Inductive Generalization in Reinforcement Learning from Specifications

Rohit Kushwah\textsuperscript{1}, Vignesh Subramanian\textsuperscript{2}, Suguman Bansal\textsuperscript{2}, Subhajit Roy\textsuperscript{1}
\textsuperscript{1}Department of Computer Science and Engineering, IIT Kanpur, Kanpur, India
\textsuperscript{2}School of Computer Science, Georgia Institute of Technology, Atlanta, GA
\texttt{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 \textit{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 \cite{Sutton2018}. When combined with neural-networks (NN), RL has made remarkable strides in control synthesis in real-world domains, including challenging continuous (infinite-state) environments with non-linear dynamics or unknown models. Few examples include tasks such as walking \cite{Collins2005} and grasping \cite{Andrychowicz2020}, control of multi-agent systems \cite{Lowe2017}, \cite{Inala2021}, \cite{Jothimurugan2022}, and control from visual inputs \cite{Levine2016}.

A key challenge facing RL is the difficulty in specifying the goal. Specifying rewards for complex, long-horizon tasks, can be highly non-intuitive; Poor reward functions can make it hard for the RL algorithm to achieve the goal; e.g., it can result in reward hacking \cite{Amodei2016}, 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 \textit{temporal logic} \cite{de2013temporal}, \cite{Donze2013}, \cite{Pnueli1977}, 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 \cite{Icarte2018}, \cite{Jothimurugan2021}, \cite{Li2017}. Their theoretical implications have also been studied extensively \cite{Alur2022}, \cite{Hahn2019}, \cite{Yang2021}. Furthermore, logical specifications facilitate testing and verification, which could be used to rigorously evaluate the correctness of the learned policy \cite{Tran2021}. Details can be found here \cite{Alur2023}.

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 $\mathbf{R} = (\mathbf{R}_0, \text{update}\_\text{pred}, \text{update}\_\text{init})$ where $\mathbf{R}_0$ is the base task, the predicate update function $\text{update}\_\text{pred} : \mathcal{P} \mapsto \mathcal{P}$ defines inductive updates to predicates, and the initial distribution update function $\text{update}\_\text{init} : \mathcal{D}(\mathcal{S}) \mapsto \mathcal{D}(\mathcal{S})$ defines inductive updates to the initial state distribution. The, the tasks in $\mathbf{R}$ are given by:

$$\mathbf{R}_0 = (\phi(P_0), \eta_0) \quad \text{and} \quad \mathbf{R}_{i+1} = (\phi(P_{i+1}), \eta_{i+1}) \quad \text{for} \quad i > 0$$
We formulate the generalization problem to capture the intuition that if an RL agent learns to satisfy

\[ \phi(P_0) \] is a SPECTRL specification defined over the predicates \( P_0 \) and \( \eta_0 \) 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 \( g_{\text{top}} \) 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 \( \phi \) (Appendix A.1). This repeated task is formalized as such: Let the base predicates be given by \( P_0 = \{ \text{reach}(\text{goal}), \text{avoid}(\text{obs}) \} \). Then the base specification is given by

\[ \phi(P_0) = \text{achieve}(\text{reach}(\text{goal})) \text{ ensuring } (\text{avoid}(\text{obs})). \]

The predicate update function is given by

\[ \text{update_pred}(\text{reach}(s)) = \text{reach}(s + (0.5, 0)), \text{update_pred}(\text{avoid}(s)) = \text{avoid}(s) \]

and the initial distribution function is given by \( \text{update_init}(\eta(s)) = \eta(s + (0.5, 0)) \).

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 \( R = (R_0, \text{update_pred}, \text{update_init}) \), letting \( \pi_i : S \rightarrow A \) denote the policy satisfying the task \( R_i \), we define the policy \( \pi_{i+1} \) 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 \( \kappa_k : S \rightarrow [0,1] \) 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 \( \pi_0 \) and the kappa policy \( \kappa \) 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 \( \text{Train} \) such that the base task \( R_0 \in \text{Train} \), the problem of learning generalizable policies for repeated tasks is to learn a base policy \( \pi_0 \) and a kappa policy \( \kappa^* \) such that

\[ \pi_0 \in \arg \max_{\pi} \Pr_{\zeta \sim D_\pi} [\zeta \models \phi_0, \eta_0] \]

and

\[ \kappa^* \in \arg \max_{\kappa} \frac{1}{|\text{Train} \setminus \{R_0\}|} \sum_{R_j \in \text{Train} \setminus \{R_0\}} \Pr_{\zeta \sim D_{\pi_j}} [\zeta \models \phi_j, \eta_j] \]

where the policy \( \pi_j \) is derived from \( \kappa \) and the base policy \( \pi_0 \) using the inductive policy template.

In other words, (a), the base policy \( \pi_0 \) is the optimal policy for the base task \( R_0 \) and (b) the kappa policy simultaneously optimizes the policy \( \pi_j \) for all tasks \( R_j \in \text{Train} \setminus \{R_0\} \).

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

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 \text{reach} holds true when point \(s\) is near the point \(goal\) w.r.t euclidean norm \(\|\cdot\|_2\) i.e., \(\text{reach}(goal)(s) = (\|s - goal\|_2 < \varepsilon_1)\) 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 \(0 < i \leq n\)), the base specification is given by

\[
\text{achieve (reach}(g_1)); \ldots; \text{achieve (reach}(g_n)).
\]

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

\[
\text{update}_{\text{pred}}(\text{reach}(g)) = \text{reach}(g + (0.5, 0)) \quad \text{and} \quad \text{update}_{\text{init}}(\eta(s)) = \eta(s + (0.5, 0)).
\]

Note that the motivating example from Figure [11] illustrates \(1\)-goals.

4.2 Experimental Setup

We evaluate our algorithm across the five repeated tasks \(1\)-goals, \(\ldots\), \(5\)-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 \(p \in \{0.5, \ldots, 0.9\}\) 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 \(|R_{\text{test,success}}|/|R_{\text{train}}|\) 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 DTRL.

4.3 Observations

Generalizability. Table [11] 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 [11]). Figure [23] shows the trajectories of the car in the 3-goals repeated task. It illustrates both the \(R_{\text{train}}\) trajectories (in blue) and the \(R_{\text{unseen}}\) trajectories (in red). This figure demonstrates that the agent successfully reaches all of its goals, even in the \(R_{\text{unseen}}\) tasks.

\footnote{We defer algorithm details to a longer version of this work.}
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 success threshold as 0.85).

<table>
<thead>
<tr>
<th>Repeated Task</th>
<th>Success Threshold</th>
<th>Iteration (No. of episodes)</th>
<th>% Gen.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.5</td>
<td>0.6</td>
<td>0.7</td>
</tr>
<tr>
<td>1-goals</td>
<td>22</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>2-goals</td>
<td>20</td>
<td>19</td>
<td>14</td>
</tr>
<tr>
<td>3-goals</td>
<td>16</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>4-goals</td>
<td>14</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>5-goals</td>
<td>14</td>
<td>12</td>
<td>7</td>
</tr>
</tbody>
</table>

*(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 success threshold as 0.85)*

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 2, 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


We consider the specification language $S$ where $b \phi$ from logical specifications. The problem of Reinforcement Learning from Specifications iteratively taking the action $a_\pi \eta$ is either an infinite sequence $\eta$ states $S$ continuous states $\xi$ is defined over a set of atomic predicates $P_0$, where every $p \in P_0$ is associated with a function $[p]: S \to \{\text{true}, \text{false}\}$; we say a state $s$ satisfies $p$ (denoted $s \models p$) if and only if $[p](s) = \text{true}$. Finally, the syntax of SPECTRL is given by

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

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

- $\zeta \models \text{achieve } b$ if $\exists i \leq t, s_i \models b$
- $\zeta \models \phi$ 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 $\zeta_{i+1:t} \models \phi_2$
- $\zeta \models \phi_1 \text{ or } \phi_2$ 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 $\phi$ while always staying in states that satisfy $b$. The third clause says that the trajectory should sequentially satisfy $\phi_1$ followed by $\phi_2$. The fourth clause means that the trajectory should satisfy either $\phi_1$ or $\phi_2$.

### 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) $M = (S, A, P, \eta)$ with continuous states $S \subseteq \mathbb{R}^n$, continuous actions $A \subseteq \mathbb{R}^m$, transitions $P(s, a, s') = p(s' | s, a) \in \mathbb{R}_{\geq 0}$ (i.e., the probability density of transitioning from state $s$ to state $s'$ upon taking action $a$), and initial states $\eta: S \to \mathbb{R}_{\geq 0}$ (i.e., $\eta(s)$ is the probability density of the initial state being $s$). A trajectory $\zeta \in Z$ 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 $\zeta$ is a subsequence $\zeta[l:k] = s_l \xrightarrow{a_l} \cdots \xrightarrow{a_{k-1}} s_k$. We let $Z_f$ denote the set of finite trajectories. A (deterministic) policy $\pi: Z_f \to A$ maps a finite trajectory to a fixed action. Given $\pi$, 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 $s_{i+1} \sim p(\cdot | s_i, a_i)$.

---

2 Here, achieve and ensuring correspond to the “eventually” and “always” operators in temporal logic.