
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**
 $O_r \leftarrow O_r \cup \{s_j \mid s_j \in \zeta_i \wedge (\text{len}(\zeta_i) - j) \leq k\}$
 Append the last k states from every trajectory that ended up in the avoid region
 end if
end for
 $c_r \leftarrow c \setminus \text{ConvexHull}(O_r)$ *Create a convex hull around the collected states and remove it from 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*
 $b_r \leftarrow \text{ReachRefine}(b, \zeta)$
 $c_r \leftarrow \text{AvoidRefine}(c, \zeta)$
 $u_r \leftarrow [\beta(u_r) = b_r]$ *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**
 $S_r \leftarrow S_r \cup \zeta_i[\text{len}(\zeta_i)/2]$ *Add center of trajectory as waypoint*
 end if
end for
 $b_r \leftarrow \text{ConvexHull}(S_r) \cap \beta(e)$
 $u'' \leftarrow [\beta(u'') = b_r]$ *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 $e = u \rightarrow u'$, 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*
 $e_r \leftarrow [u^* \rightarrow u', \text{with } \beta(e_r) := \beta(e)]$ *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 $e = u \rightarrow u'$, Graph G , set of trajectories ζ .
 $E = \{e_i \in G \mid e_i = u_i \rightarrow u'\}$ *Collect all 'parents' of u'*
for $e_i \in E$ **do**
 $e_{new} \leftarrow [u \rightarrow u_i, \text{with } \beta(e_{new}) := \beta(e) \text{ and } \beta(e_i) := \beta(e) \cap \beta(e_i)]$ *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 $e = u \rightarrow u'$. 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 $e = u \rightarrow u'$. AddRefine introduces a new vertex u'' and replaces edge e with two sequentially composed edges $e'' = u \rightarrow u''$ and $e' = u'' \rightarrow u'$ 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 $e = u \rightarrow u'$ 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 $e = u \rightarrow u'$, it identifies existing edges $e_i = u_i \rightarrow u'$ and adds new edges $e_{\text{new}} = u \rightarrow u_i$ where $\beta(e_{\text{new}}) = \beta(e)$ and $\beta(e_i) = \beta(e) \cap \beta(e_i)$.

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

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:

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

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:

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

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:

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

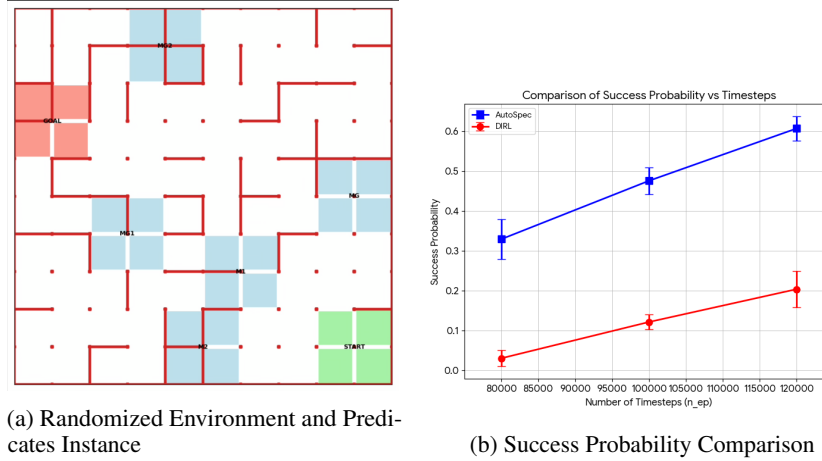


Figure 6: **Experimental Results on the 100-room Gridworld.** (a) A visualization of one of the procedurally generated environment instance, showing the randomized placement of the Start (green), Goal (red), and intermediate refinement regions (blue). (b) Comparative performance analysis showing the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

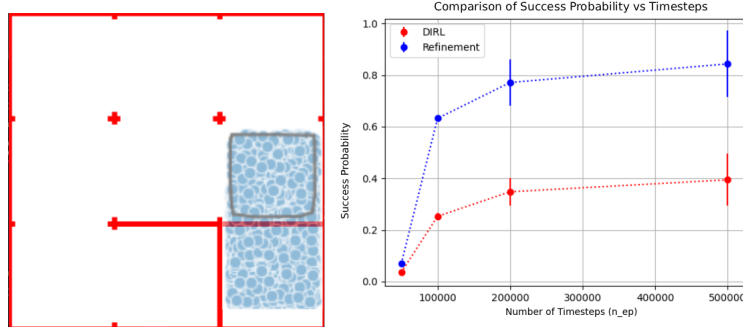


Figure 7: Evaluation of Reach Probabilities in the 9-Rooms Environment. (a) The layout of the 9-rooms environment, showing the walls, doors, and goal regions, and the estimated convex hull for the new reach region, showing how the refinement process effectively restrcuts the reachable states, leading to better satisfaction of the specification (b) A comparison of reach probabilities between DfRL 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.

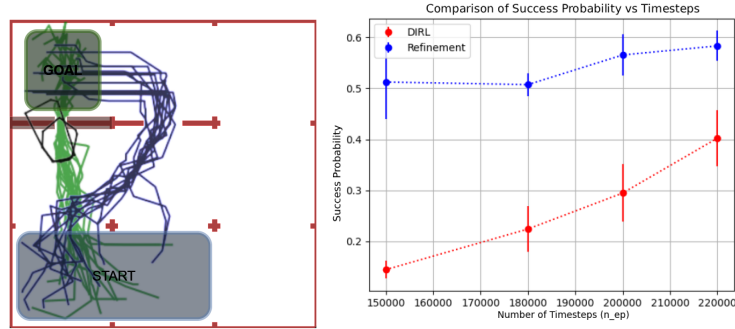


Figure 8: Results of Avoid refinement. (a) The layout of the 9-rooms environment, showing the walls, doors, goal regions and avoid regions (red) and learned trajectories before (green) and after (blue) refinement, with new estimated avoid regions (black) (b) A comparison of reach probabilities between DfRL 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 DfRL Jothimurugan et al. (2021) algorithm and our integration of DfRL 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_{goal}^r . 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.

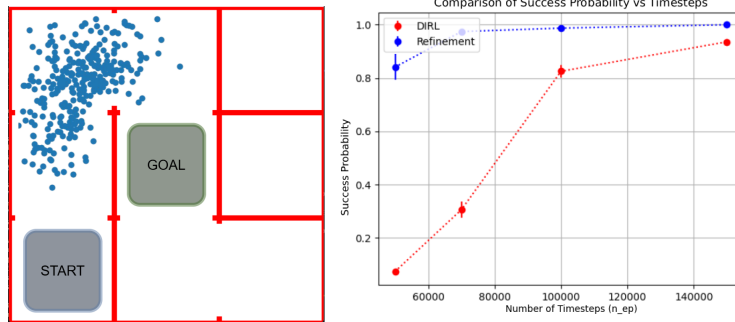


Figure 9: Results of AddRefine in the 9-Rooms Environment. (a) The environment is annotated with start and goal regions (b) Learning curves comparing reach probabilities for DiRL and AutoSpec.

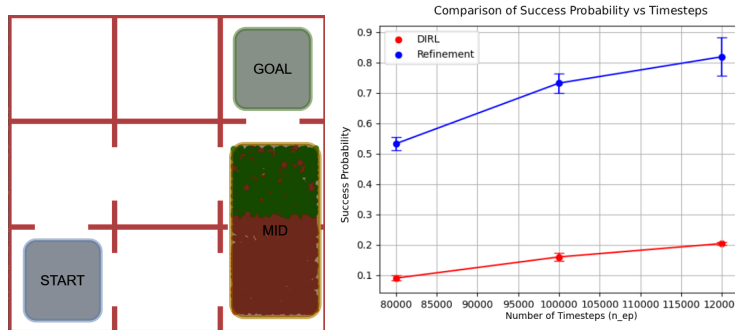


Figure 10: Results of Sequential Specification Refinement in the 9-Rooms Environment. (a) The environment annotated with the distribution of states in the MID region from which the GOAL region can be reached. (b) Learning curves comparing reach probabilities for DiRL and AutoSpec.

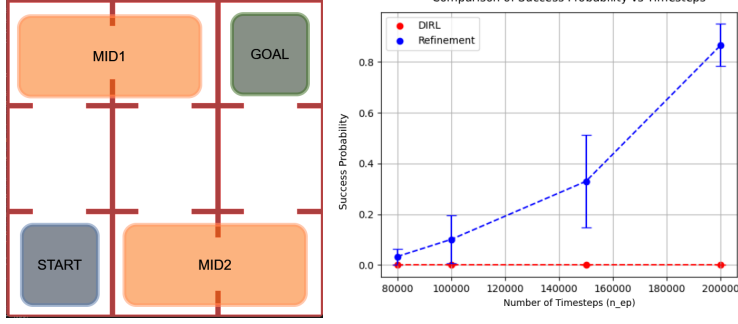


Figure 11: Disjunctive Specification Refinement in 9-Rooms. (a) The environment with regions relevant to the specification $\phi := \phi_{MID1}; \phi_{GOAL}$ or $\phi_{MID2}; \phi_{GOAL}$. (b) Learning curves showing that incorporating an additional specification $\phi_{MID1}; \phi_{MID2}$ is essential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