Specification-Guided Reinforcement Learning

Kishor Jothimurugan KISHOR@TWOSIGMA.COM
Two Sigma Investments, LP[*]

Suguman Bansal SUGUMAN@GATECH.EDU
Georgia Institute of Technology

Osbert Bastani OBASTANI@SEAS.UPENN.EDU
University of Pennsylvania

Rajeev Alur ALUR@SEAS.UPENN.EDU
University of Pennsylvania

Editors: G. Pappas, P. Ravikumar, S. A. Seshia

Abstract

This tutorial explores specification-guided reinforcement learning as an alternative to traditional reward-based approaches, where the design of effective reward functions can be tedious, error-prone, and may not capture complex objectives. We introduce formal logical specifications as a more intuitive and precise way to define agent behavior, focusing on the theoretical guarantees and algorithmic aspects of learning from specifications. We examine both fundamental limitations in infinite-horizon settings and practical approaches for finite-horizon specifications.

Keywords: Reinforcement Learning, Formal Specifications, Theoretical Guarantees, Practical Algorithms

1. Introduction

Reinforcement learning (RL) (Sutton and Barto, 2018) enables autonomous agents to learn from experience by interacting with their environment and receiving feedback. This feedback traditionally takes the form of numerical rewards—positive values for desired outcomes and negative values for mistakes. While this paradigm has led to remarkable successes, the challenge of designing effective reward functions remains a significant bottleneck in deploying RL systems (Amodei et al., 2016; Pan et al., 2022).

Consider a warehouse automation task: teaching a robot to transport boxes between locations. The robot must receive appropriate rewards for moving boxes correctly while avoiding collisions. The design of such reward functions involves several critical decisions including the timing of reward signals during task execution, the relative magnitudes of rewards for successful movements versus penalties for collisions, and the granularity of feedback throughout the task progression. These decisions fundamentally shape the learning process and resulting behavior.

The fundamental challenge is that reward engineering conflates two distinct problems: *task specification* (what the agent should accomplish) and *reward shaping* (what rewards will guide the agent to learn effectively). While the high-level goal might be "safely transport boxes to designated

[*] The views expressed herein are solely the views of the author(s) and are not necessarily the views of Two Sigma Investments, LP or any of its affiliates. They are not intended to provide, and should not be relied upon for, investment advice.

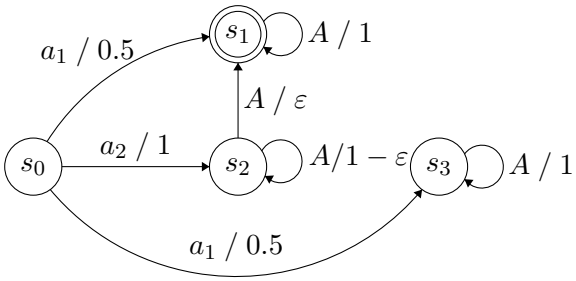

Figure 1: A Finite MDP.

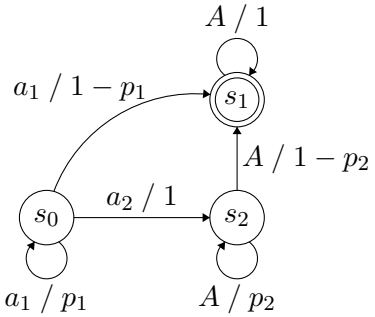

Figure 2: MDP for demonstrating PAC hardness result.

Action A on transitions denotes that any action can be taken.

locations," translating this into precise numerical rewards requires careful balancing of multiple factors, including extending the state-space of rewards to auxiliary variables.

This tutorial introduces an alternative approach: using *logical specifications* to describe desired behaviors directly. We focus particularly on *temporal logics*, such as Linear Temporal Logic (LTL) (Pnueli, 1977) and SPECTRL (Jothimurugan et al., 2021), which provide rigorous syntax and semantics for reasoning about sequences of events and states over time.

For our warehouse robot example, instead of crafting complex numerical rewards, we can express the task using predicates g for target location and o for obstacles as: Eventually $g \wedge$ Always $\neg o$. This specification is more intuitive and precise than traditional reward functions. The central challenge lies in training an RL agent to satisfy such logical specifications φ with high probability. In this tutorial we will cover the theoretical and practical implications of designing RL algorithms from logical specifications.

2. RL from Rewards

The objective of RL is to train an agent to act in an environment to accomplish a specific task – e.g., learn what action should be performed in any given state so that the robot will eventually reach a specific room. The key feature of RL is that the environment is assumed to be *unknown*. Hence, the *policy* that enables the agent to act must be learned by exploring the environment. The problem is formally described below:

Markov decision processes. The environment in RL is modeled as a *Markov Decision Process (MDP)*. An MDP is a tuple $\mathcal{M} = (S, A, \eta_0, T)$, where S is a set of *states*, η_0 is the initial state distribution, A is a set of *actions*, and $T : S \times A \times S \to [0, 1]$ is the *transition probability function* with $\sum_{s' \in S} T(s, a, s') = 1$ for all $s \in S$ and $a \in A$. Fig. 1 shows an MDP with states $S = \{s_0, s_1, s_2, s_3\}$ and actions $A = \{a_1, a_2\}$. Actions cause state transitions according to transition probabilities. For example, a_1 from s_0 leads to s_1 or s_3 with equal probability. The *initial state distribution* η describes the distribution of states the system initializes in. Here η is the dirac distribution at s_0, i..e, with $\eta(s_o) = 1$. An *infinite run* of the MDP is an infinite sequence of state-action pairs describing the evolution of the system over time. A possible infinite run in Fig 1 is $\rho = s_0 \xrightarrow{a_2} s_2 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_1 \xrightarrow{a_2} \cdots$ in which the agent reaches s_1 from s_0 via s_2.

Similarly, a *finite run* of length H describes the evolution of the system for H time steps – e.g., $\rho = s_0 \xrightarrow{a_2} s_2 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_1 \xrightarrow{a_2} s_1$ is a finite run of length 4. A *policy* $\pi : S \to D(A)$ maps states to a distribution on actions. Each policy induces a distribution over runs generated by sampling an initial state from η and iteratively using π to select actions and sample next states from T. We denote the distribution over infinite runs as $\mathcal{D}^\pi$ and over finite H-length runs as $\mathcal{D}^\pi_H$.

Simulator. In RL, the standard assumption is that the set of states S, the set of actions A, and the initial state distribution η_0 are known but the transition probability function T is unknown. The agent must learn about the environment by taking actions and observing transitions. For this, the learning algorithm has access to a *simulator* $\mathcal{S}$ which can be used to sample runs of the system $\zeta \sim \mathcal{D}^{\mathcal{M}}_\pi$ using any policy π. The simulator can also be the real system, such as a robot, that $\mathcal{M}$ represents. Internally, the simulator stores the current state of the MDP which is denoted by $\mathcal{S}.\texttt{state}$. It makes the following functions available to the learning algorithm.

$\mathcal{S}.\texttt{reset()}$: This function sets $\mathcal{S}.\texttt{state}$ to an initial state s_0 sampled from the initial state distribution i.e. $s_0 \sim \eta_0$.

$\mathcal{S}.\texttt{step}(a)$: Given as input an action a, this function samples a state $s' \in S$ according to the transition probability function T—i.e., the probability that a state s' is sampled is $P(s, a, s')$ where $s = \mathcal{S}.\texttt{state}$. It then updates $\mathcal{S}.\texttt{state}$ to the newly sampled state s' and returns s'.

The simulator lets an RL agent sample multiple runs in the MDP without knowing the transition probabilities. This guides the RL agent to estimate the transition probability T. For e.g. in Fig 1 if the agent repeatedly takes action a_1 in state s_0, it observes that the system transitions to s_1 approximately half the time.

Task Description. Traditionally tasks in RL are specified using a *reward function* $r : S \times A \to \mathbb{R}$ that assigns rewards for state-action pairs. For example, to reach s_1 in Fig. 1, we might set $r(s, a) = \mathbb{1}(s = s_1)$. For a run ρ, this naturally generates a reward sequence (e.g., $0, 1, 1, \ldots$ for a run reaching s_1 in one step). For an infinite horizon, the *expected discounted-sum* of a policy π is given by $J(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$ where the random variable r_t denotes the reward obtained at time t and *discount factor* $0 < \gamma < 1$ indicates the rate of diminishing returns. In Fig 1, under the reward function $r(s, a) = \mathbb{1}(s = s_1)$, the policy π_1 that chooses action a_1 in state s_0 will achieve a discounted-sum reward of $0 + \gamma + \gamma^2 + \cdots = \gamma/(1-\gamma)$ with probability 0.5 (when it reaches s_0) and 0 with probability 0.5 (when it reaches s_3). Hence $J(\pi_1) = \gamma/(2(1-\gamma))$. Similarly, the expected discounted-sum reward of policy π_2 that chooses action a_2 in s_0 is $\varepsilon\gamma^2/(1-\gamma) \cdot \sum_{k=0}^{\infty}(1-\varepsilon)^k\gamma^k = \varepsilon\gamma^2/[(1-\gamma)(1+\gamma\varepsilon-\gamma)]$. In the infinite horizon, RL algorithms typically maximize the expected value of a policy:

Definition 1 (RL from Rewards) *Given an* unknown *MDP with access to its simulator, a reward function, and a discount factor, the problem of RL from rewards is to learn a policy π^* that maximizes its expected discounted sum i.e. $\pi^* = \arg\max_\pi J(\pi)$.*

If instead given a finite horizon H, then RL algorithms maximize the expected reward $J_H(\pi) = \mathbb{E}[\sum_{t=0}^{H-1} r_t]$.

Algorithmic Approaches and Guarantees. RL with discounted-sum rewards has been extensively studied and provides several theoretical guarantees that make it attractive for practical applications. The problem is to learn an optimal policy in an unknown MDP. Had the MDP been known, i.e., if the transition probabilities were known, then the optimal policy could have been obtained using seminal results in MDP optimization, such as value iteration and policy iteration (Puterman, 2014). In the absence of the transition probabilities, intuitively, RL learns the optimal policy by repeatedly sampling the environment to (explicitly or implicitly) estimate its transition probabilities, then obtaining the optimal policy in the estimated MDP. Key results include:

Probably Approximately Correct (PAC) Guarantees (Valiant, 2013) These algorithms can efficiently learn near-optimal policies with high confidence using a finite number of samples, providing formal guarantees on the learning process (Kakade, 2003; Kearns and Singh, 2002).

Robustness Guarantees Discounted-sum rewards are robust in the sense that altering the transition probabilities by a small amount only impacts the value $J(\pi)$ of a policy π by a small amount (more precisely, $J(\pi)$ (or $J_H(\pi)$) is a continuous function of the MDP transitions and rewards).

Convergence Properties Discounted rewards naturally prioritize near-term outcomes, allowing algorithms such as Q-learning (Watkins and Dayan, 1992) and policy gradient methods (Sutton and Barto, 2018; Williams, 1992) to converge to optimal or near-optimal policies.

3. RL from Logical Specifications

Like earlier, the objective of RL from logical specifications is to learn a policy that satisfies a task in an unknown environment. The environment and the simulator are defined as earlier. The difference lies in the task description which is expressed in the form of a logical specification in this case.

Task Description A specification φ is a logical formula encoding whether a finite or infinite run of the MDP solves the desired task. The meaning of a specification is given by the *semantics* function (denoted $[\![\varphi]\!]$) that maps a run ρ to $\{0, 1\}$ indicating whether or not the run satisfies φ. The probability of a policy π to satisfy a task φ is given by $J^{\varphi}(\pi) = \mathbb{E}_{\rho \sim \mathcal{D}^{\pi}}[[\![\varphi]\!](\rho)]$.

Logical specifications have several advantages over reward functions w.r.t task specification. They have natural language-like syntax and rigorous semantics, making task specification less cumbersome. For example, the task of reaching s_1 can be encoded by the specification $\varphi = \mathsf{Eventually}\ s_1$. A run that starts in s_0 and transitions immediately to s_3 will never reach s_1, and therefore will not satisfy φ. Logical specifications can be straightforwardly composed to express more complex tasks; for instance, the task of sequentially visiting two locations l_1 and l_2 can be simply expressed as $\mathsf{Eventually}\ l_1; \mathsf{Eventually}\ l_2$. Furthermore, several tasks cannot be expressed as rewards but are easily expressed as logical specifications. For example, consider the task of going from an initial state s to a target state t then returning back to the state s. This is expressed in logical specifications as $\mathsf{Eventually}\ t; \mathsf{Eventually}\ s$. However, it cannot be expressed in the form of rewards $r : S \times A \to \mathbb{R}$ because the reward in a state along the path from s to t and back depends on which segment of the task it is completing. The reward function requires *memory* to store whether the agent is currently heading towards s or towards t. Memory is is not permitted by rewards of the form $r : S \times A$.

Definition 2 (RL from Logical Specifications) *Given an unknown MDP and a specification φ, the problem of RL from logical specifications (or specification-guided RL) is to learn a policy π^* that maximizes the probability of satisfying the task φ, i.e. $\pi^* = \arg\max_{\pi} J^{\varphi}(\pi)$.*

In Fig 1, the policy π_1 that chooses action a_1 in s_0 assigns a probability of 0.5 to each of the two infinite runs it produces (one that visits s_1 and the other than does not), so $J^{\varphi}(\pi_1) = 0.5$. On the other hand, the policy π_2 that chooses action a_2 in s_0 results in infinitely many trajectories (depending on how long they remain in the state s_2), but all of these trajectories eventually visit s_1. Thus, $J^{\varphi}(\pi_2) = 1$. Clearly, π_2 is optimal.

4. RL from Infinite-Horizon Specifications

This section discusses results on RL form *infinite-horizon* specifications. These specifications are important as they can express recurring tasks and persistent behaviors that finite-horizon specifications cannot capture. For instance, tasks like "repeatedly monitor an area" are naturally infinite-horizon properties. *Linear Temporal Logic (LTL)* (Pnueli, 1977) is a prominent example of an infinite-horizon specification language.

4.1. Algorithmic Approach

A popular approach to solving RL from infinite-horizon specifications is via a reduction to RL from discounted-sum rewards. Here the specification φ is first compiled into a reward function r_{φ} and then a standard RL algorithms for discounted rewards is applied. This approach is popular as learning from rewards guarantees to learn optimal policies and state-of-the-art implementations are readily available. However, this approach does not guarantee learning optimal policies w.r.t. specifications:

Theorem 3 *(Alur et al., 2021) Under the assumption that the MDP is unknown, there exist infinite-horizon specifications φ for which no discounted reward function r exists (for any discount factor γ) such that the optimal policy with respect to reward r is also optimal w.r.t to specification φ.*

Proof [Proof Sketch] This result arises due to a fundamental mismatch between how discounted rewards and logical specifications measure task completion – discounted rewards inherently care about when the task is completed since future rewards are time-discounted, whereas logical specifications such as Eventually s_1 care only about the task eventually being completed. This mismatch is illustrated by the specification Eventually s_1 and the reward function $r(s, a) = \mathbb{1}(s = s_1)$ in Fig 1. Recall that the policy π_2 which takes action a_2 in state s_0 is optimal w.r.t. Eventually s_1. However, this policy can be made suboptimal w.r.t. $r(s, a) = \mathbb{1}(s = s_1)$ for any discount factor γ by choosing an appropriate value of ε. ■

This argument can further be extended to prove the impossibility of specification translation even for history-dependent reward functions (Alur et al., 2021). This impossibility result has profound implications: there are specifications that simply cannot be solved by the strategy of reduction to reward functions. Recent work shows some promising directions under additional assumptions: For instance, (Bozkurt et al., 2020) has shown that any LTL specification can be reduced to discounted rewards with an appropriate discount factor that depends on the MDP transitions. Another approach is based on reducing the problem of learning an optimal policy w.r.t. an LTL specification to learning an optimal policy for a reachability objective in an MDP obtained by taking the product of the MDP with a *good-for-MDP* automaton corresponding to the LTL specification (Hahn et al., 2020a). These reachability objectives can be reduced to learning from discounted-sum rewards, however, the corresponding discount-factor depends on the MDP transition probabilities as well (Hahn et al.,

2020b). A novel framework of *eventual discounting* has been introduced that preserves optimal behaviors (Voloshin et al., 2023). More recently, (Alur et al., 2023) have studied the use of *discounted LTL*, which naturally prioritizes near-term events over distant ones, allowing a direct reduction to discounted rewards.

4.2. Theoretical Guarantees

Given the impossibility of such reductions to rewards, the natural question to ask is whether there exists an RL algorithm with theoretical guarantees. A common theoretical framework for answering such questions is the *Probably Approximately Correct (PAC)* framework (Valiant, 2013): Given a confidence level $\delta > 0$ and an error tolerance $\epsilon > 0$, an RL algorithm is said to be PAC if after sampling a sufficient number of transitions (as a function of n, m, δ, ϵ) in an MDP with n states and m actions, the probability that the learned policy π is ϵ-close to optimal (i.e., $J^* - J(\pi) \leq \epsilon$) is at least $1 - \delta$, where J^* is the value of the optimal policy. Note that the required sample size does not depend on the transition probabilities, allowing agents to determine termination without knowing the true environment dynamics. While algorithms with PAC guarantees exist for discounted rewards (Kakade, 2003; Kearns and Singh, 2002), surprisingly, they are impossible for even simple reachability specifications such as Eventually g (where g is a goal state). This occurs due to the lack of robustness of logical specifications, where robustness refers to small changes in the transition probability resulting in small changes to the policies. Formally,

Theorem 4 *(Littman et al., 2017) Infinite-horizon logical specifications are not robust, i.e. small changes to the transition probabilities can result in significant changes to the policies.*

Proof Consider the MDP shown in Fig 2 and the specification Eventually s_1. When $p_1 = 1$ and $p_2 < 1$, the optimal policy chooses action a_2 in state s_0 since the probability of eventually reaching s_1 from s_2 is 1, whereas the probability of reaching s_1 from s_0 is 0 if the agent chooses action a_1 in state s_0. Similarly, when $p_1 < 1$ and $p_2 = 1$, the optimal action at state s_0 is a_1. Note that altering the transition probabilities slightly can significantly impact the optimal policy. For example, if $p_1 = 1$ and $p_2 = 1 - x$ for an infinitesimally small x, altering the values to $p_1' = p_1 - x = 1 - x$ and $p_2' = p_2 + x = 1$ causes the optimal action at s_0 to change from a_2 to a_1. Furthermore, an optimal policy in the original MDP (with $p_1 = 1$) that reaches s_1 with probability 1 will attain a zero probability of reaching s_1 in the new MDP. This example illustrates that the specification Eventually s_1 is not robust – small changes to the MDP can drastically affect the corresponding optimal policy. ■

This lack of robustness is why PAC algorithms do not exist for infinite-horizon specifications:

Theorem 5 *(Alur et al., 2021; Yang et al., 2021) No RL algorithm with infinite-horizon specifications offers PAC guarantees.*

Proof [Proof Sketch] For the sake of contradiction, suppose there is an RL algorithm with a PAC guarantee for Eventually s_1. We can run this algorithm for the above two scenarios with slightly different transition probabilities and the same values of $\delta > 0.9$ and $\epsilon < 1$. Then, for a sufficiently small x, it is highly likely that the RL algorithm will see the same samples from the two different MDPs. Thus, the RL algorithm will output the same policy π, meaning it must attain a zero

probability of reaching s_1 in one of the two MDPs. As a result, $J^* - J(\pi) = 1 > \epsilon$ for that MDP, which is a contradiction to the PAC property. ■

This result is extremely powerful as our definition of PAC does not assume a polynomial number of samples. Our negative result holds for an arbitrary function in the inputs.

Furthermore, (Yang et al., 2021) show that PAC guarantees are possible precisely for *finitary* specifications (an infinite-horizon specification φ is *finitary* if there exists a fixed H such that we can decide whether or not an infinite run $\rho = (s_0, s_1, \ldots)$ of the system satisfies φ by only looking at the first H steps). PAC algorithms also exist under additional assumptions on the MDP – e.g., if a lower bound on all non-zero transition probabilities of the MDP is given (Ashok et al., 2019; Daca et al., 2016), mixing time of the MDP (Perez et al., 2024), known topology of the underlying MDP (Fu and Topcu, 2014), and stochastic bounds on the length of runs (Svoboda et al., 2024).

5. Finite Horizon Specifications

While infinite horizon poses many theoretical challenges for specification-guided RL, in practice, it is often sufficient to consider finitary specifications. Here we are additionally given a fixed finite horizon H, and the satisfaction of the specification φ only depends on the first H steps of the MDP. Finite-horizon specifications are suitable for properties such as reachability (including multiple reachable targets) and bounded safety.

5.1. Theoretical Guarantees

Unlike the infinite-horizon case, there exists a simple optimality-preserving reduction of (finite-horizon) specification to rewards:

Theorem 6 *There exists a reduction of finite-horizon specifications to discounted-sum rewards such that optimal policy w.r.t. rewards is also optimal w.r.t. specification.*

Proof Given a specification φ and a finite-horizon specification H, a straightforward approach is to define a reward function r_φ that assigns a reward of 1 after H steps if φ is satisfied and 0 otherwise. It is easy to see that r_φ faithfully represents φ, and a policy π that maximizes the expected discounted-sum of rewards $J_H(\pi)$ also maximizes the probability of satisfying φ. ■

An immediate implication of this result is that RL from finite-horizon specifications offers strong theoretical guarantees, including the existence of algorithms with PAC and convergence guarantees, as the guarantees of RL from discounted-sum rewards over the finite-horizon simply translate over.

5.2. Algorithmic Approaches

At first glance, Theorem 6 seems to present an approach to solve RL from finite-horizon specifications: First, convert finite-horizon specification to discounted-sum rewards then apply an existing off-the-shelf finite-horizon discounted rewards based RL approach to learn an optimal policy. However, there are several challenges to designing practical specification guided RL algorithms. We briefly summarize a few of these challenges and their solution approaches below (Aksaray et al., 2016; Brafman et al., 2018; De Giacomo et al., 2019; Hasanbeig et al., 2018, 2019; Yuan et al., 2019; Xu and Topcu, 2019; Li et al., 2017):

Sparsity of Rewards. Sparse reward functions "rarely" assign non-zero rewards. As a result, they often require a large number of samples from the environment, impeding the speed of learning. Consider a warehouse robot navigating from region A to B within H steps. With sparse rewards, the agent receives feedback only upon reaching B, potentially requiring extensive exploration before earning any reward – a problem that worsens with complex specifications. *Reward shaping* is a technique used to generate *dense* reward functions which assign rewards more frequently to guide the agent towards solving the task (Li et al., 2017; Xu and Topcu, 2019). For example, we can define a reward function $r_\phi(s, a) = -\texttt{dist}(s, B)$ measuring the distance to B from the agent's current state s and the goal region B, which encourages the robot to move towards B. Another effective strategy is to use *quantitative semantics* for LTL specifications, which assign numerical values to runs of the system that capture how "robustly" the specification is satisfied (while ensuring runs that satisfy the specification are assigned higher values compared to those that do not) (Jothimurugan et al., 2019).

Non-Markovian Rewards. Another challenge with encoding specifications using reward functions is that specifications can be *non-Markovian* or *history dependent*. However, existing implementations of RL algorithms do not allow rewards to be history dependent – i.e., the reward at the current step $r(s, a)$ must be a function of the current state s and the action a and should be independent of the states visited and the actions taken prior to the current step. While history-dependent tasks are easy to encode using logical specifications, they cannot be directly represented as rewards. To handle history-dependent tasks, one approach is to alter the state space to include relevant information about the history – e.g., using *reward machine* (Icarte et al., 2018) or deterministic finite-state automata (De Giacomo et al., 2019).

Myopic Learning. RL algorithms tend to be myopic, focusing on immediate rewards over a long term ones. Consider our example specification φ_3; a typical reward function implementing this task would provide similar rewards for visiting X and E, since these two cannot be distinguished without understanding the structure of the MDP. In the MDP, there is no direct way to reach C from E, so in fact X is prefereable to E. Thus, if by chance an RL algorithm learns to solve the initial subtask $\mathsf{Eventually}\ X \vee \mathsf{Eventually}\ E$ by visiting E instead of X, then it might struggle to reach C. This requires techniques that combine high-level planning at the level of the specification with low-level reasoning of RL. We describe one such technique in detail next.

5.3. Compositional Approach to RL from Specifications

The benefits of compositional approaches in RL for long-horizon tasks are well noted (Vazquez-Chanlatte et al., 2018). We describe a compositional approach DIRL to learning long-horizon tasks (Jothimurugan et al., 2021). This leverages the structure of the specification, expressed in the specification language SPECTRL, to interleave high-level planning at the level of the specification with low-level RL.

Specification Language. We use the language SPECTRL (Jothimurugan et al., 2019). It can encode reachability, safety, and their combinations. Its given by: $\varphi := \mathsf{Eventually}\ p \mid \varphi\ \mathsf{Ensuring}\ p \mid \varphi; \varphi \mid \varphi \vee \varphi$ where $p \subset S$ is a predicate representing a subset of the states. $\mathsf{Eventually}\ p$ denotes eventually (within H steps) reaching a state $s \in p$, and $\varphi\ \mathsf{Ensuring}\ p$ denotes performing φ while remaining in the "safe" set p at every step. Furthermore, sequencing $\varphi_1; \varphi_2$ denotes first performing φ_1 and then φ_2, and disjunction $\varphi_1 \vee \varphi_2$ gives the agent the choice of either performing φ_1 or φ_2. Formal semantics of SPECTRL is given in Appendix B. In our warehouse example, the task of visiting B

while avoiding the obstacle region O (union of all regions marked in red) can be encoded in SPECTRL as $\varphi_1 = (\mathsf{Eventually}\ B\ \mathsf{Ensuring}\ \neg O)$, where $\neg O = S \setminus O$ are states that do not belong in O.

Compositional Algorithm. Consider a robot navigating in the warehouse (Fig 4). The warehouse environment with interconnected rooms shown in Fig 4). Here, a state is a pair of coordinates $s = (x, y) \in \mathbb{R}^2$ indicating the location of the robot, and actions are velocities $a = (v_x, v_y) \in \mathbb{R}^2$. The initial distribution over states is $\eta = \text{Uniform}(A)$ (where A is the purple circle). Consider the specification $\varphi = A \rightarrow B$ (Fig 9) specifying that a robot must first visit either top-corner C_1 or bottom corner C_2 and then visit B, all while avoiding red obstacles O in the warehouse environment (Fig 4). The optimal policy accomplishes the task via C_1 as there is no direct path from C_2 to B.

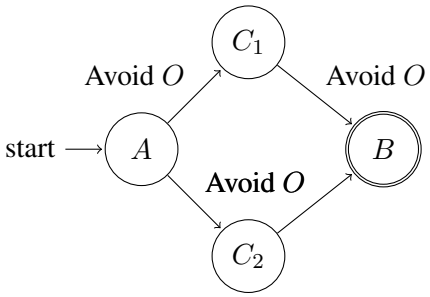

Figure 3: Example of an abstract graph.

To learn this policy, DIRL constructs an *abstract graph* G_φ from the specification φ which is a DAG representing a specification-guided abstraction of the environment. The abstract graph of our example specification φ is shown in Fig 3. Each vertex represents a set of states in the MDP. In this example, the abstract graph has four vertices representing the four regions A, C_1, C_2, and B. Each directed edge represents a *reachability subtask with a safety constraint* – namely, the subtask of reaching the set of states represented by the target vertex while adhering to the safety constraint. For instance, the edge from A to C_1 denotes the subtask of reaching C_1 from A while avoiding O.

The abstract graph maintains the property that all paths from the initial state to the goal state satisfy the original task, i.e. in our example, all path from A to B satisfy the input specification. Since there could be multiple such paths, our objective is to find the "best" path. We let the cost of each edge be the probability of satisfaction of the subtask corresponding to the edge. Thus, the "best" path is the one that maximizes the product of edge costs which is equivalent to maximizing the probability of completing φ.

Then, a simple algorithm would first run learning for subtasks on all edges, estimate the probability of satisfaction of the learned policy on each edge, and then compute the best path, sequentially. In practice, however, this approach does not work because to learn policies for edges starting from an intermediate state in the abstract graph, the learning agent requires an initial state-distribution on the intermediate state. Apriori, as per RL assumptions, we are given the initial-state distribution of the initial state of the abstract graph only. Hence, in order to accomplish these steps, the algorithm must also learn initial-state distributions of intermediate states. A heuristic to learn state-distributions on intermediate states is to induce a state-distribution using the policy along the best path to the intermediate state. As a result, the order of learning policies for edges and induced state-distributions should be such that for every state a state-distribution should be induced before policies are learned for outgoing edges from it. Since the underlying graph is a DAG, an efficient way ensuring this is to choose states and edges similar to the way Dijkstra's algorithm does for shortest path.

Algorithm Performance. While this approach no longer guarantees optimality, it has been shown to scale well to complex specifications, including realistic applications such as robotic Pick-And-Place environments, where it outperforms prior approaches while requiring fewer than 4×10^5 samples. Its performance in the warehouse environment on increasingly complex specifications has been shown in Figure 9.

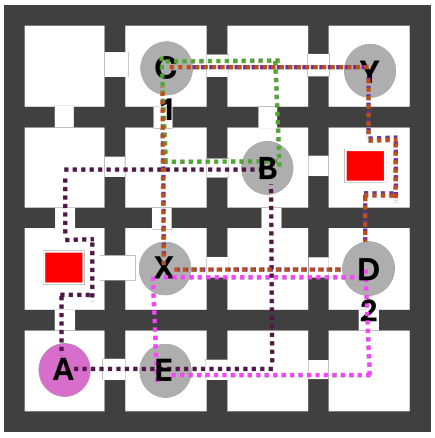

Figure 4: Warehouse environment.

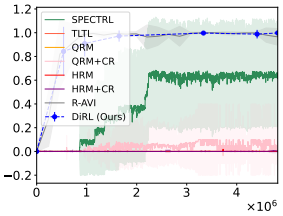

Figure 5: A→B

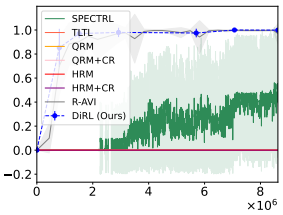

Figure 6: A→B→C

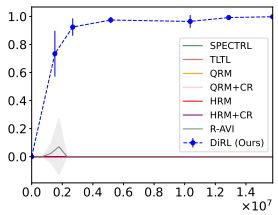

Figure 7: A→B→C→D

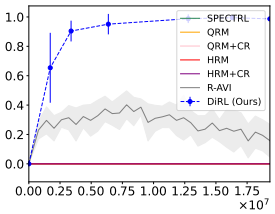

Figure 8: A→B→C→D →E

Figure 9: Performance plots showing probability of specification satisfaction (y-axis) across specifications of increasing complexity against number of samples taken to learn (x-axis). X→Y is shorthand for specification to reach region Y from region X after visiting one of the corners in the square/rectangle formed by X and Y while avoiding all red obstacles.

6. Other Related Works

Generalization. The lack of generalization in RL, i.e. the inability of RL-enabled policies to fulfil tasks other than the one they are trained on, is a widely-studied topic under reward-based RL (Beck et al., 2023; Finn et al., 2017; Zintgraf et al., 2021). Specification-guided RL is particularly suited to address the generalization problem as the logical structure can be leveraged to learn policies that can be adapted to tasks unseen during training time. These include multi-task RL (Jackermeier and Abate; Vaezipoor et al., 2021), goal-conditioned RL (Qiu et al., 2023), and inductive generalization (Inala et al., 2020; Subramanian et al., 2024).

RL Verification. There exists a vast literature on the verification of RL-enabled policies (Landers and Doryab, 2023). This line of work is orthogonal to specification-guided RL as the primary question here is to perform verification of the deep neural networks that represent the learned policy while also accounting for the sequential nature of the RL task, requiring the DNN policy to be executed repeatedly. The field encompasses several methodological approaches: reduction to simpler-to-verify forms through neurosymbolic program synthesis (Bastani et al., 2018; Verma et al., 2018), construction of verified barrier functions that prevent specification violations during runtime (Anderson et al., 2020; Zhu et al., 2019), reachability analysis methods that determine system safety by computing reachable state sets (Ivanov et al., 2019; Tran et al., 2019), and model checking techniques that traverse state-transition graphs to verify properties (Amir et al., 2021).

Acknowledgements. This research was partially supported by NSF award SLES 2331783.

References

Derya Aksaray, Austin Jones, Zhaodan Kong, Mac Schwager, and Calin Belta. Q-learning for robust satisfaction of signal temporal logic specifications. In *Conference on Decision and Control (CDC)*, pages 6565–6570. IEEE, 2016.

Rajeev Alur, Suguman Bansal, Osbert Bastani, and Kishor Jothimurugan. A Framework for Transforming Specifications in Reinforcement Learning. *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, 2021.

Rajeev Alur, Osbert Bastani, Kishor Jothimurugan, Mateo Perez, Fabio Somenzi, and Ashutosh Trivedi. Policy synthesis and reinforcement learning for discounted ltl. In *Computer Aided Verification*, pages 415–435, Cham, 2023. Springer Nature Switzerland.

Guy Amir, Michael Schapira, and Guy Katz. Towards scalable verification of deep reinforcement learning. In *2021 formal methods in computer aided design (FMCAD)*, pages 193–203. IEEE, 2021.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Greg Anderson, Abhinav Verma, Isil Dillig, and Swarat Chaudhuri. Neurosymbolic reinforcement learning with formally verified exploration. *Advances in neural information processing systems*, 33:6172–6183, 2020.

Pranav Ashok, Jan Křetínskỳ, and Maximilian Weininger. Pac statistical model checking for markov decision processes and stochastic games. In *Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31*, pages 497–519. Springer, 2019.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31, 2018.

Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.

Alper Kamil Bozkurt, Yu Wang, Michael M Zavlanos, and Miroslav Pajic. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10349–10355. IEEE, 2020.

Ronen Brafman, Giuseppe De Giacomo, and Fabio Patrizi. Ltlf/ldlf non-markovian rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Przemysław Daca, Thomas A Henzinger, Jan Křetínskỳ, and Tatjana Petrov. Faster statistical model checking for unbounded temporal properties. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 112–129. Springer, 2016.

Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 128–136, 2019.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, pages 1126–1135, 2017.

Jie Fu and Ufuk Topcu. Probably approximately correct MDP learning and control with temporal logic constraints. In *Robotics: Science and Systems*, 2014.

Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Good-for-mdps automata for probabilistic analysis and reinforcement learning. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 306–323. Springer, 2020a.

Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Model-free reinforcement learning for stochastic parity games. In *31st International Conference on Concurrency Theory (CONCUR 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020b.

M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. Lee. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In *Conference on Decision and Control (CDC)*, pages 5338–5343, 2019.

Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pages 2107–2116. PMLR, 2018.

Jeevana Priya Inala, Osbert Bastani, Zenna Tavares, and Armando Solar-Lezama. Synthesizing programmatic policies that inductively generalize. In *International Conference on Learning Representations*, 2020.

Radoslav Ivanov, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 169–178, 2019.

Mathias Jackermeier and Alessandro Abate. Deepltl: Learning to efficiently satisfy complex ltl specifications for multi-task rl. In *The Thirteenth International Conference on Learning Representations*.

Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A composable specification language for reinforcement learning tasks. In *Advances in Neural Information Processing Systems*, volume 32, pages 13041–13051, 2019.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional reinforcement learning from logical specifications. In *Advances in Neural Information Processing Systems*, 2021.

Sham Machandranath Kakade. *On the sample complexity of reinforcement learning*. University of London, University College London (United Kingdom), 2003.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2):209–232, 2002.

Matthew Landers and Afsaneh Doryab. Deep reinforcement learning verification: a survey. *ACM Computing Surveys*, 55(14s):1–31, 2023.

Xiao Li, Cristian-Ioan Vasile, and Calin Belta. Reinforcement learning with temporal logic rewards. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3834–3839. IEEE, 2017.

Michael L Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-independent task specifications via GLTL. *arXiv preprint arXiv:1704.04341*, 2017.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. *arXiv preprint arXiv:2201.03544*, 2022.

Mateo Perez, Fabio Somenzi, and Ashutosh Trivedi. A pac learning algorithm for ltl and omega-regular objectives in mdps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 21510–21517, 2024.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.

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

Wenjie Qiu, Wensen Mao, and He Zhu. Instructing goal-conditioned reinforcement learning agents with temporal logic objectives. *Advances in Neural Information Processing Systems*, 36:39147–39175, 2023.

Vignesh Subramanian, Rohit Kushwah, Subhajit Roy, and Suguman Bansal. Inductive generalization in reinforcement learning from specifications. *arXiv preprint arXiv:2406.03651*, 2024.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Jakub Svoboda, Suguman Bansal, and Krishnendu Chatterjee. Reinforcement learning from reachability specifications: Pac guarantees with expected conditional distance. In *Forty-first International Conference on Machine Learning*, 2024.

Hoang-Dung Tran, Feiyang Cai, Manzanas Lopez Diego, Patrick Musau, Taylor T Johnson, and Xenofon Koutsoukos. Safety verification of cyber-physical systems with reinforcement learning control. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22, 2019.

Pashootan Vaezipoor, Andrew C Li, Rodrigo A Toro Icarte, and Sheila A Mcilraith. Ltl2action: Generalizing ltl instructions for multi-task rl. In *International Conference on Machine Learning*, pages 10497–10508. PMLR, 2021.

Leslie Valiant. *Probably approximately correct: nature's algorithms for learning and prospering in a complex world*. Basic Books, 2013.

Marcell Vazquez-Chanlatte, Susmit Jha, Ashish Tiwari, Mark K Ho, and Sanjit Seshia. Learning task specifications from demonstrations. *Advances in neural information processing systems*, 31, 2018.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International conference on machine learning*, pages 5045–5054. PMLR, 2018.

Cameron Voloshin, Abhinav Verma, and Yisong Yue. Eventual discounting temporal logic counterfactual experience replay. In *International Conference on Machine Learning*, pages 35137–35150. PMLR, 2023.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Zhe Xu and Ufuk Topcu. Transfer of temporal logic formulas in reinforcement learning. In *International Joint Conference on Artificial Intelligence*, pages 4010–4018, 7 2019.

Cambridge Yang, Michael Littman, and Michael Carbin. Reinforcement learning for general ltl objectives is intractable. *arXiv preprint arXiv:2111.12679*, 2021.

Lim Zun Yuan, Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Modular deep reinforcement learning with temporal logic specifications. *arXiv preprint arXiv:1909.11591*, 2019.

He Zhu, Zikang Xiong, Stephen Magill, and Suresh Jagannathan. An inductive synthesis framework for verifiable reinforcement learning. In *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, pages 686–701, 2019.

Luisa Zintgraf, Sebastian Schulze, Cong Lu, Leo Feng, Maximilian Igl, Kyriacos Shiarlis, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: Variational bayes-adaptive deep rl via meta-learning. *Journal of Machine Learning Research*, 22(289):1–39, 2021.

Appendix A. Computation of Expected Discounted-Sum

Consider the policy π_2 that chooses action a_2 in s_0 from Fig 1. The discounted-sum reward of $\gamma^k/(1-\gamma)$ if s_1 is reached in exactly k steps which occurs with probability $(1-\varepsilon)^{k-2}\varepsilon$. Thus, the expected discounted-sum reward is $\varepsilon\gamma^2/(1-\gamma) \cdot \sum_{k=0}^{\infty}(1-\varepsilon)^k\gamma^k = \varepsilon\gamma^2/[(1-\gamma)(1+\gamma\varepsilon-\gamma)]$.

Appendix B. SPECTRL Semantics

We consider the specification language SPECTRL for specifying reinforcement learning tasks Jothimurugan et al. (2019). A specification ϕ in this language is a logical formula over trajectories that

indicates whether a given trajectory ζ successfully accomplishes the desired task. As described below, it can be interpreted as a function $\phi : \mathcal{Z} \to \mathbb{B}$, where $\mathbb{B} = \{\texttt{true}, \texttt{false}\}$.

Formally, a specification is defined over a set of *atomic predicates* $\mathcal{P}_0$, where every $p \in \mathcal{P}_0$ is associated with a function $[\![p]\!] : S \to \mathbb{B}$; we say a state s *satisfies* p (denoted $s \models p$) if and only if $[\![p]\!](s) = \texttt{true}$. For example, given a state $s \in S$, the atomic predicate $[\![\texttt{reach } s]\!](s') = \big(\|s' - s\| < 1\big)$ indicates whether the system is in a state close to s with respect to the norm $\|\cdot\|$. The set of *predicates* $\mathcal{P}$ consists of conjunctions and disjunctions of atomic predicates. The syntax of a predicate $b \in \mathcal{P}$ is given by the grammar $b ::= p \mid (b_1 \wedge b_2) \mid (b_1 \vee b_2)$, where $p \in \mathcal{P}_0$. Similar to atomic predicates, each predicate $b \in \mathcal{P}$ corresponds to a function $[\![b]\!] : S \to \mathbb{B}$ defined naturally over Boolean logic. Finally, the syntax of SPECTRL specifications is given by [1]

$$\phi ::= \textsf{Eventually } b \mid \phi_1 \textsf{ Ensuring } b \mid \phi_1; \phi_2 \mid \phi_1 \texttt{ or } \phi_2,$$

where $b \in \mathcal{P}$. In this case, each specification ϕ corresponds to a function $[\![\phi]\!] : \mathcal{Z} \to \mathbb{B}$, and we say $\zeta \in \mathcal{Z}$ satisfies ϕ (denoted $\zeta \models \phi$) if and only if $[\![\phi]\!](\zeta) = \texttt{true}$. Letting ζ be a finite trajectory of length t, this function is defined by

$\zeta \models \textsf{Eventually } b$	if $\exists\, i \leq t,\ s_i \models b$
$\zeta \models \phi \textsf{ Ensuring } b$	if $\zeta \models \phi$ and $\forall\, i \leq t,\ s_i \models b$
$\zeta \models \phi_1; \phi_2$	if $\exists\, i < t,\ \zeta_{0:i} \models \phi_1$ and $\zeta_{i+1:t} \models \phi_2$
$\zeta \models \phi_1 \texttt{ or } \phi_2$	if $\zeta \models \phi_1$ or $\zeta \models \phi_2$.

Intuitively, the first clause means that the trajectory should eventually reach a state that satisfies the predicate b. The second clause says that the trajectory should satisfy specification ϕ while always staying in states that satisfy b. The third clause says that the trajectory should sequentially satisfy ϕ_1 followed by ϕ_2. The fourth clause means that the trajectory should satisfy either ϕ_1 or ϕ_2. An infinite trajectory ζ satisfies ϕ if there is a t such that the prefix $\zeta_{0:t}$ satisfies ϕ.

We assume that we are able to evaluate $[\![p]\!](s)$ for any atomic predicate p and any state s. This is a common assumption in the literature on learning from specifications, and is necessary to interpret a given specification ϕ.

1. Here, `achieve` and `ensuring` correspond to the "eventually" and "always" operators in temporal logic.