Such representations use only the minimal amount of state information necessary for task-level planning. This is appealing since it mitigates the issue of *reward sparsity* and admits solutions to long-horizon tasks, but raises the question of how to build the appropriate abstract representation of a problem. This is often resolved manually, requiring substantial effort and expertise. Fortunately, recent work demonstrates how to learn a provably sound symbolic representation autonomously, given only the data obtained by executing the high-level actions available to the agent (Konidaris et al., 2018).

A major shortcoming of that framework is the lack of generalisability—an agent must relearn the appropriate symbolic representation for each new task it encounters. This is a data- and computation-intensive procedure involving clustering, probabilistic multi-class classification, and density estimation in high-dimensional spaces.

We introduce a framework for deriving a symbolic abstraction over a portable state space known as *agent space* (Konidaris et al., 2012). Because agent space depends only on the sensing capabilities of the agent (which remain constant regardless of the environment), it is independent of the underlying state space and thus a suitable mechanism for transfer.

We demonstrate successful transfer in the *Treasure Game* (Konidaris et al., 2015), indicating that an agent is able to learn symbols that generalise to different tasks, reducing the amount of experience required to learn a high-level representation of a new task.

## 2. Background

We assume that the tasks faced by an agent can be modelled as a semi-Markov decision process (SMDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{S} \subseteq \mathbb{R}^n$ is the $n$-dimensional continuous state space and $\mathcal{O}(s)$ is the set of temporally-extended actions known as *options* available to the agent at state $s$. The reward function $\mathcal{R}(s, o, \tau, s')$ specifies the feedback

[1]University of the Witwatersrand, Johannesburg, South Africa [2]Council for Scientific and Industrial Research, Pretoria, South Africa [3]Brown University, Providence RI 02912, USA. Correspondence to: Steven James <Steven.James@wits.ac.za>.

Accepted to the *ICML/IJCAI/AAMAS 2018 Workshop on Planning and Learning (PAL-18)*, Stockholm, Sweden.

the agent receives from the environment when it executes option $o$ from state $s$ and arrives in state $s'$ after $\tau$ steps. $\mathcal{T}$ describes the dynamics of the environment, specifying the probability of arriving in state $s'$ after option $o$ is executed from $s$ for $\tau$ timesteps: $\mathcal{T}^o_{ss'} = \Pr(s', \tau \mid s, o)$.

An option $o$ is defined by the tuple $\langle I_o, \pi_o, \beta_o \rangle$, where $I_o$ is the *initiation set* that specifies the states in which the option can be executed, $\pi_o$ is the *option policy* which specifies the action to execute, and $\beta_o$ is the *termination condition*, where $\beta_o(s)$ is the probability of option $o$ halting in state $s$.

### 2.1. Portable Skills

We adopt the approach of Konidaris et al. (2012) whereby tasks are related because they are faced by the same agent. For example, consider a robot equipped with various sensors that is required to perform a number of as yet unspecified tasks. The only aspect that remains constant across all these tasks is the presence of the robot, and more importantly its sensors, which map the state space to a portable observation space $\mathcal{D}$ known as *agent space*.

We define an observation function $\phi : \mathcal{S} \to \mathcal{D}$ that maps states to observations and depends on the sensors available to an agent. We assume the sensors may be noisy, but that the noise has mean 0 in expectation, so that if $s, t \in \mathcal{S}$, then $s = t \implies \mathbb{E}[\phi(s)] = \mathbb{E}[\phi(t)]$. We refer to the SMDP's original state space as *problem space*.

Augmenting an SMDP with this new agent space produces the tuple $\langle \mathcal{S}, \mathcal{O}, \mathcal{T}, \mathcal{R}, \gamma, \mathcal{D} \rangle$, where the observation space $\mathcal{D}$ remains constant across all tasks. We can use $\mathcal{D}$ to learn *agent-space options*, whose option policies, initiation sets and termination conditions are all defined in agent space. Because $\mathcal{D}$ remains constant regardless of the underlying SMDP, these options can be transferred across tasks.

### 2.2. Abstract Representations

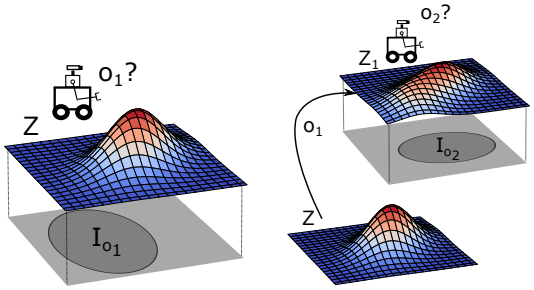
Much like Konidaris et al. (2018), we are interested in learning an abstract representation to facilitate planning—that is, *learning to plan*. We define a *probabilistic plan* $p_Z = \{o_1, \ldots, o_n\}$ to be the sequence of options to be executed, starting from some state drawn from distribution $Z$. It is useful to introduce the notion of a *goal option*, which can only be executed when the agent has reached its goal. Appending this option to a plan means that the probability of successfully executing a plan is equivalent to the probability of reaching some goal.

A representation suitable for planning must allow us to calculate the probability of a given plan executing to completion. As a plan is simply a chain of options, we must therefore learn when an option can be executed, as well as the outcome of doing so. This corresponds to learning the *precondition*, which expresses the probability that option $o$ can be executed at a given state, and the *image*, which represents the distribution of states an agent may find itself in after executing $o$ from states drawn from some distribution. Figure 1 illustrates how the precondition and image are used to calculate the probability of executing a two-step plan.

For continuous state spaces, we cannot represent the image of an arbitrary option; however, we can do so for a subclass known as *subgoal options* (Precup, 2000), whose terminating states are independent of their starting states (Konidaris et al., 2018). That is, for any subgoal option $o$, $\Pr(s' \mid s, o) = \Pr(s' \mid o)$. We can thus substitute the option's image for its *effect*. If an option is not subgoal, we may be able to *partition* its initiation set into a finite number of subsets, so that it becomes subgoal when initiated from each of the individual subsets. That is, we divide an option $o$'s start states into classes $\mathcal{C}$ such that $P(s'|s, o, c) \approx P(s'|o, c) \forall c \in \mathcal{C}$. Given subgoal options, we can construct a plan graph corresponding to an abstract MDP.

We may also assume that the option is *abstract*—that is, it obeys the frame and action outcomes assumptions (Pasula et al., 2004). For each option, we can decompose the state into two sets of variables $s = [a, b]$ such that executing the option results in state $s' = [a, b']$, where $a$ is the subset of variables that remain unchanged.

Whereas subgoal options induce an abstract MDP, abstract subgoal options allow us to construct a model corresponding to a *factored* abstract MDP. Equivalently, subgoal options induce a PPDDL description (Younes & Littman, 2004), where each operator's precondition and positive effect is a single proposition. Abstract subgoal options result in preconditions and effects with conjunctive propositions.




(a) The agent begins at distribution $Z$, and must determine the probability with which it can execute the first option $o_1$.

(b) The agent estimates the effect of executing $o_1$, given by $Z_1$. It must then determine the probability of executing $o_2$ from $Z_1$.

*Figure 1.* An agent attempting to calculate the probability of executing the plan $p_Z = \{o_1, o_2\}$, which requires knowledge of the conditions under which $o_1$ and $o_2$ can be executed, as well as the effect of executing $o_1$ (Konidaris et al., 2018).

## 3. Building a Portable Symbolic Vocabulary

Prior work (Konidaris et al., 2018) has defined *symbols* as names for precondition and effect distributions over low-level states, which are directly tied to the SMDP in which they were learned. We instead propose learning a symbolic representation over agent space $\mathcal{D}$. Transfer can be achieved in this manner (provided $\phi$ is *non-injective*[1]), because $\mathcal{D}$ remains consistent both within the same SMDP and across SMDPs, even if the state space or transition function do not.

We assume that the agent possesses subgoal agent-space options, but that the goal (and hence goal option) is defined in problem space. This produces an issue because, given a current distribution over agent-space observations, *we cannot determine the probability with which a problem-space option can be executed*. This follows naturally from the property that $\phi$ is non-injective. We therefore require additional information to disambiguate such situations, allowing us to map from agent space back into problem space.

We can accomplish this by partitioning our agent-space options based on their effects in $\mathcal{S}$, resulting in options that are subgoal in both $\mathcal{D}$ and $\mathcal{S}$. This necessitates having access to both problem- and agent-space observations. Recall that options are partitioned to ensure the subgoal property holds, and so each partition defines its own unique image distribution. If we label each class in $\mathcal{C}$, then each label refers to a unique distribution in $\mathcal{S}$ and is sufficient for disambiguating our agent-space symbols.

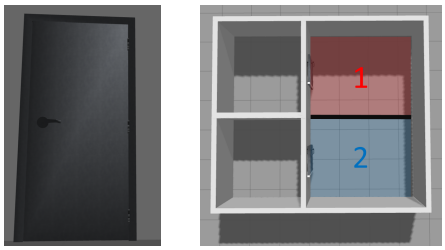
Consider an option `GoToDoor` that causes the agent to approach the door in its field of view. Figure 2a illustrates its effect given an agent-centric view, which is then combined with partition labels (Figure 2b). This produces the lifted symbol `InFrontOfDoor(X)`, where `InFrontOfDoor` is the name for a distribution over $\mathcal{D}$ and `X` is simply a partition number. Note that the only time problem-specific information is required is when determining the values of `X`.

## 4. Generating a Forward Model

We now show how to build a forward model by combining agent-space distributions and partition labels, which results in parameterised symbols. This can be viewed as a two-step process. The agent first learns domain-independent symbols in agent space, which are not strictly tied to the current task, and then determines their parameters and how they link to each other, which depends on $\mathcal{S}$ and thus the current task.

The first phase is equivalent to learning propositions in agent space, which are portable because agent space is shared between SMDPs. An example is an agent that learns that the effect of passing through a door is that it finds itself in a room. The second phase learns the specifics of the current environment: which doors connect to which rooms, for example, which differs between buildings. This involves learning how the parameter of the precondition relates to the effect's parameter.

[1]We require that $\phi$ be non-injective since the ability to effect transfer relies on two distinct states in $\mathcal{S}$ being identical in $\mathcal{D}$.




(a) (b)

*Figure 2.* (a) A single agent-space symbol representing the effect of the option `GoToDoor`. (b) `GoToDoor` is subgoal in $\mathcal{D}$ but not in $xy$-space, since the door it approaches depends upon the agent's location. However, we can partition the option into two options `GoToDoor(1)` and `GoToDoor(2)` which *are* subgoal in $\mathcal{S}$. The coloured regions represent the two partitions.

We describe the approach for the *Treasure Game*, a dungeon-crawler video game in which an agent navigates a maze in search of treasure. The domain contains ladders as well as doors which impede the agent. Some doors can be opened and closed with levers, while others require a key to unlock. The agent possesses a number of high-level agent-space subgoal options, each of which executes many pixel-level primitive actions such as `JumpLeft` and `ClimbLadder`. More details are given by Konidaris et al. (2018).

The problem space consists of the $xy$-position of the agent, key and treasure, the angle of the levers (which determines whether a door is open) and the state of the lock. We construct the agent space by first tiling the screen into cells.We then produce a vector of length 9 whose elements are the sprite types in the cells adjacent to the agent.

We learn a representation using agent-space transitions only, following a procedure identical to Konidaris et al. (2018). First, we collect agent-space transitions by interacting with the environment. We use SVMs (Cortes & Vapnik, 1995) with Platt scaling (Platt, 1999) to estimate preconditions, and use kernel density estimation (Rosenblatt, 1956; Parzen, 1962) to model effect distributions. Finally, for all valid combinations of effect distributions, we compute the probability that states drawn from their grounding lie within the precondition of each option, discarding rules with a success probability of less than $5\%$. This results in portable action rules, one of which is illustrated by Figure 3a.

We then partition the above options according to their effects in problem space. We follow previous work (Konidaris et al., 2015; Andersen & Konidaris, 2017) by clustering options'

effect states using DBSCAN (Ester et al., 1996), and then assigning each cluster a label. For each transition, we record the start and end partition label, which allows us to learn the link between the precondition and effect parameter. This instantiates the rules for the given task. Figure 3b shows the different partition labels for the `DownLadder` operator.

## 5. Inter-Task Transfer

We construct a set of seven tasks corresponding to different levels. Because the levels have different configurations, constructing a representation in problem space requires that we relearn each level from scratch. However, when constructing an agent-centric representation, rules learned in one task can immediately be used in subsequent tasks. We gather $k$ transition samples from each task by executing options uniformly at random, and use these samples to build both task-specific and agent-centric (portable) models.

(a) (b)

*Figure 3.* (a) The precondition (top) and effect (bottom) for the `DownLadder` operator, which states that in order to execute the option, the agent must be standing above the ladder. The option results in the agent standing on the ground below it. The black spaces refer to unchanged low-level state variables. (b) Three problem-space partitions for the `DownLadder` operator. We assign a unique label to each of the circled partitions and combine it with the portable rule in (a) to produce a grounded operator.

To evaluate model accuracy, we record a set of 100 two-step plans $\langle s_1, o_1, s_2, o_2 \rangle$ for each task. Let $\mathcal{M}_k^{\rho_i}$ be the model constructed for task $\rho_i$ using $k$ samples. We then calculate the likelihood of each plan under the model: $\Pr(s_1 \in I_{o_1} \mid \mathcal{M}_k^{\rho_i}) \times \Pr(s' \in I_{o_2} \mid \mathcal{M}_k^{\rho_i})$, where $s' \sim \text{Eff}(o_1)$. We build models using increasing numbers of samples until the likelihood averaged over all plans is greater than some acceptable threshold (we use a value of 0.75), at which point we continue to the next task. The results are given by Figure 4.

The results show a sample complexity that scales linearly with the number of tasks when learning problem-space symbols. This is to be expected, since each task is independent and must be learned from scratch. Conversely, learning and reusing portable symbols require fewer samples as we encounter more tasks, leading to a *sublinear* increase.

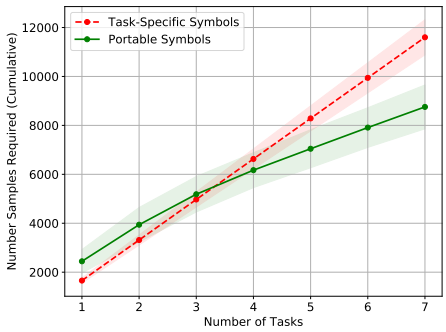



*Figure 4.* Cumulative number of samples required to learn sufficiently accurate models as a function of the number of tasks encountered. Results are averaged over 100 random permutations of the task order. Standard errors are specified by the shaded areas.

We might have expected to see a plateauing in sample collection as we learn more portable rules. However, this was not the case owing to uniform random exploration—a better approach would have been to use knowledge of the currently learned partitions to guide exploration (e.g. partitions that we have yet to model should receive more attention). Despite this, our approach still demonstrates the advantage of transfer.

## 6. Related Work

There has been some work in autonomously learning parameterised skills, particularly in the field of relational reinforcement learning. Zettlemoyer et al. (2005), for instance, are able to learn parameterised operators; however, the high-level symbols that constitute the state space are given. *Relocatable action models* (Leffler et al., 2007) aggregate states into "types" which determine the transition behaviour. State-independent representations of the outcomes from different types are learned and shown to improve the learning rate in a single task. Jetchev et al. (2013), Ugur & Piater (2015) and Kaelbling & Lozano-Pérez (2017) are able to discover parameterised symbols directly from low-level states. In these cases, the parameters refer to the state variables modified by the action. Zhang et al. (2018) learn models of an environment's dynamics which can generalise across tasks, but the abstraction is given.

## 7. Summary

We specified a framework for learning portable symbols, showing that the addition of problem-specific information can be used to construct a representation for planning. This allows us to leverage experience in solving new unseen tasks—a step towards creating adaptable, long-lived agents.

## References


Andersen, G. and Konidaris, G.D. Active exploration for learning symbolic representations. In *Advances in Neural Information Processing Systems*, pp. 5016–5026, 2017.

Barto, A.G. and Mahadevan, S. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

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

Coulom, R. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games*, pp. 72–83. Springer, 2006.

Ester, M., Kriegel, H., Sander, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *2nd International Conference on Knowledge Discovery and Data Mining*, volume 96, pp. 226–231, 1996.

Ghallab, M., Nau, D., and Traverso, P. *Automated Planning: theory and practice*. Elsevier, 2004.

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

Kaelbling, L.P. and Lozano-Pérez, T. Learning composable models of parameterized skills. In *Proceedings of the 2017 IEEE International Conference on Robotics and Automation*, pp. 886–893. IEEE, 2017.

Kocsis, L. and Szepesvári, C. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pp. 282–293. Springer, 2006.

Konidaris, G.D., Scheidwasser, I., and Barto, A.G. Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research*, 13(May):1333–1371, 2012.

Konidaris, G.D., Kaelbling, L.P., and Lozano-Pérez, T. Symbol acquisition for probabilistic high-level planning. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, pp. 3619–3627, 2015.

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

Leffler, Bethany R, Littman, Michael L, and Edmunds, Timothy. Efficient reinforcement learning with relocatable action models. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, volume 7, pp. 572–577, 2007.

Parzen, E. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3): 1065–1076, 1962.

Pasula, H., Zettlemoyer, L.S., and Kaelbling, L.P. Learning probabilistic relational planning rules. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pp. 73–81, 2004.

Platt, J. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

Precup, D. *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 2000.

Rosenblatt, N. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, pp. 832–837, 1956.

Sutton, R.S. and Barto, A.G. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

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

Younes, H.L.S. and Littman, M.L. PPDDL 1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical report, 2004.

Zettlemoyer, L.S., Pasula, H., and Kaelbling, L.P. Learning planning rules in noisy stochastic worlds. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pp. 911–918, 2005.

Zhang, A., Lerer, A., Sukhbaatar, S., Fergus, R., and Szlam, A. Composable planning with attributes. *arXiv preprint arXiv:1803.00512*, 2018.