Structured Exploration in Reinforcement Learning by Hypothesizing Linear Temporal Logic Formulas

Yichen Wei[1], **Xiaochen Li**[1], **Jason Xinyu Liu**[1], **Naman Shah**[1], **Benedict Quartey**[1]
George Konidaris[1], **Stefanie Tellex**[1], **Akhil Bagaria**[2*]
[1]Brown University [2]Amazon
`{yichen_wei, xiaochen_li, xinyu_liu, naman_shah, benedict_quartey}@brown.edu`
`{gdk, stefie10}@cs.brown.edu`
`akhilbg@amazon.com`

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 [1]. Recent work like proto-goal RL [2] 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) [3] have proven to be a powerful abstraction for temporal structure in RL because it is compositional and has unambiguous semantics [4, 5]. 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[5, 6], 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 pre-trained policies that can be used to compose to solve LTL tasks [7, 8]. 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 [9].

[*]Work done while at Brown University

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 [2]. 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 [10] to achieve.

We evaluate our method via two tests. The first test evaluates the agent's ability to solve a set of pre-defined 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 [11].

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 $\gamma_g : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ [10]. 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 [12]; notably, **Hindsight Experience Replay (HER)** [13] 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 [14, 15] 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 [16, 17, 18, 19]: 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 [20]) 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 [13]). **Proto-goal RL** [2] 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 [2] 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 [2], **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.

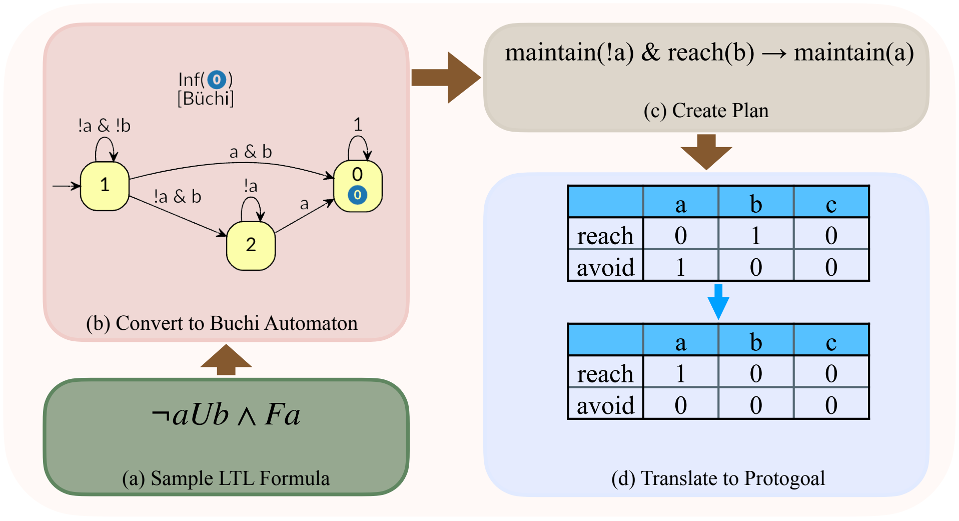


Figure 1: **Overview of our approach.** First a coarse policy outputs an LTL formula, which is then converted into a deterministic finite automaton (DFA). Planning on the DFA results in a sequence of goals, which are pursued one-at-a-time by a low-level goal-conditioned policy.

Linear Temporal Logic and Buchi Automaton. Linear Temporal Logic [3] 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 [4]. In this work, we consider a subset of LTLs defined over a finite time horizon called *co-safe LTL*. Following [21], we define the grammar of co-safe LTL formulas as:

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

where $\alpha \in AP$ 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 $\mathcal{Q} \times \Sigma \to \mathcal{Q}$, 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 [4]. Numerous frameworks like Reward Machines [22, 5] and SPECTRL [23] 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 [24], and requires domain knowledge. On the other hand, works like Logic Options Framework[25], LTL-Transfer [26], GCRL-LTL[7] and Skill Machine[8] 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 [9, 27, 28].

3 LTL Guided Exploration

We introduce our approach for exploiting LTL structure to aid RL exploration. Like prior goal-conditioned RL work, the agent has access to a rich abstract goal space and a labeling function $L : \mathcal{S} \rightarrow \mathcal{G}$, which maps the low-level state space to the discrete goal space. We further augment the MDP with an LTL task space, Φ, constructed using the atomic goals in $\mathcal{G}$ 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 represent 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 3.3).

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` [29]. Each edge in the DFA represents boolean formula, which can be converted into disjunctive normal form (`or` of `ands`). For each edge that contains `or`, we split it into multiple edges so that all edges in the resulting DFA only contains conjunctions (i.e., `ands`) and negations that are applied to atomic propositions.

We condition the low-level RL policy $\pi(s|g)$ on the edges of the automaton. 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 reach goals while obeying constraints.

3.1.1 Representing automaton edges in binary proto-goal space

We want our protogoal space to be expressive enough to cover edges in most LTL formulas and LTL-generated automatons, while being compact enough so that it can be reused. Prior work has demonstrated that each self-, and out-edge in a Buchi Automaton can be mapped to an option in RL [26]. Given a planned path through the automaton, the policy should follow the planned path as long as we ensure the edge-conditioned low-level policy only takes the self-edge or the planned out-edge at each state.

We consider each individual 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 consider 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.

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

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 will contain `maintain`(a), `maintain`$(\neg a)$, `reach`(a), and `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. $\texttt{reach}(a) + \texttt{maintain}(a) + \texttt{reach}(\neg a) + \texttt{maintain}(\neg a)$

At run time, if an edge in the LTL contains unsupported goals, they can be deemed implausible and removed from the graph. As we expand the set of goals to include not just reaching goals but also maintaining and avoiding goals, the task space we're exploring becomes more expressive, but the exploration and goal-tracking cost is also growing. Creating a 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.

3.1.2 Assigning reward for low-level edge-conditioned RL policy

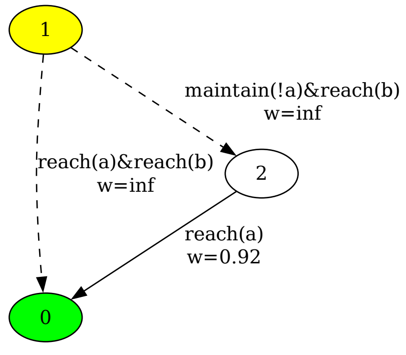
Given the reach/maintain information for each atomic proposition, We can assign rewards to each requirement. For "maintain" goals, we want the policy to maintain the truth value of the proposition until progression to the next node. Hence, we assign a negative reward and terminate if the agent fails to maintain the constraint and a zero reward otherwise. For "reach" goals, we want the policy to try to change the propositions to the desired truth value. So, a positive reward and termination are assigned when the agent reaches the goal, and zero reward when it fails to do so.

We use the following reward function to give reward to an edge-conditioned RL policy, terminating when the reward is non-zero:

$$R(s, s') = \begin{cases} 1 & \text{if all reach goals are satisfied and none of the maintain goals is violated} \\ -1 & \text{if any maintain goals is violated} \\ 0 & \text{otherwise} \end{cases}$$

3.2 Planning through the high-level automaton

With the correspondence between the DFA and the proto-goal space defined, we now need to specify how to plan a path on a high level through the automaton. Conveniently, we convert the DFA into nodes connected by proto-goals and treat it as a task graph (Figure 2). This creates a graph of proto-goals representing the steps required to reach the goal.



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

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

However, this graph is insufficient to create a plan to solve the LTL task. The first complication is that edges in the graph might be implausible (unreachable and/or uncontrollable). For example, in Figure 2, the edge $\texttt{reach}$(a) & $\texttt{reach}$(b) might never be plausible if there's no overlap between the two zones and thus both can't be true at the same time. Those edges that are implausible will have to be removed from the DFA. In addition to pruning edges, traversing through the graph requires finding the shortest path from the starting state and the accepting state. So, it is also crucial to know the cost of the agent taking each edge in the graph.

All of this requires effective management of the set of relevant edges and an estimate of the agent's current capabilities and environment rules.

3.2.1 Goal Pruning

Following proto-goal RL [2], 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 [30] estimated using two iterations of LSPI [31] with the following cumulants [2]:

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

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.

3.2.2 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 3.2.1.

3.2.3 Estimating edge weights and path-finding through the DFA

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 [32] to measure the value of the policy traversing from q to q' through edge representing goal g:

$$v(q, g, q') = \mathbb{E}_{\psi_{\text{in}} \sim q} \left[\mathbb{E}_{s \sim \gamma(\psi_{\text{in}})} [V_{\text{seek}}(s, g)] \right]$$

where ψ_{in} represents the abstract state in the previous state q, and $\gamma(\psi)$ is the function that returns the set of stored concrete states.

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 [33, 7]. 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.

3.3 Generating desirable LTL tasks to explore

Now that we can represent the LTL task and solve it using goal-conditioned RL and planning, we need to find the best LTL that can lead to learning progress. Similar to the proto-goal framework, we use the same simple count-based novelty metric for the 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 [34], and Minigrid [35].

4.1 LTL-conditioned RL

To verify that our framework's capability of covering the task space of LTL, we first benchmark on an LTL-conditioned RL environment, ZoneENV [34]. 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 proposition, each representing whether the agent is in each of the zone.

The avoidance task is the hardest amongst the few specified in the original ZoneENV. In this set of task, a sequence of zones must be satisfied while avoiding some other zones. An example LTL in this category is $\neg\text{zone\_Y}\ \mathbf{U}\ (\text{zone\_W} \wedge (\neg\text{zone\_J}\ \ \mathbf{U}\ \text{zone\_R}))$. To achieve this LTL task, the agent will need to first visit the 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. GCRL-LTL [7] learns a goal-conditioned policy for reaching each individual proposition, and uses a high-level planner to reach zones while dynamically avoiding zones by enumerating the Q functions. On the other hand, LTL2Action [34] 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 shows that with our current framework, we're able to build a good coverage of LTL tasks, even to unseen tasks, which allows us to more effectively explore in this LTL task space.

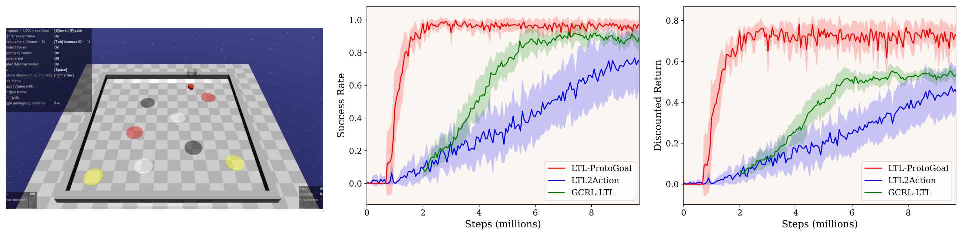


Figure 3: Performance metrics for the avoidance task in ZoneEnv. Success rate (left) and discounted return (right) averaged over 5 random seeds with 20 episodes per data point. LTL2Action and GCRL-LTL curves taken from [7]. Discounted reward computed using $\gamma = 0.998$ to maintain consistency.

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 task of the agent is to pick up both the red key and the green key, and unlock two doors. The agent also needs to learn to drop the key in order 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 [2].

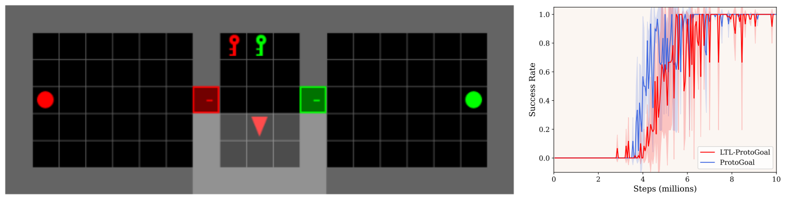


Figure 4: Rendering and success rate for the unlock door task in Minigrid.

Results show that our new LTL+protogoal model is able to roughly achieve a similar performance as protogoal RL. We suspect that 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.

Acknowledgement

This work is supported by ONR under grant award numbers N00014-23-1-2794 and N00014-22-1-2592, NSF under grant award number IIS-1844960, and with funding from Amazon Robotics.

References

[1] K. Javed and R. S. Sutton. The big world hypothesis and its ramifications for artificial intelligence. In *Finding the Frame: An RLC Workshop for Examining Conceptual Frameworks*, 2024. URL https://openreview.net/forum?id=Sv7DazuCn8.

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

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

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

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

[6] Y. Shukla, T. Burman, A. N. Kulkarni, R. Wright, A. Velasquez, and J. Sinapov. Logical specifications-guided dynamic task sampling for reinforcement learning agents. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 532–540, 2024.

[7] W. Qiu, W. Mao, and H. Zhu. Instructing goal-conditioned reinforcement learning agents with temporal logic objectives. *Advances in Neural Information Processing Systems*, 36, 2024.

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

[9] F. Kaplan and P. Oudeyer. Maximizing learning progress: An internal reward system for development. In S. Pierre, M. Barbeau, and E. Kranakis, editors, *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*, pages 259–270. Springer, 2003. doi:10.1007/978-3-540-27833-7_19. URL https://doi.org/10.1007/978-3-540-27833-7_19.

[10] T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In F. R. Bach and D. M. Blei, editors, *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*, pages 1312–1320. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/schaul15.html.

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

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

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

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

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

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

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

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

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

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

[21] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal methods in system design*, 19:291–314, 2001.

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

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

[24] B. Greenman, S. Prasad, A. Di Stasio, S. Zhu, G. De Giacomo, S. Krishnamurthi, M. Montali, T. Nelson, and M. Zizyte. Misconceptions in finite-trace and infinite-trace linear temporal logic. In *International Symposium on Formal Methods*, 2024.

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

[26] J. X. Liu, A. Shah, E. Rosen, M. Jia, G. Konidaris, and S. Tellex. Ltl-transfer: Skill transfer for temporal task specification. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[27] A. Stout and A. G. Barto. Competence progress intrinsic motivation. In B. Kuipers, T. R. Shultz, A. Stoytchev, and C. Yu, editors, *2010 IEEE 9th International Conference on Development and Learning, ICDL 2010, Ann Arbor, MI, USA, August 18-21, 2010*, pages 257–262. IEEE, 2010. doi:10.1109/DEVLRN.2010.5578835. URL `https://doi.org/10.1109/DEVLRN.2010.5578835`.

[28] B. Quartey, A. Shah, and G. Konidaris. Exploiting contextual structure to generate useful auxiliary tasks. In *NeurIPS 2023 Workshop on Generalization in Planning*, 2023. URL `https://openreview.net/forum?id=XwJ44iYC7A`.

[29] A. Duret-Lutz, E. Renault, M. Colange, F. Renkin, A. G. Aisse, P. Schlehuber-Caissier, T. Medioni, A. Martin, J. Dubois, C. Gillard, and H. 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*, pages 174–187. Springer, Aug. 2022. doi:10.1007/978-3-031-13188-2_9.

[30] R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup. Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In L. Sonenberg, P. Stone, K. Tumer, and P. Yolum, editors, *10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, May 2-6, 2011, Volume 1-3*, pages 761–768. IFAAMAS, 2011. URL `http://portal.acm.org/citation.cfm?id=2031726&CFID=54178199&CFTOKEN=61392764`.

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

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

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

[34] P. Vaezipoor, A. C. Li, R. A. T. Icarte, and S. A. Mcilraith. Ltl2action: Generalizing ltl instructions for multi-task rl. In *International Conference on Machine Learning*, pages 10497–10508. PMLR, 2021.

[35] M. Chevalier-Boisvert, B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

A DFA conversion to graph and annotation

This process converts the LTT 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 $\mathcal{B} = \{\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_{\text{accept}}\}$
for all edges $(q_i, \psi, q_j)$ in the automaton do
    Convert the transition condition $\psi$ into disjunctive normal form. (or of ands).
    if edge contains "or" then
        Split $\psi$ into edges with its conjunctive clauses.
Remove all nodes $q$ without self edges $(q_i, \psi, q_i)$, and all edges connecting them.
for all non-self edges $(q_i, \psi_{\text{out}}, q_j)$ where $i \neq j$ do
    Find corresponding self-edge for the starting node $q_i$: $(q_i, \psi, q_i)$
    $g \leftarrow$ CONVERT_REACH_MAINTAIN_GOAL$(\mathcal{B}, \psi_{\text{out}}, \psi_{\text{self}})$
    update edge as $(q_i, g, q_k)$
Remove all self edges $(q_i, \psi, q_i)$.
return $\{\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_{\text{accept}}\}$.
```

The following 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 $\{\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_{\text{accept}}\}$, value function $v_{\text{seek}} : S \times \Sigma \to [0, 1]$, mapping from
high-level state to a set of concrete states $\gamma : \Sigma \to \{s\}$
Remove edges $(q_i, g, q_j)$ where $g$ is globally implausible (unless a sub-component of it is uncon-
trollable and the other part is plausible).
Remove all nodes $q$ with no path from the initial state $q_0$.
for all nodes $q_i$ in the automaton do                          ▷ Match graph state with concrete states
    Initialize all corresponding concrete states for the node $S_i = \varnothing$
    if $q_i$ is the initial state then
        $S_i \leftarrow$ all stored concrete states
    else
        Find all other edges leading into the starting node $q_i$: $\{(q_k, \psi_k, q_i)\}$
        for all $(q_k, g_{\text{prev}}, q_i)$ do          ▷ Use prev edge's formula to determine $q_i$'s abstract state
            $\psi_{\text{prev}} \leftarrow$ corresponding state of $g_{\text{prev}}$          ▷ Estimate abstract state of the node
            $S_i \leftarrow S_i \cup \gamma(\psi_{\text{prev}})$
for all edges $(q_i, g, q_k)$ do                                ▷ Assign value and cost to edges
    $v_g \leftarrow \mathbb{E}_{s \sim S_k}[v_{\text{seek}}(s, g)]$                          ▷ discounted value of the edge
    $u_g \leftarrow \text{novel}(g) + R(g)$                                    ▷ utility of the edge
    $c(q_i, g, q_j) \leftarrow -\log(v_g + u_g)$                           ▷ cost function of the edge
Remove all edges with $v_g$ unseen, uncontrollable, or $v_g <$ threshold.
return the graph $\{\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_{\text{accept}}\}$, edge cost function $c : \mathcal{Q} \times \Sigma \times \mathcal{Q} \to \mathbb{R}^*$
```

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

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