AUTOSPEC: AUTOMATING THE REFINEMENT OF REINFORCEMENT LEARNING SPECIFICATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Logical specifications have been shown to help reinforcement learning algorithms in achieving complex tasks. However, when a task is under-specified, agents might fail to learn useful policies. In this work, we explore the possibility of improving coarse-grained logical specifications via an exploration-guided strategy. We propose AUTOSPEC, a framework that searches for a logical specification refinement whose satisfaction implies satisfaction of the original specification, but which provides additional guidance therefore making it easier for reinforcement learning algorithms to learn useful policies. AUTOSPEC is applicable to reinforcement learning tasks specified via the SpectRL specification logic. We exploit the compositional nature of specifications written in SpectRL, and design four refinement procedures that modify the abstract graph of the specification by either refining its existing edge specifications or by introducing new edge specifications. We prove that all four procedures maintain specification soundness, i.e. any trajectory satisfying the refined specification also satisfies the original. We then show how AUTOSPEC can be integrated with existing reinforcement learning algorithms for learning policies from logical specifications. Our experiments demonstrate that AUTOSPEC yields promising improvements in terms of the complexity of control tasks that can be solved, when refined logical specifications produced by AUTOSPEC are utilized.

1 INTRODUCTION

Reinforcement Learning (RL) algorithms have made tremendous strides in recent years Sutton & Barto (2018); Silver et al. (2016); Mnih et al. (2015); Levine et al. (2016). However, most algorithms assume access to a scalar reward function that must be carefully engineered to make environments amenable to RL—a practice known as reward engineering Ibrahim et al. (2024). This creates challenges in applying RL to new environments where useful reward functions are hard to construct. Furthermore, scalar Markovian rewards cannot provide sufficient feedback for certain tasks Abel et al. (2021); Bowling et al. (2023), leading to growing interest in non-Markovian reward functions Li et al. (2017a); Jothimurugan et al. (2021); Alur et al. (2023).

To make non-Markovian rewards tractable, it is standard to represent them via logical specification formulas that capture the intended task. These approaches, known as specification-guided reinforcement learning Aksaray et al. (2016); Li et al. (2017b); Icarte et al. (2018); Jothimurugan et al. (2019; 2021), derive reward functions from logical specifications. However, this creates two challenges: (i) providing specifications granular enough to guide RL algorithms, and (ii) defining accurate labeling functions mapping environment states to specification predicates. Users often create coarse specifications or labeling functions that, while logically correct, provide insufficient guidance for learning.

We present AUTOSPEC, a framework for automatically refining coarse specifications without user intervention. We say that a logical specification is *coarse* (or *under-specified*) if its predicate labelings or logical structure are too coarse to allow specification-guided RL algorithms to translate logical specifications into reward functions that allow for effective learning of RL policies. AUTOSPEC starts with an initial logical specification, translates it to a reward function, and attempts to learn a policy. If the learned policy's performance is unsatisfactory, AUTOSPEC identifies which specification components cause learning failures and automatically refines both the specification formula and

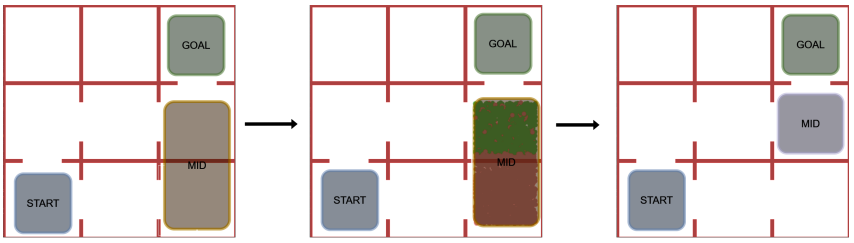


Figure 1: Example of refinement by AUTOSPEC in a 9-rooms environment. The original MID-node region includes a trap state from which recovery is impossible. The refined specification excludes this trap, enabling the agent to learn a policy with higher satisfaction probability.

labeling function. The refined specification's satisfaction implies the original's satisfaction while providing additional structure for learning. This process repeats until a satisfactory policy is learned.

AUTOSPEC works with SpectRL specifications; boolean and sequential combinations of reach-avoid tasks Jothimurugan et al. (2019). Any SpectRL specification decomposes into an abstract graph where edges specify reach-avoid tasks Jothimurugan et al. (2021). AUTOSPEC identifies problematic edges and applies targeted refinements: either modifying the labeling function for regions (Figure 1) or restructuring the graph to add alternative paths. These problematic edges are identified by employing an *exploration-guided strategy* that utilizes empirical trajectory data to identify edges in the abstract graph whose reach-avoid tasks (i.e. initial, target or unsafe regions) make it hard to learn a good RL policy. For instance, in Figure 1, the initial `MID` region of the `MID-GOAL` edge in the abstract graph is under-specified, as it overlaps with a trap state. The trap state is not immediately obvious as there is a path from `MID` to `GOAL`. By analyzing explored traces, AUTOSPEC identifies problematic start states in `MID` and refines the region to exclude the trap (as shown in Figure 1), thereby refining the logical reach-avoid specification associated to the `MID-GOAL` edge.

We prove that all refinements maintain soundness, where satisfaction of the refined specification implies satisfaction of the original. AUTOSPEC integrates with existing SpectRL-compatible algorithms as demonstrated with DIRL Jothimurugan et al. (2021) and LSTS Shukla et al. (2024).

Our contributions:

1. A framework for automated refinement of logical RL specifications with four refinement procedures, all with formal soundness guarantees (Section 3).
2. Integration with existing specification-guided RL algorithms, enabling them to solve tasks with coarse specifications (Section 3).
3. Empirical demonstration that AUTOSPEC enables learning from specifications that existing methods cannot handle (Section 4).

Related work. Recent years have seen substantial progress in solving RL tasks specified via logical specifications Aksaray et al. (2016); Li et al. (2017b); Icarte et al. (2018); Camacho et al. (2019); Giacomo et al. (2019); Hasanbeig et al. (2022; 2019); Hahn et al. (2019); Jothimurugan et al. (2019; 2021); Xu & Topcu (2019). Many works consider different fragments of Linear Temporal Logic (LTL) or their variants for specifying RL tasks. Icarte et al. (2018); Camacho et al. (2019) consider tasks that can be specified using deterministic finite automata (DFA) and solve them by *reward machines*, which decompose these tasks and translate them into a reward function. The reward function can then be used to train existing RL algorithms. Li et al. (2017b) considers a variant of LTL called TLTL for specifying tasks and propose a method for translating these specifications into continuous reward functions. Hasanbeig et al. (2022; 2019); Hahn et al. (2019) study the translation of tasks specified in LTL into reward functions. Alur et al. (2022) examines the theoretical questions related to the translation of logical specifications into reward functions.

Jothimurugan et al. (2019) defines the specification language SpectRL, a finitary fragment of LTL and provides justification for using this language to define specifications for RL tasks. A compositional method that decomposes SpectRL specifications into an abstract graph and constructs a reward function for each abstract graph edge was proposed in Jothimurugan et al. (2021). The approach by Toro Icarte et al. (2019) focuses on discovering optimal reward structures through environmental

exploration and reward analysis. Compositional methods are further explored by Neary et al. (2022), who propose removing unfulfillable subtasks, and Neary et al. (2023), who introduce verification techniques to certify learned policies. Zikelic et al. (2023b) propose CLAPS, a compositional method for learning neural network policies with formal guarantees on the satisfaction of SpectRL specifications, thus advancing the applicability to safety-critical RL applications by utilizing prior methods for learning reach-avoid policies with formal guarantees Lechner et al. (2022); Zikelic et al. (2023a); Chatterjee et al. (2023).

Recent advancements also include LTL2Action Vaezipoor et al. (2021), which translates LTL specifications into sequences of tasks for RL agents. Other recent approaches, such as Qiu et al. (2023) and DeepLTL Jackermeier & Abate (2025), leverage goal-conditioned RL and automata-based architectures to solve complex LTL and ω-regular tasks zero-shot or in multi-task settings. Trainify Jin et al. (2022) employs counterexample-guided abstraction and refinement (CEGAR) to iteratively improve policies by addressing failure cases identified through counterexamples. However, these works primarily focus on learning policies for fixed, well-defined specifications. In contrast, AUTOSPEC studies the problem of automated logical specification refinement towards improving reward functions obtained by translation from coarse logical RL specifications. Thus, our work is complementary to the works on RL from logical specifications and can be integrated into off-the-shelf specification-guided RL algorithms to improve the performance of learned agents.

2 PRELIMINARIES

MDPs. A Markov Decision Process (MDP) is a tuple $M = (S, A, P, R, \gamma)$, where $S \subseteq \mathbb{R}^n$ is the state space, $A \subseteq \mathbb{R}^m$ is the action space, $P : S \times A \times S \to [0, 1]$ is the probabilistic transition function, R is the (possibly non-Markovian) reward function, and γ is the discount factor. Let $\eta : S \to [0, 1]$ be the initial state distribution. A trajectory ζ in M is a sequence of states and actions $\zeta = s_0, a_0, s_1, a_1, \ldots$ where $s_i \in S$ and $a_i \in A$. We use $\mathcal{Z}$ to denote the set of all trajectories in M and $\mathcal{Z}_f$ to denote the set of all finite trajectories in M, which are finite prefixes of trajectories ending in states. A (pure) policy $\pi : \mathcal{Z}_f \to A$ assigns an action to each finite trajectory, and a non-Markovian reward $R : \mathcal{Z}_f \to \mathbb{R}$ assigns a reward to a finite trajectory. The MDP M under any policy π gives rise to a probability space over the set of all trajectories in the MDP Puterman (1994). We use $\mathbb{P}^\pi$ and $\mathbb{E}^\pi$ to denote the probability measure and the expectation operator in this probability space, respectively.

Logical specifications for Reinforcement Learning. In this work, we are solving RL tasks defined by logical specifications. Formally, a *logical specification* (or, simply, a *specification*) is a boolean function $\phi : \mathcal{Z} \to \{\text{true}, \text{false}\}$ which specifies whether a trajectory in the MDP satisfies the specification. We write $\zeta \models \phi$ whenever a trajectory ζ satisfies the specification ϕ. The objective of a specification-guided RL task is to find a policy π^* that maximizes the probability of satisfying the given specification ϕ, i.e. $\pi^* \in \textit{argmax}_\pi \mathbb{P}^\pi[\zeta \models \phi]$. Specification-guided RL algorithms use the specification to create a dense reward that guides the policy search, and therefore outperform algorithms that cannot leverage the specification for learning and instead require manual reward engineering Jothimurugan et al. (2021); Shukla et al. (2024).

SpectRL specification logic. We consider RL tasks specified in the SpectRL specification logic. SpectRL Jothimurugan et al. (2019) is a fragment of Linear Temporal Logic (LTL) which consists of all boolean and sequential combinations of reach-avoid tasks. Formally, a specification in SpectRL is defined in terms of *predicates* and *specification formulas*. An atomic predicate is a function $a : S \to \{\text{true}, \text{false}\}$ which defines a set of states that satisfy the atomic predicate. A predicate is a boolean combination of atomic predicates, i.e. $b := a \mid b_1 \wedge b_2 \mid b_1 \vee b_2$, where a is an atomic predicate and b_1 and b_2 are predicates. Specification formulas in SpectRL are defined by the grammar

$$\phi := \text{achieve } b \mid \phi_1 \text{ ensuring } b \mid \phi_1; \phi_2 \mid \phi_1 \text{ or } \phi_2 \tag{1}$$

where b is a predicate and ϕ_1 and ϕ_2 are specification formulas. Intuitively, "achieve b" requires the agent to reach a state in which the predicate b is satisfied. The clause "ϕ_1 ensuring b" requires the agent to satisfy the specification ϕ while only visiting states in which the predicate b is satisfied. The clause "$\phi_1; \phi_2$" requires the agent to first satisfy specification ϕ_1 and then satisfy specification ϕ_2. The clause "ϕ_1 or ϕ_2" requires satisfaction of at least one of ϕ_1 or ϕ_2. See Jothimurugan et al. (2019) for the formal definition of the semantics of each clause.

Abstract graphs for SpectRL specifications. It was shown in Jothimurugan et al. (2021) that each specification written in the SpectRL specification logic can be translated into an equivalent abstract graph. An *abstract graph* is a directed acyclic graph (DAG) whose vertices represent sets of MDP states and whose edges are annotated with sets of safe MDP states. Hence, each abstract graph edge defines a *reach-avoid specification*, where the task is to reach the set of states defined by the target vertex of the edge starting from the set of states defined by the source vertex of the edge, while staying within the set of safe states defined by the edge.

Definition 1 (Abstract graph)**.** *An* abstract graph $G = (V, E, \beta, s, t)$ *is a DAG, where* V *is a finite set of vertices,* E *is a finite set of edges,* $\beta : V \cup E \to \mathcal{B}(S)$ *is a labeling function that maps each vertex and each edge to a subset of the MDP states* S*,* $s \in V$ *is the source vertex and* $t \in V$ *is the target vertex. Furthermore, we require that* $\beta(s) = \textit{support}(\eta)$ *is the support of the initial state distribution* η *of the MDP.*

Given a trajectory ζ in the MDP and an abstract graph $G = (V, E, \beta, s, t)$, we say that ζ satisfies *abstract reachability* for G (written $\zeta \models G$) if it gives rise to a path in G that traverses G from s to t and satisfies the reach-avoid specifications of every traversed edge. It was shown in Jothimurugan et al. (2021) that, given any SpectRL specification ϕ, one can construct an abstract graph G such that $\zeta \models \phi$ if and only if $\zeta \models G$ holds for each trajectory ζ in the MDP. Hence, solving an RL task for a SpectRL specification reduces to solving an abstract reachability task in the abstract graph G.

Problem statement. Given an MDP M and a SpectRL specification ϕ, our goal is to learn a policy π such that the probability $\mathbb{P}^\pi[\zeta \models \phi]$ of a trajectory induced by the policy satisfying the specification is maximized.

Specification refinement. In order to solve this problem, we will utilize a common approach in specification-guided RL, to first translate the logical specification ϕ to a (non-sparse) reward function and then learn a policy by using existing RL algorithms with this reward function. However, if the probability of the specification being satisfied under the learned policy is unsatisfactory (i.e. below some desired probability threshold $p \in [0, 1]$), we will then refine the logical specification ϕ into a new SpectRL specification ϕ_r. We will then repeat the above process until the probability of the specification being satisfied under the learned policy becomes satisfactory.

Definition 2 (Specification refinement)**.** *Given two logical specifications* ϕ *and* ϕ_r*, we say that* ϕ_r refines ϕ*, if any MDP trajectory that satisfies the refined specification* ϕ_r *also satisfies the specification* ϕ*. That is, if for an MDP trajectory* ζ *we have* $(\zeta \models \phi_r) \implies (\zeta \models \phi)$*.*

3 AUTOMATED REFINEMENT OF RL SPECIFICATIONS

We now present AUTOSPEC, a framework for automated refinement of logical specifications in RL tasks. The key insight is that specification failures often stem from identifiable issues that can be systematically addressed: overly broad target regions, insufficient safety constraints, missing waypoints, or lack of alternative paths. When a specification-guided RL algorithm $\mathcal{A}$ fails to learn a satisfactory policy for specification ϕ, AUTOSPEC identifies which components caused the failure and applies targeted refinements to improve learnability while maintaining soundness.

AUTOSPEC operates as a wrapper around any SpectRL-compatible algorithm. It monitors the learning process, and when a policy π fails to satisfy the specification with probability at least $p \in [0, 1]$ (a user-provided threshold), it computes a refined specification ϕ_r such that: (1) satisfaction of ϕ_r implies satisfaction of ϕ (soundness), and (2) ϕ_r provides additional structure that makes it easier to learn. This refined specification is returned to algorithm $\mathcal{A}$ to continue learning. Through this iterative refinement process, AUTOSPEC enables solving RL tasks with coarse specifications that would otherwise be unlearnable.

Overview of AUTOSPEC. Algorithm 1 shows the complete AUTOSPEC framework. The algorithm takes as input an MDP M, a SpectRL specification ϕ, a satisfaction threshold p, and any specification-guided RL algorithm $\mathcal{A}$. It first translates ϕ into an abstract graph G and uses $\mathcal{A}$ to learn policies for the graph edges. For each edge e where $\mathcal{A}$ learned a policy but that policy fails to achieve satisfaction probability p, AUTOSPEC applies four refinement procedures in sequence: SeqRefine, AddRefine, PastRefine, and OrRefine. This ordering reflects increasing levels of structural modification; from

local predicate adjustments to graph topology changes. The first refinement that successfully improves performance above threshold p is applied, the graph is updated, and policies are relearned before proceeding to the next edge.

Algorithm 1 AUTOSPEC

Require: MDP M, specification ϕ, threshold $p \in [0, 1]$, spec-guided RL algorithm $\mathcal{A}$
  $G \leftarrow$ abstract graph corresponding to ϕ
  $\Pi \leftarrow \mathcal{A}(G)$ [set of policies for edges in G learned by algorithm $\mathcal{A}$]
  for $e = u \rightarrow u'$ an edge in G **do**
    $\pi_e \in \Pi \leftarrow$ policy learned for edge e (Null if $\mathcal{A}$ does not learn a policy for edge e)
    if π_e is not Null and $\mathbb{P}(\pi_e) < p$ **then**
      $\zeta \leftarrow$ sampled trajectories of the system
      if LEARNPOLICY(e,SEQREFINE(e,G,ζ))$> p$ **then**
        $G \leftarrow$ SEQREFINE(e,G,ζ)
      else if LEARNPOLICY(e,ADDREFINE(e,G,ζ))$> p$ **then**
        $G \leftarrow$ ADDREFINE(e,G,ζ)
      else if LEARNPOLICY(e,PASTREFINE(e,G,ζ))$> p$ **then**
        $G \leftarrow$ PASTREFINE(e,G,ζ)
      else if LEARNPOLICY(e,ORREFINE(e,G,ζ))$> p$ **then**
        $G \leftarrow$ ORREFINE(e,G,ζ)
      end if
      $\Pi \leftarrow \mathcal{A}(G)$ [set of policies for updated abstract graph G]
    end if
  end for
  Return G and Π

AUTOSPEC iterates through all edges $e = u \rightarrow u'$ of the abstract graph G for which the specification-guided RL algorithm $\mathcal{A}$ has learned a policy but for which the reach-avoid task satisfaction probability is below the provided probability threshold p. For each such edge, AUTOSPEC performs four refinement procedures that focus on different possible reasons for the edge $e = u \rightarrow u'$ being challenging for learning a satisfactory policy.

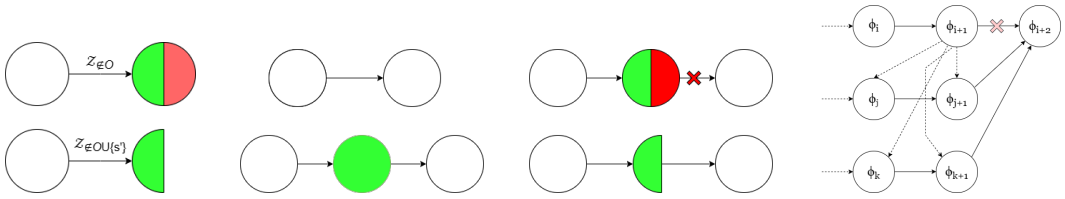
SeqRefine, which is invoked first, tries to locally refine the problematic edge $e = u \rightarrow u'$ by using predicate refinement techniques to refine the labeling function at the target region associated to the vertex u' and the safety region associated to the edge e. If SeqRefine fails to improve performance above threshold, AUTOSPEC invokes AddRefine which attempts to add a waypoint (i.e. a new abstract graph vertex) between the vertices u and u', making path-finding easier. If AddRefine also fails, AUTOSPEC invokes PastRefine which tries to refine the source node u. Finally, if other refinement procedures fail, AUTOSPEC invokes OrRefine which aims to find alternative paths to u'.

After each attempted refinement, an off-the-shelf RL algorithm (LEARNPOLICY in Algorithm 1) is used to estimate the satisfaction probability of the refined edge. When a refinement succeeds in achieving satisfaction probability above p, the refined abstract graph G is updated and AUTOSPEC applies the specification-guided RL algorithm $\mathcal{A}$ to learn a new set of edge policies Π for the entire graph. At the end, the final abstract graph G corresponds to the refined specification ϕ_r of the input specification ϕ.

3.1 SPECIFICATION REFINEMENT SUBPROCEDURES

We now define the four specification refinement subprocedures used by AUTOSPEC in Algorithm 1. Each procedure addresses a specific type of specification inadequacy, and they are applied in order of increasing structural modification to the abstract graph. The detailed pseudocodes are provided in the Appendix. Once a problematic edge $e = u \rightarrow u'$ is identified (an edge with satisfaction probability below threshold p), AUTOSPEC samples trajectories ζ using the learned policy, where the number of trajectories is an algorithm hyperparameter.

SeqRefine: Refining Predicates. The first refinement subprocedure addresses overly coarse predicates in the reach and avoid conditions. For edge $e = u \rightarrow u'$, SeqRefine refines both the target predicate $b = \beta(u')$ and the safety predicate $c = \beta(e)$ by calling two subprocedures:



(a) ReachRefine (b) AddRefine (c) PastRefine (d) OrRefine

Figure 2: Illustrations of abstract graph refinement processes. (a) ReachRefine demonstrating removal of failed part of goal region and addition of unsafe states set to existing avoid. (b) AddRefine demonstrating addition of waypoint between 2 nodes. (c) PastRefine removes part of the source node from where the agent failed (red) to get to the target, and keeps successful start states (green). (d) OrRefine shows how alternative paths (dotted lines) to target are constructed using existing specification nodes.

ReachRefine collects all states along sampled trajectories that successfully reached the goal region b. The refined goal region is computed as $b_r = b \cap \text{ConvexHull}(\text{reached states})$, effectively excluding unreachable portions of the original target region.

AvoidRefine collects states where trajectories entered unsafe regions (complement of c). The refined safe region is computed as $c_r = c \setminus \text{ConvexHull}(\text{last } k \text{ unsafe states})$, where k is a hyperparameter controlling how much of the trajectory tail to consider. This removes demonstrated unsafe areas from the safety region. We cannot relax or reduce the avoid regions during refinement because it would break soundness. In the relaxation, a policy can enter states that were part of the avoid in the original specification. Thus, the refined specification in that case is not a subset of the original specification.

SeqRefine returns a refined graph G_r identical to G except with updated labeling: $\beta_r(u') = b_r$ and $\beta_r(e) = c_r$. This refinement provides more precise guidance by excluding problematic regions discovered through exploration.

AddRefine: Introducing Waypoints. The second refinement addresses long or complex paths by decomposing them. When direct navigation from u to u' proves difficult, AddRefine introduces an intermediate vertex u'' by collecting midpoint states from successful trajectories that reached $\beta(u')$, defining $\beta(u'') = \beta(e) \cap \text{ConvexHull}(\text{midpoints})$, and replacing edge $e = u \to u'$ with two edges: $e'' = u \to u''$ and $e' = u'' \to u'$. This decomposition breaks a challenging long-horizon task into two shorter subtasks that are easier to learn.

PastRefine: Partitioning Source Regions. The third refinement addresses heterogeneous starting conditions where some initial states in u consistently lead to success while others lead to failure. PastRefine separates trajectories into successful and failing sets based on whether they satisfied edge e, then learns a hyperplane separating successful from failing initial states. It creates region b_r containing successful starting states and introduces new vertex u^* with $\beta(u^*) = b_r$ having the same incoming edges as u. The refinement replaces problematic edge $e = u \to u'$ with $e^* = u^* \to u'$. As shown in Figure 2(a), this refinement identifies and isolates promising initial conditions while preserving the original vertex u and its connections.

OrRefine: Exploiting Alternative Paths. The fourth refinement addresses blocked or infeasible direct paths by leveraging the existing graph structure. When the path through edge $e = u \to u'$ cannot be made satisfactory, OrRefine identifies alternative parents of u' (vertices u_i with existing edges $u_i \to u'$), and for each viable u_i, adds new edge $e_{ui} = u \to u_i$ with $\beta(e_{ui}) = \beta(e)$. It then tests if the alternative path $u \to u_i \to u'$ achieves the threshold. As illustrated in Figure 2(b), this creates alternative routes to the target using only existing vertices, maintaining all original safety constraints. OrRefine can iteratively explore ancestors of u_i if the direct connection fails.

As shown in Algorithm 1, any specification-guided RL algorithm that is applicable to SpectRL specifications and that learns policies for edges in the abstract graph can be integrated into the AUTOSPEC framework. The specification-guided RL algorithm learns policies for edges in the abstract graph, until it is unable to proceed beyond an edge with a sufficient satisfaction probability. We then perform the refinements in AUTOSPEC, using sampling to estimate the satisfaction probability of each refinement until one is found to exceed the threshold. This refinement is used to create an updated abstract graph and an updated set of edge policies are learned with respect to this graph.

3.2 CORRECTNESS OF AUTOSPEC

The following theorem establishes correctness of AUTOSPEC, showing that the specification ϕ_r computed by AUTOSPEC is indeed a refinement of the input specification ϕ. The proof, provided in Appendix A.2, proceeds by proving that each of the four refinement procedures results in a specification refinement.

Theorem 1 (Correctness of AUTOSPEC)**.** *Given an abstract graph G of a SpectRL specification ϕ and an edge e,* AUTOSPEC *computes a specification ϕ_r and returns an abstract graph G_r and an edge e_r such that ϕ_r refines ϕ. That is, for any MDP trajectory ζ, we have $(\zeta \models \phi_r) \implies (\zeta \models \phi)$.*

Incompleteness of the Specification Refinement Problem. AUTOSPEC provides *soundness* guarantees – as shown in Theorem 1, the produced specification is *guaranteed* to be a refinement of the original specification as in Definition 2, and every trajectory that satisfies the refined satisfaction must also satisfy the original specification. However, AUTOSPEC does not provide completeness guarantees. This is not a limitation of AUTOSPEC, but an inherent property of the specification refinement problem itself because the problem is *undecidable*. This is because, even for the simplest case of reachability specifications (e.g., "reach region G with probability at least p"), deciding whether a given policy satisfies a specification is *undecidable* for general continuous-state MDPs or probabilistic programs capable of encoding Turing-complete behavior Kaminski & Katoen (2015). Consequently, no specification refinement algorithm can be both sound and complete. AUTOSPEC therefore focuses on soundness, which is important towards ensuring that a policy for the refined task also solves the task defined by the original specification.

4 EXPERIMENTAL EVALUATION

We evaluate AUTOSPEC on its ability to diagnose and repair specification failures that prevent existing algorithms from learning satisfactory policies. Our experiments address three questions: (1) Can AUTOSPEC correctly identify which refinement type is needed for different failure modes? (2) Do the refinements enable learning from previously unlearnable specifications? (3) What are the requirements and limitations of the refinement process?

4.1 EXPERIMENTAL SETUP

We integrate AUTOSPEC with two specification-guided RL algorithms: DIRL Jothimurugan et al. (2021), which uses Dijkstra-style graph search with systematic exploration, and LSTS Shukla et al. (2024), which uses multi-armed bandits for edge selection with epsilon-greedy exploration. These algorithms differ fundamentally in their exploration strategies, allowing us to examine how AUTOSPEC's effectiveness depends on the underlying learning algorithm.

We evaluate on two domains specifically chosen to stress-test different aspects of specification refinement:

n-Rooms: Grid-based navigation with walls and doors, providing controlled tests of specific failure modes. State space: $(x, y, \theta, d) \in \mathbb{R}^4$ (position, angle to goal, distance). Action space: $(v, \theta) \in \mathbb{R}^2$ (velocity, direction). The n-rooms domain has been extensively used in specification-guided RL research Jothimurugan et al. (2021; 2019); Zikelic et al. (2023b) as it provides clear geometric structure while still presenting challenging long-horizon tasks. Its modular room structure naturally creates the types of specification failures we aim to address: trap states at room boundaries, dangerous narrow passages between rooms, and multiple alternative paths through different door configurations.

PandaGym Gallouédec et al. (2021): Robotic manipulation requiring 3D navigation around obstacles. This domain tests refinement in high-dimensional continuous control where geometric intuitions may not apply directly. Following recent work showing the challenges of specification-guided RL in manipulation tasks Shukla et al. (2024), we use this domain to validate that our convex hull and hyperplane-based refinements remain effective in high-dimensional spaces where human intuition about specification failures is limited.

For learning edge policies, both algorithms use PPO Schulman et al. (2017) with stable-baselines3 Raffin et al. (2021) implementation, following the standard practice in recent specification-guided RL work Jothimurugan et al. (2021); Zikelic et al. (2023b). We use 2-layer networks (64 neurons each),

learning rate 0.0003, and standard PPO hyperparameters. In all experiments we evaluate refinements using a deliberately high satisfaction threshold ($p = 0.99$). The purpose of this choice is methodological: by selecting a probability level that is difficult to achieve under coarse or under-specified predicates, we can clearly observe how the cumulative probability of satisfying the specification improves as AutoSpec performs successive refinements. Using such a stringent threshold ensures that even small improvements in guidance become visible in the satisfaction curves and allows us to measure the full extent of the benefit provided by refinement, independent of how poorly the initial specification performs.

All experiments are repeated over five random seeds. Plots report the mean across the five runs, with error bars showing the empirical mean $\pm$ variance. Specifically, each data point in the learning curves represents the performance of a policy trained for the distinct number of timesteps indicated on the x-axis (e.g., policies are trained for 80,000, 100,000, and 120,000 steps independently across 5 seeds). To estimate the specification satisfaction probability (y-axis), the trained policy for each edge is evaluated over 1000 rollout trajectories to empirically count successful versus failed attempts. The final success probabilities displayed in the plots are calculated using the product of success probability of the best path from start to goal.

4.2 Algorithm-Dependent Effectiveness: DiRL vs LSTS

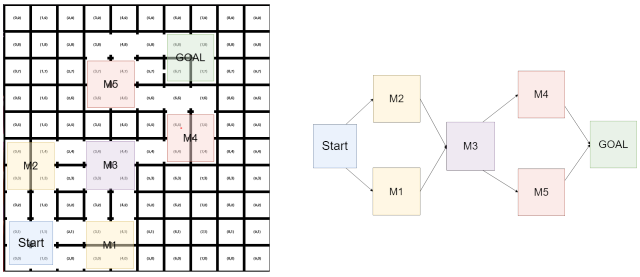
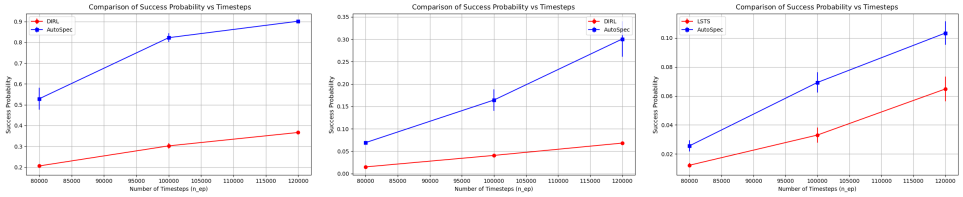


Figure 3: 100-rooms Environment with marked regions its DAG specification

Our experiments reveal that AutoSpec's effectiveness depends critically on the base algorithm's exploration strategy. We demonstrate this through a 100-rooms environment (Figure 3) with the complex specification: $\phi = \phi_{start}; (\phi_{m1} \text{ or } \phi_{m2}); \phi_{m3}; (\phi_{m4} \text{ or } \phi_{m5}); \phi_{goal}$

This specification structure, with multiple disjunctive branches and sequential compositions, represents the type of complex task decomposition that prior work Jothimurugan et al. (2019; 2021) has identified as necessary for real-world applications but challenging for existing algorithms. The 100-rooms scale specifically tests whether refinements remain effective when the state space is large enough that exhaustive exploration is infeasible, reflecting concerns raised in Shukla et al. (2024) about scalability of compositional methods.

With DiRL (Successful Refinement). As shown in Figure 4(a-b), DiRL's systematic exploration enables successful refinement. The algorithm explores edges in order of estimated difficulty, providing sufficient trajectory data for each edge before moving to the next. AutoSpec successfully applies



(a) Mid-goal DIRL performance (b) Full-spec DIRL performance (c) Mid-goal LSTS performance

Figure 4: Task satisfiability curves representing performances of DIRL and LSTS for sub-specifications and complete specification

ReachRefine on the ϕ_{m1} edge to remove unreachable portions of the target region, PastRefine on the ϕ_{m3} edge to identify successful starting regions, and OrRefine when direct paths fail to find alternative routes through ϕ_{m2}. The satisfaction probability improves from near 0% to approximately 60% through these refinements.

With LSTS (Refinement Failure). Figure 4(c) shows LSTS failing on the same specification. The bandit-based exploration spreads effort across all edges simultaneously, preventing deep exploration of any single edge. Consequently, edges to M4, M5, and Goal achieve 0% satisfaction, providing no successful trajectories for refinement computation. AUTOSPEC correctly reports its inability to refine without samples, demonstrating that refinement quality fundamentally depends on the base algorithm's exploration strategy.

Evaluation on Randomized 100-rooms and predicate placement To evaluate the generalization capabilities of our framework, we deployed AUTOSPEC in a procedurally generated 4-Way Grid-world where wall connectivity, and predicate placement were fully randomized for each seed (see Appendix A.3.1 for generation details). This setup specifically tests the system's ability to synthesize policies without reliance on hand-engineered specifications or environment-specific heuristics. As shown in Figure 6, AUTOSPEC significantly outperforms the DIRL baseline, achieving a terminal success probability of approximately 60% compared to the baseline's stagnation at 20%. These results confirm that AUTOSPEC autonomously identifies and resolves task bottlenecks, raising the success rate of critical transitions from $< 20\%$ to $> 90\%$ via automatic refinement. (Appendix A.3.5))

4.3 EVALUATION OF INDIVIDUAL REFINEMENTS

We design targeted experiments isolating specific failure modes to validate each refinement procedure.

SeqRefine: Trap State Elimination (Figure 7). Setup: 9-rooms environment where the goal region includes a blocked room creating a trap state. **Failure mode:** Agent reaches the trap portion of the goal and cannot escape. **Refinement:** ReachRefine identifies that successful trajectories only reach the accessible portion of the goal. The refined specification excludes the trap region: $b_r = b \cap \text{ConvexHull}(\text{reached states})$. **Result:** Satisfaction probability improves from 15% to 85%, demonstrating AUTOSPEC's ability to learn environmental constraints not captured in the original specification.

SeqRefine: Safety Constraint Discovery (Figure 8). Setup: 9-rooms with a narrow dangerous passage below the goal. **Failure mode:** Shortest path goes through narrow passage where agent frequently fails. **Refinement:** AvoidRefine identifies failure states near the narrow passage. The refined specification expands the avoid region: $c_r = c \setminus \text{ConvexHull}(\text{last 10 failure states})$. **Result:** Agent learns to use wider but longer safe path, improving satisfaction from 30% to 75%.

AddRefine: Waypoint Introduction (Figure 9). Setup: Long-horizon navigation across multiple rooms. **Failure mode:** Direct path too complex for single policy to learn reliably. **Refinement:** AddRefine identifies midpoints of successful trajectories and introduces intermediate vertex u''. **Result:** Decomposes task into two manageable subtasks, improving satisfaction from 20% to 90%.

PastRefine: Initial State Partitioning (Figure 10). Setup: Starting region includes states from which goal is unreachable. **Failure mode:** Policy cannot succeed from certain initial states. **Refinement:** PastRefine learns hyperplane separating successful from failing starts. **Result:** Focuses learning on viable initial states, improving satisfaction from 40% to 80%.

OrRefine: Alternative Path Discovery (Figure 11). Setup: Specification with multiple possible paths: $\phi_{MID1}; \phi_{GOAL}$ or $\phi_{MID2}; \phi_{GOAL}$. **Failure mode:** Direct path through MID1 blocked. **Refinement:** OrRefine adds edge $\phi_{MID1} \rightarrow \phi_{MID2}$, creating alternative route. **Result:** Enables satisfaction through alternate path when direct path has 0% success.

4.4 HIGH-DIMENSIONAL VALIDATION: PANDAGYM

To validate beyond grid environments, we test AUTOSPEC on PandaGym's continuous 3D manipulation task. The specification requires navigating around an invisible wall: $(reach$ red-region avoid wall$); (reach$ green-region avoid wall$)$. The invisible wall creates a chal-

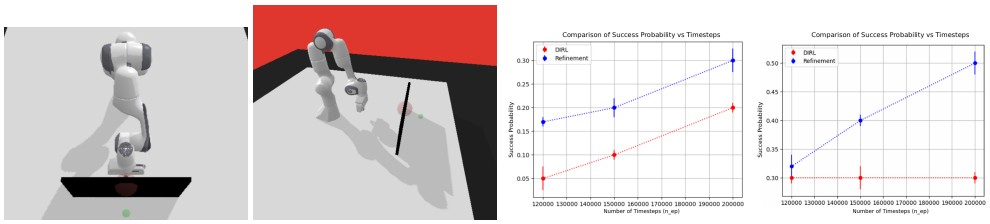


Figure 5: Evaluation of AUTOSPEC on PandaGym: (a) Two perspectives of the environment (1st and 2nd Figures), where the red region is an intermediate goal and an invisible wall blocks direct paths. (b) Performance of DiRL with and without AUTOSPEC: ReachRefine on first edge (3rd Figure) and PastRefine on second edge (4th Figure).

lenging scenario where the agent cannot directly observe the obstacle, making specification refinement crucial.

As shown in Figure 5, AUTOSPEC with DiRL successfully applies ReachRefine on the first edge to identify and exclude unreachable portions of the red region behind the wall, focusing the policy on achievable subgoals. On the second edge, PastRefine learns that only certain approach angles from the red region lead to successful reaching of the green region, effectively partitioning the intermediate state space based on trajectory outcomes. This demonstrates that AUTOSPEC's geometric refinements (convex hulls for ReachRefine, hyperplanes for PastRefine) remain effective in high-dimensional spaces where human intuition about the specification failures would be difficult. The success in this domain is particularly noteworthy because the refinements must capture 3D spatial relationships without explicit knowledge of the obstacle geometry.

Computational Overhead. AUTOSPEC avoids full retraining by only updating the policies associated with the identified subset of refined edges $\mathcal{R}$. The total computational cost is formalized as $T_{\text{total}} = T_{\text{base}} + \sum_{e \in \mathcal{R}} T_e$. Since $|\mathcal{R}|$ is typically small relative to the initial graph size, the aggregate overhead is bounded (empirically $T_{\text{total}} \leq 2T_{\text{base}}$). In the 100-room experiments, the baseline required ~ 240s to evaluate the 8 fixed edges, while AUTOSPEC averaged 390 ± 42s. This overhead corresponds directly to the training of 4–7 additional refinement edges per seed, with the observed variance ($\sigma \approx 42$s) driven by the differing topological complexity of the randomized environments. This computational investment is highly efficient, scaling linearly with the number of detected bottlenecks rather than the global state space size. Given the substantial improvement in success probability (from $\approx 20\%$ to $\approx 60\%$), this bounded overhead represents a favorable trade-off for achieving robust autonomy in stochastic domains.

5 CONCLUSION

We presented AUTOSPEC, a framework for automated refinement of coarse-grained logical specifications in reinforcement learning. AUTOSPEC addresses two common specification issues — coarse formulas and coarse labeling functions through four refinement procedures that maintain formal soundness. Our experiments on n-rooms and PandaGym environments demonstrate that AUTOSPEC can improve specification satisfiability when integrated with existing algorithms like DiRL and LSTS.

Our evaluation also reveals fundamental limitations: AUTOSPEC requires sufficient exploration data from the base algorithm to compute meaningful refinements. When algorithms fail to generate successful trajectories (as LSTS did on complex specifications), refinement becomes impossible. Despite these limitations, AUTOSPEC represents the first systematic approach to automatically refining logical specifications based on learning failures. Future work should address reducing exploration requirements for refinement and extending beyond SpectRL to more expressive temporal logics, such as infinite-horizon ω-regular specifications. While AUTOSPEC currently relies on finite witnesses, it could be adapted to these settings by decomposing tasks into a finite prefix (amenable to our current DAG-based refinement) and a cyclic suffix (which would require extending our witness analysis to handle infinite behaviors). The design of good specifications remains challenging in practice, and automated refinement is an important step toward making specification-guided RL more practical.

REFERENCES

David Abel, Will Dabney, Anna Harutyunyan, Mark K. Ho, Michael L. Littman, Doina Precup, and Satinder Singh. On the expressivity of Markov reward. In *Advances in Neural Information Processing Systems*, 2021.

Derya Aksaray, Austin Jones, Zhaodan Kong, Mac Schwager, and Calin Belta. Q-learning for robust satisfaction of signal temporal logic specifications. In *55th IEEE Conference on Decision and Control, CDC 2016, Las Vegas, NV, USA, December 12-14, 2016*, pp. 6565–6570. IEEE, 2016. doi: 10.1109/CDC.2016.7799279. URL https://doi.org/10.1109/CDC.2016.7799279.

Rajeev Alur, Suguman Bansal, Osbert Bastani, and Kishor Jothimurugan. A framework for transforming specifications in reinforcement learning. pp. 604–624, 2022.

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

Michael Bowling, John D. Martin, David Abel, and Will Dabney. Settling the reward hypothesis. In *Proceedings of the International Conference on Machine Learning*, 2023.

Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 6065–6073. ijcai.org, 2019. doi: 10.24963/IJCAI.2019/840. URL https://doi.org/10.24963/ijcai.2019/840.

Krishnendu Chatterjee, Thomas A. Henzinger, Mathias Lechner, and Dorde Zikelic. A learner-verifier framework for neural network controllers and certificates of stochastic systems. In Sriram Sankaranarayanan and Natasha Sharygina (eds.), *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I*, volume 13993 of *Lecture Notes in Computer Science*, pp. 3–25. Springer, 2023. doi: 10.1007/978-3-031-30823-9_1. URL https://doi.org/10.1007/978-3-031-30823-9_1.

Quentin Gallouédec, Nicolas Cazin, Emmanuel Dellandréa, and Liming Chen. panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning. *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*, 2021.

Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. In J. Benton, Nir Lipovetzky, Eva Onaindia, David E. Smith, and Siddharth Srivastava (eds.), *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019, Berkeley, CA, USA, July 11-15, 2019*, pp. 128–136. AAAI Press, 2019. URL https://ojs.aaai.org/index.php/ICAPS/article/view/3549.

Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Omega-regular objectives in model-free reinforcement learning. In Tomás Vojnar and Lijun Zhang (eds.), *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I*, volume 11427 of *Lecture Notes in Computer Science*, pp. 395–412. Springer, 2019. doi: 10.1007/978-3-030-17462-0_27. URL https://doi.org/10.1007/978-3-030-17462-0_27.

Mohammadhosein Hasanbeig, Yiannis Kantaros, Alessandro Abate, Daniel Kroening, George J. Pappas, and Insup Lee. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In *58th IEEE Conference on Decision and Control, CDC 2019, Nice, France, December 11-13, 2019*, pp. 5338–5343. IEEE, 2019. doi: 10.1109/CDC40024.2019.9028919. URL https://doi.org/10.1109/CDC40024.2019.9028919.

Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate. LCRL: certified policy synthesis via logically-constrained reinforcement learning. In Erika Ábrahám and Marco Paolieri (eds.), *Quantitative Evaluation of Systems - 19th International Conference, QEST 2022, Warsaw, Poland, September 12-16, 2022, Proceedings*, volume 13479 of *Lecture Notes in Computer Science*, pp. 217–231. Springer, 2022. doi: 10.1007/978-3-031-16336-4_11. URL https://doi.org/10.1007/978-3-031-16336-4_11.

Sinan Ibrahim, Mostafa Mostafa, Ali Jnadi, Hadi Salloum, and Pavel Osinenko. Comprehensive overview of reward engineering and shaping in advancing reinforcement learning applications, 2024. URL https://arxiv.org/abs/2408.10215.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2112–2121. PMLR, 2018. URL http://proceedings.mlr.press/v80/icarte18a.html.

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*, 2025. URL https://openreview.net/forum?id=9pW2J49flQ.

Peng Jin, Jiaxu Tian, Dapeng Zhi, Xuejun Wen, and Min Zhang. Trainify: A cegar-driven training and verification framework for safe deep reinforcement learning. In Sharon Shoham and Yakir Vizel (eds.), *Computer Aided Verification*, pp. 193–218, Cham, 2022. Springer International Publishing. ISBN 978-3-031-13185-1.

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. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=ion6Lo5tKtJ.

Benjamin Lucien Kaminski and Joost-Pieter Katoen. On the hardness of almost–sure termination. In *International Symposium on Mathematical Foundations of Computer Science*, pp. 307–318. Springer, 2015.

Mathias Lechner, Dorde Zikelic, Krishnendu Chatterjee, and Thomas A. Henzinger. Stability verification in stochastic control systems via neural network supermartingales. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pp. 7326–7336. AAAI Press, 2022. doi: 10.1609/AAAI.V36I7.20695. URL https://doi.org/10.1609/aaai.v36i7.20695.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17:39:1–39:40, 2016. URL http://jmlr.org/papers/v17/15-522.html.

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

Xiao Li, Cristian Ioan Vasile, and Calin Belta. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, pp. 3834–3839. IEEE, 2017b. doi: 10.1109/IROS.2017.8206234. URL https://doi.org/10.1109/IROS.2017.8206234.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015. doi: 10.1038/NATURE14236. URL https://doi.org/10.1038/nature14236.

Cyrus Neary, Christos Verginis, Murat Cubuktepe, and Ufuk Topcu. Verifiable and compositional reinforcement learning systems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, pp. 615–623, 2022.

Cyrus Neary, Aryaman Singh Samyal, Christos Verginis, Murat Cubuktepe, and Ufuk Topcu. Verifiable reinforcement learning systems via compositionality. *arXiv preprint arXiv:2309.06420*, 2023.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994. ISBN 978-0-47161977-2. doi: 10.1002/9780470316887. URL https://doi.org/10.1002/9780470316887.

Wenjie Qiu, Wensen Mao, and He Zhu. Instructing goal-conditioned reinforcement learning agents with temporal logic objectives. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=19AgWnmyoV.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Yash Shukla, Tanushree Burman, Abhishek Kulkarni, Robert Wright, Alvaro Velasquez, and Jivko Sinapov. Logical specifications-guided dynamic task sampling for reinforcement learning agents. In *34th International Conference on Automated Planning and Scheduling*, 2024. URL https://openreview.net/forum?id=okLobjqfjx.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016. doi: 10.1038/NATURE16961. URL https://doi.org/10.1038/nature16961.

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

Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. Learning reward machines for partially observable reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/532435c44bec236b471a47a88d63513d-Paper.pdf.

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*, pp. 10497–10508. PMLR, 2021.

Zhe Xu and Ufuk Topcu. Transfer of temporal logic formulas in reinforcement learning. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 4010–4018. ijcai.org, 2019. doi: 10.24963/IJCAI.2019/557. URL https://doi.org/10.24963/ijcai.2019/557.

Dorde Zikelic, Mathias Lechner, Thomas A. Henzinger, and Krishnendu Chatterjee. Learning control policies for stochastic systems with reach-avoid guarantees. In Brian Williams, Yiling Chen, and Jennifer Neville (eds.), *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pp. 11926–11935. AAAI Press, 2023a. doi: 10.1609/AAAI.V37I10.26407. URL https://doi.org/10.1609/aaai.v37i10.26407.

Dorde Zikelic, Mathias Lechner, Abhinav Verma, Krishnendu Chatterjee, and Thomas A. Henzinger. Compositional policy learning in stochastic control systems with formal guarantees. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023b. URL http://papers.nips.cc/paper_files/paper/2023/hash/95827e011b9e899f189a01fe2f4ef316-Abstract-Conference.html.

A Appendix / supplemental material

A.1 Refinement Algorithms

Here we present pseudo-code for the individual refinement algorithms described in Section 3.1

Algorithm 2 ReachRefine

Require: $b := \beta(u'), \zeta$
  $S_r \leftarrow \{s \mid s \in b \cap \zeta\}$ *Collect all the goal region states from the trajectories*
  $b_r \leftarrow b \cap \text{ConvexHull}(S_r)$ *Create the convex hull of the collected states*
  return b_r

Algorithm 3 AvoidRefine

Require: $c := \beta(e), \zeta$
  $O_r \leftarrow \{\}$ *Initialize new avoid region with an empty set*
  for ζ_i **in** ζ **do**
    if $\zeta_i[-1] \notin c$ **then**
      $O_r \leftarrow O_r \cup \{s_j \mid s_j \in \zeta_i \bigwedge (len(\zeta_i) - j) \leq k\}$
      Append the last k states from every trajectory that ended up in the avoid region
    end if
  end for
  $c_r \leftarrow c \backslash \text{ConvexHull}(O_r)$ *Create a convex hull around the collected states and remove it form the original safe region*
  return c_r

Algorithm 4 SeqRefine: Refining Edge $e = u \rightarrow u'$

Require: Edge $e = u \rightarrow u'$, Graph G, set of trajectories ζ.
  $b = \beta(u')$ *States in the reach predicate*
  $c = \beta(e)$ *States in the avoid predicate*
  $b_r \leftarrow$ *ReachRefine*(b, ζ)
  $c_r \leftarrow$ *AvoidRefine*(c, ζ)
  $u_r \leftarrow [\beta(u_r) = b_r]$ *Redefine target node with new predicate*
  $e_r \leftarrow [u \rightarrow u_r, \text{with } \beta(e_r) = c_r]$ *Redefine edge with new predicate*
  $G' \leftarrow G \setminus [e \leftarrow e_r]$ *Replace edge with refinement*
  return G'

Algorithm 5 AddRefine

Require: Edge $e = u \rightarrow u'$, Graph G, set of trajectories ζ.
  $S_r \leftarrow \{\}$
  for ζ_i **in** ζ **do**
    if $\zeta_i \models e$ [i.e. trajectory was successful] **then**
      $S_r \leftarrow S_r \cup \zeta_i[len(\zeta_i)//2]$ *Add center of trajectory as waypoint*
    end if
  end for
  $b_r \leftarrow$ *ConvexHull*$(S_r) \cap \beta(e)$
  $u'' \leftarrow [\beta(u'') = b_r]$ *Define target node for waypoint*
  $e'' \leftarrow [u \rightarrow u'', \text{with } \beta(e'') := \beta(e)]$
  $e' \leftarrow [u'' \rightarrow u', \text{with } \beta(e') := \beta(e)]$ *Define edges with new waypoint predicate*
  $G' \leftarrow G \setminus [e \leftarrow [e''; e']]$ *Replace edge e with composition of new edges*
  return G'

Algorithm 6 PastRefine: Refining Abstract Graph Exploration

Require: Edge $e = u \to u'$, Graph G, set of trajectories ζ.
  $S \leftarrow \{\}$ $S_r \leftarrow \{\}$
  for ζ_i **in** ζ **do**
    $S \leftarrow S \cup \zeta_i[0]$ *Collect start states from all trajectories*
    if $\zeta_i \models e$ **then**
      $S_r \leftarrow S_r \cup \zeta_i[0]$ *Collect start states from successful trajectories*
    end if
  end for
  Identify a hyperplane H separating the S_r and $S \setminus S_r$
  $b_r \leftarrow \{s \in S : H(s) \geq 0\}$
  $u^* \leftarrow [\beta(u^*) = b_r]$ *Redefine initial node with new predicate*
  $e_r \leftarrow [u^* \to u, \text{with } \beta(e_r) := \beta(e)]$ *Redefine edge with new predicate*
  $G' \leftarrow G \setminus [e \leftarrow e_r]$ *Replace edge with refinement*
  return G'

Algorithm 7 OrRefine: Disjunctive Specification Refinement

Require: Edge $e = u \to u'$, Graph G, set of trajectories ζ.
  $E = \{e_i \in G \mid e_i = u_i \to u'\}$ *Collect all 'parents' of* u'
  for $e_i \in E$ **do**
    $e_{ui} \leftarrow [u \to u_i, \text{with } \beta(e_r) := \beta(e)]$ *Define edges from source to parents*
    $G \leftarrow G \cup [e_{ui}]$ *Add new edge to graph*
  end for
  return G

A.2 Proof of Theorem 1

Theorem 1 (Correctness of AutoSpec) *Given an abstract graph G of a SpectRL specification ϕ and an edge e,* AutoSpec *computes a specification ϕ_r with abstract graph G_r such that ϕ_r refines ϕ. That is, for any MDP trajectory ζ, we have $(\zeta \models \phi_r) \implies (\zeta \models \phi)$.*

Proof. To prove the theorem, it suffices to show that for each of the four refinement subprocedures, if they return an abstract graph G_r, then the corresponding specification ϕ_r is a refinement of the input specification ϕ.

By the definition of abstract reachability, we have $(\zeta \models \phi) \Leftrightarrow (\zeta \models G)$ and $(\zeta \models \phi_r) \Leftrightarrow (\zeta \models G_r)$. Hence, to prove that $(\zeta \models \phi_r) \Rightarrow (\zeta \models \phi)$ which is the definition of specification refinement as in Definition 2, it suffices to prove that $(\zeta \models G_r) \Rightarrow (\zeta \models G)$. We prove this claim for each refinement subprocedure.

SeqRefine. Suppose that $G_r = \text{SeqRefine}(e, G, \zeta)$. Let $e = u \to u'$. By our design of SeqRefine, the abstract graph G_r has the same vertex set, edge set and labeling function as G, with the only difference being that $\beta_r(u') \subseteq \beta(u')$ due to ReachRefine and $\beta_r(e) \subseteq \beta(e)$ due to AvoidRefine. Hence, every trajectory ζ that satisfies all reach-avoid tasks in G_r must also satisfy those in G, giving us $(\zeta \models G_r) \Rightarrow (\zeta \models G)$.

AddRefine. Suppose that $G_r = \text{AddRefine}(e, G, \zeta)$. Let $e = u \to u'$. AddRefine introduces a new vertex u'' and replaces edge e with two sequentially composed edges $e'' = u \to u''$ and $e' = u'' \to u'$ where $\beta_r(e'') = \beta_r(e') = \beta(e)$. Any trajectory satisfying the refined path through u'' must visit the intermediate waypoint while respecting the original safety constraints, thus also satisfying the original edge specification. Therefore, $(\zeta \models G_r) \Rightarrow (\zeta \models G)$.

PastRefine. PastRefine refines the region associated to vertex u by restricting it to $\beta_r(u) \subseteq \beta(u)$. This refinement affects both edge $e = u \to u'$ and all edges incoming to u. Since the refined region is a subset of the original, any trajectory satisfying the refined specification must originate from states that were valid in the original specification. Hence, $(\zeta \models G_r) \Rightarrow (\zeta \models G)$.

OrRefine. Suppose that $G_r = \text{ORREFINE}(e, G, \zeta)$. OrRefine only adds edges between existing vertices in G. Specifically, for a problematic edge $e = u \to u'$, it identifies existing edges $e_i = u_i \to u'$ and adds new edges $e_{new} = u \to u_i$ where $\beta(e_{new}) = \beta(e)$.

Consider a trajectory ζ that satisfies G_r via a newly added path $u \to u_i \to u'$. Since: (1) Both u_i and u' existed in the original vertex set of G, (2) The edge $u_i \to u'$ existed in the original edge set of G, (3) The new edge $u \to u_i$ maintains the safety constraints of the original edge ($\beta(e_{new}) = \beta(e)$),

the trajectory ζ reaches u' through a combination of transitions that respect all original safety constraints and only uses vertices from the original specification. The path through u_i represents a valid alternative route in the original specification structure. Therefore, $(\zeta \models G_r) \Rightarrow (\zeta \models G)$.

Thus, all four refinement procedures preserve specification soundness. □

A.3 EXPERIMENTS

A.3.1 PROCEDURAL ENVIRONMENT GENERATION

To evaluate the robustness of the proposed refinement framework, we utilize a procedurally generated Continuous Gridworld environment (Figure 6. The environment layout and region locations are randomized for each experimental seed to ensure that the agent learns generalized navigation skills rather than memorizing a specific map layout. We use the same topological layout as in Figure 3. The generation process follows a three-step strategy:

A.3.2 GRID AND REGION DEFINITIONS

The domain is defined as a grid of 10×10 rooms, where each room has a spatial dimension of 8.0×8.0 units. We define a set of logical regions $\mathcal{R} = \{Start, Goal, M_1, M_2, MG, MG_1, MG_2\}$. Each region $r \in \mathcal{R}$ is physically instantiated as a 2×2 block of contiguous rooms.

A.3.3 RANDOMIZED PLACEMENT STRATEGY

The placement of regions is performed stochastically to maximize variability while maintaining a solvable structure:

1. **Start and Goal Initialization:** We define the four corners of the grid as candidate zones. To ensure traversal complexity, the $Start$ and $Goal$ regions are restricted to *opposite corners*. A pair of opposite corner zones (e.g., Top-Left and Bottom-Right) is selected uniformly at random. The $Start$ region is assigned to one zone and the $Goal$ to the other, denoted as:

$$(Pos_{start}, Pos_{goal}) \sim \text{Uniform}(\{(C_{TL}, C_{BR}), (C_{TR}, C_{BL})\}) \tag{2}$$

   where C represents the set of valid indices for a 2×2 block in a specific corner.

2. **Intermediate Region Placement:** The remaining intermediate regions (e.g., $M_1, M_2, \dots$) are placed randomly within the grid. Their locations are sampled uniformly from the set of all valid 2×2 block coordinates, subject to the constraint that no two regions may spatially overlap:

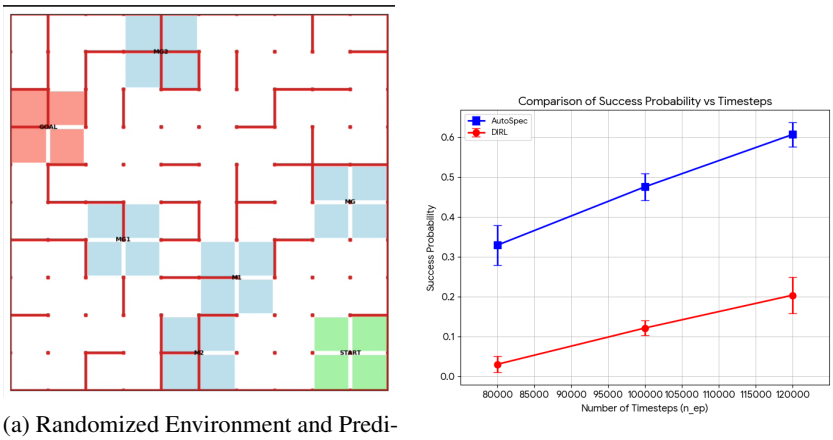
$$Pos_r \sim \text{Uniform}(\mathcal{G}_{valid} \setminus \bigcup_{r' \in \mathcal{R}_{placed}} Area(r')) \tag{3}$$

   This ensures that all sub-goals are distinct and distributed stochastically across the map.

A.3.4 STOCHASTIC CONNECTIVITY

Once regions are placed, the connectivity between adjacent rooms is generated probabilistically. For every pair of adjacent rooms (u, v) in the grid, a connecting door is instantiated via a Bernoulli trial:

$$P(\text{Door}_{u,v}) = 0.5 \tag{4}$$



(a) Randomized Environment and Predicates Instance

(b) Success Probability Comparison

Figure 6: **Experimental Results on the 100-room Gridworld.** (a) A visualization of one of the procedurally generated environment instance, showing the randomized placement of the Start (green), Goal (red), and intermediate refinement regions (blue). (b) Comparative performance analysis showing the mean success probability of the proposed AutoSpec method versus the DIRL baseline over 80k and 100k timesteps.

A.3.5 Results for Randomized Predicate Location

The experimental results on the randomized 100-room Gridworld demonstrate that AutoSpec operates effectively without the need for carefully crafted specifications or hand-engineered environments. As illustrated in Figure 6, the environment introduces significant complexity through randomized wall configurations and arbitrarily placed predicate regions. We evaluated performance across 5 different procedurally generated environments (one per seed) and report the aggregated results in Figure 6b (b). Under these stochastic conditions, the DIRL baseline fails to synthesize a robust policy, stagnating at a low success probability.

In contrast, AutoSpec demonstrates superior adaptability by automatically refining the specification graph to overcome structural anomalies. While the global end-to-end success rate plateaus at approximately 50%, this limitation is not a failure of the refinement process itself, but rather the result of specific topological challenges inherent to the randomized domain. Our analysis of the transition logs reveals two specific phenomena that dictate performance:

The "Bridge" Bottleneck. In 80% of the random seeds, the primary failure mode is the transition exiting the central convergence point ($MG \rightarrow MG_1$). While agents can reliably reach the MG region ($> 90\%$ success), the unrefined transition to the subsequent Mid_1 region frequently collapses to a success probability of 10–15% due to randomized wall placements creating narrow or non-linear passages. AutoSpec addresses this by applying *AddRefine* to introduce intermediate waypoints ($MG \rightarrow MG_{add} \rightarrow MG_1$) or *ReachRefine* to tighten target constraints. Empirical logs show these refinements consistently raise the local success rate of this specific bottleneck edge to 50–70%, thereby recovering end-to-end traversability where standard methods fail.

Autonomous Corridor Switching. A critical advantage of the refinement process is the ability to dynamically switch logical corridors. The global task allows for alternate paths ($Start \rightarrow \{M_1, M_2\} \rightarrow MG \rightarrow \{MG_1, MG_2\} \rightarrow Goal$). In instances where the path through MG_1 remains stochastically blocked (success $< 5\%$) even after refinement attempts, AutoSpec effectively prunes this branch. The search strategy then shifts probability mass to the alternative MG_2 branch. In our experiments, we observed seeds where the unrefined agent effectively "gave up" due to the difficulty of MG_1, whereas the refined agent successfully discovered and optimized the alternative $MG \rightarrow MG_2$ route ($> 78\%$ local success), proving that automated refinement acts as a form of robust structural exploration.

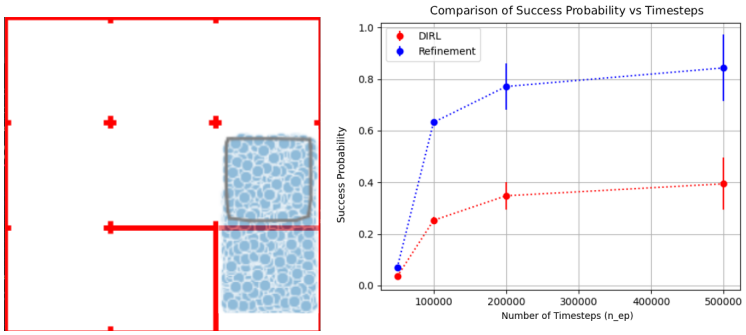


Figure 7: Evaluation of Reach Probabilities in the 9-Rooms Environment. (a) The layout of the 9-rooms environment, showing the walls, doors, and goal regions, and the estimated convex hull for the new reach region, showing how the refinement process effectively restructs the reachable states, leading to better satisfaction of the specification (b) A comparison of reach probabilities between DIRL Jothimurugan et al. (2021) and the proposed AutoSpec approach. The x-axis denotes the number of steps, and the y-axis denotes the estimated probability of success.

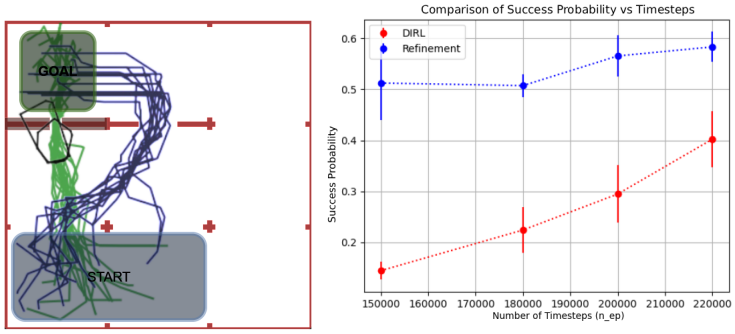


Figure 8: Results of Avoid refinement. (a) The layout of the 9-rooms environment, showing the walls, doors, goal regions and avoid regions (red) and learned trajectories before (green) and after (blue) refinement, with new estimated avoid regions (black) (b) A comparison of reach probabilities between DiRL and the proposed Avoid Refinement.

A.3.6 Results for Individual Refinements

We conducted multiple experiments to validate our approach to refining coarse-grained SpectRL specifications for solving RL tasks. The goal of our experiments is to compare the performance of the original DiRL Jothimurugan et al. (2021) algorithm and our integration of DiRL with AutoSpec, thus showcasing the ability of AutoSpec to refine SpectRL specifications that are challenging for the existing algorithms for RL from logical specifications. For learning edge policies, in both cases we use Proximal Policy Optimization (PPO) Schulman et al. (2017), implemented using stable-baselines3 Raffin et al. (2021). We employ a 2-layer neural network, each layer containing 64 neurons. We consider two environments.

9 Rooms. The 9 Rooms environment consists of walls blocking access to some rooms and doors allowing access to adjacent rooms. It has a 4-D continuous state space $(x, y, \theta, d) \in \mathbb{R}^4$, representing the 2D position, angle to the goal, and distance to the goal. We consider several SpectRL specifications which are translated into abstract graphs. The start position is sampled from the region associated to the source vertex, and the goal position is sampled from the region associated to the target vertex of the abstract graph. The 2-D continuous action space determines the velocity and direction of the agent $(v, \theta) \in \mathbb{R}^2$, with the new position calculated as $s' = s + (v\cos(\theta), v\sin(\theta))$.

PandaGym. The Pandagym Gallouédec et al. (2021) reach environment has a robotic arm with an object picked up and the task is to place the object at the correct location. A wall blocking the path to

the goal is invisible to the robot. The state space consists of the current position of the gripper arm in 3D and the goal position.

Experiment 1: Atomic predicate refinement. To illustrate how an incorrect specification is identified and corrected using Algorithm 2, we consider a 9 Rooms environment in Figure 7. In this environment, one room in the goal region is blocked, representing the incorrect specification. Figure 7 displays the learning curves for both the original and refined specifications, demonstrating the performance improvements achieved through the refinement process. Algorithm 3 can be empirically verified by creating a 9 Rooms environment as shown in Figure 8, where Figure 8 (a) shows the goal region along with avoid region (red). To improve probability of satisfaction the agent should avoid the narrow door below the goal and use the longer but safer route to approach the goal from the side. We see the learned trajectories before and after refinement in Figure 8 (a), where the new avoid region blocks the narrow door, effectively causing the agent to learn a policy that uses the wider door on the side. This helped improve specification satisfiability.

Experiment 2: Sequential refinement. We created a specification ϕ_{goal} to evaluate Algorithm 5. Figure 9 show that AddRefine is extremely sample efficient, and can construct a new specification to aid the current edge with an extremely high success probability. We also show the distribution of states that make up the new specification ϕ^r_{goal}. To verify Algorithm 6, we designed a specification $\phi_{mid}; \phi_{goal}$, as depicted in Figure 10. The learning curves, also shown in Figure 10, indicate that the proposed refinement significantly enhances the reach probability compared to the original specification. Additionally, Figure 10 illustrates the distribution of states in the MID region from which the GOAL region can be reached, which informs the refinement process. Figure 5 shows how sequential refinement can be applied on higher dimensional state spaces. Different refinements produce varying results, as shown in Figure 5.

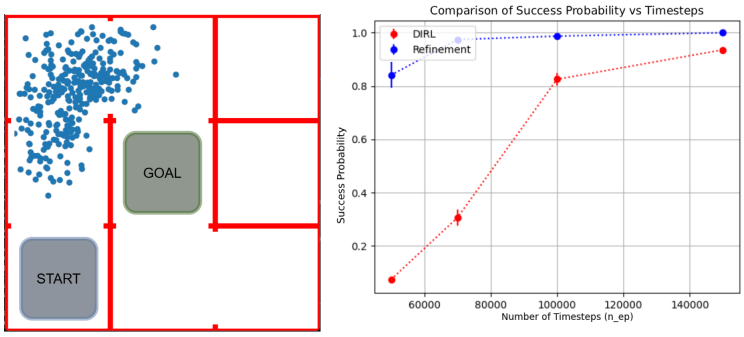


Figure 9: Results of AddRefine in the 9-Rooms Environment. (a) The environment is annotated with start and goal regions (b) Learning curves comparing reach probabilities for DiRL and AutoSpec.

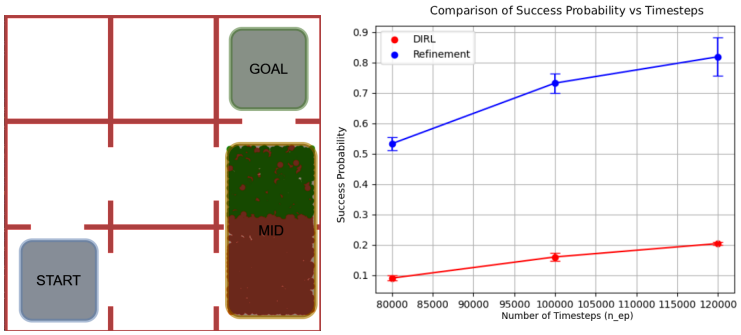


Figure 10: Results of Sequential Specification Refinement in the 9-Rooms Environment. (a) The environment annotated with the distribution of states in the MID region from which the GOAL region can be reached. (b) Learning curves comparing reach probabilities for DiRL and AutoSpec.

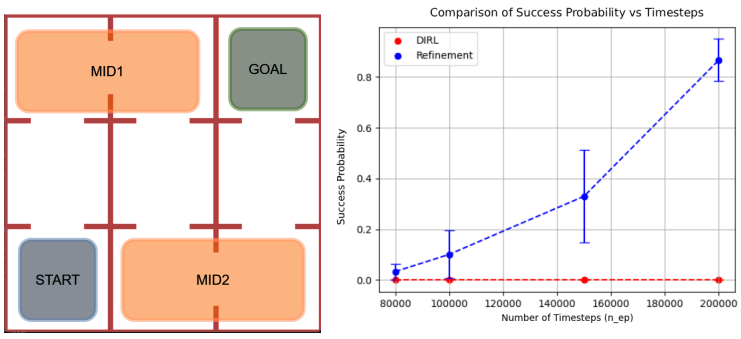


Figure 11: Disjunctive Specification Refinement in 9-Rooms. (a) The environment with regions relevant to the specification $\phi := \phi_{MID1}; \phi_{GOAL}$ or $\phi_{MID2}; \phi_{GOAL}$. (b) Learning curves showing that incorporating an additional specification $\phi_{MID1}; \phi_{MID2}$ is essential to achieve the desired success probability.

Experiment 3: Disjunctive refinement. To validate Algorithm 7, we constructed a 9 Rooms environment with a specification featuring two distinct paths to the goal. Figure 11 illustrates the environment and the regions relevant to the specification $\phi := \phi_{MID1}; \phi_{GOAL}$ **or** $\phi_{MID2}; \phi_{GOAL}$. The learning curves for **OrRefine**, shown in Figure 11, demonstrate that incorporating an additional specification $\phi_{MID1}; \phi_{MID2}$ is essential to achieve good success probability. In contrast, Algorithm 4 and Algorithm 6 fail to perform adequately due to the subspecification having zero reach probability, preventing effective local refinement. This validation underscores the necessity of **OrRefine** in scenarios where sequential modifications alone are insufficient.

All Experiments have been performed using i7-8750H with 32GB RAM and no GPU. Trajectories were collected after training the policy for n timesteps and 5 different seeds.

Hyperparameters for learning algorithms:

1. Learning Rate: 0.0003
2. n steps: 2048
3. Batch size: 64
4. Epochs: 10
5. γ: 0.99

A.4 Limitations

AutoSpec requires finite witnesses to specification satisfaction and hence can only work on finite trajectories. This means that we must consider only finitary fragments of LTL, like SpectRL. While Autospec is sound, i.e if a refinement is found satisfactory, trajectories for the refinement will also satisfy the original specification (Theorem 3.1), it is not complete, i.e. it might fail to find a candidate refinement even if such a refinement exists, especially if the specification satisfiability is extremely low. It is also not guaranteed that the candidate refinement is an 'optimal' refinement, in terms of the tightest bounds possible on the refined predicates.

A.5 Societal Impacts

We wish to improve the performance of Reinforcement Learning algorithms and attempt to improve under-specified human specifications, which have an impact on various applications that aim to deploy RL agents with multi-objective tasks. The applications extend to robotics, path-finding tasks and any tasks that involve manual specifications which could be incorrect. This may have both positive or negative societal impacts depending on the use case of such RL deployments, positive impacts include applications to manufacturing, healthcare, and in-home robotic assistants; while negative impacts would be most consequential in military or surveillance infrastructure. These issues are shared across most work on reinforcement learning algorithms.