LTL-Constrained Policy Optimization with Cycle Experience Replay

Anonymous authors
Paper under double-blind review

Abstract

Linear Temporal Logic (LTL) offers a precise means for constraining the behavior of reinforcement learning agents. However, in many settings, LTL is insufficient for task specification; LTL-constrained policy optimization, where the goal is to optimize a scalar reward under LTL constraints, is needed. Prior methods for this constrained problem are restricted to finite state spaces, limiting its applicability in deep Reinforcement Learning (DRL) settings. In this work, we present Cycle Experience Replay (CyclER), a reward-shaping approach to this problem that succeeds in using DRL to learn performant policies in continuous state and action spaces. CyclER guides a policy towards satisfaction by encouraging partial behaviors compliant with the LTL constraint, using the structure of the constraint. In doing so, it addresses the optimization challenges stemming from the sparse nature of LTL satisfaction. We evaluate CyclER in three continuous control domains. On these tasks, CyclER outperforms existing reward-shaping methods at finding effective and LTL-satisfying policies.

1 Introduction

Significant research effort has explored *Linear Temporal Logic* (LTL) as an alternative means of specifying objectives for reinforcement learning (RL) agents (Sadigh et al., 2014; Hasanbeig et al., 2018; Camacho et al., 2019; Wang et al., 2020; Vaezipoor et al., 2021; Alur et al., 2022; De Giacomo et al., 2020; Voloshin et al., 2023). LTL provides a flexible language for defining objectives, or *specifications*, that are often not reducible to scalar Markovian rewards (Abel et al., 2021). Unlike typical reward functions, objectives defined in LTL are composable, easily transferred across environments, and offer a precise notion of satisfaction.

LTL specifications and Markovian reward functions have been used separately in a variety of RL settings, but few works consider both rewards *and* specifications in the same setting. The combination of the two is important: an LTL specification can define the meaning of achieving a task, and a reward function can be optimized to find the best way of achieving that task. For example, in robot motion planning, an LTL specification can describe the waypoints a robot should reach and obstacles it should avoid, and a reward function can optimize for factors like energy consumption, stability of motion, and so forth.

This work considers the problem setting of RL-based reward optimization under an LTL constraint. Previous works that solve LTL satisfaction in a reward-maximizing setting propose planning-based solutions that are limited to discrete state spaces (Voloshin et al., 2022). To the best of our knowledge, our work is the first to approach this problem with Deep RL (DRL) to scale to continuous state and action spaces.

Our learning problem can be naturally formulated in an unconstrained form through a Lagrange-style relaxation (Le et al., 2019; Achiam et al., 2017) where the LTL constraint is represented by a *proxy* reward function. Some versions of such reward-shaping have been studied in the literature (Hasanbeig et al., 2020; Voloshin et al., 2023; Camacho et al., 2019). However, these methods produce proxy rewards that are sparse and hard to optimize. Due to this sparsity, learned policies in practice often end up ignoring the LTL constraint entirely and focus only on optimizing the reward function.

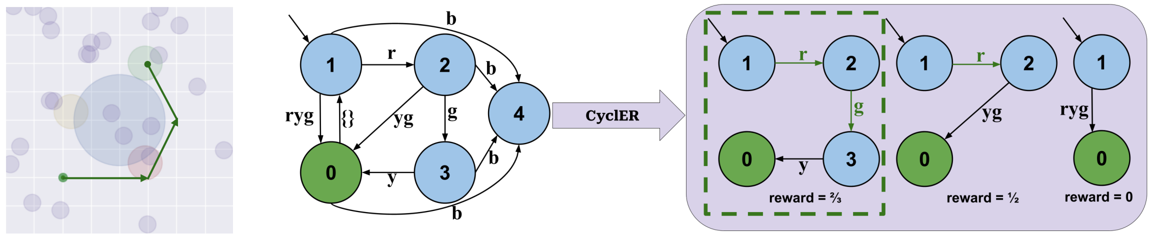

Figure 1: Left: The FlatWorld MDP and an example trajectory. Right: An LDBA for the LTL formula $\varphi = G(F(r)\&F(g)\&F(y))\&G(\neg b)$ (some edges omitted for readability); the accepting state 0 is coded in green. The CyclER method considers all accepting paths within an LDBA and selects the most reward-ful path for the trajectory to shape LTL reward. Unlike other approaches, CyclER offers dense reward, even without visiting the accepting state.

In this paper, we address this reward-sparsity issue by introducing a novel reward-shaping proxy for LTL called *Cycle Experience Replay* (CyclER). CyclER encourages partial behavior compliant with the specification by exploiting the underlying automaton structure of LTL. Briefly, given the (known) automaton representing an LTL objective, CyclER computes all possible infinite paths, or cycles, through the automaton that define accepting behavior for our task. When an agent collects a finite episode of experience, CyclER counterfactually reasons over the episode by considering how much progress it made through each cycle. The cycle that the agent progressed through the furthest is then used to shape LTL reward. Under certain assumptions, CyclER maintains theoretical guarantees on LTL optimality that are competitive with state-of-the-art LTL proxy rewards (Voloshin et al., 2023). A key advantage of CyclER is how it readily incorporates quantitative semantics (QS), a popular technique for reward design in temporal logic that has yet to be extended to infinite-horizon LTL tasks. Our empirical results demonstrate CyclER's effectiveness during policy learning.

To summarize, the paper makes the following contributions. We present the first problem formulation for LTL-constrained policy optimization in the presence of continuous spaces and function approximators. We propose a technique for this problem setting, CyclER, that alleviates the proxy reward sparsity issue, and provides guarantees that CyclER reward-shaping will ensure approximate optimality of LTL satisfaction. Third, we introduce a new way of using quantitative semantics in reward shaping for LTL. Lastly, we present promising experimental results using CyclER in LTL-constrained optimization settings, outperforming existing approaches.

2 Problem Setting

2.1 Preliminaries

Linear Temporal Logic (LTL) Linear Temporal Logic (Pnueli, 1977) is a specification language that composes atomic propositions with logical and temporal operators to precisely define tasks. An atomic proposition is a variable that takes on a Boolean truth value. We define an *alphabet* Σ as all possible combinations over a finite set of atomic propositions (AP); that is, $\Sigma = 2^{\text{AP}}$. For example, if $\text{AP} = \{a, b\}$, then $\Sigma = \{\{a, b\}, \{b\}, \{a\}, \{\}\}$. We will refer to individual combinations of atomic propositions, or predicates, in Σ as ν. We use the symbol φ to refer to an LTL task specification, also called an LTL formula.

In LTL, specifications are constructed using both logical connectives: not ($\neg$), and ($\&$), and implies ($\rightarrow$); and temporal operators: next (X), repeatedly/always/globally (G), eventually (F), and until (U). For more detail on the exact semantics of LTL operators, see Baier & Katoen (2008).

As an example, consider the "FlatWorld" environment in Figure 1 (left), where $\text{AP} = \{r, g, b, y\}$, corresponding to whether the agent is in the red, green, blue, or yellow region at any point in time. LTL can easily define some simple objectives, such as safety $G(\neg b)$, reachability $F(g)$, or progress $F(y)\&X(F(r))$. We can also combine operators to bring together these objectives into more complex specifications, such as $G(F(r)\&F(y)\&F(g))\&G(\neg b)$, which instructs an agent to oscillate amongst the red, yellow, and green regions indefinitely while avoiding the blue region.

In order to determine the logical satisfaction of an LTL specification, we can transform it into a specialized automaton called a *Limit Deterministic Büchi Automaton (LDBA)*. See Sickert et al. (2016); Hahn et al. (2013); Křetínskỳ et al. (2018) for details on how LTL specifications can be transformed into semantically equivalent LDBA.

More precisely, a (de-generalized) LDBA is a tuple $\mathbb{B} = (\mathcal{B}, \Sigma, T^{\mathbb{B}}, \mathcal{B}^*, \mathcal{E}, b_0)$ with a set of states $\mathcal{B}$, the alphabet Σ of predicates ν that defines deterministic transitions in the automaton, a transition function $T^{\mathbb{B}} : \mathcal{B} \times (\Sigma \cup \mathcal{E}) \to \mathcal{B}$, a set of accepting states $\mathcal{B}^*$, and an initial state b_{-1}. An LDBA may have separate deterministic and nondeterministic components $\mathcal{B} = \mathcal{B}_D \cup \mathcal{B}_N$, such that $\mathcal{B}^* \subseteq \mathcal{B}_D$, and for $b \in \mathcal{B}_D$, $x \in \Sigma$ then $T^{\mathbb{B}}(b, x) \subseteq \mathcal{B}_D$. $\mathcal{E}$ is a set of "jump" actions, also known as epsilon-transitions, for $b \in \mathcal{B}_N$ that transitions to $\mathcal{B}_D$ without evaluating any atomic propositions. A path $\xi = (b_0, b_1, \dots)$ is a sequence of states in $\mathcal{B}$ reached through successive transitions under $T^{\mathbb{B}}$.

Definition 2.1 (Acceptance of ξ)**.** We accept a path $\xi = (b_0, b_1, \dots)$ if an accepting state of the Büchi automaton is visited infinitely often by ξ.

Labeled MDPs We formulate our environment as a labelled Markov Decision Process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T^{\mathcal{M}}, d_0, \gamma, r, L^{\mathcal{M}})$, containing a state space $\mathcal{S}$, an action space $\mathcal{A}$, an *unknown* transition function, $T^{\mathcal{M}} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, an initial state distribution $d_0 \in \Delta(\mathcal{S})$, a discount factor $0 < \gamma < 1$, a reward function $r : \mathcal{S} \times \mathcal{A} \to [R_{\min}, R_{\max}]$, and a labelling function $L^{\mathcal{M}} : \mathcal{S} \to \Sigma$. The labelling function returns which atomic propositions in our set AP are true for a given MDP state.

2.2 Problem Statement

We would like to learn a policy that produces satisfactory (accepting) trajectories with respect to a given LTL formula φ while maximizing r, the reward function from the MDP. Before we define our formal problem statement, we introduce more notation:

Definition 2.2 (Product MDP)**.** A **product MDP** synchronizes the MDP with an LDBA. Specifically, let $\mathcal{M}^{\varphi}$ be an MDP with state space $\mathcal{S}^{\varphi} = \mathcal{S} \times \mathcal{B}$. Policies over our product MDP space can be defined as $\pi : \mathcal{S}^{\varphi} \to \Delta(\mathcal{A}^{\varphi})$, where our new set of actions combine $\mathcal{A}^{\varphi}((s, b)) = \mathcal{A}(s) \cup \mathcal{E}$, to include the jump transitions in $\mathbb{B}$ as possible actions. We define the space of all possible policies as Π. The new probabilistic transition relation of our product MDP is defined as:

$$T(s, b, a, s', b') = \begin{cases} T^{\mathcal{M}}(s, a, s') & a \in \mathcal{A}(s), b' = T^{\mathbb{B}}(b, L(s')) \\ 1 & a \in \mathcal{E}, b' = T^{\mathbb{B}}(b, a), s = s' \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

A policy generates trajectories $\tau = ((s_0, b_0, a_0), (s_1, b_1, a_1), \dots)$ in the product MDP. Define $\mathcal{R}(\tau) \equiv \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$ as the total reward along a trajectory τ.

Definition 2.3 (Trajectory acceptance)**.** A trajectory is said to be **accepting** with respect to φ ($\tau \models \varphi$, or "φ accepts τ") if there exists some $b \in \mathcal{B}^*$ that is visited infinitely often.

Definition 2.4 (Policy satisfaction)**.** A policy $\pi \in \Pi$ *satisfies* φ with some probability $\mathbb{P}[\pi \models \varphi] = \mathbb{E}_{\tau \sim \mathcal{M}^{\varphi}_{\pi}}[\mathbf{1}_{\tau \models \varphi}]$. Here, $\mathbf{1}$ is an indicator variable that checks whether or not a trajectory τ is accepted by φ, and $\mathcal{M}^{\varphi}_{\pi}$ is the distribution of trajectories induced by policy π in a product MDP $\mathcal{M}^{\varphi}$.

Definition 2.5 (Probability-optimal policies)**.** We will denote Π^* as the set of policies that maximize the probability of satisfaction with respect to φ; that is, the policies that have the highest probability of producing an accepted trajectory: $\Pi^* = \{\pi \in \Pi | \mathbb{P}[\pi \models \varphi] = \max_{\pi' \in \Pi} \mathbb{P}[\pi' \models \varphi]\}$.

Our aim is to find a policy in the probability-optimal set Π^* that collects the largest expected cumulative discounted reward. We state this constrained objective formally as follows:

$$\pi^* \in \text{argmax}_{\pi \in \Pi^*} \mathbb{E}_{\tau \sim \mathcal{M}^{\varphi}_{\pi}} \left[\mathcal{R}(\tau) \right] \tag{2}$$

For notational convenience, we will refer to the MDP value function as $\mathcal{R}_\pi \equiv \mathbb{E}_{\tau\sim\mathcal{M}_\pi^\varphi}\big[\mathcal{R}(\tau)\big]$.

In certain cases, the probability-optimal set of policies Π^* may be empty; consequently, a solution to 2 may not exist. In section 3, we introduce a proxy objective with a similar potential of non-existence and discuss how our ultimate optimization objective behaves in this setting.

To align with our intended applications towards deep RL, we consider stochastic, memoryless policies over the product MDP. Such policies are capable of capturing probability-optimal policies for a given LTL specification if an optimal policy exists (Bozkurt et al., 2020; Voloshin et al., 2022).

3 LTL-Constrained Policy Optimization

Finding a policy within Π^* is, in general, not tractable: an LTL constraint φ is defined over infinite-length trajectories but policy rollouts in practice produce only finite-length trajectories (Yang et al., 2022). We adopt *eventual discounting* (Voloshin et al., 2023), a common approach in the existing literature which aims to optimize a proxy value function that approximates the satisfaction of φ. Eventual discounting is defined as:

$$V_\pi = \mathop{\mathbb{E}}_{\tau\sim\mathcal{M}_\pi^\varphi}\Big[\sum_{t=0}^{\infty}\Gamma_t r_{\text{LTL}}(b_t)\Big], \qquad \Gamma_t = \gamma_\varphi^j, \quad r_{\text{LTL}}(b_t) = \begin{cases} 1 & \text{if } (b_t \in \mathcal{B}^*) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $j = \sum_{k=0}^{t} r_{\text{LTL}}(b_k)$ counts how many times the set $\mathbb{B}^*$ has been visited (up to and including the current timestep). Notably, eventual discounting does not discount based on the amount of time between visits to an accepting state. A policy that maximizes this eventual discounting reward is approximately probability-optimal with respect to φ when γ_φ, the discounting factor associated with φ, is selected properly (see Theorem 4.2 in Voloshin et al. (2023) for an exact bound).

As a result of eventual discounting, we can replace Π^* in objective 2 with the set of policies that maximize V_π. Let $V_{\max} = \max_{\pi\in\Pi} V_\pi$ be the maximal value.

$$\pi^* = \mathop{\text{argmax}}_{\pi\in\{\pi\in\Pi \mid V_\pi = V_{\max}\}}\big[\mathcal{R}_\pi\big] \tag{4}$$

We now form the Lagrangian dual of objective 4 as $\pi^* = \min_\lambda \text{argmax}_{\pi\in\Pi}\big[\mathcal{R}_\pi + \lambda(V_\pi - V_{\max})\big]$. In theorem 3.2 we show that because we only care about constraint-maximizing policies, there exists $\lambda^* \in \mathbb{R}$ such that solving the inner maximization of the Lagrangian dual must be constraint optimal for any fixed $\lambda > \lambda^*$. Intuitively, the higher λ is, the more our learned policy will account for V_π during optimization until the constraint must be satisfied. At that point, because we are already achieving the maximum possible V_π, any additional lift will only come from maximizing over the MDP value $\mathcal{R}$, even if we continue to increase λ. With this observation, we can form an unconstrained objective function from objective 4 to be the following:

$$\pi^* = \arg\max_{\pi\in\Pi}\big[\mathcal{R}_\pi + \lambda V_\pi\big] \tag{5}$$

where we have dropped the dependence on $V_{\max}$ since it is a constant and fixed $\lambda > \lambda^*$. We show that under certain assumptions, an exact value for λ^* can be found to ensure that a policy that maximizes eq. 5 will certainly maximize V_π.

Assumption 3.1. *There exists a positive nonzero gap $\epsilon > 0$ between the value V_π of policies in $\pi \in \Pi^*$ and the highest-value policies that are not; that is, $V_{\max} - \max_{\pi\in(\Pi\setminus\Pi^*)}(V_\pi) > \epsilon$.*

Theorem 3.2. *Under Assumption 3.1, for any choice of $\lambda > \frac{\mathcal{R}_{\max}-\mathcal{R}_{\min}}{\epsilon(1-\gamma)}$, the solution to objective 5 must be a solution to objective 4. See Appendix Section B for the proof.*

We note that Assumption 3.1 can be found in previous literature (Voloshin et al., 2023) and serves as a sufficient but not necessary condition for our results. We provide further analysis for the existence of Assumption 3.1 in Appendix Section B.1.

As briefly mentioned in Section 2, the probability-optimal set of policies with respect to φ may be empty. The same is true for our updated definition of Π^* that contain policies that achieve $V_{\max}$. In the case of this non-existence, Assumption 3.1 does not hold, and a policy that optimizes 5 will prioritize improving V_π at the potential expense of $\mathcal{R}_\pi$. We provide an extended discussion of this consequence in Section B.2.

Empirical Considerations. Since the conditions for Assumption 3.1 are often unknown, there may not be a verifiable way to know that that our learned policy is maximizing V_π. Because of this, we will treat λ as a tunable hyperparameter that allows a user to trade off the relative importance of empirically satisfying the LTL constraint. There are a number of strategies one can use to find an appropriate λ: for example, one can iteratively increase λ until a desired LTL reward is achieved. In our experiments, we show an example of this trade off, and notice that the trade off lessens in severity once λ exceeds a value that enables learning LTL-satisfying policies (table 2).

4 Cycle Experience Replay (CyclER)

To distinguish between the MDP's reward function and the eventual-discounting proxy reward in 3, we write the MDP reward function $r(s,a)$ as $r_{\text{MDP}}(s,a)$. In Deep RL settings, we maximize objective 5 using the reward function $r_{\text{DUAL}}(s_t, b_t, a_t) = \gamma^t r_{\text{MDP}}(s_t, a_t) + \Gamma_t \lambda r_{\text{LTL}}(b_t)$.

However, optimizing objective 5 is challenging due to the sparsity of r_{LTL}. r_{LTL} is nonzero only when an accepting state in $\mathbb{B}$ is visited, which may require a long, precise sequence of actions.

Consider the FlatWorld MDP and LDBA in Figure 1. The MDP's reward function incentivizes visiting the small purple regions in the world. Under r_{LTL}, a policy will receive no reward until it completes the entire task of avoiding blue and visiting the red, yellow, and green regions through random exploration. If r_{MDP} is dense, a policy may fall into an unsatisfactory 'local optimum' by optimizing for r_{MDP} it receives early during learning, and ignore r_{LTL} entirely. In Figure 2, we see that a policy trained on r_{DUAL} makes such an error.

We seek to address this shortcoming by *automatically* shaping r_{LTL} so that a more dense reward for φ is available during training. Below, we present our approach, which exploits the known structure of the LDBA $\mathbb{B}$ and cycles within $\mathbb{B}$ that visit accepting states.

4.1 Rewarding Accepting Cycles in $\mathbb{B}$

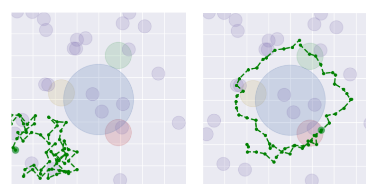

Figure 2: Trajectories from unshaped r_{DUAL} (left) and CyclER r_{DUAL} (right) for the formula $G(F(r)\&F(g)\&F(y))$ & $G(\neg b)$.

By definition of LTL satisfaction (def. 2.3), a trajectory must repeatedly visit an accepting state b^* in an LDBA. In the context of the automaton itself, that means that an accepting trajectory will traverse an *accepting path* from the initial state to an accepting state, and then repeatedly traverse *accepting cycles* within $\mathbb{B}$ that continually visit accepting states.

Definition 4.1 (Accepting Initial Path (AIP))**.** An accepting initial path in $\mathbb{B}$ is a set of valid transitions (b_i, ν, b_j) (i.e., the predicate ν that transitions b_i to b_j) in $\mathbb{B}$ that starts at the initial state b_0 and ends at an accepting state $b_k^* \in \mathcal{B}^*$.

Definition 4.2 (Accepting Cycle (AC))**.** An accepting cycle in $\mathbb{B}$ is a set of valid transitions (b_i, ν, b_j) in $\mathbb{B}$ that start and end at accepting states $b_k^*, b_l^* \in \mathcal{B}^*$.[1]

Our key insight is that we can use accepting paths and cycles in $\mathbb{B}$ to shape r_{LTL}. Instead of only providing reward when an accepting state in $\mathbb{B}$ is visited (as per previous approaches e.g. Voloshin et al. (2023)), we reward progress within an accepting path or cycle. In our example from fig 1, if we reward each transition in the initial path $\{1, 2, 3, 0\}$ and the cycle with the same states, the agent would receive rewards for visiting the red region, then yellow, then green, then for returning to red, and so on.

Multiple accepting paths and cycles may exist in $\mathbb{B}$. The path and cycle that is used to shape r_{LTL} cannot be picked arbitrarily, since they may be infeasible under the dynamics of the MDP. For example, the cycle

[1] Our usage of the word "cycle" is not a cycle in the traditional sense of graph search, but instead refers to paths that connect two accepting states in $\mathbb{B}$ (allowing for "cyclical" acceptance).

$\{1, 2, 0\}$ in Figure 1 cannot effectively shape r_{LTL} because it is impossible to be both in the yellow and green regions at the same time.

4.2 Reward Shaping with CyclER

CyclER is a reward function that automatically selects paths and cycles to shape r_{LTL} based on collected experience.

Definition 4.3 (Minimal AIP (MAIP))**.** A minimal accepting initial path for accepting state b_k^* is an AIP that does not contain a subcycle for any node b_i in the path where $b_i \neq b_k^*$.

Definition 4.4 (Minimal AC (MAC))**.** A minimal accepting cycle c for accepting states b_k^* and b_l^* is an AC that does not contain a subcycle for any node b_i in the cycle where $b_i \notin \{b_k^*, b_l^*\}$.

We provide CyclER with all MAIPs and MACs in an LDBA using Depth-First Search with backtracking (see Appendix Algs. 2 and 3). Let $\mathcal{P}$ and $\mathcal{C}$ be the set of MAIPs and MACs, respectively.

We also maintain a frontier e of visited transitions in $\mathbb{B}$ at each timestep in a trajectory that ensures reward will only be given once per transition until an accepting state is visited. In particular, we set $e[(b_i, \nu, b_i)] = 1$ when a transition (b_i, ν, b_i) is taken and reset all $e \equiv 0$ when $b_j \in \mathcal{B}^*$. Policies that use CyclER observe s, b, and e as their current state.

Now we describe the CyclER reward computation. We first collect a complete trajectory τ induced by $\mathcal{M}_\pi^\varphi$ for a given policy π. Then, at each timestep t from 0 to $|\tau| - 1$, we compute $r_{\mathcal{C}}$ for every path in $\mathcal{P}$ if we have not yet visited an accepting state, or every cycle in $\mathcal{C}$ if we have. We will abuse notation slightly and use c to refer to elements in either $\mathcal{P}$ or $\mathcal{C}$:

$$r_{\mathcal{C}}(b_t, s_{t+1}, b_{t+1}, e, c) = \begin{cases} \frac{1}{|c|} & \text{if } (b, L^{\mathcal{M}}(s_{t+1}), b_{t+1}) \in c \ \text{ and } e[b, L^{\mathcal{M}}(s_{t+1}), b_{t+1}] = 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

This function rewards every transition taken in a given c. In other words, when an agent "gets closer" to an accepting state by progressing along a path or cycle, we reward that progress once per visit to an accepting state. To account for c of varying length, rewards are normalized by the length $|c|$.

Algorithm 1: Cycle Experience Replay (**CyclER**)

Input: Trajectory τ, $\mathcal{B}^*$, cycles $\mathcal{C}$, paths $\mathcal{P}$
Initialize matrix $R_{\mathcal{C}}$ size $\max(|\mathcal{C}|, |\mathcal{P}|) \times (|\tau| - 1)$;
Initialize r_{CyclER} to an array of size $(|\tau| - 1)$;
Initialize $j = 0$;
foreach $t = 0, \ldots, |\tau| - 1$ **do**
  if $j = 0$ **then**
    foreach *Path* $p_i \in \mathcal{P}$ **do**
      $R_{\mathcal{C}}[i, t] = r_{\mathcal{C}}(b_t, s_{t+1}, b_{t+1}, e_t, p_i)$
  else
    foreach *Cycle* $c_i \in \mathcal{C}$ **do**
      $R_{\mathcal{C}}[i, t] = r_{\mathcal{C}}(b_t, s_{t+1}, b_{t+1}, e_t, c_i)$
  if $b_{t+1} \in \mathcal{B}^*$ *or* $t + 1 = |\tau|$ **then**
    Select $i = \text{argmax}_{i \in |C|}(\sum_{j'=j}^{t} R_{\mathcal{C}}[i, j'])$;
    foreach t' *from* j *to* $t + 1$ **do**
      $r_{\text{CyclER}}[t'] = R_{\mathcal{C}}[i, t']$
    $j = t + 1$;
return r_{CyclER};

If we visit an accepting state b^* or reach the end of a trajectory, we retroactively assign rewards to the timesteps that preceded this point, up to the most recent accepting state visit (if one exists). Assigned rewards correspond to the cycle with the highest total reward for that partial trajectory. Put simply, CyclER picks the 'best' cycle for a partial trajectory and uses it to shape reward. Even if a trajectory does not manage to visit an accepting state, CyclER will still provide reward if it was able to take *any* transition along *any* MAIP. The specifics are given in Algorithm 1.

We denote the rewards returned from Alg. 1 as r_{CyclER}. r_{CyclER} can be used in place of the unshaped r_{LTL} in function 3 to provide a more dense reward for τ.

Theorem 4.1 (Informal)**.** *By replacing r_{LTL} with r_{CyclER} in 3, the solution to problem 5 remains (approximately) probability optimal in satisfying the LTL formula φ. See Appendix Lemma C.3 for the proof.*

4.3 CyclER with Quantitative Semantics

A number of recent works have explored the usage of *Quantitative Semantics* (QS) to help shape rewards for temporal logic tasks (Li et al., 2017; Balakrishnan & Deshmukh, 2019; Jothimurugan et al., 2021; Kalagarla et al., 2021; Ikemoto & Ushio, 2022). QS defines a set of rules for temporal logic which extend Boolean logic to operations over real values. Using QS, we can take real-valued signals from our atomic propositions $x \in \mathrm{AP}$, and compose them with the QS version of our logical connectives (&, $\neg$ and $\rightarrow$) and temporal operators ((X), (G), (F), and (U)) to compute a real-valued signal for how close a trajectory comes to satisfying a specification. We will refer to this computation as the *quantitative evaluation* of a trajectory with respect to an LTL task. The exact quantitative semantics of the aforementioned LTL operations are provided in Fig. 6 of the Appendix.

Unfortunately, there are several shortcomings of "off-the-shelf" usage of QS as a reward function. Quantitative evaluation produces a single value for an entire trajectory, which makes credit assignment for individual transitions difficult. More pressingly, quantitative evaluation of a finite trace frequently produces values that do not correlate with visits to accepting states in $\mathbb{B}$, especially for LTL formulae with indefinite horizons or arbitrarily ordered sub-goals. Existing approaches have circumvented these issues by using QS for explicitly time-bounded temporal logics with simple tasks (Kalagarla et al., 2021; Balakrishnan & Deshmukh, 2019), or by considering fragments of LTL that can be reasoned about as finite sequences of ordered sub-tasks (Jothimurugan et al., 2021). In what follows, we show that CyclER easily incorporates QS for more effective LTL reward shaping by considering each transition in $\mathbb{B}$ as an independently evaluable sub-task.

We first define some notation. In order to use QS, we assign *robustness measures* (real-valued signals) $f_x : \mathcal{S} \rightarrow \mathbb{R}$ to each atomic proposition $x \in \mathrm{AP}$, where x is true when $f_x(s) \geq c_x$ (a constant threshold). We will follow standard practice and notate the quantitative evaluation of an LTL formula φ over a trajectory τ as $\rho_\varphi(\tau)$, where φ is true when $\rho_\varphi(\tau) > 0$. We also define a maximum and minimum for ρ as $\rho_{\max}$ and $\rho_{\min}$, respectively.

Now we explain how to incorporate QS into CyclER. At a given state b in $\mathbb{B}$, we can think of our sub-task as taking the next transition in the accepting path or cycle currently under consideration by CyclER. We need to only consider one transition at a time because CyclER reasons about each accepting path and cycle independently. Our approach to incorporating QS builds on this idea by rewarding quantitative *progress* towards taking the next transition in a path or cycle. Importantly, each transition ν in $\mathbb{B}$ is associated with an atomic predicate, which can be quantitatively evaluated at individual states rather than entire trajectories (i.e., $\rho_\nu(s) : \mathcal{S} \rightarrow \mathbb{R}$). If we move from state s to s' in $\mathcal{M}$, we can evaluate how much closer we are to satisfying the predicate of our next transition ν by taking the difference in quantitative evaluation between successive states: $\rho_\nu(s') - \rho_\nu(s)$. This measure of progress is used to shape reward.

Specifically, our approach to incorporating QS uses the following reward function for a given cycle or initial path c. We use $c[b]$ to refer to the transition predicate in c with parent node b:

$$r_{\mathrm{qs}}(s, b, s', b', e, c) = \begin{cases} \frac{\rho_{c[b]}(s') - \rho_{c[b]}(s)}{(\rho_{\max} - \rho_{\min}) * |c|} & \text{if } (b, L^{\mathcal{M}}(s_{t+1}), b_{t+1}) \in c \text{ and } e[b, L^{\mathcal{M}}(s_{t+1}), b_{t+1}] = 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

We use $\rho_{\max}$ and $\rho_{\min}$ to normalize the quantitative progress made towards taking a transition[2]. Reward function 7 acts as a direct substitute to function 6 in Alg. 1 to compute r_{CyclER}. In our experiments (section 5), we show that incorporating quantitative semantics into CyclER for shaping r_{LTL} leads to improvement in empirical performance when compared to existing methods of using QS. We provide the full QS for LTL, along with additional explanation and examples for CyclER+QS in Appendix E.

[2]We note that the reward in fn. 7 is most well-shaped when the robustness measures for all $x \in \mathrm{AP}$ are of similar scale, but we make no such assumptions for the sake of generality.

5 Experiments

We demonstrate experimental results in several domains with continuous state and action spaces on LTL tasks of varying complexity. We seek to answer the following questions: **(1)** Does CyclER learn satisfactory policies and avoid ignoring r_{LTL} in favor of r_{MDP}? **(2)** Does a policy that optimizes the dual reward formulation in r_{DUAL} gain higher r_{MDP} than a policy that only seeks to satisfy the LTL constraint? **(3)** How does the value of λ in r_{DUAL} affect the performance of the learned policy?

5.1 Experimental Domains and Tasks

In our experiments, we evaluate the efficacy of CyclER on indefinite-horizon (ω-regular) tasks expressible by LTL. We use environments where r_{MDP} does not explicitly correlate with r_{LTL} in order to effectively distinguish between policies that learn to only optimize r_{MDP} and policies that learn to satisfy the LTL specification.

FlatWorld The FlatWorld domain (1) is a two dimensional world with continuous state and action spaces. The agent (denoted by a green dot) starts at (-1, -1). The agent's state, denoted by x, is updated by an action a via $x' = x + a/10$ where $x \in \mathbb{R}^2$ and $a \in [0, 1]^2$. There exists a set of randomly generated purple 'bonus regions', which offer a small reward when visited. We use the specification from Figure 1 as our LTL task.

ZonesEnv We use the Zones environment from the MuJoCo-based Safety-Gymnasium suite of environments (Ji et al., 2023). In this domain, a robot must navigate the environment, which includes four differently colored goal regions and 'hazard' areas that offer a small negative reward. The robot receives an observation of lidar data that detects the presence of nearby objects at each timestep. The LTL task description instructs the agent to oscillate amongst visiting the four colored regions.

ButtonsEnv We use the Buttons environment, also from Safety-Gymnasium. This domain is a more challenging version of the Zones environment, where an agent must press a number of small buttons in a larger space while avoiding cube-shaped 'gremlins' that move in a fixed circular path. The LTL task description instructs the agent to press two specific buttons infinitely often, while avoiding making contact with gremlins. Unlike the ZonesEnv, 'bonus' regions are scattered around the environment, offering a small reward if visited.

5.2 Implementation Details and Baselines

We use entropy-regularized PPO (Schulman et al., 2017) with a Gaussian policy over the action space as our policy class.

Although we are not aware of an existing approach that considers reward optimization under general LTL constraints for deep RL, we compare against a number of existing reward methods for temporal logic-guided RL as r_{LTL} in our r_{DUAL} formulation. We use a baseline policy trained using the LCER method (Voloshin et al., 2023), a state-of-the-art approach to RL for general LTL that uses an unshaped reward with counterfactual experience replay to improve the sample efficiency of learning. Additionally, we compare against the LCER baseline, but trained *only* on the LTL reward function r_{LTL}, in order to observe the performance of a policy that does not get 'distracted' during training by r_{MDP}.

In the ZonesEnv and ButtonsEnv domains, where the dynamics are more complex, we define simple robustness measures for each atomic proposition and use the QS version of CyclER defined in section 4.3. In these environments, we compare against two additional baselines that also use QS for reward shaping and are computable for infinite-horizon LTL tasks: a TLTL-based reward (Li et al., 2017) and BHNR (Balakrishnan & Deshmukh, 2019). For each baseline, λ was chosen to be that which led to best performance (on unshaped r_{LTL}, using r_{MDP} as a tie-breaker) from a hyperparameter sweep. The robustness measures used in these domains along with all hyperparameters used during training are available in Appendix H.

5.3 Results

(1) Does CyclER learn satisfying policies and prevent ignoring r_{LTL}? Yes - our results demonstrate that CyclER achieves significant improvement in performance in satisfying the LTL task when compared to

	FlatWorld		ZonesEnv		ButtonsEnv	
	r_{LTL}	r_{MDP}	r_{LTL}	r_{MDP}	r_{LTL}	r_{MDP}
CyclER	$\mathbf{2.0 \pm 0.5}$	45.3 ± 8.5	$\mathbf{1.8 \pm 0.4}$	-27.8 ± 4.55	$\mathbf{2.6 \pm 0.3}$	30.4 ± 5.6
LCER	0.0 ± 0.0	103.4 ± 76.6	0.0 ± 0.0	-3.8 ± 1.9	0.0 ± 0.0	118.8 ± 143.2
LCER, no r_{MDP}	0.8 ± 0.4	30.8 ± 10.1	0.0 ± 0.0	-2.7 ± 0.9	0.6 ± 0.4	13.2 ± 8.53
TLTL	-	-	0.0 ± 0.0	-4.0 ± 2.2	0.0 ± 0.0	9.6 ± 6.7
BHNR	-	-	0.0 ± 0.0	-0.8 ± 0.6	0.0 ± 0.0	35.8 ± 7.9

Table 1: Reward average and standard deviation achieved on each domain with an extended horizon. r_{LTL} identifies the average number of visits to an accepting state in $\mathbb{B}$ achieved for a trajectory from π, and r_{MDP} refers to the average MDP reward collected during a trajectory.

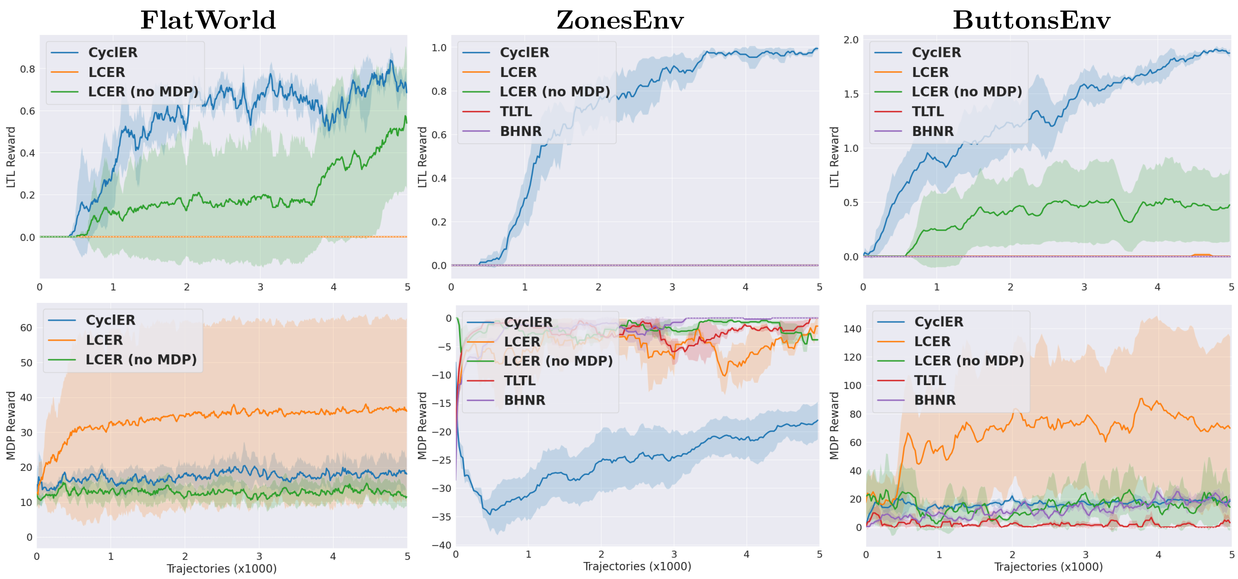

Figure 3: Training curves showing unshaped r_{LTL} (top) and r_{MDP} (bottom) performance averaged over 5 random seeds. Each point is the mean of 10 stochastic policy rollouts.

our baseline methods. In Figure 3, we plot the learning curves for both the *unshaped* r_{LTL} (i.e., the number of times an agent visits an accepting state) and r_{MDP}. We record the (stochastic) performance of the best policies found during training on an extended horizon to enable repeated visits to the accepting state, and present the results (averaged over 50 rollouts) in Table 1.

We find that unshaped rewards (LCER) quickly ignore r_{LTL} in most trials. From Table 1, we see that the LCER baseline even without r_{MDP} was not able to accomplish the LTL task as consistently as CyclER, even when successful in visiting an accepting state. This implies that reward shaping is critical for LTL-guided RL even in settings where no r_{MDP} is present. In ZonesEnv and ButtonsEnv, the TLTL and BHNR baselines, which are not suited to infinite-horizon tasks with multiple unordered subgoals, learned behavior that optimized their respective QS-shaped LTL rewards but did not correlate with task satisfaction (zero r_{LTL} achieved in Fig. 3). CyclER with QS, on the other hand, quickly learned to achieve the tasks in these two domains. Most meaningfully, CyclER is able to repeatedly visit the accepting state in all domains (as evidenced by Table 1), demonstrating that our technique enables consistent-behaving policies that can indefinitely traverse accepting cycles. We additionally provide a qualitative analysis of learned behavior in the FlatWorld domain in Appendix D.

(2) Does optimizing r_{DUAL} improve r_{MDP}? To evaluate this question, we conducted an ablation study where we trained a CyclER-based policy in the FlatWorld domain, making it completely unaware of r_{MDP}, and then evaluated its performance to observe if the r_{DUAL} formulation led to a nontrivial difference in behavior between policies. We found that the r_{DUAL}-optimizing CyclER policy achieved LTL reward of 2.0 (std. dev. 0.3) and MDP reward of 45.3 (std. dev. 8.5), and the r_{LTL}-only policy achieved LTL reward of 2.2

(std. dev. 0.4) and MDP reward of 27.4 (std. dev. 0.7). Optimizing r_{DUAL} does lead to an improvement in r_{MDP}, albeit at the potential cost of r_{LTL}.

(3) How does varying λ affect the resulting policy? In Table 2, we report results from a study where we vary the value of λ in r_{DUAL} for the FlatWorld domain experiment. We observe, as expected, a tradeoff in the performance of r_{LTL} and r_{MDP} as λ increases. However, we notice that the tradeoff diminishes once a value for λ is reached that enables LTL-satisfying behavior. This supports our intuition that λ can effectively be used as a hyperparameter to trade off the empirical performance of LTL satisfaction and the MDP reward achieved by a policy.

	FlatWorld	
	r_{LTL}	r_{MDP}
$\lambda = 100$	0.0 ± 0.0	62.7 ± 4.36
$\lambda = 200$	0.0 ± 2.0	69.6 ± 4.1
$\lambda = 300$	2.0 ± 0.4	37.2 ± 9.0
$\lambda = 400$	2.1 ± 0.4	34.3 ± 7.3

Table 2: Performance results for CyclER with differing λ.

6 Related Work

Temporal Logic-Constrained Policy Optimization. Previous work has explored cost-optimal control under linear temporal logic constraints with known dynamics (Ding et al., 2014; Cai et al., 2021). More recently, interest has emerged in RL-based approaches to logic-constrained policy optimization. Voloshin et al. (2022) provides an exact solution method for policy optimization under general LTL constraints in discrete settings where the dynamics are unknown by assuming a lower bound on transition probabilities in $\mathcal{M}$. Other works focus on Signal Temporal Logic (STL) and are either designed for discrete spaces (Kalagarla et al., 2021) or lack guarantees (Ikemoto & Ushio, 2022). To the best of our knowledge, this work is the first to extend policy optimization under general LTL constraints to continuous spaces (and thereby DRL), providing theoretical guarantees of soundness for both our objective formulation and reward shaping technique.

RL with Temporal Logic Objectives. In contrast to settings with both temporal logic constraints and reward functions, a significant amount of work has been devoted to developing RL approaches with temporal logic specification(s) as the lone objective. Early efforts focused primarily on using Q-learning-style methods over augmentations of $\mathcal{M}$ (Sadigh et al., 2014; Aksaray et al., 2016; Venkataraman et al., 2020; Cai et al., 2021). Subsequent works (Hasanbeig et al., 2020; Toro Icarte et al., 2022; Camacho et al., 2019; Jothimurugan et al., 2019) extend temporal logic-guided RL to deep RL settings. In developing the theoretical limitations of temporal-logic guided RL, (Yang et al., 2022; Alur et al., 2022) show that guarantees on RL for LTL cannot in general be made. To obtain (approximate) guarantees on learning, existing works have made assumptions on the environment dynamics (Fu & Topcu, 2014; Voloshin et al., 2022; Wolff et al., 2012) or finitized the policy's horizon through discounting or recurrence time (Alur et al., 2023; Perez et al., 2023). In continuous spaces, prior works provide guarantees that the optimal policy under a proxy objective will satisfy the original logical specification of interest (Voloshin et al., 2023; Hasanbeig et al., 2020; Jothimurugan et al., 2021; Camacho et al., 2019), similar to the guarantees made in our work.

To handle longer-horizon specifications, previous endeavors proposed compositional RL approaches that leverage the DAG-like structure for finitary fragments of temporal logic (Jothimurugan et al., 2021; Bonassi et al., 2023). Other works take a multi-task RL approach that learns subtasks, which allows for the completion of extended-horizon tasks and unseen tasks over the same AP set (Vaezipoor et al., 2021; Qiu et al., 2023; León et al., 2022; Liu et al., 2022). More recent work considers problem settings where there is uncertainty in an agent's knowledge of atomic propositions and proposes a belief-based approach to policy learning in this setting (Li et al., 2024). Our approach is able to handle indefinite-horizon specifications for single tasks and we see the integration of our reward shaping into both multi-task frameworks and noisy environments as exciting directions for future work. An extended discussion of related work is available in Appendix F.

7 Conclusion

This paper proposes a novel approach to finding policies that are both reward-maximal and probability-optimal with respect to an LTL constraint. Specifically, we introduce CyclER, an experience replay technique that automatically shapes the LTL proxy reward based on cycles within a Büchi automaton, alleviating a sparsity issue that often plagues LTL-driven RL approaches. CyclER enables LTL-constrained policy optimization in continuous spaces using function approximators. We extend CyclER to effectively use quantitative semantics for full LTL and demonstrate its success empirically.

There are numerous directions for future work. For example, the reward shaping idea behind CyclER can be extended to other classes of logical specifications, such as Reward Machines (Toro Icarte et al., 2022). We are also interested in applying CyclER to accelerate learning in multi-task LTL settings, such as (Vaezipoor et al., 2021; Qiu et al., 2023).

References

David Abel, Will Dabney, Anna Harutyunyan, Mark K Ho, Michael Littman, Doina Precup, and Satinder Singh. On the expressivity of markov reward. In *Advances in Neural Information Processing Systems*, 2021. URL `https://proceedings.neurips.cc/paper/2021/file/4079016d940210b4ae9ae7d41c4a2065-Paper.pdf`.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pp. 22–31. PMLR, 2017.

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

Eitan Altman. *Constrained Markov decision processes*. Routledge, 2021.

Rajeev Alur, Suguman Bansal, Osbert Bastani, and Kishor Jothimurugan. A framework for transforming specifications in reinforcement learning. In Jean-François Raskin, Krishnendu Chatterjee, Laurent Doyen, and Rupak Majumdar (eds.), *Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, volume 13660 of *Lecture Notes in Computer Science*, pp. 604–624. Springer, 2022. doi: 10.1007/978-3-031-22337-2_29. URL `https://doi.org/10.1007/978-3-031-22337-2_29`.

Rajeev Alur, Osbert Bastani, Kishor Jothimurugan, Mateo Perez, Fabio Somenzi, and Ashutosh Trivedi. Policy synthesis and reinforcement learning for discounted LTL. In Constantin Enea and Akash Lal (eds.), *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part I*, volume 13964 of *Lecture Notes in Computer Science*, pp. 415–435. Springer, 2023. doi: 10.1007/978-3-031-37706-8_21. URL `https://doi.org/10.1007/978-3-031-37706-8_21`.

Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5048–5058, 2017. URL `https://proceedings.neurips.cc/paper/2017/hash/453fadbd8a1a3af50a9df4df899537b5-Abstract.html`.

Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. The MIT Press, Cambridge, Mass, 2008. ISBN 978-0-262-02649-9.

Anand Balakrishnan and Jyotirmoy V. Deshmukh. Structured reward shaping using signal temporal logic specifications. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*, pp. 3481–3486. IEEE, 2019. doi: 10.1109/IROS40897.2019.8968254. URL `https://doi.org/10.1109/IROS40897.2019.8968254`.

Luigi Bonassi, Giuseppe De Giacomo, Marco Favorito, Francesco Fuggitti, Alfonso Emilio Gerevini, and Enrico Scala. Planning for temporally extended goals in pure-past linear temporal logic. In Sven Koenig, Roni Stern, and Mauro Vallati (eds.), *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, July 8-13, 2023, Prague, Czech Republic*, pp. 61–69. AAAI Press, 2023. doi: 10.1609/ICAPS.V33I1.27179. URL `https://doi.org/10.1609/icaps.v33i1.27179`.

Alper Kamil Bozkurt, Yu Wang, Michael M Zavlanos, and Miroslav Pajic. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10349–10355. IEEE, 2020.

Mingyu Cai, Shaoping Xiao, Zhijun Li, and Zhen Kan. Optimal probabilistic motion planning with potential infeasible ltl constraints. *IEEE Transactions on Automatic Control*, 68(1):301–316, 2021.

Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. 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, 7 2019. doi: 10.24963/ijcai.2019/840. URL `https://doi.org/10.24963/ijcai.2019/840`.

Giuseppe De Giacomo, Marco Favorito, Luca Iocchi, Fabio Patrizi, and Alessandro Ronca. Temporal Logic Monitoring Rewards via Transducers. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 860–870, 9 2020. doi: 10.24963/kr.2020/89. URL `https://doi.org/10.24963/kr.2020/89`.

Xuchu Ding, Stephen L. Smith, Calin Belta, and Daniela Rus. Optimal control of markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control*, 59(5):1244–1257, May 2014. ISSN 0018-9286, 1558-2523. doi: 10.1109/TAC.2014.2298143.

Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From Spot 2.0 to Spot 2.10: What's new? In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*, volume 13372 of *Lecture Notes in Computer Science*, pp. 174–187. Springer, August 2022. doi: 10.1007/978-3-031-13188-2_9.

Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009. doi: 10.1016/J.TCS.2009.06.021. URL `https://doi.org/10.1016/j.tcs.2009.06.021`.

Jie Fu and Ufuk Topcu. Probably approximately correct MDP learning and control with temporal logic constraints. In Dieter Fox, Lydia E. Kavraki, and Hanna Kurniawati (eds.), *Robotics: Science and Systems X, University of California, Berkeley, USA, July 12-16, 2014*, 2014. doi: 10.15607/RSS.2014.X.039. URL `http://www.roboticsproceedings.org/rss10/p39.html`.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

Ernst Moritz Hahn, Guangyuan Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. Lazy probabilistic model checking without determinisation. *arXiv preprint arXiv:1311.2928*, 2013.

Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained reinforcement learning, 2018. URL `https://arxiv.org/abs/1801.08099`.

Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate. Deep reinforcement learning with temporal logics. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pp. 1–22. Springer, 2020.

Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/b710915795b9e9c02cf10d6d2bdb688c-Abstract.html`.

Junya Ikemoto and Toshimitsu Ushio. Deep reinforcement learning under signal temporal logic constraints using lagrangian relaxation. *IEEE Access*, 10:114814–114828, 2022. doi: 10.1109/ACCESS.2022.3218216. URL https://doi.org/10.1109/ACCESS.2022.3218216.

Jiaming Ji, Borong Zhang, Jiayi Zhou, Xuehai Pan, Weidong Huang, Ruiyang Sun, Yiran Geng, Yifan Zhong, Juntao Dai, and Yaodong Yang. Safety-gymnasium: A unified safe reinforcement learning benchmark. *arXiv preprint arXiv:2310.12567*, 2023.

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

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional reinforcement learning from logical specifications. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 10026–10039, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/531db99cb00833bcd414459069dc7387-Abstract.html.

Krishna C. Kalagarla, Rahul Jain, and Pierluigi Nuzzo. Cost-optimal control of markov decision processes under signal temporal logic constraints. In *2021 Seventh Indian Control Conference (ICC)*, pp. 317–322, 2021. doi: 10.1109/ICC54714.2021.9703164.

Jan Křetínskỳ, Tobias Meggendorfer, and Salomon Sickert. Owl: a library for ω-words, automata, and ltl. In *International Symposium on Automated Technology for Verification and Analysis*, pp. 543–550. Springer, 2018.

Hoang Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, pp. 3703–3712. PMLR, 2019.

Borja G. León, Murray Shanahan, and Francesco Belardinelli. In a nutshell, the human asked for this: Latent goals for following temporal specifications. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=rUwm9wCjURV.

Andrew C. Li, Zizhao Chen, Toryn Q. Klassen, Pashootan Vaezipoor, Rodrigo Toro Icarte, and Sheila A. McIlraith. Reward machines for deep RL in noisy and uncertain environments. *CoRR*, abs/2406.00120, 2024. doi: 10.48550/ARXIV.2406.00120. URL https://doi.org/10.48550/arXiv.2406.00120.

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

Jason Xinyu Liu, Ankit Shah, Eric Rosen, George Konidaris, and Stefanie Tellex. Skill transfer for temporally-extended task specifications. *CoRR*, abs/2206.05096, 2022. doi: 10.48550/ARXIV.2206.05096. URL https://doi.org/10.48550/arXiv.2206.05096.

Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In Yassine Lakhnech and Sergio Yovine (eds.), *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings*, volume 3253 of *Lecture Notes in Computer Science*, pp. 152–166. Springer, 2004. doi: 10.1007/978-3-540-30206-3_12. URL https://doi.org/10.1007/978-3-540-30206-3_12.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pp. 6292–6299. IEEE, 2018. doi: 10.1109/ICRA.2018.8463162. URL https://doi.org/10.1109/ICRA.2018.8463162.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287, 1999.

Ryohei Oura, Ami Sakakibara, and Toshimitsu Ushio. Reinforcement learning of control policy for linear temporal logic specifications using limit-deterministic generalized büchi automata. *IEEE Control. Syst. Lett.*, 4(3):761–766, 2020. doi: 10.1109/LCSYS.2020.2980552. URL `https://doi.org/10.1109/LCSYS.2020.2980552`.

Mateo Perez, Fabio Somenzi, and Ashutosh Trivedi. A PAC learning algorithm for LTL and omega-regular objectives in mdps. *CoRR*, abs/2310.12248, 2023. doi: 10.48550/ARXIV.2310.12248. URL `https://doi.org/10.48550/arXiv.2310.12248`.

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

Wenjie Qiu, Wensen Mao, and He Zhu. Instructing goal-conditioned reinforcement learning agents with temporal logic objectives. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL `http://papers.nips.cc/paper_files/paper/2023/hash/7b35a69f434b5eb07ed1b1ef16ace52c-Abstract-Conference.html`.

Dorsa Sadigh, Eric S Kim, Samuel Coogan, S Shankar Sastry, and Sanjit A Seshia. A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pp. 1091–1096. IEEE, 2014.

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

Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský. Limit-deterministic büchi automata for linear temporal logic. In Swarat Chaudhuri and Azadeh Farzan (eds.), *Computer Aided Verification*, pp. 312–332, Cham, 2016. Springer International Publishing. ISBN 978-3-319-41540-6.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *J. Artif. Int. Res.*, 73, may 2022. ISSN 1076-9757. doi: 10.1613/jair.1.12440. URL `https://doi.org/10.1613/jair.1.12440`.

Pashootan Vaezipoor, Andrew C. Li, Rodrigo Toro Icarte, and Sheila A. McIlraith. Ltl2action: Generalizing LTL instructions for multi-task RL. In *Proceedings of the 38th International Conference on Machine Learning, ICML*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10497–10508, 2021. URL `http://proceedings.mlr.press/v139/vaezipoor21a.html`.

Harish K. Venkataraman, Derya Aksaray, and Peter Seiler. Tractable reinforcement learning of signal temporal logic objectives. In Alexandre M. Bayen, Ali Jadbabaie, George J. Pappas, Pablo A. Parrilo, Benjamin Recht, Claire J. Tomlin, and Melanie N. Zeilinger (eds.), *Proceedings of the 2nd Annual Conference on Learning for Dynamics and Control, L4DC 2020, Online Event, Berkeley, CA, USA, 11-12 June 2020*, volume 120 of *Proceedings of Machine Learning Research*, pp. 308–317. PMLR, 2020. URL `http://proceedings.mlr.press/v120/venkataraman20a.html`.

Cameron Voloshin, Hoang Minh Le, Swarat Chaudhuri, and Yisong Yue. Policy optimization with linear temporal logic constraints. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=yZcPRIZEwOG`.

Cameron Voloshin, Abhinav Verma, and Yisong Yue. Eventual discounting temporal logic counterfactual experience replay. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 35137–35150. PMLR, 2023. URL `https://proceedings.mlr.press/v202/voloshin23a.html`.

Chuanzheng Wang, Yinan Li, Stephen L Smith, and Jun Liu. Continuous motion planning with temporal logic specifications using deep neural networks. *arXiv preprint arXiv:2004.02610*, 2020.

Eric M. Wolff, Ufuk Topcu, and Richard M. Murray. Robust control of uncertain markov decision processes with temporal logic specifications. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 3372–3379, 2012. doi: 10.1109/CDC.2012.6426174.

Cambridge Yang, Michael L. Littman, and Michael Carbin. On the (in)tractability of reinforcement learning for ltl objectives. In Lud De Raedt (ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 3650–3658. International Joint Conferences on Artificial Intelligence Organization, 7 2022. doi: 10.24963/ijcai.2022/507. URL https://doi.org/10.24963/ijcai.2022/507. Main Track.

Contents

1 Introduction **1**

2 Problem Setting **2**

2.1 Preliminaries 2

2.2 Problem Statement 3

3 LTL-Constrained Policy Optimization **4**

4 Cycle Experience Replay (CyclER) **5**

4.1 Rewarding Accepting Cycles in $\mathbb{B}$ 5

4.2 Reward Shaping with CyclER 6

4.3 CyclER with Quantitative Semantics 7

5 Experiments **8**

5.1 Experimental Domains and Tasks 8

5.2 Implementation Details and Baselines 8

5.3 Results 8

6 Related Work **10**

7 Conclusion **10**

A Limitations **17**

B Proof for Theorem 3.2 **17**

B.1 On the existence of Assumption 3.1. 17

B.2 On the existence of a solution to 4 18

C Proof for Theorem 4.1 **18**

D Additional Experimental Results **20**

E Quantitative Semantics for LTL **20**

F Extended Related Work **22**

G Additional Algorithmic Details **24**

G.1 Finding Minimal Accepting Cycles and Accepting Initial Paths 24

H Additional Experimental Details **24**

A Limitations

The success of CyclER is ultimately limited to the quality and achievability of the atomic propositional variables in a given environment. If robustness measures are not available and a task specification has relatively few variables that are difficult to satisfy, CyclER will not provide significant improvement over existing unshaped LTL reward proxies. For example, the specification $F(G(x))$ with no robustness measure for x will offer the same reward under CyclER and existing methods. When robustness measures are available, it is important that measures for each variable in the set AP are of a similar scale, so that the QS-shaped rewards do not vary highly across different transitions in $\mathbb{B}$. The issue of needing similar scale for robustness measures is well-known in the temporal logic literature and is an open direction for future work Balakrishnan & Deshmukh (2019); Li et al. (2017).

Although we show that the performance of policy learning is somewhat robust to λ in our formulation of r_{DUAL}, we do not have a systematic way of finding an appropriate value for λ beyond traditional hyperparameter search methods. We see an interesting opportunity for future work in intelligently searching for λ based on the desired r_{LTL} and r_{MDP} of a user.

The CyclER approach incurs computational overhead by (1) computing all possible cycles prior to policy learning and (2) keeping track of all potential reward values for each cycle for each trajectory stored in an agent's replay buffer. Although we did not observe a significant slowdown or memory increase in our experiments as a result of this overhead, we acknowledge that in complex specifications with a large number of cycles this overhead may become meaningful.

B Proof for Theorem 3.2

Proof. Consider two policies: (1) $\pi \in \Pi \setminus \Pi^*$, which does not achieve $V_{\max}$, (2) $\tilde{\pi} \in \Pi^*$, achieving $V_{\max}$. Let $\mathcal{R}_{\max}$ and $\mathcal{R}_{\min}$ be upper and lower bounds on the maximum and minimum achievable $\mathcal{R}$ in $\mathcal{M}$, respectively. Evaluating objective 5 for both of these policies satisfies the following series of inequalities:

$$\mathcal{R}_{\tilde{\pi}} + \lambda V_{\tilde{\pi}} \overset{(a)}{\geq} \frac{\mathcal{R}_{\min}}{1-\gamma} + \lambda(V_\pi + \epsilon) \overset{(b)}{\geq} \frac{\mathcal{R}_{\max}}{1-\gamma} + \lambda V_\pi \overset{(c)}{\geq} \mathcal{R}_\pi + \lambda V_\pi$$

where (a) follows from assumption 3.1 and bounding the worst-case MDP value, (b) follows from selecting $\lambda > \frac{R_{\max} - R_{\min}}{\epsilon(1-\gamma)} (\equiv \lambda^*)$, (c) follows since the highest MDP value achievable by π must be upper bounded by the best-case MDP value.

As a consequence of $(a-c)$ we see that policies achieving $V_{\max}$ are preferred by objective 5. Consider $\pi^* \in \Pi^*$, the solution to objective 5. Thus, since $\pi^* \in \Pi^*$, then π^* must also achieve $V_{\pi^*} = V_{\tilde{\pi}} = V_{\max}$. Therefore, in comparing objective 5 for both π^* and $\tilde{\pi}$ it follows immediately that $\mathcal{R}_{\pi^*} \geq \mathcal{R}_{\tilde{\pi}}$ since π^* is optimal for objective 5. Since the choice of $\tilde{\pi}$ is arbitrary, we have shown that π^* is also a solution to objective 4. □

B.1 On the existence of Assumption 3.1.

If we are restricted to stationary policies and the space of policies Π is finite, then assumption 3.1 will always hold. A finite space of policies can be enumerated over, and we can take the difference between the optimal and next-best policies to find ϵ. As an example, consider a toy MDP with a continuous, 1-dimensional state space $[0, 1]$ and continuous, 1-dimensional action space $[0, 1]$, where the transition function determines the next state as the agent's action, i.e. $T^{\mathcal{M}}(s, a, s')$. Suppose we are given a task specification $G(F(1))$. Under this specification, the agent will receive a reward of 1 every time it outputs 1, and 0 otherwise.

Consider a finite-sized policy class Π of just two deterministic policies: π_0 that only outputs 0, and π_1 that only outputs 1. Here, assumption 3.1 holds even in this continuous space.

However, assumption 3.1 is not limited to just finite-sized policy classes. Consider an infinite-sized Π, where one policy in Π, called π^*, always outputs 1, and all other policies are Gaussian policies with $\sigma = 0.0001$ and μ uniformly sampled from the interval $[0, 0.0001]$. Here, even when Π is infinite and contains stochastic policies in a continuous space, assumption B.1 holds.

Although the aforementioned examples are toyish, they demonstrate that although assumption 3.1 is always true when Π is finite, this is not the only case. Further characterization for when the assumption holds is left for future work.

B.2 On the existence of a solution to 4

In our definition of Π^* in 2.5 and the formulation of our objective in 2, it is possible that no solution to 2 exists in settings where Π^* is empty. This non-existence issue reoccurs in objective 4, where it is possible that no policies exist that achieve $V_{\max}$. In all of these cases, non-existence is a result of an infinite-sized Π where a sequence of policies exists in Π that come increasingly arbitrarily close to achieving $V_{\max}$ (or, in the case of 2, to achieving the optimal probability of satisfying φ), without ever reaching the maximum value.

In these cases, we clarify what a policy that optimizes our true objective 5 is actually achieving. Recall that we ultimately aim to optimize a proxy objective of $\mathcal{R}_\pi + \lambda V_\pi$, under a sufficiently large λ. If the set Π^* is empty, then by definition, there does not exist a "gap" between policies in Π^* and policies outside of it. In other words, the value of ϵ is 0, and Assumption 3.1 does not hold. As a result, if λ is sufficiently large, the sequence of policies that is increasingly optimal with respect to $\mathcal{R}_\pi + \lambda V_\pi$ will always prioritize increasing V_π over $\mathcal{R}_\pi$. In other words, if assumption 3.1 does not hold, optimizing 5 will continually try to improve the proxy reward for LTL satisfaction and will ignore resulting changes to the MDP reward $\mathcal{R}_\pi$.

It is theoretically possible that subsequent policies along the aforementioned sequence have arbitrarily different $\mathcal{R}_\pi$, which is undesirable. However, in practice, this does not tend to be the case, and policies that learn to optimize objective 5 exhibit behavior that is apparently both LTL-satisfying and performant in MDP reward, as evidenced by our experiments in Section 5. Generally, values for $\mathcal{R}_\pi$ are not highly unstable along the sequence of policies that are increasingly optimal with respect to V_π, and allowing λ to serve as a hyperparameter effectively enables a tradeoff to value $\mathcal{R}_\pi$ against V_π (see Section 5).

C Proof for Theorem 4.1

We start with some notation. Let r^τ_{CyclER} represent the reward function for a trajectory τ that are returned by the execution of Alg. 1. Let $T(\tau)$ be the set of timesteps when an accepting state in $\mathbb{B}$ is visited for a trajectory. We write the value function for CyclER, letting Γ_t be the same function as defined in function 3:

Assumption C.1. *Suppose* $T_{\max} = \max_{\pi\in\Pi} \mathbb{E}_{\tau\sim\mathcal{M}^P_\pi}\left[T(\tau)\middle|\tau \not\models \varphi\right] < M$, *there is a uniform bound on the last time a bad (non-accepting) trajectory visits an accepting state across all bad trajectories induced by any policy.*

Lemma C.2. *Under Assumption C.1, for any* $\pi \in \Pi$ *and* $\epsilon > 0$ *we have*

$$|(1-\gamma)V^{cyc}_\pi - \mathbb{P}[\pi \models \varphi]| \leq \epsilon$$

when $\gamma \geq (1-\epsilon)^{\frac{1}{M+1}}$ *is chosen appropriately.*

Proof. We follow the proof style of Lemma 4.1 from Voloshin et al. (2023). Let $\mathbb{P}[\pi \models \varphi] = p$ be the probability that π satisfies the LTL specification φ. Recall the value function

$$V^{\text{cyc}}_\pi = \mathop{\mathbb{E}}_{\tau\sim\mathcal{M}^\varphi_\pi}\left[\sum_{t=0}^{\infty}\Gamma_t r^\tau_{\text{CyclER}}[t]\right]$$

Let $T(\tau)_{(i)}$ be the (random) i-th visit to an accepting state in $\mathbb{B}$. Let $T(\tau)_{(0)}, T(\tau)_{(-1)}$ refer to the first and last visit, respectively.

Because $r^\tau_{\text{CyclER}}[t] = \frac{1}{|c|}$ only when a transition in a cycle is taken (and only once per that transition) and the distance between successive visits to an accepting state is $|c|$ then at most γ^i reward is accumulated between successive visits to an accepting state. In other words $\mathbb{E}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=T_{(i)}+1}^{T_{(i+1)}}\Gamma_t r^\tau_{\text{CyclER}}[t]\right] = \gamma^i$ and therefore $V_\pi^{\text{cyc}} \leq \frac{1}{1-\gamma}$.

Further, every trajectory τ is decomposable into (1) the partial trajectory up to the first visit (ie. at time $T(\tau)_{(0)}$), the partial trajectory between the first and last visit (ie. between time $T(\tau)_{(0)}$ and $T(\tau)_{(-1)}$), and (3) the remainder of the trajectory. For trajectories that satisfy the LTL specification, $T(\tau)_{(-1)} = \infty$, otherwise $T(\tau)_{(-1)} \leq M$, finite and bounded (by Assumption C.1). For ease of notation, we omit the dependence of T on τ (i.e. we write $T(\tau)_{(0)}$ as $T_{(0)}$). By linearity of expectation, we can rewrite our previous equation as:

$$V_\pi^{\text{cyc}} = \mathop{\mathbb{E}}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=0}^{T_{(0)}}\Gamma_t r^\tau_{\text{CyclER}}[t]\right] + \mathop{\mathbb{E}}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=T_{(0)}+1}^{T_{(-1)}}\Gamma_t r^\tau_{\text{CyclER}}[t]\right] + \mathop{\mathbb{E}}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=T_{(-1)}+1}^{\infty}\Gamma_t r^\tau_{\text{CyclER}}[t]\right].$$

When a path τ is accepting, by definition, $\mathbb{E}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=0}^{T_{(0)}}\Gamma_t r^\tau_{\text{CyclER}}[t]\middle|\tau\models\varphi\right] = 1$ because every accepting initial path will achieve a reward of 1.

By considering accepting trajectories of LTL formula φ, then $T_{(-1)} = \infty$:

$$V_\pi^{\text{cyc}} \geq \left(1 + \mathop{\mathbb{E}}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=T_{(0)}+1}^{\infty}\Gamma_t r^\tau_{\text{CyclER}}[t]\middle|\tau\models\varphi\right]\right)\mathbb{P}[\pi\models\varphi] = \frac{p}{1-\gamma}$$

where the inequality follows from having dropped any value from non-satisfying trajectories. On the other hand, by the law of total expectation, for a lower bound we have:

$$\begin{aligned} V_\pi^{\text{cyc}} &= p\mathop{\mathbb{E}}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=0}^{\infty}\Gamma_t r^\tau_{\text{CyclER}}[t]\middle|\tau\models\varphi\right] + (1-p)\mathop{\mathbb{E}}_{\tau\sim\mathcal{M}_\pi^\varphi}\left[\sum_{t=0}^{\infty}\Gamma_t r^\tau_{\text{CyclER}}[t]\middle|\tau\not\models\varphi\right] \\ &\leq p\frac{1}{1-\gamma} + (1-p)\frac{1-\gamma^{M+1}}{1-\gamma} \end{aligned}$$

where the first term comes from the upper bound on $V_\pi^{\text{cyc}} \leq \frac{1}{1-\gamma}$ and the second term comes from bounding $T_{(0)}$ with a uniform upper bound M by Assumption C.1

Combining the upper and lower bound together and subtracting off p from both sides, we have

$$0 \leq (1-\gamma)V_\pi^{\text{cyc}} - p \leq 1 - \gamma^{M+1}$$

Select $\gamma \geq (1-\epsilon)^{\frac{1}{M+1}}$ which implies that

$$|(1-\gamma)V_\pi^{\text{cyc}} - p| \leq \epsilon$$

□

Lemma C.3. *Let $p^* = \max_{\pi\in\Pi}\mathbb{P}[\pi\models\varphi]$. Under Assumption C.1, then any policy π optimizing V_π^{cyc} (ie. achieving $V_{\max}^{cyc}$} maintains $|V_\pi^{cyc} - p^*| \leq \epsilon$.*

Proof. This follows by an identical argument as in Theorem 4.2 in Voloshin et al. (2023), by using Lemma C.2: V^{cyc}-optimizing policy π^*_{cyc} must satisfy $|(1-\gamma)V_{\max}^{\text{cyc}} - p^*| \leq \epsilon$ when γ is selected as in Lemma C.2.

□

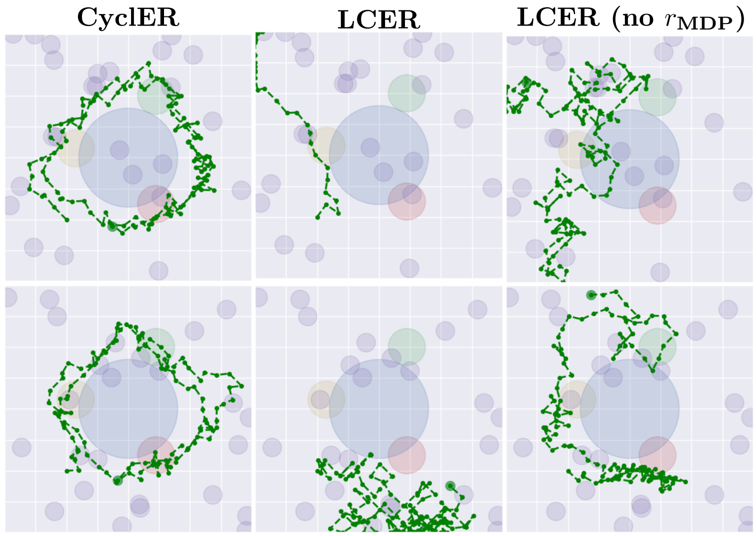

Figure 4: Sample trajectories from each baseline method in the FlatWorld domain (each row is a different seed). CyclER is able to consistently learn behavior that repeatedly visits and accepting state. The LCER baseline cannot achieve this long horizon task and instead optimizes for r_{MDP}. The LCER (no r_{MDP}) baseline, even when successful in visiting the accepting state (bottom right), does qualitatively learn satisfying behavior despite visiting the accepting state.

D Additional Experimental Results

Qualitative Analysis To provide more insight into how CyclER learns to repeatedly visit the accepting state, we visualize sample trajectories from policies learned by each of our baselines in the FlatWorld domain, and present these samples in Figure 4. Recall that in this domain, our LTL specification instructs an agent to traverse the red, yellow, and green regions indefinitely, while avoiding the blue region. It is obvious that the most efficient path to achieve this behavior is to navigate around the blue region and visit each colored region along a circular path.

On the left hand column of Fig. 4, we see that policies learned using CyclER are able to perform this behavior, only slightly diverting from the circular path to visit purple regions and collect reward from r_{MDP}. In the middle column, we visualize trajectories from policies learned where LCER is used as r_{LTL}. Due to the sparsity of this reward function, the policy quickly finds areas in the MDP where purple regions are clustered together, and repeatedly visits those areas (in the top middle, it traverses to a corner where the entropy of the policy will affect its position the least.) On the right hand column, we show trajectories from LCER trained without r_{MDP} in its reward formulation. On the top right, we see that the baseline fails to achieve the task at all. More interestingly, however, on the bottom right, we find that the baseline does indeed achieve the task (i.e., visits the accepting state once), but does not exhibit behavior that qualitatively satisfies our task specification. After completing the task once, the agent turns back in the opposite direction and does not make obvious progress back towards either the red or yellow regions, suggesting that this baseline has not learned to repeatedly satisfy the task.

E Quantitative Semantics for LTL

Practitioners have defined *Quantitative Semantics* (QS) for a number of temporal logics in order to quantify the measure of satisfaction for a given specification. These semantics, originally developed to monitor how close hybrid control systems are to violating properties reliant on continuous-value sensor data (Maler & Nickovic, 2004), have since been used in reinforcement learning to learn policies that satisfy formulae in a variety of specification languages, including Signal Temporal Logic (STL) (Li et al., 2017; Balakrishnan & Deshmukh, 2019). However, existing methods of QS for reward shaping fail to effectively extend to indefinite-horizon LTL tasks. In what follows, we will introduce QS for LTL, discuss why naïvely using QS

$$
\begin{aligned}
\rho(s_{t:t+k}, \top) &= \rho_{\max}, \\
\rho(s_{t:t+k}, f_x(s_t) < c_x) &= c_x - f_x(s_t), \\
\rho(s_{t:t+k}, \neg\varphi) &= -\rho(s_{t:t+k}, \varphi) \\
\rho(s_{t:t+k}, \varphi \implies \psi) &= \max(-\rho(s_{t:t+k}, \varphi), \rho(s_{t:t+k}, \psi)) \\
\rho(s_{t:t+k}, \varphi \& \psi) &= \min(\rho(s_{t:t+k}, \varphi), \rho(s_{t:t+k}, \psi)) \\
\rho(s_{t:t+k}, \varphi \| \psi) &= \max(-\rho(s_{t:t+k}, \varphi), \rho(s_{t:t+k}, \psi)) \\
\rho(s_{t:t+k}, G(\varphi)) &= \min_{t' \in [t, t+k)} (\rho(s_{t':t+k}, \varphi)) \\
\rho(s_{t:t+k}, F(\varphi)) &= \max_{t' \in [t, t+k)} (\rho(s_{t':t+k}, \varphi)) \\
\rho(s_{t:t+k}, X(\varphi)) &= \rho(s_{t+1:t+k}, \varphi)(k > 0) \\
\rho(s_{t:t+k}, (\varphi U \psi)) &= \max_{t' \in [t, t+k)} (\min(\rho(s_{t':t+k}, \psi), \min_{t'' \in [t, t')} (\rho(s_{t'':t'}, \varphi))))
\end{aligned}
$$

Figure 6: Quantitative Semantics for LTL.

as reward is ill-fitted to deep RL for LTL, and introduce our own approach, extending the CyclER reward shaping method to effectively incorporate QS.

To use QS, we associate each atomic predicate $x \in \text{AP}$ with a *robustness measure* $f_x : \mathcal{S} \to \mathbb{R}$ that quantifies how close x is to being satisfied at state s. x evaluates to true at a given state iff $f_x(s) \geq c_x$, where c_x is a constant threshold. We can compose variables in AP with the logical and temporal operators of LTL by introducing QS for each operator, which follows the standard semantics defined for languages like TLTL (Li et al., 2017) and STL (Maler & Nickovic, 2004; Fainekos & Pappas, 2009) and is provided in figure 6. An LTL formula φ is true if the quantitative evaluation of $\rho_\varphi > 0$ and false otherwise. We also define a maximum and minimum achievable value for ρ in a given MDP as $\rho_{\max}$ and $\rho_{\min}$, respectively.

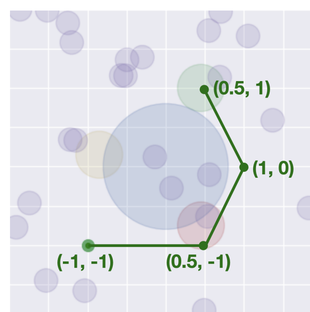

Figure 5: A toy trajectory in the Flatworld MDP.

To better understand quantitative evaluation for a given LTL formula, consider a trajectory (which we will denote as ξ) from the Flatworld MDP, shown in figure 5 and the formula $\varphi_{\text{toy}} = F(r) \& G(\neg b)$. We can define robustness measures for our atomic propositional variables as $f_x(s) = \text{distance}(s, x_{\text{center}}) < x_{\text{radius}}$, requiring that the agent be within a region for its variable to evaluate to true.

Let's quantitatively evaluate φ_{toy} on the trajectory ξ. For the expression $F(r)$, we find the maximum quantitative evaluation for the variable r in our trajectory. Since our trajectory visits the red region at point (0.5, -1), the evaluation for r at that point is some positive value $c_r > 0$, so the expression $F(r)$ will evaluate to true. For the expression $G(\neg b)$, we negate the quantitative evaluation and find the minimum value for the negated evaluation of b. At the point (-1, 0), the agent is closest to the blue region, but since it does not enter it, the minimum value is some positive value $c_b > 0$. The quantitative evaluation for φ_{toy} on ξ is therefore a positive value $\min(c_r, c_b)$, so φ_{toy} evaluates to true for ξ.

Suppose we were to naively use the quantitative evaluation of a given LTL specification "off-the-shelf" as a reward function for an RL agent. Since the quantitative evaluation of a trajectory at any given point in time requires evaluating the future states of the trajectory, we can only assign a reward for entire trajectories. For the toy specification considered in our example, the reward would be the smaller of (1) the maximum distance the agent reaches from the blue region and (2) the minimum distance the agent reaches from the red region during the trajectory. Although this seems reasonable as a reward for our example, the quantitative evaluation of an LTL formula becomes increasingly more obscure as the specification increases in complexity.

For example, for the specification defined in Figure 1, which instructs the agent to indefinitely oscillate amongst the red, green and yellow regions while avoiding blue, ξ would have a lower quantitative evaluation than a trajectory that does not visit any of the three regions but just barely enters the blue region, even though the latter violates the specification without making any qualitative progress. Moreover, there would be no meaningful difference in quantitative evaluation between ξ and a trajectory that visits red, green and yellow while avoiding blue, or between a trajectory that completes the task twice, or thrice, and so on.

The quantitative evaluation of a trajectory as a reward signal for LTL fails because there are no well-defined terminal conditions for evaluating a finite trace under an infinite-horizon specification (e.g., φ from Figure 1), which means that value produced is often useless when used as a reward signal. This is made worse due to the fact that quantitative evaluation of an LTL formula assigns a single value for an entire trajectory, making credit assignment difficult. We propose an alternative reward that avoids these issues, extending the CyclER approach to handle quantitative semantics by rewarding "progress" made by traversing individual transitions in $\mathbb{B}$.

The high-level intuition for our method is as follows: recall that for a given trajectory, CyclER computes hypothetical rewards for each cycle in $\mathbb{B}$, where reward is given if the next transition is taken within that cycle during the trajectory. Our key insight is that we can straightforwardly extend this paradigm to use QS by instead rewarding quantitative progress made towards taking the next transition within an individual cycle. Each transition in $\mathbb{B}$ corresponds to a non-temporal predicate of atomic propositions. Crucially, we can quantitatively evaluate these predicates on individual states, rather than trajectories. When an agent moves to a new state in $\mathcal{M}$, we can quantify how close the agent is to satisfying the transition predicate (and therefore taking the transition in the cycle), and compare it to how close the agent was to satisfying the transition predicate in the previous state. If there is a positive difference between these two values, we reward that progress made towards satisfying the transition predicate.

We present our reward function in function 7 in the main text. In the context of Alg. 1, the reward function defined in 7 will replace $r_{\mathcal{C}}$. Intuitively, we can interpret the reward function defined in 7 as identical to $r_{\mathcal{C}}$, but rewarding quantitative progress towards taking a transition in a cycle rather than only offering reward once that transition is taken. We normalize progress using $\rho_{\max}$ and $\rho_{\min}$ to ensure that the scale of rewards from function 7 remain consistent.

Let us once again return to the example trajectory in Figure 5, with the LTL specification and Buchi automaton from Figure 1 as our objective, and consider the cycle $\{1, 2, 3, 0\}$. We begin in state 1 of $\mathbb{B}$, where the transition we aim to take has the corresponding predicate r. When we transition from (-1, -1) to (0.5, -1), we satisfy r, and receive positive reward from CyclER because we are closer to r than we were in the previous state. We are now in state 2 of $\mathbb{B}$ and seek to take the transition with predicate g. For the transition from (0.5, -1) to (1, 0), we receive positive reward for getting closer to g, and for the transition from (1, 0) to (0.5, 1) we receive positive reward for successfully moving closer to g. Note that each transition of our trajectory ξ qualitatively makes progress towards visiting the accepting state of $\mathbb{B}$, and this is reflected in the positive rewards assigned by r_{qs}.

In environments where the individual variables in AP are difficult to satisfy, the usage of QS can offer a more dense reward that still leverages the full expressivity of LTL. In our experiments, we show that using the QS version of CyclER strongly outperforms existing approaches of using QS in learning to satisfy indefinite-horizon LTL specifications (Li et al., 2017; Balakrishnan & Deshmukh, 2019) and enables the learning of LTL-compliant policies in complex environments.

F Extended Related Work

Temporal Logic-Constrained Policy Optimization. The problem of optimizing a cost or reward function under a set of temporal logic constraints has primarily been explored from a control-theoretic perspective. For general LTL constraints, Ding et al. (2014) provides a solution to this LTL-constrained cost optimization under assumptions that the dynamics of $\mathcal{M}$ are known and that a stationary policy exists in Π that will always satisfy the LTL constraints in the given MDP. Cai et al. (2021) improves on this result by relaxing the assumption that the LTL constraints can always be satisfied. In terms of RL approaches to our problem setting, Voloshin et al. (2022) provides an exact solution method in discrete settings where the dynamics are unknown by assuming a lower bound on transition probabilities in $\mathcal{M}$. Our work takes a step forward in this space, extending policy optimization under general LTL constraints to continuous state and action spaces. Although PAC-learning guarantees cannot be made for general LTL objectives in RL without certain assumptions (as we will discuss later), our formulation does provide theoretical guarantees of soundness for both our objective formulation and reward shaping technique.

Beyond general LTL constraints, there exist efforts that consider policy optimization under different classes of temporal logic constraints, particularly Signal Temporal Logic (STL). Kalagarla et al. (2021) provides an optimal learning approach for discrete state spaces under constraints in a fragment of STL. Separately, Ikemoto & Ushio (2022) considers STL-constrained policy optimization in a deep RL setting. Our objective formulation is similar to that of Ikemoto & Ushio (2022), but we extend the setting to the space of general LTL and provide guarantees on the correctness of our formulation.

RL with LTL Objectives. In contrast to settings with both temporal logic constraints and separate reward functions, a significant amount of work has been devoted to developing RL approaches with LTL specification(s) as the lone objective. Earlier works primarily focus on using Q-learning-style methods over augmentations of $\mathcal{M}$ (Sadigh et al., 2014; Aksaray et al., 2016; Venkataraman et al., 2020; Cai et al., 2021). Subsequent works (Hasanbeig et al., 2020; Toro Icarte et al., 2022; Camacho et al., 2019; Jothimurugan et al., 2019) extend temporal logic-guided RL to deep RL settings. Although previous work shows that guarantees on learning an optimal policy cannot be made for RL with LTL objectives in general (Yang et al., 2022; Alur et al., 2022), existing approaches have obtained (approximate) learning guarantees by making assumptions on the environment dynamics (Fu et al., 2020; Fu & Topcu, 2014; Voloshin et al., 2022; Wolff et al., 2012) or by finitizing the policy's horizon through discounting or recurrence time (Alur et al., 2023; Perez et al., 2023). In DRL-based approaches, where learning guarantees cannot be made, previous approaches make guarantees of varying strength and approximation on their objective formulations, ensuring that the optimal policy under a proxy objective will satisfy the original logical specification (Voloshin et al., 2023; Hasanbeig et al., 2020; Jothimurugan et al., 2021; Camacho et al., 2019). The guarantees made in our work are of a similar vein, with comparable approximative strength to Voloshin et al. (2023).

The compositionality and generalizability of temporal logic specifications have spurred a number of works that exploit these advantages. Jothimurugan et al. (2021) proposes a compositional RL approach that leverages the DAG-like structure of finite temporal logic specifications. Other works leverage these properties to design multi-task RL methods that can generalize to unseen tasks over the same AP set (Vaezipoor et al., 2021; Qiu et al., 2023; León et al., 2022; Liu et al., 2022). Our reward shaping approach can be used in conjunction with these multi-task frameworks to potentially accelerate learning; we leave this as a promising direction for future work.

Reward Shaping (for LTL). Reward shaping for RL is a well-studied problem that seeks to alleviate reward sparsity by offering denser rewards that does not change the optimal policy (Ng et al., 1999). Common approaches to reward shaping include using potential functions (Ng et al., 1999; Hu et al., 2020), using hindsight experience replay (Andrychowicz et al., 2017), or leveraging expert inductive bias (Nair et al., 2018). Similar to potential-based reward shaping, the CyclER approach ensures maintenance of the optimal policy between the shaped and unshaped rewards.

In the context of Reward Shaping for LTL, there have been previous efforts to use the structure and properties of $\mathbb{B}$ to shape the reward proxy of an LTL specification (Hasanbeig et al., 2020; Oura et al., 2020; Wang et al., 2020). Unlike CyclER, these methods rely on generalized LDBAs (GLDBAs) with acceptance conditions that require visiting multiple states in $\mathbb{B}$. GLDBAs often do not encode progress as monotonically as their de-generalized counterparts (for example, many GLDBAs require revisiting previous states in $\mathbb{B}$ to collect all acceptance conditions when the de-generalized version does not), which complicates learning. Moreover, since GLDBA-based approaches must often reason over multiple transitions from a single state, these approaches cannot easily leverage QS for reward shaping in the way CyclER can.

Quantitative Semantics. There have also been previous approaches to temporal logic-guided RL that leverage QS to shape reward. In continuous settings, Li et al. (2017) and Balakrishnan & Deshmukh (2019) leverage the recursive semantics of QS in temporal logic to shape rewards. We directly compare

Algorithm 2: Find Minimal Accepting Initial Paths and Cycles (**FindMAIPsAndMACs**)

Input: Buchi Automaton $\mathbb{B}$, accepting states set $\mathcal{B}^*$
Initialize $\mathcal{C}$ to an empty set;
Initialize $\mathcal{P}$ to an empty set;
DFS(b_{-1}, {}, $\mathcal{P}$);
foreach *accepting state* $b^* \in \mathcal{B}^*$ **do**
    Initialize *visited* to an empty set;
    Initialize C to an empty set;
    DFS(b^*, {}, C);
    Add C to $\mathcal{C}$;
return $\mathcal{C}$, $\mathcal{P}$

against both approaches and show that reasoning over the QS of an entire LTL specification (including its temporal operators) often fails to properly shape reward. QS have been used in reasoning over individual predicates assigned to edges of a temporal logic's corresponding automaton (Jothimurugan et al., 2021), but this approach is limited to finite-time tasks and explicitly makes use of the reach-avoid nature of tasks considered to shape reward. In contrast, CyclER can use QS for the full expressiveness of LTL specifications, as demonstrated in our experiments where QS is used to shape the reward for infinite-time tasks.

Constrained Policy Optimization. The broader constrained policy optimization works mostly relate to the constrained Markov Decision Process (CMDP) framework (Le et al., 2019; Achiam et al., 2017; Altman, 2021), which enforce penalties over expected constraint violations rather than absolute constraint violations. In contrast, our work aims to satisfy absolute constraints in the form of LTL.

G Additional Algorithmic Details

Motivation of visiting frontier. To motivate the importance of maintaining the visited frontier e introduced in section 4.2, we show via example that the existence of non-accepting cycles in $\mathbb{B}$ may allow for trajectories that infinitely take transitions in a MAC or MAIP without ever visiting an accepting state.

Consider the accepting cycle $\{3, 1, 2\}$ in the partial automaton in Figure 7. Although this cycle is a MAC, there does exist a separate cycle starting and ending at state 1 (i.e. the cycle $\{1, 2, 0\}$.) If we give reward every time a transition in the cycle $\{3, 1, 2\}$ is taken, a policy may be able to collect infinite reward without ever visiting an accepting state. For example, in Figure 7, a path $\{1, 2, 0, 1, 2, 0 \dots\}$ would infinitely take transitions in a MAC, and therefore collect infinite reward without ever visiting the accepting state 3. Our visited frontier e will ensure that rewards will only be given once per transition until an accepting state is visited.

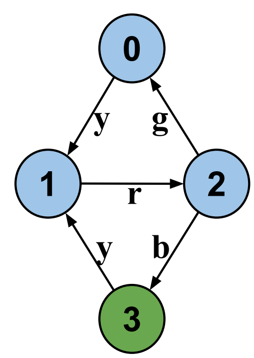

Figure 7: A partial Büchi automaton that necessitates a visited frontier.

G.1 Finding Minimal Accepting Cycles and Accepting Initial Paths

In algorithms 2 and 3, we include the psuedocode for finding minimal accepting initial paths and minimal accepting cycles in a given $\mathbb{B}$, which constitute the sets $\mathcal{P}$ and $\mathcal{C}$ respectively for usage in algorithm 1.

H Additional Experimental Details

Environments and Tasks. For each random seed of training, the locations of the following objects were randomized and fixed: in FlatWorld, the location of the bonus areas, in ZonesEnv, the locations of all colored regions and hazard regions, and in ButtonsEnv, the locations of the buttons, bonus areas, and gremlins. In ZonesEnv and ButtonsEnv, we use the Point robot from Ji et al. (2023) as our agent, which has a 2-dimensional action space $a \in [-1, 1]^2$.

Algorithm 3: DFS (Helper for Alg. 2

Input: Starting node b, Path p, set S
Add node b to *visited*;
foreach *Outgoing transition* (b, ν, b') *from* b **do**
  if $b' \in \mathcal{B}^*$ **then**
    Add the transition (b, ν, b') to p;
    Add p to S;
  else
    if $b' \notin$ *visited* **then**
      Add the transition (b, ν, b') to p;
      DFS(b', p, S);
Remove node b from *visited*;

The observation space and environment used in our ZonesEnv are the the default spaces provided the Zones Level 1 environment in Ji et al. (2023), with the following changes: there are additional lidar observations for each of the four colored zones, and we place four static collidable walls as boundaries to enclose the agent's environments at the border of where objects can be randomly placed. The observation space and environment used in our ButtonsEnv experiments are the default spaces provided the Button Level 1 environment in Ji et al. (2023), with two gremlins and eight bonus regions.

In ZonesEnv and ButtonsEnv, we define a simple robustness measure for each atomic propositional variable in the environments (the red, yellow, green, and purple regions in ZonesEnv, and buttons 1 through 4 and gremlin for ButtonsEnv). The robustness measure for a general variable x at a given state s is defined as follows:

$$f_x(s) = \text{distance}(s, x) \leq 0$$

For example, if an agent was a distance of 2 units away from the red region in ZonesEnv, the robustness measure of the variable "red" at that state would evaluate to -2. For ButtonsEnv, where the gremlin variable refers to multiple moving objects, the robustness measure corresponds to the minimum distance from the agent to any gremlin. We set $\rho_{\max} = 0$ in our environments and $\rho_{\min}$ to be the negative largest distance achievable in each environment.

Our LTL task specifications are defined in Table 3. We use the Spot tool Duret-Lutz et al. (2022) to convert our specifications into corresponding Büchi automata.

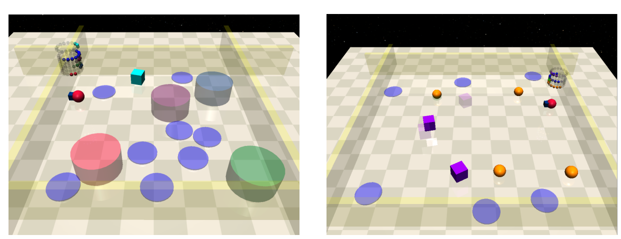

Figure 8: Example visualizations of the ZonesEnv (left) and ButtonsEnv (right) environments.

Training details. For all experiments, results are averaged over five random seeds. We provide hyper-parameter choices for PPO for each experiment in Table 5 and choices for λ in Table 4. In Table 5, batch size refers to the number of trajectories. In our PPO implementation, we use a 3-layer, 64-hidden unit network as the actor using ReLU activations, and a 3-layer, 64-hidden unit network architecture with tanh activations in between layers and no final activation function for the critic. The actor outputs the mean of a Gaussian, the variance for which is learned by a 3-layer, 64-hidden unit network that shares the first 2 layers with the actor policy itself. All experiments were done on an Intel Core i9 processor with 10 cores equipped with an NVIDIA RTX A4500 GPU. We use the Adam optimizer in all experiments.

In Figure 3, reward was computed by evaluating the policy every ten trajectories in the case of FlatWorld, and every 25 trajectories for ZonesEnv and ButtonsEnv. The r_{LTL} and r_{MDP} values shown are from averaging performance over ten rollouts for each data point with a smoothing window of size 5.

Environment	**LTL φ**
FlatWorld	$G(F(\text{red})\&X(F(\text{green})\&X(F(\text{yellow}))))\&G(\neg\text{blue})$
ZonesEnv	$G(F(\text{blue})\&F(\text{purple})\&F(\text{red})\&F(\text{green}))$
ButtonsEnv	$G(F(\text{button1})\&F(\text{button2}))\&G(\neg\text{gremlin})$

Table 3: Specification for each domain.

Environment	**CyclER**	**LCER**	**TLTL**	**BHNR**
FlatWorld	400	1000	-	-
ZonesEnv	200	1000	10	1
ButtonsEnv	100	1000	10	1

Table 4: Values for λ used by baselines for each domain.

For the BHNR baseline, we use a partial signal window size of 60 for FlatWorld, 700 for ZonesEnv, and 750 for ButtonsEnv, treating this value as a hyperparameter and performing a sweep to select the window size. Our TLTL baseline is trained by assigning the TLTL value of a trajectory as the reward at the end of the trajectory, and using the discounted reward-to-go for each prior timestep as the reward signal. For our TLTL and BHNR baselines, we tried computing r_{LTL} in two ways: first, we tried using the original quantitative evaluation of the formula as the reward, where the resulting rewards were mostly negative due to how we defined our robustness measures for each variable. To evaluate if the sign and magnitude of the reward caused difficulty during learning, we normalized the quantitative evaluation done in TLTL and BHNR using $\rho_{\min}$ and $\rho_{\max}$, so that the reward value would be between 0 and 1. We found that this adjustment did not have a significant impact on either baseline's ability to learn r_{LTL}, but it did allow for easier optimization of r_{MDP}; therefore, we report the results from the normalized evaluation in our experiments.

Environment	Critic LR	Actor LR	α	Update freq.	γ	Batch size	$\lvert\tau\rvert$ (training)
FlatWorld	0.001	0.0003	-	1	0.98	128	120
ZonesEnv	0.0125	0.0025	0.3	3	0.99	128	700
ButtonsEnv	0.001	0.0003	0.2	3	0.99	128	750

Table 5: Hyperparameters used during training for each domain.