Structured Exploration in Reinforcement Learning by Hypothesizing Linear Temporal Logic Formulas

Anonymous submission

Abstract

Exploration in vast domains is a core challenge in reinforcement learning (RL). Existing methods commonly explore by adding noise to the learning process, but they do not scale to complex, long-horizon problems. Goal-based exploration is a promising alternative, but it requires useful goals. We propose an approach that structures an agent's exploration by constraining the goal space to tasks that can be expressed using a particular formal language: linear temporal logic (LTL). Our agent proposes LTL expressions that it conjectures to be achievable and desirable for maximizing its learning progress in the environment. Upon proposing an LTL expression, the agent uses a combination of planning and goal-conditioned RL to solve the task described by that LTL. The result is a structured exploration process that learns about the environment by hypothesizing various logical and sequential compositions of atomic goals. We demonstrate the performance of our algorithm outperforms in two challenging sparse-reward problems.

1 Introduction

Training reinforcement learning (RL) agents to effectively explore and solve long-horizon tasks with sparse rewards is challenging. Currently, exploration in RL is often guided by action noise, which is ineffective and leads to sample inefficient learning. At the same time, the world is vast, and it is often infeasible for agents to achieve exhaustive coverage (Javed and Sutton 2024). Recent work like proto-goal RL (Bagaria and Schaul 2023) demonstrates exploration in abstract goal spaces, but rely on myopic, step-by-step sampling of subgoals, which fails to account for temporal structure present in the goal-space.

Formal languages like linear temporal logic (LTL) (Pnueli 1977) have proven to be a powerful abstraction for temporal structure in RL because it is compositional and has unambiguous semantics (Littman et al. 2017; Toro Icarte et al. 2022). By considering each abstract goal as an atomic proposition, we can define a rich and expressive LTL task space. However, the resulting space of all LTL formulas is huge: they can be constructed by sequencing a combination of atomic propositions, logical operators, and temporal operators. This results in an exponentially large space of possible formulas, rendering exhaustive search computationally intractable.

How do we perform efficient exploration in this intractably huge space of LTL expressions? Many previous works rely on manually constructed LTL expressions to guide RL agents (Toro Icarte et al. 2022; Shukla et al. 2024), this approach requires substantial domain expertise. Moreover, even when such expert-crafted expressions are available, they often prove too challenging to achieve due to insufficient exploration. Others assume access to a set of pretrained policies that can be used to compose to solve LTL tasks (Qiu, Mao, and Zhu 2024; Tasse et al. 2022). But, in reality, all policies must be learned, and the agent must carefully manage its exploration budget to focus on learning policies for LTL expressions that likely to lead to learning progress (Kaplan and Oudeyer 2003).

We propose a method that efficiently explores the space of LTL expressions while managing the vastness of the LTL space. Our approach involves training a high-level policy that learns to map the agent's current state to an LTL expression that is *plausible* (as measured by controllability and reachability) and *desirable* (as measured by novelty and reward-relevance) for maximizing learning progress (Bagaria and Schaul 2023). The LTL formula generated by the higher-level policy is then converted into a deterministic finite automaton (DFA). Then, via planning on the DFA, the agent outputs a sequence of goals for a lower-level goal-conditioned policy (Schaul et al. 2015) to achieve.

We evaluate our method via two tests. The first test evaluates the agent's ability to solve a set of predefined tasks (encoded as LTL formulas) in a continuous control problem; we find that our method significantly outperforms existing LTL-conditioned RL approaches in terms of sample efficiency and reward. In the second test, we evaluate our method in a larger, hard-to-explore domain against existing hierarchical RL methods, achieving similar performance to vanilla proto-goal RL.

2 Background and Related Work

We consider problems modeled as Markov Decision Processes (MDP). They can be formulated as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R}$ is the reward function, $\mathcal{T}$ is the transition function, and γ is the discount factor. As is common in RL, we do not assume access to $\mathcal{T}$ and $\mathcal{R}$, and wish to learn a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the sum of discounted rewards

(Sutton and Barto 2018).

Goal-Conditioned RL. In goal-conditioned RL, the agent's policy also conditions its outputs on *goals*: $\pi : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$, where $\mathcal{G}$ is a *goal-space*. A goal $g \in \mathcal{G}$ is formally described using a cumulant $c_g : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ and a continuation function $\gamma_g : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ (Schaul et al. 2015). We consider the subclass of *endgoals*, which imply a binary reward that is paired with termination, i.e, either $(c_g = 0, \gamma_g > 0)$ or $(c_g = 1, \gamma_g = 0)$. Goal-conditioned policies can be learned using all the usual tools from RL (e.g, Q-learning), but certain algorithms boost the sample efficiency of learning (Kaelbling 1993); notably, **Hindsight Experience Replay (HER)** (Andrychowicz et al. 2017) relabels past experience with actually reached goals to deal with the sparse nature of binary end goal reward functions.

Goal-based exploration. Goals provide a convenient way to achieve temporal abstraction (Sutton, Precup, and Singh 1999; Dayan and Hinton 1992) in RL: a higher-level policy $\Pi : \mathcal{S} \rightarrow \mathcal{G}$ outputs goals for a lower-level policy $\pi : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$ to achieve; the higher-level policy typically makes decisions at a coarser timescale than the lower-level policy, which outputs primitive actions at every timestep. This hierarchical approach has been used for exploration (Jinnai et al. 2020; Pong et al. 2019; Ecoffet et al. 2019; Pitis et al. 2020): the higher-level policy outputs goals that lead to "jumpier" forms of exploration than single timestep methods such as ϵ-greedy.

Goal discovery and Proto-goal RL. A key open question for effective goal-based exploration is that of *discovery*: what is the space of goals $\mathcal{G}$ and specific useful subgoals $g \in \mathcal{G}$ that the agent should use to shape its behavior? Most methods either assume that useful goals are already given (but this requires domain knowledge; for example, Option Keyboard (Barreto et al. 2019)) or they assume that the goal space is the same as the state space (but the benefits of abstraction begin to vanish as the environment gets larger; for example, HER (Andrychowicz et al. 2017)). **Proto-goal RL** (Bagaria and Schaul 2023) strikes a balance between these two approaches: it assumes a large space of potential goals (or *proto-goals*) $\mathcal{B}$, but learns a function that outputs a smaller, more useful space of goals for goal-conditioned RL. Each goal is represented as a one-hot binary vector, and goals can be combined to form more complex, multi-hot goals via simple logical operations. To map the proto-goal space into a useful goal space, Bagaria and Schaul (2023) provide sample-based methods for measuring the controllability, reachability, novelty, and reward-relevance of each goal $g \in \mathcal{B}$. Since our work builds on the work of Bagaria and Schaul (2023), **we also assume access to a proto-goal space** $\mathcal{B}$. In their work, the higher-level policy is a multi-armed bandit that outputs a single goal for the lower-level policy to achieve; instead, we leverage the temporal structure of LTLs to develop a high-level policy that outputs *sequences* of goals that can be used to solve more complex long-horizon tasks.

Propositions and Symbols. A proposition α defines a boolean classifier $f_\alpha : S \rightarrow [\text{True}, \text{False}]$ and the corresponding set of states, $S_\alpha = \{s | f_\alpha(s) = \text{True}\}$. Conjunction, disjunction, and negations of propositions can produce *boolean expressions*. The corresponding states of each boolean expression can be constructed from the union, intersection, and complement of propositions:

- $S_{\phi_1 \wedge \phi_2} = S_{\phi_1} \cap S_{\phi_2}$
- $S_{\phi_1 \vee \phi_2} = S_{\phi_1} \cup S_{\phi_2}$
- $S_{\neg\phi} = S_\phi^{\complement}$

Linear Temporal Logic and Buchi Automaton. Linear Temporal Logic (Pnueli 1977) is a formal logic defined over sequences of states. It is commonly used for task specification because it can express complex temporal relations and non-Markovian reward functions (Littman et al. 2017). In this work, we consider a subset of LTLs defined over a finite time horizon called *co-safe LTL*. Following (Lacerda, Parker, and Hawes 2015), we define the grammar of co-safe LTL formulas as:

$$\phi := \alpha \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathbf{X}(\phi) \mid \phi_1 \, \mathbf{U} \, \phi_2.$$

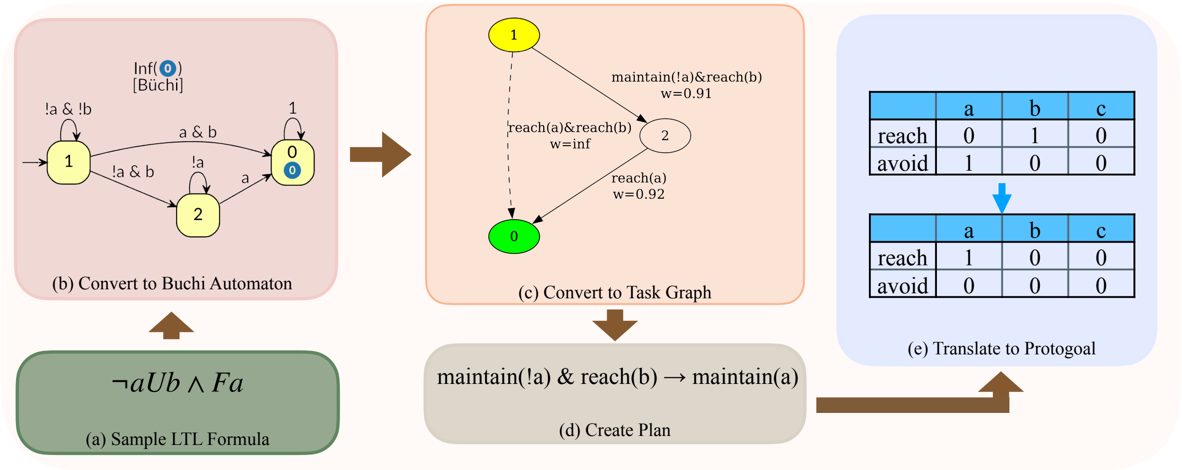
where α is an atomic proposition that maps a state to a boolean value. $\mathbf{X}(\phi)$ ("next") indicates ϕ will happen in the next time step. $\phi_1 \mathbf{U} \phi_2$ ("until") indicates that ϕ_2 will eventually become true, and we should maintain ϕ_1 until ϕ_2 becomes true.

We can convert the co-safe LTL into a *deterministic finite automaton* (DFA), which is described using the following quintuple:

$$(\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_{\text{accept}})$$

where $\mathcal{Q}$ is the set of DFA states, $\Sigma = 2^{\text{AP}}$ is the alphabet of the atomic propositions, δ is the transition function $\mathcal{Q} \times \Sigma \rightarrow \mathcal{Q}$, q_0 is the initial state, and $\mathcal{Q}_{accept}$ is the set of final (accepting) states. The automaton enables us to decompose the task down into a sequence of smaller and more manageable subgoals to reach.

Temporal Logic guided RL. LTL structure has been used as an alternative to scalar rewards for specifying tasks (Littman et al. 2017). Numerous frameworks like Reward Machines (Toro Icarte et al. 2018, 2022) and SPECTRL (Jothimurugan, Alur, and Bastani 2019) use temporal logic to guide RL by generating a product MDP of the state space and the automaton constructed from LTL specifications. They usually assume that LTL tasks to solve are given. However, hand-specifying LTL tasks is tricky for large domains (Greenman et al. 2024), and requires domain knowledge. On the other hand, works like Logic Options Framework (Araki et al. 2021), LTL-Transfer (Liu et al. 2024), GCRL-LTL (Qiu, Mao, and Zhu 2024) and Skill Machine (Tasse et al. 2022) focus on zero-shot generalization to new LTL tasks through the composition of pre-trained policies (*skills*), which are assumed to be given. However, they do not consider the cost of pre-training the skills in the first place—the space of possible policies is large, and an RL agent must balance its exploration budget so that it preferentially collects data that could improve the quality of skills that are more likely to result in learning progress (Kaplan and Oudeyer 2003; Stout and Barto 2010; Quartey, Shah, and Konidaris 2023).



	a	b	c
reach	0	1	0
avoid	1	0	0

	a	b	c
reach	1	0	0
avoid	0	0	0

Figure 1: **Overview of our approach.** First, a coarse policy outputs an LTL formula, which is then converted into a deterministic finite automaton (DFA), and then into a task graph. Planning on the task graph results in a sequence of goals, which are pursued one at a time by a low-level goal-conditioned policy.

3 LTL Guided Exploration

We introduce our approach for exploiting the LTL structure to aid RL exploration. The agent has access to a rich set of propositions, each having a labeling function. We further augment the MDP with an LTL task space, Φ, constructed using the atomic propositions and the LTL grammar. The goal is to learn to utilize the structure of these LTL tasks to help us reach more useful states and solve complex tasks.

In this section, we first describe our approach to representing LTL tasks (Section 3.1) and how to solve LTL tasks using a combination of goal-conditioned RL and planning (Section 3.2). Then, we describe ways to manage the vast LTL task space to find plausible and desirable tasks to pursue given the agent's history of experiences and its current state in the environment (Section 3.2).

3.1 Representing the LTL task

The first step of representing the LTL formula is to convert it into a deterministic finite automaton (DFA); this can be done easily using off-the-shelf software such as `Spot` (Duret-Lutz et al. 2022).

Each edge in the DFA represents a boolean formula. On a high level, the task structure is represented through the DFA and the transition conditions in its edge. On the low level, each edge can be individually treated as a subgoal, from which we can construct a goal space for the goal-conditioned RL policy. By carefully encoding the requirements of each edge in the DFA as a goal input to the policy π, we can instruct the low-level RL policy to solve tasks while obeying constraints.

Representing edges in binary proto-goal space We note that given a planned path through the automaton, the policy should follow the planned path if we ensure the edge-conditioned low-level policy only takes the self-edge or the planned out-edge at each state. So, one simple way is to encode the self-, and out-edge in a Buchi Automaton in the goal space (Liu et al. 2024).

We consider each proposition that appears in the self or out edge. Given the self-edge of the current DFA node and the out-edge to be traversed according to the plan, the agent should work towards the out-edge while trying to maintain the properties of the self-edge during the process. We regard propositions having the same truth value on both edges and propositions only present on the self-edge as constraints and represent them as "**maintain**" goals, as they should always be enforced throughout the entire process while attempting to traverse the edge. For propositions that have a different value on self-edge versus the out-edge or only present on the out-edge, we take the truth value of the proposition in the out-edge and call those "**reach**" goals. Table 1 shows the translation of the edges for each individual proposition.

With the four goals defined above for each proposition, we have defined our proto-goal space. To fully represent all possible edges in the buchi automaton generated from LTL, the proto-goal space $\mathcal{G}$ will contain $\texttt{maintain}(a)$, $\texttt{maintain}(\neg a)$, $\texttt{reach}(a)$, and $\texttt{reach}(\neg a)$ for all propositions α in the atomic proposition space.

At the same time, not all of the above four types of goals are useful for solving tasks. From these, we can pick a subset of the four different sets of goals:

1. $\texttt{reach}(a)$
2. $\texttt{reach}(a) + \texttt{reach}(\neg a)$
3. $\texttt{reach}(a) + \texttt{maintain}(\neg a)$
4. $\texttt{reach}(a) + \texttt{maintain}(a) + \texttt{reach}(\neg a) + \texttt{maintain}(\neg a)$

As we expand the set of goals to include not just reaching goals but also maintaining goals or constraints, the task space we're exploring becomes more expressive, but the exploration and goal-tracking cost is also growing. Creating a

		out edge		
		a	$\neg a$	$\varnothing$
self edge	a	maintain(a)	reach($\neg a$)	maintain(a)
	$\neg a$	reach(a)	maintain($\neg a$)	maintain($\neg a$)
	$\varnothing$	reach(a)	reach($\neg a$)	-

Table 1: Correspondance between the goal type and self/out edge types in the DFA.

balance of expressiveness and the goal space size is a tradeoff, and we leave the inclusion or exclusion of reach/maintain goals as a hyperparameter to be decided by the goal space designer.

Multi-hot goals and reward assignment Multiple goals can be active simultaneously in the binary goal space, beyond just individual `reach` and `maintain` goals. For a policy to be considered successful, it must satisfy all active goals' requirements.

This goal space encoding naturally represents conjunctions (`and` operations) in Boolean formulas. To handle all boolean expressions, including disjunctions (OR operations), we first convert each formula to disjunctive normal form—a representation using `or`s of `and`s. We then split formulas containing `or`s into multiple separate formulas. Finally, we apply our transformation table to create individual goals for each pair of split self and out edges.

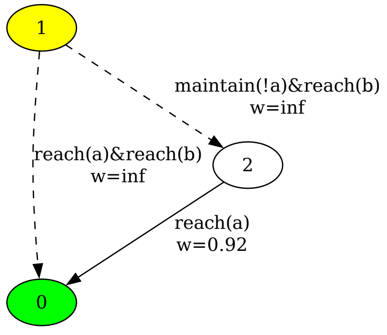
Given the reach/maintain information for each atomic proposition in the goal, We can assign rewards to each requirement. We use the following reward function to give rewards to a goal-conditioned RL policy conditioned on multi-hot goals, terminating when the reward is non-zero:

$$R(s, s') = \begin{cases} 1 & \text{if all reach goals satisfied and} \\ & \quad \text{no maintain goals violated} \\ -1 & \text{if any maintain goals is violated} \\ 0 & \text{otherwise} \end{cases}$$

Conversion of automata into task graphs An automaton only passively verifies whether a sequence of states satisfies the specification. We define task graphs to enable active planning by encoding transition conditions into the RL goal space:

Definition 1 (Task Graph)**.** *A Task Graph for an LTL task is a labeled directed graph* $G = \langle N, E, \Sigma, \mathcal{G}, l_n, l_e, c, n_0, N_{goal} \rangle$ *where* N *is the set of nodes,* $E \in (N \times N)$ *is a tuple defining the edges with its start node and end node,* Σ *is the set of boolean expressions constructable by the atomic propositions,* $\mathcal{G}$ *is the goal space of the RL policy,* $l_n : N \to \Sigma$ *maps nodes to the boolean expression at each node,* $l_e : E \to \mathcal{G}$ *maps each edge to RL goal needed to traverse the edge, and* $c : E \to \mathbb{R}^+$ *is the cost of traversing the edge;* n_0 *is the initial node, and* N_{goal} *is the set of goal nodes.*

We use Algorithm 1 to convert the DFA into a plannable Task Graph where each edge encodes the goal needed. This graph can then be annotated to plan a path to reach the goal node.



$\neg$a **U** b $\wedge$ $\mathbf{F}(a)$

Figure 2: Example converted graph with edges annotated with weights. Dashed edges are edges deemed implausible and pruned.

3.2 Goal management and high-level task graph planning

The conversion of DFA into a task graph allows us to encode the transition conditions into markovian goals. We now need to specify how to find an optimal path through the graph.

Many key challenges emerge: We need a principled way to select which LTLs to execute and which goals to focus on. Second, some edges in the graph may be implausible - either unreachable or uncontrollable. Consider Figure 2, where the edge `reach`(a) & `reach`(b) becomes implausible if the zones have no overlap, making simultaneous satisfaction impossible. Such implausible edges must be pruned from the DFA. Lastly, finding the shortest path between starting and accepting states requires knowing the cost of traversing each edge. All of this requires effectively managing the set of relevant edges and estimating the agent's current capabilities and environment rules.

Goal Pruning We start by tracking all reach and maintain goals of the individual atomic propositions and their negations in the goal space. We estimate the plausibility of each goal for evaluation using the data sampled from the replay buffer $\mathcal{B}$ and define the following three criteria for a goal to be plausible:

- **Observed** – The goal of interest has to be observed in the agent's experience. For example, you can't be in two places simultaneously. A goal is observed if the global count of each goal $N(g) > 0$.
- **Reachable** – The goal must be reachable by the agent. Even if a goal is observed and thus possible, it might have

happened only a few times, which is extremely unlikely to be reachable by the agent. A goal is globally reachable if $\max_{s \sim \mathcal{B}} V_{\text{seek}}(s, g) > \tau_1$.

- **Controllable** – The agent should be able to control whether it reaches the goal. For example, the current weather is outside our control, even if it's reachable. A goal is controllable if $\mathbb{E}[V_{\text{seek}}] - \mathbb{E}[-V_{\text{avoid}}] < \tau_2$.

Here, V_{seek} and V_{avoid} are general value functions (Sutton et al. 2011) estimated using two iterations of LSPI (Lagoudakis and Parr 2003) with the following cumulants (Bagaria and Schaul 2023):

$$R_{\text{seek}}(s, g) = 1 \text{ if } g \text{ is achieved in } s \text{ else } 0,$$
$$R_{\text{avoid}}(s, g) = -1 \text{ if } g \text{ is achieved in } s \text{ else } 0.$$

If the goals on each edge satisfy the above properties, then we say the edge is plausible. Otherwise, the edge is not plausible and will be removed.

Goal Recombination Solely tracking the individual success of reaching atomic propositions is not enough, as the edges in the graph also include conjunctions of goals. In addition to the above three metrics, we maintain the active pursue success rate counter.

If the agent has actively pursued a goal and the recent success rate of such goal exceeds a threshold τ_3, we deem these goals mastered. We create combined goals from conjunctions of individual mastered goals, and the newly recombined goal enters the tracking cycle again and is evaluated using the three plausibility metrics mentioned in Section 3.2.

Estimating edge weights and path-finding through the graph After pruning the graph, we are left with a graph where each edge is plausible on its own. To find the shortest path to the accepting state, we must assign weight to each edge. The proto-goal framework used the expected value of V_{seek} to estimate the reachability of each goal. This is not sufficient, as the feasibility of the agent in traversing each edge depends on the state where the edge originates. For example, you cannot wash an apple after you have already eaten it.

To access the success probability and weight of each edge, we first need to know the value of each edge conditioned on each DFA state q where it originates from. Here, we use the common technique of using associated concrete states to estimate the values of abstract states (Bagaria, Senthil, and Konidaris 2021) to measure the value of the policy traversing from n to n' through edge representing goal g:

$$v(n, n') = \mathbb{E}_{s \sim l_n(n)}[V_{\text{seek}}(s, l_e(e_{n,n'}))],$$

where $l_n(n)$ represents the boolean formula representing the state in the starting node n, and $l_e(n, n')$ represents the RL goal needed to traverse the edge from n to n'.

Intuitively, we first match the DFA state q with a set of abstract states $\{\psi_{\text{in}}\}$ that match the edges going into the previous DFA state q. We then find all corresponding concrete states s where one of ψ_{in} is true, stored in memory buffer during prior agent interaction. The value of reaching g is then computed by the expected value of all concrete states associated with the DFA state q.

Finally, with all the implausible edges pruned and the values assigned to each edge, we utilize the negative log of weight $-\log(v(g))$ as the weight on the graph (Jothimurugan et al. 2021; Qiu, Mao, and Zhu 2024). Using Dijkstra's algorithm, we can find the shortest path between the starting state and any of the accepting states. More details of our edge labeling and path-finding algorithm are in the appendix.

Generating desirable LTL tasks to explore Similar to the proto-goal framework, we use the same simple count-based novelty metric for each of the goals tracked. The desirability score for each goal is

$$u(g) = R(g) + \text{novel}(g).$$

where $R(g)$ is the average reward received when attempting to reach g, and the novelty is the inverse of the number of times g has been achieved: $\text{novel}(g) = 1/N(g)$.

And the probability of each goal being sampled is thus

$$p(g) = \frac{u(g)}{\sum_{g' \in \mathcal{G}} u(g')}.$$

With the desirability sampling probability for each goal defined, we take the simple approach of sampling a set of goals up to a certain novelty threshold $\text{novel}_{\text{max}}$, and up to a max number of goals. We fill these goals into the pre-compiled set of LTL templates, which are listed in the appendix. We keep sampling until we land at an LTL where we can find a path from the initial state to any of the accepting states, which is added to the queue as a desired task.

Finally, at runtime, the agent takes 5 sampled LTLs from the queue, and picks the one where the first edge in the path is the most likely to be achieved. This allows the agent to pick the best LTL without being distracted to solve the hardest LTLs sampled that the agent cannot achieve yet.

4 Experiments

We test our method in two environments: ZoneENV (Vaezipoor et al. 2021), and Minigrid (Chevalier-Boisvert et al. 2023).

4.1 LTL-conditioned RL

To verify our framework's capability of covering the task space of LTL, we first benchmark on an LTL-conditioned RL environment, ZoneENV (Vaezipoor et al. 2021). In this environment, the agent controls a point mass with two degrees of freedom: accelerate/decelerate and turn left/right. We provide the agent with four propositions, each representing whether the agent is in each of the zones.

The avoidance task is the hardest among the few specified in the original ZoneENV. In this set of tasks, a sequence of zones must be satisfied while avoiding some other zones. An example LTL in this category is $\neg$zone_Y **U** (zone_W $\wedge$ ($\neg$zone_J **U** zone_R). To achieve this LTL task, the agent must first visit Zone W while avoiding Zone Y and Zone J and then visit Zone R while avoiding Zone J. Violation of the constraint will terminate the episode.

We compare our framework against two baselines that also do not assume access to the LTL task distribution.

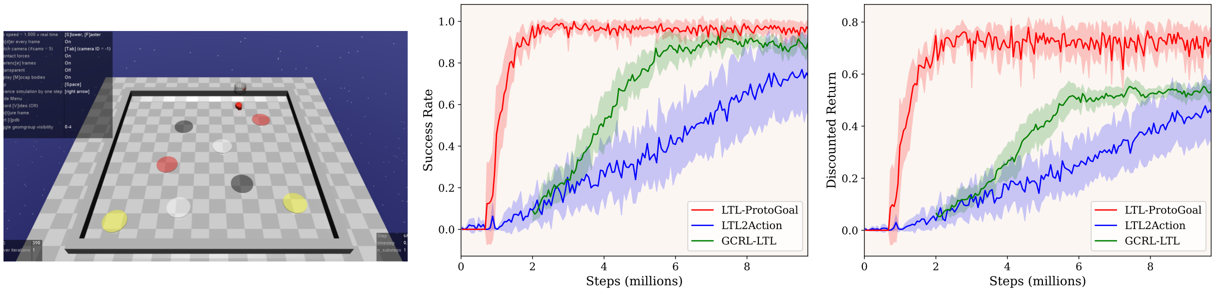


Figure 3: Performance metrics for the avoidance task in ZoneEnv. Success rate (left) and discounted return (right) averaged over 5 random seeds with 20 episodes per data point. LTL2Action and GCRL-LTL curves taken from (Qiu, Mao, and Zhu 2024). Discounted reward computed using $\gamma = 0.998$ to maintain consistency.

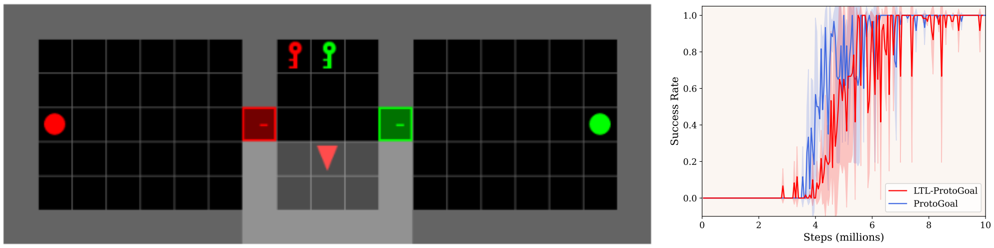


Figure 4: Rendering and success rate for the unlock door task in Minigrid.

GCRL-LTL (Qiu, Mao, and Zhu 2024) learns a goal-conditioned policy for reaching each proposition and uses a high-level planner to reach zones while dynamically avoiding zones by enumerating the Q functions. On the other hand, LTL2Action (Vaezipoor et al. 2021) learns a graph neural network encoder for the LTL structure and relies on that to generalize to new LTL tasks.

Our approach took the middle ground of learning an edge-conditioned policy, which eliminates the need to enumerate all Q functions for dynamic zone avoidance in GCRL-LTL but is still able to take advantage of the automaton. This, along with our exploration algorithm, allows our framework to achieve a far better performance and sample efficiency than both of the baselines on this set of unseen tasks. The results further show that with our current framework, we can build a good coverage of LTL tasks in our policy, even to unseen tasks, which allows us to more effectively explore this LTL task space.

4.2 Minigrid

Next, we move to a sparse-reward image-observation Minigrid environment. In this environment, there are two locked doors and two keys, and the agent's task is to pick up both the red key and the green key and unlock two doors. The agent must also learn to drop the key to pick up the second key, as its inventory can only hold one item. The observation space is the RGB image. The action space is discrete, consisting of `Forward`, `Backward`, `TurnLeft`, `TurnRight`, `PickUp`, `Dropoff`, and `Toggle`. The agent has access to the set of propositions indicating which object it's facing, which object it's holding, and whether the door is unlocked. Figure 4 shows a rendering of the environment.

In this environment, the reward is sparse, and no reward shaping or policy sketch is provided. We compare our framework to the baseline non-LTL protogoal RL (Bagaria and Schaul 2023).

Results show that our new LTL+protogoal model is able to roughly achieve a similar performance as protogoal RL. We suspect the task is too easy to see the benefit of the temporal logic exploration. If the domain included more complex temporally extended tasks or implicit avoidance requirements, our method might perform better than plain goal-based Protogoal exploration.

5 Conclusion

In this paper, we introduced a novel way of RL exploration using the LTL task space. We developed a way to encode automaton edges and train a joint goal-conditioned RL policy to traverse the edges in the DFA. We also presented a way to estimate the weights of the edges, allowing us to find a path through the DFA to solve LTL tasks. Lastly, we introduced a way to actively sample LTL formulas that is most likely to lead to learning progress.

Our framework is able to beat all of the LTL-conditioned RL baselines. and can match the performance of the state-of-the-art baseline, proto-goal RL, showing the superior sample efficiency of our method and representation of the LTL task.

Future work includes testing the algorithm on more complex tasks and allowing the LTL generator to generate more diverse LTL formulas.

References

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Pieter Abbeel, O.; and Zaremba, W. 2017. Hindsight experience replay. *Advances in neural information processing systems*, 30.

Araki, B.; Li, X.; Vodrahalli, K.; DeCastro, J.; Fry, M.; and Rus, D. 2021. The logical options framework. In *International Conference on Machine Learning*, 307–317. PMLR.

Bagaria, A.; and Schaul, T. 2023. Scaling goal-based exploration via pruning proto-goals. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 3451–3460.

Bagaria, A.; Senthil, J. K.; and Konidaris, G. 2021. Skill discovery for exploration and planning using deep skill graphs. In *International Conference on Machine Learning*, 521–531. PMLR.

Barreto, A.; Borsa, D.; Hou, S.; Comanici, G.; Aygün, E.; Hamel, P.; Toyama, D.; Mourad, S.; Silver, D.; Precup, D.; et al. 2019. The option keyboard: Combining skills in reinforcement learning. *Advances in Neural Information Processing Systems*, 32.

Chevalier-Boisvert, M.; Dai, B.; Towers, M.; de Lazcano, R.; Willems, L.; Lahlou, S.; Pal, S.; Castro, P. S.; and Terry, J. 2023. Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. *CoRR*, abs/2306.13831.

Dayan, P.; and Hinton, G. E. 1992. Feudal reinforcement learning. *Advances in neural information processing systems*, 5.

Duret-Lutz, A.; Renault, E.; Colange, M.; Renkin, F.; Aisse, A. G.; Schlehuber-Caissier, P.; Medioni, T.; Martin, A.; Dubois, J.; Gillard, C.; and Lauko, H. 2022. 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*, 174–187. Springer.

Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K. O.; and Clune, J. 2019. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.

Greenman, B.; Prasad, S.; Di Stasio, A.; Zhu, S.; De Giacomo, G.; Krishnamurthi, S.; Montali, M.; Nelson, T.; and Zizyte, M. 2024. Misconceptions in Finite-Trace and Infinite-Trace Linear Temporal Logic. In *International Symposium on Formal Methods*.

Javed, K.; and Sutton, R. S. 2024. The Big World Hypothesis and its Ramifications for Artificial Intelligence. In *Finding the Frame: An RLC Workshop for Examining Conceptual Frameworks*.

Jinnai, Y.; Park, J. W.; Machado, M. C.; and Konidaris, G. 2020. Exploration in reinforcement learning with deep covering options. In *International Conference on Learning Representations*.

Jothimurugan, K.; Alur, R.; and Bastani, O. 2019. A composable specification language for reinforcement learning tasks. *Advances in Neural Information Processing Systems*, 32.

Jothimurugan, K.; Bansal, S.; Bastani, O.; and Alur, R. 2021. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems*, 34: 10026–10039.

Kaelbling, L. P. 1993. Learning to achieve goals. In *IJCAI*, volume 2, 1094–8. Citeseer.

Kaplan, F.; and Oudeyer, P. 2003. Maximizing Learning Progress: An Internal Reward System for Development. In Pierre, S.; Barbeau, M.; and Kranakis, E., eds., *Ad-Hoc, Mobile, and Wireless Networks, Second International Conference, ADHOC-NOW 2003 Montreal, Canada, October 8-10, 2003, Proceedings*, volume 2865 of *Lecture Notes in Computer Science*, 259–270. Springer.

Lacerda, B.; Parker, D.; and Hawes, N. 2015. Optimal Policy Generation for Partially Satisfiable Co-Safe LTL Specifications. In *IJCAI*, volume 15, 1587–1593. Citeseer.

Lagoudakis, M. G.; and Parr, R. 2003. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4: 1107–1149.

Littman, M. L.; Topcu, U.; Fu, J.; Isbell, C.; Wen, M.; and MacGlashan, J. 2017. Environment-independent task specifications via GLTL. *arXiv preprint arXiv:1704.04341*.

Liu, J. X.; Shah, A.; Rosen, E.; Jia, M.; Konidaris, G.; and Tellex, S. 2024. LTL-Transfer: Skill Transfer for Temporal Task Specification. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*.

Pitis, S.; Chan, H.; Zhao, S.; Stadie, B.; and Ba, J. 2020. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In *International Conference on Machine Learning*, 7750–7761. PMLR.

Pnueli, A. 1977. The temporal logic of programs. In *18th annual symposium on foundations of computer science (sfcs 1977)*, 46–57. ieee.

Pong, V. H.; Dalal, M.; Lin, S.; Nair, A.; Bahl, S.; and Levine, S. 2019. Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*.

Qiu, W.; Mao, W.; and Zhu, H. 2024. Instructing Goal-Conditioned Reinforcement Learning Agents with Temporal Logic Objectives. *Advances in Neural Information Processing Systems*, 36.

Quartey, B.; Shah, A.; and Konidaris, G. 2023. Exploiting Contextual Structure to Generate Useful Auxiliary Tasks. In *NeurIPS 2023 Workshop on Generalization in Planning*.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal Value Function Approximators. In Bach, F. R.; and Blei, D. M., eds., *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, 1312–1320. JMLR.org.

Shukla, Y.; Burman, T.; Kulkarni, A. N.; Wright, R.; Velasquez, A.; and Sinapov, J. 2024. Logical Specifications-guided Dynamic Task Sampling for Reinforcement Learning Agents. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 532–540.

Stout, A.; and Barto, A. G. 2010. Competence progress intrinsic motivation. In Kuipers, B.; Shultz, T. R.; Stoytchev, A.; and Yu, C., eds., *2010 IEEE 9th International Conference on Development and Learning, ICDL 2010, Ann Arbor, MI, USA, August 18-21, 2010*, 257–262. IEEE.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S.; Modayil, J.; Delp, M.; Degris, T.; Pilarski, P. M.; White, A.; and Precup, D. 2011. Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In Sonenberg, L.; Stone, P.; Tumer, K.; and Yolum, P., eds., *10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, May 2-6, 2011, Volume 1-3*, 761–768. IFAAMAS.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211.

Tasse, G. N.; Jarvis, D.; James, S.; and Rosman, B. 2022. Skill machines: Temporal logic composition in reinforcement learning. *arXiv preprint arXiv:2205.12532*.

Toro Icarte, R.; Klassen, T.; Valenzano, R.; and McIlraith, S. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, 2107–2116. PMLR.

Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2022. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73: 173–208.

Vaezipoor, P.; Li, A. C.; Icarte, R. A. T.; and Mcilraith, S. A. 2021. Ltl2action: Generalizing ltl instructions for multitask rl. In *International Conference on Machine Learning*, 10497–10508. PMLR.

A Algorithms

A.1 DFA conversion to graph and annotation

This process converts the LTL into DFA and rewrites it into a task graph.

Algorithm 1: Conversion of Buchi Automaton into Task Graph

```
inputs LTL $\phi$
Convert $\phi$ into Finite Buchi Automaton $\mathcal{B} = \{\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_{\text{accept}}\}$
Initialize the Task Graph $G : \{N = \varnothing, E = \varnothing, \Sigma, l_n, l_e, c, n_0, N_{\text{goal}}\}$
for all states $q_i \in \mathcal{Q}$ do
    $N = N \cup \{n_i\}$
    if $q_i \in \mathcal{Q}_{\text{accept}}$ then
        $N_{\text{goal}} \leftarrow N_{\text{goal}} \cup \{n_i\}$
    if $q_i$ is the initial state $q_0$ then
        $l_n(n_i) \leftarrow 1$
    else
        $l_n(n_i) \leftarrow$ False
        for all edges $(q_k, \psi, q_j)$ in $\mathcal{B}$ leading into $q_j$ do
            $l_n(n_i) \leftarrow l_n(n_i) \vee \psi$
for all edges $(q_i, \psi, q_j)$ in $\mathcal{B}$ such that $i \neq j$ do
    Find the corresponding self-edge for the source node $(q_i, \psi', q_i)$.
    if self-edge does not exist then
        continue ▷ No self-edge exists, impossible for RL to guarantee to solve
    $E \leftarrow E \cup (v_i, v_j)$
    Convert the transition condition $\psi$ and $\psi_{\text{self}}$ into disjunctive normal form. (or of and).
    Convert $\bar{\psi}$ into DNF and split ”or” into individual conjunctive clauses $\Psi_{\text{self}} = \{\bar{\psi}_1, \bar{\psi}_2, ...\}$
    Convert $\psi$ into DNF and split ”or” into individual conjunctive clauses $\Psi_{\text{out}} = \{\psi_1, \psi_2, ...\}$
    for each $\bar{\psi}_k \in \Psi_{\text{self}}$ do
        for each $\psi_l \in \Psi_{\text{out}}$ do
            $g \leftarrow$ CONVERT_REACH_MAINTAIN_GOAL$(\mathcal{B}, \bar{\psi}_k, \psi_l)$
            $l_e(e_{ij}) \leftarrow l_e(e_{ij}) \vee g$
return $\{N, E, \Sigma, l_n, l_e, c, n_0, N_{\text{goal}}\}$.
```

A.2 Task graph annotation

This procedure labels the task graph with cost and samples around 10 the highest score task graph as candidate.

Algorithm 2: Annotation of the task graphs with cost

```
inputs Task graph $\{N, E, \Sigma, l_n, l_e, c, n_0, N_{\text{goal}}\}$, value function $v_{\text{seek}} : S \times \mathcal{G} \to [0, 1]$, mapping from abstract boolean formula to a set of concrete states $S_\psi : \Sigma \to S$

Remove edges $e$ where $g = l_e(e)$ is globally implausible
Remove all nodes $n$ with no path from the initial state $q_0$.
for all nodes $n_i$ in the task graph do
    $\psi_s = l_n(n_i)$
    $S_i \leftarrow S_\psi$
for all edges $e_{ij}$ do                ▷ Assign value and cost to edges
    $g \leftarrow l_e(e_{ij})$
    $v_{e_{ij}} \leftarrow \mathbb{E}_{s \sim S_i}[v_{\text{seek}}(s, g)]$
    $u_{e_{ij}} \leftarrow \text{novel}(g) + R(g)$        ▷ utility of the edge
    $c(e_{ij}) \leftarrow -\log(v_{e_{ij}} + u_{e_{ij}})$     ▷ weight of the edge
Remove all edges with $v_\psi$ unseen, uncontrollable, or $v_{\psi'} <$ threshold.
return the updated graph $\{N, E, \Sigma, \mathcal{G}, \sigma, n_0, N_{\text{goal}}\}$
```

B LTL sampling algorithm

LTLs are sampled by first sampling states to reach, then filling in the LTL template.

Algorithm 3: Sample LTL

```
inputs list of plausible goals $g$, their expected value functions $v_{\text{seek}}(g)$, and novel$(g)$
hyperparameters maximum novelty $novelty_{\text{max}}$, maximum number of goals $l_{\text{max}}$
Sampled LTL $\phi \leftarrow$ null
while not is_plausible($\phi$) do
    $G = \{\}$ ▷ Set of goals to be used in LTL construction
    $g_{\text{last}} = \varnothing$
    while $\sum_{g_i \in G}$ novel$(g_i) <$ novel$_{\text{max}}$ and $|G| < l_{\text{max}}$ do
        Sample goal $g$ based on novelty.
        $G \leftarrow G \cup \{g_{\text{next}}\}$
    $\phi =$ construct_LTL($G$)
return $\phi$
```

B.1 LTL Generation templates

- $\mathbf{F}(p_1)$
- $\mathbf{F}(p_1 \wedge \mathbf{F}(p_2))$
- $\mathbf{F}(p_1 \wedge \mathbf{F}(p_2 \wedge \mathbf{F}(p_3)))$
- $\mathbf{F}(p_1 \wedge \mathbf{F}(p_2 \wedge \mathbf{F}(p_3 \wedge \mathbf{F}(p_4))))$
- $\mathbf{F}(p_1 \wedge \mathbf{F}(p_2 \wedge \mathbf{F}(p_3 \wedge \mathbf{F}(p_4 \wedge \mathbf{F}(p_5)))))$
- $\mathbf{F}(p_1 \wedge \mathbf{XF}(p_2))$
- $\mathbf{F}(p_1 \wedge \mathbf{XF}(p_2 \wedge \mathbf{F}(p_3)))$
- $\mathbf{F}(p_1 \wedge \mathbf{XF}(p_2 \wedge \mathbf{XF}(p_3 \wedge \mathbf{XF}(p_4))))$
- $\mathbf{F}(p_1 \wedge \mathbf{XF}(p_2 \wedge \mathbf{XF}(p_3 \wedge \mathbf{XF}(p_4 \wedge \mathbf{XF}(p_5)))))$
- $\neg p_2 \,\mathbf{U}\, p_1 \wedge \mathbf{F}(p_2)$
- $\neg p_2 \,\mathbf{U}\, p_1 \wedge \neg p_3 \,\mathbf{U}\, p_2 \wedge \mathbf{F}(p_3)$
- $\neg p_2 \,\mathbf{U}\, p_1 \wedge \neg p_3 \,\mathbf{U}\, p_2 \wedge \neg p_4 \,\mathbf{U}\, p_3 \wedge \mathbf{F}(p_4)$
- $\neg p_2 \,\mathbf{U}\, p_1 \wedge \neg p_3 \,\mathbf{U}\, p_2 \wedge \neg p_4 \,\mathbf{U}\, p_3 \wedge \neg p_5 \,\mathbf{U}\, p_4 \wedge \mathbf{F}(p_5)$