Interpretable Model-based Hierarchical Reinforcement Learning Using Inductive Logic Programming

Duo Xu and Faramarz Fekri

June 3, 2021

Interpretable Model-based Hierarchical Reinforcement Learning Using Inductive Logic Programming

Duo Xu [1] Faramarz Fekri [1]

Abstract

Recently deep reinforcement learning has achieved many success in wide range of applications, but it notoriously lacks data-efficiency and interpretability. Data-efficiency is important as interacting with the environment is expensive. Interpretability can increase the transparency of the black-box-style deep RL model and gain trust from the users of RL systems. In this work, we propose a new hierarchical framework of symbolic RL, leveraging a symbolic transition model to improve the data-efficiency and introduce the interpretability of learned policy. This framework consists of a high-level agent, a subtask solver and a symbolic transition model. Without assuming any prior knowledge on the state transition, we adopt inductive logic programming (ILP) to learn the rules of symbolic state transitions, introducing interpretability and making the learned behavior understandable to users. In empirical experiments, we confirmed that the data-efficiency of the proposed framework over previous methods can be improved by 30%∼40%.

1. Introduction

Reinforcement learning (RL) methods are the state of the art for solving complex sequential decision making problems in games and robotics, achieving a lot of success in practical applications (Van Hoof et al., 2015; Mnih et al., 2015; Kumar et al., 2016a;b; Akkaya et al., 2019; Falco et al., 2018). However, since the environment model and reward function are unknown initially, RL methods mostly rely on random exploration to collect rewards and then improve their current policy accordingly. Therefore, RL methods are notoriously sample inefficient, requiring billions of interactions with the environment before learning policies better than random exploration. This problem becomes more severe in long-horizon tasks (Andrychowicz et al., 2020). Another problem in deep RL is the lack of interpretability (Lyu et al., 2019; Puiutta & Veith, 2020). The learned behavior based on the black-box neural network is nontransparent and difficult to explain and understand. The goal of interpretability is to describe the internals of a system or learned behavior in a way that they are readable and verifiable by humans. In real-world applications of RL, it is instrumental to make the system behavior interpretable to human, so as to make the system user-friendly and gain trust from the user to employ a model or system (Israelsen & Ahmed, 2019; Došilović et al., 2018).

In this work we propose to use model-based Hierarchical RL (HRL) via inductive logic programming (ILP) to tackle problems mentioned above. First, HRL is a promising approach to reducing sample complexity and scaling RL to long-horizon tasks (Peng et al., 2017). The idea is to use a high-level policy to generate a sequence of high-level goals, forming subtasks, and then use low-level policies to generate sequences of actions to solve every subtask. By abstracting many details of the low-level states, the high-level policy can efficiently plan over much longer time horizons, reducing sample complexity in many tasks. In addition, because of explicit knowledge representation in the hierarchical formulation, performing reasoning and planning on high-level goals become an effective way to introduce interpretability into deep RL. Different from previous work on HRL with symbolic planning (Leonetti et al., 2016; Lyu et al., 2019; Illanes et al., 2020), we do not need any prior knowledge on symbolic transitions in the high-level part. Instead, by leveraging the inductive logic programming (ILP) (Evans & Grefenstette, 2018; Garcez et al., 2019), we adopt the model-based RL (Janner et al., 2019; Kurutach et al., 2018), which learns a transition model of the high-level symbolic states via predicate logic language in ILP and utilize this model to generate subtask sequences improving data-efficiency and interpretability. Moreover, we propose to use an abstract graph to abstract the primitive state in the high level, encoding each edge of the graph as a goal-oriented RL subtask.

As a result, the proposed framework provides the following benefits: (1) it improves the sample efficiency by leveraging the hierarchical learning framework and the symbolic state

[1]Georgia Institute of Technology. Correspondence to: Duo Xu <dxu301@gatech.edu>.

transition model, (2) it introduces interpretability into deep RL via the learned symbolic state transition rules, and (3) it provides the compositional generalization via the ILP. The effectiveness of the proposed method is verified by empirical experiments, compared with previous methods such as HRL (Kulkarni et al., 2016).

2. Preliminary

In this section, we establish relevant notation and review key aspects of symbolic reinforcement learning.

2.1. Reinforcement Learning

For the purposes of this work, we will say that the environment with which an RL agent interacts is formalized as a Markov Decision Process (MDP) $M = (S, A, r, p, \gamma)$, where S is the state space, A is the set of actions, $r : S \times A \to \mathbb{R}$ is the corresponding reward function, $p(s_{t+1}|s_t, a_t)$ is the state transition probability given any state-action pair, and $\gamma \in [0, 1)$ is the discount factor. A policy for M is defined as a probability distribution $\pi(a|s)$ representing the probability of the agent taking action a given that its current state is s. Therefore, the RL problem is to find the optimal policy π^* maximizing the expected discounted future reward obtained from all states $s \in S$ (Sutton et al., 1999):

$$\pi^* = \arg\max_{\pi} \sum_{s \in S} v_\pi(s)$$

where $v_\pi(s)$ is defined as the value function, approximating the expected discounted future reward obtained when starting at state $s \in S$ following the policy π, i.e.,

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s \right]$$

In this work, our method is built on Q learning (Watkins & Dayan, 1992), which is an RL approach that learns optimal policies (in the limit) by using sampled experiences to estimate the optimal q-function $q^*(s, a)$ for every state $s \in S$ and action $a \in A$. The optimal q-function $q^*(s, a)$ is equal to the expected discounted future reward received by performing action a in state s and following an optimal policy. Given an experience tuple (s, a, r', s'), the q-value estimate $\tilde{q}(s, a)$ is updated as follows

$$\tilde{q}(s, a) \longleftarrow \left(r' + \gamma \max_{a' \in A} \tilde{q}(s, a') \right)$$

Here the optimal policy π^* can be easily derived from $q^*(s, a)$ by selecting the action $a \in A$ with the largest q-value under the current state $s \in S$. In order to explore the environment, the ϵ-greedy exploration strategy is often used in Q-learning, selecting the random action with probability ϵ and the action with the largest $\tilde{q}(s, \cdot)$ value with probability $1 - \epsilon$.

2.2. Using Options in RL

Standard RL techniques are faced with significant problems when applied to environments with large state or action spaces. In practical terms, RL algorithms need a large amount of interactions with the environment before convergence. A popular technique for dealing with these issues is to consider temporally extended macro-actions that represent useful high-level behaviours, forming the basis of hierarchical reinforcement learning (HRL) (Barto & Mahadevan, 2003). Generally, human make decisions by utilizing temporal abstractions. An *option* is temporally extended course of action consisting of three components: a policy $\pi : S \times A \to [0, 1)$, a termination condition $\beta : S \to [0, 1]$, and an initial set $I \in S$. An option (I, π, β) is available in state s_t if and only $s_t \in I$. After the option is taken, a sequence of actions is selected according to π until the option is terminated with the probability of the termination condition β. With the introduction of options, we can formulate the decision-making as a hierarchical process with two levels, where the high level is the option level (also termed as task level) and the lower level is the action (sub-task) level. Markovian property exists among different options at the option level.

2.3. Inductive Logic Programming

Logic programming languages are a class of programming languages using logic rules rather than imperative commands. By adopting the programming language of *DataLog* (Koller et al., 2007), we define our logic language as below. Having predicate names (predicates), constants, and variables as three primitives, the predicate name is defined as a relation name, and a constant is termed as an entity. An *atom* α is defined as a predicate followed by a tuple $p(t_1, \ldots, t_n)$, where p is an n-ary predicate and $t_1, \ldots, t_n$ are terms, i.e., variables or constants. For example, the atom $on(X, \text{ground})$, denotes the predicate called on with X as variable and ground as constant. If all terms in an atom are constants, this atom is called a *ground atom*. In this work the set of all ground atoms is denoted as G. A predicate, which can be defined by a set of ground atoms, is called an *extensional predicate*. Further, a *clause* is defined as a rule in the form of $\alpha \leftarrow \alpha_1, \ldots, \alpha_n$, where α is the *head atom*, and $\alpha_1, \ldots, \alpha_n$ are *body atoms*. The predicates defined by clauses are termed as *intensional predicates*.

Inductive logic programming (ILP) is a task to derive a definition (set of clauses) of some intensional predicates, given some positive examples and negative examples (Koller et al., 2007; Evans & Grefenstette, 2018). Conducting ILP with differentiable architectures has been investigated in many previous work (Evans & Grefenstette, 2018; Rocktäschel & Riedel, 2017; Dong et al., 2019; Payani & Fekri, 2019). In this work, we adopt ∂ILP (Evans & Grefenstette, 2018) as the base method. With the differentiable deduction, the system can be trained with gradient-based methods. The loss value is defined as the cross-entropy between the confidence of predicted atoms and the ground truth. Compared with traditional ILP methods, ∂ILP has advantages in terms of robustness against noise and ability to deal with fuzzy data (Evans & Grefenstette, 2018).

3. Related Work

Interpretability There have been a lot of recent papers investigating the interpretability in deep learning (Doshi-Velez & Kim, 2017; Gilpin et al., 2018; Roscher et al., 2020). Making the black-box deep neural network explainable to human is also an active research area, having strong practical impact (Tjoa & Guan, 2020). There are some papers studying interpretable RL from the perspective of programming synthesis (Bunel et al., 2018; Verma et al., 2018). However, many unsolved problems on interpretable RL are left to be investigated.

Symbolic RL Some recent papers study the interpretability of RL by integrating symbolic planning (Leonetti et al., 2016; Lu et al., 2018; Yang et al., 2018; Lyu et al., 2019; Illanes et al., 2020), which inherit the interpretability of symbolic planning with symbolic knowledge. However, all of them require prior knowledge on action description, i.e., the effects of symbolic actions on the symbolic state representations, and they only conduct model-free RL in the symbolic state space. In this work, this prior knowledge is not required, and the learned symbolic transition model can enable the model-based RL in the high level and improve both the data-efficiency and interpretability.

4. Methodology

As discussed above, the primitive state and action spaces are defined as S, A, whereas symbolic state and action spaces in the high level are defined as $\tilde{S}$ and $\tilde{A}$. The target of the proposed method is to learn the optimal sequence of subtasks and find policies to solve them, so that the agent can achieve maximal cumulative reward executing the policy for each subtask sequentially.

Following previous work (Leonetti et al., 2016; Lyu et al., 2019; Illanes et al., 2020), we assume the availability of symbolic information given by human experts, i.e., a set of propositional symbols, representing important state properties and structures that may determine the outcomes of actions or their reward. It has been observed that majority of discrete dynamic domains have similar structures and can be generally formulated as action modules (Erdogan & Lifschitz, 2006; Garnelo et al., 2016; Garcez et al., 2018). Different from previous work, we do not need any prior knowledge on the symbolic state transition, i.e., the effects of symbolic actions.

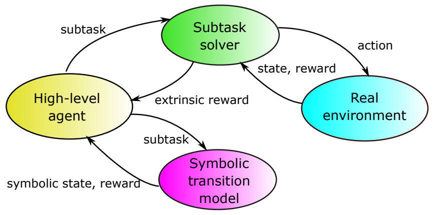


Figure 1: Diagram of Hierarchical Framework

4.1. Hierarchical Framework

By utilizing the symbolic representations given by the human expert, we can decompose the original MDP into low-level and high-level parts, and solve it in a hierarchical way, as shown in Figure 1. The high-level agent works on abstracted states with environmental symbolic information and generates sequence of subtasks, whereas the sub-task solver in the low level works on the primitive states and solves subtasks in the given sequence one by one. We assume that there is a set of *propositional symbols* (existential predicates) available, describing abstracted states and symbolic information given by the environment. We denote the set of propositional symbols as $\mathcal{F}$. E.g, in the room environment in Figure 2(a), $\mathcal{F}$ consists of room indices and possible status of locks and keys.

In our work the state abstraction is formulated over an *abstract graph*. Similar to previous works (Gopalan et al., 2017; Lyu et al., 2019; Abel et al., 2020), which assume that the state abstractions are provided by the user or expert, we also assume that the mapping from the primitive state to its abstraction (a node in the abstract graph) is given by the user. Given the abstract graph and its mapping, our algorithm uses model-free RL to automatically train options that serve as transitions between nodes in the abstract graph including changes in the symbolic information. Further, the logic rules governing these transitions are induced via ILP. For example, the primitive state of the room environment is shown in Figure 2(a), and the state abstraction is formulated as an abstract graph in Figure 2(c), where the node denotes the index of the robot's current room and each edge represents an transition between two rooms including changes of status of locks and keys. Thus, every edge corresponds to an option, forming a subtask to be solved by the sub-task solver in the low level.

In the high-level, based on propositional symbols, the policy of selecting subtasks is learned in the symbolic state $\tilde{S}$ and action $\tilde{A}$ spaces which are power sets of propositional symbols in $\mathcal{F}$. Different from previous HRL approaches, leveraging ∂ILP, we specifically learn a transition model for high-level symbolic states. In additional to the real environment M, as shown in Figure 1, the high-level agent updates its policy also by interacting with the symbolic transition model in a symbolic MDP $\tilde{M}$. This transition model introduces both compositional generalization and task-level interpretability. In the low-level part, the subtask solver solves subtasks with Q learning, working on primitive states S.

4.2. Options and Rewards

In this section, we are going to formally define the symbolic MDP, subtask, and the intrinsic and extrinsic rewards used in the hierarchical learning. The high-level learning is conducted over abstracted states with symbolic information from the environment. This can form the symbolic MDP $\tilde{M} = \langle \tilde{S}, \tilde{A} \rangle$, where $\tilde{S}$ is symbolic state space, i.e., a power set of propositional symbols (existential predicates) $\tilde{S} := 2^{\mathcal{F}}$ including indices of abstracted states (nodes in the abstract graph) and symbolic information of the environment. For any $\tilde{s} \in \tilde{S}$, the atoms in $\tilde{s}$ are all true and those not in $\tilde{s}$ are false. We also assume the existence of a labeling function from the primitive state to the corresponding symbolic state, i.e., $L : S \to \tilde{S}$, giving all the symbolic information received and abstracted state entered when a state $s \in S$ is reached. For example, in the room Environment shown in Figure 2(a), $\tilde{s} \in \tilde{S}$ can be the set of the index of robot's current room and status of keys and locks, and each $\tilde{a}$ can denote the index of robot's intended (next) room.

Each subtask is defined by the symbolic state-action pair $(\tilde{s}, \tilde{a})$ for $\forall \tilde{s}, \tilde{a} \in \tilde{S} \times \tilde{A}$, where the termination condition of each subtask is only determined by the symbolic action $\tilde{a}$. E.g., the subtask in room environment is defined by the pair of indices of current and intended rooms together with the status of locks and keys, and its termination condition is whether the robot enters the intended room or not.

For every $\tilde{a} \in \tilde{A}$, we define a termination condition set $\alpha(\tilde{a}) \in \tilde{S}$ containing every symbol (atom) in $\mathcal{F}$ related with the termination condition defined by $\tilde{a}$, and denote $T_{\tilde{a}}$ as the set of all primitive states in S satisfying the termination condition of $\tilde{a}$. Then we can have the relationship $T_{\tilde{a}} = L^{-1}(\alpha(\tilde{a}))$, where L^{-1} is the inverse of the labeling function. Specifically, in room environment, for subtask $(\tilde{s}, \tilde{a})$, $\alpha(\tilde{a})$ is the index of robot's intended room and $T_{\tilde{a}}$ can be the set containing all the primitive states where the robot has entered room $\alpha(\tilde{a})$. Both $\alpha(\tilde{a})$ and $T_{\tilde{a}}$ can determine the success of the subtask. Hence we can define the set of options as $O(M) = \{\langle \pi_{\tilde{a}}, T_{\tilde{a}} \rangle | \tilde{a} \in \tilde{A}\}$, where every policy $\pi_{\tilde{a}}$ is specifically trained for reaching states in $T_{\tilde{a}}$. E.g., in room environment, the policy $\pi_{\tilde{a}}$ can be a policy specifically trained for a certain subtask, such as going to the neighboring room on the right.

In the proposed framework, as shown in Figure 1, the subtask solver and high-level agent are trained by *intrinsic rewards* and *extrinsic rewards*, respectively (Yamamoto et al., 2018; Le et al., 2018; Illanes et al., 2020). The policies in the low level for solving subtasks are learned using Q-learning with intrinsic rewards, which have pseudo-rewards to encourage the agent to finish each subtask successfully. For certain subtask $\tilde{a} \in \tilde{A}$, the *intrinsic reward* at state $s \in S$ is defined as

$$r_i(s; \tilde{a}) = \begin{cases} \eta & \alpha(\tilde{a}) \subseteq L(s) \\ r & \text{otherwise} \end{cases} \tag{1}$$

where η is a large number to encourage the agent to achieve the target of the subtask, and r is the reward for valid movement or environmental reward. We know that $\alpha(\tilde{a})$ denotes symbols representing the success conditions of the subtask, e.g., the index of robot's intended room, and $L(s)$ is the symbolic state corresponding to $s \in S$, e.g., robot's current room and status of locks and keys. Hence $\alpha(\tilde{a}) \subseteq L(s)$ denotes that successful conditions of the subtask have been satisfied in the current state. Moreover, the high-level agent performs Q learning with extrinsic rewards. For subtask $\tilde{a}$ with initial symbolic state $L(s)$, the *extrinsic reward* is defined as

$$r_e(L(s), \tilde{a}) = \begin{cases} R(L(s), \tilde{a}), & 0.9 < t(L(s), \tilde{a}) \\ -\xi_0, & 0 < t(L(s), \tilde{a}) < 0.9 \\ -\xi_1, & t(L(s), \tilde{a}) < 0.9 \text{ and} \\ & \quad N < n(L(s), \tilde{a}) \end{cases} \tag{2}$$

where $t(L(s), \tilde{a})$ denotes the success rate of the subtask based on learning history, and $n(L(s), \tilde{a})$ is the number of that subtask having been tried so far. Specifically $0 < \xi_0 \ll \xi_1$ refers to the penalty for immature and unlearnable subtasks respectively. Immature subtasks refer to those without sufficient training, and unlearnable subtasks are those too difficult to solve. We penalize unlearnable subtasks more heavily than immature ones. If the subtask can be solved robustly, the extrinsic reward is set to be $R(L(s), \tilde{a})$ which is the environmental rewards accumulated when solving the subtask $\tilde{a}$ starting at $s \in S$.

4.3. Learning Symbolic Transition Model

For symbolic state and action spaces, we define the state transition and reward as $\tilde{P} : \tilde{S} \times \tilde{A} \to \tilde{S}$. We denote the logic transition model as $\tilde{P}_{\boldsymbol{\phi}}$, parameterized by $\boldsymbol{\phi}$. In this work, we propose to use ∂ILP to learn the logic rules describing the state transition in the symbolic space. It is to conduct regression over collected experience tuples in the symbolic state and action spaces, which uses the symbolic state of current state ($L(s)$) and symbolic action ($\tilde{a}$) as inputs to predict the next abstracted state (next position in the abstract graph) and reward information described in a predicate language.

Specifically ∂ILP operates on the valuation vectors whose space is $E = [0, 1]^{|\mathcal{F}|}$, each element of which represents the confidence that a related grounded symbol (atom) in $\mathcal{F}$ is true. Denote $\boldsymbol{e}_0$ as the valuation (true or false) of all the symbols in $\mathcal{F}$. We define a mapping $d_{\boldsymbol{\phi}} : E \to E$ with parameters $\boldsymbol{\phi}$, which performs deduction of facts $\boldsymbol{e}_0$ using weights $\boldsymbol{\omega}$ associated with all the possible clauses. $d_{\boldsymbol{\phi}}$ always consists of repeated applications of single-step deduction function $g_{\boldsymbol{\phi}}$, shown as below,

$$d_{\phi}^{t}(\boldsymbol{e}_0) = \begin{cases} g_{\phi}(d_{\phi}^{t-1}(\boldsymbol{e}_0)) & \text{if } t > 0 \\ \boldsymbol{e}_0 & \text{if } t = 0 \end{cases} \qquad (3)$$

where t is the deduction step, and $g_{\boldsymbol{\phi}}$ represents one-step deduction of all the possible clauses weighted by their confidences. Defining probabilistic sum $\oplus$ as $\boldsymbol{a} \oplus \boldsymbol{b} = \boldsymbol{a} + \boldsymbol{b} - \boldsymbol{a} \odot \boldsymbol{b}, \forall \boldsymbol{a}, \boldsymbol{b} \in E$, we can express the operation of single-step deduction as below

$$g_{\boldsymbol{\phi}}(\boldsymbol{e}) = \left(\sum_{i}^{\oplus} \sum_{j} \omega_{i,j} f_{i,j}(\boldsymbol{e}) \right) + \boldsymbol{e}_0 \qquad (4)$$

where function $f_{i,j}$ implements one-step deduction using jth definition of ith possible clause, with $\omega_{i,j}$ as its weight (Evans & Grefenstette, 2018; Jiang & Luo, 2019). For the specific ith clause, we can constrain the sum of its weights to be 1 by letting $\boldsymbol{\omega}_i = \text{softmax}(\boldsymbol{\phi}_i)$, where $\boldsymbol{\phi}_i$ are related parameters to be trained. Then the transition model can be denoted as $\tilde{P}_{\boldsymbol{\phi}}$, parameterized by the same parameters as ILP model. In addition, we propose to define an extensional predicate $\text{CurAct}(X, Y)$, with variables X and Y denoting current symbolic state and intended symbolic action respectively. Several auxiliary predicates are also introduced to facilitate the logic induction in ∂ILP.

4.4. Algorithm

The details of the proposed method is described in Algorithm 1 in Appendix. It is based on the HRL setting shown in Figure 1, where the high-level agent selects the goal, forming a subtask, and asks the sub-task solver to learn the policy for this subtask by regular Q-learning. For the symbolic MDP $\tilde{\mathcal{M}}$, we propose to learn a symbolic state transition model $\tilde{P}_{\boldsymbol{\phi}}$ by ILP and form a simulated environment $\tilde{\mathcal{M}}'$, so as to reduce the sample complexity and introduce task-level interpretability. The high-level agent interacts with the real environment via sub-task solver and simulated environment $\tilde{\mathcal{M}}'$ in an alternating way, described in Line 10-18 and Line 20 of Algorithm 1 in Appendix respectively. The transition model $\tilde{P}_{\boldsymbol{\phi}}$ is updated by ∂ILP in Line 32, with the objectives (3)(4).

In order to better control estimation bias which is especially harmful in discrete state-action environments, the high-level agent performs *Maxmin Q learning* (Lan et al., 2019) with extrinsic rewards (2). Its core idea is to obtain N estimates of the action (Q) value, i.e., $Q_h^1, \dots, Q_h^N$, and use the minimum of these estimates as the target in Q learning, i.e., $\max_{\tilde{a}} \min_{i \in \{1,\dots,N\}} Q_h^i(\tilde{s}, \tilde{a})$.

5. Experiments

We evaluate the proposed approach in two environments, having different propositional symbols and dynamics. The first environment is the modified room environment (Le et al., 2018; Abel et al., 2020). We extended it to have more complex high-level states, with more propositional symbols added. The second is Montezuma's Revenge, one of the most difficult game in Atari games (Mnih et al., 2015), concerning the movement of an Avatar among a set of locations (ladders, platforms, doors, ropes, etc), picking up a key and using the key to open a door. The first environment has simple low-level subtasks and complex high-level states, whereas the second one has complex subtasks and relatively simple high-level states. We quantitatively verify the advantage of the proposed method in sample efficiency and compositional generalization. The interpretability can only be presented qualitatively.

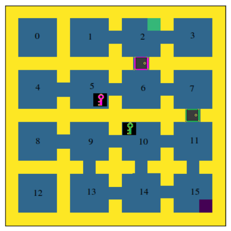
In the experiments, only minimal set of propositional symbols is provided to describe the background and auxiliary predicates are not provided as prior knowledge. The relationships about auxiliary predicates are learned by the agent automatically. Our proposed method is not dependent on that the agent knows the *meaning* of provided symbolic propositions.

5.1. Navigation in Room Environment

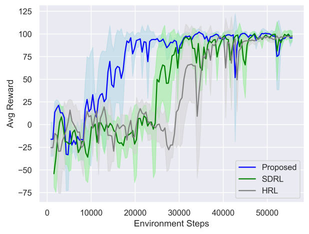
The room environment was a classical testbed for hierarchical RL, used in many previous papers (Gopalan et al., 2017; Le et al., 2018; Abel et al., 2020). It is to navigate the robot to the target room. In this work, in order to complicate the symbolic state, we add locks and keys in various colors and place them in different rooms.

Setup The training map consists of 17×17 grids, evenly partitioned into 4×4 rooms, shown in Figure 2(a). Every room occupies 3×3 grids, and adjacent rooms are separated by wall segments (yellow blocks). Some pairs of adjacent rooms are connected by corridors. Some rooms have keys, and some corridors are blocked by locks. The lock can only be opened by the key in the same color. And the robot has to open several locks before reaching the target room. In addition, the robot can only observes the current room without knowing the connectivity of rooms or locations of locks and keys.

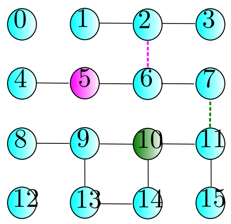
Every movement of the agent incurs a reward of -1, encouraging the agent to follow the shortest path. The reward of reaching the target is 100, and opening each lock can receive a reward of 10. There are no rewards for other situations, making the environmental rewards sparse. The robot can



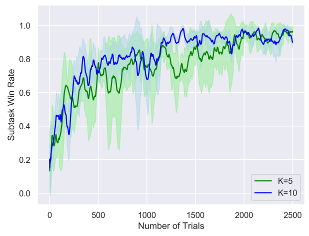
(a) Training Environment



(b) Learning Curves



(c) Abstract Graph



(d) Subtask Success Rate

Figure 2: Room Environment for Training and Learning Performance. There are two pairs of locks and keys in red and green. The robot starts at black grid and targets at the green grid. The state abstraction is also shown in abstract graph.

only hold one key at the same time.

Symbolic Representation The high-level state consists of the index of current room, the intended symbolic action of the robot, whether the room has been visited or not, and color of the lock and key in the current room. The graph of the training environment is shown in Figure 2(c). Rooms are nodes and corridors or locks are edges. Keys are contained in rooms in different colors and locks are represented by dashed lines. The symbolic actions are room indices, referring to the next room the robot intends to go.

The predicates used in learning the abstract graph are formally defined in Table 1 in Appendix, where variables are X and A, referring to room indices and colors of locks and keys, respectively. Specifically, if Connect(X, Y) is true, subtask of going from room X to Y is feasible. They are all initialized to be false. These properties are unknown to the robot apriori. The reward of opening a lock is formulated as part of the transition rules, and the reward of reaching the target is not formulated due to its simplicity.

Experiment Result The first experiment is to compare the sample efficiency of the proposed method with baselines, i.e., symbolic deep RL (SDRL) (Lyu et al., 2019) and hierarchical DQN (Kulkarni et al., 2016), on the training map in Figure 2(a). The second experiment is to verify the compositional generalization introduced in the proposed method, where the baseline is hierarchical DQN. The maximum episode in training and testing is 200.

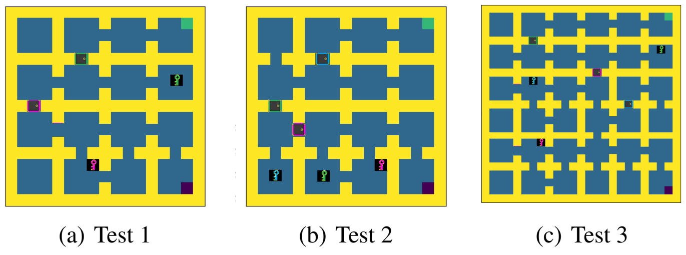
The symbolic state transition model $\tilde{M}_\phi$ is trained by the experience collected from the symbolic MDP $\tilde{M}$. By optimizing the objective (4), the symbolic state transition rules, readable to human, can be induced as below,

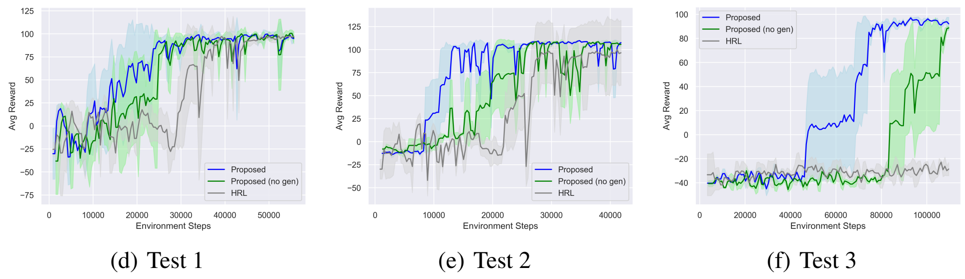
- 1: CurAct(X,Y), Connect(X,Y)→Next(Y)
- 2: RoomHasKey(X,Y), Visited(X)→invent1(A)
- 3: invent1(A), Lock(X,Y,A)→invent2(X,Y)
- 4: CurAct(X,Y), invent2(X,Y)→Next(Y)
- 5: CurAct(X,Y), invent2(X,Y)→RewardOpenLock(Y)

In the induced transition rules, several invented predicates, labelled as *invent1*, *invent2*, are used to represent auxiliary concepts inherent in symbolic state transitions. *CurAct* is the input to the transition model, and *Next* is the output which is also the training target. The rest of above predicates are all *extensional predicates*, grounded by the subtask solver. The rule 1 above denotes the transition of going to another room through a corridor. The rule 2 denotes that the key in color A has been obtained by the robot (agent). The rule 3 tells us that the lock in color A has been opened by the correct key. Finally the rule 4 refers to the event that the agent goes to the intended room Y through an opened lock, with reward information formulated in 5. These rules 1-5 are learned via the ILP and unknown to the agent apriori.

We first verify the sample efficiency by comparing the proposed method with baselines on the training environment. Specifically, the action description in SDRL (Lyu et al., 2019) is reformulated here and anything about state transitions are removed. We adopt ϵ-greedy for action selection in the high level, where ϵ is linearly decreasing from 0.3 to 0.03. And in order to reduce estimation bias, we use $N = 4$ tables for Q values in Maxmin Q-learning (Lan et al., 2019). The learning curves of cumulative rewards are shown in Figure 2(b). We can see that the proposed method is around 40% more sampling efficient than hDQN, and 30% more efficient than SDRL. That is because the Q function in the high level is also updated via the learned transition model, and the induced rules can also explain new transition tuples, e.g., the rule for opening locks can be generalized to locks in different locations. Moreover, the number of trials for each subtask K is set to 10, and the subtask success rate is shown in Figure 2(d), compared with the case of $K = 5$. It is observed that the subtask success rate can quickly become close to 1, and it is not equal to 1 because the high-level agent can always select some infeasible subtasks with some probability due to the ϵ-greedy strategy.

We then evaluate the capability of compositional generalization of the proposed method. The training is still performed on the map in Figure 2(a). However, the testing is on three different maps shown in Figure 3. The first testing map in Figure 2(a) has similar difficulty as the training map, the second one in Figure 3(b) has more locks to open than training map, and the third one in Figure 3(c) has larger size with


(a) Test 1 (b) Test 2 (c) Test 3



(d) Test 1 (e) Test 2 (f) Test 3

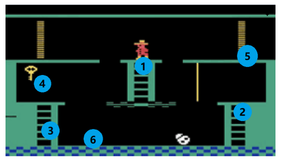
Figure 3: Testing Environments and Compositional Generalization. Three testing environments are shown in the first row, and performance comparisons are in the second row. The hierarchical DQN did not solve test 3 in 200 episodes.

a deceptive lock leading to dead-end. The first baseline is the proposed method without utilizing the transition model learned in the training environment, and the second baseline is the HRL (Kulkarni et al., 2016) without any specific design for generalization.

During testing, the Q function learned in training cannot be used directly, since the room connectivity has changed in the testing maps. However, as shown in Figure 3, the proposed method can still solve testing maps significantly faster than baselines, since the symbolic transition rules learned in the training map still hold in the testing maps. More interestingly, the regular HRL cannot solve Test 3 in 200 episodes, and the proposed can solve that within 100 episodes. We can also see that the first baseline, i.e., the proposed method without using learned transition rules in the training map, learn slower in all the testing maps, showing the effectiveness of the symbolic transition model.

5.2. Montezuma's Revenge

"Montezuma's Revenge" requires the player to navigate the agent through several rooms while collecting treasures. For the first room, as shown in Figure 4(a), in order to open the door, the player has to first climb down the ladders to pick up the key, jumping over the moving skull for twice, resulting in a long sequence of actions before receiving a reward for collecting the key $(+100)$. After that, the player has to return back and walk towards the door and open it, which results in another reward $(+300)$. Optimal execution requires more than 200 primitive actions. Vanilla DQN frequently achieves a score of 0 on this domain (Mnih et al., 2015).



(a) Game Screen and Abstraction

$$\begin{tikzcd}
 & 1 \arrow[r] \arrow[dr, bend left] & 5 \\
4 \arrow[d, bend left] & & \\
3 \arrow[u, bend left] \arrow[rr, bend left] & 6 & 2 \arrow[ul, bend left] \arrow[ll] \arrow[l]
\end{tikzcd}$$

(b) Abstract Graph

Figure 4: The game screen with 6 pre-defined nodes is shown in the left picture. And feasible subtasks are also presented as edges in the abstract graph.

Setup For this high-dimensional environment, our experiment is based on the DQN architecture (Kulkarni et al., 2016) with double-Q learning (Hasselt et al., 2016) and prioritized experience replay (Schaul et al., 2015). Based on the Arcade Learning Environment (ALE) (Bellemare et al., 2013), we use ALE API to recognize the locations of the agent, the skull, ladders and platforms from pixels, realizing the mapping function L from observation to symbolic propositions $\mathcal{F}$. Here we only use hierarchical DQN (hDQN) (Kulkarni et al., 2016) as baseline. That's because SDRL had many assumptions on symbolic transition rules as prior knowledge (Lyu et al., 2019), which are not avoidable here. The prior knowledge in the proposed method is only about 6 pre-defined locations and symbolic propositions given by the environment.

The high-level representation (abstraction) of this environment is based on 6 pre-defined locations: middle platform (1), lower left ladder (2), lower right ladder (3), key (4), right door (5) and left of rotating skull (6), with indices denoted in the brackets. All of these nodes are shown in Figure 4(a). The symbolic proposition given by the envi-

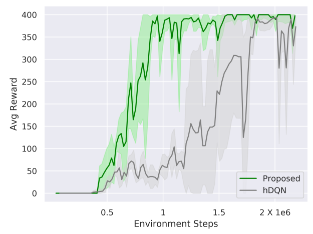


Figure 5: Learning Curves for Symbolic Monetzumar's Revenge.

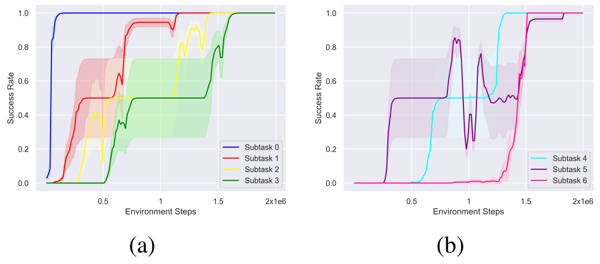


Figure 6: The success rate for all the subtasks.

ronment is the predicate hasKey() with 0 arity, denoting whether the key has been picked up or not. The subtask in this environment is defined as the transition from one node to another with or without key, on the abstract graph in Figure 4(b). We only consider 13 subtasks here, defined in Table 2 in Appendix. Each subtask is handled by an independent DQN, and the architecture of deep neural networks follows the classical design (Mnih et al., 2015), used in most Atari games. Similar as the previous section, besides hasKey(), we also adopt some regular predicates in this environment, including CurAct$(\cdot,\cdot)$, Connect$(\cdot,\cdot)$ and Next$(\cdot)$. In addition, to formulate reward function here, we define predicates RewardKey() and RewardDoor(), corresponding to the events of receiving reward $+100$ and $+300$ for getting the key and opening the right door, respectively.

The baseline is hDQN, where subtask is associated with an object and the state transitions in the high level are restricted to cases shown in Table 2 in Appendix, same as those in the propose method. However, our method learns a logic-based transition model in the high-level by ∂ILP (Evans & Grefenstette, 2018). Further, instead of interacting with the environment, the optimal sequence of subtasks is found by planning on the learned transition model, i.e., traversing over the abstract graph, which improves data-efficiency and introduces the interpretability of the learned policy.

Experiment Results In this environment, we learn transition rules for both symbolic states and reward function. Based on the experience tuples collected during the exploration, we induce the following transition rules via ∂ILP, by optimizing objectives in (3) and (4).

- 1: CurAct(X,Y), Connect(X,Y)→Next(Y)
- 2: ¬hasKey(), Next(4)→RewardKey()
- 3: hasKey(), Next(5)→RewardDoor().

Here node 4 and 5 denote the key and right door respectively. The rule 1 above denotes that the agent goes from node X to Y by successfully finishing subtask (X, Y). The rules 2 and 3 formulate the reward of obtaining the key and opening the right door. All of these rules are learned based on collected experience via ILP and unknown apriori.

As shown in Figure 5, the proposed method is around 50% faster than hDQN. That is because the sequence of subtasks is determined over the abstract graph. Although optimal low-level policies for the subtasks may not be well learned (subtask success rate < 0.9) in early episodes, the optimal subtask sequence with highest accumulated rewards can be discovered, as long as the connectivity of nodes in the graph is discovered. For the environment given in Figure 4(a), the optimal path on the abstract graph is $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 5$. Then the subtask solver can focus on subtasks along this sequence, without trying other subtasks, improving data efficiency significantly. We also present the success rate of 7 subtasks along the optimal path in Figure 6. It reflects more details on how the optimal solution is learned.

6. Conclusion

In this work, we proposed a new hierarchical framework for symbolic RL. In order to improve data efficiency and interpretability, leveraging the power of inductive logic programming (ILP), we learn a symbolic transition model in abstracted states and the high-level agent can also conduct learning over this learned model in addition to the real environment, which saves a lot of samples and improves data-efficiency. The transition rules induced by ILP can also reveal the working mechanism inherent in the symbolic information and the environment, introducing task-level interpretability and gaining more trust from the human user. In the future, we are going to investigate new hierarchical algorithm which can specify the state abstraction automatically without relying on any prior knowledge given by an expert.

References

Abel, D., Umbanhowar, N., Khetarpal, K., Arumugam, D., Precup, D., and Littman, M. Value preserving state-action abstractions. In *International Conference on Artificial Intelligence and Statistics*, pp. 1639–1650. PMLR, 2020.

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Barto, A. G. and Mahadevan, S. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Bunel, R., Hausknecht, M., Devlin, J., Singh, R., and Kohli, P. Leveraging grammar and reinforcement learning for neural program synthesis. In *International Conference on Learning Representations*, 2018.

Dong, H., Mao, J., Lin, T., Wang, C., Li, L., and Zhou, D. Neural logic machines. *arXiv preprint arXiv:1904.11694*, 2019.

Doshi-Velez, F. and Kim, B. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Došilović, F. K., Brčić, M., and Hlupić, N. Explainable artificial intelligence: A survey. In *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, pp. 0210–0215. IEEE, 2018.

Erdogan, S. T. and Lifschitz, V. Actions as special cases. In *KR*, pp. 377–388, 2006.

Evans, R. and Grefenstette, E. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

Falco, P., Attawia, A., Saveriano, M., and Lee, D. On policy learning robust to irreversible events: An application to robotic in-hand manipulation. *IEEE Robotics and Automation Letters*, 3(3):1482–1489, 2018.

Garcez, A., Gori, M., Lamb, L., Serafini, L., Spranger, M., and Tran, S. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *Journal of Applied Logics*, 6(4): 611–632, 2019.

Garcez, A. d., Dutra, A. R. R., and Alonso, E. Towards symbolic reinforcement learning with common sense. *arXiv preprint arXiv:1804.08597*, 2018.

Garnelo, M., Arulkumaran, K., and Shanahan, M. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*, 2016.

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., and Kagal, L. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pp. 80–89. IEEE, 2018.

Gopalan, N., desJardins, M., Littman, M. L., MacGlashan, J., Squire, S., Tellex, S., Winder, J., and Wong, L. L. Planning with abstract markov decision processes. In *27th International Conference on Automated Planning and Scheduling*, 2017.

Hasselt, H. v., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 2094–2100, 2016.

Illanes, L., Yan, X., Icarte, R. T., and McIlraith, S. A. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pp. 540–550, 2020.

Israelsen, B. W. and Ahmed, N. R. "dave... i can assure you... that it's going to be all right..." a definition, case for, and survey of algorithmic assurances in human-autonomy trust relationships. *ACM Computing Surveys (CSUR)*, 51 (6):1–37, 2019.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pp. 12519–12530, 2019.

Jiang, Z. and Luo, S. Neural logic reinforcement learning. In *International Conference on Machine Learning*, pp. 3110–3119. PMLR, 2019.

Koller, D., Friedman, N., Džeroski, S., Sutton, C., McCallum, A., Pfeffer, A., Abbeel, P., Wong, M.-F., Heckerman, D., Meek, C., et al. *Introduction to statistical relational learning*. MIT press, 2007.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29: 3675–3683, 2016.

Kumar, V., Gupta, A., Todorov, E., and Levine, S. Learning dexterous manipulation policies from experience and imitation. *arXiv preprint arXiv:1611.05095*, 2016a.

Kumar, V., Todorov, E., and Levine, S. Optimal control with learned local models: Application to dexterous manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 378–383. IEEE, 2016b.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. Model-ensemble trust-region policy optimization. In *International Conference on Learning Representations*, 2018.

Lan, Q., Pan, Y., Fyshe, A., and White, M. Maxmin q-learning: Controlling the estimation bias of q-learning. In *International Conference on Learning Representations*, 2019.

Le, H. M., Jiang, N., Agarwal, A., Dudík, M., Yue, Y., and Daumé, H. Hierarchical imitation and reinforcement learning. In *35th International Conference on Machine Learning, ICML 2018*, pp. 4560–4573. International Machine Learning Society (IMLS), 2018.

Leonetti, M., Iocchi, L., and Stone, P. A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artificial Intelligence*, 241:103–130, 2016.

Lu, K., Zhang, S., Stone, P., and Chen, X. Robot representation and reasoning with knowledge from reinforcement learning. *arXiv preprint arXiv:1809.11074*, 2018.

Lyu, D., Yang, F., Liu, B., and Gustafson, S. Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 2970–2977, 2019.

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

Payani, A. and Fekri, F. Learning algorithms via neural logic networks. *arXiv preprint arXiv:1904.01554*, 2019.

Peng, X. B., Berseth, G., Yin, K., and Van De Panne, M. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):1–13, 2017.

Puiutta, E. and Veith, E. Explainable reinforcement learning: A survey. *arXiv preprint arXiv:2005.06247*, 2020.

Rocktäschel, T. and Riedel, S. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems*, pp. 3788–3800, 2017.

Roscher, R., Bohn, B., Duarte, M. F., and Garcke, J. Explainable machine learning for scientific insights and discoveries. *IEEE Access*, 8:42200–42216, 2020.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

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

Tjoa, E. and Guan, C. A survey on explainable artificial intelligence (xai): Toward medical xai. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Van Hoof, H., Hermans, T., Neumann, G., and Peters, J. Learning robot in-hand manipulation with tactile features. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 121–127. IEEE, 2015.

Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pp. 5045–5054. PMLR, 2018.

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Yamamoto, K., Onishi, T., and Tsuruoka, Y. Hierarchical reinforcement learning with abductive planning. *arXiv preprint arXiv:1806.10792*, 2018.

Yang, F., Lyu, D., Liu, B., and Gustafson, S. Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. *arXiv preprint arXiv:1804.07779*, 2018.

Algorithm 1 Model-based Hierarchical Reinforcement Learning

```
Require: Propositional symbols (fluents) $\mathcal{F}$, labeling function $L$, termination condition $\alpha(\tilde{a}), \forall \tilde{a} \in \tilde{A}$, replay buffer $\mathcal{B}$,
    number of Q estimates $N$, exploration parameter $\epsilon > 0$, maximum steps in one episode $T_{\max}$, symbolic transition model
    $\tilde{P}_{\boldsymbol{\phi}}$;
 1: Initialize $N$ high-level Q networks $Q_h^1, \ldots, Q_h^N$ randomly;
 2: for $e = 1, \ldots,$ do
 3:     $l \leftarrow 0$
 4:     Reset the environment and observe the initial state s;
 5:     Obtain the current symbolic state $\tilde{s} \leftarrow L(s)$
 6:     while the goal has not been reached or $l < T_{\max}$ do
 7:         $Q_h^{\min}(\tilde{s}, \tilde{a}) \longleftarrow \min_{k \in \{1,\ldots,N\}} Q_h^k(\tilde{s}, \tilde{a}), \forall \tilde{a} \in \tilde{A}$
 8:         Choose symbolic action $\tilde{a}$ by $\epsilon$-greedy according to $Q_h^{\min}$
 9:         if e is even then
10:             Starting at initial state $s$, the subtask solver tries to solve subtask $(\tilde{s}, \tilde{a})$ for $K$ times.
11:             Update success ratio $t$ and number of trials $n$ for subtask $(\tilde{s}, \tilde{a})$
12:             Compute the extrinsic reward $r_e(\tilde{s}, \tilde{a})$ as (2)
13:             $s' \leftarrow s$ and $\tilde{s}' \leftarrow L(s)$
14:             if any successful trials then
15:                 Assign $s'$ by the last state of certain successful trial.
16:                 Update $\tilde{s}' \leftarrow L(s')$ and $s \leftarrow s'$
17:             end if
18:             Store the transition tuple $(\tilde{s}, \tilde{a}, \tilde{s'}, r_e)$ into replay buffer $\mathcal{B}$
19:         else
20:             Predict the reward $r_e$ and next state $\tilde{s}'$ by transition model $\tilde{P}_{\boldsymbol{\phi}}$
21:         end if
22:         Update Q networks
23:         $l \leftarrow l + 1$
24:         $\tilde{s} \leftarrow \tilde{s}'$
25:     end while
26:     Randomly sample a minibatch of transitions $\{(\tilde{s}, \tilde{a}, \tilde{s}', r_e)\}$ from $\mathcal{B}$, and fit the model $\tilde{P}_{\boldsymbol{\phi}}$ over these transitions by
        $\partial$ILP
27: end for
```

Algorithm 2 Maxmin Q Learning

```
Require: Q networks $Q_h^1, \ldots, Q_h^N$, and replay buffer $\mathcal{B}$
 1: Select a subset $S$ from $\{1, \ldots, N\}$
 2: for each $i \in S$ do
 3:     Randomly sample a minibatch of transitions $\{(\tilde{s}, \tilde{a}, \tilde{s}', r_e)\}$ from $\mathcal{B}$
 4:     Obtain the target for every sampled transition:
            $$Y \leftarrow r_e + \gamma \max_{\tilde{a} \in \tilde{A}} Q_h^{\min}(\tilde{s}', \tilde{a}) \qquad (5)$$
 5:     Update Q network $Q_h^i(\tilde{s}, \tilde{a}) \leftarrow Q_h^i(\tilde{s}, \tilde{a}) + \alpha[Y - Q_h^i(\tilde{s}, \tilde{a})]$, with step size $\alpha$.
 6: end for
```

Table 1: Definitions of Propositions (Predicates) in Room Environment

Name	Definition
CurAct(X, Y)	The room X the robot currently stays, and the intended symbolic action Y of the robot
Next(X)	The robot will come to room X at next time step
Visited(X)	Room X has been visited by the robot in current episode
Connect(X, Y)	Room X and room Y are connected by a corridor without any lock
RoomHasKey(X, A)	Room X has a key in color A
Lock(X, Y, A)	There is a lock between room X and Y in color A
RewardOpenLock()	Reward for opening a lock

Table 2: Subtasks Definition (From: source node, To: target node)

No.	From	To	hasKey()	Feasible
1	1	2	False	Yes
2	2	3	False	Yes
3	3	4	False	Yes
4	4	3	True	Yes
5	3	2	True or False	Yes
6	2	1	True or False	Yes
7	1	5	True	Yes
8	2	6	True or False	Yes
9	6	4	True or False	No
10	1	5	False	Yes
11	2	4	True or False	No
12	4	2	True	No
13	2	5	True	No