Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

Verifiable and Compositional Reinforcement Learning Systems

Cyrus Neary[1], **Christos Verginis**[2], **Murat Cubuktepe**[3], **Ufuk Topcu**[1,3]

[1] Oden Institute for Computational Engineering and Sciences, The University of Texas at Austin, USA
[2] Division of Signals and Systems, Department of Electrical Engineering, Uppsala University, Sweden
[3] Department of Aerospace Engineering and Engineering Mechanics, The University of Texas at Austin, USA
{cneary, mcubuktepe, utopcu}@utexas.edu, christos.verginis@angstrom.uu.se

Abstract

We propose a framework for verifiable and compositional reinforcement learning (RL) in which a collection of RL subsystems, each of which learns to accomplish a separate subtask, are composed to achieve an overall task. The framework consists of a *high-level* model, represented as a parametric Markov decision process (pMDP) which is used to plan and to analyze compositions of subsystems, and of the collection of *low-level* subsystems themselves. By defining interfaces between the subsystems, the framework enables automatic decompositions of task specifications, *e.g., reach a target set of states with a probability of at least 0.95*, into individual subtask specifications, *i.e. achieve the subsystem's exit conditions with at least some minimum probability, given that its entry conditions are met*. This in turn allows for the independent training and testing of the subsystems; if they each learn a policy satisfying the appropriate subtask specification, then their composition is guaranteed to satisfy the overall task specification. Conversely, if the subtask specifications cannot all be satisfied by the learned policies, we present a method, formulated as the problem of finding an optimal set of parameters in the pMDP, to automatically update the subtask specifications to account for the observed shortcomings. The result is an iterative procedure for defining subtask specifications, and for training the subsystems to meet them. As an additional benefit, this procedure allows for particularly challenging or important components of an overall task to be identified automatically, and focused on, during training. Experimental results demonstrate the presented framework's novel capabilities in both discrete and continuous RL settings. A collection of RL subsystems are trained, using proximal policy optimization algorithms, to navigate different portions of a labyrinth environment. A cross-labyrinth task specification is then decomposed into subtask specifications. Challenging portions of the labyrinth are automatically avoided if their corresponding subsystems cannot learn satisfactory policies within allowed training budgets. Unnecessary subsystems are not trained at all. The result is a compositional RL system that efficiently learns to satisfy task specifications.

Introduction

Reinforcement learning (RL) algorithms offer tremendous capabilities in systems that work with unknown environments. However, there remain significant barriers to their deployment in safety-critical engineering applications. Autonomous vehicles, manufacturing robotics, and power systems management are examples of complex application domains that require strict adherence of the system's behavior to stakeholder requirements. However, the verification of RL systems is difficult. This is particularly true of monolithic end-to-end RL approaches; many model-free RL algorithms, for instance, only output the learned policy and its estimated value function, rendering them opaque for verification purposes. The difficulty of verification is compounded in engineering application domains, which often require large observation and action spaces, and complicated reward functions.

How do we build complex engineering systems we can trust? Engineering design principles have long prescribed system modularity as a means to reduce the complexity of individual subsystems (Haberfellner et al. 2019; Nuseibeh and Easterbrook 2000). By creating well-defined interfaces between subsystems, system-level requirements may be decomposed into component-level ones. Conversely, each component may be developed and tested independently, and the satisfaction of component-level requirements may then be used to place assurances on the behavior of the system as a whole. Building RL systems that incorporate such engineering practices and guarantees is a crucial step toward their widespread deployment.

Toward this end, we develop a framework for verifiable and compositional reinforcement learning. The framework comprises two levels of abstraction. The *high level* is used to plan *meta-policies* and to verify their adherence to task specifications, *e.g., reach a particular goal state with a probability of at least 0.9*. Meta-policies dictate sequences of *subsystems* to execute, each of which is designed to accomplish a specific *subtask*, *i.e. achieve a particular exit condition, given the subsystem is executed from one of its entry conditions*. We assume a collection of *partially instantiated* subsystems to be given a priori; their entry and exit conditions are known, but the policies they implement are not. These entry and exit conditions might be defined by pre-existing engineering capabilities, explicitly by a task designer, or by entities within the environment. At the *low level* of the framework, each subsystem employs RL algorithms to learn policies accomplishing its subtask. Figure 1 illustrates the major components of the proposed framework.

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

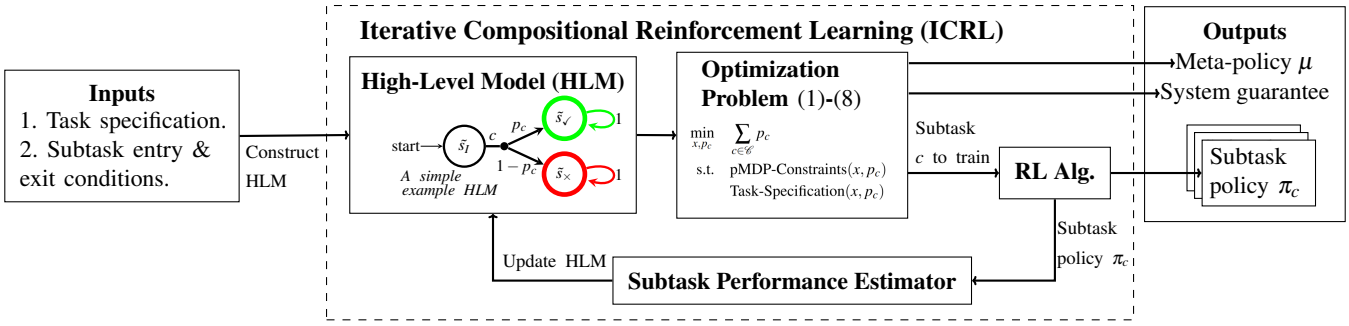


Figure 1: An illustration of the proposed framework. The task specification and the subtask entry and exit conditions are used to build the *high-level model* (HLM) of the compositional RL system. We use the HLM to formulate an optimization problem whose outputs yield a *meta-policy*, the probability of overall task success, and separate specifications for each subtask. The subtask specifications are used to select the next subsystem to train using the RL algorithm of choice. Estimates of the resulting subsystem policies are then used to update the HLM. This iterative process repeats until either the composite system satisfies the task specification, or a user-defined training budget has been exhausted.

We model the high level of the framework using a parametric Markov decision process (pMDP) (Cubuktepe et al. 2018; Junges et al. 2019). Each action in the pMDP represents an individual RL subsystem, and the parametric transition probabilities in the pMDP thus represent the likelihoods of outcomes that could occur when the subsystem is executed. Using sampling-based estimates of subsystem policies, we assign values to the model parameters and use existing MDP techniques for the planning and verification of meta-policies (Puterman 2014; Baier and Katoen 2008). Beyond this capability, the framework offers the following novel features.

1. Automatic decomposition of task specifications. We formulate, as the problem of finding an optimal set of parameters in the pMDP, a method to automatically decompose the task specification into subtask specifications, allowing for independent learning and verification of the subsystems.

2. Learning to satisfy subtask specifications. Any RL method can be used to learn the subsystem policies, so long as the learned policies satisfy the relevant subtask specification. We present a subsystem reward function definition, in terms of the exit conditions of the subsystem, that motivates the learning of policies satisfying the subtask specification. Furthermore, these subtask specifications provide an *interface* between the subsystems, allowing for the analysis of their compositions. In particular, we guarantee that if each of the learned subsystem policies satisfies its subtask specifications, a composition of them exists satisfying the specifications on the overall task.

3. Iterative specification refinement. However, if some of the subtask specifications cannot be satisfied by the corresponding learned policies, sampling-based estimates of their behavior are used to update the high-level model. We present a method to use this information to refine the subtask specifications, in order to better reflect what might realistically be achieved by the subsystems. This automatic refinement naturally leads to a compositional RL algorithm that iteratively computes subtask specifications, and then trains the corresponding subsystems to achieve them.

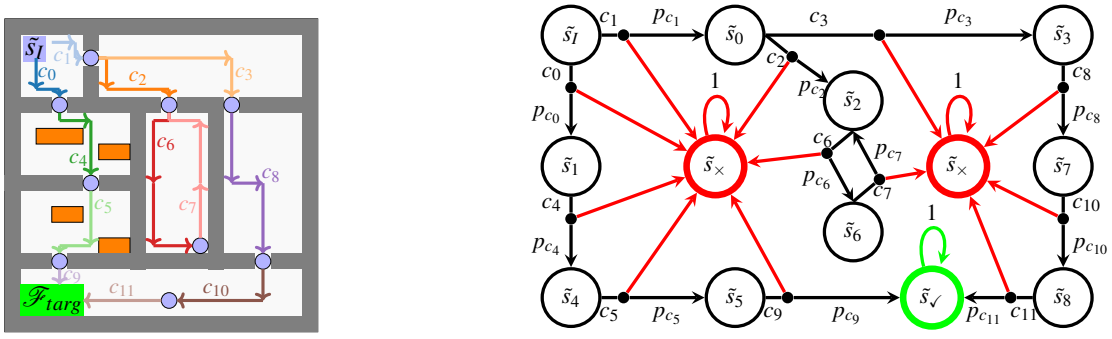
4. System modularity: prediction and verification in task transfer. By providing an interface between the subtasks, the presented framework allows for previously learned subtask policies to be re-used as components of new high-level models, designed to solve different tasks. Furthermore, the subtask specifications themselves may be re-used to perform verification within these new models, without the need for further training.

Experimental results exemplify these novel capabilities in both discrete and continuous versions of a labyrinth navigation task. We use proximal policy optimization algorithms (Schulman et al. 2017) to train individual subsystems to navigate portions of the environment, which are then composed to complete a cross-labyrinth navigation task. Through the aforementioned compositional RL algorithm, the task specification is decomposed and challenging portions of the labyrinth are avoided if their corresponding subsystems cannot satisfy their specifications.

The Compositional RL Framework

To provide intuitive examples of the notions of tasks, subtasks, systems, and subsystems, we consider the example labyrinth environment shown in Figure 2a. The *system* executes its constituent *subsystems* in this environment to complete an overall task. The *task* is to safely navigate from the labyrinth's initial state in the top left corner to the goal state in the bottom left corner. Satisfaction of the *task specification* requires that the system successfully completes the task with a probability of at least 0.95. As an added difficulty, lava exists within some of the rooms, represented in the figure by the orange rectangles. If the lava is touched, the task is automatically failed. This task is naturally decomposed into separate *subtasks*, each of which navigates an individual room, and is executed by a separate subsystem.

Preliminaries. We model the task environment as a Markov decision process (MDP), which is defined by a tuple $M = (S, A, P)$. Here, S is a set of states, A is a set of actions, and $P : S \times A \times S \to [0,1]$ is a transition probability



(a) The labyrinth task environment.

(b) The HLM corresponding to the labyrinth example.

Figure 2: An example labyrinth navigation task. Figure (a) illustrates the environment, as well as an example collection of subsystems, represented by the colored paths. Entry and exit conditions for the various subsystems are shown as blue circles. Figure (b) illustrates the corresponding HLM. Each subsystem c causes a transition to its successor state with probability p_c. Otherwise, the HLM transitions to the failure state $\tilde{s}_\times$ with probability $1-p_c$, visualized by the red transitions.

function. A stationary policy π within the MDP is a function $\pi : S \times A \to [0,1]$ such that $\sum_{a \in A} \pi(s,a) = 1$ for every $s \in S$. Intuitively, $\pi(s,a)$ assigns the probability of taking action a from state s under policy π. Given an MDP M, a policy π, and a target set of states $S_{targ} \subseteq S$, we define $\mathbb{P}^s_{M,\pi}(\Diamond S_{targ})$ to be the probability of eventually reaching some state $s' \in S_{targ}$, beginning from the initial state s, under policy π. Similarly, $\mathbb{P}^s_{M,\pi}(\Diamond_{\leq T} S_{targ})$ denotes the probability of reaching the target set from state s within some finite time horizon T.

The framework we present is agnostic to the implementation details of the RL algorithms that interact with the low-level environment. As such, S and A can either be uncountably infinite subsets of Euclidean space, or they can be countable sets indexing the states and actions. Our experiments examine both cases. For notational simplicity, we present the framework for countable sets S and A.

RL Subsystems and Subtasks. We define each RL subsystem c acting within the environment by the tuple $c = (\mathscr{I}_c, \mathscr{F}_c, T_c, \pi_c)$. Here, $\mathscr{I}_c \subseteq S$ is a set defining the subsystem's *entry conditions*, $\mathscr{F}_c \subseteq S$ is a set representing the subsystem's *exit conditions*, and $T_c \in \mathbb{N}$ is the subsystem's allowed *time horizon*. The *subtask* associated with each subsystem, is to navigate from any entry condition $s \in \mathscr{I}_c$ to any exit condition $s' \in \mathscr{F}_c$ within the subsystem's time horizon T_c. The time horizon is included to ensure that the compositional system will complete its task in finite time. We assume that each subsystem may only be *executed*, or begun, from an entry condition $s \in \mathscr{I}_c$ and that its execution ends either when it achieves an exit condition $s \in \mathscr{F}_c$, or when it runs out of time. Finally, $\pi_c : S \times A \to [0,1]$ is the policy that the component implements to complete this objective.

For notational convenience, we define $\sigma^c_{\pi_c}(s) := \mathbb{P}^s_{M,\pi_c}(\Diamond_{\leq T_c} \mathscr{F}_c)$. A *subtask specification*, is then defined as the requirement that $\sigma^c_{\pi_c}(s) \geq p_c$ for every entry condition $s \in \mathscr{I}_c$ of the subsystem. Here, $p_c \in [0,1]$ is a value representing the minimum allowable probability of the subtask success. We note that such reachability-based task specifications are very expressive. Temporal logic specifications can be expressed as reachability specifications in a so-called product MDP (Baier and Katoen 2008; Hahn et al. 2019).

We say a subsystem c is *partially instantiated* when $\mathscr{I}_c$, $\mathscr{F}_c$, and T_c are defined, but its policy π_c is not. We define a collection $\mathscr{C} = \{c_1, c_2, ..., c_k\}$ of subsystems to be *composable*, if and only if for every $i, j \in \{1, 2, \ldots, k\}$, either $\mathscr{F}_{c_i} \subseteq \mathscr{I}_{c_j}$ or $\mathscr{F}_{c_i} \cap \mathscr{I}_{c_j} = \emptyset$. In words, subsystems are composable when the set of exit conditions of each subsystem is a subset of all the sets of entry conditions that it intersects. This ensures that regardless of the specific exit condition $s \in \mathscr{F}_c$ in which subsystem c terminates, s will be a valid entry condition for the *same* collection of other subsystems.

Compositions of RL Subsystems. Compositions of subsystems are specified by *meta-policies* $\mu : S \times \mathscr{C} \to [0,1]$, which assign probability values to the execution of different subsystems, given the current environment state $s \in S$. So, execution of the composite system occurs as follows. From a given state s, the meta-policy is used to select a subsystem c to execute. The subsystem's policy π_c is then followed until it either reaches an exit condition $s' \in \mathscr{F}_c$, or it reaches the end of its time horizon T_c. If the former is true, the meta-policy selects the next subsystem to execute from s', and the process repeats. Conversely, if the latter is true, the subsystem has failed to complete its subtask in time, and the execution of the meta-policy stops. In the labyrinth example, the meta-policy selects which rooms to pass through, while the subsystems policies navigate the individual rooms.

The *task* of the composite system is, beginning from an initial state s_I, to eventually reach a particular target exit condition $\mathscr{F}_{targ} \subseteq S$. We assume that $\mathscr{F}_{targ}$ is equivalent to $\mathscr{F}_c$ for at least one of the subsystems. That is, there is some subsystem $c \in \mathscr{C}$ such that $\mathscr{F}_{targ} = \mathscr{F}_c$. Furthermore, to simplify theoretical analysis, we assume that for every $c \in \mathscr{C}$, either $\mathscr{F}_c = \mathscr{F}_{targ}$ or $\mathscr{F}_c \cap \mathscr{F}_{targ} = \emptyset$. This assumption removes ambiguity as to whether or not completion of a given subtask results in the immediate completion of the system's

task. Finally, we assume that at least one subsystem c can be executed from the initial state s_I, i.e. there exists a subsystem $c \in \mathscr{C}$ such that $s_I \in \mathscr{I}_c$. We say that the execution of a meta-policy reaches the target set $\mathscr{F}_{targ}$, when one of the subsystems c with $\mathscr{F}_c = \mathscr{F}_{targ}$ is executed, and successfully completes its subtask. With a slight abuse of notation, we denote the probability of eventually reaching the target set under meta-policy μ by $\mathbb{P}^{s_I}_{M,\mu}(\Diamond \mathscr{F}_{targ})$.

A *task specification* places a requirement on the probability of the compositional RL system reaching $\mathscr{F}_{targ}$. That is, for some allowable failure probability $\delta \in [0,1]$, the task specification is satisfied if $\mathbb{P}^{s_I}_{M,\mu}(\Diamond \mathscr{F}_{targ}) \geq 1-\delta$. With these definitions in place, we now deliver our problem statement.

Problem Statement. *Given an allowable failure probability $\delta \in [0,1]$, an initial state s_I, a target set $\mathscr{F}_{targ}$, and a partially instantiated collection $\mathscr{C}$ of composable subsystems, learn policies π_c for each subsystem $c \in \mathscr{C}$ and compute a meta-policy μ such that $\mathbb{P}^{s_I}_{M,\mu}(\Diamond \mathscr{F}_{targ}) \geq 1-\delta$.*

The High-Level Decision-Making Model

We now introduce the high-level model (HLM) of the compositional RL framework, which is used to compute meta-policies, and to decompose task specifications into subtask specifications to be satisfied by the individual subsystems.

Defining the High-Level Model (HLM). To construct the HLM, we use a given collection $\mathscr{C} = \{c_1, c_2, \ldots, c_k\}$ of partially instantiated subsystems, an initial state s_I, and a target set $\mathscr{F}_{targ}$. We begin by defining a state abstraction, which groups together environment states in order to define the state space of the HLM. To do so, we define the equivalence relation $R \subseteq S \times S$ such that $(s,s') \in R$ if and only if the following two conditions hold.

1. For every $c \in \mathscr{C}, s \in \mathscr{I}_c$ if and only if $s' \in \mathscr{I}_c$, and,
2. $s \in \mathscr{F}_{targ}$ if and only if $s' \in \mathscr{F}_{targ}$.

The equivalence class of any state $s \in S$ under equivalence relation R is given by $[s]_R = \{s' \in S | (s,s') \in R\}$. The quotient set of S by R is defined as the set of all equivalence classes $S/_R = \{[s]_R | s \in S\}$. Intuitively, this equivalence relation groups together all the states in the target set, and it also groups together states that are entry conditions to the same subset of subsystems.

We may now define the HLM corresponding to the collection $\mathscr{C}$ by the parametric MDP $\tilde{M} = (\tilde{S}, \tilde{s}_I, \tilde{s}_\checkmark, \tilde{s}_\times, \mathscr{C}, \tilde{P})$. Here, the high-level states $\tilde{S}$ are defined to be $S/_R$; states in the HLM correspond to equivalence classes of environment states. The initial state $\tilde{s}_I$ of the HLM is defined as $\tilde{s}_I = [s_I]_R$, the equivalence class of the environment's initial state. The *goal state* $\tilde{s}_\checkmark \in \tilde{S}$ is similarly defined as $[s]_R$ such that $s \in \mathscr{F}_{targ}$. Recall that $\mathscr{F}_{targ} = \mathscr{F}_c$ for at least one of the subsystems $c \in \mathscr{C}$. Finally, the *failure state* $\tilde{s}_\times \in \tilde{S}$ is defined as $[s]_R$ such that $s \in S \setminus [\bigcup_{c \in \mathscr{C}} \mathscr{I}_c] \cup \mathscr{F}_{targ}$, i.e., the equivalence class of states *not* belonging to the initial states of any component, or to the target set.

As an example, Figure 2b illustrates the HLM corresponding to the collection of subsystems from Figure 2a. The overlapping entry and exit conditions, represented by the blue circles in Figure 2a, define the states of the HLM. The target set $\mathscr{F}_{targ}$ defines the HLM's goal state $\tilde{s}_\checkmark$, and all other environment states are absorbed into the failure state $\tilde{s}_\times$.

The collection of subsystems $\mathscr{C}$ defines the HLM's set of actions. By definition of the equivalence relation R, for every HLM state $\tilde{s} \in \tilde{S}$ there is a well-defined subset of the subsystems $\mathscr{C}(\tilde{s}) \subseteq \mathscr{C}$ that can be executed. That is, for every environment state $s \in \tilde{s}$, $s \in \mathscr{I}_c$ for all $c \in \mathscr{C}(\tilde{s})$. We define $\mathscr{C}(\tilde{s})$ as the set of *available subsystems* at high-level state $\tilde{s}$.

Furthermore, consider any subsystem $c \in \mathscr{C}(\tilde{s})$. As a direct result of the definition of equivalence relation R and of the subsystems in collection $\mathscr{C}$ being composable, every state s within set $\mathscr{F}_c$ belongs to the *same* equivalence class $[s]_R$. In other words, we may uniquely define the successor HLM state of any component $c \in \mathscr{C}$ as $succ(c) = [s]_R$ such that $s \in \mathscr{F}_c$. We then construct the HLM transition probability function in terms of parameters $p_c \in [0,1]$ as follows.

$$\tilde{P}(\tilde{s}, c, \tilde{s}') = \begin{cases} p_c, & if \;\; c \in \mathscr{C}(\tilde{s}), \;\; \tilde{s}' = succ(c) \\ 1 - p_c, & if \;\; c \in \mathscr{C}(\tilde{s}), \;\; \tilde{s}' = \tilde{s}_\times \\ 0, & \text{Otherwise} \end{cases}$$

The interpretation of this definition of $\tilde{P}$ is as follows. After selecting component $c \in \mathscr{C}(\tilde{s})$ from HLM state $\tilde{s}$, the component either succeeds in reaching an exit condition $s \in \mathscr{F}_c$ within its time horizon T_c with probability p_c, resulting in an HLM transition to $succ(c)$, or it fails to do so with probability $1 - p_c$, resulting in a transition to the HLM failure state $\tilde{s}_\times$.

The parameters p_c may thus be interpreted as estimates of the probabilities that the subsystems complete their subtasks, given they are executed from one of their entry conditions. Their values come either from empirical rollouts of learned subsystem policies π_c, or as the solution to the aforementioned automatic decomposition of the task specification, which is discussed further below.

Relating the HLM to Compositions of RL Subsystems. We note that while parameters p_c are meant to estimate the probabilities of successful subtask completion, they cannot capture these probabilities exactly. In reality, while parameter p_c is constant, it's possible for this probability to vary, given the entry condition $s \in \mathscr{I}_c$ from which the component is executed. However, the simplicity of the presented parametrization of $\tilde{P}$ enables tractable solutions to planning and verification problems in $\tilde{M}$. Furthermore, by establishing relationships between policies in $\tilde{M}$, and meta-policies composing RL subsystems, the HLM becomes practically useful in the analysis of composite RL systems.

Towards this idea, we note that any stationary policy $\tilde{\mu} : \tilde{S} \times \mathscr{C} \to [0,1]$ acting in HLM $\tilde{M}$ defines a unique compositional meta-policy $\mu : S \times \mathscr{C} \to [0,1]$ as follows: for any environment state s and component c, define $\mu(s,c) := \tilde{\mu}([s]_R, c)$. So, solutions to planning problems in $\tilde{M}$ can be used directly as meta-policies to specify compositions of the RL subsystems. Of particular interest, is the problem of computing an HLM policy $\tilde{\mu}$ that maximizes $\mathbb{P}^{\tilde{s}_I}_{\tilde{M},\tilde{\mu}}(\Diamond \tilde{s}_\checkmark)$, the probability of eventually reaching the goal state $\tilde{s}_\checkmark$ from the HLM's initial state $\tilde{s}_I$. Theorem 1 relates this probability to the corresponding meta-policy's probability of completing its task, $\mathbb{P}^{s_I}_{M,\mu}(\Diamond \mathscr{F}_{targ})$, in the environment.

Theorem 1. *Let $\mathscr{C} = \{c_1, c_2, ..., c_k\}$ be a collection of composable subsystems with respect to initial state s_I and target set $\mathscr{F}_{targ}$ within the environment MDP M. Define $\tilde{M}$ to be the corresponding HLM and let $\tilde{\mu}$ be a policy in $\tilde{M}$. If, for every subsystem $c \in \mathscr{C}$ and for every entry condition $s \in \mathscr{I}_c$, $\sigma^c_{\pi_c}(s) \geq p_c$, then $\mathbb{P}^{s_I}_{M,\mu}(\Diamond \mathscr{F}_{targ}) \geq \mathbb{P}^{\tilde{s}_I}_{\tilde{M},\tilde{\mu}}(\Diamond \tilde{s}_{\checkmark})$.*

For example, consider the labyrinth task from Figure 2a, and its corresponding HLM from Figure 2b. Suppose the HLM's parameters p_c are specified such that they lower bound the true probabilities of subtask success, i.e. the transition probabilities in Figure 2b lower bound the probabilities of the subsystems successfully navigating their respective rooms in Figure 2a. By planning a policy $\tilde{\mu}$ in the HLM that, for example, reaches $\tilde{s}_{\checkmark}$ with probability 0.95, we ensure that the corresponding composition of the subsystems will reach $\mathscr{F}_{targ}$ in the labyrinth with a probability of *at least* 0.95.

Automatic Decomposition of Task Specifications. Recall that our objective is not only to compute a meta-policy μ, but also to *learn* the subsystem policies $\pi_{c_1}, \pi_{c_2}, ..., \pi_{c_k}$ that this meta-policy will execute, such that the system's task specification $\mathbb{P}^{s_I}_{M,\mu}(\Diamond \mathscr{F}_{targ}) \geq 1-\delta$ is satisfied. Suppose that we choose a set of HLM parameters $\{p_{c_1}, p_{c_2}, ..., p_{c_{c_k}}\}$ such that a policy $\tilde{\mu}$ in the HLM exists with $\mathbb{P}^{s_I}_{M,\mu}(\Diamond S_{targ}) \geq 1-\delta$. Then, so long as each of the corresponding subsystems c are able to learn a policy π_c such that $\sigma^c_{\pi_c}(s) \geq p_c$ for every $s \in \mathscr{I}_c$, Theorem 1 tells us that the meta-policy defined by $\mu(s,c) := \tilde{\mu}([s]_R, c)$ will satisfy the task specification.

We may thus interpret the values of parameters p_c as *subtask specifications*. Each subsystem must achieve one of its exit conditions $s' \in \mathscr{F}_c$ within its allowed time horizon T_c with a probability of at least p_c, given its execution began from some entry condition $s \in \mathscr{I}_c$. With this interpretation in mind, we take the following approach to the decomposition of the task specification: find the smallest values of parameters $p_{c_1}, p_{c_2}, ..., p_{c_k}$ such that an HLM policy $\tilde{\mu}$ exists satisfying $\mathbb{P}^{\tilde{s}_I}_{\tilde{M},\tilde{\mu}}(\Diamond \tilde{s}_{\checkmark}) \geq 1-\delta$. We formulate this constrained parameter optimization problem as the bilinear program given in equations (1)-(5). In (2) and (5), we define $pred(\tilde{s}) := \{(\tilde{s}', c') | c' \in \mathscr{C}(\tilde{s}') \text{ and } \tilde{s} = succ(c')\}$.

$$\min_{x, p_c} \quad \sum_{c \in \mathscr{C}} p_c \tag{1}$$

$$\text{s.t.} \quad \sum_{c \in \mathscr{C}(\tilde{s})} x(\tilde{s}, c) = \delta_{\tilde{s}_I}(\tilde{s}) + \sum_{(\tilde{s}', c') \in pred(\tilde{s})} x(\tilde{s}', c') p_{c'}, \quad \forall \tilde{s} \in \tilde{S} \setminus \{\tilde{s}_\times, \tilde{s}_{\checkmark}\} \tag{2}$$

$$x(\tilde{s}, c) \geq 0, \ \forall \tilde{s} \in \tilde{S} \setminus \{\tilde{s}_\times, \tilde{s}_{\checkmark}\}, \ \forall c \in \mathscr{C}(\tilde{s}) \tag{3}$$

$$0 \leq p_c \leq 1, \ \forall c \in \mathscr{C} \tag{4}$$

$$\sum_{(\tilde{s}', c') \in pred(\tilde{s}_{\checkmark})} x(\tilde{s}', c') p_{c'} \geq 1 - \delta \tag{5}$$

The decision variables in (1)-(5) are the HLM parameters p_c for every $c \in \mathscr{C}$, and $x(\tilde{s}, c)$ for every $\tilde{s} \in \tilde{S} \setminus \{\tilde{s}_\times, \tilde{s}_{\checkmark}\}$. The value of $\delta_{\tilde{s}_I}(\tilde{s})$ is 1 if $\tilde{s} = \tilde{s}_I$ and 0 otherwise. The constraint (2) is the so-called Bellman-flow constraint; it ensures that the variable $x(\tilde{s}, c)$ defines the expected number of times subsystem c is executed in state $\tilde{s}$. The constraint (5) enforces the HLM policy $\tilde{\mu}$'s satisfaction of $\mathbb{P}^{\tilde{s}_I}_{\tilde{M},\tilde{\mu}}(\Diamond \tilde{s}_{\checkmark}) \geq 1-\delta$. We refer to Etessami et al. (2007) and Puterman (2014) for further details on these variables and the constraints.

Iterative Compositional Reinforcement Learning (ICRL)

In this section, we discuss how subsystem policies are learned to satisfy the subtask specifications discussed above, and we present how the bilinear program given in (1)-(5) is modified to refine the subtask specifications, after some training of the subsystems has been completed.

Learning and Verifying Subsystem Policies. Let $p_{c_1}, p_{c_2}, ..., p_{c_k}$ be the parameter values output as a solution to problem (1)-(5). We want each subsystem c to learn a policy π_c satisfying the subtask specification: $\sigma^c_{\pi_c}(s) \geq p_c$ for each entry condition $s \in \mathscr{I}_c$ of the subsystem. We note that any RL algorithm and reward function may be used, so long as the resulting learned policy can be verified to satisfy its subtask specification. A particularly simple candidate reward function R_c outputs 1 when an exit condition $s \in \mathscr{F}_c$ is first reached, and outputs 0 otherwise. Under this reward function, we have $\sigma^c_{\pi_c}(s) = \mathbb{E}[\sum_{t \in [T_c]} R_c(s_t) | \pi_c, s_0 = s]$. We can maximize the probability of reaching an exit condition by maximizing the expected undiscounted sum of rewards.

To verify that a learned subsystem policy π_c satisfies its subtask specification, we consider $\bar{\sigma}_c = \inf\{\sigma^c_{\pi_c}(s) | s \in \mathscr{I}_c\}$, the greatest lower bound of the policy's probability of subtask succcess, beginning from any of the subsystem's entry conditions. So long as $\bar{\sigma}_c \geq p_c$, the subtask specification is satisfied. In practice, the value of $\bar{\sigma}_c$ cannot be known exactly, but we may obtain an estimate $\hat{\sigma}_c$ of its value through empirical rollouts of π_c, beginning from the different entry conditions $s \in \mathscr{I}_c$. We note that one may additionally use Hoeffding's inequality to obtain a high-confidence range of values for $\bar{\sigma}_c$, given the number of rollouts used. We refer to $\hat{\sigma}_c$ as the *estimated performance value* of policy π_c.

Automatic Refinement of the Subtask Specifications. The estimated performance values $\hat{\sigma}_c$ are useful not only for the empirical verification of the learned policies, but also as additional information used periodically during training to refine the subtask specifications. To do so, we re-solve the optimization problem (1)-(5), with a modified objective (6), and additional constraints (7)-(8).

$$obj(\mathscr{L}) = \sum_{c \in \mathscr{C}} (p_c - \hat{\sigma}_c) \tag{6}$$

$$LBConst(\mathscr{L}) = \{p_c \geq \hat{\sigma}_c | \forall \hat{\sigma}_c \in \mathscr{L}\} \tag{7}$$

$$UBConst(\mathscr{U}) = \{p_c \leq \hat{\sigma}_c | \forall \hat{\sigma}_c \in \mathscr{U}\} \tag{8}$$

Here, we assume that the subsystems have learned policies $\pi_{c_1}, \pi_{c_2}, ..., \pi_{c_k}$. Let $\mathscr{L} = \{\hat{\sigma}_{c_1}, \hat{\sigma}_{c_2}, ..., \hat{\sigma}_{c_k}\}$ be the set of the corresponding estimated performance values. The objective function (6) minimizes the performance gap between the subtask specifications p_c and the current estimated performance values $\hat{\sigma}_c$. The rationale behind the additional constraints defined by $LBConst(\mathscr{L})$ is as follows: the subsystems have already learned policies achieving probabilities of subtask success greater than the estimated performance values $\hat{\sigma}_c$,

Algorithm 1: Iterative Compositional RL (ICRL)

Input: Partially instantiated subsystems $\mathscr{C} = \{c_1, c_2, ..., c_k\}$, δ, N_{train}, N_{max}.
Output: Subsystem policies $\{\pi_{c_1}, \pi_{c_2}, ..., \pi_{c_k}\}$, meta-policy μ, success probability $\hat{\sigma}_\mu$.

1 $\tilde{M} \leftarrow ConstructHLM(\mathscr{C})$
2 $\hat{\sigma}_{c_1}, \hat{\sigma}_{c_2}, ..., \hat{\sigma}_{c_k}, \hat{\sigma}_\mu \leftarrow 0$; $N_{c_1}, N_{c_2}, ..., N_{c_k} \leftarrow 0$
3 $\mathscr{L} \leftarrow \{\hat{\sigma}_{c_1}, \hat{\sigma}_{c_2}, ..., \hat{\sigma}_{c_k}\}$; $\mathscr{U} \leftarrow \{\}$
4 **while** $\hat{\sigma}_\mu \leq 1 - \delta$ **do**
5 **if** (1)-(8) infeasible **then**
6 **return** Problem is infeasible.
7 $\{p_{c_1}, \ldots, p_{c_k}\} \leftarrow$ Solve (1)-(8) using $(\tilde{M}, \mathscr{L}, \mathscr{U})$
8 $c_j \leftarrow selectSubSystem(p_{c_1}, \ldots, p_{c_k}, \hat{\sigma}_{c_1}, \ldots, \hat{\sigma}_{c_k})$
9 $\pi_{c_j} \leftarrow RLTrain(c_j, \pi_{c_j}, N_{train})$; $N_{c_j} \leftarrow N_{c_j} + N_{train}$
10 $\hat{\sigma}_{c_j} \leftarrow estimateSubTaskSuccessProb(c_j, \pi_{c_j})$
11 $\mathscr{L}.update(\hat{\sigma}_{c_j})$
12 **if** $N_{c_j} \geq N_{max}$ **then**
13 $\mathscr{U}.add(\hat{\sigma}_{c_j})$
14 $\mu \leftarrow solveOptimalHLMPolicy(\tilde{M}, \mathscr{L})$
15 $\hat{\sigma}_\mu \leftarrow predictTaskSuccessProbability(\tilde{M}, \mu, \mathscr{L})$
16 **return** $\{\pi_{c_1}, \pi_{c_2}, ..., \pi_{c_k}\}$, μ, $\hat{\sigma}_\mu$

and so there is no reason to consider subtask specifications p_c that are below these values.

Conversely, if the RL algorithm of a particular subsystem c has *converged* – i.e. the value of $\hat{\sigma}_c$ will no longer increase with additional training steps – we add the constraint $p_c \leq \hat{\sigma}_c$. This ensures that solutions to the optimization problem will *not* yield a subtask specification p_c that is larger than what the subsystem can realistically achieve. In practice, as a proxy to convergence, we allow each subsystem a maximum budget of N_{max} training steps. Once any subsystem c has exceeded this training budget, we append $\hat{\sigma}_c$ to the set $\mathscr{U}$, which is used to define $UBConst(\mathscr{U})$ in (8).

Iterative Compositional Reinforcement Learning (ICRL). By alternating between the training of the subsystems and the refinement of the subtask specifications, we obtain Algorithm 1. In lines $1-3$, the HLM is constructed from the collection of partially instantiated subsystems $\mathscr{C}$ and the subsystem policies are initialized. The while loop in lines $4-12$ is the main loop controlling the subtask specifications and training of the subsystems. In line 5, the bilinear program (1)-(8) is solved to update the values of p_c. These values are used, along with the estimated performance values, to select a subsystem to train. A simple selection scheme, is to choose the subsystem c_j maximizing the current performance gap between p_{c_j} and $\hat{\sigma}_{c_j}$. In line 7, the subsystem is trained for N_{train} steps using the RL algorithm of choice. The subsystem's initial state is sampled uniformly from its entry conditions during training. Finally, in line 12, the HLM $\tilde{M}$ and the current estimated performance values $\mathscr{L}$ are used to plan a meta-policy μ maximizing the probability $\hat{\sigma}_\mu$ of reaching the HLM goal state $\tilde{s}_\checkmark$. This step uses standard MDP algorithms (Puterman 2014).

We note that the conditions in lines 4 and 5 ensure that the algorithm only terminates once a meta-policy that satisfies the task specification exists, or the optimization problem (1)-(8) has become infeasible. One of these two outcomes is guaranteed to eventually occur. In particular, by our construction of $\mathscr{U}$ and the corresponding constraints in (8), the problem will become infeasible if all of the allotted subsystem training budgets N_{max} have been exhausted and a satisfactory meta-policy still does not exist. In such circumstances the task designer may wish to lower δ, to increase N_{max}, or to further decompose the task using additional subtasks.

Numerical Examples

In this section, we present the results of applying the proposed framework to the labyrinth navigation task used as a running example throughout the paper. We begin by discussing the results obtained using a discrete gridworld implementation of the labyrinth. However, to help demonstrate the framework's generality, we also present results for a continuous-state and continuous-action labyrinth, whose dynamics are goverened by a rigid-body physics simulator. Project code is available at: *github.com/cyrusneary/verifiable-compositional-rl*.

Figure 2a illustrates the labyrinth environment, and highlights each subtask with a different color, matching the colors used to represent the different subtasks in the presentation of the numerical results. Recall that the overall task specification is to safely navigate from the labyrinth's initial state in the top left corner to the goal state marked by a green square in the bottom left corner, with a probability of at least 0.95.

Discrete Gridworld Labyrinth Environment. We implement the gridworld labyrinth environment using MiniGrid (Chevalier-Boisvert, Willems, and Pal 2018). The environment's state space consists of the current position and orientation within the labyrinth, resulting in 1600 total states. The allowed actions are: *turn left*, *turn right*, and *move forward*. A slip probability is added to the environment dynamics to render them stochastic; each action has a 10% probability of accidentally causing the result of a different action to occur. Subtask entry $\mathscr{I}_c$ and exit $\mathscr{F}_c$ conditions are implemented as finite collections of states.

ICRL Algorithm Implementation. Each RL subsystem is trained using the Stable-Baselines3 (Raffin et al. 2021) implementation of the proximal policy optimization (PPO) algorithm (Schulman et al. 2017). Whenever estimates of task or subtask success probabilities are needed, we roll out the corresponding (sub)system 300 times from initial states randomly sampled from $\mathscr{I}_c$, and compute the empirical success rate. We solve the bilinear program in (1)–(5) using *Gurobi* (Gurobi Optimization, LLC 2021). Gurobi transforms the bilinear program into an equivalent mixed-integer linear program, and computes a globally optimal solution to this program by using cutting plane and branch and bound methods. For further details please see the supplementary materials in the extended version of the paper (Neary et al. 2021a).

Empirical Validation of Theorem 1. At regular intervals during training, marked by diamonds in Figure 3, each subsystem's probability of subtask success is estimated and used to update $\mathscr{L}$ and $\mathscr{U}$, as described in the previous section.

Subsystem Index	0	1	2	3	4	5	6	7	8	9	10	11
p_c at $t = 6e5$	.97	.00	.00	.00	.97	1.0	.00	.00	.00	1.0	.00	.57
p_c at $t = 10e5$	.95	.99	.00	.99	.88	1.0	.00	.00	.99	1.0	.99	.99

Table 1: Demonstration of automatic subtask specification refinement. Each value corresponds to a subtask specification, i.e. the minimum allowable probability of subtask success. The two rows of the table show these values at two distinct points of the system's training; before and after the subtask specification refinement illustrated by the dotted red lines in Figure 3. The cells highlighted in grey indicate which subsystems are used by the meta-policy, at the specified point.

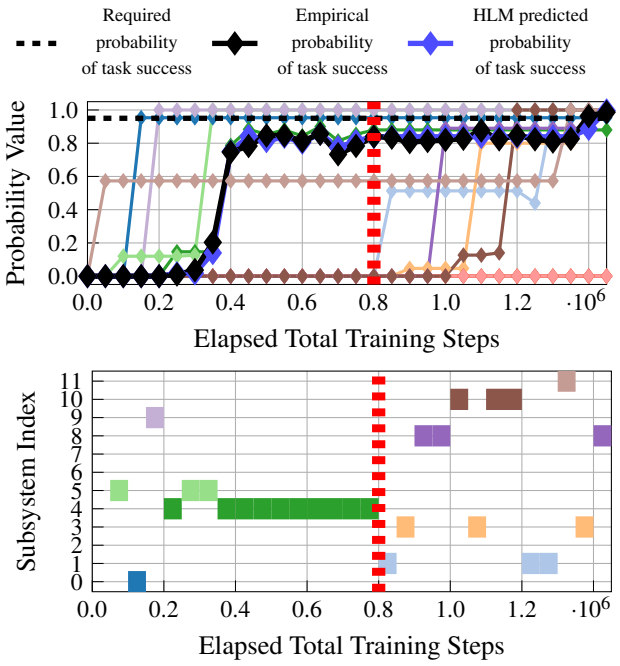


Figure 3: Discrete labyrinth experimental results. Top: Estimated task and subtask success probabilities during training. Bottom: Automatically generated subsystem training schedule. Each subtask is represented by a different color, matching those used in Figure 2a. The dotted red lines illustrate the point in training at which the HLM automatically refines the subtask specifications. Step counts do not include the rollouts used to estimate subtask success probabilities.

That is, each diamond in Figure 3 corresponds to a pass through the main loop of algorithm 1. The HLM-predicted probability of the meta-policy completing the overall task is illustrated in Figure 3 by the navy blue curve. For comparison, we plot empirical measurements of the success rate of the meta-policy in black. We clearly observe that the HLM predictions closely match the empirical measurements.

Subtask Specification Refinements Lead to Meta-Policy Adaptation and Targeted Subsystem Training. Figure 3 illustrates the subsystem training schedule. Table 1 lists the values of p_c for each subsystem c. We observe from Table 1 that prior to $8e5$ elapsed training steps, the value of p_c is only specified to be close to 1.0 for subsystems c_0, c_4, c_5, and, c_9. As can be seen in Figure 2a, these are the subsystems needed to move straight down, through the rooms containing lava, to the goal. The HLM has selected a meta-policy that will only use these subsystems because their composition yields the shortest path to goal; this path only requires training of 4 of the subsystems. Furthermore, because the meta-policy does not use any of the other subsystems, it places no requirements on their probability of subtask success. Figure 3 agrees with this observation: only this small collection of the subsystems are trained prior to $8e5$ elapsed training steps. In particular, subsystem 4, which must navigate the top lava room and is represented by dark green, is trained extensively. However, due to the environment slip probability, this subsystem is unable to meet its subtask specification, *safely navigate to the room's exit with probability 0.97*, regardless of the number of training iterations it receives. As a result, subsystem 4 exhausts its individual training budget after $8e5$ elapsed system training steps, marked by the vertical dotted red lines in Figure 3. At this point, subsystem 4's empirically estimated success rate of 0.88 is used to update the HLM, which then refines the subtask specifications. The result of this refinement is a new meta-policy, which instead uses subsystems c_1, c_3, c_8, c_{10}, and c_{11} to take an alternate path that avoids the lava rooms altogether. The updated subtask specifications are listed in the second row of Table 1, and in Figure 3 we observe a distinct change in the subsystems that are trained. Once subsystems c_1, c_3, c_8, c_{10}, and c_{11} learn to satisfy their new subtask specifications with the required probability, the composite system's probability of task success rises above 0.95, satisfying the overall task specification.

Comparison to a Monolithic RL Approach. The proposed ICRL algorithm takes less than two million training steps to satisfy the task specification. By comparison, a monolithic approach in which the entire task is treated as a single subsystem takes roughly thirty million training steps. We note that this is not a fair comparison because the proposed compositional approach has a priori knowledge of the subsystem entry and exit conditions. However, such information is often available through natural decompositions of complex systems. The proposed framework provides a method to take advantage of such information when it is available.

Results in a Continuous Labyrinth Environment. To demonstrate the framework's ability to generalize to different RL settings, we also implemented a continuous-state and continuous-action version of the labyrinth environment in the video game engine *Unity* (Juliani et al. 2018). In this version of the task, the RL system must roll a ball from the initial location to the goal location. The set A of available actions consists of all of the force vectors, with magnitude of at most

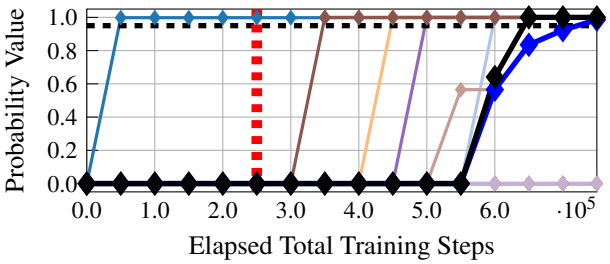


Figure 4: Continuous labyrinth experimental results.

1, that can be applied to the ball in the horizontal plane. The set S of environment states is given by all possible locations (x, y) and velocities $(\dot{x}, \dot{y})$ of the ball within the labyrinth. The action space A is thus a compact subset of $\mathbb{R}^2$ while the state space S is a compact subset of $\mathbb{R}^4$. The transition dynamics are governed by *Unity*'s rigid-body physics simulator. Subtask entry $\mathscr{I}_c$ and exit $\mathscr{F}_c$ conditions are implemented as subsets of $\mathbb{R}^4$ such that $\sqrt{(x-x_c)^2+(y-y_c)^2} \leq 0.5m$ and $\sqrt{\dot{x}^2+\dot{y}^2} \leq 0.5\frac{m}{s}$ respectively, for some pre-specified x_c and y_c. We use the PPO algorithm to train the RL subsystem policies. Each RL subsystem receives rewards that are proportional to its negative distance to the exit conditions, and incurs a large penalty if the lava is touched. We refer to the extended version of the paper for additional details and figures of this continuous environment (Neary et al. 2021a).

Figure 4 illustrates the experimental results in the continuous labyrinth environment. Qualitatively, these results closely resemble our observations from the discrete labyrinth, despite significant differences in the environment's dynamics and in its representations of states and actions. The ICRL algorithm again initially attempts to move straight down past the lava, before automatically refining the subtask specifications in order to focus on training the subsystems that take the alternate route through the labyrinth. This similarity in the algorithm's behavior when applied to different types of environments helps illustrate the generality of the proposed framework; ICRL is agnostic to the details of the environment dynamics and of the individual RL subsystems.

Additional Discussion. We note that all predictions made using the HLM will be sensitive to the values of $\hat{\sigma}_c$ – the estimated lower bounds on the probability of subtask success. In our experiments, we compute $\hat{\sigma}_c$ empirically by rolling out the subsystems from randomly sampled entry conditions. While this technique provides only rough estimates of the true value of the lower bound (particularly in the case of the continuous labyrinth environment which has an uncountably infinite number of entry conditions per subtask), our results demonstrate that these empirical approximations are sufficient for high-level decision making. The algorithm makes effective use of the HLM predictions to automatically select the subsystems that require training. Any methods to further improve the estimates of $\hat{\sigma}_c$ will only improve the performance of the ICRL algorithm.

Related Work

While the proposed framework is closely related to hierarchical RL (HRL) (Sutton, Precup, and Singh 1999; Barto and Mahadevan 2003; Kulkarni et al. 2016; Vezhnevets et al. 2017; Nachum et al. 2018; Levy et al. 2019), our framework adds several benefits to existing HRL methods. These benefits include: a systematic means to decompose and to refine task specifications, explicit reasoning over the probabilities of events, the use of planning-based solution techniques (which could incorporate additional problem constraints), and flexibility in the choice of RL algorithm used to learn subsystem policies. HRL methods use task decompositions to reduce computational complexity, particularly in problems with large state and action spaces (Pateria et al. 2021). However, they typically focus on the efficient maximization of discounted reward and they require the meta-policy to be learned; no model of the high-level problem is explicitly constructed. By contrast, we present a framework that builds a model of the high-level problem with the specific aim of enabling verifiable RL against a rich set of task specifications (*e.g., safely reach a target set with a required probability of success*), while enjoying a similar reduction in sample complexity.

Compositional verification has been studied in formal methods (Nam, Madhusudan, and Alur 2008; Feng, Kwiatkowska, and Parker 2011), but not in the context of RL. Conversely, recent works have used structured task knowledge to decompose RL problems, however, they do not study how such information can be used to verify RL systems. Camacho et al. (2017) and Littman et al. (2017) both define a task specification language based on linear temporal logic, and subsequently use it to generate reward functions for RL. Sarathy et al. (2021) incorporates RL with symbolic planning models to learn new operators – similar to our subtasks – to aid in the completion of planning objectives. Meanwhile, Toro Icarte et al. (2018, 2019); Xu et al. (2020); Toro Icarte et al. (2022) use reward machines, finite-state machines encoding temporally extended tasks in terms of atomic propositions, to break tasks into stages for which separate policies can be learned. Neary et al. (2021b) extends the use of reward machines to the multi-agent RL setting, decomposing team tasks into subtasks for individual learners. These works all use structured task knowledge to decompose RL problems, however, they do not provide methods for the automated verification and decomposition of task success probabilities, or for the targeted training of subsystems.

Conclusions

The verification of reinforcement learning (RL) systems is a critical step towards their widespread deployment in engineering applications. We develop a framework for verifiable and compositional RL in which collections of RL subsystems are composed to achieve an overall task. We automatically decompose system-level task specifications into individual subtask specifications, and iteratively refine these subtask specifications while training subsystems to satisfy them. Future directions will study extensions of the framework to multi-level task hierarchies, compositional multi-agent RL systems, and to systems involving partial information.

Acknowledgements

This work was supported in part by ONR N00014-20-1-2115, ARO W911NF-20-1-0140, and NSF 1652113.

References

Baier, C.; and Katoen, J.-P. 2008. *Principles of model checking*. MIT press.

Barto, A. G.; and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1): 41–77.

Camacho, A.; Chen, O.; Sanner, S.; and McIlraith, S. A. 2017. Non-markovian rewards expressed in LTL: guiding search via reward shaping. *10th Annual Symposium on Combinatorial Search*.

Chevalier-Boisvert, M.; Willems, L.; and Pal, S. 2018. Minimalistic Gridworld Environment for OpenAI Gym. https://github.com/maximecb/gym-minigrid.

Cubuktepe, M.; Jansen, N.; Junges, S.; Katoen, J.-P.; and Topcu, U. 2018. Synthesis in pMDPs: A Tale of 1001 Parameters. In *International Symposium on Automated Technology for Verification and Analysis*, 160–176. Springer.

Etessami, K.; Kwiatkowska, M.; Vardi, M. Y.; and Yannakakis, M. 2007. Multi-objective model checking of Markov decision processes. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 50–65. Springer.

Feng, L.; Kwiatkowska, M.; and Parker, D. 2011. Automated learning of probabilistic assumptions for compositional reasoning. In *International Conference on Fundamental Approaches to Software Engineering*, 2–17. Springer.

Gurobi Optimization, LLC. 2021. Gurobi Optimizer Reference Manual. https://www.gurobi.com/documentation/9.1/refman/index.html. Accessed: 2021-12-15.

Haberfellner, R.; Nagel, P.; Becker, M.; Büchel, A.; and von Massow, H. 2019. *Systems engineering*. Springer.

Hahn, E. M.; Perez, M.; Schewe, S.; Somenzi, F.; Trivedi, A.; and Wojtczak, D. 2019. Omega-regular objectives in model-free reinforcement learning. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 395–412. Springer.

Juliani, A.; Berges, V.-P.; Teng, E.; Cohen, A.; Harper, J.; Elion, C.; Goy, C.; Gao, Y.; Henry, H.; Mattar, M.; et al. 2018. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.

Junges, S.; Abraham, E.; Hensel, C.; Jansen, N.; Katoen, J.-P.; Quatmann, T.; and Volk, M. 2019. Parameter Synthesis for Markov Models. arXiv:1903.07993.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In *Advances in Neural Information Processing Systems*, volume 29.

Levy, A.; Konidaris, G. D.; Platt, R. W.; and Saenko, K. 2019. Learning Multi-Level Hierarchies with Hindsight. In *International Conference on Learning Representations*.

Littman, M. L.; Topcu, U.; Fu, J.; Isbell, C.; Wen, M.; and MacGlashan, J. 2017. Environment-Independent Task Specifications via GLTL. arXiv:1704.04341.

Nachum, O.; Gu, S. S.; Lee, H.; and Levine, S. 2018. Data-Efficient Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 31.

Nam, W.; Madhusudan, P.; and Alur, R. 2008. Automatic Symbolic Compositional Verification by Learning Assumptions. *Formal Methods in System Design*, 32(3): 207–234.

Neary, C.; Verginis, C.; Cubuktepe, M.; and Topcu, U. 2021a. Verifiable and Compositional Reinforcement Learning Systems. *arXiv preprint arXiv:2106.05864*.

Neary, C.; Xu, Z.; Wu, B.; and Topcu, U. 2021b. Reward Machines for Cooperative Multi-Agent Reinforcement Learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, 934–942.

Nuseibeh, B.; and Easterbrook, S. 2000. Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, 35–46.

Pateria, S.; Subagdja, B.; Tan, A.-h.; and Quek, C. 2021. Hierarchical Reinforcement Learning: A Comprehensive Survey. *ACM Computing Surveys (CSUR)*, 54(5): 1–35.

Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; and Dormann, N. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268): 1–8.

Sarathy, V.; Kasenberg, D.; Goel, S.; Sinapov, J.; and Scheutz, M. 2021. SPOTTER: Extending Symbolic Planning Operators through Targeted Reinforcement Learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, 1118–1126.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347.

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

Toro Icarte, R.; Klassen, T.; Valenzano, R.; and McIlraith, S. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. *International Conference on Machine Learning*, 2107–2116.

Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2022. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73: 173–208.

Toro Icarte, R.; Waldie, E.; Klassen, T.; Valenzano, R.; Castro, M.; and McIlraith, S. 2019. Learning Reward Machines for Partially Observable Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 32.

Vezhnevets, A. S.; Osindero, S.; Schaul, T.; Heess, N.; Jaderberg, M.; Silver, D.; and Kavukcuoglu, K. 2017. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, 3540–3549. PMLR.

Xu, Z.; Gavran, I.; Ahmad, Y.; Majumdar, R.; Neider, D.; Topcu, U.; and Wu, B. 2020. Joint inference of reward machines and policies for reinforcement learning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 30: 590–598.