Symbolic Plans as High-Level Instructions for Reinforcement Learning

Unabridged Version

Authors are anonymous

Abstract

Reinforcement learning (RL) agents seek to maximize the cumulative reward obtained when interacting with their environment. Users define tasks or goals for RL agents by designing specialized reward functions such that maximization aligns with task satisfaction. This work explores the use of high-level symbolic action models as a framework for defining final-state goal tasks and automatically producing their corresponding reward functions. We also show how automated planning can be used to synthesize high-level plans that can guide hierarchical RL (HRL) techniques towards efficiently learning adequate policies. We provide a formal characterization of taskable RL environments and describe sufficient conditions that guarantee we can satisfy various notions of optimality (e.g., minimize total cost, maximize probability of reaching the goal). In addition, we do an empirical evaluation that shows that our approach converges to near-optimal solutions faster than standard RL and HRL methods and that it provides an effective framework for transferring learned skills across multiple tasks in a given environment.

1 Introduction

Reinforcement learning (RL) methods represent the state of the art for solving complex continuous control problems in robotics (Van Hoof et al. 2015; Kumar, Todorov, and Levine 2016; Kumar et al. 2016; Falco et al. 2018; Andrychowicz et al. 2018; Akkaya et al. 2019). For instance, the OpenAI lab recently showed that model-free RL can be used to learn to control a human-like robot hand to purposefully manipulate complex objects, such as a Rubik's Cube (Akkaya et al. 2019). The strength of model-free RL comes from being able to learn policies that maximize an external reward signal by directly interacting with the environment—without requiring a predefined model of the complex physics equations that control it (nor trying to learn them).

This generality comes with a cost, though. As the environment dynamics and reward structures are initially unknown, RL methods mostly rely on random exploration to collect rewards and then improve their current policy accordingly. As such, these methods are *sample inefficient* (i.e., require billions of interactions with the environment before learning better-than-random policies). Further, these systems are typically not *taskable*. If you would like an RL agent to solve task A, then you would have to program a reward function such that its optimal policy would solve A. If, later on, you would like the agent to perform task B, then you would have to program a new reward function for B and the RL agent would have to learn a brand new policy for B from scratch—which is a problem known as *transfer learning* (Taylor and Stone 2009). A number of approaches have been proposed to address these shortcomings including efforts to learn hierarchical representations (Dietterich 2000), to define *options* or macro-actions that can be used by the RL system (Sutton, Precup, and Singh 1999), or to learn skills that are independent of the state space where they were learned (Konidaris and Barto 2007).

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 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 a formal description of actions that cause those properties to change in predictable ways—while leaving (possibly complex) low-level aspects of the environment (e.g., the frame by frame outcome of dropping a pen) unspecified.

As a result, our AI agent gets the best of both worlds: (1) it is taskable as the user can define tasks as goal conditions in the symbolic domain (and a reward function is automatically computed for such a task), (2) it improves sample efficiency as the high-level plans can be used for transferring learning from previously learned policies, and (3) it can learn complex low-level control policies as it relies on model-free RL to accommodate for all the information missing in the high-level model. To achieve this, we build on ideas for *learning by instructions* in RL (Andreas, Klein, and Levine 2017; Toro Icarte et al. 2018a; 2018c). That work shows that sample efficiency can be improved if a *manually* generated description of the task is given to the agent. In this work, we propose to *automatically* generate useful instructions for RL

agents using the high-level model of the environment and describe an approach—based on hierarchical reinforcement learning (HRL)—that exploits such instructions. We compare our approach to standard forms of HRL and show that the combination of high-level symbolic planning and low-level reinforcement learning is an effective method for specifying tasks to RL agents and, more importantly, for learning high-quality policies—for previously unseen tasks—up to an order of magnitude faster than using standard RL.

Note that the idea of combining high-level symbolic planning with low-level RL has a long history. Some well-known examples include work done by Ryan (2002), Grounds and Kudenko (2008), Grześ and Kudenko (2008), Yang et al. (2018), and Lyu et al. (2019). Informed by this previous work, we contribute a formal characterization of a relevant problem, which we call *taskable RL*, and a novel approach to transfer learned policies and guide exploration in RL based on high-level plans. Building on this, we provide theoretical analysis regarding sufficient conditions for ensuring our approach satisfies various notions of optimality and show empirical results that validate the efficiency of our approach.

2 Preliminaries

In this section we establish relevant notation and review key aspects of reinforcement learning and automated planning. In addition, we describe a simple running example.

Reinforcement Learning

For the purposes of this work, we will say that the environment in which an RL agent acts is formalized as an *Markov Decision Process (MDP)* $M = \langle S, A, r, p, \gamma \rangle$, where S is its set of states, A is the set of available actions, $r\colon S \times A \times S \to \mathbb{R}$ is its reward function, $p(s_{t+1}|s_t, a_t)$ is its transition probability distribution, and $\gamma \in (0, 1]$ is the discount factor. A policy for M is defined as a probability distribution $\pi(a|s)$ that establishes the probability of the agent taking action a given that its current state is s. Then, the RL problem consists of finding an optimal policy π^* that maximizes the expected discounted future reward obtained from all $s \in S$:

$$\pi^* = \arg\max_{\pi} \sum_{s \in S} v_\pi(s)$$

where $v_\pi(s)$ is known as the *value function* and models the expected discounted future reward obtained when starting at state $s \in S$ and selecting actions according to π:

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s \right]$$

Crucially, the agent is not given access to the model of the environment (i.e., p and r). Instead, the agent must learn optimal behaviour by interacting with the environment. At any time step, the agent observes the current state $s \in S$ and executes an action $a \in A$ according to its current policy π. As a result, the environment returns the next state $s' \in S$ (sampled from $p(s'|s, a)$) and an immediate reward $r' = r(s, a, s')$. The experience (s, a, r', s') is then used by the agent to improve its current policy π. The main distinctions between RL methods are on how to select the next action and how to improve the current policy using sampled experiences.

For example, *q-learning* (Watkins and Dayan 1992) is an RL approach that learns optimal policies (in the limit) by using sampled experiences to estimate the optimal q-function $q^*(s, a)$ for every state $s \in S$ and action $a \in A$. The optimal q-function $q^*(s, a)$ is equal to the expected discounted future reward received by performing action a in state s and following an optimal policy afterwards. Given an experience (s, a, r', s'), the q-value estimate $\tilde{q}(s, a)$ is updated as follows:

$$\tilde{q}(s, a) \xleftarrow{\alpha} \left(r' + \gamma \max_{a' \in A} \tilde{q}(s', a') \right),$$

where $x \xleftarrow{\alpha} y$ is shorthand notation for $x \leftarrow x + \alpha \cdot (y - x)$ and $\alpha \in (0, 1]$ is a hyperparameter called the *learning rate*. Note that an optimal policy π^* can be extracted from $q^*(s, a)$ by always selecting the action $a \in A$ with the largest q-value for the current state s. To explore the environment, q-learning uses an ϵ-greedy exploratory policy. This is, it selects a random action with probability ϵ and the action with the largest $\tilde{q}(s, \cdot)$ value with probability $1 - \epsilon$.

Temporal Abstraction Standard RL techniques are faced with significant issues when applied on large-scale problems. In practical terms, RL algorithms need a large amount of training steps in order to converge. A popular technique for dealing with these issues is to consider temporally-extended macro-actions that represent useful high-level behaviours. In particular, the options framework proposes the use of policies that are trained for achieving specific high-level behaviours, coupled with well-defined criteria for their termination (Sutton, Precup, and Singh 1999). Given the current state, an agent acting within this framework chooses one among the high-level options and executes its policy until it terminates. In order to select which options will be executed, the agent has a higher order policy or meta-controller which can also be learned through RL.

For a given environment $M = \langle S, A, r, p, \gamma \rangle$, an option is formalized as $o = \langle \pi_o, r_o, T_o \rangle$ where π_o is the option's policy, r_o is a reward function used for training option o, and $T_o \subseteq S$ is the set of states where the option terminates.

In the options framework, a set of predefined options O is given to the agent. Then, learning occurs at two levels. At the options' level, the policies are updated using the usual experiences s, a, s', and the reward that comes from r_o. If q-learning were used, then one q-function $\tilde{q}_o$ would be learned per option $o \in O$ (where $\tilde{q}_o(s, a) = 0$ for all $s \in T_o$) and all of them would be updated as follows:

$$\tilde{q}_o(s, a) \xleftarrow{\alpha} \left(r_o(s, a, s') + \gamma \max_{a' \in A} \tilde{q}_o(s', a') \right).$$

At the level of the meta-controller, the learning task consists of finding an optimal policy $\pi^*(o|s)$ to select the next option $o \in O$ to execute given the current state $s \in S$. Learning at the level of the meta-controller occurs only when the current option terminates. If option o was executed in state s and terminated in state s' after executing k actions a_t and receiving

k immediate rewards r_t, then a q-learning algorithm would use this experience to update the meta-controller estimate of the q-function $\tilde{q}$ as follows:

$$\tilde{q}(s,o) \xleftarrow{\alpha} \left(\sum_{t=1}^{k} \gamma^{k-1} r_t + \gamma^k \max_{o' \in O} \tilde{q}(s', o') \right).$$

Symbolic Planning

We specify planning domains in terms of a tuple $\mathcal{D} = \langle \mathcal{F}, \mathcal{A} \rangle$. $\mathcal{F}$ 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 $\mathcal{F}$, so that state $\mathcal{S} \subseteq \mathcal{F}$ represents the situation in which the fluents in $\mathcal{S}$ are all true and those not in $\mathcal{S}$ are false. An action $a \in \mathcal{A}$ is defined by a tuple $a = \langle \mathit{pre}^+, \mathit{pre}^-, \mathit{eff}^+, \mathit{eff}^- \rangle$ where each of its elements is a subset of $\mathcal{F}$. Here, pre^+ and pre^- (eff^+ and eff^-) are disjoint and represent the positive and negative preconditions (effects) of a, respectively. We say $a \in \mathcal{A}$ is applicable over state $\mathcal{S}$ when $\mathit{pre}^+ \subseteq \mathcal{S}$ and $\mathit{pre}^- \cap \mathcal{S} = \varnothing$. The result of applying a over state $\mathcal{S}$ is a state given by the function $\delta(\mathcal{S}, a) = \left(\mathcal{S} \setminus \mathit{eff}^-\right) \cup \mathit{eff}^+$. When a is not applicable over $\mathcal{S}$, $\delta(\mathcal{S}, a)$ is undefined.

A planning task for domain $\mathcal{D}$ is formalized as $\mathcal{T} = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{I}$ is an initial state and $\mathcal{G}$ is the task's goal condition. The goal $\mathcal{G} = \langle \mathcal{G}^+, \mathcal{G}^- \rangle$ where $\mathcal{G}^+$ and $\mathcal{G}^-$ is a pair of disjoint subsets of $\mathcal{F}$ that, respectively, represent positive and negative subgoals. Any state $\mathcal{S}$ such that $\mathcal{G}^+ \subseteq \mathcal{S}$ and $\mathcal{G}^- \cap \mathcal{S} = \varnothing$ is said to be a goal state. A sequence of actions $\Pi = [a_0, a_1, \cdots, a_n]$ is known as a sequential plan for a task when it is possible to sequentially apply the actions starting at $\mathcal{I}$, and doing so reaches a goal state.

Partial-Order Plans Partial-order plans generalize sequential plans by relaxing the ordering condition over the actions. A partial-order plan is represented by 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. Note that a plan may require using the same action twice. As such, two action occurrences in $\overline{\mathcal{A}}$ may be repetitions of the same action, distinguished only for the purposes of defining the particular partial-order plan.

Running Example

We consider a version of the OFFICEWORLD domain described by Toro Icarte et al. (2018c). The low-level environment is represented by the grid displayed in Figure 1. A robot 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 important features of the environment and different events occur whenever the robot reaches a marked cell. Specifically, the robot will pick up coffee or mail when it reaches the locations marked with blue cups or the green envelope, respectively. If the robot is already carrying coffee or mail, it will deliver it to the office upon reaching the cell marked with the purple writing hand. The robot can also visit the four named locations (A, B, C, D). The locations marked ✻ are places that the robot should not step on, and doing so results in a large penalty or negative reward (-10). All other successful actions result in a small penalty (-1), whereas failed actions (i.e., attempting to move through a wall) are heavily penalized (-10).

3 Taskable Reinforcement Learning

One of the great advantages of symbolic planning is that specifying new simple tasks for a given domain model is very easy. It is with this in mind that we define the problem of taskable RL, where final-state goal tasks can be specified trivially for a given RL environment.

To define goals for tasks in such an environment, we assume the existence of a set of high-level propositions, P, and a labeling function $L\colon S \to 2^P$ that establishes a mapping between low-level states and high-level propositions. These propositions are supposed to represent important state properties that may affect the outcomes of actions or their costs, or that may be of significance to an end user of the system. Finally, we also assume the existence of a constant R that establishes a reward bonus received by the agent when it accomplishes a task. With this, we define taskable RL environments and their associated final-state goal tasks.

Definition 1 (Taskable RL Environment)**.** A *taskable reinforcement learning environment* is defined by a tuple $E = \langle S, A, r, p, \gamma, P, L, R \rangle$, where $\langle S, A, r, p, \gamma \rangle$ is an MDP, P is a set of propositional variables, $L\colon S \to 2^P$ is a labeling function, and $R \in \mathbb{R}$ is a parameter called the *goal reward*.

Definition 2 (Final-state Goal Task)**.** A *final-state goal task* for taskable RL environment $E = \langle S, A, r, p, \gamma, P, L, R \rangle$ is defined as a tuple $G = \langle G^+, G^- \rangle$ where its elements are disjoint subsets of P. We say a state $g \in S$ is a goal state when $G^+ \subseteq L(g)$ and $G^- \cap L(g) = \varnothing$. We denote the set of all goal states as $\mathbb{G}$. The objective for this task is to find the optimal policy for the MDP $M_G = \langle S, A, r_G, p_G, \gamma \rangle$, where r_G and p_G are defined as follows:

$$r_G(s,a,s') = \begin{cases} R + r(s,a,s') & \text{if } s' \in \mathbb{G} \text{ and } s \neq s' \\ 0 & \text{if } s' \in \mathbb{G} \text{ and } s = s' \\ r(s,a,s') & \text{otherwise} \end{cases}$$

$$p_G(s'|s,a) = \begin{cases} 0 & \text{if } s \in \mathbb{G} \text{ and } s \neq s' \\ 1 & \text{if } s \in \mathbb{G} \text{ and } s = s' \\ p(s'|s,a) & \text{otherwise} \end{cases}$$

Intuitively, the goal conditions are used for defining *fictional* terminal states in the environment. The modified transition probabilities ensure that exiting a goal state is impossible. In turn, the modified reward function ensures that a reward bonus is given when a goal state is first reached, and that no further reward can be accrued after that.

The main motivation behind Definitions 1 and 2 is to allow end-users to define tasks for RL agents with minimal effort. In the same spirit, the main guarantee that we should provide to that end-user is that the RL agent will optimize its

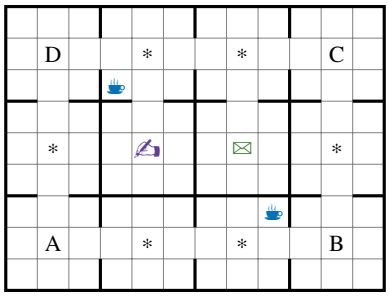


Figure 1: The OFFICEWORLD running example.

behaviour towards actually accomplishing their tasks. Interestingly, whether a taskable RL environment provides such a guarantee depends on how it defines r, γ, and R.

Important Properties of Taskable RL

The purpose of this section is to identify sufficient conditions under which optimal low-level policies would reach the high-level final-state goal defined by the user. We also provide some understanding of the quality of such policies. To do so, we begin by introducing the notion of *proper policies*, which comes from the stochastic shortest path literature (Eaton and Zadeh 1962; Bertsekas and Tsitsiklis 1991):

Definition 3 (Proper Policy)**.** Given a final-state goal task G, we say a policy π for its corresponding MDP $M_G = \langle S, A, r_G, p_G, \gamma \rangle$ is a *proper policy* if the probability of eventually reaching a goal state by starting at any $s \in S$ and following π is 1. A policy that is not proper is said to be improper.

Proper policies formalize the intuition behind the main property that we would expect from any taskable RL environment: the optimal low-level policies are proper policies for reaching the high-level goals defined by the user. With this, we can establish three theorems that identify sufficient conditions under which such a property holds (Theorems 1 and 2), or a best-case scenario for environments where the property does not hold (Theorem 3). The conditions outlined by the theorems are the ones that should be taken into consideration when designing taskable RL environments.

Theorem 1 (Cost based problems)**.** *Let taskable RL environment $E = \langle S, A, r, p, \gamma, P, L, R \rangle$ be such that r is of the restricted form $r\colon S \times A \times S \to \mathbb{R}^-$, $\gamma = 1$, and $R = 0$. Let G be a final-state goal task for E. If there exists at least one proper policy for M_G, then the optimal policy is a proper policy that minimizes the expected cost of reaching a goal state.*

Proof sketch. First, assume that the optimal policy, π^*, is improper. Then, there exists at least one state $s \in S$ such that the probability of never reaching a goal state when following π^* from s is greater than 0. Since rewards are negative everywhere except when reaching a goal state, this means that the expected future reward obtained by following π^* from s tends to $-\infty$ and, therefore, the value obtained from summing over all states in S also tends to $-\infty$. Note, however, that any proper policy has, for any state, finite expected future reward. Then, any proper policy has higher expected reward than π^*, and since we know that proper policies exist this leads to a contradiction that proves π^* must be proper. Now, by definition, π^* maximizes the cumulative expected reward. Since all traces must eventually reach a goal state, and no reward is given after this, then π^* minimizes the expected cost of reaching a goal state. □

Theorem 2 (Goal reward problems)**.** *Let taskable RL environment $E = \langle S, A, r, p, \gamma, P, L, R \rangle$ be such that $\gamma < 1$, $R = 1$, and $r(s, a, s') = 0$ for any s, a, s'. Let G be a final-state goal task for E. If there exists at least one proper policy for M_G, then the optimal policy is a proper policy that minimizes the expected number of steps before reaching a goal state.*

Proof sketch. By following a similar argument to the one used in the proof of Theorem 1, we know that π^* must be a proper policy. Again, by definition, we know that π^* maximizes the cumulative discounted expected reward. Since reward is only given when a goal is reached, and it is discounted with $\gamma < 1$, we immediately know that π^* minimizes the expected steps before reaching a goal. □

Theorem 3 (Max-prob problems)**.** *Let taskable RL environment $E = \langle S, A, r, p, \gamma, P, L, R \rangle$ be such that $\gamma = 1$, $R = 1$, and $r(s, a, s') = 0$ for any s, a, s'. Let G be a final-state goal task for E. Here, the optimal policy for M_G maximizes the probability of reaching a goal state.*

Proof sketch. Define a function $g\colon S \to \mathbb{R}$ as follows:

$$g(s) = \begin{cases} 1 & s \in \mathbb{G} \\ v^*(s) & \text{otherwise} \end{cases}$$

Where, v^* is the value function for the optimal policy of M_G. Since $v^*(s) = 0$ at every $s \in \mathbb{G}$ and v^* is optimal, we immediately know that g is optimal. We will show that $g(s)$ represents the maximum achievable probability of eventually reaching a goal state from s. The proposition is trivial for $s \in \mathbb{G}$. For $s \notin \mathbb{G}$, we know:

$$g(s) = v^*(s) = \max_{a \in A} \sum_{s' \in S} p_G(s'|s, a) \cdot (r(s, a, s') + v^*(s'))$$

We can split the sum over S into a sum over $\mathbb{G}$ and one over $S \setminus \mathbb{G}$. Note that for every $s' \in \mathbb{G}$ we know that $r(s, a, s') = 1$ and $v^*(s') = 0$, and that for every $s' \notin \mathbb{G}$ we know that $r(s, a, s') = 0$ and $v^*(s') = g(s')$. Then, we can rewrite as:

$$g(s) = \max_{a \in A} \sum_{s' \in \mathbb{G}} p_G(s'|s, a) + \sum_{s' \notin \mathbb{G}} p_G(s'|s, a) \cdot g(s')$$

This corresponds exactly to the functional equation that defines $g(s)$ as the maximum probability of eventually reaching a goal state from s. □

Finally, it is worth noting that other combinations of forms of r and values of γ and R may occasionally result in undesirable behaviour. For instance, allowing for positive rewards might cause the agent to prefer to collect those rewards instead of achieving the goals set by the user. On the

other extreme, using only negative rewards but $\gamma < 1$ might cause the agent to prefer to stay in an area where it will pay a small penalty for eternity instead of paying the possibly high immediate penalty required for achieving a goal state. The same behaviour might occur if we use only negative rewards, $\gamma = 1$, and a finite horizon.

Taskable RL and the Running Example

We now explain how to represent the OFFICEWORLD running example as a taskable environment. Given that its low-level states must be able to distinguish the current position of the robot, whether it is carrying coffee or mail, whether it has already delivered coffee or mail, and whether it has already visited the office or any of the other named locations, then we can define a set of high-level propositions that contains most of the same information, but omits the exact position of the robot, including instead some propositions that represent if the robot is currently at some of the marked locations:

$$P = \{\texttt{have-coffee}, \texttt{have-mail}, \texttt{delivered-coffee}, \texttt{delivered-mail}, \texttt{visited-office}, \texttt{visited-A}, \texttt{visited-B}, \texttt{visited-C}, \texttt{visited-D}, \texttt{at-office}, \texttt{at-A}, \texttt{at-B}, \texttt{at-C}, \texttt{at-D}\}.$$

This would allow an end-user to easily define tasks for the agent as goal conditions over these propositions. For example, we could ask the agent to deliver both coffee and mail to the office by defining the following goal condition: $\langle\{\texttt{delivered-coffee}, \texttt{delivered-mail}\}, \varnothing\rangle$.

Finally, as the objective of a final-state goal task is still to learn policies for a particular MDP, it can be potentially solved by any RL algorithm. Given, however, that we know of the existence of the high-level propositions in P and the labeling function L, we devise a specialized RL algorithm based on symbolic planning that can learn effective policies considerably faster than off-the-shelf RL methods.

4 Planning Models in RL

The general idea is to use planning models and solutions computed for them as guidance for solving RL tasks in the low-level environment. To do so, we associate symbolic models to taskable RL environments. Given a taskable environment $E = \langle S, A, r, p, \gamma, P, L, R\rangle$, a symbolic model for E is specified as $\mathcal{M} = \langle \mathcal{D}, \alpha\rangle$, where $\mathcal{D} = \langle \mathcal{F}, \mathcal{A}\rangle$ is a planning domain with $\mathcal{F} = P$ and $\alpha\colon \mathcal{A} \to 2^P \times 2^P$ is a function that associates planning actions with conditions over P.

We will use α to associate finite-state goal tasks for the taskable RL environment E with the planning actions. Each such task defines an option. Whenever the task is accomplished, the option terminates. Formally, given a condition $C = \langle C^+, C^-\rangle$, we can define the set of all low-level states that satisfy it as $T(C) = \{s \in S \mid C^+ \subseteq L(s), C^- \cap L(s) = \varnothing\}$. Then, for each planning action $a \in \mathcal{A}$, we can define an associated option with termination set $T_a = T(\alpha(a))$ and reward function $r_a = r_{\alpha(a)}$ (see Definition 2). Finally, we will define an option set O consisting of one option for each distinct $\alpha(a)$ generated this way. Note that two or more planning actions may be associated with the same option.

Model Quality

Intuitively, we would expect that following a high-level plan for a task by sequentially executing its associated low-level option policies should reach a goal state with probability 1. However, to provide such a formal guarantee we need to assume certain properties about the relation between the planning models and the low-level environment. These properties are formalized in the following definition:

Definition 4 (Consistency of symbolic models)**.** We say a symbolic model $\mathcal{M} = \langle \mathcal{D}, \alpha\rangle$ is *consistent* with a taskable environment $E = \langle S, A, r, p, \gamma, P, L, R\rangle$ if $\mathcal{D} = \langle \mathcal{F}, \mathcal{A}\rangle$, $\mathcal{F} = P$, $\alpha\colon \mathcal{A} \to 2^P \times 2^P$, and every optimal option policy $\pi^*_{\alpha(a)}$ associated with any planning action $a \in \mathcal{A}$ terminates with probability 1 and respects $\delta(\mathcal{S}, a)$. This is, if $\pi^*_{\alpha(a)}$ initiates in state $s_i \in S$ and might terminate in state $s_t \in S$, then $L(s_t) = \big(L(s_i) \setminus \mathit{eff}_a^-\big) \cup \mathit{eff}_a^+$ for every state $s_i \in S$ where a is applicable ($\mathit{pre}^+ \subseteq L(s_i)$ and $\mathit{pre}^- \cap L(s_i) = \varnothing$).

Note that defining consistent models is non-trivial since it requires considering the behaviour of optimal policies acting in a possibly complex low-level environment. As such, we do not assume that models are necessarily consistent throughout the rest of the paper (except when analyzing formal properties of our approach).

Planning Models in the Running Example

In the running example domain, we consider the following set of high-level actions:

$$\mathcal{A} = \{\texttt{get-mail}, \texttt{get-coffee}, \texttt{deliver-mail}, \texttt{deliver-coffee}, \texttt{visit-A}, \texttt{visit-B}, \texttt{visit-C}, \texttt{visit-D}, \texttt{visit-office}\}.$$

Figure 2 shows the preconditions and effects for two of these actions. In addition, it shows the termination conditions for their corresponding options. Note, however, that the final-state goal task associated with the `deliver-coffee` action makes no reference to the coffee itself. This is by design and it serves an important purpose: the same final-

get-coffee:

pre^+:	$\varnothing$
pre^-:	$\varnothing$
eff^+:	{have-coffee}
eff^-:	$\varnothing$
α(get-coffee):	$\langle\{\texttt{have-coffee}\}, \varnothing\rangle$

deliver-coffee:

pre^+:	{have-coffee}
pre^-:	$\varnothing$
eff^+:	{delivered-coffee, at-office}
eff^-:	{have-coffee}
α(deliver-coffee):	$\langle\{\texttt{at-office}\}, \varnothing\rangle$

Figure 2: Example actions in the OFFICEWORLD, including associated option termination.

state goal—and option—will be associated with other planning actions that have different preconditions and effects but are realized through the same low-level actions as `deliver-coffee` (e.g., the `deliver-mail` action).

For the example task of delivering both coffee and mail, we consider a planning task with initial state $\mathcal{I} = \varnothing$ and goal $\mathcal{G} = \langle \{\texttt{delivered-coffee}, \texttt{delivered-mail}\}, \varnothing \rangle$. Here, a reasonable solution plan might be the following:

$$\Pi = [\texttt{get-coffee}, \texttt{get-mail}, \texttt{deliver-coffee}, \texttt{deliver-mail}].$$

Note that the actual number of low-level actions needed to execute the high-level actions depends on the actual position of the robot in the grid. This means that—even if we have optimal low-level policies for performing each of the high-level actions—the plan may be optimal or suboptimal depending on the exact low-level initial state.

5 Executing Plans

For a given symbolic model and planning task, we can easily compute a sequential plan by using an off-the-shelf planner. Such a plan can subsequently be used as a naive meta-controller for an HRL system by directly executing—in the provided order—each of the options associated with the actions in the plan. For the example task and corresponding plan, this would result in first executing a policy that achieves `have-coffee`. After this, the system would execute a policy that terminates when `have-mail` becomes true. It would continue by executing the option policy associated with the `deliver-coffee` action, which achieves `at-office`. At this point, the system would attempt to execute the policy associated with `deliver-mail`. Since `at-office` is already true in the current state, the policy would immediately terminate.

This approach has a number of issues, though. First, it may be significantly better to execute the actions in a different order than the strict one defined by the plan. In the example, it may be better to attempt getting the mail before the coffee. Second, if the model is inconsistent with the environment, then executing an option policy might affect propositions that are not mentioned in the description of its high-level action—possibly invalidating the current plan.

To address the first problem, we consider the use of partial-order plans instead of sequential ones. The execution of a partial-order plan $\overline{\Pi} = \langle \overline{\mathcal{A}}, \prec \rangle$ consists of selecting, at every step, some action occurrence $a \in \overline{\mathcal{A}}$ such that there is no other action occurrence $a' \prec a$ that has not already been executed. Thus, we can use a partial-order plan as a meta-controller that is restricted to choose among the options that correspond to the high-level actions that can advance the execution of the plan. This meta-controller can be further trained in order to eventually learn the optimal way to execute the partial-order plan.

In the running example, the plan Π can be relaxed into a partial-order plan with the same four actions and the following ordering constraints:

$$\texttt{get-coffee} \prec \texttt{deliver-coffee}$$
$$\texttt{get-mail} \prec \texttt{deliver-mail}.$$

In the initial state, the `get-coffee` and `get-mail` actions are valid for the execution of the plan. The meta-controller policy will, then, learn to select one of them based on the low-level state, which includes relevant information about the position of the robot.

To address the second issue, where an option policy affects some propositions other than those mentioned in the corresponding high-level action's effects, we use the notion of regression of a plan to identify—for each action in the plan—the conditions that ensure the plan is still valid. Formally, the regression of a condition given as a pair of positive and negative conditions $\mathcal{C} = \langle \mathcal{C}^+, \mathcal{C}^- \rangle$ for an action $a = \langle \mathit{pre}^+, \mathit{pre}^-, \mathit{eff}^+, \mathit{eff}^- \rangle$ such that $\mathit{eff}^+ \in \mathcal{C}^+$ and $\mathit{eff}^- \in \mathcal{C}^-$ is defined as

$$\mathcal{R}(\mathcal{C}, a) = \langle (\mathcal{C}^+ \setminus \mathit{eff}^+) \cup \mathit{pre}^+, (\mathcal{C}^- \setminus \mathit{eff}^-) \cup \mathit{pre}^- \rangle.$$

The regression of a goal $\mathcal{G}$ for a plan Π, denoted $\mathcal{R}^\star(\mathcal{G}, \Pi)$, is just the repeated application of $\mathcal{R}$ backwards through all actions in the plan starting from $\mathcal{G}$. For example, if $\Pi = [a_0, a_1, a_2]$ then $\mathcal{R}^\star(\mathcal{G}, \Pi) = \mathcal{R}(\mathcal{R}(\mathcal{R}(\mathcal{G}, a_2), a_1), a_0)$.

Note that $\mathcal{R}^\star(\mathcal{G}, \Pi)$ represents the necessary and sufficient conditions that a state $\mathcal{I}$ must satisfy for Π to be a valid plan for reaching $\mathcal{G}$ from $\mathcal{I}$. Moreover, given $\Pi = [a_0, a_1, \cdots, a_n]$, we have that $\mathcal{R}^\star(\mathcal{G}, [a_i, a_{i+1}, \cdots, a_n])$ represents the conditions that need to be satisfied before applying the suffix $[a_i, a_{i+1}, \cdots, a_n]$ of Π. Following this idea, Fritz and McIlraith (2007) defined an execution monitoring policy that, given a state, checks for the shortest suffix such that its condition is satisfied and returns the first action in it. This system takes advantage of serendipitous unexpected outcomes, skipping parts of the plan that become unnecessary, and can recover from some negative outcomes.

This approach can be extended to work for partial-order plans, effectively regressing through all possible linearizations without having to explicitly construct them (Muise, McIlraith, and Beck 2011). Intuitively, the regression exploits the structure of the partial order, taking advantage of the shared conditions and actions across multiple linearization suffixes. In our work, we use this approach and the simple sequential plan regression approach to produce high-level policies to guide the selection of options.

A generic overview of our approach is shown in Algorithm 1. The algorithm maintains a policy π for the meta-controller, and keeps references to the option policies through the high-level actions ($a.\pi$ for every $a \in A$). It first computes a high-level plan (Π), and keeps track of the low-level state (s) and the high-level action currently being executed (current). If no high-level action has been selected, it queries the plan for the set of actions that could be applied (line 8) and queries the meta-controller policy for the best action out of that set . The algorithm proceeds by evaluating the policy of the option associated with the current high-level action to get a low-level action (line 11). After executing the action in the environment and receiving reward, it updates all the option policies based on that experience (line 16). If the option terminates, the algorithm updates the meta-controller policy and the state of the plan (line 19).

Algorithm 1: Reinforcement Learning guided by a partial-order plan.

Input: $\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G}, O$
Initialize π and $a.\pi$ for $a \in \mathcal{A}$
$\Pi \leftarrow \text{plan}(\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$
foreach *episode* **do**
  $s \leftarrow$ get state from environment
  current $\leftarrow$ *None*
  foreach *step* **do**
    if *current is None* **then**
      HLAs $\leftarrow \Pi.next()$
      current $\leftarrow \pi(s, \text{HLAs})$
      $s_0 \leftarrow s, R \leftarrow 0$
    a $\leftarrow$ current.$\pi(s)$
    $s' \leftarrow apply(s, a)$
    $r' = r(s, a, s')$
    $R \leftarrow R + r'$
    foreach *option* $o \in O$ **do**
      o.π.update(s, a, r', s')
    if *current.terminates(*s'*)* **then**
      π.update$(s_0, \text{current}, R, s')$
      $\Pi.advance(\text{current})$
      current $\leftarrow$ *None*
    $s \leftarrow s'$

In what remains of this paper, we consider different instances of this approach. The basic approach in which we only use a sequential plan is denoted `seq`. An approach that first relaxes the sequential plan into a partial-order plan is called `pop`. For both cases, we also consider the use of regression-based plan execution monitoring as described above ($\texttt{seq}_\texttt{m}$ and $\texttt{pop}_\texttt{m}$). Finally, we compare and contrast against two basic benchmark approaches. The first is the direct use of q-learning over the low-level environment (`ql`). The second is the use of the options framework over the set of options associated with the model (`hrl`).

Theoretical Analysis

Before analyzing the properties of our approaches, it is important to understand the role that hierarchies play in RL. Hierarchies impose constraints over policies. These constraints effectively prune otherwise feasible policies from consideration, allowing RL agents to focus their learning efforts on the smaller set of hierarchically consistent policies. Unfortunately, hierarchies might (unintentionally) prune optimal policies too, denying the agent the possibility of converging to globally optimal policies. As such, our HRL methods can at best converge to a hierarchically optimal policy—which were first introduced by Dietterich (2000):

Definition 5 (Hierarchically optimal policies)**.** A hierarchically optimal policy for an MDP M is a policy that achieves the highest cumulative reward among all policies consistent with the given hierarchy.

We now analyze properties of hierarchically optimal policies for `hrl`, `pop`, and `seq`. We begin by identifying sufficient conditions under which those policies are proper policies for reaching a goal state and then we compare their quality w.r.t. the amount of reward they get.

Theorem 4 (Optimal policies are proper policies)**.** *Let* $E = \langle S, A, r, p, \gamma, P, L, R\rangle$ *be a taskable RL environment where:*

- $r\colon S \times A \times S \to \mathbb{R}^-$, $\gamma = 1$, *and* $R = 0$, *or*
- $r(s, a, s') = 0$ *for any* s, a, *and* s', *and* $R = 1$,

and $\mathcal{M} = \langle \mathcal{D}, \alpha\rangle$ *be a consistent symbolic model for* E. *Let* G *be a final-sate goal task for* E. *If there exists a high-level sequential plan* Π *that solves the planning task* $\langle \mathcal{D}, L(s), G\rangle$ *from the initial low-level state* s, *then all hierarchically optimal policies for* `hrl`, `pop`, *and* `seq` *reach a goal state from* s *with probability 1.*

Proof sketch. Since $\mathcal{M}$ is a consistent symbolic model for E, we know that the policy π resulting from sequentially executing the optimal low-level policies π^*_a of each macro action in the plan Π would reach a goal state with probability 1 from the initial state s. Moreover, the conditions imposed on the taskable RL environment ensure that $v_\pi(s) > v_{\pi'}(s)$ for every policy π' that does not reach a goal state with probability 1 (this follows from Theorems 1, 2, and 3). Finally, as policy π is consistent with the hierarchies induced by `hrl`, `pop`, and `seq`, then the hierarchically optimal policies for each of those approaches must reach a goal state from s with probability 1. □

Theorem 5 (Dominance among optimal policies)**.** *Let* E, $\mathcal{M}$, *and* G *be defined as in Theorem 4. Let* π^*_h, π^*_p, *and* π^*_s *be the hierarchically optimal policy for the MDP corresponding to* G *when using algorithms* `hrl`, `pop`, *and* `seq`, *respectively. If there exists a high-level sequential plan* Π *that solves the planning task* $\langle \mathcal{D}, L(s), G\rangle$ *from the initial low-level state* s, *then,*

$$\pi^* \geq \pi^*_h \geq \pi^*_p \geq \pi^*_s$$

where π^* *represents the optimal policy for the MDP corresponding to* G *and* $\pi_1 \geq \pi_2$ *iff* $v_{\pi_1}(s) \geq v_{\pi_2}(s)$.

Proof sketch. Hierarchies constrain the space of feasible policies. As such, the quality of the best hierarchically optimal policy can only decrease as more policies are pruned. Given any high-level plan Π, we know that all the policies consistent with `seq` are consistent with `pop` and all the policies consistent with `pop` are consistent with `hrl`. Therefore, $\pi^*_h \geq \pi^*_p \geq \pi^*_s$. Finally, as `hrl` imposes some constraints over the set of feasible policies, then $\pi^* \geq \pi^*_h$. □

Finally, it is worth discussing (at least informally) the following properties. First, `ql` does converge to globally optimal policies regardless of the quality of the symbolic model (since it does not use it). For the same reasons, `ql` is expected to learn slower than our hierarchical methods. Second, hierarchically optimal policies for $\texttt{seq}_\texttt{m}$ and $\texttt{pop}_\texttt{m}$ also reach goal states with probability 1 under the same conditions detailed in Theorem 4. However, this is the case only if monitoring is exclusively used for moving from a longer to a shorter high-level plan. Third, there is no dominance on the quality of hierarchically optimal policies when monitoring

is included. In fact, the hierarchically optimal policies for $\texttt{seq}_\texttt{m}$ or $\texttt{pop}_\texttt{m}$ might be better than π_h^* under certain conditions. However, they might also be worse than π_s^*. Lastly, note that the gap between the different policies in Theorem 5 partially depends on the quality of the symbolic model.

6 Empirical Evaluation

We evaluated our approach by considering two low-level environments and respective high-level models. To best evaluate the effectiveness at leveraging previous experience, we also defined a sequence of 4 tasks for each environment, ordered roughly by level of complexity. For each tested algorithm, the evaluation proceeded as follows. Each task was trained on for a fixed number of training steps. Whenever a goal state was reached, the task was restarted. If a task ran for more than $1,000$ steps without reaching the goal, it was also restarted. When the limit of total training steps for a task was reached, the meta-controller policy was reset and training began on the next task. When using `seq`, `pop`, $\texttt{seq}_\texttt{m}$, $\texttt{pop}_\texttt{m}$, or `hrl`, the trained policies for the options were transferred between tasks. In all cases, option policies and meta-controllers were trained by q-learning. To actually evaluate the quality of the learned policies, we paused the training every $10,000$ training steps and ran a number of independent trials using the policy as learned at that point.

Benchmark Environments

The first test environment is the OFFICEWORLD running example. Each training episode was initialized with a random initial state, and the evaluation trials were done from 10 different predefined initial states. To account for different outcomes when tie-breaking, each such trial was run 10 times.

Our second environment is the Minecraft-inspired grid-world described by Andreas, Klein, and Levine (2017). 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 *"make a pickaxe,"* which requires getting wood and iron and taking them to various locations, or *"get a gem,"* which requires first making a pickaxe and then going to the location with the gem. We ran experiments using random initial states for training and evaluating on 5 predefined initial states. Each experiment was run 5 times.

Results and Discussion

To adequately display how our approach is capable of converging quickly to high-quality solutions, Figure 3 displays a comparison between our main approaches—`seq` and `pop`—and the two basic baseline algorithms in both benchmark domains. Each graph displays the reward obtained after training with the labeled algorithm for the specified number of steps.

The experimental results show that—once the option policies are sufficiently well trained—our approach can significantly outperform `ql` and `hrl` when the number of training steps is limited. For instance, in the last task of the OFFICEWORLD, `pop` needed only $70,000$ training steps to converge to a policy that resulted in traces that were typically 10 steps away from optimal. In contrast, `hrl` needed at least $1,800,000$ steps before finding a policy of comparable quality and did not appear to converge to stability in less than the $5,000,000$ training steps we allowed. That said, `hrl` did reach policies that resulted in slightly better solutions—only 5 steps worse than optimal. Q-learning converged to optimality after $3,850,000$ training steps.

The tasks in the MINECRAFTWORLD domain are significantly harder than those of the OFFICEWORLD domain and serve as a stress test for our approaches. In particular, the planning model is not strictly consistent with the low-level environment so the tasks exhibit a variety of pitfalls that make accidentally undoing previous work very easy. For example, if the agent is carrying a piece of wood and walks through a cell marked as a tool bench, it will automatically convert the wood into a wooden plank, even if it actually needs the wood for some other reason. Despite this, our results show that `seq` leads to reasonable results after very little training. In contrast, `pop` does not perform any better than `hrl`. In Figure 4, we show what happens when we address the unexpected outcomes with execution monitoring. In the OFFICEWORLD, the policies obtained by $\texttt{seq}_\texttt{m}$ seem to result in slightly more stable performance. For the MINECRAFTWORLD, $\texttt{pop}_\texttt{m}$ significantly outperforms `pop`, even if it still does not converge to high-quality policies.

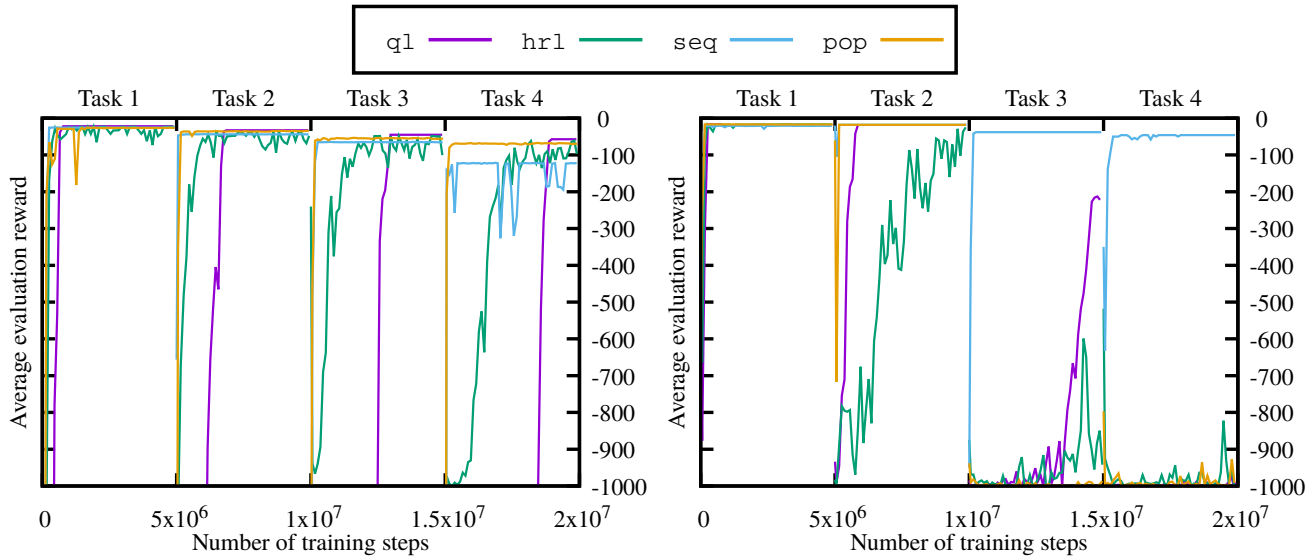
7 Related Work

In this section, we discuss a variety of work that touches upon aspects that are related to taskable RL, or to the use of symbolic planning to enhance RL systems.

Symbolic Planning and Reinforcement Learning

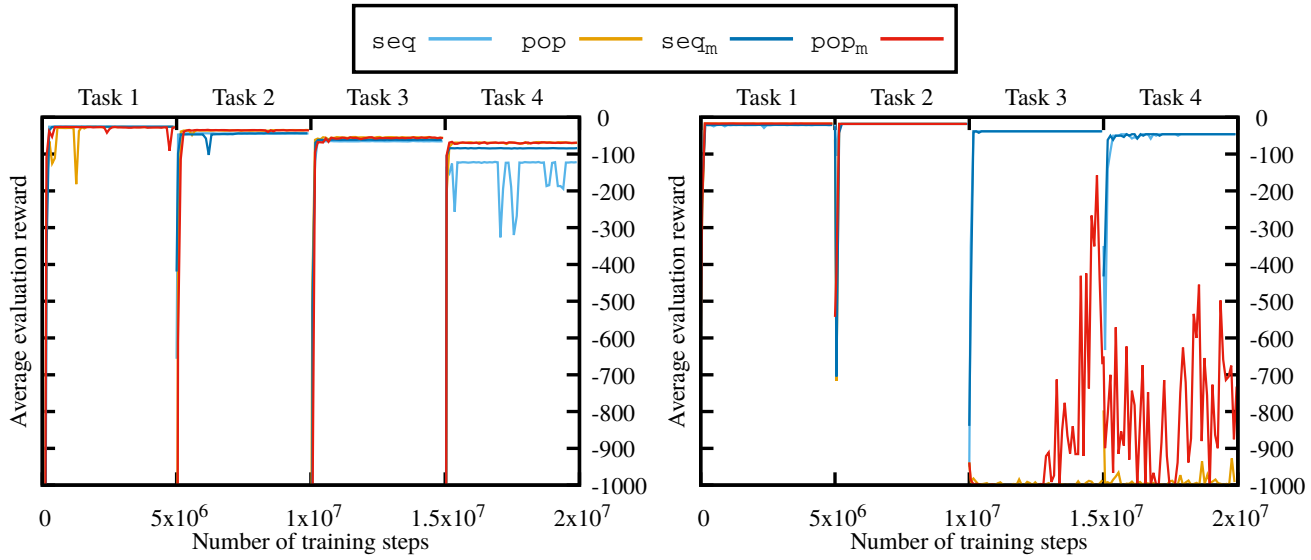
The idea of using a symbolic planning model to define tasks, hierarchically decompose them, and provide high-level guidance was first introduced by Ryan (2002). Our work further advances this same direction by relaxing some key assumptions and by developing further theoretical foundations for the use of symbolic models in RL. In particular, Ryan's work uses a high-level planner to produce *universal* and *complete* plans. That is, the plans produced are policies that tell exactly what must be done for every possible high-level state. In addition, the high-level actions are assumed to be *teleo-operators* (Nilsson 1994): they are defined in terms of preconditions and postconditions, but also require that their preconditions hold throughout their whole execution. Finally, the overall reward signal was restricted to be 1 when a goal state is reached and 0 anywhere else.

Other work explored the use of a symbolic planner coupled into an RL agent (Grounds and Kudenko 2008). 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 (Grześ and Kudenko 2008). Both approaches rely on modifying the reward signal in order to make progress towards solving the task. In contrast,



(a) OFFICEWORLD (b) MINECRAFTWORLD

Figure 3: Experimental performance obtained in two separate environments. We report the mean reward obtained by two baseline algorithms, q-learning (`ql`) and standard HRL (`hrl`), by our basic approach in which a sequential plan is executed directly (`seq`), and by our main approach in which HRL is restricted to execute a partial-order plan (`pop`).



(a) OFFICEWORLD (b) MINECRAFTWORLD

Figure 4: Comparison of different variants of our approach. In addition to `seq` and `pop`, we display the results obtained when regressing sequential and partial-order plans into structured policies ($\texttt{seq}_\texttt{m}$ and $\texttt{pop}_\texttt{m}$, respectively).

our approach relies on exploiting temporal abstractions from Hierarchical RL—which allows for transferring previously learned policies to solve new tasks faster. In fact, we performed additional experiments where we included a reward shaping approach to enhance `ql` and `hrl`. However, as the results show, this had limited impact in the performance over our benchmark domains (see Appendix).

Recently, Lyu et al. (2019)—building on the work done by Yang et al. (2018)—proposed an agent that uses hierarchical RL to integrate symbolic planning and reinforcement learning. Their problem setup considerably departs from taskable RL, though. In their case, the high-level model is given as prior knowledge of the environment without any particular goal condition. As such, their main contribution relies on

how to generate meaningful goals for the planner in order to learn a policy that optimizes the unknown reward function—which is not an issue that arises in taskable RL.

There has also been work that has focused on learning explicit state-transition systems that represent high-level models (Zhang et al. 2018; Nasiriany et al. 2019; Eysenbach, Salakhutdinov, and Levine 2019). Our work considers *implicit* state-transition systems described as classical planning domains. This allows us to consider highly combinatorial problems that correspond to state-transition systems that are far too large to be represented explicitly.

Taskable RL vs Multi-Task RL vs Multi-Goal RL

In Multi-Task RL, the goal is to create RL agents that get better at finding policies for novel tasks over time. Formally, the agent must learn strategies to maximize expected future rewards over a probability distribution of MDPs. On every episode, a completely new MDP is sampled and the agent is expected to do well on it (Brunskill and Li 2013). While related, the focus of taskable RL is different. It considers an environment with fixed dynamics but changing goal conditions—which are easy to specify by the user and are given to the agent as key information for solving the task.

In Multi-Goal RL, the objective is to learn one policy that can achieve different goals (e.g., Kaelbling (1993), Schaul et al. (2015), and Andrychowicz et al. (2017)). A goal $g \in G$ is defined by a reward function r_g, a function $f_g : S \to \{0, 1\}$ that identifies when the goal is achieved, and a set of features ϕ_g to describe the goal. Then, the idea is to learn a policy that can achieve any goal $g \in G$ from any state $s \in S$. While Multi-Goal RL takes a state-centric approach to define a fixed set of possible goal conditions G, taskable RL takes a property-centric approach—where a set of high-level properties of the states are composed by the user to define novel tasks for the agent on-demand.

Instructions and Advice in RL

Finally, our work exploits ideas from the learning from instructions (and advice) literature in RL. The main observation is that RL agents can benefit from having a formal description of the task to be accomplished. These descriptions can take the form of a policy sketch (Andreas, Klein, and Levine 2017), a Linear Temporal Logic (LTL) formula (Toro Icarte et al. 2018b), or an automaton (Toro Icarte et al. 2018c; Toro Icarte et al. 2019), among others. Given such description, the sample efficiency of an RL agent can be improved by task decomposition (Andreas, Klein, and Levine 2017), reward shaping (Camacho et al. 2019), or guiding the exploration policy (Toro Icarte et al. 2018a).

8 Conclusions and Future Work

To conclude, we believe that the automatic generation of goal-relevant instructions is one of the key aspects that will enable RL systems to be both taskable and scalable. The combination of symbolic action models with model-free RL allows for solving problems that require both intricate control and long-term combinatorial planning. Taskable RL represents a valuable formalism for describing problems of this kind, and planning has shown to be a useful technique to aid in improving sample efficiency by enabling structured methods of exploration and transfer learning.

Appendix: Reward Shaping

The idea behind reward shaping is that artificially modifying the reward signal of an MDP can improve sample efficiency by providing better feedback to an RL agent. Potential-based reward shaping methods (Ng, Harada, and Russell 1999) guarantee that the optimal policy for the modified MDP is also optimal for the original MDP by requiring the shaped reward to be of the form $r'(s, a, s') = r(s, a, s') + F(s, s')$, where $F(s, s') = \gamma\Phi(s') - \Phi(s)$ for some *potential function* $\Phi\colon S \to \mathbb{R}$. Grześ and Kudenko (2008) proposed a plan-based potential function that serves to guide an RL agent towards following a given high-level sequential plan Π. Given the sequence of high-level states $[\mathcal{S}_0, \mathcal{S}_1, \cdots, \mathcal{S}_n]$ that results by following the execution of Π, define $\Phi(s) = i$ for every $s \in S$ such that $L(s) = \mathcal{S}_i$. That is, low-level states increase in potential when their corresponding high-level counterparts are closer to the goal according to Π. In addition, states whose high-level representation does not appear in the plan are assigned the potential of the last state visited that did correspond to the plan.

In Figure 5, we show the results of using this reward shaping approach over `ql` and `hrl` (labeled `sql` and `shrl`, respectively) in the MINECRAFTWORLD. We omit displaying the results for the OFFICEWORLD, since the approach had no discernible impact in the performance over this domain.

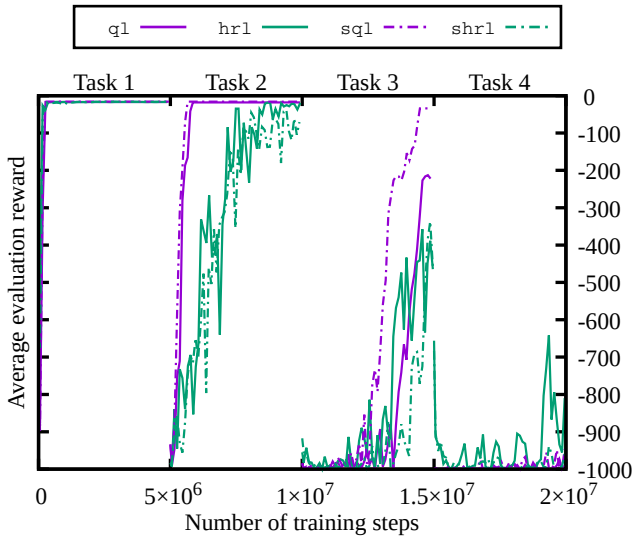


Figure 5: Comparison of the basic benchmarks applied directly (`ql` and `hrl`) and over the plan-based shaped MDPs (`sql` and `shrl`) in the MINECRAFTWORLD.

References

Akkaya, I.; Andrychowicz, M.; Chociej, M.; Litwin, M.; McGrew, B.; Petron, A.; Paino, A.; Plappert, M.; Powell, G.; Ribas, R.; et al.

2019. Solving Rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*.

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

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; and Zaremba, W. 2017. Hindsight experience replay. In *NIPS*, 5048–5058.

Andrychowicz, M.; Baker, B.; Chociej, M.; Jozefowicz, R.; McGrew, B.; Pachocki, J.; Petron, A.; Plappert, M.; Powell, G.; Ray, A.; et al. 2018. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*.

Bertsekas, D. P., and Tsitsiklis, J. N. 1991. An analysis of stochastic shortest path problems. *Mathematics of Operations Research* 16(3):580–595.

Brunskill, E., and Li, L. 2013. Sample complexity of multi-task reinforcement learning. In *UAI*, 122–131.

Camacho, A.; Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2019. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *IJCAI*, 6065–6073.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Eaton, J., and Zadeh, L. 1962. Optimal pursuit strategies in discrete-state probabilistic systems. *Journal of Basic Engineering* 84(1):23–29.

Eysenbach, B.; Salakhutdinov, R. R.; and Levine, S. 2019. Search on the replay buffer: Bridging planning and reinforcement learning. In *NeurIPS*, 15246–15257.

Falco, P.; Attawia, A.; Saveriano, M.; and Lee, D. 2018. On policy learning robust to irreversible events: An application to robotic in-hand manipulation. *IEEE Robotics and Automation Letters* 3(3):1482–1489.

Fritz, C., and McIlraith, S. A. 2007. Monitoring plan optimality during execution. In *ICAPS*, 144–151.

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

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

Kaelbling, L. P. 1993. Learning to achieve goals. In *IJCAI*, 1094–1099.

Konidaris, G., and Barto, A. G. 2007. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, 895–900.

Kumar, V.; Gupta, A.; Todorov, E.; and Levine, S. 2016. Learning dexterous manipulation policies from experience and imitation. *arXiv preprint arXiv:1611.05095*.

Kumar, V.; Todorov, E.; and Levine, S. 2016. Optimal control with learned local models: Application to dexterous manipulation. In *ICRA*, 378–383.

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

Muise, C. J.; McIlraith, S. A.; and Beck, J. C. 2011. Monitoring the execution of partial-order plans via regression. In *IJCAI*, 1975–1982.

Nasiriany, S.; Pong, V.; Lin, S.; and Levine, S. 2019. Planning with goal-conditioned policies. In *NeurIPS*, 14843–14854.

Ng, A. Y.; Harada, D.; and Russell, S. J. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 278–287.

Nilsson, N. J. 1994. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research* 1:139–158.

Ryan, M. R. K. 2002. Using abstract models of behaviours to automatically generate reinforcement learning hierarchies. In *ICML*, 522–529.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal value function approximators. In *ICML*, 1312–1320.

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

Taylor, M. E., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10(Jul):1633–1685.

Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2018a. Advice-based exploration in model-based reinforcement learning. In *Canadian Conf. on Artificial Intelligence* , 72–83.

Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2018b. Teaching multiple tasks to an RL agent using LTL. In *AAMAS*, 452–461.

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

Toro Icarte, R.; Waldie, E.; Klassen, T.; Valenzano, R.; Castro, M.; and McIlraith, S. 2019. Learning reward machines for partially observable reinforcement learning. In *NeurIPS*, 15523–15534.

Van Hoof, H.; Hermans, T.; Neumann, G.; and Peters, J. 2015. Learning robot in-hand manipulation with tactile features. In *Humanoids*, 121–127.

Watkins, C. J. C. H., and Dayan, P. 1992. Q-learning. *Machine Learning* 8(3-4):279–292.

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

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