Grounding Subgoals of Temporal Logic Tasks in Online Reinforcement Learning

Duo Xu and Faramarz Fekri

June 18, 2024

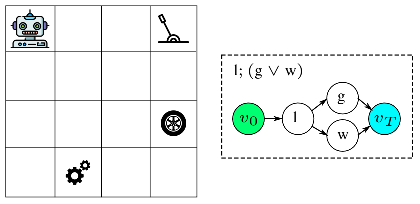
Grounding Subgoals of Temporal Logic Tasks via Online Reinforcement Learning

Duo Xu, Faramarz Fekri
Department of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA, 30332

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 for 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 collects raw observations from the environment and learns the labeling function. In other words, it learns to find out key states that are associated with subgoal symbols used to define the TL task. The proposed framework consists of exploration and labeling parts. In the exploration part, the agent actively explores the environment and discovers all the sequences of key states which can complete the TL task. Specifically, GSTLO formulates the discovery of key state sequences as a sequential multi-armed bandits (MAB) problem, and detects a single key state by contrastive learning based on the first-occupancy representations (FR) of collected trajectories. In the labeling part, the discovered key states are mapped to subgoal symbols by solving an integer linear programming (ILP) problem, yielding the labeling function. The GSTLO framework is evaluated on three environments, showing significant improvement over baseline methods.

Keywords: Reinforcement learning, symbol grounding, temporal logic, exploration

1 Introduction



(a) Observation (b) FSM

Figure 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.

Reinforcement learning (RL) algorithms have achieved many successes in recent breakthroughs like human-level video game playing from raw sensory input [1] and mastering complex board games [2]. 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) [3]. 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, solving TL tasks becomes a hot topic in RL community. In particular, a lot of RL algorithms have been proposed to solve TL tasks in the form of reward machine (RM) [4, 5, 6] and linear temporal logic (LTL) formulas [7, 8, 9, 10]. However, all of them assume the availability of a *labeling function* which maps raw states to subgoal symbols (e.g., indicating state that contains lever, wheel or gear in this example) for the task completion. It aligns the environmental raw observation into a Boolean interpretation over a set of predefined (subgoal) symbols. But in many real-world applications, the sequences of agent's observations are often not directly interpretable, whose mapping to symbols of TL formula is not known. In this case, the agent can only leverage the binary label given at the end of the episode, which indicates the result of task completion (success or not), to train its strategy to solve the task. Some previous POMDP algorithms can be used here, which build policy or value function by using recurrent neural networks (RNNs) [11, 12, 13]. However, these algorithms will suffer from poor learning efficiency, since the time horizon of TL task can be long and its temporal structure cannot be utilized. Thus, it is necessary to develop new algorithms for solving TL tasks without relying on such a labeling function given as prior knowledge.

In this work, we propose a novel framework for **G**rounding **S**ubgoals of **T**emporal **L**ogic tasks via **O**line reinforcement learning, short for GSTLO. It essentially learns the labeling function which can make the agent know the subgoal symbol whenever any of its associated states is reached, so that the agent can leverage the temporally compositional structure of TL task to improve its learning efficiency. The learned policies for reaching subgoals can also make the agent generalize to other unseen TL tasks. First define a *key state* as the raw state associated with a subgoal symbol of the TL task, e.g., the state where the robot is in the grid of wheel, gear or lever in Figure 1(a). The GSTLO framework consists of exploration and labeling parts.

In the exploration part, the agent actively collects trajectories from the environment and tries to discover every sequence of key states which can complete the TL task. We formulate the problem of detecting key state sequences as a sequential multi-armed bandit (MAB)[14, 15] problem. Following previously discovered sequence of key states, the agent only focuses on detecting next key states to visit based on the reward signal about task completion. The agent will first explore and try to visit every possible key state for completing the task, then select the state which can maximally increase the success rate of task completion as next key state. This works similarly as solving an MAB problem [15]. Then, the agent will continue exploring potential key states next to the selected state, stepping into solving another MAB problem. Since the TL task has constraints on the temporal ordering of subgoal visitation, only the first visit to the correct subgoal is meaningful to task completion. Therefore, the GSTLO computes the first-occupancy representation (FR) [16] of raw states in the collected trajectories and detects the key state by using contrastive learning based on FRs. After the agent has discovered sufficient number of key state sequences, every of which can satisfy the TL task, the labeling part can start to derive the labeling function. In the labeling part, we formulate an integer linear programming (ILP) problem to determine the mapping from the discovered key states to subgoal symbols, which can make every discovered sequence of key states reach the accepting state of the FSM of given TL task. When this mapping is solved, the labeling function is obtained.

We evaluate GSTLO in three environments, including Letter world, AntZone [17], and MiniHack [18]. 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.

2 Related Works

Recently linear temporal logic (LTL) formulas have been widely used in Reinforcement Learning (RL) to specify temporal logic tasks [19]. Some papers develop RL algorithms to solve tasks in the LTL specification [20, 21, 9]. In some other papers, authors focus on learning the task machine from traces of symbolic observations based on binary labels received from the environment [22, 23, 24]. However, all these papers assume the access to a labeling function which maps raw states into propositional symbols, working in the labeled MDP [8].

There are some papers assuming to have an imperfect labeling function, where the predicted symbols can be erroneous or uncertain [25, 26]. But these papers do not address the problem of symbol grounding. A recent paper studies the problem of grounding LTLf formulas in image sequences [27]. However, their method is only applicable to offline problems with static dataset and does not consider the online exploration in the environment. In addition, authors in [28] 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.

3 Preliminaries

3.1 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 [29]. The problems addressed by RL can be formalized as Markov decision processes (MDP), 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. We define a *key state* as the state associated with a subgoal symbol. Most states of the environment are not key states, where the output of L is empty. For example, as shown in Figure 1 for the state when the robot is at (0,0), the output of L is empty. However, when the robot is at (0,3), the state is a key state and the output of L will be "l" (lever).

3.2 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 [30] which is a finite fragment of LTL without using always operator $\Box$.

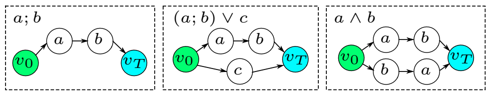


Figure 2: Examples of TL formulas and their corresponding FSMs. The initial node is v_0 and the accepting (terminal) node is v_T.

Every task specification $\varphi \in \mathcal{TL}$ can be represented by a non-deterministic finite-state machine (FSM) [28], 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 [28]. In this work, we only consider the FSMs which do not contain any loops. Hence, the FSM used here can be decomposed into sequences of subgoal symbols. 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.

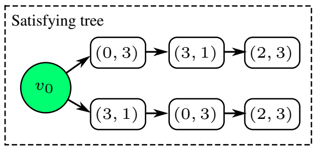


Figure 3: The satisfying tree of the task $(l \wedge g); w$.

In addition, we need to introduce the concepts of satisfying sequences and satisfying tree of the TL task, which are important to the proposed framework. First, we define the *satisfying sequence* as the sequence of key states which can satisfy the task. Second, the tree formed by all the satisfying sequences of the task is defined as the *satisfying tree*. For example, for the problem in Figure 1(a), assuming the task is $\varphi := (l \wedge g); w$, its satisfying sequences are $[(0,3),(3,1),(2,3)]$ and $[(3,1),(0,3),(2,3)]$, where $(0,3),(3,1)$ and $(2,3)$ represent key states associated with subgoals "l", "g" and "w", respectively. Its satisfying tree are shown in Figure 3.

3.3 First-occupancy Representation

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

Definition 1.[16] 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

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

where $\{s_{t:t+k}\} = \{s_t, s_{t+1}, \dots, s_{t+k-1}\}$ 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:

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

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

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

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 not interpretable, we learn a contrastive representation of states to measure the similarity of two sates, which is used to compute the indicator function in (3).

4 Methodology

In this work, we propose the GSTLO framework for solving TL tasks based on raw observations without relying on the labeling function, which grounds the subgoal symbols of TL task by actively exploring the environment. The policies trained to achieve grounded symbols can also make the agent zero-shot generalize to other unseen TL tasks. In the following, we will first have a general introduction to the GSTLO framework, and then present every component with details.

4.1 Framework

Grounding subgoals of TL task is essentially to learn the labeling function $L : \mathcal{S} \to \mathcal{G}$ which is the mapping from raw states in $\mathcal{S}$ of the environment to subgoal symbols in $\mathcal{G}$. Since the TL task may consist of multiple temporally extended subgoals, it is difficult to collect sufficient informative

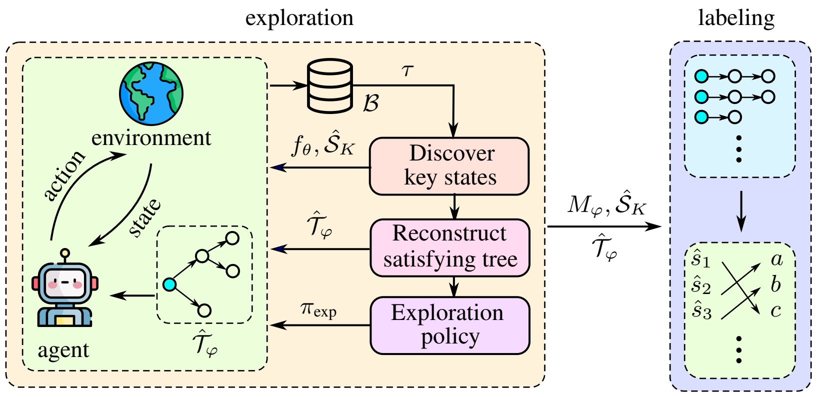


Figure 4: Diagram of the GSTLO framework. In the exploration part, $\mathcal{B}$ represents the positive and negative buffers with τ denoting trajectories, f_θ is the state representation function, $\hat{\mathcal{S}}_K$ is the set of discovered key states, $\hat{\mathcal{T}}_\varphi$ is the reconstructed satisfying tree, and π_{exp} is the exploration policy. The trajectory collection is guided by $\hat{\mathcal{T}}_\varphi$ and π_{exp}. In the labeling part, based on $\mathcal{M}_\varphi$, $\hat{\mathcal{S}}_K$ and $\hat{\mathcal{T}}_\varphi$, the mapping from key states to subgoals is determined by solving an ILP problem, yielding the labeling function. $\mathcal{M}_\varphi$ denotes the FSM of the task φ.

trajectories for an offline supervised learning approach to ground all the subgoals at once. As introduced in Section 3.2, there are finitely many sequences of visits to *key states* which can satisfy the task FSM, defined as *satisfying sequences*. Therefore, in this work, we propose the GSTLO framework which first discovers all the satisfying sequences of key states by actively exploring the environment, and second learns the labeling function by solving an ILP problem based on them.

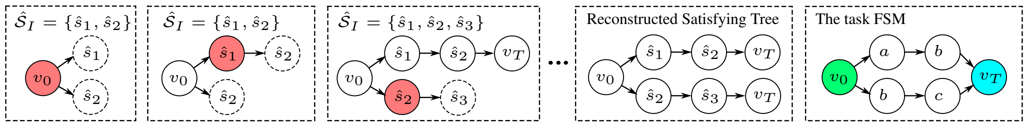
The GSTLO framework consists of exploration and labeling parts. The diagram of GSTLO is shown in Figure 4. In the exploration part, the agent actively collects trajectories from the environment. Inspired by the multi-stage nature of TL task, the agent discovers satisfying sequences of key states by solving a sequential MAB problem. Each stage of the satisfying sequence discovery is to find next key states to visit which can maximally increase the success rate of task satisfaction of the current stage. This can be formulated as an MAB problem [15] and solved by contrastive learning. The exploration policy is used to collect trajectories for discovering key states. Based on the discovered satisfying sequences of key states, the labeling part of GSTLO determines the mapping from discovered key states to subgoal symbols in $\mathcal{G}$ by solving an ILP problem, yielding the labeling function L. The detailed algorithm is presented in Appendix E.

Before introducing details of every component, we need to present the basic setup and notations here. Denote the TL task formula as φ, the FSM of task formula as $\mathcal{M}_\varphi$ and the set of satisfying sequences as $\mathcal{E}_\varphi$. The agent knows the task formula φ and its FSM $\mathcal{M}_\varphi$. In the episode k of exploration, the agent collects a trajectory τ_k by actively interacting with the environment and receives a binary label l_k indicating the task completion at the end of the episode (1 for success and 0 otherwise). Positive ($l_k = 1$) and negative ($l_k = 0$) trajectories are stored into positive ($\mathcal{B}_P$) and negative buffers ($\mathcal{B}_N$), respectively. By comparing the positive and negative trajectories, the agent uses contrastive learning to detect next key states. Repeating this process can make the agent discover all the key states one-by-one, forming the set of discovered satisfying sequences $\hat{\mathcal{E}}_\varphi$.

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

4.2 Exploration

The target of the exploration part is to discover all the satisfying sequences of the task. According to the discussion in Section 3.2, since satisfying sequences can be obtained by decomposing satisfying



$\hat{\mathcal{S}}_I = \{\hat{s}_1, \hat{s}_2\}$
$\hat{\mathcal{S}}_I = \{\hat{s}_1, \hat{s}_2\}$
$\hat{\mathcal{S}}_I = \{\hat{s}_1, \hat{s}_2, \hat{s}_3\}$
Reconstructed Satisfying Tree
The task FSM

Figure 6: Examples of reconstructing satisfying tree $\hat{\mathcal{T}}_\varphi$. In the left three figures, the red node denotes the working node v_w, and $\hat{\mathcal{S}}_K$ is given on the upper left corner. The dashed nodes are frontier nodes to be explored. The discovered key state and its index are used as attributes of each node. The fourth figure shows a fully reconstructed $\hat{\mathcal{T}}_\varphi$. The task FSM in the rightmost figure is transformed from the task formula, composed by subgoal symbols.

tree into paths, the exploration part of GSTLO is same as reconstructing the satisfying tree composed by discovered key states in $\hat{\mathcal{S}}_K$, denoted as $\hat{\mathcal{T}}_\varphi$, where every path of $\hat{\mathcal{T}}_\varphi$ from root to a leaf can form a satisfying sequence to complete the task φ. Due to the multi-stage nature of TL task, the reconstruction of $\hat{\mathcal{T}}_\varphi$ can be regarded as a sequential MAB problem, and every expansion of $\hat{\mathcal{T}}_\varphi$ is to find the next key states to visit for task completion, which is essentially an MAB problem [14].

In $\hat{\mathcal{T}}_\varphi$, except the root node, each node v_n of $\hat{\mathcal{T}}_\varphi$ has a key state attribute of $\hat{s}_{k_{v_n}}$ indicating the k_{v_n}-th discovered key state in $\hat{\mathcal{S}}_K$. Initially, only the root node v_0 is in $\hat{\mathcal{T}}_\varphi$. To reconstruct $\hat{\mathcal{T}}_\varphi$, the agent will focus on any leaf node whose path towards task completion (v_T) is not clear yet (defined as frontier nodes), and expand $\hat{\mathcal{T}}_\varphi$ starting from this kind of nodes. Specifically, the agent selects a *working node* v_w from the frontier nodes, and focuses on discovering key states *next to* v_w which are next key states to visit given that the agent has visited key states of nodes from v_0 to v_w in $\hat{\mathcal{T}}_\varphi$. The next key states is discovered by using contrastive learning based on first-occupancy representation (FR). This is the "discover key state" process with details introduced in Appendix A.2. Then, the discovered key states are used to expand $\hat{\mathcal{T}}_\varphi$, which is the "reconstruct satisfying tree" introduced in Appendix A.1. The expansion will not stop until $\hat{\mathcal{T}}_\varphi$ is fully reconstructed, where every leaf node of $\hat{\mathcal{T}}_\varphi$ is connected to v_T. In this situation, the sequence of key states from the root to every leaf of $\hat{\mathcal{T}}_\varphi$ can form a satisfying sequence and complete the task. Examples of reconstructed satisfying trees are shown in Figure 5 and 6, where the dashed nodes are frontier nodes.

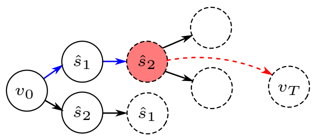


Figure 5: Collecting a trajectory conditioned on v_w in one episode. The red node is the working node v_w. The dashed nodes are in the frontier set. Blue path: collected by following the reference sequence $\xi_w := [\hat{s}_1, \hat{s}_2]$. Red path: collected by using the exploration policy π_{exp} till the end of the episode.

When discovering key states next to working node v_w, the collection of trajectory in every episode has two stages. First, the agent sequentially visits key states of nodes along the path from v_0 to v_w of $\hat{\mathcal{T}}_\varphi$, forming a sequence of key states, defined as *reference sequence* ξ_w. Then, after reaching the key state $\hat{s}_{k_{v_w}}$ of node v_w, the agent uses an exploration policy π_{exp} to continue collecting the trajectory τ until the end of the episode. This trajectory following the reference sequence ξ_w is called a trajectory *conditioned on* v_w. An example of trajectory collection process is shown in Figure 5. In this example, the reference sequence ξ_w for working node v_w is $[\hat{s}_1, \hat{s}_2]$. The details of the exploration policy are introduced in Appendix A.3.

Remark. Generally, ξ defines a reference sequence of key states that we want the agent to follow. To track this sequence we assume access to a low-level controller that takes actions a to move the agent along the key states in ξ. There are a variety of controllers for following reference sequences [31], and our approach is not tied to any specific controller choice. For example, a robotic agent can use motion planning to arrive any key states in ξ. We can also train options to make the agent arrive any selected state.

4.3 Labeling

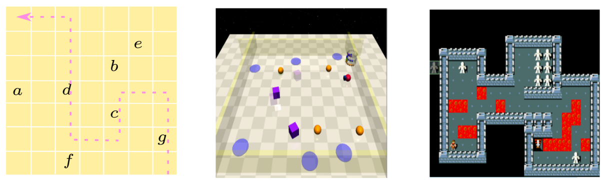
Given discovered satisfying sequences $\hat{\mathcal{E}}_\varphi$, discovered key states $\hat{\mathcal{S}}_K$ and task FSM $\mathcal{M}_\varphi$, in the labeling part of GSTLO, the mapping from $\hat{\mathcal{S}}_K$ to subgoal symbols $\mathcal{G}$ is determined by solving an ILP problem, yielding the labeling function. Specifically, $\hat{\mathcal{E}}_\varphi$ is obtained by directly decomposing the tree $\hat{\mathcal{T}}_\varphi$ to satisfying sequences. Note that some states in $\hat{\mathcal{S}}_K$ may not be associated with any subgoals, such as some bottleneck states of the environment. We will show that our approach can ignore these states when determining the mapping. The detailed formulation of the ILP problem is presented in Appendix B.

5 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, AntZone and MiniHack, as shown in Figure 7. The details of environments are introduced in Appendix C. Implementation details and other results are included in Appendix.

5.1 Baselines

Baseline-1. Contrastive learning has been widely used to learn key states or subgoals for solving a designated task or maximizing the rewards in previous papers [32, 33]. 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.



(a) Letter (b) AntZone (c) MiniHack

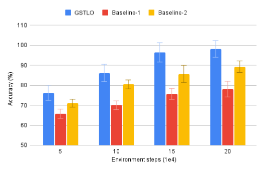
Figure 7: Environments.

Baseline-2. Recently authors in [27] propose a neural architecture consisting of a trainable convolutional neural network (CNN) and a non-trainable finite state automaton (FSA) which is a subclass 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 a number of additional 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.

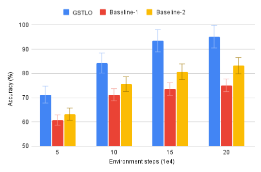
5.2 Results

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 D.2.

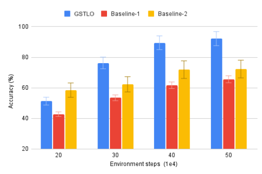
The accuracy of grounding subgoals is defined as the ratio of subgoals in $\mathcal{G}$ whose corresponding states are correctly discovered in the state space $\mathcal{S}$. In every environment, the performance is the average of 6 randomly generated tasks. The performance comparison of accuracy versus the environmental samples is shown in Figure 8. For all the evaluations in this section, the map in the letter environment has the width of $m = 11$ 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 9. 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.



(a) Letter

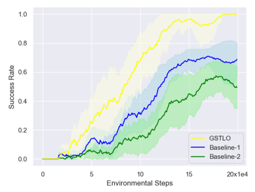


(b) AntZone

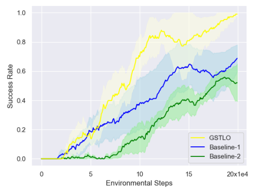


(c) MiniHack

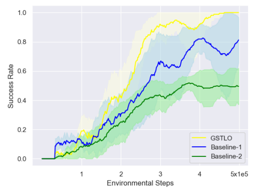
Figure 8: Comparison of accuracy of learned subgoals (discovered key states).



(a) Letter



(b) AntZone



(c) MiniHack

Figure 9: Comparison of success rate of solving the given task.

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 key states of subgoals next to a previously grounded subgoal. Based on the FF of discovered key 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 key 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.

6 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. In the exploration part, by discovering key states based on contrastive learning, the agent progressively reconstructs the satisfying tree of the task. In the labeling part, the mapping from discovered key states in the raw state space to subgoal symbols of the task, which yields the labeling function.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[3] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT press, 2008.

[4] R. T. 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.

[5] R. Toro Icarte, E. Waldie, T. Klassen, R. Valenzano, M. Castro, and S. McIlraith. Learning reward machines for partially observable reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

[6] Z. Xu, I. Gavran, Y. Ahmad, R. Majumdar, D. Neider, U. Topcu, and B. 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.

[7] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. 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.

[8] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. 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.

[9] A. K. Bozkurt, Y. Wang, M. M. Zavlanos, and M. 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.

[10] 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.

[11] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

[12] D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31, 2018.

[13] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[14] T. Lattimore and C. Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

[15] Y. Abbasi-Yadkori, D. Pál, and C. Szepesvári. Improved algorithms for linear stochastic bandits. *Advances in neural information processing systems*, 24, 2011.

[16] T. Moskovitz, S. R. Wilson, and M. Sahani. A first-occupancy representation for reinforcement learning. In *International Conference on Learning Representations*, 2021.

[17] J. Ji, B. Zhang, J. Zhou, X. Pan, W. Huang, R. Sun, Y. Geng, Y. Zhong, J. Dai, and Y. Yang. Safety gymnasium: A unified safe reinforcement learning benchmark. *Advances in Neural Information Processing Systems*, 36, 2023.

[18] M. Samvelyan, R. Kirk, V. Kurin, J. Parker-Holder, M. Jiang, E. Hambro, F. Petroni, H. Küttler, E. Grefenstette, and T. Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. *arXiv preprint arXiv:2109.13202*, 2021.

[19] 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.

[20] A. Camacho, R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *IJCAI*, volume 19, pages 6065–6073, 2019.

[21] G. De Giacomo, L. Iocchi, M. Favorito, and F. 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.

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

[23] Z. Xu, B. Wu, A. Ojha, D. Neider, and U. 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.

[24] 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.

[25] A. C. Li, Z. Chen, P. Vaezipoor, T. Q. Klassen, R. T. Icarte, and S. A. McIlraith. Noisy symbolic abstractions for deep rl: A case study with reward machines. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

[26] W. Hatanaka, R. Yamashina, and T. Matsubara. Reinforcement learning of action and query policies with ltl instructions under uncertain event detector. *IEEE Robotics and Automation Letters*, 2023.

[27] E. Umili, R. Capobianco, and G. 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.

[28] Z. Luo, J. Mao, J. Wu, T. Lozano-Pérez, J. B. Tenenbaum, and L. P. Kaelbling. Learning rational subgoals from demonstrations and instructions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12068–12078, 2023.

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

[30] G. De Giacomo and M. 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.

[31] M. W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot modeling and control*. John Wiley & Sons, 2020.

[32] G. Zhang and H. Kashima. Learning state importance for preference-based reinforcement learning. *Machine Learning*, pages 1–17, 2023.

[33] S. Casper, X. Davies, C. Shi, T. K. Gilbert, J. Scheurer, J. Rando, R. Freedman, T. Korbak, D. Lindner, P. Freire, et al. Open problems and fundamental limitations of reinforcement learning from human feedback. *arXiv preprint arXiv:2307.15217*, 2023.

[34] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[35] A. Goyal, A. Lamb, J. Hoffmann, S. Sodhani, S. Levine, Y. Bengio, and B. Schölkopf. Recurrent independent mechanisms. In *International Conference on Learning Representations*, 2020.

[36] T. Ni, B. Eysenbach, S. Levine, and R. Salakhutdinov. Recurrent model-free rl is a strong baseline for many pomdps. 2021.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[38] L. GurobiOptimization. Gurobi optimizer reference manual. 2023.

A The Exploration Part

Due to the page limit, we present the detailed operations of the exploration part of GSTLO here. The source codes are at https://drive.google.com/drive/folders/1O-51jLRnku5VlXCPGU2CpplxJyhR8AZ0?usp=sharing

A.1 Reconstruct Satisfying Tree

Assuming that the working node is v_w and key state next to v_w is selected as $\hat{s}_{\text{new}}$, a new leaf node v_{new} will be added to $\hat{\mathcal{T}}_\varphi$ as the child of v_w, and $\hat{s}_{\text{new}}$ will be the key state attribute of v_{new}. The new working node will be moved to v_{new}.

Given the working node v_w and reference sequence ξ_w, after following ξ_w, if the agent's *first visit* to a state $\hat{s}$ can immediately complete the task by receiving a success label from the environment, $\hat{s}$ will be naturally denoted as the key state next to v_w, and the sequence $\xi_w \cup [\hat{s}]$ is a discovered satisfying sequence. Then, a new leaf node v' will be added as a child of v_w in the tree $\hat{\mathcal{T}}_\varphi$, whose key state attribute is $\hat{s}$. No exploration starting from $\hat{s}$ is needed and a v_T node is added to v', since the task is completed after following the sequence $\xi_w \cup [\hat{s}]$. Afterwards, the agent will continue exploring other possible key states (except $\hat{s}$) next to v_w, until no more potential key states can be found to increase R after following ξ_w. Then, the working node will be moved back to the parent of v_w and continue exploring there, until $\hat{\mathcal{T}}_\varphi$ is fully reconstructed. An example of reconstructing a satisfying tree $\hat{\mathcal{T}}_\varphi$ is shown in Figure 6.

A.2 Discovering Key States

Define the reward function R as the success rate of task completion. This section introduces the discovery of key states next to the working node v_w. It has two steps: 1) contrastive learning: the agent discover potential key states (stored in $\tilde{\mathcal{S}}_{\text{next}}$) by using contrastive learning based on positive and negative trajectories conditioned on v_w; 2) selecting key state: the agent tries to visit any state of $\tilde{\mathcal{S}}_{\text{next}}$ after following the reference sequence ξ_w, and compute their impact on R, selecting the state which can maximally increase R as the key state next to v_w. After that, reconstructed $\hat{\mathcal{T}}_\varphi$ is updated according to Section A.1.

Contrastive Learning. When the raw state of environment has a lot of redundant information and not directly comparable, we propose to learn a distinguishable representation of states, i.e., $f_\theta(\cdot) : \mathcal{S} \to \mathbb{R}^d$, based on the InfoNCE loss [34]. Based on state representation f_θ, we first define $\tilde{L}_\omega : \mathbb{R}^d \to [0, 1]$ as the importance function. In GSTLO, the agent discovers key states by formulating the return of each trajectory in terms of $\tilde{L}_\omega$ and training $\tilde{L}_\omega$ to give higher returns to positive trajectories rather than negative ones. With sufficient training, any state s with value of $\tilde{L}_\omega(f_\theta(s))$ close to 1 is regarded as an key state.

After following the reference sequence of working node, only the *first* visit to a key state can make progress towards task completion. Thus, the contrastive learning objective used to train L_ω is formulated by the MCFR (introduced in Section 3.3) of positive and negative trajectories. Since the agent only discovers key states next to the working node v_w of $\hat{\mathcal{T}}_\varphi$, for any trajectory conditioned on node v_w, the agent computes the MCFR by using the sub-trajectory starting from the key state $\hat{s}_{k_{v_w}}$ of node v_w. The process of discovering key states based on contrastive learning consists of the three steps, which are presented as below:

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, 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 $\tilde{\tau}'_i$:

$$\begin{aligned} F^{\text{MC}}(s, s'; \tau) &= \sum_{\tilde{\tau}'_i \in \text{pre}_{\text{FR}}(\tau; s)} F^{\text{MC}}(s, s'; \tilde{\tau}'_i) \\ &= \sum_{\tilde{\tau}'_i \in \text{pre}_{\text{FR}}(\tau; s)} \sum_{t'=1}^{\text{len}(\tilde{\tau}'_i)} \gamma^{t'} \mathbb{1}(s_{t'} = s') \end{aligned} \tag{4}$$

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 (4) is still intractable to compute when states are not comparable and interpretable, 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 $\tilde{\tau} \in \tilde{\mathcal{D}}_P \cup \tilde{\mathcal{D}}_N$. Therefore, based on the pre-processed datasets $\tilde{\mathcal{D}}_P$ and $\tilde{\mathcal{D}}_N$, the contrastive learning objective for discovering important states next to v_w is expressed as in (5), where the set $\tilde{\mathcal{D}}_P$ ($\tilde{\mathcal{D}}_N$) is obtained by pre_{FR}. Any state s' with importance value higher than a threshold κ, i.e., $\tilde{L}_\omega(f_\theta(s')) \geq \kappa$, is chosen as an potential key state next to v_w, added to the set $\tilde{\mathcal{S}}_{\text{next}}$. Then, if state s' does not exist in the set of discovered key states $\hat{\mathcal{S}}_K$, s' will be added into $\hat{\mathcal{S}}_K$ with a new index $|\hat{\mathcal{S}}_K| + 1$ assigned and a newly detected subgoal $|\hat{\mathcal{S}}_K| + 1$ is also created.

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

Selecting Key State. After discovering potential key states in $\tilde{\mathcal{S}}_{\text{next}}$, the agent needs to select the key state next to v_w from $\tilde{\mathcal{S}}_{\text{next}}$. The agent collects trajectories to compute the impact of every state of $\tilde{\mathcal{S}}_{\text{next}}$ on the reward (success rate of task completion). Specifically, in every trajectory, the agent first follows the reference sequence ξ_w of working node v_w, then visits one state of $\tilde{\mathcal{S}}_{\text{next}}$, and then finish the rest of the trajectory by using the exploration policy π_{exp}. By comparing the reward of these trajectories with those without visiting states in $\tilde{\mathcal{S}}_{\text{next}}$ on purpose, the agent can determine the state in $\tilde{\mathcal{S}}_{\text{next}}$ which can maximally increase the reward function, and use this state to build the node next to v_w of reconstructed $\hat{\mathcal{T}}_\varphi$.

A.3 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 [35, 36] which extends the classical PPO [37] 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.

B The ILP Problem for Learning Labeling Function

Denote the state space of task FSM $\mathcal{M}_\varphi$ as $\mathcal{U}$ which has U states. The transition function of $\mathcal{M}_\varphi$ is defined as $\delta_\varphi : \mathcal{U} \times \mathcal{G} \times \mathcal{U} \to \{0, 1\}$, and the transition from state u to u' conditioned on symbol g

is expressed as $\delta_\varphi(u, g, u') = 1$. Assume that $\hat{\mathcal{E}}_\varphi$ has M sequences, and the m-th sequence has l_m elements.

Now we start formulating the ILP problem for learning the labeling function. The binary variables of this ILP problem are composed by state transition variables $u_{m,n,i,j}$ and mapping variables $v_{k,l}$, where $u_{m,n,i,j} = 1$ denotes that the n-th element of m-th sequence of $\hat{\mathcal{E}}_\varphi$ makes the agent transit from state i to j over FSM $\mathcal{M}_\varphi$, and $v_{k,l} = 1$ denotes that k-th state in $\hat{\mathcal{S}}_K$ is mapped to l-th subgoal symbol in $\mathcal{G}$. Based on their definitions, we can first have 5 constraints on these binary variables:

$$\sum_{i,j=1}^{U} u_{m,n,i,j} = 1,\ \ \forall m = 1, \ldots, M, n = 1, \ldots, l_n \tag{6}$$

$$\sum_{j=1}^{U} u_{m,1,1,j} = 1,\ \ \forall m = 1, \ldots, M \tag{7}$$

$$\sum_{i=1}^{U} u_{m,n,i,j} = \sum_{i=1}^{U} u_{m,n+1,j,i},\ \ \forall m = 1, \ldots, M, n = 1, \ldots, l_n - 1, j = 1, \ldots, U \tag{8}$$

$$\sum_{l=1}^{|\mathcal{G}|} v_{k,l} \leq 1,\ \ \forall k = 1, \ldots, |\hat{\mathcal{S}}_K| \tag{9}$$

$$\sum_{k=1}^{|\hat{\mathcal{S}}_K|} v_{k,l} = 1,\ \ \forall l = 1, \ldots, |\mathcal{G}| \tag{10}$$

where these constraints mean: 1) every element of every sequence in $\hat{\mathcal{E}}_\varphi$ makes a transition, including staying at the same state of FSM $\mathcal{M}_\varphi$; 2) the first element of every sequence is in the first state of $\mathcal{M}_\varphi$; 3) for any pair of consecutive elements of every sequence, the out-going state of the previous element is the same as the in-coming state of the other one; 4) every state in $\hat{\mathcal{S}}_K$ is mapped to at most one subgoal symbol; 5) every subgoal symbol in $\mathcal{M}_\varphi$ is associated with one state in $\hat{\mathcal{S}}_K$.

Since the transition variables $u_{m,n,i,j}$ and mapping variables $v_{k,l}$ must be consistent with the state transitions of $\mathcal{M}_\varphi$ (δ_φ), we have another set of constraints:

$$u_{m,n,i,j} \leq \delta_\varphi(i, j, l) \cdot v_{k,l} \tag{11}$$

where m, n, i, j and k, l have the same range of values as above constraints.

Finally, we have another set of constraints which make sure that the last element of every sequence in $\hat{\mathcal{E}}_\varphi$ makes the agent stay in any accepting state of FSM $\mathcal{M}_\varphi$. Then, we have

$$\sum_{j \in \mathcal{U}_F} u_{m,n,i,j} = 1 \tag{12}$$

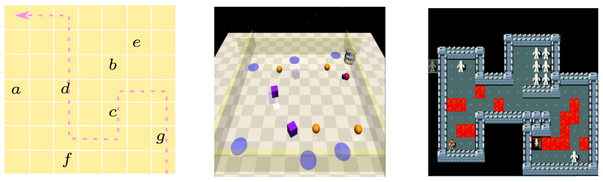
where $\mathcal{U}_F$ denotes the set of accepting states of $\mathcal{M}_\varphi$. In order to ignore the states in $\hat{\mathcal{S}}_K$ not associated with any subgoals during mapping, such as bottleneck state in the environmental layout, we use the sum of mapping variables as the objective:

$$\sum_{k=1}^{|\hat{\mathcal{S}}_K|} \sum_{l=1}^{|\mathcal{G}|} v_{k,l} \tag{13}$$

The formulated ILP problem has the objective (13) and constraints (6)-(12). We solve it by Gurobi solver [38].

C Environments

In this work, we evaluate the performance of the GSTLO framework in three environments, including letter world, AntZone and MiniHack [18]. 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.



(a) Letter (b) AntZone (c) MiniHack

Figure 10: Environments.

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 $n = 11, m = 7$. At each step the agent can move along the cardinal directions (up, down, left and right). The agent is given the task specification 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.

AntZone. We use the Zones environment from the MuJoCo-based Safety-Gymnasium suite of environments [17]. In this domain, a robot must navigate the environment whose 2D layout has a size of 10×10 and is divided into 100 integer grids, e.g., the grid from $[0, 0]$ to $[1, 1]$. The environment contains 5 differently colored goal regions, where every region is randomly located and covers only one integer grid. The robot receives an observation of lidar data that detects the presence of nearby objects at each timestep. The robot is a simulated ant robot shown in Figure 11. The LTL task description instructs the agent to oscillate amongst visiting different colored regions. The movement of robot is guided by the pre-trained options moving to the left, right, up and down neighboring integer grid. The current position is observable to the agent, and the symbol grounding is to determine the positions of subgoal symbols among integer grids.

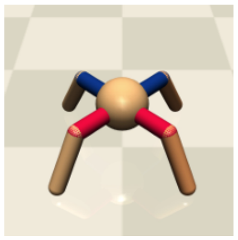


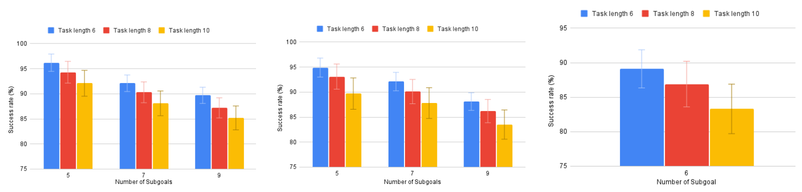
Figure 11: Simulated Ant Robot

MiniHack. MiniHack is a powerful sandbox for designing custom environments [18] 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.

D Additional Experimental Results

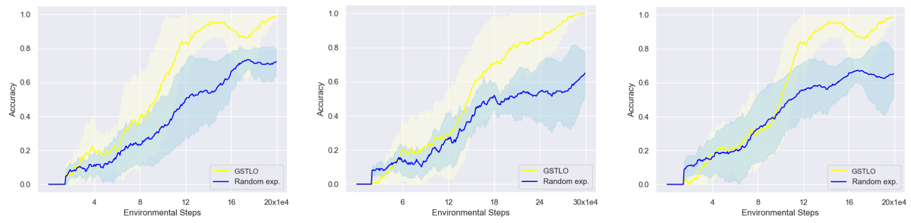
D.1 Generalization

Additionally, the generalizability of GSTLO is compared with baselines in Figure 12. 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 $\mathcal{G}$. The generalizability is evaluated for cases with different number of subgoals and task lengths, where the task length is the number of subgoals to be achieved for completing the given task.



(a) Letter (b) AntZone (c) MiniHack

Figure 12: Comparison of generalizability across different number of subgoals and task lengths. In MiniHack environment, the number of subgoal is fixed to be 6.



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

Figure 13: Ablation study on the exploration policy in GSTLO.

D.2 Ablation Studies

In addition, we also conduct the ablation study on the exploration policy π_{exp}. As introduced in Section A.3, 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 13. 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.

E Algorithms

The exploration operations conducted at working node v_w are shown in Algorithm 1, denoted as function $G(v_w)$. The operations for discovering key states are in Algorithm 2.

F 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/AntZone 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 [1], 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.

Algorithm 1 Grounding subgoals via online RL: $G(v_w)$

```
1: Require: The given task $\varphi$; the set of subgoal symbols $\mathcal{G}$; task FSM $\mathcal{M}_\varphi$; the reconstructed satisfying tree $\hat{\mathcal{T}}_\varphi$; the working node $v_w$; state representation function $f_\theta$; set of discovered key states $\hat{\mathcal{S}}_K$; the exploration policy $\pi_{\text{exp}}$; the importance function for detecting key states $\tilde{L}_\omega$; the pre-processing function for computing MCFR $\text{pre}_{\text{FR}}$; the positive and negative buffers $\mathcal{B}_P$ and $\mathcal{B}_N$; discovery period $T$;
2: Initialize $\tilde{S}_\neg := \{\}$
3: for $p = 1, 2, \ldots$ do
4:     % Collect an exploration trajectory $\tau$
5:     Initialize the environment;
6:     On $\hat{\mathcal{T}}_\varphi$, obtain discovered key states along the path from $v_0$ to the working node $v_w$ and form the reference sequence $\xi_w = [\hat{s}_1, \ldots]$;
7:     for $\hat{s}_i \in \xi_w$ do
8:         Guide the agent to reach $\hat{s}_i$ by using options or motion planning, where the index $i$ denotes the index of discovered key state in $\hat{\mathcal{S}}_K$;
9:     end for
10:    Use the exploration policy $\pi_{\text{exp}}$ to finish the rest of this episode;
11:    Store the collected trajectory $\tau$ into $\mathcal{B}_P$ or $\mathcal{B}_N$ according to the label of task completion;
12:    % Discovery part (conducted periodically)
13:    if $p \bmod T == 0$ then
14:        % Discovering key states
15:        Call Algorithm 2 to get $\hat{s}_n$ as the discovered key state next to $v_w$;
16:        if $\hat{s}_n$ is not in $\hat{\mathcal{S}}_K$ then
17:            Add $\hat{s}_n$ into $\hat{\mathcal{S}}_K$ and create a new index of learned subgoal accordingly;
18:        end if
19:        % Training the exploration policy
20:        Sample trajectories from $\mathcal{B}_P$ and $\mathcal{B}_N$
21:        Use PPO to train $\pi_{\text{exp}}$ with labels of task completion as rewards
22:        % Expand the reconstructed satisfying tree $\hat{\mathcal{T}}_\varphi$
23:        Add a new leaf node $v_n$ to $\hat{\mathcal{T}}_\varphi$ as a child of $v_w$, whose key state attribute is $\hat{s}_n$;
24:        Add $v_n$ to the frontier set $\mathcal{V}_f$;
25:        Select $v_n$ as the new working node and call $G(v_n)$ recursively; % The exploration of key states is conducted in a manner of depth-first search
26:        Set the working node as $v_w$ again; % Try to discover key state next to $v_w$ other than $\hat{s}_n$
27:        Add $\hat{s}_n$ into $\tilde{S}_\neg$; % Avoid visiting node $v_n$ again
28:    end if
29:    if no new key states has been discovered for $K$ iterations then
30:        The working node $v_w$ is fully discovered and removed from the frontier $\mathcal{V}_f$;
31:        Return
32:    end if
33: end for
```

Regarding f_θ, the state representation has 64 dimensions in letter/AntZone 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/AntZone 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.

Algorithm 2 Discovery of key states next to the working node

```
1: Update the state representation $f_\theta$ with NCE loss;
2: Select trajectories from $\mathcal{B}_P$ and $\mathcal{B}_N$ which are conditioned on $v_w$ on $\hat{\mathcal{T}}_\varphi$ and do not visit any states in $\tilde{S}_\neg$ after $v_w$;
3: Compute MCFR of every trajectory in $\tilde{D}$ by using function $\text{pre}_{\text{FR}}$ (defined in Section A.2), and store these pre-processed trajectories into $\tilde{D}$;
4: Formulate contrastive learning objective in (5) and train the importance function $\tilde{L}_\omega$;
5: $\tilde{S}_{\text{tmp}} := \{\}$;
6: In every trajectory in $\tilde{D}$, store every state which has high value at $\tilde{L}_\omega$ into $\tilde{S}_{\text{tmp}}$ as potential key states;
7: % Try to visit every potential key state
8: for $\hat{s} \in \tilde{S}_{\text{tmp}}$ do
9:     for $i = 1, \ldots, K$ do
10:        Initialize the environment; % Start a new episode
11:        Guide the agent to visit discovered key states by following the sequence $\xi_w \cup \{\hat{s}\}$;
12:        Use the exploration policy $\pi_{\text{exp}}$ to finish the reset of the episode;
13:    end for
14:    Based on trajectories collected in the above for loop, compute the success rate of visiting $\hat{s}$;
15: end for
16: Select the state with maximal success rate in $\tilde{S}_{\text{tmp}}$ as the newly discovered key state $\hat{s}_n$;
17: Return $\hat{s}_n$;
```

G Hyperparameters

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 1.

Table 1: 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