Hierarchical Reinforcement Learning with AI Planning Models

Junkyu Lee
IBM Research
junkyu.lee@ibm.com

Michael Katz
IBM Research
Michael.Katz1@ibm.com

Don Joven Agravante
IBM Research
don.joven.r.agravante@ibm.com

Miao Liu
IBM Research
miao.liu1@ibm.com

Geraud Nangue Tasse
University of the Witwatersrand
geraudnt@gmail.com

Tim Klinger
IBM Research
tklinger@us.ibm.com

Shirin Sohrabi
IBM Research
ssohrab@us.ibm.com

Abstract

Deep Reinforcement Learning (DRL) has shown breakthroughs in solving challenging problems, such as pixel-based games and continuous control tasks. In complex environments, infusing prior domain knowledge is essential to achieve sample efficiency and generalization. Neuro-symbolic AI seeks systematic domain knowledge infusion into neural network-based learning, and existing neuro-symbolic approaches for sequential decision-making leverage hierarchical reinforcement learning (HRL) by infusing symbolically specified prior knowledge on desired trajectories. However, this requires finding symbolic solutions in RL environments before learning, and it is difficult to handle the divergence between unknown RL dynamics and prior knowledge. Such shortcomings result in loose and manual neuro-symbolic integration and degrade the generalization capability. In this paper, we integrate the options framework in HRL with an AI planning model to resolve the shortcomings in earlier approaches and generalize beyond RL environments where pre-specified partial solutions are valid. Our approach defines options from AI planning operators by establishing the connection between the two transition systems in the options framework and the AI planning task. Then, we show an option policy learning method that integrates an AI planner and model-free DRL algorithms with intrinsic rewards, encouraging consistency between the two transition systems. We design a suite of MiniGrid environments that cover the increasing levels of difficulties in exploration, where our empirical evaluation clearly shows the advantage of HRL with AI planning models. The code is available at https://github.com/IBM/parl_agents and https://github.com/IBM/parl_annotations.

1 Introduction

Neuro-symbolic AI [1] combines neural network-based learning and symbolic reasoning in a principled manner [2] to leverage deep learning with data efficiency and trustworthiness. Deep reinforcement learning (DRL) has shown remarkable achievements in solving sequential decision-making problems in high-dimensional feature space or complex continuous control problems, reaching the super-human level performance [3, 4]. DRL is agnostic to the domain knowledge, but it learns non-interpretable and brittle policies sensitive to delicate changes in the input features. Advances in

NeurIPS 2023 Workshop on Generalization in Planning (GenPlan 2023)

hardware won't resolve the data inefficiency since many real-world problems violate ideal assumptions in deep reinforcement learning. A remedy would be injecting prior knowledge as inductive bias [5] to neural architectures or optimization formulation, which heavily relies on human design.

Neuro-symbolic approaches address how to infuse prior domain knowledge systematically to augment deep learning methods with robustness and reusability [6, 7, 8]. A widely accepted neuro-symbolic architecture integrates a low-level neural perception layer and a high-level symbolic reasoning layer, reflecting cognitive theory *Thinking Fast and Slow* [9]. In this paper, we focus on neuro-symbolic approaches that integrate symbolic reasoning and hierarchical reinforcement learning (HRL). HRL [10] improves sample efficiency by decomposing the inherent task structures in abstract states and action spaces [11]. Notable earlier frameworks, such as options framework [12] and MAXQ [13], rely on precise domain knowledge to define options or value function decomposition. Recent work extend these frameworks with deep neural networks. Learning-only approaches [14], however, do not work well for RL environments with complex task structure, where the main challenges are both handling the state space dimensionality and exploration issues due to sparse rewards or dead-end states. Thus, recent years have seen growing interest in integrating symbolic methods into DRL.

In reward machines (RM) [15], temporal relations between symbolic events detected in RL environments are specified in linear temporal logic (LTL), and translated into finite state machines (FSM). Hierarchical RMs (HRM) map state transitions over the FSM to options [16, 17, 18, 19]. Similar to RM, taskable RL [20] infuses temporal relations between AI planning operators to define options. Symbolic DRL (SDRL) integrates action language and answer set programming into HRL [21].

Although existing neuro-symbolic approaches improved the sample efficiency in goal-oriented RL environments, they still have significant room for improvement. First, HRM lacks a symbolic reasoning layer, and it directly infuses abstract state trajectory in an ad-hoc manner, mapping a single transition or multiple state transitions in FSM to an option. Furthermore, RMs assume that RL transitions strictly follow the LTL formula. SDRL maps each abstract state transition to an option, resulting in a large number of options $O(|S|^2)$, where $|S|$ denotes the total number of symbolic states.

To overcome the shortcomings in earlier approaches, we integrate AI planning and options framework [12]. Infusing AI planning models for decomposing the task structure in HRL has several advantages. Symbolic reasoning is tightly combined in HRL and generates symbolic plans on the fly, allowing us to systematically generalize to multi-task RL environments that share the abstract transition model. We can relax the earlier assumption that prior knowledge should capture the precise behavior in unknown RL dynamics, and offer a mechanism for handling the divergence between an AI planning task and unknown RL dynamics. We designed a suite of `MiniGrid` environments [22, 23] associated with AI planning models and empirically show the advantage of HRL with AI planning models.

2 Background

Goal-oriented Environments In this paper, we focus on a Markov Decision Process (MDP) in the context of generalized stochastic shortest-path MDPs [24], where MDP is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, s_0, G, \gamma \rangle$ with states $\mathcal{S}$, actions $\mathcal{A}$, a state transition function $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$, a reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, an initial state $s_0 \in \mathcal{S}$, a set of goal states $G \subset \mathcal{S}$, and a discounting factor $\gamma \in (0, 1)$ for the rewards. Unlike usual assumptions on MDP environments, we don't assume the goal to be reachable from any state, allowing dead-end states. Additionally, we simplify a sparse reward function to return a positive reward upon reaching the goal G and 0 otherwise, allowing 0-reward cycles. Note that value iteration is not guaranteed to converge to the optimal value in generalized stochastic shortest-path MDPs [24]. In this goal-oriented environment, our objective is to learn a stationary policy $\pi^*_{\mathcal{M}}$ that maximizes the expected reward starting from the initial state s_0, $\pi^*_{\mathcal{M}} = \arg\max_\pi \mathbb{E}_\pi \big[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 \big]$, where $\pi(a|s)$ is a stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$.

Multi-task Environments In real-world applications, a collection of MDPs may share a common task structure or a meta-model, where muti-task reinforcement learning (MTRL) aims to improve the sample efficiency and, more importantly, generalize to solve unseen tasks [25, 26, 27]. In multi-task environments, the objective is to learn a policy π^* that maximizes the expected reward over the distribution of tasks, $\pi^* = \arg\max_\pi \mathbb{E}_{\mathcal{M}^i \sim p(\mathcal{M})} \Big[\mathbb{E}_\pi \big[\sum_{t=0}^{\infty} \gamma^t r^i_t | s^i_0 \big] \Big]$, where each MDP $\mathcal{M}^i = \langle \mathcal{S}, \mathcal{A}, P^i, r^i, s^i_0, G^i, \gamma \rangle$ is sampled from $p\big(\mathcal{M}\big)$. Some common choices for the task generation process $p\big(\mathcal{M}\big)$ are parameterizing $\mathcal{M}^i$ with contexts [27], or incorporating a task prior [25].

Options Framework in HRL In options framework [28], a set of options $\mathcal{O}^*$ formalizes the temporally extended actions that defines a semi-MDP (SMDP) over the original MDP $\mathcal{M}$. A Markovian option $O^* \in \mathcal{O}^*$ is a triple $\langle \mathcal{I}_{O^*}, \pi_{O^*}, \beta_{O^*} \rangle$, where $\mathcal{I}_{O^*}$ is the initiation set where O^* begins, π_{O^*} is an option policy $\pi_{O^*} : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, and β_{O^*} is a termination set where O^* terminates. Following the call-and-return option execution model, an agent selects an option O^* in state s at time t if $s \in \mathcal{I}_{O^*}$, and generates a sequence of actions according to the option policy $\pi_{O^*}(a|s)$ until it reaches β_{O^*}.

AI Planning We follow the SAS$^+$ formalism for representing planning tasks [29]. A planning task Π is a tuple $\langle \mathcal{V}, \mathcal{O}, S'_0, S_* \rangle$ of multi-valued state variables $\mathcal{V}$ and operators $\mathcal{O}$. Each state variable $v \in \mathcal{V}$ has a finite domain $dom(v)$ of values. A (partial) assignment to $\mathcal{V}$ is called a (partial) state, with the full state S'_0 being the initial state and the partial state S_* being the goal. A fact is a pair $\langle v, \vartheta \rangle$ that assigns a value $\vartheta \in dom(v)$ to a variable v, and we denote the variables in a partial assignment p by $\mathcal{V}(p)$, and the value ϑ assigned to v in p by $p[v] = \vartheta$. It is convenient to view a partial state p as a set of facts with $\langle v, \vartheta \rangle \in p$ if and only if $p[v] = \vartheta$. A partial state p is consistent with state s if $p \subseteq s$. We denote the set of states of Π by $\mathcal{S}'$. Each operator $o \in \mathcal{O}$ is a pair $\langle \mathit{pre}(o), \mathit{eff}(o) \rangle$ of partial states called preconditions and effects. The (possibly empty) subset of preconditions that do not involve variables from the effect is called prevail condition, $\mathit{prv}(o) = \{\langle v, \vartheta \rangle \mid \langle v, \vartheta \rangle \in \mathit{pre}(o), v \notin \mathcal{V}(\mathit{eff}(o))\}$. An operator o is applicable in a state $s \in \mathcal{S}'$ if and only if $\mathit{pre}(o)$ is consistent with s ($\mathit{pre}(o) \subseteq s$). Applying o changes the value of v to $\mathit{eff}(o)[v]$, if defined. The resulting state is denoted by $s[\![o]\!]$. An operator sequence $\pi = \langle o_1, \ldots, o_k \rangle$ is applicable in s if there exist states $s_0, \cdots, s_k$ such that (1) $s_0 = s$, and (2) for each $1 \leq i \leq k$, $\mathit{pre}(o_i) \subset s_{i\text{-}1}$ and $s_i = s_{i\text{-}1}[\![o_i]\!]$. We denote the state s_k by $s[\![\pi]\!]$. π is a plan for s iff π is applicable in s and $S_* \subseteq s[\![\pi]\!]$.

A transition graph of a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, S'_0, S_* \rangle$ is a triple $\mathcal{T}_\Pi = \langle \mathcal{S}, T_\Pi, S_* \rangle$, where $\mathcal{S}$ are the states of Π, $T_\Pi \subseteq \mathcal{S} \times \mathcal{O} \times \mathcal{S}$ is a set of labeled transitions, and $S_* \subseteq \mathcal{S}$ is the set of goal states. An abstraction of the transition graph $\mathcal{T}$ is a pair $\langle \mathcal{T}', \alpha \rangle$, where $\mathcal{T}' = \langle \mathcal{S}', T', S'_* \rangle$ is an abstract transition graph and $\alpha : \mathcal{S} \mapsto \mathcal{S}'$ is an abstraction mapping, such that $\langle \alpha(s), o, \alpha(s') \rangle \in T'$ for all $\langle s, o, s' \rangle \in T$, and $\alpha(s) \in S'_*$ for all $s \in S_*$.

3 Symbolic Abstraction of RL Environments

We first formulate an AI planning task as a symbolic abstraction of goal-oriented RL environments. Then, we define *plan options* derived from planning operators and introduce implicit frame constraints to the options framework, with intrinsic rewards that enforce consistency between the two models.

3.1 Options from Planning Operators

Neuro-symbolic methods integrate the neural and symbolic layers via mapping representation of the two. Various approaches were proposed, such as semantic parsing for the text-based game environments [30, 31], training neural predicates [32, 6], or geometric mapping in robotics environments [33, 34]. To integrate AI planning and HRL, we define a state mapping function as follows.

Definition 1 (state mapping function) Given a planning task $\Pi := \langle \mathcal{V}, \mathcal{O}, S'_0, S_* \rangle$ and a goal-oriented MDP $\mathcal{M} := \langle \mathcal{S}, \mathcal{A}, P, r, s_0, G, \gamma \rangle$, we define a state mapping function $L : \mathcal{S} \mapsto \mathcal{S}'$ as a surjective mapping from the MDP states $\mathcal{S}$ to symbolic states $\mathcal{S}'$ such that $s'_0 = L(s_0)$ and s_* is consistent with $L(s)$ for all $s \in G$. $L^{-1}(s')$ denotes the pre-image of $s' \in \mathcal{S}'$, namely, a subset of MDP states $\{s \in \mathcal{S} | L(s) = s'\}$.

Definition 2 (proper abstraction) Let $\mathcal{T}_\Pi := \langle \mathcal{S}', T_\Pi, S_* \rangle$ be a transition graph of Π and $\mathcal{T}_\mathcal{M} := \langle S, T_\mathcal{M}, G \rangle$ be a transition graph of $\mathcal{M}$. We say $\langle \mathcal{T}_\Pi, L \rangle$ is a proper abstraction of $\mathcal{T}_\mathcal{M}$ if for all state transitions $\langle s, a, s' \rangle \in \mathcal{T}_\mathcal{M}$, we have $P(s'|s, a) > 0$, iff $\langle L(s), o, L(s') \rangle \in T_\Pi$ for some $o \in \mathcal{O}$ or $L(s) = L(s')$.

In words, state mapping functions partition the state space of a goal-oriented MDP $\mathcal{M}$ into planning task Π states while preserving the initial state and the goal states, and a proper abstraction ensures that every state transition $\langle s, a, s' \rangle \in \mathcal{M}$ maps to either a symbolic state transition in Π or s' remains in the same symbolic state $L(s)$. Next, we define options in HRL from operators in a planning task.

Definition 3 (plan options) Given a planning task Π, a goal-oriented MDP task $\mathcal{M}$, and a state mapping function L, we define *plan options* as follows.
For each planning operator $o \in \mathcal{O}$, an *operator option* is a tuple $O^*_o := \langle \mathcal{I}_{O^*_o}, \pi_{O^*_o}, \beta_{O^*_o} \rangle$, where

$$\mathcal{I}_{O^*_o} := \{s \in \mathcal{S} \,|\, \mathit{pre}(o) \subseteq L(s)\} \text{ and } \beta_{O^*_o} := \{s \in \mathcal{S} \mid (\mathit{prv}(o) \cup \mathit{eff}(o)) \subseteq L(s)\}.$$

For a goal G in $\mathcal{M}$, a *goal option* is a tuple $O^*{}_* := \langle \mathcal{I}_{O^*{}_*}, \pi_{O^*{}_*}, \beta_{O^*{}_*} \rangle$, where

$$\mathcal{I}_{O^*{}_*} := \{s \in \mathcal{S} \mid s_* \subset L(s)\} \text{ and } \beta_{O^*{}_*} := G.$$

We denote a set of plan options by $\mathcal{O}^*_{\mathcal{M}}$.

In words, we declare options from planning operators by mapping the precondition of a planning operator to the initiation set and the postcondition to the termination set. Earlier works also have suggested using symbolic transitions to define options: [20] assumes additional manual mapping and [21] maps every symbolic transition to a separate option, both without specifying the initiation sets.

A set of plan options induces an SMDP $\mathcal{M}' := \langle \mathcal{S}, \mathcal{O}^*_{\mathcal{M}}, P, r, s_0, G, \gamma \rangle$, where we replaced the primitive actions $\mathcal{A}$ in $\mathcal{M}$ with $\mathcal{O}^*_{\mathcal{M}}$. Next, we define a transition graph $\mathcal{T}_{\mathcal{M}'}$ of $\mathcal{M}'$, where a multi-step state transitions during the execution of an option O^*_o is collapsed to a single labeled transition that connects each a state $s \in \mathcal{I}_{O^*_o}$ to the states $s' \in \beta_{O^*_o}$.

Definition 4 (SMDP transition graph) *A transition graph of an SMDP* $\mathcal{M}' := \langle \mathcal{S}, \mathcal{O}^*_{\mathcal{M}}, P, r, s_0, G, \gamma \rangle$ *is a tuple* $\mathcal{T}_{\mathcal{M}'} := \langle \mathcal{S}, T_{\mathcal{M}'}, G \rangle$*, where* $\mathcal{S}$ *is the states of* $\mathcal{M}$*,* $T_{\mathcal{M}'}$ *is a set of non-deterministic labeled transitions* $\{\langle s, o, s' \rangle \mid s \in \mathcal{I}_{O^*_o}, s' \in \beta_{O^*_o}, P(s'|s, O^*_o) > 0\}$*, and* G *is the goal states in* $\mathcal{M}$*.*

3.2 Frame Constraints in Plan Options

Although we do not assume to have an exact model of $\mathcal{M}$, it is desirable to have a planning task Π that behaves similarly to $\mathcal{M}$. To characterize the similarity between the two tasks, we introduce the context and frame of an option O^*_o in an MDP state s to capture the subset of symbolic facts in the planning task that prevail after applying a planning operator o to the planning state $L(s)$.

Definition 5 (context and frame of an option) Given a planning operator o and the operator option O^*_o, we define the *context* of O^*_o in an MDP state $s \in \mathcal{S}$ by $\mathcal{C}_{O^*_o}(s) := L(s) \setminus (\mathit{pre}(o) \cup \mathit{eff}(o))$, and the *frame* of O^*_o in an MDP state $s \in \mathcal{S}$ by $\mathcal{F}_{O^*_o}(s) := \mathit{prv}(o) \cup \mathcal{C}_{O^*_o}(s)$. We call a subset of a frame $\mathcal{F}_{O^*_o}(s)$ in an MDP state s as a partial frame, and denote it by $\mathcal{F}^p_{O^*_o}(s)$.

Intuitively, the context and frame of an operator option reflects the implicit constraints, a planning operator only modifies the facts that appear in the operator effect. We say an SMDP transition due to a plan operator option O^*_o is frame preserving if $\mathcal{F}_{O^*_o}(s) = \mathcal{F}_{O^*_o}(s')$ for every $\langle s, o, s' \rangle \in T_{\mathcal{M}'}$, and $\mathcal{T}_{\mathcal{M}'}$ is frame preserving if it holds for all operator options in $\mathcal{O}^*_{\mathcal{M}}$. We can formally state the similarity between the two transition systems under plan options as follows.

Theorem 1 If $\mathcal{T}_{\mathcal{M}'}$ is frame preserving under the operator options $\mathcal{O}^*_{\mathcal{M}}$ derived from a planning task Π, then $\mathcal{T}_{\Pi}$ and $\mathcal{T}_{\mathcal{M}'}$ are bisimilar.

Proof: Consider a binary relation $\{\langle s, t \rangle \in \mathcal{S} \times \mathcal{S} \,|\, L(s) = L(t)\}$. For a pair of a planning operator o and the operator option $O^*{}_o$, every $\langle s, O^*{}_o, t \rangle \in T_{\mathcal{M}'}$ satisfies $L(t) = \big[L(s) \setminus \big(\mathit{pre}(o) \cup \mathcal{F}_{O^*_o}(s)\big)\big] \cup \big(\mathit{eff}(o) \cup \mathcal{F}_{O^*_o}(s)\big) = L(s)[\![o]\!]$. For a transition $\langle t, o, t' \rangle \in T_{\mathcal{M}'}$ such that $L(t) = L(s)$, $L(t') = L(t)[\![o]\!] = L(s)[\![o]\!]$. ∎

The desiderata in HRL is that a task hierarchy captures the decomposition of $\mathcal{M}$ into sub-MDPs that are easier to solve in a local state space, and those sub-tasks are reusable. A sub-problem analysis by [35] shows that HRL methods can improve the sample efficiency if the total sum of the size of each partitioned state space is smaller than the size of the original state space. Following this intuition behind the MDP decomposition in HRL, we now characterize the sub-problem decomposition imposed by the frame-constrained option MDPs.

Algorithm 1 Online Option Learning

Require: Planning task Π, Goal-oriented MDP $\mathcal{M}$, State mapping function L
Ensure: Option policies $\pi_{O^*{}_o}(a|s)$.

```
Initialize trajectory buffer B
Initialize a set D for storing options
while iter < N do
    // rollout samples from the current option policy
    while iter_rollout < N_rollout do
        s ← current state
        Select an option O*_o where o is the first planning operator of a plan starting in L(s), found by AI planner
        if O*_o ∉ D then
            Create plan option O*_o
            Initialize π_{O*_o} and Add O*_o to D
        while s ∉ β_{O*_o} do
            Sample (s, a, r_e, u) from RL environment
            Compute intrinsic reward r_i
            Store (O*_o, s, a, r_e, r_i) to buffer B
            s ← u
    // train option policies with model-free algorithms
    for each option O*_o ∈ D do
        Train option policy function π_{O*_o} with RL
```

Definition 6 (frame constrained option sub-MDP) Given a goal-oriented MDP $\mathcal{M}$ and an operator option derived from a planning task $O^*_o{:=}\langle\mathcal{I}_{O^*_o},\pi_{O^*_o},\beta_{O^*_o}\rangle$, a *frame constrained option sub-MDP* is a sub-MDP of $\mathcal{M}$, $\mathcal{M}_{o,s_0} := \langle\mathcal{S}_{\mathcal{F}_o(s_0)}, \mathcal{A}, P_{\mathcal{F}_o(s_0)}, r, s_0, \beta_{O^*_o}, \gamma, \mathcal{D}_{\mathcal{F}_o(s_0)}\rangle$, where $\mathcal{S}_{\mathcal{F}_o(s_0)}$ is the local state space, $P_{\mathcal{F}_o(s_0)}$ is a frame constrained state transition probability, $s_0 \in \mathcal{I}_{O^*_o}$ is an initial state, $\beta_{O^*_o}$ is the goal, and $\mathcal{D}_{\mathcal{F}_o(s_0)}$ is a set of constraints enforcing the state transitions to preserve $\mathcal{F}_o(s_0)$, $\{\mathcal{F}_{O^*_o}(s') = \mathcal{F}_{O^*_o}(s_0) \mid \forall\langle s, a, s'\rangle \in \mathcal{T}_{\mathcal{M}_{o,s_0}}, \pi_{O^*_o}(a|s) > 0\}$.

We can see that introducing frame constraints to each option MDP reduces the size of the state space subject to the number of facts in the frame of the option.

Theorem 2 For frame-constrained option sub-MDPs $\mathcal{M}^p_{o,s_0}$ and $\mathcal{M}^q_{o,s_0}$ induced by partial frames $\mathcal{F}^p_{O^*_o}(s_0)$ and $\mathcal{F}^q_{O^*_o}(s_0)$, if $\mathcal{F}^p_{O^*_o}(s_0) \subset \mathcal{F}^q_{O^*_o}(s_0)$, the state space of $\mathcal{M}^p_{o,s_0}$ is a super-set of $\mathcal{M}^q_{o,s_0}$.

Proof: Let $\mathcal{S}_p$ and $\mathcal{S}_q$ denote the states of the $\mathcal{M}^p_{o,s_0}$ and $\mathcal{M}^q_{o,s_0}$. For every $s \in \mathcal{S}_q$, we can see that $s \in \mathcal{S}_p$ since $\mathcal{F}^p_{O^*_o}(s_0) \subset \mathcal{F}^q_{O^*_o}(s_0) \subset L(s)$. ∎

If all option MDPs are frame-constrained, we have two advantages: (1) *improved sample efficiency* due to the reduction in the state space, and (2) options *re-usability by composition*, relying solely on symbolic knowledge.

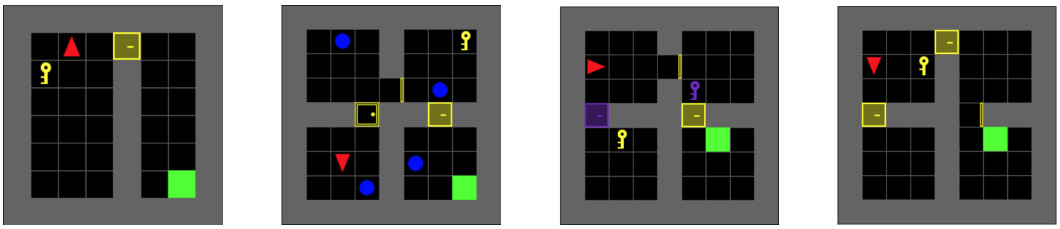
3.3 Intrinsic Rewards for Learning Plan Options

In practice, we don't assume that a planning task Π exactly simulates the underlying $\mathcal{M}$. Therefore, we relax the constraints in the frame-constrained option sub-MDPs and reformulate optimization objective function with a penalty imposed per violation of frame constraints, resulting in negative intrinsic rewards for option learning.

Definition 7 (frame penalized option sub-MDP) Given a goal-oriented MDP $\mathcal{M}$ and an operator option derived from a planning task $O^*_o{:=}\langle\mathcal{I}_{O^*_o},\pi_{O^*_o},\beta_{O^*_o}\rangle$, a *frame penalized option sub-MDP* is a sub-MDP of $\mathcal{M}$. $\overline{\mathcal{M}}_{o,s_0} := \{\mathcal{S}, \mathcal{A}, P, \overline{r}, \overline{s}_0, \beta_{O^*_o}, \gamma\}$, where we replace the reward function in $\mathcal{M}$ with an intrinsic reward $\overline{r} := \mathcal{S} \times \mathcal{S} \to \mathbb{R}$,

$$\overline{r}(s, s') = \sum_{v \in \mathcal{V}\left(\mathcal{F}_{O^*_o}(s)\right)} c_1 \cdot \mathbb{I}\big(L(s')[v] \neq \mathcal{F}_{O^*_o}[v]\big) + c_2 \cdot \mathbb{I}\big(s' \notin \beta_{O^*_o}\big), \tag{1}$$

where $\mathbb{I}$ is an indicator function, c_1 and c_2 are negative rewards, and the initial state and the goal with a $\overline{s}_0 \in \mathcal{I}_{O^*_o}$, and $\beta_{O^*_o}$, respectively.



(a) MiniGrid Door Key (b) 2x2 Locked Door (c) 2x2 Two Keys (d) 2x2 One-Use Key

Figure 1: Example of `MiniGrid` environments: From left to right, we show a sample MDP $\mathcal{M}$ paired with a planning task. A symbolic state of the planning task is an abstract state of $\mathcal{M}$. The prior domain knowledge encoded in a symbolic planning model is the room layouts, the location of objects known up to the room, and the state of doors, locked or not. Each $\mathcal{M}$ varies the location of objects within the room, or the location of doors or balls, and the additional state of doors, closed or open,

In the above definition, $\overline{r}(s, s')$ is Markovian since the first term is a frame-based penalty depending on two consecutive states, and the second term is a penalty for the current state s' being outside of the termination set. Note that the state space of the frame penalized option MDP can be as large as the original state space in the worst case. However, if prior knowledge correctly captures the high-level structure of $\mathcal{M}$, the state space per option sub-MDPs will be localized and empirical results show improvement in the sample efficiency.

4 Online Learning Plan Options

In this section, we present HRL algorithms that learn plan options. For any pair of initial state $s_0 \in \mathcal{S}$ and a goal $s_g \in G$ in $\mathcal{M}$, we can generate a sequence of options $\{O^*_{o_1}, O^*_{o_2}, \ldots, O^*_{o_k}\}$ from a plan in Π that reaches the goal state $L(s_g) \in \mathcal{S}'$ from the initial state $L(s_0) \in \mathcal{S}'$. Therefore, we can integrate AI planners in two ways, either pre-compute option-level plans offline or generate plans online while training option policies. In this paper, we focus on the online approach that integrates AI planner as a higher-level symbolic reasoning agent and model-free RL as a lower-level neural network learning agent in the neuro-symbolic architecture. Algorithm 1 shows the outline of the HRL agent that learns plan options online. The algorithm alternates the rollout and training phases until the limit on the number of iterations is reached. During the rollout phase, HRL agent selects an option O^*_o by solving a planning task Π with the initial planning state s'_0 being the current planning state $L(s)$ and returning the applicable planning operator in s'_0 (line 6). Note that this re-planning step at every option selection is similar to the action selection in online planning [36], and empirical results show that it is much more efficient than learning-based approaches. If O^*_o was not created before, we create the option and initialize the policy $\pi_{O^*_o}$ and add it to a set D (lines 7-9). Next, sample trajectories are generated by using $\pi_{O^*_o}$ until it terminates, and for each state transition, we compute the intrinsic reward following Definition 7 (lines 10-14). Then, HRL agent updates the option policy using the samples stored in the buffer during the training phase by any model-RL algorithm (line 15). In our experiments, we integrated A^* search algorithm implemented in `Pyperplan` [40] with Proximal Policy Optimization `PPO` [41] for option policy training algorithm, which we call `HplanPPO`. Since `PPO` is an on-policy algorithm, option training phases do not share samples across options.

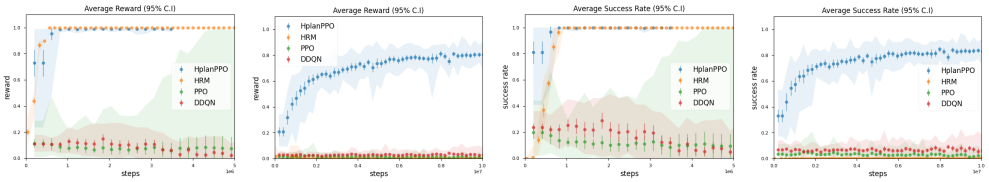
5 Experiment

All experiments are conducted in a cluster computing environment equipped with Intel (R) Xeon(R) Gold 6258R CPUs and NVIDIA A-100/V-100 GPUs. For each run, we limited computational resources to utilize up to 16 GB of memory with 2 CPUs and 1 GPU.

5.1 Benchmark Domains

For HRL experiments, we extended `MiniGrid` environments with task structures captured by AI planning tasks [1]. All those environments are grid navigation domains, where an agent uses keys to

[1]The code is available at https://github.com/IBM/parl_minigrid.



(a) Door Key (b) 2x2 Locked Door (c) Door Key (d) 2x2 Locked Door

Figure 2: Baseline Comparison in `Door Key` and `2x2 Locked Door`: Each figure shows the average reward and the success rate of four algorithms evaluated on the evaluation tasks. The x-axis is the number of sample steps, and the y-axis is the reward or success rate. The success rate only counts on whether an episode reached the goal within the maximum episode length, 2048.

open locked doors to move from the initial locations to the desired location. To study the impact of sparse rewards and dead-end states on training HRL agents, we designed benchmark problems considering the following aspects.

- In grid navigation environments, random restarting at the initial location helps exploration in relatively small 2D grids. Therefore, we only allow random variations on the initial and goal locations within pre-specified rooms.
- We used the *sparse reward* in the original `MiniGrid` environment without any reward shaping, namely, the reward is all zero but $\big(1 - 0.9 \cdot (S/L)\big)$ when the agent reaches the goal location after S steps. L is the maximum episode length.
- To introduce *dead-end states*, we introduce RL environments, where a key can only be used once.
- During training and evaluation, we sample a goal-oriented MDP $\mathcal{M} \sim p(\mathcal{M})$ in multi-task environments, where each $\mathcal{M}$ shares a common structure captured by an AI planning task, such as the connectivity of rooms, and the state of the keys or doors. However, all MDP tasks vary on the location of the doors, distribution of distracting balls, location of the goal tiles, location of the keys, etc.

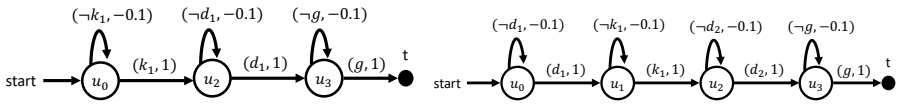
Figure 1 shows four selected `MiniGrid` MDP instances:

- `Door Key` environment in Figure 1a is the one already existing in the standard `MiniGrid` environment,
- `2x2 Locked Door` environment in Figure 1b can vary the initial location from any of the two rooms on the left, and randomly distributed blue balls, which are not visible in symbolic planning state space, must be removed if it blocks the way.
- `2x2 Two Keys` environment in Figure 1c requires an agent to navigate rooms back and forth to unlock the doors, and
- `2x2 One-Use Key` environment in Figure 1d may encounter a dead-end state if the key was used to unlock the wrong door.

In the following experiments, we controlled the environment to generate 1 million MDP instances for training and 1 thousand MDP instances for evaluation. In addition, we also created pure symbolic environments that generates trajectories of symbolic states to evaluate the performance of option-level policy learning.

5.2 Algorithms

We evaluated `HplanPPO` and several baseline algorithms. For model-free DRL, we selected `PPO`, `DDQN`, `PPO` with curiosity module, and `RAINBOW`. For HRL baseline, we modified open-sourced version of Deep Hierarchical Reward Machines (`HRM`) [17]. We utilized existing RL libraries `stable-baselines3` [37] for `HplanPPO`, `PPO`, and `DDQN`, `rllib` [38] for `PPO` with curiosity module, and `RAINBOW`. Due to the space limit, we will provide details on general experiment setups and implementation of deep neural network architectures and hyper-parameter choices in Appendix. For all experiments, we evaluated each algorithm 10 to 20 times and sampled up to 10 million



(a) Door Key Environment (b) 2x2 Locked Door Environment

Figure 3: Reward Machines for HRM: A state transition in the FSM can be triggered by the occurrence of an event, where d_i, k_i, and g represent the predicates for the i-th door unlocked, holding the i-th key, and the goal room was reached. The numbers next to an event predicate are rewards.

frames. We measured the moving average of episode lengths, success rates, and rewards over one hundred randomly selected evaluation tasks, and summarized them by the average, the minimum, and maximum range, and 95 percent confidence intervals over 10 to 20 trials.

5.3 Comparison against Baselines

Average Rewards and Success Rates Figure 2 shows the average rewards and success rates evaluated on the unseen evaluation tasks during training. The results from `Door Key` environment in Figure 2a and 2c show that two HRL algorithms, `HplanPPO` and `HRM`, were able to train policies that generalize to unseen evaluation tasks with the average reward and success rate close to 1.0. On the other hand, the average of rewards and success rates of `PPO` and `DDQN` are less than 0.2 with high variance, `RAINBOW` achieved the average and the maximum rewards 0.310 and 0.667, respectively, and `PPO` with curiosity [39] achieved 0.29 and 0.884, respectively. The evaluation result from `2x2 Locked Door` environment in Figure 2b and 2d show that all but `HplanPPO` failed. `RAINBOW` achieved the average and the maximum rewards 0.094 and 0.376, and `PPO` with curiosity [39] achieved 0.042 and 0.677.

RMs Figure 3 shows the reward machines [17] of the two environments. Since reward machines infuse symbolic solutions, FSM is a chain with guard conditions on symbolic events that trigger state transitions.

Number of Options When we examine the number of options trained, an agent can solve problems by training only three options, *pick up a key*, *unlock the door*, and *move to the adjacent room*, in `Door Key` environments. However, in `2x2 Locked Door`, the starting location can be any of the two rooms on the left, meaning that sometimes the door between the two rooms on the left should be opened, and sometimes not. `HplanPPO` was able to successfully solve the environment by training 9 plan options and 1 goal option on 20 trials [2].

5.4 Impact of Intrinsic Rewards

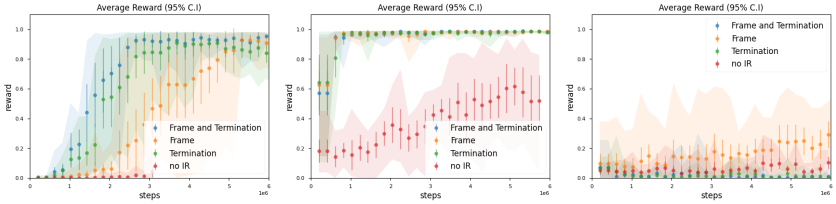
Figure 4 shows the impact of intrinsic rewards in `2x2 Two Keys`, `2x2 One-Use Key`, and `2x2 Two One-Use Keys` environments. Note that all other baseline algorithms failed to learn a policy. We can see that intrinsic rewards play an important role in option training. In `2x2 Two Keys`, `HplanPPO` without intrinsic rewards consistently failed on all trials. The shortest plan length is 11 for solving `2x2 Two Keys`, but `HplanPPO` trained 16 options. Ideally, the agent would pick up the purple key in the upper right room and directly move to unlock the purple door. However, the agent could drop the key before using the key. Therefore, we see that `HplanPPO` trained option to pick up the purple key in the upper left corner.

In the next two environments, the agent may encounter dead-end states due to the one-use keys. Although the shortest plan length for `2x2 One-Use Key` and `2x2 Two One-Use Keys` are 4 and 7, which are shorter than the plans in `2x2 Two Keys`, we see the performance is worse due to the dead-end states.

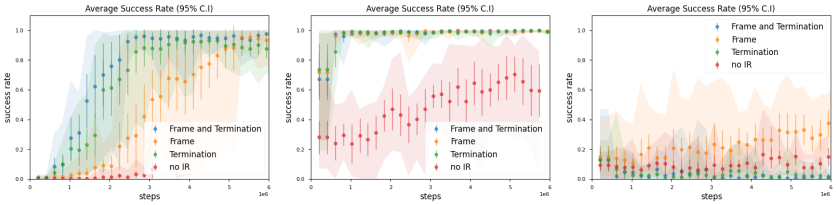
5.5 Training Option Level Policy

Existing integrated approaches such as Symbolic DRL (SDRL) [21] learn the value function or policy networks for the symbolic states. In this experiment, we trained `PPO` agent directly on the symbolic

[2] For the full option sequence as well as results from other environments, please see Appendix.



(a) 2x2 Two Keys (b) 2x2 One-Use Key (c) 2x2 Two One-Use Keys



(d) 2x2 Two Keys (e) 2x2 One-Use Key (f) 2x2 Two One-Use Keys

Figure 4: Impact of Intrinsic Rewards: 4 different configurations for defining Markovian intrinsic rewards as shown in Definition 7. `Frame and Termination` configuration uses both penalty terms subject to the frame and the termination set, `Frame` uses penalty terms subject to the frame only, `Termination` uses penalty terms subject to the termination set only, and `No IR` does not utilize intrinsic rewards. The average rewards are shown in (a,b,c), the success rates in (d,e,f).

RL environments that generates symbolic trajectories [3]. This case is akin to training high-level policy in an ideal situation, where all options are perfectly trained. In this experiment, we made the RL task easier than earlier multi-task settings by fixing the initial state and the goal. In `2x2 Locked`, `PPO` achieved a success rate close to 0.5 after 40,000 sample steps and the average plan length is roughly 500. If we combined this high-level policy learning with options training, a single trajectory should have unrolled 500 options with the probability of reaching a goal being around 0.5. More importantly, we observed that the variance in training performance is very large. For example, 95 % confidence interval of the average episode length from the `2x2 Two One-Use Keys` environment ranges up to 500. From this experiment, we can see that learning high-level policy quickly starts to fail when tasks get more complicated. On the other hand, the same planning tasks can be solved optimally using a symbolic planner. Even a simple planner such as `Pyperplan` [40] found the shortest plan with length 4 in 0.0007 seconds for `2x2 Locked`, with length 11 in 0.0028 seconds for `2x2 Two Keys`, with length 4 in 0.0008 seconds for `2x2 One-Use Keys`, and with length 7 in 0.002 seconds for `2x2 Two One-Use Keys`.

6 Conclusion

We have presented a neuro-symbolic sequential decision-making framework that integrates hierarchical reinforcement learning and AI planning following a systematic approach for infusing prior domain knowledge into deep learning agents to solve goal-oriented multi-task reinforcement learning environments. Namely, we formulate AI planning tasks as a symbolic abstraction of underlying RL environments and declare options using the operators in the planning task. In addition, we enforce the behavior of option policies to follow implicit frame constraints in planning operators, which results in improved sample efficiency and reusability of option policies. Empirical evaluation results on various mini-grid environments show that the proposed framework resolves the shortcomings of earlier approaches and benefits of the neuro-symbolic method for solving complex goal-oriented multi-task reinforcement learning environments.

[3] Please see Appendix for the plots summarizing the experiment

References

[1] Artur d'Avila Garcez and Luis C Lamb. Neurosymbolic ai: The 3 rd wave. *Artificial Intelligence Review*, pages 1–20, 2023.

[2] Leslie G Valiant. Three problems in computer science. *Journal of the ACM (JACM)*, 50(1):96–99, 2003.

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

[4] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[5] Anirudh Goyal and Yoshua Bengio. Inductive biases for deep learning of higher-level cognition. *Proceedings of the Royal Society A*, 478(2266), 2022.

[6] Arseny Skryagin, Wolfgang Stammer, Daniel Ochs, Devendra Singh Dhami, and Kristian Kersting. Neural-probabilistic answer set programming. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, pages 463–473, 2022.

[7] Cristina Cornelio, Jan Stuehmer, Shell Xu Hu, and Timothy Hospedales. Learning where and when to reason in neuro-symbolic inference. In *The Eleventh International Conference on Learning Representations*, 2022.

[8] Wen-Chi Yang, Giuseppe Marra, Gavin Rens, and Luc De Raedt. Safe reinforcement learning via probabilistic logic shields. *arXiv preprint arXiv:2303.03226*, 2023.

[9] Daniel Kahneman. *Thinking, fast and slow*. macmillan, 2011.

[10] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1):41–77, 2003.

[11] Thomas Dean and Shieu-Hong Lin. Decomposition techniques for planning in stochastic domains. In *IJCAI*, volume 2, page 3, 1995.

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

[13] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

[14] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proc. AAAI 2017*, pages 1726–1734, 2017.

[15] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proc. ICML 2018*, 2018.

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

[17] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. 73:173–208, 2022.

[18] Brandon Araki, Xiao Li, Kiran Vodrahalli, Jonathan DeCastro, Micah J Fry, and Daniela Rus. The logical options framework. *arXiv preprint arXiv:2102.12571*, 2021.

[19] Floris Den Hengst, Vincent François-Lavet, Mark Hoogendoorn, and Frank van Harmelen. Reinforcement learning with option machines. In *Proc. IJCAI 2022*, pages 2909–2915, 2022.

[20] Leon Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A. McIlraith. Symbolic plans as high-level instructions for reinforcement learning. In *Proc. ICAPS 2020*, pages 540–550, 2020.

[21] Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. Sdrl: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proc. AAAI 2019*, pages 2970–2977, 2019.

[22] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/Farama-Foundation/gym-minigrid, 2018.

[23] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *Proc. ICLR 2019*, 2019.

[24] Andrey Kolobov, Mausam, Daniel S. Weld, and Hector Geffner. Heuristic search for generalized stochastic shortest path MDPs. In *Proc. ICAPS 2011*, pages 130–137, 2011.

[25] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proc. ICML 2007*, pages 1015–1022, 2007.

[26] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Proc. CoRL 2020*, 2020.

[27] Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *Proc. ICML 2021*, pages 9767–9779, 2021.

[28] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

[29] Christer Bäckström and Bernhard Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[30] Don Joven Agravante and Michiaki Tatsubori. Learning neuro-symbolic world models for text-based game playing agents. In *The Third Wordplay: When Language Meets Games Workshop*, 2022.

[31] Kinjal Basu, Keerthiram Murugesan, Mattia Atzeni, Pavan Kapanipathi, Kartik Talamadupula, Tim Klinger, Murray Campbell, Mrinmaya Sachan, and Gopal Gupta. A hybrid neuro-symbolic approach for text-based games using inductive logic programming. In *Combining Learning and Reasoning: Programming Languages, Formalisms, and Representations*, 2022.

[32] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31, 2018.

[33] Manfred Eppe, Phuong DH Nguyen, and Stefan Wermter. From semantics to execution: Integrating action planning with reinforcement learning for robotic causal problem-solving. *Frontiers in Robotics and AI*, 2019.

[34] Tom Silver, Ashay Athalye, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning neuro-symbolic skills for bilevel planning. In *Conference on Robot Learning*, pages 701–714. PMLR, 2023.

[35] Zheng Wen, Doina Precup, Morteza Ibrahimi, Andre Barreto, Benjamin Roy Van, and Satinder Singh. On efficiency in hierarchical reinforcement learning. In *Proc. NeurIPS 2020*, pages 6708–6718, 2020.

[36] Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[37] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[38] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael I. Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3059–3068. PMLR, 2018.

[39] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[40] Yusra Alkhazraji, Matthias Frorath, Markus Grützner, Malte Helmert, Thomas Liebetraut, Robert Mattmüller, Manuela Ortlieb, Jendrik Seipp, Tobias Springenberg, Philip Stahl, and Jan Wülfing. Pyperplan. https://doi.org/10.5281/zenodo.3700819, 2020.

[41] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[42] Hoang Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III. Hierarchical imitation and reinforcement learning. In *International conference on machine learning*, pages 2917–2926. PMLR, 2018.

A Planning Tasks and Plan for MiniGrid RL Environments

This section summarizes the planning tasks for `MiniGrid` domains that we evaluated in the experiment section. The RL environment maintains rooms over an $N \times N$ grid, blue balls, a green goal tile, the agent location and orientation, and doors with states, open, closed, locked, and unlocked. In planning tasks, we abstract away information relevant to each cell in the grid. Namely, the exact location and orientation of the agent, the exact location of the key, blue balls, and a green goal tile are all ignored. In addition, the states of a door is simplified to two states, locked or unlocked.

On resetting the RL environment, we implemented gym environments such that objects that are only visible to RL environments are randomized, as usual in the standard `MiniGrid` gym environment. However, we restricted the information relevant to the planning task remains the same. For example, the agent's initial location will be randomized within a predefined room (the room at the upper left corner), and the goal location will also be randomized within a room at the lower right corner. A key will appear in the same room, and the initial state of the door, whether it is locked or unlocked, will remain the same. This choice doesn't limit algorithms but it simplifies the experiment to start with a single PDDL instance to annotate the environment, although the agent will generate additional PDDL instances when it solves planning tasks with a new initial planning state while selecting options online.

A.1 PDDL domain

PDDL domain file was manually generated by modifying existing similar PDDL domains.

A.1.1 MazeRooms

This domain can be used to all except for the domains with one-use keys.

```
(define (domain MazeRooms)
  (:requirements :strips :typing)
    (:types
        room - object
        key - object
        door - object
    )
    (:predicates
        (at-agent ?r - room)
        (at ?k - key ?r - room)
        (carry ?k - key)
        (empty-hand)
        (KEYMATCH ?k - key ?d - door)
        (LINK ?d - door ?r1 - room ?r2 - room)
        (locked ?d - door)
        (unlocked ?d - door)
        (CONNECTED-ROOMS ?r1 - room ?r2 - room)
    )
    (:action move-room
        :parameters (?d - door ?r1 - room ?r2 - room)
        :precondition (and
            (CONNECTED-ROOMS ?r1 ?r2)
            (at-agent ?r1)
            (LINK ?d ?r1 ?r2)
            (unlocked ?d)
        )
        :effect (and
            (not (at-agent ?r1))
            (at-agent ?r2)
        )
    )
    (:action pickup
        :parameters (?k - key ?r - room)
        :precondition (and
            (at ?k ?r)
            (at-agent ?r)
            (empty-hand)
        )
```

```
        :effect (and
            (not (at ?k ?r))
            (not (empty-hand))
            (carry ?k)
        )
    )
    (:action drop
        :parameters (?k - key ?r - room)
        :precondition (and
            (carry ?k)
            (at-agent ?r)
        )
        :effect (and
            (at ?k ?r)
            (empty-hand)
            (not (carry ?k))
        )
    )
    (:action unlock
        :parameters (?k - key ?d - door ?r1 - room ?r2 - room)
        :precondition (and
            (CONNECTED-ROOMS ?r1 ?r2)
            (at-agent ?r1)
            (LINK ?d ?r1 ?r2)
            (carry ?k)
            (locked ?d)
            (KEYMATCH ?k ?d)
        )
        :effect (and
            (not (locked ?d))
            (unlocked ?d)
        )
    )
)
```

A.1.2 MazeRoomsOneUseKeys

This domain introduces one-use keys.

```
(define (domain MazeRoomsDisposableKeys)
  (:requirements :strips :typing)
    (:types
        room - object
        key - object
        door - object
    )
    (:predicates
        (at-agent ?r - room)
        (at ?k - key ?r - room)
        (carry ?k - key)
        (empty-hand)
        (KEYMATCH ?k - key ?d - door)
        (LINK ?d - door ?r1 - room ?r2 - room)
        (locked ?d - door)
        (unlocked ?d - door)
        (CONNECTED-ROOMS ?r1 - room ?r2 - room)
        (key-unused ?k - key)
    )
    (:action move-room
        :parameters (?d - door ?r1 - room ?r2 - room)
        :precondition (and
            (CONNECTED-ROOMS ?r1 ?r2)
            (at-agent ?r1)
            (LINK ?d ?r1 ?r2)
```

```
            (unlocked ?d)
        )
        :effect (and
            (not (at-agent ?r1))
            (at-agent ?r2)
        )
    )
    (:action pickup
        :parameters (?k - key ?r - room)
        :precondition (and
            (at ?k ?r)
            (at-agent ?r)
            (empty-hand)
        )
        :effect (and
            (not (at ?k ?r))
            (not (empty-hand))
            (carry ?k)
        )
    )
    (:action drop
        :parameters (?k - key ?r - room)
        :precondition (and
            (carry ?k)
            (at-agent ?r)
        )
        :effect (and
            (at ?k ?r)
            (empty-hand)
            (not (carry ?k))
        )
    )
    (:action unlock
        :parameters (?k - key ?d - door ?r1 - room ?r2 - room)
        :precondition (and
            (CONNECTED-ROOMS ?r1 ?r2)
            (at-agent ?r1)
            (LINK ?d ?r1 ?r2)
            (carry ?k)
            (locked ?d)
            (KEYMATCH ?k ?d)
            (key-unused ?k)
        )
        :effect (and
            (not (locked ?d))
            (unlocked ?d)
            (not (key-unused ?k))
        )
    )
)
```

A.2 PDDL instance

All PDDL problem instances were generated by our benchmark script code by processing internal state information available in `MiniGrid` gym environments.

A.2.1 DoorKey PDDL instance

We show PDDL instance, and the shortest plan. Note that option sequence can diverge from the shorest plan when side-effect occurs.

```
(define (problem MazeRooms-8by8-DoorKey)
    (:domain MazeRooms)
    (:objects
        R-0-0 R-1-0 -  room
```

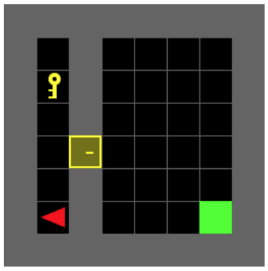


Figure 5: DoorKey

```
        K-yellow-0 - key
        D-yellow-0-0-1-0 - door
    )
    (:init
        (LINK D-yellow-0-0-1-0 R-0-0 R-1-0)
        (LINK D-yellow-0-0-1-0 R-1-0 R-0-0)
        (KEYMATCH K-yellow-0 D-yellow-0-0-1-0)
        (at-agent R-0-0)
        (at K-yellow-0 R-0-0)
        (locked D-yellow-0-0-1-0)
        (empty-hand)
    )
    (:goal (and
        (at-agent R-1-0))
    )
)
```

The shortest plan

```
state:0
(at-agent r-0-0)
(at k-yellow-0 r-0-0)
(locked d-yellow-0-0-1-0)
(empty-hand)

action:0
(pickup k-yellow-0 r-0-0)
  PRE: (at-agent r-0-0)
  PRE: (at k-yellow-0 r-0-0)
  PRE: (empty-hand)
  ADD: (carry k-yellow-0)
  DEL: (at k-yellow-0 r-0-0)
  DEL: (empty-hand)

state:1
(at-agent r-0-0)
(carry k-yellow-0)
(locked d-yellow-0-0-1-0)

action:1
(unlock k-yellow-0 d-yellow-0-0-1-0 r-0-0 r-1-0)
  PRE: (at-agent r-0-0)
  PRE: (carry k-yellow-0)
  PRE: (locked d-yellow-0-0-1-0)
  ADD: (unlocked d-yellow-0-0-1-0)
  DEL: (locked d-yellow-0-0-1-0)

state:2
(at-agent r-0-0)
(carry k-yellow-0)
(unlocked d-yellow-0-0-1-0)
```

```
action:2
(move-room d-yellow-0-0-1-0 r-0-0 r-1-0)
  PRE: (at-agent r-0-0)
  PRE: (unlocked d-yellow-0-0-1-0)
  ADD: (at-agent r-1-0)
  DEL: (at-agent r-0-0)
```

A.2.2 2x2 Locked Door PDDL instance

We show PDDL instance, and the shortest plan. Note that option sequence can diverge from the shorest plan when side-effect occurs.

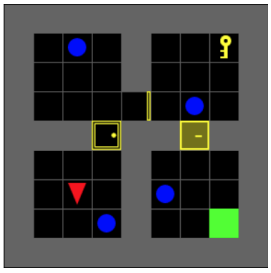


Figure 6: 2x2 Locked Door

```
(define (problem MazeRooms-2by2-LockedSmall)
        (:domain MazeRooms)
        (:objects
            R-0-0 R-0-1 R-1-0 R-1-1 -  room
            K-yellow-0 - key
            D-yellow-0-0-1-0 D-yellow-1-0-1-1 D-yellow-0-0-0-1 - door
        )
        (:init
(CONNECTED-ROOMS R-0-0 R-0-1)
(CONNECTED-ROOMS R-0-0 R-1-0)
(CONNECTED-ROOMS R-0-1 R-0-0)
(CONNECTED-ROOMS R-1-0 R-0-0)
(CONNECTED-ROOMS R-1-0 R-1-1)
(CONNECTED-ROOMS R-1-1 R-1-0)
(LINK D-yellow-0-0-0-1 R-0-0 R-0-1)
(LINK D-yellow-0-0-0-1 R-0-1 R-0-0)
(LINK D-yellow-0-0-1-0 R-0-0 R-1-0)
(LINK D-yellow-0-0-1-0 R-1-0 R-0-0)
(LINK D-yellow-1-0-1-1 R-1-0 R-1-1)
(LINK D-yellow-1-0-1-1 R-1-1 R-1-0)
(KEYMATCH K-yellow-0 D-yellow-0-0-0-1)
(KEYMATCH K-yellow-0 D-yellow-0-0-1-0)
(KEYMATCH K-yellow-0 D-yellow-1-0-1-1)
(at-agent R-0-0)
(at K-yellow-0 R-1-0)
(locked D-yellow-1-0-1-1)
(unlocked D-yellow-0-0-0-1)
(unlocked D-yellow-0-0-1-0)
(empty-hand)
        )
        (:goal
            (and
(at-agent R-1-1)
            )
        )
)
```

The shortest plan

```
state:0
(unlocked d-yellow-0-0-0-1)
(unlocked d-yellow-0-0-1-0)
(locked d-yellow-1-0-1-1)
(empty-hand)
(at-agent r-0-0)
(at k-yellow-0 r-1-0)

action:0
(move-room d-yellow-0-0-1-0 r-0-0 r-1-0)
  PRE: (at-agent r-0-0)
  PRE: (unlocked d-yellow-0-0-1-0)
  ADD: (at-agent r-1-0)
  DEL: (at-agent r-0-0)

state:1
(unlocked d-yellow-0-0-0-1)
(empty-hand)
(at k-yellow-0 r-1-0)
(unlocked d-yellow-0-0-1-0)
(at-agent r-1-0)
(locked d-yellow-1-0-1-1)

action:1
(pickup k-yellow-0 r-1-0)
  PRE: (at k-yellow-0 r-1-0)
  PRE: (empty-hand)
  PRE: (at-agent r-1-0)
  ADD: (carry k-yellow-0)
  DEL: (at k-yellow-0 r-1-0)
  DEL: (empty-hand)

state:2
(unlocked d-yellow-0-0-0-1)
(carry k-yellow-0)
(unlocked d-yellow-0-0-1-0)
(at-agent r-1-0)
(locked d-yellow-1-0-1-1)

action:2
(unlock k-yellow-0 d-yellow-1-0-1-1 r-1-0 r-1-1)
  PRE: (carry k-yellow-0)
  PRE: (locked d-yellow-1-0-1-1)
  PRE: (at-agent r-1-0)
  ADD: (unlocked d-yellow-1-0-1-1)
  DEL: (locked d-yellow-1-0-1-1)

state:3
(unlocked d-yellow-0-0-0-1)
(carry k-yellow-0)
(unlocked d-yellow-1-0-1-1)
(unlocked d-yellow-0-0-1-0)
(at-agent r-1-0)

action:3
(move-room d-yellow-1-0-1-1 r-1-0 r-1-1)
  PRE: (unlocked d-yellow-1-0-1-1)
  PRE: (at-agent r-1-0)
  ADD: (at-agent r-1-1)
  DEL: (at-agent r-1-0)
```

A.2.3 2x2 Two Keys PDDL instance

We show PDDL instance, and the shortest plan. Note that option sequence can diverge from the shorest plan when side-effect occurs.

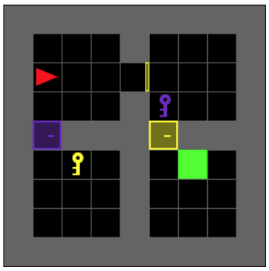


Figure 7: 2x2 Two Keys

```
(define (problem MazeRooms-2by2-TwoKeysSmall)
        (:domain MazeRooms)
        (:objects
            R-0-0 R-0-1 R-1-0 R-1-1 -  room
            K-purple-0 K-yellow-1 - key
            D-yellow-1-0-1-1 D-purple-0-0-0-1 D-yellow-0-0-1-0 - door
        )
        (:init
(CONNECTED-ROOMS R-0-0 R-0-1)
(CONNECTED-ROOMS R-0-0 R-1-0)
(CONNECTED-ROOMS R-0-1 R-0-0)
(CONNECTED-ROOMS R-1-0 R-0-0)
(CONNECTED-ROOMS R-1-0 R-1-1)
(CONNECTED-ROOMS R-1-1 R-1-0)
(LINK D-purple-0-0-0-1 R-0-0 R-0-1)
(LINK D-purple-0-0-0-1 R-0-1 R-0-0)
(LINK D-yellow-0-0-1-0 R-0-0 R-1-0)
(LINK D-yellow-0-0-1-0 R-1-0 R-0-0)
(LINK D-yellow-1-0-1-1 R-1-0 R-1-1)
(LINK D-yellow-1-0-1-1 R-1-1 R-1-0)
(KEYMATCH K-purple-0 D-purple-0-0-0-1)
(KEYMATCH K-yellow-1 D-yellow-0-0-1-0)
(KEYMATCH K-yellow-1 D-yellow-1-0-1-1)
(at-agent R-0-0)
(at K-purple-0 R-1-0)
(at K-yellow-1 R-0-1)
(locked D-purple-0-0-0-1)
(locked D-yellow-1-0-1-1)
(unlocked D-yellow-0-0-1-0)
(empty-hand)
        )
        (:goal
            (and
(at-agent R-1-1)
            )
        )
)
```

The shortest plan

```
state:0
(at k-purple-0 r-1-0)
(locked d-purple-0-0-0-1)
(unlocked d-yellow-0-0-1-0)
(locked d-yellow-1-0-1-1)
(at k-yellow-1 r-0-1)
```

```
(empty-hand)
(at-agent r-0-0)

action:0
(move-room d-yellow-0-0-1-0 r-0-0 r-1-0)
  PRE: (at-agent r-0-0)
  PRE: (unlocked d-yellow-0-0-1-0)
  ADD: (at-agent r-1-0)
  DEL: (at-agent r-0-0)

state:1
(at k-purple-0 r-1-0)
(at k-yellow-1 r-0-1)
(empty-hand)
(locked d-purple-0-0-0-1)
(unlocked d-yellow-0-0-1-0)
(at-agent r-1-0)
(locked d-yellow-1-0-1-1)

action:1
(pickup k-purple-0 r-1-0)
  PRE: (at k-purple-0 r-1-0)
  PRE: (empty-hand)
  PRE: (at-agent r-1-0)
  ADD: (carry k-purple-0)
  DEL: (at k-purple-0 r-1-0)
  DEL: (empty-hand)

state:2
(carry k-purple-0)
(at k-yellow-1 r-0-1)
(locked d-purple-0-0-0-1)
(unlocked d-yellow-0-0-1-0)
(at-agent r-1-0)
(locked d-yellow-1-0-1-1)

action:2
(move-room d-yellow-0-0-1-0 r-1-0 r-0-0)
  PRE: (unlocked d-yellow-0-0-1-0)
  PRE: (at-agent r-1-0)
  ADD: (at-agent r-0-0)
  DEL: (at-agent r-1-0)

state:3
(carry k-purple-0)
(at k-yellow-1 r-0-1)
(at-agent r-0-0)
(locked d-purple-0-0-0-1)
(unlocked d-yellow-0-0-1-0)
(locked d-yellow-1-0-1-1)

action:3
(unlock k-purple-0 d-purple-0-0-0-1 r-0-0 r-0-1)
  PRE: (at-agent r-0-0)
  PRE: (carry k-purple-0)
  PRE: (locked d-purple-0-0-0-1)
  ADD: (unlocked d-purple-0-0-0-1)
  DEL: (locked d-purple-0-0-0-1)

state:4
(carry k-purple-0)
(at k-yellow-1 r-0-1)
(unlocked d-purple-0-0-0-1)
(at-agent r-0-0)
(unlocked d-yellow-0-0-1-0)
```

```
(locked d-yellow-1-0-1-1)

action:4
(drop k-purple-0 r-0-0)
  PRE: (at-agent r-0-0)
  PRE: (carry k-purple-0)
  ADD: (at k-purple-0 r-0-0)
  ADD: (empty-hand)
  DEL: (carry k-purple-0)

state:5
(at k-purple-0 r-0-0)
(at k-yellow-1 r-0-1)
(unlocked d-purple-0-0-0-1)
(empty-hand)
(at-agent r-0-0)
(unlocked d-yellow-0-0-1-0)
(locked d-yellow-1-0-1-1)

action:5
(move-room d-purple-0-0-0-1 r-0-0 r-0-1)
  PRE: (at-agent r-0-0)
  PRE: (unlocked d-purple-0-0-0-1)
  ADD: (at-agent r-0-1)
  DEL: (at-agent r-0-0)

state:6
(at k-purple-0 r-0-0)
(at k-yellow-1 r-0-1)
(empty-hand)
(unlocked d-purple-0-0-0-1)
(at-agent r-0-1)
(locked d-yellow-1-0-1-1)
(unlocked d-yellow-0-0-1-0)

action:6
(pickup k-yellow-1 r-0-1)
  PRE: (at-agent r-0-1)
  PRE: (at k-yellow-1 r-0-1)
  PRE: (empty-hand)
  ADD: (carry k-yellow-1)
  DEL: (at k-yellow-1 r-0-1)
  DEL: (empty-hand)

state:7
(at k-purple-0 r-0-0)
(carry k-yellow-1)
(unlocked d-purple-0-0-0-1)
(at-agent r-0-1)
(locked d-yellow-1-0-1-1)
(unlocked d-yellow-0-0-1-0)

action:7
(move-room d-purple-0-0-0-1 r-0-1 r-0-0)
  PRE: (at-agent r-0-1)
  PRE: (unlocked d-purple-0-0-0-1)
  ADD: (at-agent r-0-0)
  DEL: (at-agent r-0-1)

state:8
(at k-purple-0 r-0-0)
(carry k-yellow-1)
(unlocked d-purple-0-0-0-1)
(at-agent r-0-0)
(locked d-yellow-1-0-1-1)
```

```
(unlocked d-yellow-0-0-1-0)

action:8
(move-room d-yellow-0-0-1-0 r-0-0 r-1-0)
  PRE: (at-agent r-0-0)
  PRE: (unlocked d-yellow-0-0-1-0)
  ADD: (at-agent r-1-0)
  DEL: (at-agent r-0-0)

state:9
(at-agent r-1-0)
(at k-purple-0 r-0-0)
(carry k-yellow-1)
(unlocked d-purple-0-0-0-1)
(locked d-yellow-1-0-1-1)
(unlocked d-yellow-0-0-1-0)

action:9
(unlock k-yellow-1 d-yellow-1-0-1-1 r-1-0 r-1-1)
  PRE: (locked d-yellow-1-0-1-1)
  PRE: (carry k-yellow-1)
  PRE: (at-agent r-1-0)
  ADD: (unlocked d-yellow-1-0-1-1)
  DEL: (locked d-yellow-1-0-1-1)

state:10
(at k-purple-0 r-0-0)
(carry k-yellow-1)
(unlocked d-purple-0-0-0-1)
(unlocked d-yellow-1-0-1-1)
(at-agent r-1-0)
(unlocked d-yellow-0-0-1-0)

action:10
(move-room d-yellow-1-0-1-1 r-1-0 r-1-1)
  PRE: (unlocked d-yellow-1-0-1-1)
  PRE: (at-agent r-1-0)
  ADD: (at-agent r-1-1)
  DEL: (at-agent r-1-0)
```

A.2.4 2x2 One-Use Key PDDL instance

We show PDDL instance, and the shortest plan. Note that option sequence can diverge from the shortest plan when side-effect occurs.

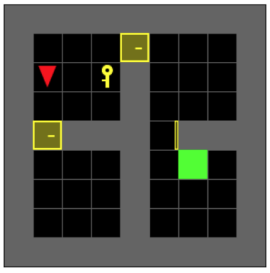


Figure 8: 2x2 One-Use Key

```
(define (problem MazeRooms-2by2-OneDisposableKey)
        (:domain MazeRoomsDisposableKeys)
        (:objects
            R-0-0 R-0-1 R-1-0 R-1-1 -  room
            K-yellow-0 - key
```

```
            D-yellow-1-0-1-1 D-yellow-0-0-0-1 D-yellow-0-0-1-0 - door
        )
        (:init
(CONNECTED-ROOMS R-0-0 R-0-1)
(CONNECTED-ROOMS R-0-0 R-1-0)
(CONNECTED-ROOMS R-0-1 R-0-0)
(CONNECTED-ROOMS R-1-0 R-0-0)
(CONNECTED-ROOMS R-1-0 R-1-1)
(CONNECTED-ROOMS R-1-1 R-1-0)
(LINK D-yellow-0-0-0-1 R-0-0 R-0-1)
(LINK D-yellow-0-0-0-1 R-0-1 R-0-0)
(LINK D-yellow-0-0-1-0 R-0-0 R-1-0)
(LINK D-yellow-0-0-1-0 R-1-0 R-0-0)
(LINK D-yellow-1-0-1-1 R-1-0 R-1-1)
(LINK D-yellow-1-0-1-1 R-1-1 R-1-0)
(KEYMATCH K-yellow-0 D-yellow-0-0-0-1)
(KEYMATCH K-yellow-0 D-yellow-0-0-1-0)
(KEYMATCH K-yellow-0 D-yellow-1-0-1-1)
(at-agent R-0-0)
(at K-yellow-0 R-0-0)
(locked D-yellow-0-0-0-1)
(locked D-yellow-0-0-1-0)
(unlocked D-yellow-1-0-1-1)
(empty-hand)
(key-unused K-yellow-0)
        )
        (:goal
            (and
(at-agent R-1-1)
            )
        )
)
```

The shortest plan

```
state:0
(locked d-yellow-0-0-0-1)
(key-unused k-yellow-0)
(at k-yellow-0 r-0-0)
(unlocked d-yellow-1-0-1-1)
(at-agent r-0-0)
(locked d-yellow-0-0-1-0)
(empty-hand)

action:0
(pickup k-yellow-0 r-0-0)
  PRE: (at-agent r-0-0)
  PRE: (at k-yellow-0 r-0-0)
  PRE: (empty-hand)
  ADD: (carry k-yellow-0)
  DEL: (at k-yellow-0 r-0-0)
  DEL: (empty-hand)

state:1
(unlocked d-yellow-1-0-1-1)
(locked d-yellow-0-0-0-1)
(at-agent r-0-0)
(key-unused k-yellow-0)
(locked d-yellow-0-0-1-0)
(carry k-yellow-0)

action:1
(unlock k-yellow-0 d-yellow-0-0-1-0 r-0-0 r-1-0)
  PRE: (at-agent r-0-0)
  PRE: (locked d-yellow-0-0-1-0)
```

```
  PRE: (carry k-yellow-0)
  PRE: (key-unused k-yellow-0)
  ADD: (unlocked d-yellow-0-0-1-0)
  DEL: (locked d-yellow-0-0-1-0)
  DEL: (key-unused k-yellow-0)

state:2
(unlocked d-yellow-1-0-1-1)
(locked d-yellow-0-0-0-1)
(at-agent r-0-0)
(unlocked d-yellow-0-0-1-0)
(carry k-yellow-0)

action:2
(move-room d-yellow-0-0-1-0 r-0-0 r-1-0)
  PRE: (at-agent r-0-0)
  PRE: (unlocked d-yellow-0-0-1-0)
  ADD: (at-agent r-1-0)
  DEL: (at-agent r-0-0)

state:3
(unlocked d-yellow-1-0-1-1)
(locked d-yellow-0-0-0-1)
(at-agent r-1-0)
(unlocked d-yellow-0-0-1-0)
(carry k-yellow-0)

action:3
(move-room d-yellow-1-0-1-1 r-1-0 r-1-1)
  PRE: (unlocked d-yellow-1-0-1-1)
  PRE: (at-agent r-1-0)
  ADD: (at-agent r-1-1)
  DEL: (at-agent r-1-0)
```

A.2.5 2x2 Two One-Use Keys PDDL instance

We show PDDL instance, and the shortest plan. Note that option sequence can diverge from the shortest plan when side-effect occurs.

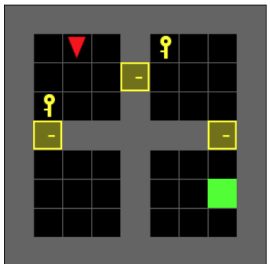


Figure 9: 2x2 Two One-Use Keys

```
(define (problem MazeRooms-2by2-TwoDisposableKeys)
        (:domain MazeRoomsDisposableKeys)
        (:objects
            R-0-0 R-0-1 R-1-0 R-1-1 -  room
            K-yellow-0 K-yellow-1 - key
            D-yellow-0-0-1-0 D-yellow-0-0-0-1 D-yellow-1-0-1-1 - door
        )
        (:init
(CONNECTED-ROOMS R-0-0 R-0-1)
(CONNECTED-ROOMS R-0-0 R-1-0)
(CONNECTED-ROOMS R-0-1 R-0-0)
(CONNECTED-ROOMS R-1-0 R-0-0)
```

```
(CONNECTED-ROOMS R-1-0 R-1-1)
(CONNECTED-ROOMS R-1-1 R-1-0)
(LINK D-yellow-0-0-0-1 R-0-0 R-0-1)
(LINK D-yellow-0-0-0-1 R-0-1 R-0-0)
(LINK D-yellow-0-0-1-0 R-0-0 R-1-0)
(LINK D-yellow-0-0-1-0 R-1-0 R-0-0)
(LINK D-yellow-1-0-1-1 R-1-0 R-1-1)
(LINK D-yellow-1-0-1-1 R-1-1 R-1-0)
(KEYMATCH K-yellow-0 D-yellow-0-0-0-1)
(KEYMATCH K-yellow-0 D-yellow-0-0-1-0)
(KEYMATCH K-yellow-0 D-yellow-1-0-1-1)
(KEYMATCH K-yellow-1 D-yellow-0-0-0-1)
(KEYMATCH K-yellow-1 D-yellow-0-0-1-0)
(KEYMATCH K-yellow-1 D-yellow-1-0-1-1)
(at-agent R-0-0)
(at K-yellow-0 R-0-0)
(at K-yellow-1 R-1-0)
(locked D-yellow-0-0-0-1)
(locked D-yellow-0-0-1-0)
(locked D-yellow-1-0-1-1)
(empty-hand)
(key-unused K-yellow-0)
(key-unused K-yellow-1)
        )
        (:goal
            (and
(at-agent R-1-1)
            )
        )
)
```

The shortest plan

```
state:0
(locked d-yellow-0-0-0-1)
(locked d-yellow-1-0-1-1)
(key-unused k-yellow-0)
(at k-yellow-0 r-0-0)
(at k-yellow-1 r-1-0)
(at-agent r-0-0)
(key-unused k-yellow-1)
(locked d-yellow-0-0-1-0)
(empty-hand)

action:0
(pickup k-yellow-0 r-0-0)
  PRE: (at-agent r-0-0)
  PRE: (at k-yellow-0 r-0-0)
  PRE: (empty-hand)
  ADD: (carry k-yellow-0)
  DEL: (at k-yellow-0 r-0-0)
  DEL: (empty-hand)

state:1
(locked d-yellow-0-0-0-1)
(locked d-yellow-1-0-1-1)
(at-agent r-0-0)
(key-unused k-yellow-0)
(key-unused k-yellow-1)
(at k-yellow-1 r-1-0)
(locked d-yellow-0-0-1-0)
(carry k-yellow-0)

action:1
(unlock k-yellow-0 d-yellow-0-0-1-0 r-0-0 r-1-0)
```

```
  PRE: (at-agent r-0-0)
  PRE: (locked d-yellow-0-0-1-0)
  PRE: (carry k-yellow-0)
  PRE: (key-unused k-yellow-0)
  ADD: (unlocked d-yellow-0-0-1-0)
  DEL: (locked d-yellow-0-0-1-0)
  DEL: (key-unused k-yellow-0)

state:2
(locked d-yellow-0-0-0-1)
(locked d-yellow-1-0-1-1)
(at-agent r-0-0)
(key-unused k-yellow-1)
(unlocked d-yellow-0-0-1-0)
(at k-yellow-1 r-1-0)
(carry k-yellow-0)

action:2
(move-room d-yellow-0-0-1-0 r-0-0 r-1-0)
  PRE: (at-agent r-0-0)
  PRE: (unlocked d-yellow-0-0-1-0)
  ADD: (at-agent r-1-0)
  DEL: (at-agent r-0-0)

state:3
(locked d-yellow-0-0-0-1)
(locked d-yellow-1-0-1-1)
(key-unused k-yellow-1)
(at-agent r-1-0)
(unlocked d-yellow-0-0-1-0)
(at k-yellow-1 r-1-0)
(carry k-yellow-0)

action:3
(drop k-yellow-0 r-1-0)
  PRE: (carry k-yellow-0)
  PRE: (at-agent r-1-0)
  ADD: (at k-yellow-0 r-1-0)
  ADD: (empty-hand)
  DEL: (carry k-yellow-0)

state:4
(locked d-yellow-0-0-0-1)
(locked d-yellow-1-0-1-1)
(key-unused k-yellow-1)
(at k-yellow-0 r-1-0)
(at-agent r-1-0)
(unlocked d-yellow-0-0-1-0)
(at k-yellow-1 r-1-0)
(empty-hand)

action:4
(pickup k-yellow-1 r-1-0)
  PRE: (at k-yellow-1 r-1-0)
  PRE: (at-agent r-1-0)
  PRE: (empty-hand)
  ADD: (carry k-yellow-1)
  DEL: (at k-yellow-1 r-1-0)
  DEL: (empty-hand)

state:5
(locked d-yellow-0-0-0-1)
(locked d-yellow-1-0-1-1)
(key-unused k-yellow-1)
(at k-yellow-0 r-1-0)
```

```
(at-agent r-1-0)
(unlocked d-yellow-0-0-1-0)
(carry k-yellow-1)

action:5
(unlock k-yellow-1 d-yellow-1-0-1-1 r-1-0 r-1-1)
  PRE: (carry k-yellow-1)
  PRE: (locked d-yellow-1-0-1-1)
  PRE: (key-unused k-yellow-1)
  PRE: (at-agent r-1-0)
  ADD: (unlocked d-yellow-1-0-1-1)
  DEL: (locked d-yellow-1-0-1-1)
  DEL: (key-unused k-yellow-1)

state:6
(unlocked d-yellow-1-0-1-1)
(locked d-yellow-0-0-0-1)
(at k-yellow-0 r-1-0)
(at-agent r-1-0)
(unlocked d-yellow-0-0-1-0)
(carry k-yellow-1)

action:6
(move-room d-yellow-1-0-1-1 r-1-0 r-1-1)
  PRE: (unlocked d-yellow-1-0-1-1)
  PRE: (at-agent r-1-0)
  ADD: (at-agent r-1-1)
  DEL: (at-agent r-1-0)
```

B Hierarchical Reward Machines for MiniGrid Experiments

Hierarchical Reward Machine (HRM) needs a Finite State Machine (FSM) that describes the transitions between symbolic states and events that trigger the transitions. It can either be written directly or translated from Linear Temporal Logic (LTL) expressions. In this paper, we defined FSMs for MiniGrid environments following the structures that commonly appear in papers based on reward machines. Note that FSMs in HRL algorithms based on LTL/RM encode knowledge about the solution to the problem. FSMs are defined per instance basis, or a human expert must know a partial solution that is general enough so that it can be applied to multiple instances. As the problem domain gets more complicated, this manual task is not at all trivial. In this paper, we chose hierarchical reward machines (HRM) [17] as a baseline HRL algorithm since it is very difficult to find a reliable implementation that integrates deep RL agents. While extending the baseline for solving MiniGrid environments, we defined FSMs similar to the ones in the baseline method.

B.1 MiniGrid - DoorKey

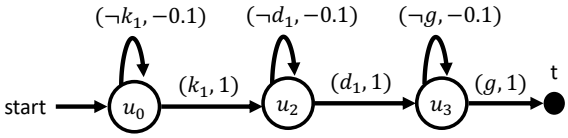


Figure 10: FSM of HRM - DoorKey

Nodes u_0, u_1, and u_2 are FSM states. Upon resetting the RL environment, FSM enters the first node u_0, and events defined over the edges trigger the state transitions. This reward structure can be used for defining rewards for the RL environment in a reward machine, or one could define options over FSMs that encapsulates temporarily extended actions. The events are defined as follows: k_1 entails true if the agent picked up the key at the room, d_1 entails true if the agent was able to unlock the door connecting two rooms, g entails true if the agent arrived at the goal room. Finally, the FSM terminates when the agent arrives at the goal tile. The value next to the event is the reward that the agent receives. For example, when the agent was in state u_0 and did not pick up the key, then the reward is -0.1. On the other hand, if the agent picked up the key, then the state transition occurs, and the agent receives a reward of 1.

B.2 MiniGrid - 2x2 Locked Door

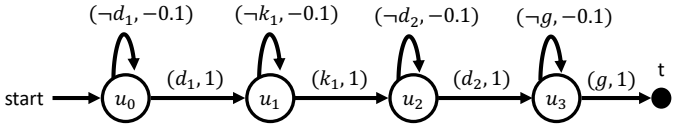


Figure 11: FSM of HRM - 2x2 Locked Door

In this problem domain, the agent must use a key to unlock the goal room. Therefore, FSM encodes such knowledge in the FSM; from state u_1, the agent can transit to the next state if the agent picked up the key in the room at the upper right corner (k_1). As we can see, as the solution to the problem becomes more complex, the FSM has to incorporate such knowledge in more complex diagrams, one per domain. It is worth noting that in order to incorporate the knowledge of solutions in the FSM, one needs first to obtain such knowledge. While for small problems humans can easily spot what a solution is, as problems become more complex, it becomes harder.

B.3 MiniGrid - 2x2 Two Keys

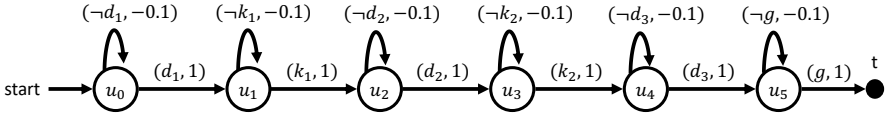


Figure 12: FSM of HRM - 2x2 Two Keys

In this problem domain, the agent must use two keys by moving rooms back and forth. It first picks up the purple key to unlock a room at the lower left corner to pick up the yellow key. Then, the agent must navigate to the goal room to unlock the goal room. The encode this behavior using events, we need to define a longer chain as shown in the above figure.

C Implementation Notes on MiniGrid Experiments

In this section, we provide implementation details for `HplanPPO`, and `HRM`, and hyperparameter tunings. For additional details, please refer to the python script code available in the code supplementary material.

C.1 Feature Extractors

For the problem domains generated by `MiniGrid` environment, we modified Convolutional Neural Network (CNN) based architecture presented in `BabyAI` RL environment [23]. The main differences between `BabyAI` and our `MiniGrid`-based gym environments are: (1) our experiments are fully-observable, (2) there's no natural language goal description available in our experiments.

C.1.1 CNN Feature Extractors for 4 Rooms Environments

The CNN feature extractors first process three-channel input grid into the embedding layer since the value at each grid encodes symbolic state information in integers. Next, we pass 3-layer CNN, and finally we added the last linear layer to the output the feature vector of size 128.

```
class BabyAIFullyObsCNN(BaseFeaturesExtractor):
    def __init__(
        self,
        observation_space: gym.Space,
        features_dim: int = 128,
    ):
        super().__init__(observation_space, features_dim)
        self.max_value = 147
        self.embedding = nn.Embedding(3 * self.max_value, features_dim)
        self.cnn = nn.Sequential(
            nn.Conv2d(in_channels=features_dim, out_channels=features_dim,
            kernel_size=(3, 3), stride=(2, 2), padding=1),
```

```
            nn.BatchNorm2d(features_dim),
            nn.ReLU(),
            nn.Conv2d(in_channels=features_dim, out_channels=features_dim,
            kernel_size=(3, 3), stride=(2, 2), padding=1),
            nn.BatchNorm2d(features_dim),
            nn.ReLU(),
            nn.Conv2d(in_channels=features_dim, out_channels=features_dim,
            kernel_size=(3, 3), stride=(2, 2), padding=1),
            nn.BatchNorm2d(features_dim),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0),
            nn.Flatten()
        )
        self.linear = nn.Sequential(
            nn.Linear(n_flatten, features_dim),
            nn.ReLU()
        )
        self.apply(initialize_parameters)

    def forward(self, observations: th.Tensor):
        offsets = th.Tensor([0, self.max_value, 2 * self.max_value])
        x = (observations + offsets[None, :, None, None]).long()
        x =  self.embedding(x).sum(1).permute(0, 3, 1, 2)
        x = self.cnn(x)
        x = self.linear(x)
        return x
```

C.1.2 CNN Feature Extractors for Door Key environment

The architecture remains the same as above except for the CNN only has two layers when the input dimension becomes smaller. In addition to processing the feature values in the grid, the following code snippet also shows the option labels will also be concatenated with the feature vector after passing an embedding layer and one additional linear layer. These option label features are necessary for implementing DDQN-based algorithms.

```
class BabyAIFullyObsSmallCNNDict(BaseFeaturesExtractor):
    def __init__(
            self,
            observation_space: gym.Space,
            features_dim: int = 128,
    ):
        super().__init__(observation_space, features_dim)
        image_observation_space = observation_space.spaces['image']

        self.max_value = 147
        self.embedding = nn.Embedding(3 * self.max_value, features_dim)
        self.cnn = nn.Sequential(
            nn.Conv2d(in_channels=features_dim, out_channels=features_dim,
            kernel_size=(3, 3), stride=(2, 2), padding=1),
            nn.BatchNorm2d(features_dim),
            nn.ReLU(),
            nn.Conv2d(in_channels=features_dim, out_channels=features_dim,
            kernel_size=(3, 3), stride=(2, 2), padding=1),
            nn.BatchNorm2d(features_dim),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0),
            nn.Flatten()
        )
        self.linear = nn.Sequential(
            nn.Linear(n_flatten, features_dim),
            nn.ReLU()
        )
        label_observation_space = observation_space.spaces['label']
        self.label_embedding = nn.Linear(label_observation_space.n, features_dim)
        self.linear2 = nn.Sequential(
            nn.Linear(features_dim * 2, features_dim),
```

```
            nn.ReLU()
        )
        self.apply(initialize_parameters)  # Initialize parameters correctly

    def forward(self, observations: th.Tensor) -> th.Tensor:
        x = observations['image']
        offsets = th.Tensor([0, self.max_value, 2 * self.max_value]).to(x.device)
        x = (x + offsets[None, :, None, None]).long()
        x = self.embedding(x).sum(1).permute(0, 3, 1, 2)
        x = self.cnn(x)
        x = self.linear(x)
        y = observations['label']
        y = th.squeeze(y)
        y = self.label_embedding(y)
        if y.ndim == 1:
            y = y.reshape((1, -1))
        z = th.cat((x,y), dim=1)
        z = self.linear2(z)
        return z
```

C.2 PPO Hyperparameters

Door Key

```
learning_rate=1.0206760062018722e-5,
n_steps=2048,
batch_size=128,
n_epochs=50,
gamma=0.9833047938219175,
gae_lambda=0.95,
ent_coef=0.004845529343815526,
vf_coef=0.6628235140402716,
max_grad_norm=9.807565080094877
```

2x2 Locked

```
learning_rate=8.63971021360162e-05,
n_steps=2048,
batch_size=128,
n_epochs=30,
gamma=0.9747508289308954,
gae_lambda=0.95,
ent_coef=0.006155835398338309,
vf_coef=0.6788881163701028,
max_grad_norm=3.959536653406463
```

C.3 PPO-ICM Hyperparameters

Door Key

```
gamma=0.9643644696068203,
lr=2.137424219745892e-05,
vf_loss_coeff=0.21801595987072098
entropy_coeff=0.0003088560129766813
sgd_minibatch_size=32,
num_sgd_iter=143,
clip_param=0.2,
vf_clip_param=10.0,
grad_clip=10.0,
train_batch_size=2048,
kl_coeff=0.0
```

2x2 Locked

```
gamma=0.9130324059140903,
```

```
lr=0.00033384977161037265,
vf_loss_coeff=0.2612257420727645,
entropy_coeff=0.00044831677402583,
sgd_minibatch_size=512,
num_sgd_iter=15
clip_param=0.2,
vf_clip_param=10.0,
grad_clip=10.0,
train_batch_size=2048,
kl_coeff=0.0
```

C.4 DDQN Hyperparameters

Door Key

```
learning_rate=0.000681590954892754,
buffer_size=409600,
learning_starts=7876,
batch_size=128,
tau=1.0,
gamma=0.9514622035384503,
train_freq=(2862, 'step'),
gradient_steps=142,
target_update_interval=8230,
exploration_fraction=0.3312761633788077,
exploration_initial_eps=1.0,
exploration_final_eps=0.189177300078208,
max_grad_norm=10
```

2x2 Locked

```
learning_rate=3.948924019726062e-05,
buffer_size=409600,
learning_starts=40960,
batch_size=128,
tau=1.0,
gamma=0.96185178826382,
train_freq=(2048, 'step'),
gradient_steps=150,
target_update_interval=8192,
exploration_fraction=0.3017598585099074,
exploration_initial_eps=1.0,
exploration_final_eps=0.17251992085542112,
max_grad_norm=10
```

C.5 Rainbow Hyperparameters

Door Key

```
gamma=0.9849743584371063,
lr=1.1473370153687392e-05,
train_batch_size=512,
n_step=5,
sigma0=0.4434930536144698,
num_atoms=16,
target_network_update_freq=1979
arget_network_update_freq=41721
dueling=True,
hiddens=[256],
double_q=True,
```

2x2 Locked

```
gamma=0.9211595438968194,
```

```
lr=0.00012196986271591993,
rain_batch_size=256,
n_step=5,
sigma0=0.35020035545149664,
num_atoms=9,
target_network_update_freq=41721
dueling=True,
hiddens=[256],
double_q=True,
```

C.6 HplanPPO Hyperparameters

Door Key

```
option_policy_learning_rate=5.338528208876637e-05,
max_episode_len=2048,
option_termination_reward=1.0,
option_step_cost=0.0,
option_penalty_cost=0.005253418767263643 if args.frame_ir == 1 else 0.0,
option_terminal_cost=0.6491480172177918 if args.term_ir == 1 else 0.0,
n_steps=2048,
batch_size=32,
n_epochs=20,
gamma=0.9280090211332641,
gae_lambda=0.95,
ent_coef=0.005526152849292115,
vf_coef=0.722641202436966,
max_grad_norm=4.450973669431999
```

2x2 Locked Door

```
option_policy_learning_rate=2.483713270605326e-05,
max_episode_len=2048,
option_termination_reward=1.0,
option_step_cost=0.0,
option_penalty_cost=0.005327486099747157 if args.frame_ir == 1 else 0.0,
option_terminal_cost=0.011665771575084822 if args.term_ir == 1 else 0.0,
n_steps=2048,
batch_size=128,
n_epochs=10,
gamma=0.9889237786929247,
gae_lambda=0.95,
ent_coef=0.005571793418295724,
vf_coef=0.7347714098789104,
max_grad_norm=10,
```

2x2 Two Keys

```
option_policy_learning_rate=1.3424328073531024e-05,
option_termination_reward=1.0,
option_step_cost=0.0,
option_penalty_cost=0.00911114780697196 if args.frame_ir == 1 else 0.0,
option_terminal_cost=0.016975392630852796 if args.term_ir == 1 else 0.0,
n_steps=2048,
batch_size=64,
n_epochs=40,
gamma=0.9363696925451804,
gae_lambda=0.95,
ent_coef=0.0008151389080119535,
vf_coef=0.3777478706019491,
max_grad_norm=10,
```

2x2 One-Use Key

```
option_policy_learning_rate=2.483713270605326e-05,
```

```
max_episode_len=2048,
option_termination_reward=1.0,
option_step_cost=0.0,
option_penalty_cost=0.008634793137792446 if args.frame_ir == 1 else 0.0,
option_terminal_cost=0.010137896834674922 if args.term_ir == 1 else 0.0,
n_steps=2048,
batch_size=64,
n_epochs=20,
gamma=0.980977021289778,
gae_lambda=0.95,
ent_coef=0.006943688563757862,
vf_coef=0.5331594110889363,
max_grad_norm=10,
```

2x2 Two One-Use Keys

```
option_policy_learning_rate=1.2069618067476194e-05,
option_termination_reward=1.0,
option_step_cost=0.0,
option_penalty_cost=0.005155317730172398 if args.frame_ir == 1 else 0.0,
option_terminal_cost=0.00896013032359551 if args.term_ir == 1 else 0.0,
n_steps-2048,
batch_size=128,
n_epochs=60,
gamma=0.9762354297259361,
gae_lambda=0.95,
ent_coef=0.00012158162774537214,
vf_coef=0.16581733465315923,
max_grad_norm=10,
```

D Supplemental Python Scripts

In this section, we explain the supplemental python scripts used in the experiment.

D.1 HRL with AI Planning Model

We developed 3 python packages for implementing the proposed method. We recommend installing **parl annotations**, **parl minigird**, and **parl agents** in a python virtual environment with version 3.7 or greater. We also included other necessary open-source libraries together to avoid errors due to version mismatches.

- **parl annotations** package defines base classes for classes that define options from planning tasks. It also defines basic interface to use AI planner inside the agent. The code is available at https://github.com/IBM/parl_annotations.
- **parl minigrid** extends `MiniGird` environment with planning models. We also developed additional models used in the experiments. Under `annotations` folder we collected all planning tasks as well as FSMs for reward machines. The code is available at https://github.com/IBM/parl_minigrid.
- **parl agent** implements HRL with AI Planning model. For DRL agents, we used `stable-baselines3` and HRL agents also extend the base class of `stable-baselines3`. The code is available at https://github.com/IBM/parl_agents.
- for the usage, we provide test-scripts under the parl-agent package.

D.2 Baseline Algorithms

We also add open-source baseline algorithm codes. **rllib** and **reward-machines** contain the code used during the experiment.

D.3 Code Examples from Earlier Approaches

In this section, we show code examples from earlier approaches that hard code abstract plan inside python scripts. The hard-coded approaches cannot handle side effects or multi-task environments that we used during the experiments. Furthermore, algorithms are combined with the experiment environment so it is very difficult to use those algorithms as baselines we modify algorithms per environment.

D.3.1 Taskable RL

This is an example from Taskable RL [20] for solving the office world problem, a grid navigation domain, with tabular Q-learning as an RL agent. We can see that the symbolic states are hard coded with integers and high-level actions are defined manually inside the algorithm.

```
states = []
for s in [(0, 0), (11, 8), (2, 6), (9, 2), (0, 4), (11, 4), (5, 8), (6, 0),
          (3, 4), (8, 4)]:
    states.append(OfficeState(s[0], s[1], []))

rng = Random(2019)
env = Office(rng)

tasks = [([0], 200000), ([1], 200000), ([2], 200000), ([3], 200000),
         ([4], 200000), ([5], 200000), ([6], 200000), ([7], 500000),
         ([8], 500000), ([7, 8], 1000000), ([0, 1, 2, 3], 1000000),
         ([0, 1, 2, 3, 7, 8], 3000000), ([0, 1, 2, 3, 4, 5, 6, 7, 8], 5000000)]
tasks = tasks[START_TASK:END_TASK+1]
```

(a) Symbolic State Representation in Taskble RL

```
a0 = HLAction([], [], [0], 0)
a1 = HLAction([], [], [1], 1)
a2 = HLAction([], [], [2], 2)
a3 = HLAction([], [], [3], 3)
a4 = HLAction([], [], [4], 4)
a5 = HLAction([], [], [5], 5)
a6 = HLAction([], [], [6], 6)
a7 = HLAction([4], [4], [6, 7], 6)
a8 = HLAction([5], [5], [6, 8], 6)

plan0 = [a0]
plan1 = [a1]
plan2 = [a2]
plan3 = [a3]
plan4 = [a4]
plan5 = [a5]
plan6 = [a6]
plan7 = [a4, a7]
plan8 = [a5, a7]
plan9 = [a4, a7, a5, a8]
plan10 = [a0, a1, a2, a3]
plan11 = [a4, a7, a5, a8, a0, a1, a2, a3]
plan12 = [a4, a7, a5, a8, a4, a5, a0, a1, a2, a3]

plans = [plan0, plan1, plan2, plan3, plan4, plan5, plan6, plan7, plan8, plan9,
         plan10, plan11, plan12]
plans = plans[START_TASK:END_TASK+1]
```

(b) Infusing Plan to HRL Agent

The next code fragment also shows that options are also defined manually.

D.3.2 SDRL

For the Montezuma's Revenge experiment for SDRL [21]. we can see that open-source code at

`https://github.com/daomingAU/MontezumaRevenge_SDRL.`

SDRL algorithm creates options per symbolic state transition. In this domain, we see that the problem domain has a fixed number of symbolic state transitions. In practical domains, such an assumption is too restrictive that it cannot handle side-effect state transitions.

```
# go to A
policyA = EpsilonGreedy(alpha=1.0, gamma=1.0, epsilon=0.1, default_q=DEFAULT_Q,
                        num_actions=4, rng=rng)
optionA = Option(env, policyA, lambda s: 'a' in env.observe(s))  # type: ignore

# go to B
policyB = EpsilonGreedy(alpha=1.0, gamma=1.0, epsilon=0.1, default_q=DEFAULT_Q,
                        num_actions=4, rng=rng)
optionB = Option(env, policyB, lambda s: 'b' in env.observe(s))  # type: ignore

# go to C
policyC = EpsilonGreedy(alpha=1.0, gamma=1.0, epsilon=0.1, default_q=DEFAULT_Q,
                        num_actions=4, rng=rng)
optionC = Option(env, policyC, lambda s: 'c' in env.observe(s))  # type: ignore

# go to D
policyD = EpsilonGreedy(alpha=1.0, gamma=1.0, epsilon=0.1, default_q=DEFAULT_Q,
                        num_actions=4, rng=rng)
optionD = Option(env, policyD, lambda s: 'd' in env.observe(s))  # type: ignore

# get mail
policyE = EpsilonGreedy(alpha=1.0, gamma=1.0, epsilon=0.1, default_q=DEFAULT_Q,
                        num_actions=4, rng=rng)
optionE = Option(env, policyE, lambda s: 'e' in env.observe(s))  # type: ignore

# get coffee
policyF = EpsilonGreedy(alpha=1.0, gamma=1.0, epsilon=0.1, default_q=DEFAULT_Q,
                        num_actions=4, rng=rng)
optionF = Option(env, policyF, lambda s: 'f' in env.observe(s))  # type: ignore

# go to office
policyG = EpsilonGreedy(alpha=1.0, gamma=1.0, epsilon=0.1, default_q=DEFAULT_Q,
                        num_actions=4, rng=rng)
optionG = Option(env, policyG, lambda s: 'g' in env.observe(s))  # type: ignore

options = [optionA, optionB, optionC, optionD, optionE, optionF, optionG]
```

Figure 14: Options in Taskble RL

E MonteZuma's Revenge Stage 1

E.1 State mapping function and results

The state mapping function in Montezuma's revenge domain uses a mapping from an image to a symbolic state representation. For the state mapping, we used the RL environment modified in [42] that captures the location of the agent in the image by counting the different pixels in the selected bounding boxes and the pixels around the agent as shown in Figure 15. The planning task guides the agent to move from the initial location to reach the door after obtaining the key in the planning state space with the predicates, *init*, *chain*, *lru*, *lrd*, *lld*, *llu*, *key*, *lcm*, and *door*.

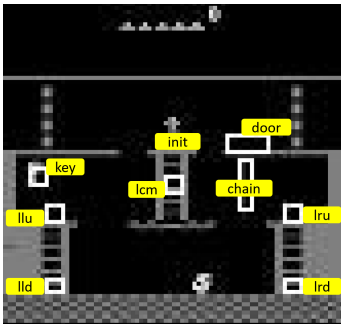


Figure 15: The state mapping from image to planning state in Montezuma's revenge domain.

	HplanPPO	PPO	HIRL+DDQN
samples	2,904,000	4,000,000	2,400,000
score	$\tilde{4}00$	42	$\tilde{4}00$

Table 1: Sample complexity and score results from HplanPPO, PPO, and HIRL+DDQN

Table 1 shows the number of sample interactions and the score achieved in the baseline algorithms reported by [41] and [42]. Note that it is not fair to evaluate three algorithms based on the results shown in Table 1 since PPO is a model-free on-policy algorithm, HIRL [42] is an imitation learning approach with the off-policy DQN, and the online on-policy plan-HRL requires the planning task that annotates the RL task. Nevertheless, we can observe that plan-HRL improves sample efficiency compared with its flat RL counter part PPO and used 20 % more samples than HIRL, which uses the demonstrations and the samples stored in the replay buffer.

E.2 Planning Task for MonteZuma's Revenge Stage 1

The RL environment maintains the images obtained from the arcade learning environment, and we selected bounding boxes in the image to generate planning states from the RL state. Figure 15 shows the bounding boxes we used. The PDDL domain file used for the planning task is desribed as follows.

```
(define (domain montezuma)
  (:requirements :strips :typing)
  (:types
      location - object
  )
  (:predicates
      (CONNECTED ?x ?y - location)
      (at ?l - location)
      (key-at ?l - location)
      (holding-key)
  )
  (:action move
      :parameters (?from ?to - location)
      :precondition (and
          (at ?from)
          (CONNECTED ?from ?to)
      )
      :effect (and
          (at ?to)
          (not (at ?from))
      )
  )
  (:action get-key
      :parameters (?from)
      :precondition (and
          (at ?from)
          (key-at ?from)
      )
      :effect (and
          (at ?from)
          (holding-key)
      )
  )
)
```

From each bounding box, we assign a location object and ground predicates by assigning objects to the variables in the PDDL domain definition. The PDDL problem file used for the planning task is desribed as follows.

```
(define (problem montezuma-room1)
  (:domain montezuma)
  (:objects
      INI LRD LCM LRU DOOR LLU LLD CHA  - location
  )
  (:init
      (CONNECTED INI CHA)
```

```
      (CONNECTED CHA LRU)
      (CONNECTED LRU LRD)
      (CONNECTED LRD LLD)
      (CONNECTED LLD LLU)
      (CONNECTED LLU LLD)
      (CONNECTED LLD LRD)
      (CONNECTED LRD LRU)
      (CONNECTED LRU CHA)
      (CONNECTED LRU CHA)
      (CONNECTED CHA LCM)
      (CONNECTED LCM DOOR)
      (key-at LLU)
      (at INI)
  )
  (:goal (and
      (holding-key)
      (at DOOR))
  )
)
```

E.3 Montezuma's revenge domain hyper-parameters for HplanPPO

- learning rate: 2.5e-4
- gae: 0.95
- clip: 0.1
- vf coeff: 1
- max episode length: 1024
- rollout length: 1024
- batch size: 64
- network dimension: CNN baed architecture as shown in (Minh, 2015).
- gamma: 0.99
- epochs: 4
- entropy coefficient: 0.01
- step cost: -0.01
- reward: +1