Inductive Generalization in Reinforcement Learning from Specifications

Rohit Kushwah[1], Vignesh Subramanian[2], Suguman Bansal[2], Subhajit Roy[1]
[1]Department of Computer Science and Engineering, IIT Kanpur, Kanpur, India
[2]School of Computer Science, Georgia Institute of Technology, Atlanta, GA
`krohitk@cse.iitk.ac.in, subhajit@iitk.ac.in`
`{vignesh, suguman}@gatech.edu`

Abstract

Reinforcement Learning (RL) from logical specifications is a promising approach to learning control policies for complex long-horizon tasks. While these algorithms showcase remarkable scalability and efficiency in learning, a persistent hurdle lies in their limited ability to generalize the policies they generate. In this work, we present an inductive framework to improve policy generalization from logical specifications. We observe that logical specifications can be used to define a class of inductive tasks known as *repeated tasks*. These are tasks with similar overarching goals but differing inductively in low-level predicates and distributions. Hence, policies for repeated tasks should also be inductive. To this end, we present an approach that learns policies for unseen repeated tasks by training on few repeated tasks only. Our approach is evaluated on long-horizon tasks in continuous state and action spaces, showing promising results in handling long-horizon tasks with improved one-shot generalization.

1 Introduction

The problem of *Reinforcement Learning* (RL) is to generate a policy for a given task in an unknown environment by continuously interacting with the environment Sutton & Barto (2018). When combined with neural-networks (NN), RL has made remarkable strides in control synthesis in real-world domains, including challenging continuous (innite-state) environments with non-linear dynamics or unknown models. Few examples include tasks such as walking Collins et al. (2005) and grasping Andrychowicz et al. (2020), control of multi-agent systems Lowe et al. (2017); Inala et al. (2021); Jothimurugan et al. (2022), and control from visual inputs Levine et al. (2016).

A key challenge facing RL is the difficulty in specifying the goal. Specifying rewards for complex, long-horizon tasks, can be highly non-intutive; Poor reward functions can make it hard for the RL algorithm to achieve the goal; e.g., it can result in reward hacking Amodei et al. (2016), where the agent learns to optimize rewards without achieving the goal.

An appealing alternative is to express the task in the form of a high-level logical specification, such as a *temporal logic* De Giacomo & Vardi (2013); Donzé (2013); Pnueli (1977), as opposed to a reward function. Logical specifications combine temporal operators with boolean connectives, enabling more natural encoding of a large class of desirable properties. Prior works have demonstrated the benefit of logical specifications scaling RL to long-horizon tasks Icarte et al. (2018); Jothimurugan et al. (2021); Li et al. (2017). Their theoretical implications have also been studied extensively Alur et al. (2022); Hahn et al. (2019); Yang et al. (2021). Furthermore, logical specifications facilitate testing and verification, which could be used to rigorously evaluate the correctness of the learned policy Ivanov et al. (2021). Details can be found here Alur et al. (2023).

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

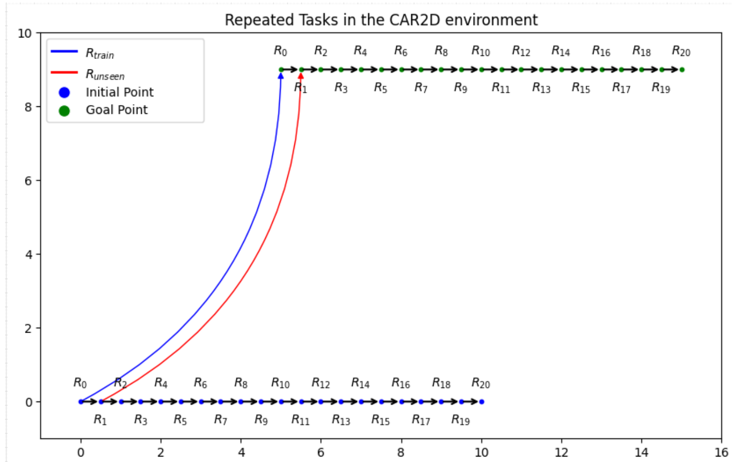


Figure 1: Repeated Task R_i navigates from initial location s_i to goal location t_i in 2D Cartesian Plane for $i \in [20]$. Initial and goal location shift to the right by 0.5 x-units each. Generalizable policy is trained on $\mathsf{R}_{\text{train}}$ and tested on $\mathsf{R}_{\text{unseen}}$.

This work-in-progress demonstrates an advantage of using logical specifications in learning *generalizable policies*. A critical shortcoming of RL is that the learnt policies do not generalize to environments or tasks other than the specific environment and task the policy was trained for. Our central insight is that logical specifications offer an inherent inductive structure that can be leveraged to learn generalizable policies for a large class of tasks.

Motivating Example Consider, a simulation of a car (point agent) in a 2D Cartesian plane (Figure 1). Our goal is to learn a policy to navigate the car from a given starting point s to a given target location g without encountering obstacles on its way. Many RL approaches, including learning from specification, can easily learn such a policy. The issue, however, is that the learned policy learns to navigate from initial position s_0 to goal position t_0 but will falter when trying to navigate from initial position $s_0 + \varepsilon$ to goal position $g_0 + \varepsilon$ for $\varepsilon \neq 0$. More generally, the policy will struggle to operate on initial and goal positions other than s_i and g_i, respectively.

We observe that these tasks share an identical overarching structure and differ only in the specifics of the location of the initial and goal positions. In Figure 1, these refer to the tasks to navigate from s_i to t_i where the $i+1$-th locations are shifted slightly to the right of the i-th location, for all $i \geq 0$. Hence, our goal should be to learn a *generalizable policy* for all $i \geq 0$. This means that despite variations in initial and goal positions during training, the policy should be adaptable to novel starting and goal points, maintaining a consistent underlying strategy.

To this end, we define *repeated tasks* to be a class of tasks that share the same overarching structure (as defined by a logical formula) and differ only in the specifics of the predicates of the formula such that the $i+1$-th task is inductively defined on the i-th task. Consequently, we define their policies inductively, i.e. the $i+1$-th policy is obtained from the i-th policy. Finally, we present an algorithmic approach that can learn these inductive policies, hence offering generalizability to a large class of unseen tasks.

2 Repeated Task

We define *repeated tasks* as a set of RL tasks that demonstrate the same overarching structure and only differ in the instantiation of their specification predicates and/or MDP initial distribution. We assume that repeated tasks build inductively, i.e. the $(i+1)$-th task builds on the first i-th task. Formally:

Definition 2.1. *A* repeated task *is given by the tuple* $\mathsf{R} = (\mathsf{R}_0, \mathsf{update\_pred}, \mathsf{update\_init})$ *where* R_0 *is the* base task*, the* predicate update function $\mathsf{update\_pred} : \mathcal{P} \mapsto \mathcal{P}$ *defines inductive updates to predicates, and the* initial distribution update function $\mathsf{update\_init} : \mathcal{D}(S) \mapsto \mathcal{D}(S)$ *defines inductive updates to the initial state distribution. The, the tasks in* R *are given by:*

$$\mathsf{R}_0 = (\phi(\mathcal{P}_0), \eta_0) \;\; \textit{and} \;\; \mathsf{R}_{i+1} = (\phi(\mathcal{P}_{i+1}), \eta_{i+1}) \; \textit{for } i > 0$$

where

$$\mathcal{P}_{i+1} = \{\mathsf{update\_pred}(p) \mid p \in \mathcal{P}_i\} \;\; \textit{and} \;\; \eta_{i+1}(s) = \eta_i(\mathsf{update\_init}(s))$$

where $\phi(\mathcal{P}_0)$ *is a* SPECTRL *specification defined over the predicates* $\mathcal{P}_0$ *and* η_0 *is the initial state distribution for the base task.*

In Figure 1, the base task is to navigate from the initial states of the environment to a location `goal = (`x_0`, `y_0`)` while avoiding a fixed obstacle positioned at the support of the predicate `obs`. In each repeated task the location `gtop` moves to the right by 0.5 units and the initial state distribution shifts to the right by 0.5 units.

We use the specification language SPECTRL to specify tasks ϕ (Appendix A.1). This repeated task is formalized as such: Let the base predicates be given by $\mathcal{P}_0 = \{\texttt{reach(goal)}, \texttt{avoid(obs)})\}$. Then the base specification is given by

$$\phi(\mathcal{P}_0) = \texttt{achieve}\;(\texttt{reach(goal)})\quad\texttt{ensuring}\quad(\texttt{avoid(obs)}).$$

The predicate update function is given by

$$\mathsf{update\_pred}(\texttt{reach}(s)) = \texttt{reach}(s + (0.5, 0)), \mathsf{update\_pred}(\texttt{avoid}(s)) = \texttt{avoid}(s)$$

and the initial distribution function is given by $\mathsf{update\_init}(\eta(s)) = \eta(s + (0.5, 0))$.

3 Generalization of Repeated Tasks

The problem of RL from logical specifications is to learn a policy that maximizes the probability of satisfaction of a given logical specification. Details have been deferred to the Appendix A.2.

We formulate the generalization problem to capture the intuition that if an RL agent learns to satisfy a few tasks from a repeated task, then the agent should be able to extrapolate to accomplish new tasks from the given repeated task. In addition, since the goal is to learn policies for tasks that are defined inductively, we assume that their policies are also defined inductively, i.e. the policy for $(i+1)$-th builds on the i-th task.

Formally, given repeated task $\mathsf{R} = (\mathsf{R}_0, \mathsf{update\_pred}, \mathsf{update\_init})$, letting $\pi_i : S \to A$ denote the policy satisfying the task R_i, we define the policy π_{i+1} for $i > 0$ as follows:

$$\pi_{i+1} = \kappa \circ \pi_i$$

where the *kappa policy* $\kappa = (\kappa_0, \ldots \kappa_{m-1})$ is an m-tuple consisting of *coefficient functions* $\kappa_k : S \to \mathbb{R}^{|A|}$ for $k \in [m-1]$. We call $m = |\kappa|$ the *degree* of the kappa policy. $\circ$ could be any template or function to generate a inductive update in the policy.

An advantage of describing policies inductively in a repeated task R is that one can generate policies for all tasks in R if the base policy π_0 and the kappa policy κ are known. To this end, we define the problem of learning generalizable policies for repeated tasks in terms of learning the base policy and the kappa policy from a set of training tasks from R. Our choice to learn the kappa policy in lieu of separate policies for all tasks in the training set renders the ability to extrapolate (generalize) to unseen tasks.

Problem Statement 3.1 (Generalizable RL for Repeated Tasks)**.** *Given a MDP with unknown transition probabilities, repeated task* R *and a set of training tasks* Train *such that the base task* $\mathsf{R}_0 \in \mathit{Train}$*, the* problem of learning generalizable policies for repeated tasks *is to learn a base policy* π_0 *and a kappa policy* κ^* *such that*

$$\pi_0 \in \arg\max_{\pi} \Pr_{\zeta \sim \mathcal{D}_\pi}[\zeta \models \phi_0, \eta_0]$$

and

$$\kappa^* \in \arg\max_{\kappa} \frac{1}{|\mathit{Train} \setminus \{\mathsf{R}_0\}|} \cdot \sum_{\mathsf{R}_j \in \mathit{Train} \setminus \{\mathsf{R}_0\}} \Pr_{\zeta \sim \mathcal{D}_{\pi_j}}[\zeta \models \phi_j, \eta_j]$$

where the policy π_j *is derived from* κ *and the base policy* π_0 *using the inductive policy template.*

In other words, (a). the base policy π_0 is the optimal policy for the base task R_0 and (b). the kappa policy simultaneously optimizes the policy π_j for all tasks $\mathsf{R}_j \in \mathsf{Train} \setminus \{\mathsf{R}_0\}$.

Generally, the *template* for inductive policies could be $\pi_{i+1} = f(\pi_i, \kappa)$. Our generalizable policy framework of learning the kappa policy as opposed to directly learning the policies for training tasks would extend to these more general templates.

4 Empirical Evaluation [1]

We summarize our empirical evaluation of our generalizable algorithm that learns a base policy and a kappa policy, given a template. Our experiments are designed to (a). evaluate the ability of the learned policy to (one-shot) generalize to unseen instances of the repeated task on long-horizon tasks, and (b). evaluate the impact of the template on generalizability.

We present a case study on a continuous 2D Cartesian Plane consisting of a car (RL agent) that is free to move in the plane. In this environment, both the state space and action space are continuous in nature. States $(x, y) \in \mathbb{R} \times \mathbb{R}$ refer to points in the cartesian plane and actions $(c, d) \in [0, 1] \times [0, 1]$ refer to a small displacement in the 2D plane. From state (x, y) on action (c, d) the environment progresses to $(x+c, y+d)$. We use pre-defined predicate `avoid`to define our tasks in the environment. Predicate `reach`is interpreted as reaching the coordinates of the specified point. Predicate `reach`holds true when point s is near the point $goal$ w.r.t euclidean norm $\|.\|_2$ i.e., $\texttt{reach}(goal)(s) = (\|s - goal\|_2 < \varepsilon_1)$ for a given error margin $\varepsilon > 0$.

4.1 Repeated Task

The overarching objective is to navigate from an initial location to n goal locations along a straight line, for a given $n \in \mathbb{N}$. With each induction on the task, the straight line shifts to the right by 0.5 units. I.e. the initial state distribution and all the goal locations shift to the right.

Formally, we denote the repeated task with n-goals by n-goals. Letting g_i denote the i-th goal on the base specification for n-goals (for $0 < i \le n$), the base specification is given by

$$\texttt{achieve}\ (\texttt{reach}(g_1)); \ldots ; \texttt{achieve}\ (\texttt{reach}(g_n)).$$

The predicate update function and the initial state distribution function are given by

$$\mathsf{update\_pred}(\texttt{reach}(g)) = \texttt{reach}(g + (0.5, 0)) \text{ and } \mathsf{update\_init}(\eta(s)) = \eta(s + (0.5, 0)).$$

Note that the motivating example from Figure 1 illustrates 1-goals.

4.2 Experimental Setup

We evaluate our algorithm across the five repeated tasks 1-goals, . . . , 5-goals corresponding to one through five reachability goals along a straight line, respectively. Each repeated task is trained on five training tasks, excluding the base task. In each task, we train to learn the inductive policy template. We evaluate our algorithm on (a) the amount of generalization and (b) the complexity of the template.

For each task, we compute the degree of generalization against the *success threshold* of $p \in \{0.5, \cdots, 0.9\}$ and the number of training episodes (iterations). The success threshold for repeated task n-goals for probability p indicates the number of unseen tasks for which the learned policy has a success probability greater than or equal to p. We also compute the degree of generalization as the number of training episodes (iterations) increases. For this, we set the success probability threshold to 0.85. Finally, we compute the generalization percentage as $|\mathsf{R}_{test,success}|/|\mathsf{R}_{train}|$ during the most successful number of iterations.

To analyze the complexity of the template, we perform experiments with two templates, one of degree one and the other of degree two. We also evaluate the degree of generalization of the state-of-the-art tool to learn from specifications DIRL.

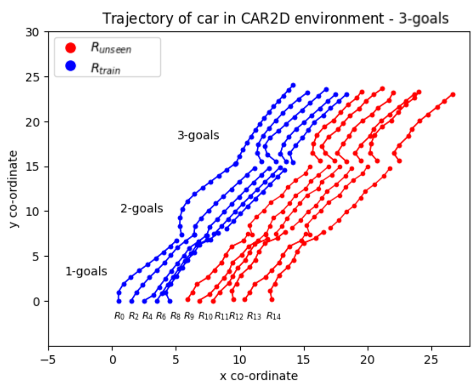
4.3 Observations

Generalizability. Table 1 presents our empirical on the degree of generalization. Our analysis reveals a notable trend of robust generalizability: our model reliably conforms to a wide range of unseen tasks, exhibiting an average generalization rate of 166.66% (the average of the final column scores from Table 1). Figure 2a shows the trajectories of the car in the 3-goals repeated task. It illustrates both the R_{train} trajectories (in blue) and the R_{unseen} trajectories (in red). This figure demonstrates that the agent successfully reaches all of its goals, even in the R_{unseen} tasks.

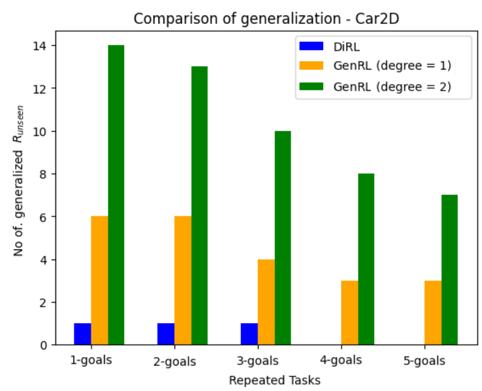
[1] We defer algorithm details to a longer version of this work.

Repeated Task	Success Threshold					Iteration (No. of episodes)					% Gen.
$\lvert \mathrm{R}_{train} \rvert = 5$	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**200**	**400**	**600**	**800**	**1000**	**(Best Iter.)**
1-goals	22	20	15	14	13	10	11	14	13	**14**	280
2-goals	20	19	14	13	13	8	10	11	11	**13**	260
3-goals	16	14	10	10	9	6	7	9	9	**10**	200
4-goals	14	12	8	8	7	6	6	7	**8**	**8**	160
5-goals	14	12	7	7	7	5	6	**7**	**7**	**7**	140

Table 1: Generalization Matrix. Success threshold p for a task indicates the number of unseen tasks that demonstrate $\geq p$ success probability. The number in **boldface** under Iterations represents the best generalization for the specification (with succes threshold as 0.85).



(a) Selected trajectories of the car on 3-goals (other test trajectories are not shown to minimize clutter)



(b) Generalization between DIRL and two templates of GenRL (with succes threshold as 0.85).

Figure 2

Significantly, the data from the table underscores that our model maintains an admirable generalization capability, even under the conditions of high reachability success thresholds. However, it's worth mentioning that there's a noticeable performance plateau upon reaching 600 iterations. We aim to delve into the cause of this stagnation to enhance the generalization outcomes beyond this point.

Template Analysis. From Figure 2b, it becomes evident that the baseline DIRL struggles with generalization. On the other hand, when using the template of degree 2, the generalization is commendable. In contrast, while using the template of degree 1 does show some capacity to generalize, its performance is weaker compared to its counterpart. This observation underscores a key insight: enhancing the complexity of our template can lead to notable improvements in generalization capabilities.

5 Concluding remarks

This work presents an inductive framework to learn generalizable policies for inductively defined repeated tasks. An empirical evaluation of a preliminary algorithm to learn these policies demonstrates the promise of our framework in learning generalizable policies for long-horizon tasks. In the future, we will evaluate our algorithm on more complex specifications in more challenging environments. We will also examine the impact of templates on generalization and seek to design the template based on the task and environment.

References

Rajeev Alur, Suguman Bansal, Osbert Bastani, and Kishor Jothimurugan. A framework for transforming specifications in reinforcement learning. In *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, pp. 604–624. Springer, 2022.

Rajeev Alur, Suguman Bansal, Osbert Bastani, and Kishor Jothimurugan. Specification-guided reinforcement learning. 2023.

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

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Steve Collins, Andy Ruina, Russ Tedrake, and Martijn Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307(5712):1082–1085, 2005.

Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pp. 854–860. Association for Computing Machinery, 2013.

Alexandre Donzé. On signal temporal logic. In *International Conference on Runtime Verification*, pp. 382–383. Springer, 2013.

Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Omega-regular objectives in model-free reinforcement learning. In *International conference on tools and algorithms for the construction and analysis of systems*, pp. 395–412. Springer, 2019.

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*, pp. 2107–2116. PMLR, 2018.

Jeevana Priya Inala, Yichen Yang, James Paulos, Yewen Pu, Osbert Bastani, Vijay Kumar, Martin Rinard, and Armando Solar-Lezama. Neurosymbolic transformers for multi-agent communication. *arXiv preprint arXiv:2101.03238*, 2021.

Radoslav Ivanov, Kishor Jothimurugan, Steve Hsu, Shaan Vaidya, Rajeev Alur, and Osbert Bastani. Compositional learning and verification of neural network controllers. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5s):1–26, 2021.

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

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems*, 34: 10026–10039, 2021.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Specification-guided learning of nash equilibria with high social welfare. In *International Conference on Computer Aided Verification*, pp. 343–363. Springer, 2022.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

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)*, pp. 3834–3839. IEEE, 2017.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

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

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

Cambridge Yang, Michael Littman, and Michael Carbin. On the (in) tractability of reinforcement learning for ltl objectives. *arXiv preprint arXiv:2111.12679*, 2021.

A Appendix

A.1 Specification Language

We consider the specification language SPECTRL for specifying RL 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. Formally, a SPECTRL specification is defined over a set of *atomic predicates* $\mathcal{P}_0$, where every $p \in \mathcal{P}_0$ is associated with a function $[\![p]\!] : \mathcal{S} \to \mathbb{B} = \{\texttt{true}, \texttt{false}\}$; we say a state s *satisfies* p (denoted $s \models p$) if and only if $[\![p]\!](s) = \texttt{true}$. Finally, the syntax of SPECTRL is given by [2]

$$\phi ::= \texttt{achieve}\ b \mid \phi_1\ \texttt{ensuring}\ b \mid \phi_1; \phi_2 \mid \phi_1\ \texttt{or}\ \phi_2,$$

where $b \in \mathcal{P}$. 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

$$\begin{array}{ll}
\zeta \models \texttt{achieve}\ b & \text{if } \exists\, i \le t,\ s_i \models b \\
\zeta \models \phi\ \texttt{ensuring}\ b & \text{if } \zeta \models \phi \text{ and } \forall\, i \le t,\ s_i \models b \\
\zeta \models \phi_1; \phi_2 & \text{if } \exists\, i < t,\ \zeta_{0:i} \models \phi_1 \text{ and } \zeta_{i+1:t} \models \phi_2 \\
\zeta \models \phi_1\ \texttt{or}\ \phi_2 & \text{if } \zeta \models \phi_1 \text{ or } \zeta \models \phi_2.
\end{array}$$

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.

A.2 Reinforcement Learning from Specifications

The problem of *RL from logical specifications* is to learn a policy in an *unknown environment* that maximizes the probability of satisfying a given logical specification describing a desired task.

Formally, the environment in RL is given by a *Markov decision process (MDP)* $\mathcal{M} = (S, A, P, \eta)$ with continuous states $S \subseteq \mathbb{R}^n$, continuous actions $A \subseteq \mathbb{R}^m$, transitions $P(s, a, s') = p(s' \mid s, a) \in \mathbb{R}_{\ge 0}$ (i.e., the probability density of transitioning from state s to state s' upon taking action a), and initial states $\eta : S \to \mathbb{R}_{\ge 0}$ (i.e., $\eta(s)$ is the probability density of the initial state being s). A *trajectory* $\zeta \in \mathcal{Z}$ is either an infinite sequence $\zeta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ or a finite sequence $\zeta = s_0 \xrightarrow{a_0} \cdots \xrightarrow{a_{t-1}} s_t$ where $s_i \in S$ and $a_i \in A$. A subtrajectory of ζ is a subsequence $\zeta_{\ell:k} = s_\ell \xrightarrow{a_\ell} \cdots \xrightarrow{a_{k-1}} s_k$. We let $\mathcal{Z}_f$ denote the set of finite trajectories. A (deterministic) *policy* $\pi : \mathcal{Z}_f \to A$ maps a finite trajectory to a fixed action. Given π, we can sample a trajectory by sampling an initial state $s_0 \sim \eta(\cdot)$, and then iteratively taking the action $a_i = \pi(\zeta_{0:i})$ and sampling a next state $s_{i+1} \sim p(\cdot \mid s_i, a_i)$.

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