Deductive Synthesis of Reinforcement Learning Agents for Infinite Horizon Tasks*

Yuning Wang[0009−0000−4317−9758] and He Zhu[0000−0001−9606−150X]

Rutgers University, New Brunswick NJ, USA
{yw895,hz375}@cs.rutgers.edu

Abstract. We propose a deductive synthesis framework for constructing reinforcement learning (RL) agents that provably satisfy temporal reach-avoid specifications over infinite horizons. Our approach decomposes these temporal specifications into a sequence of finite-horizon subtasks, for which we synthesize individual RL policies. Using formal verification techniques, we ensure that the composition of a finite number of subtask policies guarantees satisfaction of the overall specification over infinite horizons. Experimental results on a suite of benchmarks show that our synthesized agents outperform standard RL methods in both task performance and compliance with safety and temporal requirements.

Keywords: Controller Synthesis · Deductive Synthesis · Reinforcement Learning · Temporal Property Verification

1 Introduction

Reinforcement learning (RL) has emerged as a powerful framework for autonomous decision-making in dynamic environments [16,45,33]. Its ability to learn complex behaviors through interaction with the environment makes it an attractive choice for a wide range of applications. However, in many safety-critical domains, it is not sufficient for RL agents to merely optimize rewards. These systems must also adhere to strict safety and liveness constraints that can often be described by temporal logic properties. One widely studied class of such constraints is linear temporal logic (LTL) properties, which express rich behaviors over infinite horizons, such as ensuring safety ("something bad never happens") or liveness ("something good eventually happens") conditions.

Several existing approaches have investigated using *fragments* of LTL to define learning objectives for complex tasks, such as truncated LTL [32], logic formulas combining sequences and Boolean operations over subtasks [26,27], and reward machines [21,20]. These methods typically generate a reward function based on a given specification, which an RL algorithm then utilizes to learn a policy. Some deep RL approaches attempt to optimize a lower bound on the probability of satisfying *general* LTL formulas [4,18,5,46,54,47]. However, a key

* This work is supported by the National Science Foundation under grants CCF-2007799 and CCF-2124155.

challenge with these approaches lies in how rewards are assigned: since success or failure is determined at the trajectory level for temporal logic properties, it is difficult to attribute rewards to specific actions. This exacerbates the credit assignment problem, making it harder for RL algorithms to learn effectively. More importantly, they do not provide formal guarantees for the satisfaction of the specified objectives.

In the domain of formal verification for RL controllers, existing work focuses on ensuring that a controller, when composed with system dynamics and an environment model, remains safe or reaches target regions within a finite task horizon. However, approaches [51,13,11,22,49] based on reachability analysis become intractable for long task horizons due to the need for highly precise abstractions to mitigate approximation errors in reachable state set computations, making them scalable only for short-horizon tasks. Alternative methods reduce the infinite-horizon verification problem to single-step verification by inferring the inductive invariant of a closed-loop system, in the form of Lyapunov functions [9,50] or control barrier functions [1,39]. While effective for simple reach-avoid problems, they are not easily generalized to temporal tasks—such as those involving *repeating* subroutines that must traverse a sequence of subgoals *infinitely*.

We propose a deductive synthesis framework, VEL-∞ (**VE**rification-based **L**earning for Infinite Horizon Tasks), that synthesizes RL agents with formal guarantees of satisfying temporal reach-avoid specifications over *infinite horizons* (formalized in Sec. 2). Given a temporal reach-avoid property Ψ expressed in our specification language, VEL-∞ decomposes the global task into a sequence of finite-horizon subtasks, each characterized by well-defined preconditions and postconditions. These subtask controllers are independently learned and formally verified, and their composition is provably correct with respect to satisfying Ψ over an infinite execution. As an example, consider a shuttle mountain car problem, where the task is to repeatedly drive the car between Position A and Position B. VEL-∞ breaks this task into subtasks: moving from Position A to Position B (red arrow) and then back (blue arrow). Each subtask is defined by a precondition (e.g., starting at Position A with some velocity constraints) and a postcondition (e.g., reaching Position B with some other velocity constraints). Notably, the postcondition of one subtask serves as the precondition of the other, creating a cyclic dependency between the two. VEL-∞ verifies each subtask with its pre- and postconditions, ensuring that when subtasks are composed, they are always invoked in states that satisfy their preconditions—an invariant that guarantees the task can be executed indefinitely.

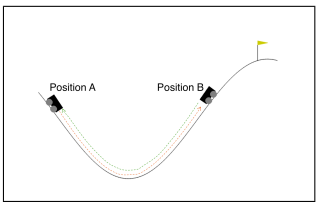


Fig. 1: A shuttle mountain car task where the car is tasked to repeatedly travel back and forth between two positions.

The key contributions of our work are as follows:

- Deductive Synthesis for RL: We introduce a framework that integrates deductive reasoning with reinforcement learning to synthesize control policies satisfying temporal reach-avoid properties over infinite horizons.
- Modular Sub-task Decomposition: Our method decomposes infinite-horizon specifications into sub-tasks, enabling efficient learning using off-the-shelf RL algorithms, and then composes the solutions to ensure global correctness.
- Experimental Validation: We demonstrate the effectiveness of our approach through experiments on a suite of benchmark tasks with continuous state and action space, showing that it outperforms existing RL-based methods in both performance and adherence to formal task specifications.

2 Problem Setup

Environment Model. We model the environment of an RL agent as a discrete-time dynamical system given by the tuple $M[\cdot] = (S, A, F, R, S_0, \cdot)$. Here, $S \subseteq \mathbb{R}^n$ is the state space, $A \subseteq \mathbb{R}^m$ is the action space. The system's dynamics are defined by the equation

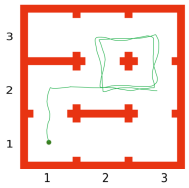
$$s_{t+1} \sim F(s_t, a_t) = f(s_t, a_t, w_t) \qquad w_t \in W$$

where $t \in \mathbb{N}_0$ is a time step, $s_t \in S$ is a state of the system, $a_t \in A$ is a control action, and ω_t is a random time-varying disturbance vector from the disturbance space $W \subseteq \mathbb{R}^p$ at time step t. $R : S \times A \to \mathbb{R}$ is a reward function and $R(s, a)$ defines the immediate reward after the transition from an environment state $s \in S$ with action $a \in A$. S_0 is the set of initial states. We explicitly model the deployment of a control policy π in $M[\cdot]$ as a closed-loop system $M[\pi]$. The system dynamics of $M[\pi]$ are defined by the dynamics function $F : S \times A \to S$ and the control policy $\pi : S \to A$, which maps states to actions, i.e., $a_t = \pi(s_t)$.

Given an initial state s_0, we denote $\zeta \sim M[\pi](s_0)$ as a trajectory (or rollout) of $M[\pi]$ sampled from s_0, i.e. $\zeta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \cdots$ where each $a_t = \pi(s_t)$, each $s_{t+1} \sim F(s_t, a_t)$. We use $\zeta \sim M[\pi]$ for any trajectory of $M[\pi]$ sampled from an initial state $s_0 \in S_0$.

Key Assumptions. We assume that the dynamics f and policy π are Lipschitz continuous functions. The dynamics function f is *known* and is composed of standard trigonometric and polynomial terms, combined with arithmetic operations. Furthermore, we assume a *bounded* disturbance set W, i.e. the system's uncertainty is bounded within a specific range. For example, the disturbance vector ω_t is drawn from a triangular noise distribution at each time step t.

Example 1. Consider the 9Rooms environment in Fig. 2a. The agent (green dot) starts in the bottom left room (1, 1) and is governed by the dynamics function, $s_{t+1} = s_t + 0.1 \min\big(\max(a_t, -1), 1\big) + \omega_t$, where ω_t is drawn from a triangular noise distribution. The state space of the environment is defined as $[0, 3] \times [0, 3] \subset \mathbb{R}^2$. The set of initial states is $[0.4, 0.6] \times [0.4, 0.6]$. The state space contains unsafe areas that should be avoided, i.e., the red wall segments.



(a) 9Rooms environment

$$\Psi_{\texttt{ex}} = \begin{array}{l} \texttt{(Room}_{1,2}\texttt{;} \\ \textbf{repeat(} \\ \quad \texttt{Room}_{3,2}\texttt{;}\ \ \texttt{Room}_{3,3}\texttt{;} \\ \quad \texttt{Room}_{2,3}\texttt{;}\ \ \texttt{Room}_{2,2} \\ \textbf{))\ ensure}\ \texttt{OutsideWall} \end{array}$$

(b) Example task specification $\Psi_{\texttt{ex}}$

Fig. 2: A 9Rooms environment with specification $\Psi_{\texttt{ex}}$ and a satisfying trajectory.

Temporal Reach-Avoid Specifications. We define a high-level specification language that concisely expresses desired RL agent behaviors for infinite-horizon reach-avoid tasks. Inspired by recent specification languages for RL tasks [21,20,26,27], our language restricts classic LTL properties to enable reward shaping for efficient learning. The syntax of our specification language extends the SPECTRL language [26].

Definition 1 (Predicate). *A predicate φ is a quantifier-free Boolean combination of linear inequalities over the environment state variables x:*

$$\varphi ::= P \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$
$$P ::= \mathcal{A} \cdot x \leq b \text{ where } \mathcal{A} \in \mathbb{R}^{|x|}, b \in \mathbb{R}$$

A state $s \in S$ satisfies a predicate φ, denoted as $s \models \varphi$, iff $\varphi(s)$ is true.

Definition 2 (Task Specification). *A task specification Ψ is either a non-repetitive task specification ψ or a repetitive task specification ψ_R.*

$$\Psi ::= \psi \mid \psi_R$$

The syntax of ψ and ψ_R defined as:

$$\begin{array}{l} \psi ::= \textbf{\textit{achieve}}\ \varphi \\ \quad \mid \psi\ \textbf{\textit{ensure}}\ \varphi \\ \quad \mid \psi_1\textbf{;}\ \psi_2 \\ \quad \mid \psi_1\ \textbf{\textit{or}}\ \psi_2 \end{array} \qquad \begin{array}{l} \psi_R ::= \textbf{\textit{repeat}}\ \psi \\ \quad \mid \psi_R\ \textbf{\textit{ensure}}\ \varphi \\ \quad \mid \psi\textbf{;}\ \psi_{R1} \\ \quad \mid \psi_{R1}\ \textbf{\textit{or}}\ \psi_{R2} \end{array}$$

Intuitively, a specification Ψ in our language is defined over trajectories with a finite prefix ψ and a possibly infinite suffix ψ_R. An infinite trajectory ζ satisfies ψ if there exists a finite prefix of ζ that satisfies ψ. In the definition for ψ, the first clause means that the prefix trajectory should eventually reach a state that satisfies the predicate φ. The second clause says that the prefix trajectory should satisfy specification ψ while always staying in states that satisfy a *safety constraint* φ. The third clause states that the prefix trajectory should sequentially satisfy ψ_1 followed by ψ_2. The fourth clause means that the prefix trajectory

should satisfy either ψ_1 or ψ_2. An infinite trajectory ζ satisfies a repetitive task ψ_R if, beyond some time step i, the suffix of the trajectory $\zeta_{i:\infty}$ conforms to a temporal property **repeat** ψ, indicating a repeated execution of the task ψ to achieve its goal states infinitely often.

Example 2. The task specification $\Psi_{\texttt{ex}}$ (Fig. 2b) for the 9Rooms environment requires the agent to first reach room $(1, 2)$ and then repeatedly traverse four rooms in a circular way infinitely, while avoiding collision with any red wall segments. $\texttt{Room}_{\texttt{x,y}}$ is the abbreviation for **achieve** `Center(x,y)` where the predicate `Center(x,y)` is true for states where the agent locates at the center of the room indexed by `(x,y)`. The `OutsideWall` predicate is true for states where the agent stays away from any wall segments. The green curve in Fig.2a shows an agent trajectory that satisfies $\Psi_{\texttt{ex}}$.

We formalize the semantics of task specification satisfaction in Ψ as follows.

Definition 3 (Task Specification Satisfaction). *Given an infinite trajectory $\zeta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \cdots$, ζ satisfies a non-repetitive task specification ψ, denoted as $\zeta \models \psi$ if and only if there exists a time step i such that the finite prefix trajectory $\zeta_{0:i}$ satisfies ψ. The satisfaction of ψ by $\zeta_{0:i}$, namely $\zeta_{0:i} \models \psi$, is defined inductively as follows:*

$$\begin{aligned}
&\zeta_{0:i} \models \textbf{\textit{achieve}}\ \varphi && \Leftrightarrow \exists\ 0 \le j \le i \ \textit{s.t.}\ s_j \models \varphi \\
&\zeta_{0:i} \models \psi\ \textbf{\textit{ensure}}\ \varphi && \Leftrightarrow \zeta_{0:i} \models \psi \ \textit{and} \ \forall\ 0 \le j < i, s_j \models \varphi \\
&\zeta_{0:i} \models \psi_1 \textbf{\textit{;}}\ \psi_2 && \Leftrightarrow \exists\ 0 \le j < i \ \textit{s.t.}\ \zeta_{0:j} \models \psi_1 \ \textit{and}\ \zeta_{j+1:i} \models \psi_2 \\
&\zeta_{0:i} \models \psi_1\ \textbf{\textit{or}}\ \psi_2 && \Leftrightarrow \zeta_{0:i} \models \psi_1 \ \textit{or}\ \zeta_{0:i} \models \psi_2
\end{aligned} \tag{1}$$

The satisfaction of a repetitive task specification ψ_R by an infinite trajectory ζ, denoted as $\zeta \models \psi_R$, is defined inductively as follows:

$$\begin{aligned}
&\zeta \models \textbf{\textit{repeat}}\ \psi && \Leftrightarrow \exists\ i \ \textit{s.t.}\ \zeta_{0:i} \models \psi \ \textit{and}\ \zeta_{i+1:\infty} \models \textbf{\textit{repeat}}\ \psi \\
&\zeta \models \psi_R\ \textbf{\textit{ensure}}\ \varphi && \Leftrightarrow \zeta \models \psi_R \ \textit{and}\ \forall i, s_i \models \varphi \\
&\zeta \models \psi \textbf{\textit{;}}\ \psi_R && \Leftrightarrow \exists\ i \ \textit{s.t.}\ \zeta_{0:i} \models \psi \ \textit{and}\ \zeta_{i+1:\infty} \models \psi_R \\
&\zeta \models \psi_{R1}\ \textbf{\textit{or}}\ \psi_{R2} && \Leftrightarrow \zeta \models \psi_{R1} \ \textit{or}\ \zeta \models \psi_{R2}
\end{aligned} \tag{2}$$

Finally, ζ satisfies Ψ, denoted as $\zeta \models \Psi$, if and only if ζ satisfies the underlying ψ or ψ_R.

Comparison with LTL. Compared to declarative LTL properties, our language Ψ is operationally direct, task-oriented, and tailored for RL agents. Notably, our language encodes a restricted fragment of LTL properties, disallowing temporal dependencies that combine temporal operators and Boolean connectives arbitrarily. The exact fragment of LTL that our specification language ψ_R supports is characterized as follows, where φ is a quantifier-free predicate over state variables:

$$\begin{aligned}
&\psi ::= \varphi \mid \mathbf{F}(\varphi \wedge \mathbf{X}\psi) \wedge \mathbf{G}\varphi \mid \psi \vee \psi \\
&\psi_R ::= \psi \mid \psi \wedge \mathbf{G}\psi
\end{aligned} \tag{3}$$

This fragment focuses on task execution order. It captures task sequencing as $\mathbf{F}(\varphi_1 \wedge \mathbf{XF}\varphi_2)$, where the agent must first reach a state satisfying φ_1 and then proceed to a state satisfying φ_2. It also enforces safety constraints through $\mathbf{F}(\varphi_1 \wedge \mathbf{XF}\varphi_2) \wedge \mathbf{G}\varphi_3$, ensuring that the agent adheres to the safety condition φ_3 at all times. Additionally, it supports infinite repetition with $\mathbf{GF}(\varphi_1 \wedge \mathbf{XF}\varphi_2)$, requiring the agent to repeatedly visit states satisfying φ_1 and φ_2 indefinitely. Our specification language employs the `achieve`, `ensure`, and `repeat` operators to naturally capture eventual satisfaction, safety constraints, and global invariance, making it well-suited for expressing this fragment. Our language does not adopt classical LTL syntax in order to improve accessibility for RL practitioners who may not have a formal methods background. Instead, it uses an intuitive structure that aligns with common RL problem formulations, allowing users to specify infinite sequences of subtasks compositionally. This restricted fragment, as demonstrated below, enables an efficient verification procedure.

Relation to SPECTRL. Our specification language is based on the SPECTRL language [26]. VEL-∞ is part of the recent research trend in RL for LTL, leveraging RL techniques to optimize LTL objectives directly in continuous state and action spaces. Existing work SPECTRL [26] and DIRL [27] in this direction has developed efficient algorithms for handling temporal reach-avoid specifications defined over a finite number of subtasks. However, there remains a significant gap in developing a general RL algorithm capable of handling infinite-horizon LTL specifications, such as those requiring infinite oscillation between key subgoals. Formally, the SPECTRL and DIRL approaches support ψ (finite reach-avoid tasks) but do not generalize to ψ_R, which encompasses temporal reach-avoid objectives over infinite horizons. VEL-∞ fills this gap by introducing a novel framework that extends RL capabilities to the setting in ψ_R, and provide formal correctness guarantees for this extension using formal methods.

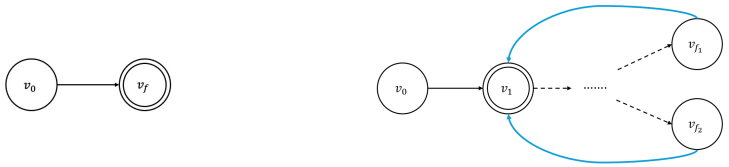
Problem Formulation. Given an environment model $M[\cdot]$ and a specification Ψ, our goal is to find a control policy π such that for all $\zeta \sim M[\pi]$, $\zeta \models \Psi$.

3 Verification-Based Learning over Infinite Horizons

Main Framework. Our approach, **Ve**rification-Based **L**earning for Infinite horizon properties (VEL-∞), leverages the compositional property of a task specification Ψ in our language to construct an abstract reachability graph for the task. We use a graph search algorithm similar to the Dijkstra's algorithm to train formally verified policies for edge subtasks within the abstract reachability graph and construct a final policy that satisfies the overall task specification by strategically combining edge policies.

3.1 Abstract Reachability Graph Construction

Our approach builds on [27] to convert a task specification Ψ into an abstract reachability graph G_Ψ, but unlike [27], we do not require G_Ψ to be acyclic.



(a) G_ψ for $\psi =$ **achieve** φ (b) G_{ψ_R} for $\psi_R =$ **repeat** ψ

Fig. 3: Visualization of abstract reachability graph construction

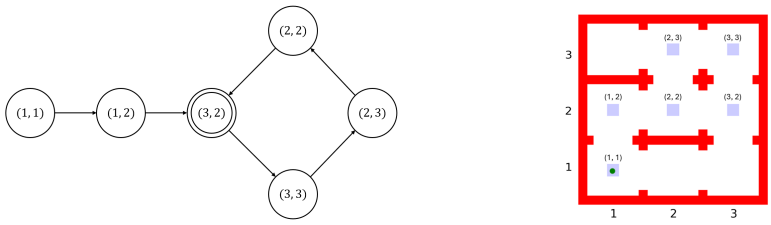
We define $G_\Psi = (V_\Psi, E_\Psi, V_{F_\Psi}, \mu_\Psi, \beta_\Psi, v_{0_\Psi})$ where each vertex $v \in V_\Psi$ represents a region in the state space captured by some predicate φ_v (Definition 1). The predicate labeling function $\mu_\Psi(v)$ maps each vertex $v \in V_\Psi$ to its corresponding predicate φ_v. The initial vertex $v_{0_\Psi} \in V_\Psi$ encodes the initial state region S_0 of the environment model. The set V_{F_Ψ} includes all final vertices that may induce cycles (introduced by the *repeat* operator) defining recurring patterns in infinite trajectories. Each edge $e = (u \to v) \in E_\Psi$ corresponds to a goal-directed subtask with precondition $\mu_\Psi(u)$ and postcondition $\mu_\Psi(v)$:

$$\mathcal{T}_e : \ \mu_\Psi(u) \rightsquigarrow \mu_\Psi(v) \mid \beta_\Psi(e)$$

that transitions the agent from subregion $\mu_\Psi(u)$ to subregion $\mu_\Psi(v)$. The safety constraint labeling function $\beta_\Psi(e)$ maps each edge e to a predicate φ_e (enforced by the *ensure* operator) that must hold true on all states during the subtask represented by e. Intuitively, a trajectory that satisfies the specification Ψ should transition through a sequence of subgoal regions in V_Ψ, starting from the initial vertex and possibly cycling at some final vertex in V_{F_Ψ}, while adhering to the safety constraints associated with the edges in E_Ψ along the path.

We provide a high-level description of the algorithm used to construct the abstract reachability graph G_Ψ for a task specification Ψ. For more details, we refer interested readers to the extended version [58]. The abstract reachability graph G_ψ for a specification in the form of $\psi =$ **achieve** φ is shown in Fig. 3a. It consists of two vertices: the initial vertex v_0 with a predicate φ_{v_0} encoding the initial state space S_0 and the final vertex v_f with the predicate φ. The directed edge $(v_0 \to v_f)$ represents the subtask $S_0 \rightsquigarrow \varphi$ of transitioning from the initial region S_0 to the target region defined by φ. The graph for $\psi = \psi_1$ **ensure** φ can be obtained by updating the safety constraint $\beta(e)$ of each edge e within G_{ψ_1} (recursively constructed) through a conjunction $\beta(e) \wedge \varphi$. The graph for $\psi = \psi_1$**;** ψ_2 is constructed by adding edges in the form of $(v_{f_i} \to v_j)$, where v_{f_i} is any final vertex of G_{ψ_1} and v_j is any neighbor vertex of the initial vertex $v_{0_{\psi_2}}$ in G_{ψ_2}. Then $v_{0_{\psi_2}}$ and all its edges are removed. This construction ensures that the subtask for ψ_2 begins after that of ψ_1 completes. The graph for G_ψ for $\psi = \psi_1$ **or** ψ_2 is derived by merging the initial vertices $v_{0_{\psi_1}}$ and $v_{0_{\psi_2}}$ into a single initial vertex, allowing the agent to choose which task to complete.

For a repetitive task specification $\psi_R =$ **repeat** ψ, the graph G_{ψ_R} is obtained by adding edges that connect all the final vertices of G_ψ to the neighbors of the



(a) Abstract reachability graph for $G_{\psi_{\texttt{ex}}}$ (b) Subregions for Each Vertex

Fig. 4: The abstract reachability graph for specification $\Psi_{\texttt{ex}}$ in Fig. 2

initial vertex v_0, allowing the task ψ to be repeated. We then set these neighbors as the final vertices of G_{ψ_R}, marking the start and end of a cycle. The added edges are shown in blue in Fig. 3b. The graphs for other types of repetitive task specifications can be recursively constructed similarly to their counterparts in ψ.

Example 3. Consider the task specification $\Psi_{\texttt{ex}}$ for 9Rooms in Fig. 2b, its abstract reachability graph $G_{\Psi_{\texttt{ex}}}$ is shown in Fig. 4a. Subregions represented by the predicates are marked in Fig. 4b. The safety constraint for each edge (no collision with red walls) is not shown.

After construction, the abstract reachability graph G_Ψ serves as a structured representation of the task specification Ψ. With subtasks encoded in the edges in G_Ψ, the completion of the full specification can be viewed as visiting vertices in G_Ψ in a specific order. Definition 4 shows how to validate if an agent trajectory accomplishes the task Ψ using the abstract reachability graph G_Ψ.

Definition 4 (Abstract Reachability Graph Satisfaction). *Given an infinite trajectory $\zeta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \cdots$, ζ satisfies the abstract reachability graph G_ψ of a non-repetitive task specification ψ, denoted as $\zeta \models G_\psi$, if and only if there exists a finite sequence of indices $i_0 \le i_1 \le \cdots \le i_n$ and a path $\rho = v_0 \to v_1 \to \cdots \to v_n$ in G_ψ such that:*

- $v_n \in V_{F_\psi}$
- $\forall\ 0 \le j \le n,\ s_{i_j} \models \mu_\psi(v_j)$
- $\forall\ 0 \le j < n,\ \zeta_{i_j:i_{j+1}} \models \beta_\psi(v_j \to v_{j+1})$

Given the abstract reachability graph G_{ψ_R} of a repetitive specification ψ_R, ζ satisfies G_{ψ_R}, denoted as $\zeta \models G_{\psi_R}$, if and only if there exists an infinite sequence of indices $i_0 \le i_1 \le \cdots \le i_n \cdots$ and an infinite path $\rho = v_0 \to v_1 \to \cdots \to v_n \cdots$ in G_{ψ_R} such that:

- $v_n \in V_{F_{\psi_R}}$ *occurs infinite number of times in* ρ
- $\forall\ j \ge 0,\ s_{i_j} \models \mu_{\psi_R}(v_j)$
- $\forall\ j \ge 0,\ \zeta_{i_j:i_{j+1}} \models \beta_{\psi_R}(v_j \to v_{j+1})$

Theorem 1. *Given a task specification Ψ and its abstract reachability graph G_Ψ, for any infinite trajectory ζ, we have $\zeta \models \Psi$ if and only if $\zeta \models G_\Psi$.*

Proof. The theorem follows from a straightforward induction on Ψ.

Hereafter, we often omit the suffix Ψ for simplicity, such as referring to G_Ψ as G when the context is clear.

3.2 Dijkstra-Style Abstract Reachability Graph Search

After converting a task specification Ψ into an abstract reachability graph G, where subtasks are modeled as directed edges, a practical learning strategy [27] exploits this graph structure by decomposing the learning problem into goal-reaching subtasks. Individual edge policies are trained for the subtasks associated with the edges and subsequently composed to form a solution for the full specification Ψ[1]. Since the edges of G represent smaller, localized subtasks, this approach avoids the complexity of satisfying the entire specification with a single RL algorithm. However, applying this decomposition strategy to infinite-horizon temporal reach-avoid problems is challenging, as it requires ensuring that the composition of finite subtasks can be executed over infinite horizons. VEL-∞ proceeds in the following three steps to tackle this challenge:

- Step 1: For each edge $e = u \to v$ in G, learn an edge policy π_e that aims to transition the system from any state $s \in \mu(u)$ to some state $s' \in \mu(v)$, while avoiding states that violate the safety constraint $\beta(e)$.
- Step 2: Formally verify that π_e can transition the system from $\mu(u)$ to $\mu(v)$ within c_e timesteps (where c_e is a parameter learned during training) while ensuring compliance with $\beta(e)$.
- Step 3: Use a graph search algorithm in conjunction with the edge costs c_e to compute a path $\rho = v_0 \to v_1 \to \cdots \to (v_k \to v_{k+1} \to \cdots \to v_{k+l})^\omega$ with a cyclic structure labeled with ω where $v_k = v_{k+l}$ and $v_k \in V_F$ is a final vertex in G. The path ρ minimizes $c(\rho) = \langle \sum_{j=k}^{k+l-1} c_{e_j}, \sum_{j=0}^{k-1} c_{e_j} \rangle$ by *lexicographic ordering*, where $e_j = v_j \to v_{j+1}$ corresponds to an edge policy $\pi_j = \pi_{e_j}$.

Our key idea is to establish a formally verified inductive invariant ensuring that, for any searched graph path ρ containing a cyclic structure $(v_k \to v_{k+1} \to \cdots \to v_{k+l})^\omega$ at step three, the set of states reachable at the end of the cycle on v_{k+l} is a subset of the states in $\mu(v_{k+l})$. Since $v_{k+l} = v_k$, we have shown that the set of states reachable at the end of the cycle is contained within the states at the beginning of the cycle, $\mu(v_k)$. This inductive property guarantees the infinite execution of the cycle along ρ.

Finally, we construct the overall controller as a *stateful* path policy:

Definition 5 (Path Policy). *Given the edge policies along with a path $\rho = v_0 \to v_1 \to \cdots \to (v_k \to v_{k+1} \to \cdots \to v_{k+l})^\omega$ where $v_k = v_{k+l}$ and $v_k \in V_F$ in G, we define the overall controller as a path policy $\pi_\rho = \pi_0 \circ \cdots \circ \pi_{k-1} \circ$*

[1] This is feasible because we assume access to a simulator with known dynamics.

$(\pi_k \circ \ldots \circ \pi_{k+l-1})^\omega$ designed to achieve the sequence of edges in ρ. It navigates from $\mu(v_0)$ to $\mu(v_k)$, and then infinitely visits $\mu(v_k)$ through the cycle from v_k. It executes each policy π_j until it reaches $\mu(v_{j+1})$, and then switch to π_{j+1} unless $j = k + l - 1$ in which case j is reset to k. Note that π_ρ is stateful since it internally keeps track of the index j of the current policy.

We explain controller verification and the graph search algorithm in detail below. Specifically, VEL-∞ decomposes the verification of a path policy to the verification of its edge policies. VEL-∞ verifies an edge policy π_e against an environment model $M[\cdot]$ and a specification $\mathcal{T}_e$ using abstract interpretation.

Definition 6 (Symbolic Rollouts for Edge Policies). *Given an environment model $M[\cdot]$, an edge policy π_e, a subtask specification $\mathcal{T}_e : \mu(u) \rightsquigarrow \mu(v) \mid \beta(e)$, an abstract domain $\mathcal{D}$ with abstraction function α and concretization function γ, an abstract transformer $F^{\mathcal{D}}$ for the state transition function F of M, and abstract transformer $\pi_e^{\mathcal{D}}$ for the policy, a symbolic rollout of $M[\pi_e]$ over $\mathcal{D}$ constructs an H-step sequence of symbolic states $\zeta^{\mathcal{D}} = S_0^{\mathcal{D}}, S_1^{\mathcal{D}}, \ldots, S_H^{\mathcal{D}}$ where $S_0^{\mathcal{D}} = \alpha(\{s \mid s \models \mu(u)\})$, the abstraction of the initial subtask states. The symbolic state $S_{t+1}^{\mathcal{D}}$ that overapproximates the set of reachable states from the initial subtask states at time step $t+1$ is computed as:*

$$S_{t+1}^{\mathcal{D}} = F^{\mathcal{D}}(S_t^{\mathcal{D}}, A_t^{\mathcal{D}})$$

where $A_t^{\mathcal{D}} = \pi_e^{\mathcal{D}}(S_t^{\mathcal{D}})$ is an overapproximation of the set of possible actions at t.

Edge Policy Verification. Our edge policy verification procedure leverages the concretization operator γ of the abstract domain $\mathcal{D}$. The concretization $\gamma(S_t^{\mathcal{D}})$ is defined as the set of concrete states represented by the abstract state $S_t^{\mathcal{D}}$. To balance precision and verification efficiency, we assume that this concretization can be approximated using a tight interval $\gamma_I(S_t^{\mathcal{D}})$, which represents the most precise interval enclosing all concrete states of $S_t^{\mathcal{D}}$. The edge policy π_e satisfies the subtask specification $\mathcal{T}_e : \mu(u) \rightsquigarrow \mu(v) \mid \beta(e)$, denoted as $\pi_e \models \mathcal{T}_e$, if the following conditions hold on the symbolic rollout of $M[\pi_e]$, $\zeta^{\mathcal{D}} = S_0^{\mathcal{D}}, S_1^{\mathcal{D}}, \ldots, S_H^{\mathcal{D}}$. First, every reachable state satisfies the safety constraint $\beta(e)$:

$$\forall 0 \le t < H,\ \forall s \in \gamma_I(S_t^{\mathcal{D}}),\ s \models \beta(e)$$

Second, the reachable states at the final step H satisfy the subtask's goal condition,

$$\forall s \in \gamma_I(S_H^{\mathcal{D}}),\ s \models \mu(v)$$

Path Policy Verification. A path policy π_ρ is formally verified $\pi_\rho \models G$ if and only if each edge policy π_e on ρ is formally verified i.e. $\pi_e \models \mathcal{T}_e$.

Example 4. Fig. 5 illustrates the symbolic rollouts of the edge policies within a verified path policy for the 9Room environment. The abstract domain $\mathcal{D}$ used for edge policy verification is Taylor Model (TM) flowpipes, which we discuss further in Sec. 3.3. Each box represents the interval concretization of a symbolic state, with colors distinguishing different edge policies.

Theorem 2 (Path Policy Verification Soundness). *Given an environment model $M[\cdot]$, an abstract reachability graph G, a path policy $\pi_\rho = \pi_0 \circ \cdots \circ \pi_{k-1} \circ (\pi_k \circ \ldots \circ \pi_{k+l-1})^\omega$ along with a path $\rho = v_0 \to v_1 \to \cdots \to (v_k \to v_{k+1} \to \cdots \to v_{k+l})^\omega$ in G where $v_k = v_{k+l}$ and $v_k \in V_F$ is a final vertex, if $\pi_\rho \models G$, then for any $\zeta \sim M[\pi_\rho]$, $\zeta \models G$.*

Proof. For any infinite trajectory $\zeta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \cdots \sim M[\pi_\rho]$, by construction according to Definition 5, there exists an infinite sequence of indices $i_0 \leq \cdots \leq i_k \leq \cdots i_{k+l} \leq \cdots \leq i_{k+2l} \leq \cdots$, such that for all $0 \leq j < k$, $\zeta_{i_j:i_{j+1}} \sim M[\pi_j](s_{i_j})$ is produced by a policy π_j on the prefix of π_ρ, and for all $j \geq k$, $\zeta_{i_j:i_{j+1}} \sim M[\pi_{j \bmod l}](s_{i_j})$ is produced by a policy $\pi_{j \bmod l}$ on the cyclic structure of π_ρ. According to Definition 6, the verification procedure guarantees that for all $0 \leq j < k$, we have $s_{i_j} \models \mu(v_j)$ and $\zeta_{i_j:i_{j+1}} \models \beta(v_j \to v_{j+1})$. Similarly, for all $j \geq k$, we have $s_{i_j} \models \mu(v_{j \bmod l})$ and $\zeta_{i_j:i_{j+1}} \models \beta(v_{j \bmod l} \to v_{j \bmod l+1})$. As such, ζ visits states in $\mu(v_k)$ at i_k, i_{k+l}, i_{k+2l}, …, etc, which means that ζ visits v_k infinitely often. Based on Definition 4, we have $\zeta \models G$ by construction.

Theorems 1 and 2 jointly demonstrate that, given an environment model $M[\cdot]$ and a specification Ψ, any trajectory $\zeta \sim M[\pi_\rho]$ generated by a formally verified path policy π_ρ satisfies $\zeta \models \Psi$.

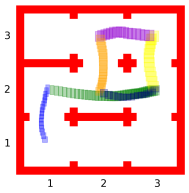


Fig. 5: A symbolic rollout for the 9Rooms environment.

Invariants at Final Vertices. The proof of Theorem 2 explains why VEL-∞ effectively handles infinite-horizon properties. It ensures that when the path policy π_ρ completes a cycle traversal at timestep $k + l$, the reached states remain within $\mu(v_{k+l})$. Since $v_k = v_{k+l}$, this means the states at the end of the cycle on v_{k+l} belong to the set of states $\mu(v_k)$ at the cycle's starting point at timestep k, where v_k is a final vertex. For example, at the center of $\texttt{Room}_{3,2}$ in Fig. 5, the last symbolic state for the edge policy in dark blue is fully enclosed within the initial yellow symbolic state, indicating cyclic behaviors.

Edge Cost and Path Cost. We measure the cost c_e of each edge e within an abstract reachability graph G as the horizon H of the symbolic rollout constructed to verify the subtask $\mathcal{T}_e$ of e (Definition 6). The cost c_ρ of a path $\rho = v_0 \to v_1 \to \cdots \to (v_k \to v_{k+1} \to \cdots \to v_{k+l})^\omega$ where $v_k = v_{k+l}$ and $v_k \in V_F$ is a tuple $\langle \sum_{j=k}^{k+l-1} c_{e_j}, \sum_{j=0}^{k-1} c_{e_j} \rangle$ where $e_j = v_j \to v_{j+1}$ corresponds to an edge on ρ. The first element measures the sum of the edge costs within the cyclic structure of ρ, and the second measures the sum of the edge costs on the prefix that leads to the cycle. VEL-∞ compares path costs using *lexicographic ordering*, meaning that we prefer the final vertex that has the least cost cycle back to itself for infinite visits. This is reasonable because, as the cycle is traversed infinitely, its cost dominates the cost of the prefix. VEL-∞ aims to learn the optimal path policy corresponding to the path in G with the least path cost.

Main Search Algorithm. The outline of our learning algorithm VEL-∞ is shown in Algorithm 1, which takes as input an environment model $M[\cdot]$ and an

Algorithm 1 VEL-∞ Policy Search Algorithm

1: **procedure** VEL-∞ $(M[\cdot], G = (V, E, V_F, \mu, \beta, v_0))$
2: **for each** $v_f \in V_F$ **do**
3: $\langle c^{\omega}_{v_f}, c^{v_0}_{v_f} \rangle$, $\pi_{v_f} \leftarrow$ LEARN$(M[\cdot], G, v_0, v_f)$
4: **return** π_{v_f} with the least cost $\langle c^{\omega}_{v_f}, c^{v_0}_{v_f} \rangle$ by *lexicographic ordering*
5:
6: **procedure** LEARN $(M[\cdot], G, v_0, v_f,$ Rec = True)
7: Initialize priority queue Q with $\{(0, v_0)\}$
8: Initialize visited set $S \leftarrow \emptyset$
9: **while** $Q \neq \emptyset$ **do**
10: $c_u, u \leftarrow$ Dequeue the vertex with the least cost in Q
11: **if** $u \notin S$ **then**
12: **for each** outgoing edge $e = (u, v) \in G.E$ of u **do**
13: ▷ Learn an edge policy $\pi_e \models \mathcal{T}_e$ with TRAINVERIFY (Algorithm 2)
14: $\pi_e, H, S^{\mathcal{D}}_H \leftarrow$ TRAINVERIFY $(M[\cdot], \mathcal{T}_e : \ \mu(u) \rightsquigarrow \mu(v) \mid \beta(e))$
15: $c_e \leftarrow H$
16: ▷ Strengthen $\mu(v)$ by the (*verified*) set of states reachable at v
17: **if** $v \notin S$ **then** $\mu(v) \ \leftarrow \mu(v) \wedge \gamma_I(S^{\mathcal{D}}_H)$
18: **if** $v = v_f \wedge$ Rec $\wedge$ v_f has outgoing edges **then**
19: ▷ Find the least-cost cycle from v_f
20: $\langle \_, \ c_\omega \rangle, \ \pi_\omega \leftarrow$ LEARN$(M[\cdot], G, v_f, v_f,$ Rec = False)
21: $\pi_{v_f} \leftarrow$ Path policy from v_0 to v_f followed by π_ω
22: **return** $\langle c_\omega, \ c_u + c_e \rangle, \pi_{v_f}$
23: **else if** $v = v_f$ **then**
24: $\pi_{v_f} \leftarrow$ Path policy from v_0 to v_f
25: **return** $\langle 0, \ c_u + c_e \rangle, \pi_{v_f}$
26: **else**
27: Enqueue $(c_u + c_e, v)$ into Q
28: $S \leftarrow S \cup \{u\}$

abstract reachability graph G that encodes a task specification Ψ. The algorithm is built on top of the Dijkstra's algorithm to traverse the reachability graph G. For each final vertex $v_f \in V_F$, the algorithm invokes the LEARN procedure at line 3 to learn edge policies that can be used to reach v_f.

In LEARN, a priority queue Q is maintained to store pairs (c, v), where c represents the path cost along the shortest path from the initial vertex v_0 to vertex v. Q is initialized to $\{(0, v_0)\}$ at line 7. Iteratively at line 9, LEARN handles an unprocessed vertex u closest to the initial vertex v_0 from Q. For each edge $e = u \to v$ in G, we learn an edge policy π_e for the subtask $\mathcal{T}_e : \ \mu(u) \rightsquigarrow \mu(v) \mid \beta(e)$ that transitions the system from any state $s \in \mu(u)$ to $s' \in \mu(v)$ safely with respect to the safety constraint $\beta(e)$. We invoke the TRAINVERIFY algorithm (given in Sec. 3.3) at line 14 to synthesize a formally verified policy π_e for the subtask: $\pi_e \models \mathcal{T}_e$. TRAINVERIFY also returns H as the horizon of the symbolic rollout constructed to verify the subtask $\mathcal{T}_e$ of e (Definition 6). We use H as the edge cost c_e of e (line 15). Importantly, TRAINVERIFY also returns the

symbolic state $S_H^{\mathcal{D}}$ verified at timestep H and, at line 17, we strengthen $\mu(v)$, the set of reachable states represented by v, as the set of concrete states in $\gamma_I(S_H^{\mathcal{D}})$, which provides a tighter bound on the initial states for the subtasks represented by the outgoing edges from v. If v is the final vertex v_f and v has outgoing edges that induce a cycle (via the **repeat** operator), we launch a separate round of Dijkstra's algorithm to find the least-cost cycle back to v_f within the strongly connected component of G that includes v_f at line 20. We construct the path policy that satisfies the full task specification at line 21 by first reaching v_f using policies along the shortest path from v_0 to v_f and then combining policies along the shortest cycle back to v_f. The algorithm returns the policy and its cost in line 22. Similarly, for non-repetitive specifications, when v is the final vertex, we simply return the path policy from v_0 to v_f (line 25).

3.3 Provably Correct Edge Policy Synthesis

We introduce the TrainVerify procedure to train an edge policy π_e that is formally verified with respect to the subtask specification $\mathcal{T}_e : \mu(u) \rightsquigarrow \mu(v) \mid \beta(e)$ for an abstract reachability graph edge $e : u \to v$, where $\mu(u)$ and $\mu(v)$ define the initial and target state space regions and $\beta(e)$ represents a safety constraint. TrainVerify is outlined in Algorithm 2.

We first employ an arbitrary deep RL algorithm to train a neural network policy, $\pi_{\texttt{NN}}$, for task $\mathcal{T}_e$ using a reward function derived from the specification of $\mathcal{T}_e$. Intuitively, trajectories that satisfy the given specification will receive higher rewards than those that do not, steering the policy toward behaviors that effectively fulfill the task requirements. The following definition introduces a standard approach to quantitatively evaluate the predicates defined in Definition 1, allowing for a continuous predicate evaluation:

Definition 7 (State Correctness Loss Function). *For a predicate φ (Definition 1) over states $s \in S$, we define a non-negative loss function $\mathcal{L}(s, \varphi)$ such that $\mathcal{L}(s, \varphi) = 0$ iff s satisfies φ, i.e. $s \models \varphi$. We define $\mathcal{L}(s, \varphi)$ recursively, based on the possible shapes of φ:*

- $\mathcal{L}(s, \mathcal{A} \cdot x \leq b) := \max(\mathcal{A} \cdot s - b, 0)$
- $\mathcal{L}(s, \varphi_1 \wedge \varphi_2) := \max(\mathcal{L}(s, \varphi_1), \mathcal{L}(s, \varphi_2))$
- $\mathcal{L}(s, \varphi_1 \vee \varphi_2) := \min(\mathcal{L}(s, \varphi_1), \mathcal{L}(s, \varphi_2))$

Notice that $\mathcal{L}(s, \varphi_1 \wedge \varphi_2) = 0$ iff $\mathcal{L}(s, \varphi_1) = 0$ and $\mathcal{L}(s, \varphi_2) = 0$, and similarly $\mathcal{L}(\varphi_1 \vee \varphi_2) = 0$ iff $\mathcal{L}(\varphi_1) = 0$ or $\mathcal{L}(\varphi_2) = 0$.

The non-negative loss function $\mathcal{L}(s, \varphi)$ quantifies how much the state s violates the predicate φ. A larger loss indicates that the state is farther from the region defined by the predicate in the state space. The reward function for $\mathcal{T}_e$ is defined as:

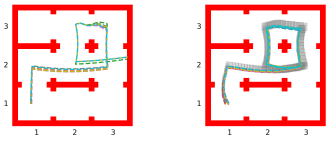
$$R(s, a) = c_1 \cdot \mathcal{L}(s, \mu(v)) + c_2 \cdot \mathcal{L}(s, \beta(e))$$

for any state $s \in S$ and action $a \in A$. Here, c_1 and c_2 are negative constants. We train $\pi_{\texttt{NN}}$ to maximize the discounted cumulative reward[2]:

$$\pi_{\texttt{NN}} = \arg\max_{\pi} \mathbb{E}_{s_0 \models \mu(u), s_0, a_0, s_1, \ldots \sim M[s_0, \pi]} \left[\sum_{t=0}^{T} \kappa^t R(s_t, a_t) \right] \tag{4}$$

where $\kappa \in [0, 1]$ is the discount factor. Once training converges, the horizon H for the subtask $\mathcal{T}_e$, which ensures a safe rollout from any state in $\mu(u)$ to a state in $\mu(v)$, can be estimated based on evaluation episodes using the policy $\pi_{\texttt{NN}}$.

Principally, we can verify $M[\pi_{\texttt{NN}}]$ against the specification $\mathcal{T}_e$ with rollout length H using Definition 6. However, reachability analysis for closed-loop systems controlled by neural networks remains a major challenge [25], and neural network policies trained merely from reward signals often fail to satisfy formal specifications. For example, in Fig. 6a, in the 9Rooms environment, a trained $\pi_{\texttt{NN}}$ agent for the edge policy from $\texttt{Room}_{2,2}$ to $\texttt{Room}_{3,2}$ fails to complete the loop and instead collides with the wall. To address this, following prior work [56,57], we distill $\pi_{\texttt{NN}}$ to a time-varying linear policy that is as similar as possible to $\pi_{\texttt{NN}}$. Importantly, this process ensures that the time-varying linear policy can be formally verified concerning the subtask $\mathcal{T}_e$. A time-varying linear policy can provide an accurate local approximation of a neural controller at each timestep (if the timestep is small) and incur a much-reduced verification cost owing to the linearity of the representation. A time-varying policy $\pi_\theta(s, t)$ with trainable parameters θ for a time horizon H $(0 \leq t < H)$ can be expressed mathematically as



(a) $\pi_{\texttt{NN}}$ trajectories (b) π_θ trajectories

Fig. 6: Policy behavior in 9Rooms. π_θ is a verified distillation of $\pi_{\texttt{NN}}$.

$$\pi_\theta(s, t) = \theta_w(t)^T \cdot s + \theta_b(t) \tag{5}$$

where $\theta_w(t)$ and $\theta_b(t)$ are the time-varying gain matrix and bias. During execution within the time horizon H, the policy $\pi_\theta(s, t)$ iteratively generates the control input at timestep t when observing the current state s at t. The objective of distilling $\pi_{\texttt{NN}}$ into a time-varying linear policy π_θ is

$$\begin{aligned} \theta^* = \arg\min_{\theta} \mathbb{E}_{s_0, s_1, \cdots, s_H \sim M[\pi_{\texttt{NN}}]} ||\pi_\theta(s_t, t) - \pi_{\texttt{NN}}(s_t)||_2 \\ \text{subject to } \textsc{Verify}(\pi_\theta, \mathcal{T}_e) \text{ is true} \end{aligned} \tag{6}$$

where $||\cdot||_2$ is L_2 norm and the Verify procedure returns true if and only if π_θ satisfies the subtask specification $\mathcal{T}_e$ according to Definition 6. In our implementation of the Verify procedure, per Definition 6, the abstract interpreter $F^{\mathcal{D}}$

[2] By constraining the predicate $\mu(u)$ on each vertex u to a verified interval of reachable states (line 17 in Algorithm 1), edge policy training for outgoing edges from u can uniformly sample initial states s_0 from this interval.

Algorithm 2 Train a verified edge policy π_θ for an abstract reachability graph edge $e : u \to v$ such that $\pi_\theta \models \mathcal{T}_e : \ \mu(u) \rightsquigarrow \mu(v) \mid \beta(e)$.

procedure TRAINVERIFY($M[\cdot], \mathcal{T}_e : \ \mu(u) \rightsquigarrow \mu(v) \mid \beta(e)$)
  Train a neural network controller $\pi_{\texttt{NN}}$ for $\mathcal{T}_e$ via Equation 4
  Estimate the horizon H for task $\mathcal{T}_e$ through sampling
  Initialize a time-varying linear policy π_θ over H timesteps via Equation 5
  Learn a *formally verified* π_θ for $\mathcal{T}_e$ that approximates $\pi_{\texttt{NN}}$ via Equation 6
  $S_0^{\mathcal{D}}, \ldots, S_H^{\mathcal{D}} \leftarrow$ REACHSET($\pi_\theta, \mu(u), H$) ▷ Definition 6
  return π_θ, H, $S_H^{\mathcal{D}}$

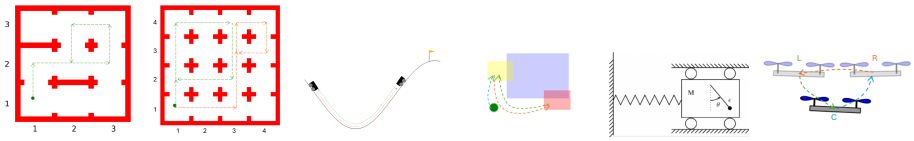
uses Taylor Model (TM) flowpipes as the abstract domain $\mathcal{D}$. For reachability analysis of $M[\pi_\theta]$, at each timestep t (where $t > 0$), we get the TM flowpipe $S_t^{\mathcal{D}}$ for the reachable set of states of $M[\pi_\theta]$ at timestep $t-1$. To obtain a TM representation for the output set of the time-varying linear policy π_θ at timestep t, we use TM arithmetic to evaluate a TM flowpipe $A_t^{\mathcal{D}}$ for $\pi_\theta(s,t) = \theta_w(t)^T \cdot s + \theta_b(t)$ for all states $s \in \gamma(S_t^{\mathcal{D}})$. The resulting TM representation $A_t^{\mathcal{D}}$ can be viewed as an overapproximation of the policy's output at timestep t. Finally, we construct the TM flowpipe overapproximation $S_{t+1}^{\mathcal{D}}$ for all reachable states at timestep t by reachability analysis over the state transition function $F^{\mathcal{D}}(S_t^{\mathcal{D}}, A_t^{\mathcal{D}})$. We use existing work [57] to solve the optimization task in Equation 6. Intuitively, this method employs Lagrangian optimization to integrate the verification constraint into the distillation objective, effectively minimizing both the L^2 loss for distillation and the violation of the verification constraint. The latter quantifies the violation of the safety or reachability property in the worst case across all concrete states subsumed by a symbolic state $S_t^{\mathcal{D}}$. For completeness, we provide a detailed explanation of this procedure based on [57] in the extended version [58]. For example, in Fig. 6b, the distilled policy π_θ from $\pi_{\texttt{NN}}$ can be formally verified to navigate the four rooms in an infinite circular sequence.

Corollary 1 (Algorithm 2 Soundness). *Given an environment model $M[\cdot]$, an edge policy specification $\mathcal{T}_e : \ \mu(u) \rightsquigarrow \mu(v) \mid \beta(e)$, if* TRAINVERIFY *returns $(\pi_\theta, H, S_H^{\mathcal{D}})$, we have $\pi_\theta \models \mathcal{T}_e$. On the symbolic rollout $\zeta^{\mathcal{D}} = S_0^{\mathcal{D}}, S_1^{\mathcal{D}}, \ldots, S_H^{\mathcal{D}}$ of $M[\pi_\theta]$, we have $\forall 0 \le t < H \wedge s \in \gamma_I(S_t^{\mathcal{D}}),\ s \models \beta(e)$, and $\forall s \in \gamma_I(S_H^{\mathcal{D}}),\ s \models \mu(v)$.*

4 Experiments

We provide an implementation of our framework, VEL-∞[3], and evaluate it on a set of challenging environments with continuous state and action spaces and infinite-horizon specifications. For neural network policy learning, we use Soft Actor-Critic (SAC) [16], a state-of-the-art deep RL algorithm. We implemented the abstract interpreter for verifying time-varying linear policies on top of Flow* [7], which uses Taylor-Model flowpipes as the abstract domain.

[3] VEL-∞ is available at `https://github.com/RU-Automated-Reasoning-Group/VEL-inf`.



(a) 9Rooms (b) 16Rooms (c) MountainCar (d) FlatWorld (e) Tora (f) QuadRotor

Ψ_1 = (Room$_{1,2}$; **repeat**(Room$_{3,2}$; Room$_{3,3}$; Room$_{2,3}$; Room$_{2,2}$)) **ensure** OutsideWalls

Ψ_2 = **repeat**(Room$_{1,2}$; Room$_{1,\ 4}$; Room$_{3,4}$; Room$_{3,2}$) **ensure** OutsideWalls **or**
(Room$_{3,1}$; **repeat**(Room$_{3,3}$; Room$_{3,4}$; Room$_{4,4}$; Room$_{4,3}$)) **ensure** OutsideWalls

Ψ_3 = **repeat**(**achieve** RightTop; **achieve** LeftTop)

Ψ_4 = **achieve** Yellow; **repeat**(**achieve** Yellow **ensure** Yellow)

Ψ_5 = **repeat**(**achieve** Yellow; **achieve** Red) **ensure** NotBlue

Ψ_6 = **achieve** Origin; **repeat** (**achieve** Origin **ensure** Origin)

Ψ_7 = **achieve** C; **repeat** (**achieve** C **ensure** C)

Ψ_8 = **repeat**(**achieve** R; **achieve** L; **achieve** C)

Fig. 7: Environments and Task Specifications. The predicates Room$_{x,y}$ and OutsideWalls are defined in Example 2. LeftTop and RightTop represent states at the left and right peaks of the mountain, respectively. The predicates Yellow, Red, and NotBlue correspond to regions that are yellow, red, and any color other than blue. Origin denotes the equilibrium state in the TORA environment. For the QuadRotor benchmark, L, R, and C specify three designated positions.

Baselines. We compare VEL-∞ with five baselines: DiRL [27], LCER [54], CyclER [46], TLTL [32], and BHNR [2]. Similar to our approach, DiRL uses the abstract reachability graph to decompose the specification and train the edge policies for each subtask. These edge policies are combined to address the full specification. However, DiRL does not provide any formal guarantee regarding the quality of generated controllers and it does not support infinite-horizon task specifications. In our experiments, we unroll repetitive specifications using the **repeat** operator five times for DiRL and reuse the edge policies as needed beyond this limit during evaluation. Another baseline, LCER, is a state-of-the-art RL algorithm for LTL specifications. LCER uses eventual discounting [54] to optimize a proxy value function that approximates the probability of satisfying a specified LTL formula and uses counterfactual experience replay to improve sample efficiency. We translate our specifications to LTL for LCER in our experiments. CyclER is a novel reward shaping technique that exploits the underlying structure of the LTL constraint to guide policy learning and combine it with quantitative semantics (QS) [32] in LTL reward shaping. TLTL and BHNR also use quantitative semantics for reward shaping and are computable for infinite-horizon LTL tasks.

Benchmarks. We use benchmarks considered in related works, visualized in Fig. 7. The 9Rooms environment and task specification Ψ_1, adapted from [61], were introduced in Example 2 and illustrated in Fig. 2. To explore more com-

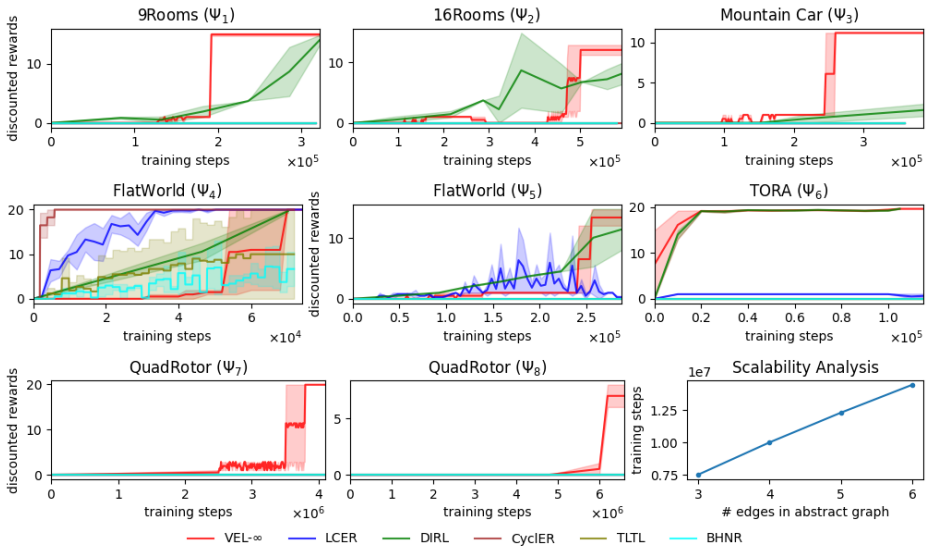


Fig. 8: Cumulative rewards under eventual discounting [54] *evaluated* over 5 seeds. A reward of 1 is given for each visit to final vertices or accepting states.

plex specifications, we designed a 16Rooms environment, where task Ψ_2 requires the agent to traverse a circular path indefinitely via one of two possible routes, represented by dashed arrows in Fig. 7b. The MountainCar environment, a classic nonlinear control problem from [35], follows task specification Ψ_3, where the agent must repeatedly drive back and forth in the valley. In the FlatWorld environment, adapted from [54], tasks Ψ_4 and Ψ_5 involve stabilizing in the yellow region and oscillating between the yellow and red regions, respectively, while avoiding the blue region. The TORA environment [23] simulates a cart connected to a wall by a spring, with the task specification Ψ_6 requiring stabilization around the origin infinitely. The QuadRotor benchmark, adapted from [60], tasks a simulated 2D quadrotor with stabilizing at the equilibrium state $[x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}] = 0$ in Ψ_7 and continuously navigating between three locations in a triangular pattern in Ψ_8. MountainCar, TORA and QuadRotor are nonlinear environments.

Results. Fig. 8 illustrates the learning performance throughout training for each benchmark, comparing VEL-∞ and the baselines. The x-axis shows environment steps, and the y-axis represents mean cumulative rewards under eventual discounting [54] for the intermediate policies evaluated. The shaded region indicates the standard deviation. VEL-∞ learned policies that are guaranteed correct for all these tasks. Computing the exact satisfaction probability (for the baseline models) over infinite horizons is an open problem. Eventually discounted return provides a proxy to the likelihood of task satisfaction [54]. It counts visits to final vertices in VEL-∞ and DiRL, and accepting states in the limit-deterministic Büchi automata of LTL formulas in the other baselines, assigning a reward of

1 per visit, with future visits discounted by $\gamma = 0.95$. We remark that this discounting schema is solely used for *evaluation* in VEL-∞. VEL-∞ demonstrates the highest sample efficiency across all benchmarks, except for FlatWorld (Ψ_4). Notably, in Fig. 8, VEL-∞'s training curve remains flat until the final edge policy is trained, after which its performance rapidly surpasses all other approaches to completely solve the tasks. DiRL is significantly less sample-efficient on our benchmarks, requiring an order of magnitude more samples during training compared to VEL-∞. Recall that repetitive specifications using the **repeat** operator are unrolled in DiRL. It must repeatedly learn new policies for similar tasks, whereas VEL-∞'s "train-and-verify" approach enables policy reuse with formal guarantees. LCER's low performance in our benchmarks may stem from its limited ability to utilize the structure of temporal logic properties during training. This challenge is particularly evident in the 9Rooms and 16Rooms environments, where the agent must accomplish several intermediate tasks before receiving any reward. Methods that do not account for task structures may find it harder to make steady progress in such settings. CyclER, TLTL, and BHNR show limited progress in our benchmarks, even though they utilize shaped rewards derived from the quantitative semantics of the given LTL formulas. These methods struggle with infinite-horizon tasks that involve multiple unordered subgoals, often leading to behaviors that optimize their respective QS-shaped LTL rewards without ensuring task completion. In contrast, VEL-∞ successfully achieves task satisfaction by explicitly decomposing temporal logic properties and synthesizing edge policies that guarantee correct composition for execution over infinite horizons. The strong performance of VEL-∞ in these benchmarks highlights the importance of synthesizing formally verified policies (with inductive invariants) within our framework. In the extended version of the paper [58], we report the average number of visits to final vertices (or accepting states) by trained policies at convergence for each benchmark. These results highlight that, unlike VEL-∞, the baselines lack formal correctness guarantees and often fail to produce policies that ensure infinite execution required by temporal reach-avoid specifications.

Scalability Analysis. We conduct a case study using QuadRotor to explore how VEL-∞ scales with increasing task specification complexity. We define specifications that require varying numbers of edge policies based on the QuadRotor configurations shown in Fig. 9. For a specification involving x edges in the repeat cycle, the quadrotor starts at state A, moves counterclockwise for $x-1$ hops, and returns to A in the final edge. For example, when $x = 4$, the task specification is: **repeat** (**achieve** A; **achieve** B; **achieve** C; **achieve** D). The bottom-right plot in Fig. 8 shows the training timesteps required for each specification for $x = 3, 4, 5, 6$ *resp.* Our results show that the number of environment steps re-

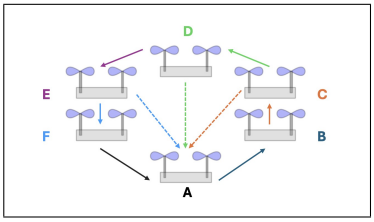


Fig. 9: Quadrotor configuration for the scalability case study.

quired for convergence by VEL-∞ increases linearly with the number of edge policies in the abstract reachability graph for a given specification.

5 Related Work

Verification in RL. RL has shown great potential in automatically acquiring new intelligent skills. However, verifying policies trained with RL algorithms to ensure their correctness and other desired properties is crucial before deployment, especially in safety-critical domains [17,48]. Some prior works, such as NNV, ReachNN*, and Verisig, have conducted reachability analysis to verify neural network-controlled systems [11,51,13,22,49]. However, these approaches verify the policies only after training is complete and do not use the verification results to improve the policy. Other works interleave policy training and verification by employing the idea of counterexample-guided abstraction and refinement or by optimizing for worst-case specification violation loss [24,56]. Policies learned together with reach-avoid supermartingales provide probabilistic guarantees of satisfying specifications [61,10]. In contrast, VEL-∞ supports synthesizing formally verified policies for temporal reach-avoid tasks.

RL for Temporal Logic Specifications. Linear Temporal Logic (LTL) has been widely adopted for specifying complex user behaviors [54]. A range of methods based on Q-learning have been developed to synthesize policies that satisfy LTL specifications, particularly in environments with discrete action spaces [4,47]. Other approaches aim to directly optimize the probability of satisfying LTL formulas [18,5,46,54,6], typically by guiding the agent toward accepting states in corresponding Büchi automata. To address the sparse reward issue that often arises in this setting, some of these methods incorporate experience replay to improve sample efficiency [54,55]. VEL-∞ takes a different approach by using abstract reachability graphs (Section 3.1) instead of Büchi automata for encoding specifications. Abstract reachability graphs provide a more structured and compositional representation at the subtask level, where each edge corresponds to a meaningful subtask from a precondition (source node) to a postcondition (target node). In contrast, Büchi automata operate at a lower level. For example, consider the LTL formula $F(\varphi_1 \wedge XF\varphi_2)$, which implies two sequential subtasks: first, reaching states that satisfy φ_1, and then proceeding to states that satisfy φ_2. In a Büchi automaton, the system often remains in the state corresponding to the satisfaction of φ_1 while attempting to reach φ_2, without explicitly encoding the completion of the first subtask or the initiation of the second. This makes it difficult to infer clear subtask boundaries and their associated preconditions and postconditions. VEL-∞ overcomes this limitation by organizing specifications through abstract reachability graphs, which naturally align with subtask-based reasoning and support more tractable verification and synthesis procedures. Truncated Linear Temporal Logic (TLTL) [32] and Bounded Horizon Nominal Robustness (BHNR) [2] use the quantitative semantics of temporal logic to define a reward function that encodes the intended agent behavior, enabling RL algorithms to learn policies from it [32,2]. Such reward

functions can be challenging for RL algorithms to maximize in complex, long-horizon task specifications. There exists work that instructs agents in multi-task settings to follow instructions in the LTL task space [53,40]. These methods do not provide formal guarantees for temporal property satisfaction.

Controller Synthesis for LTL objectives. Traditional controller synthesis algorithms, particularly those grounded in formal methods and temporal logic, often rely on automata-based approaches [12,14,15,28,30,36,38,52,59]. These techniques require abstraction and discretization [3,19,29,31,34,41,42,43,44], approximating continuous state and action spaces using finite-state models or grid-based representations. Once a discrete abstraction is constructed, standard synthesis algorithms for discrete systems can be applied. While such methods provide formal correctness guarantees, they face significant scalability challenges in high-dimensional or complex robotic systems due to issues like state explosion introduced by discretization. There are other LTL fragments, such as GR(1) [37] and GXW [8] that have been considered in the literature with favorable computational properties. Both GR(1) and GXW synthesis are formulated for discrete state and action spaces, typically in reactive synthesis for controllers. When applied to continuous systems, they require a finite abstraction step [59]. The LTL fragment VEL-∞ supports (for infinite-horizon temporal reach-avoid objectives) and GR(1)/GXW are incomparable. Based on RL, VEL-∞ is a controller learning algorithm that operates directly in continuous environments, avoiding the need for state discretization. This enables improved scalability in high-dimensional settings. Although RL methods generally lack formal guarantees of correctness, VEL-∞ addresses this limitation by incorporating verification mechanisms to formally certify the correctness of learned controllers.

6 Conclusion

We introduced VEL-∞, a deductive synthesis framework for RL agents that guarantees satisfaction of temporal reach-avoid properties over infinite horizons. VEL-∞ systematically decomposes these temporal objectives into finite subtasks while ensuring they can be composed and executed reliably over infinite time. Experimental results demonstrate that by combining the strengths of deductive reasoning with RL, VEL-∞ outperforms existing RL-based methods in both learning efficiency and adherence to formal specifications.

Limitations. VEL-∞ supports a limited fragment of LTL focused on temporal reach-avoid properties (Sec. 2), which can be naturally translated into abstract reachability graphs (Sec. 3.1) for efficient controller synthesis and verification. Extending VEL-∞ to handle general infinite-horizon LTL specifications remains an important direction for future work. Another main limitation of VEL-∞ is its reliance on an explicit environment model and sure property satisfaction. It requires a known model of the environment for reachability analysis, which may not scale to complex dynamics. Future work should focus on relaxing these assumptions by incorporating learned models and using probabilistic analysis to evaluate edge policies and their composition.

Disclosure of Interests

The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Alur, R.: Formal verification of hybrid systems. In: Chakraborty, S., Jerraya, A., Baruah, S.K., Fischmeister, S. (eds.) Proceedings of the 11th International Conference on Embedded Software, EMSOFT 2011, part of the Seventh Embedded Systems Week, ESWeek 2011, Taipei, Taiwan, October 9-14, 2011. pp. 273–278. ACM (2011)
2. Balakrishnan, A., Deshmukh, J.V.: Structured reward shaping using signal temporal logic specifications. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 3481–3486 (2019)
3. Belta, C., Sadraddini, S.: Formal methods for control synthesis: An optimization perspective. Annu. Rev. Control. Robotics Auton. Syst. **2**, 115–140 (2019)
4. Bozkurt, A.K., Wang, Y., Zavlanos, M.M., Pajic, M.: Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In: 2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020. pp. 10349–10355. IEEE (2020)
5. Cai, M., Hasanbeig, M., Xiao, S., Abate, A., Kan, Z.: Modular deep reinforcement learning for continuous motion planning with temporal logic. IEEE Robotics Autom. Lett. **6**(4), 7973–7980 (2021)
6. Camacho, A., Icarte, R.T., Klassen, T.Q., Valenzano, R., McIlraith, S.A.: Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19). pp. 6065–6073. International Joint Conferences on Artificial Intelligence Organization (2019)
7. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8044, pp. 258–263. Springer (2013)
8. Cheng, C., Hamza, Y., Ruess, H.: Structural synthesis for GXW specifications. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9779, pp. 95–117. Springer (2016)
9. Daafouz, J., Riedinger, P., Iung, C.: Stability analysis and control synthesis for switched systems: a switched lyapunov function approach. IEEE Trans. Autom. Control. **47**(11), 1883–1887 (2002)
10. Delgrange, F., Avni, G., Lukina, A., Schilling, C., Nowé, A., Pérez, G.A.: Synthesis of hierarchical controllers based on deep reinforcement learning policies. CoRR **abs/2402.13785** (2024)
11. Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: Ozay, N., Prabhakar, P. (eds.) Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019. pp. 157–168. ACM (2019)

12. Fainekos, G.E., Girard, A., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for dynamic robots. Autom. **45**(2), 343–352 (2009)
13. Fan, J., Huang, C., Chen, X., Li, W., Zhu, Q.: Reachnn*: A tool for reachability analysis of neural-network controlled systems. In: Hung, D.V., Sokolsky, O. (eds.) Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12302, pp. 537–542. Springer (2020)
14. Girard, A.: Controller synthesis for safety and reachability via approximate bisimulation. CoRR **abs/1010.4672** (2010)
15. Girard, A., Pappas, G.J.: Approximation metrics for discrete and continuous systems. IEEE Trans. Autom. Control. **52**(5), 782–798 (2007)
16. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor (2018), `https://arxiv.org/abs/1801.01290`
17. Hasanbeig, M., Kroening, D., Abate, A.: Towards verifiable and safe model-free reinforcement learning. vol. 2509. CEUR Workshop Proceedings (2020)
18. Hasanbeig, M., Kroening, D., Abate, A.: Deep reinforcement learning with temporal logics. In: Bertrand, N., Jansen, N. (eds.) Formal Modeling and Analysis of Timed Systems - 18th International Conference, FORMATS 2020, Vienna, Austria, September 1-3, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12288, pp. 1–22. Springer (2020)
19. Hsu, K., Majumdar, R., Mallik, K., Schmuck, A.: Lazy abstraction-based control for safety specifications. In: 57th IEEE Conference on Decision and Control, CDC 2018, Miami, FL, USA, December 17-19, 2018. pp. 4902–4907. IEEE (2018)
20. Icarte, R.T., Klassen, T.Q., Valenzano, R., McIlraith, S.A.: Reward machines: Exploiting reward function structure in reinforcement learning. Journal of Artificial Intelligence Research **73**, 173–208 (jan 2022)
21. Icarte, R.T., Klassen, T.Q., Valenzano, R.A., McIlraith, S.A.: Using reward machines for high-level task specification and decomposition in reinforcement learning. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 2112–2121. PMLR (2018)
22. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers. In: Ozay, N., Prabhakar, P. (eds.) Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019. pp. 169–178. ACM (2019)
23. Jankovic, M., Fontaine, D., Kokotovic, P.V.: Tora example: Cascade- and passivity-based control designs. IEEE Transactions on Control Systems Technology **4**(3), 292–297 (May 1996)
24. Jin, P., Tian, J., Zhi, D., Wen, X., Zhang, M.: Trainify: A cegar-driven training and verification framework for safe deep reinforcement learning. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification. pp. 193–218. Springer International Publishing, Cham (2022)
25. Johnson, T.T., Lopez, D.M., Benet, L., Forets, M., Guadalupe, S., Schilling, C., Ivanov, R., Carpenter, T.J., Weimer, J., Lee, I.: ARCH-COMP21 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In: Frehse, G., Althoff, M. (eds.) 8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21),

Brussels, Belgium, July 9, 2021. EPiC Series in Computing, vol. 80, pp. 90–119. EasyChair (2021)
26. Jothimurugan, K., Alur, R., Bastani, O.: A composable specification language for reinforcement learning tasks. Curran Associates Inc., Red Hook, NY, USA (2019)
27. Jothimurugan, K., Bansal, S., Bastani, O., Alur, R.: Compositional reinforcement learning from logical specifications. In: Proceedings of the 35th International Conference on Neural Information Processing Systems. NIPS '21, Curran Associates Inc., Red Hook, NY, USA (2024)
28. Jr., M.M., Davitian, A., Tabuada, P.: PESSOA: A tool for embedded controller synthesis. In: Touili, T., Cook, B., Jackson, P.B. (eds.) Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6174, pp. 566–569. Springer (2010)
29. Kim, E.S., Arcak, M., Seshia, S.A.: Symbolic control design for monotone systems with directed specifications. Autom. **83**, 10–19 (2017)
30. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal-logic-based reactive mission and motion planning. IEEE Trans. Robotics **25**(6), 1370–1381 (2009)
31. Kurtz, V., Lin, H.: Temporal logic motion planning with convex optimization via graphs of convex sets. IEEE Trans. Robotics **39**(5), 3791–3804 (2023)
32. Li, X., Vasile, C.I., Belta, C.: Reinforcement learning with temporal logic rewards. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 3834–3839 (2017)
33. Mania, H., Guy, A., Recht, B.: Simple random search provides a competitive approach to reinforcement learning (2018), `https://arxiv.org/abs/1803.07055`
34. Meyer, P., Dimarogonas, D.V.: Hierarchical decomposition of LTL synthesis problem for nonlinear control systems. IEEE Trans. Autom. Control. **64**(11), 4676–4683 (2019)
35. Moore, A.W.: Efficient memory-based learning for robot control. Tech. rep., University of Cambridge (1990)
36. Mouelhi, S., Girard, A., Gößler, G.: Cosyma: a tool for controller synthesis using multi-scale abstractions. In: Belta, C., Ivancic, F. (eds.) Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA. pp. 83–88. ACM (2013)
37. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: Emerson, E.A., Namjoshi, K.S. (eds.) Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006, Proceedings. Lecture Notes in Computer Science, vol. 3855, pp. 364–380. Springer (2006)
38. Pola, G., Girard, A., Tabuada, P.: Approximately bisimilar symbolic models for nonlinear control systems. Autom. **44**(10), 2508–2516 (2008)
39. Prajna, S., Jadbabaie, A.: Safety verification of hybrid systems using barrier certificates. In: Alur, R., Pappas, G.J. (eds.) Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2993, pp. 477–492. Springer (2004)
40. Qiu, W., Mao, W., Zhu, H.: Instructing goal-conditioned reinforcement learning agents with temporal logic objectives. In: Thirty-seventh Conference on Neural Information Processing Systems (2023)
41. Reissig, G., Weber, A., Rungger, M.: Feedback refinement relations for the synthesis of symbolic controllers. IEEE Trans. Autom. Control. **62**(4), 1781–1796 (2017)

42. Ren, W., Dimarogonas, D.V.: Logarithmic quantization based symbolic abstractions for nonlinear control systems. In: 17th European Control Conference, ECC 2019, Naples, Italy, June 25-28, 2019. pp. 1312–1317. IEEE (2019)
43. Ren, W., Jungers, R.M., Dimarogonas, D.V.: Zonotope-based symbolic controller synthesis for linear temporal logic specifications. IEEE Trans. Autom. Control. **69**(11), 7630–7645 (2024)
44. Rungger, M., Zamani, M.: SCOTS: A tool for the synthesis of symbolic controllers. In: Abate, A., Fainekos, G. (eds.) Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, Vienna, Austria, April 12-14, 2016. pp. 99–104. ACM (2016)
45. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017), https://arxiv.org/abs/1707.06347
46. Shah, A., Voloshin, C., Yang, C., Verma, A., Chaudhuri, S., Seshia, S.A.: Ltl-constrained policy optimization with cycle experience replay (2024), https://arxiv.org/abs/2404.11578
47. Shao, D., Kwiatkowska, M.: Sample efficient model-free reinforcement learning from LTL specifications with optimality guarantees. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China. pp. 4180–4189. ijcai.org (2023)
48. Srinivasan, K., Eysenbach, B., Ha, S., Tan, J., Finn, C.: Learning to be safe: Deep rl with a safety critic (2020), https://arxiv.org/abs/2010.14603
49. Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: Ozay, N., Prabhakar, P. (eds.) Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019. pp. 147–156. ACM (2019)
50. Tedrake, R., Manchester, I.R., Tobenkin, M.M., Roberts, J.W.: Lqr-trees: Feedback motion planning via sums-of-squares verification. Int. J. Robotics Res. **29**(8), 1038–1052 (2010)
51. Tran, H., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12224, pp. 3–17. Springer (2020)
52. Tumova, J., Yordanov, B., Belta, C., Cerna, I., Barnat, J.: A symbolic approach to controlling piecewise affine systems. In: Proceedings of the 49th IEEE Conference on Decision and Control, CDC 2010, December 15-17, 2010, Atlanta, Georgia, USA. pp. 4230–4235. IEEE (2010)
53. Vaezipoor, P., Li, A.C., Icarte, R.T., McIlraith, S.A.: Ltl2action: Generalizing ltl instructions for multi-task rl. In: Proceedings of the 38th International Conference on Machine Learning (ICML). vol. 139, pp. 10497–10508. Proceedings of Machine Learning Research (2021)
54. Voloshin, C., Verma, A., Yue, Y.: Eventual discounting temporal logic counterfactual experience replay. In: Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., Scarlett, J. (eds.) International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA. Proceedings of Machine Learning Research, vol. 202, pp. 35137–35150. PMLR (2023)
55. Wang, C., Li, Y., Smith, S.L., Liu, J.: Continuous motion planning with temporal logic specifications using deep neural networks (2020), https://arxiv.org/abs/2004.02610

56. Wang, Y., Zhu, H.: Verification-guided programmatic controller synthesis. In: Sankaranarayanan, S., Sharygina, N. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 229–250. Springer Nature Switzerland, Cham (2023)
57. Wang, Y., Zhu, H.: Safe exploration in reinforcement learning by reachability analysis over learned models. In: Gurfinkel, A., Ganesh, V. (eds.) Computer Aided Verification. pp. 232–255. Springer Nature Switzerland, Cham (2024)
58. Wang, Y., Zhu, H.: Deductive synthesis of reinforcement learning agents for ω-regular properties (extended version) (2025), `https://github.com/RU-Automated-Reasoning-Group/VEL-inf/blob/main/VEL-inf_extended.pdf`
59. Wongpiromsarn, T., Topcu, U., Ozay, N., Xu, H., Murray, R.M.: Tulip: a software toolbox for receding horizon temporal logic planning. In: Caccamo, M., Frazzoli, E., Grosu, R. (eds.) Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2011, Chicago, IL, USA, April 12-14, 2011. pp. 313–314. ACM (2011)
60. Yang, L., Dai, H., Shi, Z., Hsieh, C.J., Tedrake, R., Zhang, H.: Lyapunov-stable neural control for state and output feedback: a novel formulation. In: Proceedings of the 41st International Conference on Machine Learning. ICML'24, JMLR.org (2024)
61. Žikelić, Đ., Lechner, M., Verma, A., Chatterjee, K., Henzinger, T.A.: Compositional policy learning in stochastic control systems with formal guarantees. In: Thirty-seventh Conference on Neural Information Processing Systems (2023)