Symbolic Planning and Model-Free Reinforcement Learning: Training Taskable Agents

León Illanes
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
lillanes@cs.toronto.edu

Xi Yan
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
xi.yan@mail.utoronto.ca

Rodrigo Toro Icarte
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
rntoro@cs.toronto.edu

Sheila A. McIlraith
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
sheila@cs.toronto.edu

Abstract

We investigate the use of explicit symbolic action models, as typically used for Automated Planning, in the context of Reinforcement Learning (RL). Our objective is to make RL agents more sample efficient and human taskable. We say an agent is taskable when it is capable of achieving a variety of different goals and there is a simple method for goal specification. Moreover, we expect taskable agents to easily transfer skills learned for one task to other related tasks. To these ends, we consider high-level models that inexactly represent the low-level environment in which an agent acts. Given a model, defining goal-directed tasks is a simple problem, and we show how to communicate these goals to an agent by leveraging state-of-the-art symbolic planning techniques. We automatically generate families of high-level solutions and subsequently represent them as a reward machine, a recently introduced formalism for describing structured reward functions. In doing this, we not only specify what the task at hand is, but also give a high-level description of how to achieve it. The structure present in this description can be successfully exploited by a Hierarchical RL system. The reward machine represents a collection of sequential solutions and can be used to prune the options available when training. We can ensure that, at every step, the meta-controller can only select options that represent advancement in some high-level plan.

We empirically demonstrate the merits of our approach, comparing to a naive baseline where a single sequential plan is strictly followed, and to standard Hierarchical RL techniques. Our results show that the approach is an effective method for specifying tasks to an RL agent. Given adequately pretrained options, our approach reaches high-quality policies in previously unseen tasks in extremely few training steps and consistently outperforms the standard techniques.

Keywords: Reinforcement Learning
Hierarchical Reinforcement Learning
Goal Specification
Automated Planning
Symbolic Planning

Acknowledgements

We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), Becas Chile (CONICYT), and Microsoft Research.

1 Introduction

Reinforcement learning (RL) techniques allow for agents to perform tasks in complex domains, where environment dynamics and reward structures are initially unknown. These techniques are based on performing random exploration, observing the dynamics and reward returned by the environment, and synthesizing an optimal policy that maximizes expected cumulative reward. Unfortunately, when reward is sparsely distributed, as is the case in many applications, RL techniques can suffer from poor *sample efficiency*, requiring millions of episodes to learn reasonable policies. Further, these systems are typically not *taskable*: specifying new tasks is often difficult and the skills that are learned for one task are not easily transferred to others. Over the years a number of approaches have been proposed to address these shortcomings including efforts to learn hierarchical representations or to define *options*, a form of macro-action that can be used by the RL system [9] .

Our interest in this paper is in leveraging high-level symbolic planning models and automated plan synthesis techniques, in concert with state-of-the-art RL techniques, with the objective of significantly improving sample efficiency and creating systems that are human taskable. Our efforts are based on the observation that some approximated understanding of the environment can be characterized as a symbolic planning model—a set of properties of the world and actions that cause those properties to change in predictable ways.

Recent research has demonstrated the benefit of providing high-level instructions to an RL system to improve sample efficiency (e.g., policy sketches [1]). Nevertheless, these instructions must be *manually* generated. By using a symbolic model of the environment rather than a task-specific set of instructions we are able to *automatically* generate instructions in the form of one or more sequential or partial order plans—the latter compactly characterizing a multitude of sequential plans. We represent these plans as a structured reward function in the form of an automata-inspired reward machine [10]. This reward machine is then used to enhance a Hierarchical Reinforcement Learning (HRL) system by ignoring options that—based on inspection of the machine's structure—cannot result in future reward.

We compare our approach to standard forms of HRL and to a naive baseline algorithm. Our results show that the approach is an effective method for specifying tasks to an RL agent, reaching high-quality policies for previously unseen tasks in extremely few training steps.

2 Related Work

The options framework [9] has become a well-known standard approach for exploiting temporal abstraction in Reinforcement Learning. A key contribution of our work is the way in which we use explicit symbolic planning to select adequate options.

Other existing approaches have used symbolic planning to select options or macro-actions. An early approach proposed using a symbolic planner coupled into an RL agent [2]. There, the planner produces an initial high-level plan and is subsequently used to replan when the plan's preconditions are violated. A related approach uses a sequential plan to modify the reward via reward shaping [3]. In contrast to these approaches, our system uses planners to produce a single structured reward function that represents the task at hand. Our approach is orthogonal to the idea of reward shaping, and we believe that it may be applied in our setting.

Recent work has also proposed methods based on coupling a planner to an RL agent [11, 7]. There, the focus has been on two-way communication between agent and planner to continuously improve the high-level model and find better high-level solutions. In our case, we assume an adequate high-level model is given and we show how to exploit it. Other work has explored techniques for automatically building such abstractions [6, 5]. Integrating such techniques into our work may prove to be an interesting direction for future work.

Finally, there has also been work that has focused on learning explicit state-transition systems that represent high-level models [12]. With these, standard graph search algorithms can be used to find sequences of macro-actions. Our work considers *implicit* state-transition systems described as classical planning domains. This allows us to consider highly combinatorial problems that are far too large to represent explicitly.

3 Preliminaries

For the purposes of this work, we will say that the environment in which an RL agent acts is formalized as a tuple $\mathcal{E} = \langle S, A, p \rangle$, where S is its set of states, A is the set of available actions, and $p(s_{t+1} \mid s_t, a_t)$ is the transition probability distribution. A policy is defined as a probability distribution $\pi(a \mid s)$ that establishes the probability of the agent taking action a given that its current state is s.

For defining tasks, we consider a recently introduced reward mechanism known as reward machines [10]. A reward machine is a finite-state machine that can be used to specify temporally-extended and non-Markovian reward functions. The intuitive idea is that transitions in the reward machine take place based on observations made by the agent about the environment, and that the reward depends on the transitions taken in the machine. The observations are represented by a set of propositional symbols $\mathcal{P}$, which correspond to facts that the agent may perceive. For a given environment, we assume there is a labeling function $L\colon S \to 2^{\mathcal{P}}$ that establishes what is perceived when reaching a state. Then, a reward machine for environment $\mathcal{E}$ and observation propositions $\mathcal{P}$ is given by the tuple $\mathcal{R} = \langle U, u_0, \delta_u, \delta_r \rangle$. U is its set of states, u_0 is its initial state, $\delta_u\colon U \times 2^{\mathcal{P}} \to U$ is its state transition function, and $\delta_r\colon U \times U \to \mathbb{R}$ is its reward transition function. Whenever the agent makes a transition (s, a, s') in the environment and observes $P \subseteq \mathcal{P}$, the current state in the reward machine is updated from u to $u' = \delta_u(u, P)$. At this point, the agent receives reward $\delta_r(u, u')$.

Temporal Abstraction in RL The options framework proposes the use of policies that are trained for achieving specific high-level behaviours, coupled with well-defined termination criteria for their application [9]. An agent acting within this framework can choose, at every step, to either apply a low-level action or to apply one of these high-level options. We use a simplistic notion of option defined as a pair $o = \langle \pi_o, T_o \rangle$, where π_o is the corresponding policy and $T_o \subseteq S$ is a set of states that the policy is intended to reach and that will be used as the option's termination criterion.

Symbolic Planning We specify planning domains in terms of a tuple $\mathcal{D} = \langle F, \mathcal{A} \rangle$. F is a set of propositional symbols, called the fluents of $\mathcal{D}$, and $\mathcal{A}$ is the set of planning actions in the domain. Planning states are specified as subsets of F, so that state $S \subseteq F$ represents the situation in which the fluents in S are all true and those not in S are false. Actions are specified in terms of their preconditions and effects, which are given as logical formulae over the propositional fluents. Actions are only applicable in states where their preconditions are satisfied, and such application results in a transition to a state where the action's effects are true, and everything else remains unchanged.

A planning task is described by an initial state and a goal condition. The goal is given as a formula over the fluents of the domain. Any state that satisfies the goal condition is said to be a goal state. A sequence of actions $\Pi = [a_0, a_1, \ldots, a_n]$ is known as a sequential plan for a task when it is possible to sequentially apply the actions starting at the initial state, and doing so reaches a goal state. Given a plan $\Pi = [a_0, a_1, \ldots, a_n]$, we will refer to its prefix with respect to action a_i as **prefix**$(\Pi, a_i) = [a_0, a_1, \ldots, a_{i-1}]$.

Partial-order plans generalize sequential plans by relaxing the ordering condition over the actions. A partial-order plan is a tuple $\overline{\Pi} = \langle \overline{\mathcal{A}}, \prec \rangle$, where $\overline{\mathcal{A}}$ is its set of action occurrences and $\prec$ is a partial order over $\overline{\mathcal{A}}$. The set of linearizations of $\overline{\Pi}$, denoted $\Lambda(\overline{\Pi})$, is the set of all sequences of the action occurrences in $\overline{\mathcal{A}}$ that respect the partial order $\prec$. Any linearization $\Pi \in \Lambda(\overline{\Pi})$ is a sequential plan for the task. Intuitively, a partial-order plan represents a family of related sequential plans.

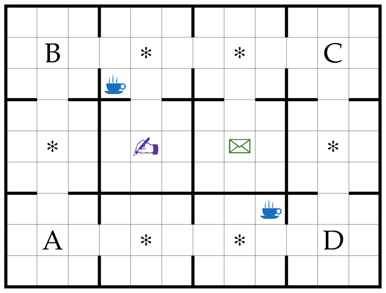


Figure 1: The OFFICEWORLD.

Running Example We consider a version of the OFFICEWORLD domain described by Toro Icarte et al. [10]. The low-level environment is represented by the grid displayed in Figure 1. An agent situated in any cell can try to move in any of the four cardinal directions, succeeding only if the movement does not go through a wall. The symbols in the grid represent events that can be perceived by the agent: it picks up coffee or mail when it reaches the locations marked with blue cups or the green envelope, respectively, or it can deliver what it picked up by reaching the office, marked by the purple writing hand, etc. The locations marked ✻ are places the agent must not step on. The four additional named locations (A, B, C, D) can also be recognized by the agent. An example of a task is that of delivering both mail and coffee to the office. For this task, any optimal policy will need to choose whether to get the coffee or mail first depending on the agent's initial position.

4 Planning Models in RL

Given a low-level environment $\mathcal{E} = \langle S, A, p \rangle$ and the set of associated symbols $\mathcal{P}$ that represent possible observations, a symbolic model for $\mathcal{E}$ is specified as $\mathcal{M} = \langle \mathcal{D}, \alpha \rangle$, where $\mathcal{D} = \langle F, \mathcal{A} \rangle$ is a planning domain and $\alpha\colon \mathcal{A} \to 2^{\mathcal{P}}$ is a function that associates planning actions with sets of observations.

Note that the symbols in F do not directly map to the observations, and that they can therefore be used to model state properties that cannot be directly perceived in the low-level environment. In the OFFICEWORLD running example, some of the symbols in F represent state properties that are non-Markovian in the low-level environment, such as the set of locations that were visited in the past, and whether or not the agent is carrying coffee or mail.

Note that one of the key properties of automated planning systems based on symbolic models is that specifying individual tasks is a very simple problem. We take advantage of this by enabling the description of such tasks—goals in the symbolic models—to be communicated to an RL agent.

From Symbolic Planning Actions to Options Given an environment $\mathcal{E}$ and a corresponding symbolic model $\mathcal{M}$, we can define a set of options that represents relevant transitions that can occur in $\mathcal{M}$. For every planning action $a \in \mathcal{A}$, we build a target set T_a comprised of all the low-level states in which every observation associated with a is perceived. Formally, this is defined in terms of L^{-1}, the preimage of the labeling function: $T_a = L^{-1}(\alpha(a)) \subseteq S$. Then, the set of options can be defined as $O(\mathcal{M}) = \{\langle \pi_a, T_a \rangle \mid a \in \mathcal{A}\}$, where every π_a is a policy trained specifically for reaching the states in T_a. Note that two or more actions in $\mathcal{A}$ may produce the same target set and be represented by a single option.

From Plans to Meta-Controllers We propose that a good way of communicating the high-level tasks to the agent is not in terms of the goals of the task, but rather in terms of high-level plans that achieve them. This allows us to specify tasks that are non-Markovian from the perspective of the low-level environment, such as delivering coffee in the OFFICEWORLD, while simultaneously enabling a natural form of task decomposition.

To actually give a specific high-level plan to an RL agent, we will design a simple reward machine that directly represents the execution of the plan. Reward of 1 will be given only upon completing this execution.

For a sequential plan $\Pi = [a_0, a_1, \ldots, a_n]$, the implementation of the corresponding reward machine is very straightforward and shown in Figure 2a. If instead of a sequential plan we consider a partial-order plan $\overline{\Pi} = \langle \overline{\mathcal{A}}, \prec \rangle$, we need a reward machine that effectively represents all possible orderings of the plan, in a way that any execution of one of them results in receiving a reward of 1 only on the last step. We build this machine by including one state for every possible subset of $\overline{\mathcal{A}}$. This ensures that the machine's executions correctly keep track of which actions have already taken place. Note that some of these states will not be reachable. In practice, we will only generate the states as needed. The possible transitions will be defined by explicitly considering $\Lambda(\overline{\Pi})$, as described in Figure 2b.

Note that the given definition of δ_u does not necessarily imply a function. Consider two actions $a, b \in \overline{\mathcal{A}}$ such that $P = \alpha(a) = \alpha(b)$, that is, actions that result in the same observations. In the OFFICEWORLD, an action to simply visit the office achieves the same event as the action to deliver coffee. Now, if there are two linearizations of $\overline{\Pi}$, Π_a and Π_b, such that $u = \textbf{prefix}(\Pi_a, a) = \textbf{prefix}(\Pi_b, b)$, then either action could be used in the definition of $\delta_u(u, P)$. In such cases, we arbitrarily choose which of the actions to use, as either choice represents a valid linearization of $\overline{\Pi}$.

From Reward Machines to Meta-Controllers Given an environment $\mathcal{E}$ and a symbolic model $\mathcal{M} = \langle \mathcal{D}, \alpha \rangle$, we want to leverage the set of options $O(\mathcal{M})$ to efficiently find a low-level policy for achieving a high-level goal g defined in terms of $\mathcal{D}$. A naive approach consists of finding a sequential plan for g and then attempting to execute the corresponding options in order. A more flexible approach is to learn a meta-controller over the options while using the plan only to specify the reward function. The latter approach also allows the use of a partial-order plan, resulting in a less restrictive reward.

- $U = \{u_0, u_1, \ldots, u_{n+1}\}$
- $\delta_r(u_i, u_j) = \begin{cases} 1 & \text{if } i = n \text{ and } j = n+1 \\ 0 & \text{in any other case} \end{cases}$
- $\delta_u(u_i, P) = \begin{cases} u_{i+1} & \text{if } P = \alpha(a_i) \\ u_i & \text{in any other case} \end{cases}$

(a) Reward machine based on a sequential plan $\Pi = [a_0, a_1, \ldots, a_n]$.

- $U = 2^{\overline{\mathcal{A}}}$, $u_0 = \varnothing$
- $\delta_r(u_i, u_j) = \begin{cases} 1 & \text{if } u_j = \overline{\mathcal{A}} \\ 0 & \text{in any other case} \end{cases}$
- $\delta_u(u, P) = \begin{cases} u \cup \{a\} & \text{if } \exists a \in \overline{\mathcal{A}}, \Pi \in \Lambda(\overline{\Pi}), \\ & \quad \text{s.t. } u = \textbf{prefix}(\Pi, a) \\ & \quad \text{and } P = \alpha(a) \\ u & \text{in any other case} \end{cases}$

(b) Reward machine based on a partial-order plan $\overline{\Pi} = \langle \overline{\mathcal{A}}, \prec \rangle$.

Figure 2: Reward machines defined by plans.

Our third and main approach combines the flexibility of partial-order plans with the simplicity of naively following a plan. Effectively, we train a meta-controller that is a priori restricted to only select among options that correspond to actions that are reasonable within the context of a partial-order plan. In practice, this means that the meta-controller is limited to selecting options that can advance the reward machine toward its goal, an idea introduced by Toro Icarte et al. as a simple way of exploiting the structure in a reward machine [10] .

5 Empirical Evaluation

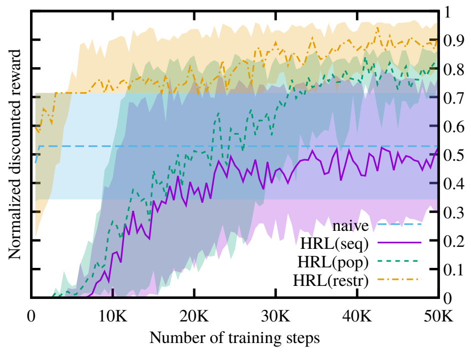
We evaluated our approach by considering two different low-level environments and respective high-level symbolic models. In each case, we defined a set of option candidates using the approach outlined in Section 4. We subsequently pretrained these options, and finally evaluated our approach on a number of new tasks for each environment. For each task we computed sequential and partial-order plans, and built the corresponding reward machines. We compare our results against the use of the same pretrained options in a standard HRL framework, and against the naive approach outlined above. In all cases, the training was done through the Q-learning algorithm.

The first test environment is the OFFICEWORLD running example. Here the high-level actions include visiting any of the named locations, getting coffee, getting mail, delivering either coffee or mail to the office, and delivering both coffee *and* mail to the office. This results in options for going to any of the named locations or to the office, and for getting coffee and getting mail. The actions for delivering to the office are all mapped to the same observation, which occurs whenever the agent reaches the office, and therefore correspond to the same option. For this environment, we tested on tasks consisting of 6 different goals and 10 random initial states for each goal.

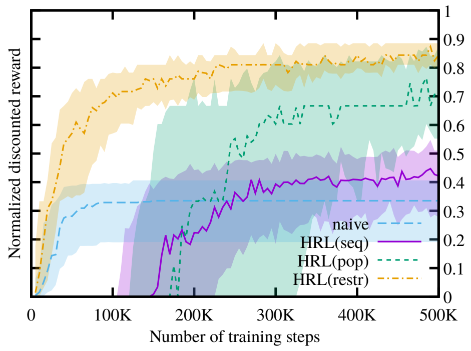
Our second environment is the Minecraft-inspired gridworld described by Andreas et al. [1]. The grid contains raw materials (e.g., wood, iron) and locations where the agent can combine materials to produce refined materials (e.g., wooden planks), tools (e.g., hammer), and goods (e.g., goldware). The high-level actions allow for collecting each of the raw materials, and for achieving the combinations. The types of tasks that we evaluated on include examples such as *"build a bridge"* or *"have a hammer and a pickaxe"*. We ran experiments on 10 different maps, 10 different final-state goals, and 4 random initial states for each combination of map and goal.

In both environments, we pretrained options by generating a small set of random initial states. We simultaneously trained a single policy for each option. The experience observed when training for any given option was used for training all other options, in an off-policy learning setting. This pretraining was restricted to a small number of reinforcement learning steps. In each independent experiment, option policies were continuously refined as the agent continued to interact with the environment.

We used the Fast Downward planning system [4] to quickly produce high-quality sequential plans. To get partial-order plans, we relaxed the sequential plans by solving a mixed integer linear program, following Muise et al. [8].



(a) OFFICEWORLD



(b) MINECRAFTWORLD

Figure 3: Experimental performance obtained on previously unseen tasks.

Results and Discussion We report the results for the OFFICEWORLD and MINECRAFTWORLD environments in Figures 3a and 3b. Each graph displays the performance obtained after training with the labeled algorithm for the specified number of steps. HRL(restr) refers to our main approach; HRL(seq) and HRL(pop) refer to the standard HRL approach, respectively using the rewards specified by the sequential and partial-order plans. In all cases, we report the normalized median discounted reward (using discount $\gamma = 0.9$) obtained on all tasks and several independent trials. We also show the median quintile performance (shaded area). Rewards were normalized by the best value obtained for each task across all independent experiments.

The aim of our experiments was to validate our approach as a method for easily solving unseen tasks. In particular, they show how by just having pretrained options and a high-level plan, we could obtain high-quality policies for the new tasks with little extra learning. Our results were positive and show that the approaches that restrict which options can be used based on the actions presented by the plan produced good policies after very few training steps. The naive method quickly reached a plateau in its performance; but even when compared to this, the other methods we tested needed up to an order of magnitude more training steps to reach comparable results. Our main approach significantly outperformed all other approaches, even after a large number of training steps.

To conclude, we believe that the automatic generation of goal-relevant options and ordering constraints over them—in conjunction with the ability to explicitly represent them in a structured reward function—is one of the key aspects that will enable RL systems to be both taskable and scalable. Moreover, we believe that symbolic planning and knowledge representation techniques can provide the ideal framework for generating such options and constraints. Planning formalisms and algorithms support a plethora of different domain and solution properties, such as temporally-extended goals, preferences, diverse plans, numeric constraints, and more. All of these could be used for representing rich reinforcement learning tasks, while preserving useful structure that can be exploited when learning policies.

References

[1] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *ICML*, volume 70 of *PMLR*, pages 166–175, 2017.

[2] M. J. Grounds and D. Kudenko. Combining reinforcement learning with symbolic planning. In *AAMAS III*, volume 4865 of *LNCS*, pages 75–86, 2008.

[3] M. Grześ and D. Kudenko. Plan-based reward shaping for reinforcement learning. In *IS*, volume 2, pages 10-22–10-29, 2008.

[4] M. Helmert. The Fast Downward planning system. *JAIR*, 26:191–246, 2006.

[5] S. James, B. Rosman, and G. Konidaris. Learning to plan with portable symbols. In *PAL@ICML/IJCAI/AAMAS*, 2018.

[6] G. Konidaris, L. P. Kaelbling, and T. Lozano-Pérez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *JAIR*, 61:215–289, 2018.

[7] D. Lyu, F. Yang, B. Liu, and S. Gustafson. SDRL: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *AAAI*, 2019.

[8] C. J. Muise, J. C. Beck, and S. A. McIlraith. Optimal partial-order plan relaxation via MaxSAT. *JAIR*, 57:113–149, 2016.

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

[10] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *ICML*, volume 80 of *PMLR*, pages 2112–2121, 2018.

[11] F. Yang, D. Lyu, B. Liu, and S. Gustafson. PEORL: integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. In *IJCAI*, pages 4860–4866, 2018.

[12] A. Zhang, S. Sukhbaatar, A. Lerer, A. Szlam, and R. Fergus. Composable planning with attributes. In *ICML*, volume 80 of *PMLR*, pages 5837–5846, 2018.