EasyChair Preprint

№ 12418

Grounding Subgoals of Temporal Logic Tasks in Online Reinforcement Learning

Duo Xu and Faramarz Fekri

March 7, 2024

Grounding Subgoals of Temporal Logic Tasks in Online Reinforcement Learning

Duo Xu
Georgia Institute of Technology
Atlanta, GA 30332, USA

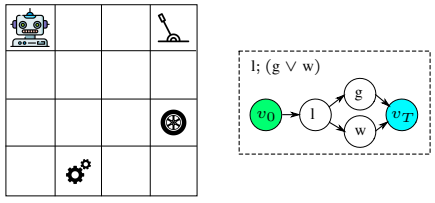
Faramarz Fekri
Georgia Institute of Technology
Atlanta, GA 30332, USA

***Abstract*—Recently, there has been a surge of research papers investigating reinforcement learning (RL) algorithms for solving temporal logic (TL) tasks. However, these algorithms are built upon the assumption of a labeling function which can map raw observations into symbols of subgoals in completing the TL task. In many practical applications, however, this labeling function often is not readily available. In this work, we propose an online RL algorithm, referred to as GSTLO, that takes non-symbolic raw input observations from the collected trajectories and learn to ground the subgoal symbols of TL tasks. In other words, it learns to label important states that are associated with the subgoals in the TL task. Specifically, to associate an important state to one of the subgoals in the TL formula, the RL agent actively explores the environment by collecting trajectories and gradually reconstructs a finite state machine (FSM) of the TL task composed by the discovered important states. Then, by comparing the reconstructed FSM and the ground truth FSM extracted from the task formula, the mapping from the important states to subgoal symbols is obtained, i.e. resulting in the labeling function. In order to discover these important states, GSTLO formulates a contrastive learning objective based on the first-occupancy representations (FR) of collected trajectories. To facilitate the exploration, the first-occupancy feature (FF) of important states is also learned, driving the agent to visit any selected subgoal and complete unseen tasks without further training. The proposed GSTLO algorithm is evaluated on three environments, showing significant improvement over baseline methods.**

***Index Terms*—Reinforcement learning, symbol grounding, temporal logic, exploration**

I. Introduction

Reinforcement learning (RL) algorithms have achieved many successes in recent breakthroughs like human-level video game playing from raw sensory input [22] and mastering complex board games [29]. Different from regular tasks solved by RL algorithms, the temporal logic (TL) task consists of multiple temporally extended subgoals in specified orders and can be transformed into a finite state machine (FSM) [1]. TL tasks have wide applications in the real-world scenarios. For example, consider a service robot on the factory floor which is tasked to fetch a set of components but in different orders depending on the product being assembled. As shown in Figure 1(a), the task of the robot is to fetch lever first and then either wheel or gear, expressed as l;(w∨g) in the TL language. The FSM transformed from the task formula is shown in Figure 1(b). Recently, there have been many works proposing specific RL algorithms for solving TL tasks. However, all of them assume the availability of a labeling function which maps raw observations into symbols of subgoals (e.g., indicating state that contains lever, wheel or gear in this example) for the task completion. In this work, assuming that this labeling function is not available, we propose a framework for grounding subgoal symbols of TL tasks via an online RL algorithm based on non-symbolic observations. By grounding subgoal symbols of TL tasks, we mean that the agent will learn how to align the non-symbolic observation (image in this example) to the important states corresponding to the subgoals in the TL task (in this example, states having l, w and g) such that the agent will know when it arrived at any of those states.



(a) Observation (b) FSM

Fig. 1. (a) TL task Example. The robot is at (0,0) initially. (b) Corresponding FSM for the TL task: l;(g∨w). Letters "l", "g" and "w" are short for lever, gear and wheel, respectively.

In particular, a lot of RL algorithms have been proposed to solve TL tasks in the form of reward machine (RM) [15], [32], [34] and linear temporal logic formulas (LTL) [11], [12], [2], [16]. All these papers assume the availability of such a labeling function which maps and aligns the environmental raw observation into a Boolean interpretation over a set of predefined (subgoal) symbols. However, in many real-world applications, the sequences of agent's observations are often not symbolic but appear 'rendered' in raw data such as images, videos, words and audio, whose symbolic representation is not known. Hence, a labeling function that can enable grounding of subgoal symbols of the TL task is not available. Thus, it is necessary to develop RL algorithms learning to solve TL tasks without assuming such a labeling function. For instance, a real-world service robot (in Figure 1) must acquire the ability to address the specified TL task solely through image-based observations, by exploiting user's episodic feedback on behavior trajectories. This is because obtaining step-by-step feedback from users for every accomplishment of intermediate subgoal symbols in the TL task is prohibitively costly. In this case, by using episodic feedbacks as sparse rewards, a naive idea

for solving this problem is to adopt the POMDP algorithms which use recurrent neural networks (RNN) to build the policy and value function [14], [9], [17].However, due to the lack of the labeling function, the temporal structure of TL task cannot be leveraged by the agent. Hence, such a naive algorithm can suffer from poor learning efficiency and cannot solve complex tasks with long-time horizon.

In this paper, we propose a novel framework for **G**rounding **S**ubgoals of **T**emporal **L**ogic tasks in **O**line reinforcement learning, short for GSTLO. In order to solve TL tasks based on non-symbolic observations without using labeling function, the proposed framework grounds subgoals of the TL task by discovering important states corresponding to subgoals and learning their temporal relationships via online RL. We say a state is important if the state is correspond to one of the subgoals in the TL task formula. Completing the TL task can be decomposed into stages of achieving different subgoals, same as traveling from the initial to final state over the FSM of the task [1]. In every stage, only the first visit to the designated subgoal is meaningful and makes progress towards the task completion. Inspired by this fact, for discovering important states of subgoals, our GSTLO framework first computes the first-occupancy representation (FR) [23] of states in collected trajectories which measures the expected duration until states of subgoals are reached for the *first time* and then formulates an FR-based contrastive learning objective to discover important states. Since the TL task containing temporally extended subgoals may have long time horizon to complete, in order to learn the temporal relations of subgoals efficiently, the GSTLO *actively* collects trajectories and *progressively* reconstructs the finite state machine (FSM) of the task composed by important states discovered by the FR-based contrastive learning. Once the reconstruction is complete, the agent compares the reconstructed FSM with the ground truth FSM transformed from the task formula (composed by subgoal symbols). By this comparison, the agent can learn the labeling function that maps from the important states to subgoal symbols. In order to facilitate the trajectory collection, the first-occupancy features (FF) of discovered important states is also trained, which can drive the agent to reach any discovered important state *for the first time* without further learning and also help the agent to zero-shot generalize to other unseen TL tasks consisting of the same subgoal symbols.

We evaluate GSTLO in three environments, including Letter world, Room, and MiniHack [27]. In these environments, the agent needs to visit different objects in the right temporal orders specified by the task formula. Our evaluations show that GSTLO can outperform baselines on grounding subgoals and efficiency of solving TL tasks. The generalizability of GSTLO is also empirically verified. Ablation study on components of GSTLO framework is conducted as well.

II. Related Works

Recently linear temporal logic (LTL) formulas have been widely used in Reinforcement Learning (RL) to specify temporal logic tasks [20]. Assuming that the LTL task specification is known, some authors develop RL algorithms by compiling the LTL specification into an automaton and leveraging the temporal abstractions provided by the automaton during the task execution [3], [5], [2]. In some other papers, based on known methods for automata induction, authors focus on learning the task machine from traces of symbolic observations and rewards or labels received from the environment [7], [35], [26]. However, all these papers are based on symbolic observation of the agent and do not consider the problem of discovering latent symbols in the traces. Therefore, they are applicable only in environments with symbolic states or non-symbolic environments where a labeling function mapping between the non-symbolic state and a symbolic interpretation is available, also called labeled MDP [12].

However, in many real-world applications the environmental state or observation is not symbolic and the labeling function is not always known a priori. Some papers assume to have an imperfect labeling function, where the predicted symbols can be erroneous or uncertain [19], [13]. They develop robust RL algorithms which are adaptable to mistakes made by the labeling function. But these papers do not address the problem of learning the labeling function to ground symbols of subgoals. A recent paper studies the problem of grounding LTLf formulas in image sequences [33]. However, their method is only applicable to offline problems with static dataset and does not consider the generalizability to other LTL tasks. In addition, authors in [21] propose an algorithm for learning rational subgoals based on dynamic programming. Their approach is based on the availability of the state transition model, which is not feasible in general real-world applications.

III. Preliminaries

A. Reinforcement Learning

Reinforcement learning (RL) is a framework for learning the strategy of selecting actions in an environment in order to maximize the collected rewards over time [30]. The problems addressed by RL can be formalized as Markov decision processes (MDP). The environmental MDP, with which the agent is interacting for any tasks, is defined as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma, \mathcal{S}_0 \rangle$, where $\mathcal{S}$ is a finite set of environment states, $\mathcal{A}$ is a finite set of agent actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a probabilistic transition function, $R : \mathcal{S} \times \mathcal{A} \to [R_{\text{min}}, R_{\text{max}}]$ is a reward function with $R_{\text{min}}, R_{\text{max}} \in \mathbb{R}$ and $\gamma \in [0, 1)$ is a discount factor. Note that $\mathcal{S}_0$ is the set of initial states where the agent starts in every episode, and $S_0 : s_0 \sim \mathcal{S}_0$ is a distribution of initial states.

In this work, we equip the environment MDP with a finite set of pre-defined propositions $\mathcal{P}$ and a finite set of pre-defined symbols of TL task subgoals $\mathcal{G} \subset 2^{\mathcal{P}}$, where each symbol $g \in \mathcal{G}$ is described by one or multiple propositions in $\mathcal{P}$. We define a *labeling function* $L : \mathcal{S} \to \mathcal{G}$ that maps a raw state to a subgoal symbol of the TL task. The output of labeling function is the symbolic observation of an environmental state. We define *important states* as the states which contain subgoal symbols, so the output of L with a non-important state as input is empty. For example shown in Figure 1, for the state when

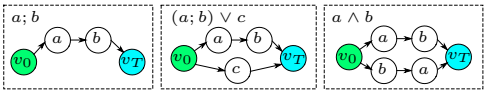


Fig. 2. Examples of TL formulas and their corresponding FSMs. The initial node is v_0 and the accepting (terminal) node is v_T.

the robot is at (0,0), the output of L is empty. However, for the state when the robot is at (0,3), the state is an important state and thus the output of L is "l" for lever. However, the labeling function is not available in our work and the agent has to learn the labeling function and solve temporal logic tasks based on non-symbolic observations.

B. Temporal Logic Specification

The temporal logic tasks used in this work is described by a formal language $\mathcal{TL}$ together with three operators. Syntactically, all subgoal symbols in $\mathcal{G}$ are in $\mathcal{TL}$, and $\forall \varphi_1, \varphi_2 \in \mathcal{TL}$, the expressions $(\varphi_1; \varphi_2)$, $(\varphi_1 \vee \varphi_2)$ and $(\varphi_1 \wedge \varphi_2)$ are all in $\mathcal{TL}$, representing "φ_1 then φ_2", "φ_1 or φ_2" and "φ_1 and φ_2", respectively. Formally, a trajectory of states $\tau = (s_1, \ldots, s_n)$ satisfies a task description φ, written as $\tau \models \varphi$, whenever one of the following holds:

- If φ is a single subgoal $g \in \mathcal{G}$, then the first state of τ must not satisfy g, and instead the last state must satisfy g, which implies that τ has at least 2 states
- If $\varphi = (\varphi_1; \varphi_2)$, then $\exists 0 < j < n$ such that $(s_1, \ldots, s_j) \models \varphi_1$ and $(s_j, \ldots, s_n) \models \varphi_2$, i.e., task φ_1 should be finished before φ_2
- If $\varphi = (\varphi_1 \vee \varphi_2)$, then $\tau \models \varphi_1$ or $\tau \models \varphi_2$, i.e., the agent should either finish φ_1 or φ_2
- If $\varphi = (\varphi_1 \wedge \varphi_2)$, then $\tau \models (\varphi_1; \varphi_2)$ or $\tau \models (\varphi_2; \varphi_1)$, i.e., the agent should finish both φ_1 and φ_2 in any order

Note that the language $\mathcal{TL}$ for specifying tasks here covers LTLf [6] which is a finite fragment of LTL without using always operator $\Box$.

Every task specification $\varphi \in \mathcal{TL}$ can be represented by a non-deterministic finite-state machine (FSM) [21], representing the temporal orderings and branching structures. Each FSM $\mathcal{M}_\varphi$ of task φ is a tuple $(V_\varphi, E_\varphi, I_\varphi, F_\varphi)$ which denote subgoal nodes, edges, the set of initial nodes and the set of accepting (terminal) nodes, respectively. Every node , excluding those in I_φ and F_φ, corresponds to a subgoal symbol in the task specification, and each edge represents a possible transition by completing a subgoal.

There exists a deterministic algorithm for transforming any specification in $\mathcal{TL}$ to a unique FSM [21]. In this work, we only consider the FSMs which do not contain any loops. We assume that in any FSM there is only a single initial and accepting state. If the FSM constructed by transforming the specifications has multiple initial or accepting nodes, we introduce a super initial node v_0 or accepting node v_F to unify them. Several examples of task formulas and transformed FSMs are shown in Figure 2.

C. First-occupancy Representation

In this work, we use first-occupancy representation (FR) for both subgoal grounding and task generalization. FR measures the duration until a policy is expected to reach states for *the first time*, which emphasizes the first occupancy.

Definition 1.[23] For an MDP with finite $\mathcal{S}$, the first-occupancy representation (FR) for a policy π $F^\pi \in [0,1]^{|\mathcal{S}|\times|\mathcal{S}|}$ is given by

$$F^\pi(s, s') := \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \mathbb{1}(s_{t+k} = s', s' \notin \{s_{t:t+k}\}) \middle| s_t = s\right] \quad (1)$$

where $\{s_{t:t+k}\} = \{s_t, s_{t+1}, \ldots, s_{t+k-1}\}$ and $\{s_{t:t+0}\} = \emptyset$. The above indicator function $\mathbb{1}$ equals 1 only when s' first occurs at time $t+k$ since time t. So $F^\pi(s, s')$ gives the expected discount at the time the policy first reaches s' starting from s. It can be shown that there is also a recursive relationship for FR:

$$F^\pi(s, s') = \mathbb{E}_{s_{t+1}\sim p^\pi(\cdot|s)}[\mathbb{1}(s_t = s') + \gamma(1 - \mathbb{1}(s_t = s'))F^\pi(s_{t+1}, s') | s_t = s] \quad (2)$$

To compute an empirical FR based on a given trajectory τ of states, the Monte Carlo FR in τ is defined as below:

$$F^{\text{MC}}(s, s'; \tau) := \sum_{t=1}^{T} \mathbb{1}(s_t = s) \cdot \sum_{k=0}^{T-t} \gamma^k \mathbb{1}(s_{t+k} = s', s' \notin \{s_{t:t+k}\}) \quad (3)$$

where the length of τ is denoted as T and the t-th state in τ is denoted as s_t. When the state space is impractically large or infinite, e.g., non-symbolic, we learn a contrastive representation of states to measure the similarity of two sates, which is used to compute the indicator function in (3).

In order to fast transfer to multiple tasks and realize generalization, we need to define a base feature function of states, i.e., $\phi(\cdot) : \mathcal{S} \to \mathbb{R}^D$, so that a linear combination of base features can predict an immediate reward for a specific task, i.e., $r(s) = \boldsymbol{w}^T \phi(s)$ for some $\boldsymbol{w} \in \mathbb{R}^D$ which is a characteristic of the specific task. Similar as successor feature (SF) [18], we can define a first-occupancy feature (FF) representation $\mathcal{F}^\pi$ to achieve a general policy evaluation and improvement [23]:

$$\begin{aligned} \mathcal{F}_d^\pi(s) &:= \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k \mathbb{1}(\phi_d(s_{t+k}) \geq \theta_d, \{\phi_d(s_{t'}) < \theta_d\}_{t'=t:t+k}) \Big| s_t = s\Big] \\ &= \mathbb{1}(\phi_d(s_t) \geq \theta_d) + \gamma(1 - \mathbb{1}(\phi_d(s_t) \geq \theta_d))\mathbb{E}_{s_{t+1}\sim p^\pi}[\mathcal{F}_d^\pi(s_{t+1})] \quad (4) \end{aligned}$$

where the first line is the formal definition and second line is the recursive definition, θ_d is a threshold, and p^π is the state distribution under policy π. If realized by deep neural networks, $\phi(\cdot)$ can also handle image-based observation where the state space is impractically large.

IV. Methodology

In this work, we propose the GSTLO framework to solve TL tasks based on non-symbolic observations by grounding subgoals of the TL task and then zero-shot generalize the agent to unseen tasks. In the rest of this section, we will first have a general introduction to the GSTLO framework, and then present every component of GSTLO with details.

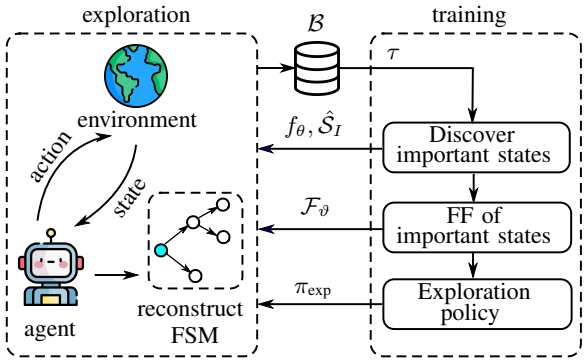


Fig. 3. Diagram of the GSTLO framework. In the exploration part, every trajectory is collected by both $\mathcal{F}_\vartheta$ (FF of the important states) and π_{exp} (the exploration policy). $\mathcal{B}$ represents the positive and negative buffers with τ denoting trajectories. f_θ is the state representation function, and $\hat{\mathcal{S}}_I$ is the set of discovered important states. By comparing current state and those in $\hat{\mathcal{S}}_I$ based on f_θ, the agent can know the achievement of any important state. The FSM is reconstructed by $\hat{\mathcal{S}}_I$.

A. Framework

Grounding subgoals of TL task is essentially to learn the labeling function $L : \mathcal{S} \to \mathcal{G}$ which is the mapping from non-symbolic states in $\mathcal{S}$ to subgoal symbols in $\mathcal{G}$. As formally defined in Section III-A, whenever they constitutes an important state, otherwise map it to empty symbol. Since the TL task may consist of multiple temporally extended subgoals and have long time-horizon to complete, grounding all the subgoals together via offline supervised learning is difficult and infeasible in general cases. Therefore, the GSTLO framework discovers *important states* corresponding to symbols of subgoals and progressively reconstructs the FSM (denoted as $\hat{\mathcal{M}}_\varphi$) composed by important states via the online RL algorithm. Starting from a single initial node v_0, $\hat{\mathcal{M}}_\varphi$ is gradually expanded by adding nodes represented by discovered important states. Once $\hat{\mathcal{M}}_\varphi$ becomes structurally same as the ground truth FSM $\mathcal{M}_\varphi$ extracted from the task formula, by comparing $\hat{\mathcal{M}}_\varphi$ and $\mathcal{M}_\varphi$, the mapping from important states to the symbols of subgoals can be discovered, which naturally results in the labeling function.

Specifically, in each episode k of GSTLO framework, given the TL task formula φ and the FSM of the task formula $\mathcal{M}_\varphi$, the agent collects a trajectory τ_k by actively interacting with the environment and receives a binary label l_k for the task completion at the end of every episode. The binary label l_k is 1 whenever the trajectory successfully completes the task φ. A trajectory τ is a sequence of state-action pairs, i.e., $\tau = (s_1, a_1, \ldots, s_T)$ with T indicating the last time step. Based on binary labels, the collected trajectories can be classified as positive ($l_k = 1$) trajectories or negative ($l_k = 0$) trajectories, all of which are stored into positive buffer ($\mathcal{B}_P$) and negative buffer ($\mathcal{B}_N$), respectively.

We define the set $\hat{\mathcal{S}}_I$ as an ordered set of discovered important states indicating where subgoals potentially are in the state space, i.e., $\hat{\mathcal{S}}_I \subset \mathcal{S}$. For the k-th state in $\hat{\mathcal{S}}_I$, i.e., $\hat{s}_k$, k is the index for indicating *learned subgoal* and $\hat{s}_k$ is the *important state* associated with the learned subgoal k. Only newly discovered important state not included in $\hat{\mathcal{S}}_I$ will be added to $\hat{\mathcal{S}}_I$, creating an index indicating a newly learned subgoal.

The GSTLO framework consists of exploration part and training part. The diagram of GSTLO is shown in Figure 3. In the exploration part, the agent actively collects trajectories from the environment, and expand the reconstructed FSM by composing important states discovered in the training part. The training part is to discover important states, train the first-occupancy feature (FF) of important states, and train the exploration policy. The FF is trained to enable the agent to generalize to visit any important state in the future without further learning. The exploration policy is used to collect trajectories for subgoal grounding. The detailed algorithm is presented in Appendix D.

B. Exploration

The target of agent's exploration is to actively collect trajectories to reconstruct the FSM $\hat{\mathcal{M}}_\varphi$ composed by discovered important states, which must have the same structure as the task FSM $\mathcal{M}_\varphi$ if every important state is discovered correctly.

In $\hat{\mathcal{M}}_\varphi$, except initial and terminal nodes v_0 and v_T, each node v_n of $\hat{\mathcal{M}}_\varphi$ has the attribute of $\hat{s}_{k_{v_n}}$ indicating the k_{v_n}-th important state in $\hat{\mathcal{S}}_I$. The nodes of $\hat{\mathcal{M}}_\varphi$ are categorized into frontier nodes in $\mathcal{V}_f$ and other nodes in $\mathcal{V}/\mathcal{V}_f$. The frontier node is defined as the node whose next important states for completing task φ are not fully discovered, meaning that the out-going edges of a frontier node are not fully constructed yet. Initially, only the initial node is the frontier node of $\hat{\mathcal{M}}_\varphi$, i.e., $\mathcal{V}_f = \{v_0\}$. With reconstructed FSMs in Figure 4 and 5 as examples, the dashed nodes are in the frontier set $\mathcal{V}_f$.

In order to reconstruct the FSM $\hat{\mathcal{M}}_\varphi$ efficiently, the agent selects a working node v_w from the frontier set $\mathcal{V}_f$ and collects trajectories conditioned on the node v_w, focusing on discovering next important states to visit after reaching the important state on node v_w (i.e., $\hat{s}_{k_{v_w}}$) of the FSM of the task. Specifically, in each episode, guided by the FF of important states $\mathcal{F}_\vartheta$, the agent first sequentially visits important states of nodes along the path from v_0 to v_w on $\hat{\mathcal{M}}_\varphi$. Then, after reaching $\hat{s}_{k_{v_w}}$ of node v_w, the agent uses an exploration policy π_{exp} to continue

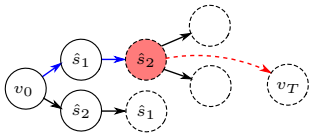


Fig. 4. Collecting a trajectory conditioned on v_w in one episode. The red node is the working node v_w. The dashed nodes are in the frontier set. Blue path: The first half of the trajectory is collected by FF of important states $\mathcal{F}_\vartheta$, visiting $\hat{s}_1$ and $\hat{s}_2$. Red path: The second half is collected by applying the exploration policy π_{exp} till the end of the episode.

collecting the trajectory τ until the end of the episode. This collected trajectory going through important states of node v_w and nodes before v_w on $\hat{\mathcal{M}}_\varphi$ is called a trajectory *conditioned on* v_w. An example of collecting trajectory is shown in Figure 4. Based on these collected trajectories conditioned on v_w, the agent discovers important states next to v_w by using contrastive learning which is introduced with details in Section IV-C1.

When the important states next to that on v_w are fully discovered, the working node v_w will be in the status of fully "discovered". The condition of a working node v_w becoming fully "discovered" is that collecting K more trajectories conditioned on v_w cannot make the agent discover more important states next to v_w. Then, the agent will expand $\hat{\mathcal{M}}_\varphi$ and select a new working node from $\mathcal{V}_f$. Specifically, the node v_w will be removed from the frontier set $\mathcal{V}_f$. Every important state discovered next to node v_w and its index in $\hat{\mathcal{S}}_I$ are used to build a new node connected with v_w. These new nodes are also new frontier nodes added to $\mathcal{V}_f$. After that, a new working node v_w is randomly selected from $\mathcal{V}_f$ by the agent. This is the process how $\hat{\mathcal{M}}_\varphi$ is expanded. An example of the expansion of $\hat{\mathcal{M}}_\varphi$ is presented in Figure 5. The number K in the condition of "fully discovered" is empirically selected and specific to the environment, which is usually small.

C. Training

Based on trajectories collected in the exploration component of the framework, as shown in Figure 3, the training module is responsible of discovering importance states, training the first-occupancy feature (FF) of learned subgoals and the exploration policy. The concepts of Monte Carlo FR (MCFR) and FF are defined in Section III-C. The FF of important states is introduced in the Appendix A.

1) Discovering Important States: **State Representation.** Since the state space considered in this work is non-symbolic and impractically large, the first step of discovering important states is to learn a compressed representation of state which can distinguish different states from each other efficiently. We propose to learn a representation function $f_\theta(\cdot) : \mathcal{S} \to \mathbb{R}^d$ by contrastive unsupervised learning [25], mapping an input state into a d-dimensional vector. Specifically, the representation function f_θ is trained by minimizing the InfoNCE loss [25], with states drawn from both positive and negative buffers.

Contrastive Learning. Building on top of the state representation f_θ, we first define $\tilde{L}_\omega : \mathbb{R}^d \to [0, 1]$ as the importance function. Any state s with value of $\tilde{L}_\omega(f_\theta(s))$ close to 1 after training is regarded as an important state. In GSTLO, the agent discovers important states by formulating the return of each trajectory in terms of $\tilde{L}_\omega$ and comparing the returns of positive and negative trajectories. Specifically, it is to train L_ω with a contrastive learning objective formulated by the MCFR of positive and negative trajectories.

As discussed above, in each training iteration the agent only discovers important states next to the working node v_w on $\hat{\mathcal{M}}_\varphi$. For any trajectory conditioned on node v_w, starting from the important state $\hat{s}_{k_{v_w}}$ of node v_w, the agent computes the MCFR to formulate the trajectory's return. Since the MCFR in (3) is expensive to compute directly, the agent needs several steps to pre-process the collected trajectories. Specifically, discovering important states based on contrastive learning consists of the following three steps:

1) **Selecting Trajectories:** We randomly select positive and negative trajectories conditioned on the current working node v_w from buffers $\mathcal{B}_P$ and $\mathcal{B}_N$. Then, for every selected positive (negative) trajectory, we discard the part before reaching $\hat{s}_{k_{v_w}}$ and store the rest into the set $\mathcal{D}_P$ ($\mathcal{D}_N$).
2) **Computing MCFR:** Note that the first state of every $\tau \in \mathcal{D}_P$ ($\mathcal{D}_N$) is always $\hat{s}_{k_{v_w}}$. Computing MCFR defined in (3) is realized by a *pre-processing* function for any trajectory $\tau \in \mathcal{D}_P$ ($\mathcal{D}_N$), which consists of two steps: 1) In order to compute the outer Σ in (3), any τ is decomposed into $N(\hat{s}_{k_{v_w}}; \tau)$ segments $\{\tau_i'\}$, where $N(s; \tau)$ is the number of occurrence s in τ and every segment τ_i' starts with i-th occurrence of $\hat{s}_{k_{v_w}}$ and ends at one-step before the $i+1$-th occurrence of $\hat{s}_{k_{v_w}}$ in τ or the end of τ; 2) For any segment τ_i', computing the inner Σ in (3) needs to remove repetitive states from τ_i', producing $\tilde{\tau}_i'$, where the similarity of states is evaluated by the cosine similarity of state representations f_θ. Define this two-step pre-processing above as a general function pre_{FR}, i.e., $\{\tilde{\tau}_i'\}_{N(s;\tau)} := \text{pre}_{\text{FR}}(\tau, s)$ with s replacing the state $\hat{s}_{k_{v_w}}$ above. Therefore, computing the MCFR of τ can be simplified as the sum over each pre-processed segment $\tilde{\tau}_i'$:

$$\begin{aligned} F^{\text{MC}}(s, s'; \tau) &= \sum_{\tilde{\tau}_i' \in \text{pre}_{\text{FR}}(\tau; s)} F^{\text{MC}}(s, s'; \tilde{\tau}_i') \\ &= \sum_{\tilde{\tau}_i' \in \text{pre}_{\text{FR}}(\tau; s)} \sum_{t'=1}^{\text{len}(\tilde{\tau}_i')} \gamma^{t'} \mathbb{1}(s_{t'} = s') \end{aligned} \quad (6)$$

where the function $\text{len}(\tau)$ gives the length of trajectory τ. For every $\tau \in \mathcal{D}_P$ ($\mathcal{D}_N$), its pre-processed segments $\{\tilde{\tau}_i'\}$ are stored into the set $\tilde{\mathcal{D}}_P$ ($\tilde{\mathcal{D}}_N$).
3) **Contrastive Objective:** Since the indicator $\mathbb{1}$ in (6) is still intractable to compute across non-symbolic states, we replace it by the state representation f_θ. Then, the return for formulating the contrastive objective can be written as $\sum_{s_t \in \tilde{\tau}} \gamma^t \tilde{L}_\omega(f_\theta(s_t))$ for any $\tilde{\tau} \in \mathcal{D}_P \cup \mathcal{D}_N$. Therefore, based on the pre-processed datasets $\tilde{D}_P$ and $\tilde{D}_N$, the contrastive learning objective for discovering important states next to v_w is expressed as in (5), where the set $\tilde{D}_P$ ($\tilde{D}_N$) is obtained by pre_{FR}. Any state s' with importance value higher than a threshold κ, i.e.,

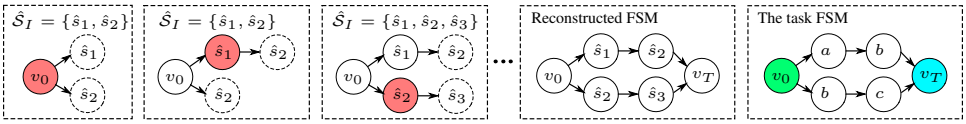
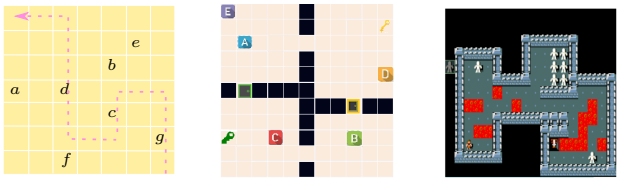


Fig. 5. Examples of reconstructing FSMs. In the left three figures, the red node denotes the working node v_w which is in the status of fully "discovered", and updated $\hat{\mathcal{S}}_I$ is given on the upper left corner. The dashed nodes are in the frontier set $\mathcal{V}_f$. The discovered important state and its index are used as attributes of each node. The fourth figure shows a completely reconstructed FSM. The task FSM in the rightmost figure is the FSM extracted from the task formula, which is composed by subgoal symbols. By comparing the reconstructed FSM and task FSM, the mapping from important states to subgoal symbols can be obtained, yielding the labeling function.

$$\mathcal{L}_{\text{contrast}}(\omega) := \sum_{\tilde{\tau}_0 \sim \tilde{\mathcal{D}}_P, \tilde{\tau}_1 \sim \tilde{\mathcal{D}}_N} \frac{\exp\left(\sum_{t=1}^{\text{len}(\tilde{\tau}_0)} \gamma^t \tilde{L}_\omega(f_\theta(\tilde{\tau}_0[t]))\right)}{\exp\left(\sum_{t=1}^{\text{len}(\tilde{\tau}_0)} \gamma^t \tilde{L}_\omega(f_\theta(\tilde{\tau}_0[t]))\right) + \exp\left(\sum_{t=1}^{\text{len}(\tilde{\tau}_1)} \gamma^t \tilde{L}_\omega(f_\theta(\tilde{\tau}_1[t]))\right)} \tag{5}$$



(a) Letter (b) Room (c) MiniHack

Fig. 6. Environments.

$\tilde{L}_\omega(f_\theta(s')) \geq \kappa$, is chosen as an important state next to v_w. Then, if state s' does not exist in the set $\hat{\mathcal{S}}_I$, s' will be added into $\hat{\mathcal{S}}_I$ with a new index $|\hat{\mathcal{S}}_I| + 1$ assigned and a newly learned subgoal $|\hat{\mathcal{S}}| + 1$ is also created.

2) Exploration Policy: The exploration policy π_{exp} is realized by a GRU-based policy model. Each action is history dependent and drawn from the action distribution at the output of the policy model. The action selection of π_{exp} is conditioned on both current state and a hidden state summarizing previous states and actions. The policy π_{exp} is trained by the recurrent PPO algorithm [8], [24] which extends the classical PPO [28] into POMDP domain. The reward for training π_{exp} is just the binary label of task satisfaction given at the end of the trajectory. This sparse and weak reward signal cannot make the agent directly solve the task, but can encourage the agent to visit states relatively closer to subgoals, improving the sample efficiency of subgoal grounding.

V. Experiments

In this section, we first introduce the basic settings of experiments, including environments and baselines. Then, we present the experimental results and analysis. In this work, the proposed framework is evaluated in three environments, including Letter, Room and MiniHack, as shown in Figure 6. The details of environments are introduced in Appendix B. Implementation details and other results are included in Appendix.

A. Baselines

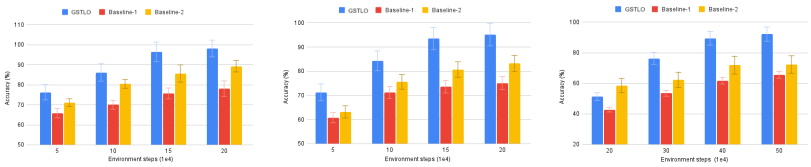
Baseline-1. Contrastive learning has been widely used to learn important states or subgoals for solving a designated task or maximizing the rewards in previous papers [36], [4]However, these papers did not consider the temporal ordering of reaching detected subgoals. By mimicking the methods of these papers, the first baseline directly compares positive and negative trajectories by extracting subgoals which can differentiate positive trajectories from negative ones. The contrastive objective is same as (5) except that the pre-processing function for computing FR of trajectories is omitted, so that FR is discarded in Baseline-1. The state representation f_θ and importance function $\tilde{L}_\omega$ are learned same as those in GSTLO. This baseline is designed to show the effect of FR in GSTLO.

Baseline-2. Recently authors in [33] propose a neural architecture consisting of a trainable convolutional neural network (CNN) and a non-trainable finite state automaton (FSA) which is a sub-class of FSM. Here the CNN predicts symbols given the input image and the FSA, which is derived from the LTLf task formula, describes the automaton state transitions and their conditions. In Baseline-2, subgoals are represented by pre-defined symbols and they are grounded by training the neural architecture to predict the binary label (positive or negative) of input trajectory. The neural architecture are trained whenever K more trajectories are collected by the exploration policy π_{exp}. In baseline-2, the processes of computing FR and learning FF of learned subgoals are discarded, and no FSM or FSA is reconstructed, since the neural architecture is designed to ground all the subgoals together.

B. Results

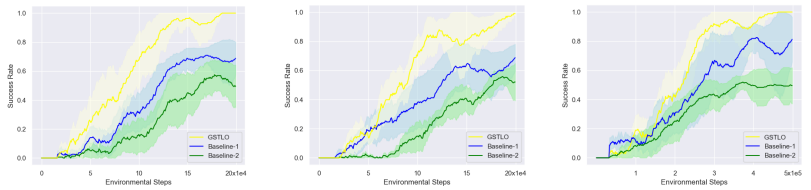
We conduct three sets of experiments to compare the performance of GSTLO and baselines from different aspects, including the accuracy of grounding subgoals, the success rate of solving TL tasks and the generalizability to other TL tasks unseen in the training. More results are included in the Appendix C.

The accuracy of grounding subgoals is defined as the ratio of subgoals in $\mathcal{G}$ whose corresponding states are correctly discovered in the state space $\mathcal{S}$. In every environment, the performance is the average of 6 randomly generated tasks. The performance comparison of accuracy versus the environmental samples is shown in Figure 7. For all the evaluations in this



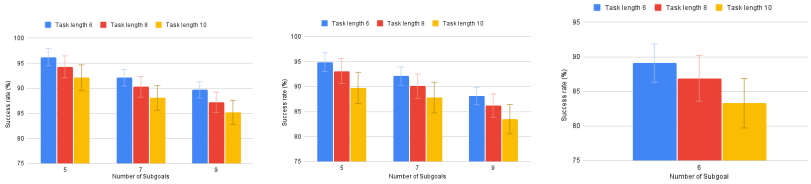
(a) Letter (b) Room (c) MiniHack

Fig. 7. Comparison of accuracy of learned subgoals (discovered important states).



(a) Letter (b) Room (c) MiniHack

Fig. 8. Comparison of success rate of solving the given task.



(a) Letter (b) Room (c) MiniHack

Fig. 9. Comparison of generalizability across different number of subgoals and task lengths. In MiniHack environment, the number of subgoal is fixed to be 6.

section, the map in the letter environment has the width of $m = 9$ with $k = 7$ different letters (subgoals), and the map in the room environment has the width of 11 with 5 subgoals. Every data point is the average of 5 random seeds. In addition, we compare the GSTLO framework with baselines on the success rate of completing TL tasks, as shown in Figure 8. The tasks evaluated here are all training tasks, and every curve is the average of 5 random seeds with standard deviation shown in the shadow. Finally, the generalizability of GSTLO is compared with baselines in Figure 9. The metric for comparing generalizability is the success rate of completing 10 randomly generated tasks unseen in the training which are composed by same subgoals in $\mathcal{G}$.

We can see that GSTLO outperforms baselines in all three experiments. In our method, the progressive reconstruction of FSM based on the FR of trajectories grounds temporally extended subgoals one-by-one, transforming the non-Markovian problem into a Markovian one. Instead of grounding all the subgoals at once, in each training iteration, it only focuses on discovering important states of subgoals next to a previously grounded subgoal. Based on the FF of discovered important states of grounded subgoals, the agent can visit any grounded subgoal on FSM efficiently without further training. This can make the agent actively collect trajectories based on the current progress of subgoal grounding. Additionally, it can also help the agent generalize to any unseen tasks composed by same subgoals grounded in the training.

Baseline-1 performs the worst, since it ignores the non-Markovian property of solving temporal logic (TL) tasks in the non-symbolic state space. This method treats every subgoal equally and ignore their temporal orders, where the importance function can be distracted by redundant visits to important states of subgoal. So, it cannot ground many subgoals correctly. In baseline-2, the non-trainable FSA derived from the task formula is non-differentiable, which can make the CNN part difficult to be trained. Besides, since TL tasks can have long time-horizon to complete, some subgoals may be visited by few or even no trajectories, especially in the early learning stage. This can make the training data imbalanced and subgoals which are never or rarely visited can not be correctly grounded. However, the active collection of trajectories in GSTLO resolves this problem.

VI. Conclusion

In this work, we propose a framework, short for GSTLO, for grounding the subgoal symbols and solving the TL task based on non-symbolic observations. By discovering important states based on contrastive learning, the agent progressively reconstructs the FSM composed by important states. By comparing the reconstructed FSM and the ground truth FSM extracted from the task formula, the agent can learn the labeling function and ground subgoal symbols.

REFERENCES

[1] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[2] 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)*, pages 10349–10355. IEEE, 2020.

[3] Alberto Camacho, Rodrigo Toro Icarte, Toryn Q Klassen, Richard Anthony Valenzano, and Sheila A McIlraith. Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *IJCAI*, volume 19, pages 6065–6073, 2019.

[4] Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, et al. Open problems and fundamental limitations of reinforcement learning from human feedback. *arXiv preprint arXiv:2307.15217*, 2023.

[5] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. In *Proceedings of the international conference on automated planning and scheduling*, volume 29, pages 128–136, 2019.

[6] Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 854–860. Association for Computing Machinery, 2013.

[7] Maor Gaon and Ronen Brafman. Reinforcement learning with non-markovian rewards. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3980–3987, 2020.

[8] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. In *International Conference on Learning Representations*, 2020.

[9] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31, 2018.

[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[11] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Omega-regular objectives in model-free reinforcement learning. In *International conference on tools and algorithms for the construction and analysis of systems*, pages 395–412. Springer, 2019.

[12] Mohammadhosein Hasanbeig, Yiannis Kantaros, Alessandro Abate, Daniel Kroening, George J Pappas, and Insup Lee. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In *2019 IEEE 58th conference on decision and control (CDC)*, pages 5338–5343. IEEE, 2019.

[13] Wataru Hatanaka, Ryota Yamashina, and Takamitsu Matsubara. Reinforcement learning of action and query policies with ltl instructions under uncertain event detector. *IEEE Robotics and Automation Letters*, 2023.

[14] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

[15] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila 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.

[16] Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems*, 34:10026–10039, 2021.

[17] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[18] Tejas D Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

[19] Andrew C Li, Zizhao Chen, Pashootan Vaezipoor, Toryn Q Klassen, Rodrigo Toro Icarte, and Sheila A McIlraith. Noisy symbolic abstractions for deep rl: A case study with reward machines. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

[20] Michael L Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-independent task specifications via gltl. *arXiv preprint arXiv:1704.04341*, 2017.

[21] Zhezheng Luo, Jiayuan Mao, Jiajun Wu, Tomás Lozano-Pérez, Joshua B Tenenbaum, and Leslie Pack Kaelbling. Learning rational subgoals from demonstrations and instructions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12068–12078, 2023.

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[23] Ted Moskovitz, Spencer R Wilson, and Maneesh Sahani. A first-occupancy representation for reinforcement learning. In *International Conference on Learning Representations*, 2021.

[24] Tianwei Ni, Benjamin Eysenbach, Sergey Levine, and Ruslan Salakhutdinov. Recurrent model-free rl is a strong baseline for many pomdps. 2021.

[25] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[26] A Ronca, G Paludo Licks, G De Giacomo, et al. Markov abstractions for pac reinforcement learning in non-markov decision processes. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022*, pages 3408–3415. International Joint Conferences on Artificial Intelligence, 2022.

[27] Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Küttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. *arXiv preprint arXiv:2109.13202*, 2021.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[30] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

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

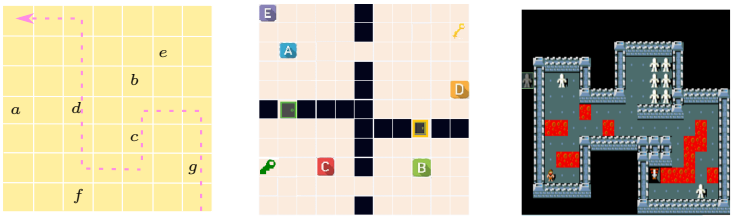
[32] Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. Learning reward machines for partially observable reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

[33] Elena Umili, Roberto Capobianco, and Giuseppe De Giacomo. Grounding ltlf specifications in image sequences. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, pages 668–678, 2023.

[34] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu. Joint inference of reward machines and policies for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 590–598, 2020.

[35] Zhe Xu, Bo Wu, Aditya Ojha, Daniel Neider, and Ufuk Topcu. Active finite reward automaton inference and reinforcement learning using queries and counterexamples. In *Machine Learning and Knowledge Extraction: 5th IFIP TC 5, TC 12, WG 8.4, WG 8.9, WG 12.9 International Cross-Domain Conference, CD-MAKE 2021, Virtual Event, August 17–20, 2021, Proceedings 5*, pages 115–135. Springer, 2021.

[36] Guoxi Zhang and Hisashi Kashima. Learning state importance for preference-based reinforcement learning. *Machine Learning*, pages 1–17, 2023.



(a) Letter (b) Room (c) MiniHack

Fig. 10. Environments.

APPENDIX

A. FF of Important States

In order to make the agent to traverse over $\hat{\mathcal{M}}_\varphi$ efficiently, the agent has to be able to reach any discovered important state ($\hat{s}_i \in \hat{\mathcal{S}}_I$) in a zero-shot manner. The option framework [31] provides us a solution, where the agent needs to train a specific option for reaching every subgoal. However, this can consume a lot of environmental samples and be wasteful if two subgoals are close to each other. In this work, we propose to learn a first-occupancy feature (FF) $\mathcal{F}_\vartheta$ of important states which can provide the agent a unified solution for visiting any discovered important state *for the first time* without further training.

According to the definition in Section III-C, the base feature function ϕ of FF can be formulated as the achievement of important states, i.e., $\phi(\cdot) : \mathcal{S} \to \mathbb{R}^{|\mathcal{G}|}$ where d-th feature $\phi_d(s)$ is computed by the cosine similarity between $f_\theta(s)$ and $f_\theta(\hat{s}_d)$ ($\hat{s}_d$ is the d-th important state in $\hat{\mathcal{S}}_I$), and $|\mathcal{G}|$ is the number of total subgoals. So FF is defined as $\mathcal{F}_\vartheta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{|\mathcal{G}|}$. Therefore, the d-th FF ($\mathcal{F}_{\vartheta,d}$) is defined to be the duration until the agent is expected to reach $\hat{s}_d$ for the *first time* by starting from a state-action pair (s, a). The FF here is defined with respect to the exploration policy π_{exp} which is omitted in the notation.

Given any transition tuple (s_t, a_t, s_{t+1}), the d-th element of $\mathcal{F}_\vartheta$ is trained to predict the target value defined as below,

$$\mathcal{F}_d^{\text{target}}(s_t, a_t, s_{t+1}) = \mathbb{1}(\phi_d(s_t) \geq \kappa) + \gamma(1 - \mathbb{1}(\phi_d(s_t) \geq \kappa)) \max_{a' \in \mathcal{A}} \mathcal{F}_{\bar{\vartheta},d}(s_{t+1}, a') \tag{7}$$

where $\mathcal{F}_{\bar{\vartheta}}$ is the non-trainable target model of $\mathcal{F}_\vartheta$ whose parameter $\bar{\vartheta}$ is a delayed version of ϑ and copied from ϑ periodically.

When training $\mathcal{F}_\vartheta$, due to the definition of FF, trajectories collected from the environment need to be pre-processed by decomposing into overlapping sub-trajectories and removing repetitive state-action pairs in every sub-trajectory. Specifically, a batch of trajectories $\mathcal{T}_m$ are first sampled from $\mathcal{B}_P \cup \mathcal{B}_N$. For every sampled trajectory τ, $\forall t = 1, \ldots, \text{len}(\tau) - 1$, a sub-trajectory τ_t can be formed by keeping the segment of τ starting from time step t till the end, i.e., $\tau_t := (s_t, a_t, \ldots, s_{\text{len}(\tau)})$, and a non-repetitive sub-trajectory $\tilde{\tau}_t$ is obtained by removing repetitive state-action pairs in τ_t. Then, we store all the non-repetitive sub-trajectories obtained from every trajectory of $\mathcal{T}_m$ into the set $\mathcal{K}$ which is the dataset for training $\mathcal{F}_\vartheta$. Therefore, the training objective of FF $\mathcal{F}_\vartheta$ can be written as below:

$$\mathcal{L}_{\text{FF}}(\vartheta) = \mathbb{E}_{(s_i,a_i,s_{i+1}) \sim \tilde{\tau}_t, \tilde{\tau}_t \sim \mathcal{K}}[(\mathcal{F}_\vartheta(s_i, a_i) - \mathcal{F}^{\text{target}}(s_i, a_i, s_{i+1}))^2] \tag{8}$$

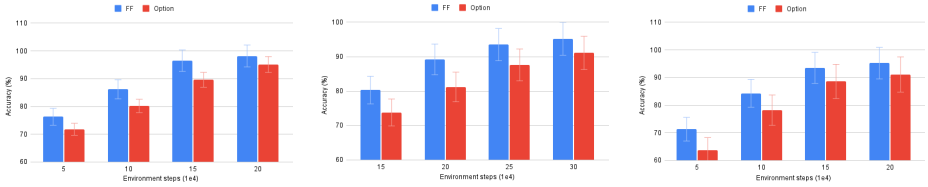
where $\mathcal{F}_\vartheta$ is realized by a neural network model with $|\mathcal{G}|$ output heads, and the d-th output of $\mathcal{F}_\vartheta$ is trained to predict $\mathcal{F}_d^{\text{target}}$ defined in (7). Note that since the agent can only be trained to reach important states $\hat{s} \in \hat{\mathcal{S}}_I$, only the first $|\hat{\mathcal{S}}_I|$ output heads of $\mathcal{F}_\vartheta$ are trained to predict the targets. When $|\hat{\mathcal{S}}_I|$ becomes same as $|\mathcal{G}|$, the important states for all the subgoals are learned and $\mathcal{F}_\vartheta$ is expected to guide the agent to reach the important state of every subgoal.

When the action space $\mathcal{A}$ is discrete, the policy of reaching $\hat{s}_d$ in state s is to greedily select the action a which maximizes $\mathcal{F}_{\vartheta,d}(s, a)$. In this work, we only consider discrete action space. The proposed approach can be easily extended to the continuous action space. For any index of learned subgoal $d \in [1, |\hat{\mathcal{S}}_u|]$, by using $\mathcal{F}_{\vartheta,d}$ as critic, the agent needs to train a policy model conditioned on d by the soft actor-critic (SAC) algorithm [10].

B. Environments

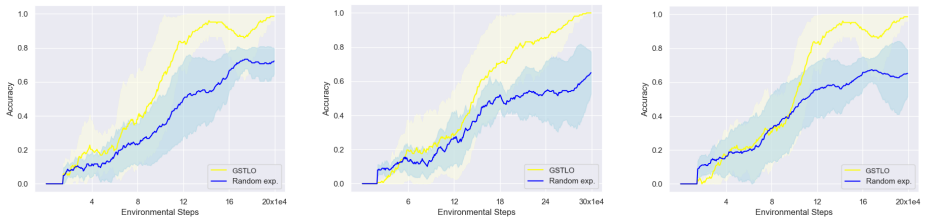
In this work, we evaluate the performance of the GSTLO framework in three environments, including letter world, room and MiniHack [27]. The first two are designed by authors. The third one has observations with higher dimensions. The observations of states are image-based in all the three environments, so simple tabular RL algorithms cannot solve any tasks in these environments. Since the labeling function is not available, the agent does not know the positions of letters or objects from state observations and their association with subgoal symbols in the TL task.

Letter World. This environment is a $n \times n$ grid game shown in Figure 10(a), replacing objects by letters. Out of the n^2 grid cells, m grids are associated propositions (letters). An example layout is shown in Figure 10(a) with $n = 7, m = 7$. At each step the agent can move along the cardinal directions (up, down, left and right). The agent is given the task specification



(a) Letter-1 (b) Letter-2 (c) Room

Fig. 11. Ablation study on using FF of learned subgoals.



(a) Letter-1 (b) Letter-2 (c) Room

Fig. 12. Ablation study on the exploration policy in GSTLO.

and is assumed to observe the full grid (and letters) from an egocentric point of view. But positions of these letters and their association with subgoal symbols in the TL task are unknown to the agent. The agent must visit these letters' locations in the right order to satisfy the task formula.

Room. This environment is also a grid-world game, but its observation is divided into four rooms as shown in Figure 10(b). There are multiple letters allocated in different positions. The agent is randomly placed into one of these rooms. Each room is connected to its neighbors by corridors. Two randomly selected corridors are blocked by locks. The agent can open a lock by using a key corresponding to that specific lock (having the same color). These (green and yellow) keys are placed in positions which the agent can reach. This environment is an upgrade of Letter World with obstacles and dependencies between objects (keys and locks) added. Every task formula is a TL formula in terms of object's letters (not including keys and locks). The agent must visit these letters in certain way to satisfy the TL formula.

MiniHack. MiniHack is a powerful sandbox for designing custom environments [27] derived from the NetHack game. The agent there can navigate in the map to visit landmarks, pick up weapons, use tools and fight against monsters. Our experiments only consider navigation tasks which are customized to be simpler than original environments. An example of the screenshot is shown in Figure 10(c). The layout of the map and items are initialized by a description file which is written by the user. In our experiments, the map is a 10×10 grid. The observation to the agent is an image, where each grid of the map is described by 16×16 pixels. The action space is customized to be small, including movement towards 4 directions, kick, and eat actions. The objects include comestible items, including apple, orange, meat and pancake, and the agent can take them by the eat action. Other objects are stone and gray rock, which can only be interacted with by using kick action. The task formula is defined in terms of these 6 objects. The agent needs to visit and interact with right objects in the right order.

C. Ablation Studies

In the proposed GSTLO framework, the agent uses FF of learned subgoals to visit discovered important states corresponding to different subgoals, which is motivated by improving sample efficiency. In order to verify the advantage of using FF here, compared with GSTLO, we evaluate a modified framework where FF is replaced by options and every option is learned to achieve a different subgoal. In evaluations of this section, the comparisons are conducted in letter and room environments, where the maps in letter-1 and letter-2 environments have the sizes of 7×7 and 11×11, respectively, and the map in room environment has the size of 11×11. The evaluation metric is the accuracy of learned subgoals. As shown in Figure 11, we can see that GSTLO which uses FF of learned subgoals achieves higher accuracy of grounded subgoals, showing its advantage of sample efficiency.

In addition, we also conduct the ablation study on the exploration policy π_{exp}. As introduced in Section IV-C2, π_{exp} is built by a GRU-based network and trained by binary episodic rewards of completing the given task. We compare this design choice of π_{exp} with a random policy, where the agent will use a uniformly random selection of actions to finish the rest of episode after reaching the working node v_w on the reconstructed FSM. The other parts of GSTLO framework are not changed. The comparison is shown in Figure 11. The exploration policy π_{exp} in GSTLO achieves higher sample efficiency. This is

because π_{exp} in GSTLO is trained by the rewards of successfully completing the given task and hence the collected trajectories contain more states closer to important states of subgoals, but the trajectories collected by random policy may cover state space uniformly.

D. Algorithms

Algorithm 1 Grounding subgoals via online RL

The given task φ; the set of subgoal symbols $\mathcal{G}$; task FSM $\mathcal{M}_\varphi$; the reconstructed FSM $\hat{\mathcal{M}}_\varphi$; the working node v_w; state representation function f_θ; set of discovered important states $\hat{\mathcal{S}}_I$; first-occupancy features (FF) of discovered important states $\mathcal{F}_\vartheta$; the exploration policy π_{exp}; the importance function for detecting important states $\tilde{L}_\omega$; the pre-processing function for computing MCFR pre_{FR}; the positive and negative buffers $\mathcal{B}_P$ and $\mathcal{B}_N$; training period T; hyper-parameter K for updating working node v_w;
Initialize $\hat{\mathcal{M}}_\varphi$ and set working node v_w at v_0;
for $p = 1, 2, \ldots$ **do**
  % Collect a trajectory τ
  Initialize the environment;
  On $\hat{\mathcal{M}}_\varphi$, obtain discovered important states along the path from v_0 to the working node v_w and store them as $\hat{\boldsymbol{s}} = [\hat{s}_1, \ldots]$;
  for $\hat{s}_i \in \hat{\boldsymbol{s}}$ **do**
    Guide the agent to reach $\hat{s}_i$ by using FF to select every action, i.e., $a_t = \arg\max_{a \in \mathcal{A}} \mathcal{F}_{\vartheta,i}(s_t, a)$ where the index i denotes the index of discovered important state in $\hat{\mathcal{S}}_I$;
  end for
  Use the exploration policy π_{exp} to finish the rest of this episode;
  Store the collected trajectory τ into $\mathcal{B}_P$ or $\mathcal{B}_N$ according to the label of task completion;
  % Training part (conducted periodically)
  if $p \bmod T == 0$ **then**
    % Discovering important states
    Update the state representation f_θ with NCE loss;
    Select trajectories from $\mathcal{B}_P$ and $\mathcal{B}_N$ which go through states in $\hat{\boldsymbol{s}}$;
    Compute MCFR of every trajectory by using function pre_{FR};
    Formulate contrastive learning objective in (5) and train the importance function $\tilde{L}_\omega$;
    if a new important state $\hat{s}_n$ is discovered by trained $\tilde{L}_\omega$ **then**
      Add $\hat{s}_n$ into $\hat{\mathcal{S}}_I$ and create a new index of learned subgoal accordingly;
    end if
    % Training FF of discovered important states
    Sample transition tuples from $\mathcal{B}_P$ and $\mathcal{B}_N$
    Compute target values for FF in (7);
    Train $\mathcal{F}_\vartheta$ with the objective in (8);
    % Training the exploration policy
    Sample trajectories from $\mathcal{B}_P$ and $\mathcal{B}_N$
    Use PPO to train π_{exp} with labels of task completion as rewards
  end if
  if no new important states is discovered for K iterations **then**
    % Update the reconstructed FSM $\hat{\mathcal{M}}_\varphi$
    The working node v_w is fully discovered and removed from the frontier $\mathcal{V}_f$;
    Expand $\hat{\mathcal{M}}_\varphi$ by adding new nodes to v_w with new important states discovered during which the working node is v_w, and these new nodes are also added to the frontier set $\mathcal{V}_f$;
    A new working node is randomly selected from $\mathcal{V}_f$;
  end if
end for

E. Model Architecture

We build neural network architectures for state representation function f_θ, importance function $\tilde{L}_\omega$, FF of important states $\mathcal{F}_\vartheta$ and the exploration policy π_{exp}. Since the observation of every environment is non-symbolic and image-based, in the models of $f_\theta, \mathcal{F}_\vartheta$ and π_{exp}, we use different convolutional neural network (CNN) modules to pre-process the input image and produce

an embedding vector for the downstream processing. The size of the CNN module is determined by the observation space of the environment. In letter/room domain with map size of $m \times m$, we used a 3-layer convolutional neural network (CNN) which have 16, 32 and 64 channels with stride of 1, respectively, and the kernel size is chosen as $l \in \{2, 3, 4\}$ where m is dividable by l. In MiniHack environment, the CNN module is the same as the classical CNN for deep RL proposed in [22], where the first convolutional layer has 32 channels with kernel size of 8 and stride of 4, the second layer has 64 channels with the kernel size of 4 and stride of 2 and the third layer has 64 channels with the kernel size of 3 and stride of 1. The CNN module produces an embedding vector with the size of 512.

Regarding f_θ, the state representation has 64 dimensions in letter/room environments and has 256 dimensions in the MiniHack environment. The importance function $\tilde{L}_\omega$ has three fully connected layers with 64 neurons in each layer. Regarding FF $\mathcal{F}_\vartheta$, following the CNN module introduced above, it has three fully connected layers with 64 neurons in first two layers, where the final layer has the same size of number of subgoals $|\mathcal{G}|$, producing the predicted FF of important states corresponding to subgoals.

The exploration policy π_{exp} is a GRU-based policy, whose architecture for CNN module is introduced above. In π_{exp}, the hidden dimension of GRU module is 64 for letter/room environments and 256 for the MiniHack environment. The outputs of π_{exp} consist of action and predicted value, which are conditioned on both the hidden state and the embedding vector of input image for the current state.

In Algorithm 1, the training period T is 5 for every environment, and the period K for updating the working node v_w is 20 for every environment. The hyperparameters of the PPO algorithm for training the exploration policy π_{pi} is presented in Table I.

TABLE I
HYPERPARAMETERS OF PPO

Hyperparameter	**Value**
Env. steps per update	1024
Minibatch size	256
Discount	0.995
Satisfaction Reward R_F	10
Optimizer	Adam
Learning rate	3×10^{-4}
GAE-λ	0.95
Entropy coefficient	0.01
Value loss coefficient	0.5
Gradient clipping	1.0
PPO clipping (ϵ)	0.2