A HIERARCHICAL BAYESIAN APPROACH TO INVERSE REINFORCEMENT LEARNING WITH SYMBOLIC REWARD MACHINES

Anonymous authors
Paper under double-blind review

ABSTRACT

A misspecified reward can degrade sample efficiency and induce undesired behaviors in reinforcement learning (RL) problems. We propose *symbolic reward machines* for incorporating high-level task knowledge when specifying the reward signals. Symbolic reward machines augment existing reward machine formalism by allowing transitions to carry predicates and symbolic reward outputs. This formalism lends itself well to inverse reinforcement learning, whereby the key challenge is determining appropriate assignments to the symbolic values from a few expert demonstrations. We propose a hierarchical Bayesian approach for inferring the most likely assignments such that the concretized reward machine can discriminate expert demonstrated trajectories from other trajectories with high accuracy. Experimental results show that learned reward machines can significantly improve training efficiency for complex RL tasks and generalize well across different task environment configurations.

1 INTRODUCTION

Reinforcement Learning (RL) agents rely on rewards to measure the utility of each interaction with the environment. As the complexity of RL tasks increases, it becomes difficult for the agent to grasp the intricacies of the task solely from goal-driven reward functions – rewarding the agent only at the end of each episode. Reward machine (RM) is a formalism proposed by Icarte et al. (2020) for representing a reward function as a finite state automaton (FSA). However, the design of RMs can quickly become cumbersome as the complexity of the tasks increases. In this paper, we draw inspiration from *symbolic finite automaton* (SFA) and *symbolic finite transducer* (SFT) Antoni & Veanes (2017) and propose *symbolic reward machine* (SRM) which augment FSA-based RMs by allowing the internal state transitions of the RM to carry predicates and functions on the trajectory. In addition to improving interpretability and conciseness of the reward model, a salient benefit of SRM is that it allows human engineers to express complex task scenarios and reward design patterns.

In many existing works on logically designed reward functions, the reward values are manually assigned based on heuristics which can undermine the effectiveness of the resulting reward functions. For instance, if a learning agent is excessively awarded for the completion of a certain stage of a task, the agent may tend to repeatedly complete the same stage to accumulate rewards instead of commencing the next stage, a phenomenon known as *reward hacking* Amodei et al. (2016). We envision that, in a typical design routine of an SRM, a human engineer constructs the SRM to incorporate high-level task information, but leave the low-level details, such as the right amount of reward for a specific event or sub-task, empty or as *holes*. The SRM formalism also facilitates the specification of *symbolic constraints* over the holes for capturing important task-specific nuances.

A major contribution of this paper is a learning framework for filling out the holes in an SRM, which we call *concretization*. The input to the framework includes a set of *example trajectories* demonstrated by a human expert, an SRM and optionally a *symbolic constraint* that the holes in the SRM must satisfy. We leverage the generative adversarial approaches from Finn et al. (2016); Jeon et al. (2018) to construct a discriminator with a neural network reward function to distinguish the expert

trajectories from the trajectories of an agent policy. However, our framework works in a hierarchical Bayesian manner. Basically, to circumvent the non-differentiability of SRMs, we employ a sampler to sample candidate instantiations of the holes to concretize the SRM. Then we introduce a stochastic reward signal as the latent variable dependent on the output of the concretized SRM and employ a neural-network reward function to perform importance sampling of the stochastic rewards for trajectory discrimination. We summarize our contributions below.

- We propose to use SRMs to represent reward functions for RL tasks.
- We develop a hierarchical Bayesian inference framework that can concretize an SRM by learning from expert demonstrations.
- Our approach enables RL agents to achieve state-of-the-art performance on highly complex environments with only a few demonstrations. In addition, we show that SRMs generalize well across different environment configurations of the same task.

2 Related Work

Inverse Reinforcement Learning. We first note that the IRL formulation proposed in Ng & Russell (2000); Abbeel & Ng (2004) has an infinite number of solutions. The Max-Entropy IRL from Ziebart et al. (2008), Max-Margin IRL from Abbeel & Ng (2004); Ratliff et al. (2006) and Bayesian IRL from Ramachandran & Amir (2007) aim at resolving the ambiguity of IRL. However, those approaches restrict the reward function to be linear on the basis of human designed feature functions. Deep learning approaches proposed in Fu et al. (2018); Ho & Ermon (2016); Jeon et al. (2018); Finn et al. (2016) have substantially improved the scalablity of IRL by drawing a connection between IRL and Generative Adversarial Networks (GANs) introduced by Goodfellow et al. (2014). Our work, while embracing the data-driven and generative-adversarial ideologies, further extends IRL to cope with symbolically represented human knowledge.

Reward Design. There have been substantial efforts on enriching the information in reward functions. Reward shaping proposed by Ng et al. (1999) adds state-based potentials to the reward in each state. Exploration driven approaches such as Bellemare et al. (2016); Pathak et al. (2017); Alshiekh et al. (2017); Flet-Berliac et al. (2021) incentivize agents with intrinsic rewards. Compared with these methods, we do not seek to generate reward functions densely ranging over the entire state space but rather design intepretable ones that selectively or even sparsely produce non-zero rewards. Reward machines from Icarte et al. (2020) directly represent the reward functions as FSAs. The symbolic reward machine in our work is also automata-based but augments RMs in a similar way to SFA for FSA Veanes et al. (2012). There have been efforts on learning a so-called perfect RM as termed in Toro Icarte et al. (2019) from the experience of an RL agent in partially observable environment. However, the RM is still based on FSA and the rewards are still manually assigned. Regarding leveraging human demonstrations, inverse reward design (IRD) proposed in Hadfield-Menell et al. (2017) is analogous to IRL but aims at inferring a true reward function from some proxy reward function perceived by a RL agent. Safety-aware apprenticeship learning from Zhou & Li (2018) pioneers the incorporation of formal verification in IRL. However, those works confine the reward functions to be linear of features as the generic IRL does. Our work does not have such limitations.

Interpretable Reinforcement Learning. There have been continual researches on designing interpretable policies Andre & Russell (2001; 2002); Verma et al. (2018); Zhu et al. (2019); Yang et al. (2021); Tian et al. (2020). This paper concerns the design of interpretable reward functions rather than interpretable policies. Our motivation is that a well-designed reward function is transferable and can be a powerful complement to the vast literature on RL policy learning.

3 Background

An RL environment is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, d_0 \rangle$ where $\mathcal{S}$ is the state space; $\mathcal{A}$ is an action space; $\mathcal{P}(s'|s,a)$ is the probability of reaching a state s' by performing an action a at a state s; d_0 is an initial state distribution. A *policy* $\pi(a|s)$ determines the probability of an RL agent performing an action a at state s. By successively performing actions for T steps after initializing from a state $s^{(0)} \sim d_0$, a *trajectory* $\tau = s^{(0)}a^{(0)}s^{(1)}a^{(1)}\dots s^{(T)}a^{(T)}$ is produced. A state-action based *reward function*

is a mapping $f : S \times A \to \mathbb{R}$ to the real space. With a slight abuse of notations, we denote the total reward along a trajectory τ as $f(\tau) = \sum_{t=0}^{T} f(s^{(t)}, a^{(t)})$ and similarly for the joint probability $p(\tau|\pi) = \prod_{t=0}^{T-1} \mathcal{P}(s^{(t+1)}|s^{(t)}, a^{(t)})\pi(a^{(t)}|s^{(t)})$ of generating a trajectory τ by following π. The objective of an entropy-regularized RL task is to maximize $J_{RL}(\pi) = \mathbb{E}_{\tau\sim\pi}[f(\tau)] + \mathcal{H}(\pi)$ where $\tau \sim \pi$ is an abbreviation for $\tau \sim p(\tau|\pi)$ and $\mathcal{H}(\pi)$ is the expected entropy of π.

When the reward function is unknown but a set of expert trajectories τ_E is sampled with some expert policy π_E, GAIL Ho & Ermon (2016) trains an agent policy π_A to match π_E by minimizing J_{adv} in Eq.2 via RL algorithms such as PPO Schulman et al. (2017). Adversarially, GAIL optimizes a discriminator $D : \mathcal{S} \times \mathcal{A} \to [0, 1]$ to accurately identify τ_E's from $\tau_A \sim \pi_A$ by maximizing J_{adv}. From a probabilistic inference perspective, Bayesian GAIL from Jeon et al. (2018) labels any expert trajectory τ_E with 1_E and 0_E to respectively indicate τ_E as being sampled from an expert demonstration set E and from some agent policy π_A. Likewise, the trajectory τ_A of π_A is labeled with 1_A and 0_A for the same indications. Assuming that the labels $0_A, 1_E$ are known *a priori*, Bayesian GAIL solves the most likely discriminator D by maximizing $p(D|0_A, 1_E; \pi_A; E) \propto p(D)p(0_A, 1_E|\pi_A, D; E) \propto \sum_{\tau_A} p(\tau_A|\pi_A)p(0_A|\tau_A; D) \sum_{\tau_E} p(\tau_E|E)p(1_E|\tau_E; D)$ of which the logarithm as in Eq.1 is lower-bounded due to Jensen's inequality by Eq.2. It is further proposed in Fu et al. (2018) that by representing $D(s, a) = \frac{\exp(f(s,a))}{\exp(f(s,a))+\pi_A(a|s)}$ with a neural network f, when Eq.2 is maximized, it holds that $f \equiv \log \pi_E$ and f equals the expert reward function which π_E is optimal w.r.t, given that $\forall\tau.p(\tau|E) \approx p(\tau|\pi_E)$. Hence, by representing D in Eq.1 and 2 with f, an objective of solving the most likely expert reward function f is obtained.

$$\log \sum_{\tau_A} p(\tau_A|\pi_A)p(0_A|\tau_A; D) \sum_{\tau_E} p(\tau_E|E)p(1_E|\tau_E; D) \tag{1}$$

$$\geq \mathop{\mathbb{E}}_{\tau_E\sim E} \left[\log \prod_{t=0}^{T} D(s_E^{(t)}, a_E^{(t)})\right] + \mathop{\mathbb{E}}_{\tau_A\sim\pi_A} \left[\log \prod_{t=0}^{T} (1 - D(s_A^{(t)}, a_A^{(t)}))\right] := J_{adv}(D) \tag{2}$$

4 Symbolic Reward Machines (SRMs)

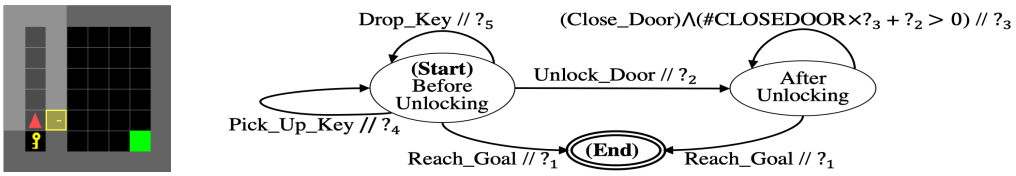
We first give a formal definition of SRM and motivate the use of SRMs with a task from a Mini-Grid environment introduced in Chevalier-Boisvert et al. (2018). We highlight that the human insights incorporated in the SRM can hardly be realized with conventional goal-driven reward mappings or function approximations. We then formulate the problem of concretizing SRMs.

4.1 Definition

The definition of SRM is inspired from those of SFA and SFT in Veanes et al. (2012). To adapt them to the RL setting, we assume a background theory equipped with fixed interpretations on the state and actions in the RL environment as well as a language of functions. Following Pierce (2002), a λ-term is a function written in the form of $\lambda x.\rho$. The type of a $\lambda x.\rho$ is a mapping from the type of input argument x to the type of function body ρ. The free variable set $FV(\rho)$ of ρ is the set of symbolically denoted variables appearing in ρ while not appearing in the input of any λ-term inside ρ. Given some concrete input $\hat{x}$, the evaluation of $\lambda x.\rho$ is written as $[\![\lambda x.\rho]\!](\hat{x})$ or $[\![\rho[\hat{x}/x]]\!]$ where $[\hat{x}/x]$ represents the replacement of x with $\hat{x}$ in ρ. The denotation stays the same if x and $\hat{x}$ are not unary. A predicate is a specific set of λ-terms mapping to Boolean type $\mathbb{B} = \{\top, \bot\}$ where $\top, \bot$ mean *True* and *False* respectively. The set of predicates is closed under Boolean operations $\wedge, \vee, \neg$.

Definition 1. *Given an RL environment $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, d_0 \rangle$, a **symbolic reward machine (SRM)** is a tuple $\mathcal{L} = \langle \mathcal{Q}, \Psi, \mathcal{R}, \delta, q_0, Acc \rangle$ where $\mathcal{Q}$ is a set of internal states; $\Psi \subseteq (\mathcal{S} \times \mathcal{A})^* \to \mathbb{B}$ is a set of predicates on trajectories in $\mathcal{M}$; $\mathcal{R}$ is a set of λ-terms of type $(\mathcal{S} \times \mathcal{A})^* \to \mathbb{R}$; δ is a set of transition rules (p, ψ, r, q), where $p, q \in \mathcal{Q}, \psi \in \Psi, r \in \mathcal{R}$; $q_0 \in \mathcal{Q}$ is an initial state; $Acc \subseteq \mathcal{Q}$ is a set of accepting states; the free variables set of $\mathcal{L}$ is defined as $FV(\mathcal{L}) = \bigcup_{\rho\in\mathcal{R}\cup\Psi} FV(\rho)$.*

We use the notation $p \xrightarrow{\psi//r} q$ for a rule $(p, \psi, r, q) \in \delta$ and call ψ its guard. The input to an SRM is a trajectory $\tau \in (\mathcal{S} \times \mathcal{A})^*$. A rule $p \xrightarrow{\psi//r} q$ is applicable at p iff $[\![\psi]\!](\tau) = \top$, in which case $\mathcal{L}$ outputs a reward $[\![r]\!](\tau)$ for the last state-action pair in τ while the state p transitions to q. If no rule is applicable, the state p does not transition and $\mathcal{L}$ outputs a fixed constant reward such as 0, in



(a) (b)

Figure 1: a) A 8x8 DoorKey task; b) The diagram of an SRM designed for the DoorKey task.

which case we dub a dummy transition $p \xrightarrow{\exists q \in \mathcal{Q}.(p,\psi,r,q)\in\delta} {}^{\bigwedge(\neg\psi)\ //0} p$. An SRM $\mathcal{L}$ is called deterministic if given an input trajectory τ at any state $p \in \mathcal{Q}$, $|\{(p,\psi,r,q) \in \delta | \psi(\tau) = \top\}| \leq 1$ in which case with a little abuse of notations we write the next state as $q = \delta(p,\tau)$ either obtained from the uniquely applicable transition rule $(p,\psi,r,q) \in \delta$ s.t. $\psi(\tau) = \top$, or from a dummy transition such that $q = p$. To deploy a deterministic SRM in an RL task is to construct a synchronous product as defined below.

Definition 2. *A synchronous product between an $\mathcal{M}$ and a deterministic $\mathcal{L}$ is a tuple $\mathcal{M} \otimes \mathcal{L} = \langle (\mathcal{S}\times\mathcal{A})^* \times \mathcal{S} \times \mathcal{Q}, \mathcal{A}, \Psi, \mathcal{R}, \mathcal{P} \odot \delta, d_0, q_0, Acc\rangle$ where a product state in $(\mathcal{S}\times\mathcal{A})^* \times \mathcal{S} \times \mathcal{Q}$ is a pair $(\tau :: s, q)$ where $::$ means concatenation; $\tau :: s$ means concatenating a trajectory $\tau \in (\mathcal{S}\times\mathcal{A})^*$ with an $\mathcal{M}$ state $s \in \mathcal{S}$; $q \in \mathcal{Q}$ is an $\mathcal{L}$ state; the product transition rule $\mathcal{P} \odot \delta$ follows Eq.3 where $\tau :: s :: a$ is a trajectory resulted from further concatenating $\tau :: s$ with an action $a \in \mathcal{A}$; the initial product state is (s_0, q_0) where $s_0 \sim d_0$; the rest follows the definitions in $\mathcal{M}$ and $\mathcal{L}$.*

$$(\mathcal{P}\odot\delta)((\tau :: s, p), a, (\tau :: s :: a :: s', q)) = \begin{cases} \mathcal{P}(s'|s,a) & if\ q = \delta(p, \tau :: s :: a) \\ 0 & Otherwise \end{cases} \tag{3}$$

Proposition 1. *Suppose that $\mathcal{L}$ is deterministic. Starting from any $p^{(0)} \in \mathcal{Q}$, as the input trajectory extends from $\tau = s^{(0)}a^{(0)}$ to $\tau = s^{(0)}a^{(0)}\ldots s^{(T)}a^{(T)}$, there can only be at most one path $\sigma = p^{(0)}p^{(1)}\ldots q^{(T+1)}$ obtained by applying the uniquely applicable or a dummy transition successively to τ from $t = 0$ to $t = T$, i.e., $\forall t = 0, 1, \ldots, T. p^{(t+1)} = \delta(q^{(t)}, s^{(0)}a^{(0)}\ldots s^{(t)}a^{(t)})$.*

Suppose that at time step t there is a transition from $(\tau :: s^{(t)}, p^{(t)})$ to $(\tau :: s^{(t)} :: a^{(t)} :: s^{(t+1)}, p^{(t+1)})$ with $(\mathcal{P}\odot\delta)((\tau :: s^{(t)}, p^{(t)}), a^{(t)}, (\tau :: s^{(t)} :: a^{(t)} :: s^{(t+1)}, p^{(t+1)})) > 0$. While $\mathcal{L}$ witnesses such product state transition, the RL agent only observes the state transition in $\mathcal{M}$. Furthermore, $\mathcal{L}$ returns a reward $[\![r^{(t)}]\!](\tau :: s^{(t)} :: a^{(t)})$ at time step t, where $r^{(t)}$ is either 0 or the λ-term associated with the rule $p^{(t)} \xrightarrow{\psi^{(t)}//r^{(t)}}_{\mathcal{L}} p^{(t+1)}$. We further inductively define $[\![\mathcal{L}]\!](\tau)$ as $[\![\mathcal{L}]\!](\tau :: s^{(t)} :: a^{(t)}) = [\![\mathcal{L}]\!](\tau) :: [\![r^{(t)}]\!](\tau :: s^{(t)} :: a^{(t)})$. For simplicity, we also write $\tau :: s :: a$ as $\tau :: (s, a)$ as if τ is a list of (s, a)'s, if it does not raise ambiguity in the context. We denote $\mathcal{L}(\tau) = \sum_{t=0}^{T-1} [\![\mathcal{L}]\!](\tau[:t])$ where $\tau[:t]$ is the partial trajectory from initialization up until step t.

For clarification, we remark that an SRM is an SFT only under stricter conditions, in which case the SRM retains all the properties of SFT, e.g., composability and decidability. While SFT emphasizes the acceptance of inputs, SRM emphasizes more on computing the rewards for the trajectories.

4.2 Motivating Example

Fig.1a shows an 8×8 DoorKey task in the Mini-Grid environment. An agent needs to pick up a key, unlock the yellow door on the grey wall and reach the green goal tile. In every step, the agent can observe at most the 7×7 tiles in front of it if the tile is not blocked by walls and doors. By default, the environment only returns a reward when the agent reaches the goal tile.

We show in Fig.1b the diagram of a SRM designed for this task. The symbols $\{?_{\texttt{id}}\}_{\texttt{id}=1}^{5}$ indexed by `id`'s are free variables, which we dub *holes*, of which values are to be determined. The ellipses in Fig.1b indicate the internal states $\mathcal{Q}$. The directed edges indicate the transition rules δ. Each edge carries a predicate ψ as well as a λ-term r separated by the $//$-sign. We omit the λ-signs and write ψ and r in plain English for explanatory purposes. Some predicates, such as `Reach_Goal`, makes proposition on the event occurring at present time step, i.e. the last state-action pair in the input

trajectory. In some other predicates, the terms starting with a #-sign, e.g., #CLOSEDOOR, takes the present trajectory as input and outputs a $\mathbb{N}_{\geq 0}$ type value, counting the number of occurrence of the event as described next to the #-sign. The initial internal state q_0 is the one that contains "**(Start)**". The state "**(End)**" constitutes a singleton Acc. We omit all the dummy transitions in the diagram, i.e., if none of the depicted transitions is applicable at a state, the output reward is a fixed constant such as 0 and this state does not transition. Determining appropriate values for the $?_{\texttt{id}}$'s, however, can be difficult. An instance of a flawed design is to have an excessively large $?_4$ which can cause *reward hacking*. Basically, the agent can repetitively pick up and drop the key to collect a higher net reward than that of reaching the goal. On the other hand, if the returned $?_5$ for dropping the key is too negative, the agent may end up avoiding picking up the key entirely. To address this issue, we can add the constraint $?_4+?_5 \leq 0$ to the SRM. Finally, we need to find appropriate assignments to the holes so that the resulting RM is effective for the RL task.

4.3 Problem Formulation

For a predicate $\lambda x.\rho$, if ρ does not include trajectory τ but includes holes $?_{\texttt{id}}$'s in its free variable set $FV(\rho)$, e.g., $\rho := ?_4+?_5 \leq 0$ in the previous example, such ρ's can be potentially used as *symbolic constraints*. When having a concrete value h_{id} for a hole $?_{\texttt{id}}$, one can concretize $\mathcal{L}$ by replacing $?_{\texttt{id}}$ with h_{id} in $\mathcal{L}$, written as $\mathcal{L}[\texttt{h}_{\texttt{id}}/?_{\texttt{id}}]$. We define the problem of concretizing an SRM below.

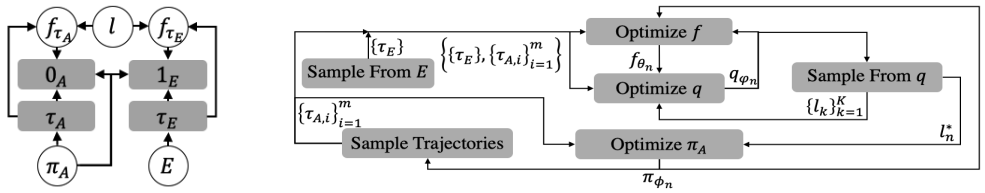
Definition 3 (**Symbolic Reward Machine Concretization**). *The concretization problem of a symbolic reward machine is a tuple $\langle \mathcal{L}, \boldsymbol{H}, c\rangle$ where $\mathcal{L}$ is an SRM with holes $\boldsymbol{?} = \{?_1, ?_2, \ldots\} \subseteq FV(\mathcal{L})$; $\boldsymbol{H} = H_1 \times H_2 \ldots$ with each $H_{\texttt{id}}$ being the assignment space of $?_{\texttt{id}}$; c is a symbolic constraint subject to $FV(c) \subseteq \boldsymbol{?}$. An SRM $\mathcal{L}$ can be concretized by any $\boldsymbol{h} \in \boldsymbol{H}$ into an $l := \mathcal{L}[\boldsymbol{h}/\boldsymbol{?}]$ iff $[\![c[\boldsymbol{h}/\boldsymbol{?}]]\!] = \top$.*

Concretizing an SRM does not readily mean that the resulting reward function will be effective for the RL task. Hence, we further assume that a set E of demonstrated trajectories is provided by the expert, thus inducing a learning from demonstration (LfD) version of the SRM concretization problem $\langle \mathcal{L}, \mathbf{H}, c, E\rangle$. The solution $\mathbf{h}$ of this problem not only concretizes the SRM $\mathcal{L}$ but also satisfies $\forall\pi.\mathbb{E}_{\tau\sim E}[\mathcal{L}[\mathbf{h}/\boldsymbol{?}](\tau)] \geq \mathbb{E}_{\tau\sim\pi}[\mathcal{L}[\mathbf{h}/\boldsymbol{?}](\tau)]$, which inherits the definition of generic IRL in Ng & Russell (2000).

5 A Hierarchical Bayesian Learning Framework

In this section, we propose an approach to solve the LfD version of the SRM concretization problem. The generic IRL approach cannot be used to solve for the holes since the SRM such as the one in Fig.1b may have the holes in the transition predicates. Our approach is inspired by Bayesian GAIL as mentioned in the Background section. However, directly using $\mathcal{L}$ to substitute f in the discriminator D in Eq.2 is not practical due to the following challenges: *a) $\mathcal{L}$ is not differentiable w.r.t the holes; b) $\mathcal{L}$ is trajectory based.* In short, stochastic gradient descent with batched data is not readily applicable. Hence, we propose a hierarchical inference framework to circumvent this issue.

An overview of the graphical model of our hierarchical inference framework is shown in Fig.2a. Given a π_A, we seek the most likely concretized SRM l from the log-likelihood $\log p(l|0_A, 1_E; \pi_A, E) = \log p(0_A, 1_E|\pi_A, E, l)p(l) + constant$ where the prior $p(l)$ can be an uniform distribution over some allowable SRM set. The log-likelihood $\log p(0_A, 1_E|\pi_A, E, l)$ is factorized as in Eq.4 by introducing two latent factors, f_{τ_A} and f_{τ_E}, which are two sequences of rewards for the state-action pairs along τ_A and τ_E. On one hand, each element of f_τ constitutes a discriminator for the labels 0_A or 1_E in the same way as the reward $f(s, a)$ does in the discriminator D of Eq.2. On the other hand, f_τ is viewed as a noisy observation of $[\![l]\!](\tau)$ in that the latent distribution $p(f_\tau|\tau; l)$ is interpreted as the likelihood of observing f_τ given $[\![l]\!](\tau)$. Here, we adopt a tractable model such as Gaussian noise $\epsilon \sim \mathcal{N}(0, 1)$ to simulate a stochastic reward $f_\tau = [\![l]\!](\tau) + \epsilon$ which means adding the same ϵ to each reward in the reward sequence $[\![l]\!](\tau)$. It is trivially provable that given any reward function f, supposed that a policy π is the optimal policy to maximize $J_{RL}(\pi)$, then this π retains optimal if f is replaced with its stochastic version $f + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 1)$, if the trajectory lengths are considered equal. To measure the integrals in Eq.4, we re-introduce a neurally simulated reward function f for the importance sampling of the stochastic f_τ's. We define $p(f_\tau|\tau; f)$ in the same way as $p(f_\tau|\tau; l)$ except for replacing $f_\tau = [\![l]\!](\tau) + \epsilon$ with $f_\tau = f(\tau[t]) + \epsilon$. With the sampled $f_\tau \sim p(f_\tau|\tau; f)$, we obtain a lower-bound Eq.5 where the



(a) (b)

Figure 2: a) The probabilistic graphical model of our framework; b) The flow chart of Algorithm 1

GAIL objective J_{adv} as defined in Eq.2 is embedded but with $D_\epsilon(s,a) := \frac{\exp(f(s,a)+\epsilon)}{\exp(f(s,a)+\epsilon)+\pi_A(a|s)}$ in place of D. We also abbreviate $p(\cdot|\tau;l)$ and $p(\cdot|\tau;f)$ as $p_l(\tau)$ and $p_f(\tau)$ in the KL-divergence $D_{KL}(p_f(\tau)||p_l(\tau))$, which can be viewed as a regularization term and turns out to be proportional to the squared error $\sum_{t=0}^{T-1}([\![l]\!](\tau[:t]) - f(\tau[t]))^2$. With Eq.5 we decouple the optimization of the GAIL objective J_{adv} from l such that a standard stochastic gradient descent with batched data can be conducted on the GAIL objective J_{adv}. We prove in the Appendix that the stochastic version $\mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}[J_{adv}(D_\epsilon)]$ has the same optimal condition as that of $J_{adv}(D)$ in Eq.2.

$$\log p(0_A, 1_E|\pi_A, E, l) := \log \sum_{\tau_A,\tau_E} p(\tau_A|\pi_A)p(\tau_E|E)$$

$$\iint_{f_{\tau_A} f_{\tau_E}} p(0_A|\tau_A;\pi_A,f_{\tau_A})p(1_E|\tau_E;\pi_A,f_{\tau_E})p(f_{\tau_E}|\tau_E;l)p(f_{\tau_A}|\tau_A;l) \tag{4}$$

$$\geq \max_f \mathop{\mathbb{E}}_{\epsilon\sim\mathcal{N}(0,1)}\Big[J_{adv}(D_\epsilon)\Big] - \mathop{\mathbb{E}}_{\tau\sim\pi_A,E}\Big[D_{KL}(p_f(\tau)||p_l(\tau))\Big] \tag{5}$$

As Eq.5 is still not differentiable w.r.t. the holes in l, we resort to finding a distribution q of l to match the likelihood $p(l|0_A, 1_E;\pi_A, E)$. It can be realized by minimizing a KL-divergence $D_{KL}\Big[q(l)||p(l|0_A,1_E;\pi_A,E)\Big] = \mathop{\mathbb{E}}_{l\sim q}\Big[\log q(l) - \log\frac{p(0_A,1_E|\pi_A,E,l)p(l)}{p(0_A,1_E|\pi_A,E)}\Big]$, or by maximizing its evidence lower-bound (ELBO) which is further lower-bounded by Eq.6. When a symbolic constraint is considered, the prior $p(l)$ can be viewed as being uniform only among those l's satisfying the symbolic constraint while being zero everywhere else. We let $J_{con}(q) := -D_{KL}[q(l)||p(l)]$ be a supervised learning objective and abbreviate everything else in Eq.6 as a soft-maximization objective $J_{soft}(q,f)$.

$$\begin{aligned} ELBO(q) &\geq \max_f \mathop{\mathbb{E}}_{\epsilon\sim\mathcal{N}(0,1)}\Big[J_{adv}(D_\epsilon)\Big] - \mathop{\mathbb{E}}_{\substack{l\sim q\\ \tau\sim\pi_A,E}}\Big[D_{KL}(p_f(\tau)||p_l(\tau))\Big] - D_{KL}\Big[q(l)||p(l)\Big] \\ &:= J_{soft}(q,f) + J_{con}(q) \end{aligned} \tag{6}$$

In our implementation, we consider the case when the holes are all real numbers, i.e., $\forall id.H_{id} = \mathbb{R}$. We construct a neurally simulated sampler q_φ to output the mean and diagonal variance matrix of a multivariate Gaussian distribution of which the dimension equals the number of holes. As each hole assignment $\mathbf{h}$ sampled from this Gaussian corresponds to a $l := \mathcal{L}[\mathbf{h/?}]$, we still denote by $q_\varphi(l)$ the distribution of l's. Besides q_φ, we let f_θ be the neurally simulated f. To calculate the gradients of $J_{soft}(q_\varphi, f_\theta)$ w.r.t φ and θ, we use the logarithmic trick from Peters & Schaal (2008) to handle $\mathbb{E}_{l\sim q_\varphi}[\cdot] \approx \frac{1}{K}\nabla_{\varphi_i}\log q_{\varphi_i}(l_k)[\cdot]$ with K samples of concretized SRMs. Reparameterization trick Kingma & Welling (2013) is also used to optimize the stochastic adversarial objective $\mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}[J_{adv}(D_\epsilon)]$ w.r.t f_θ. As for J_{con}, since we only consider the case where the symbolic constraints are all conjunctions of atomic predicates that only involve linear arithmetic, e.g., $?_4+?_5 \leq 0$, we make a relaxation by assigning the mean of the multivariate Gaussian produced by q_φ to the holes and then evaluating a binary cross-entropy loss of violating the symbolic constraint. As a result, J_{con} is differentiable w.r.t φ. We use a neural network π_ϕ to simulate the agent policy π_A and train it with the most likely $l^* = \arg\max_l[q_\varphi(l)]$ which can also be readily obtained from the mean of the multivariate Gaussian distribution specified by q_φ. We summarize the algorithm in Algorithm 1 and illustrate the flow chart of Algorithm 1 in Fig.2b.

Algorithm 1 Hierarchical Bayesian Inference For SRM Concretization

Input: Expert demonstration E, initial agent policy π_{ϕ_0}, reward function f_{θ_0}, sampler q_{φ_0}, iteration number $i = 0$, maximum iteration number N
Output: π_{ϕ_N} and q_{φ_N}

1: **while** iteration number $i < N$ **do**
2: Sample trajectory set $\{\tau_{A,i}\}_{i=1}^{m}$ by using policy π_{ϕ_i}
3: Computing reward $\{l^*(\tau_{A,i})\}_{i=1}^{m}$ with the most likely $l^* = \arg\max_l q_{\varphi_i}(l)$
4: Update $\phi_i \rightarrow \phi_{i+1}$ using policy optimization, e.g., PPO
5: Sample K samples $\{l_k\}_{i=1}^{K}$ by using q_{φ_i}
6: Sample $\{f_{\theta_i}(s,a) + \epsilon | \epsilon \sim \mathcal{N}(0,1)\}$ respectively with $(s,a) \in E$ and $\{\tau_{A,i}\}_{i=1}^{m}$
7: Update $\theta_{i+1} \leftarrow \theta_i + \alpha \nabla_{\theta_i} J_{soft}(q_{\varphi_i}, f_{\theta_i})$ with a step size parameter α
8: Update $\varphi_{i+1} \leftarrow \varphi_i + \beta \nabla_{\varphi_i} J_{soft}(q_{\varphi_i}) + \beta\eta \nabla_{\varphi_i} J_{con}(q_{\varphi_i})$ with step size parameters β, η
9: **end while**
10: **return** π_{ϕ_N} and q_{φ_N}

6 EXPERIMENTS

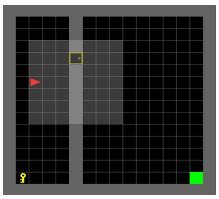
Our benchmark includes three tasks of growing difficulty in the Mini-Grid environment: *Door-Key*, *KeyCorridor* and *ObstructedMaze*. The first task has been introduced earlier. The latter two are respectively shown in Fig.3b and Fig.3c in which the agent needs to pick up a targeted purple/blue ball in a locked room. In KeyCorridor, there are multiple closed doors blocking the view. In ObstructedMaze, some doors are locked; the keys for the locked doors are hidden in grey boxes; and each locked door is obstructed by a green ball. We note that despite the difficulty of these tasks, our designed SRMs do not carry out any motion planning and are solely based on reasoning the significant events. The details are explained in the Appendix. In all three tasks, the environments can vary in size by changing the number of rooms and tiles (e.g., DoorKey-8x8 vs. DoorKey-16x16). The placements of the objects and doors are randomized in each instance of an environment.

6.1 MAIN RESULTS

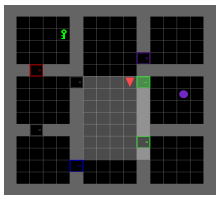
In this section, we investigate the following questions: **A. Performance**: whether Algorithm 1 can train an agent policy to achieve high average returns with a small number of environment interactions; **B. Example Efficiency**: whether Algorithm 1 can train an agent policy to achieve high performance with fewer number of demonstrated trajectories; **C. Generalization**: whether the SRM concretized by Algorithm 1 for one environment can be used to improve the performance of RL agents on a different environment for the same task.

There are two main categories of baselines: (1) generic IRL algorithms, including GAN-GCL from Fu et al. (2018) and GAIL from Ho & Ermon (2016); (2) generic RL algorithm, PPO Schulman et al. (2017), and exploration driven RL algorithms, including RIDE Raileanu & Rocktäschel (2020) and AGAC Flet-Berliac et al. (2021) which have performed well and even achieved state-of-the-art (SOTA) performance in some of the Mini-Grid tasks by generating intrinsic rewards. To answer question **A**, we implement PPO or AGAC in line 4 of Algorithm 1, and compare the results with those of the IRL baselines in which PPO is used for policy learning. We also include the results of the exploration driven RL algorithms that achieved SOTA in the benchmarks for reference. To answer question **B**, we vary the number of demonstrated trajectories and observe the results. To answer questions **C**, we directly adopt the SRMs concretized via Algorithm 1 in small environments to train RL agents in much larger environments and compare with training with the default reward.

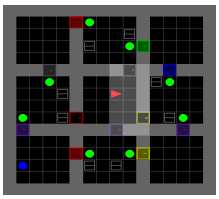
For each task, our basic setup includes 10 demonstrated trajectories, an SRM, optionally a symbolic constraint, an actor-critic agent π_ϕ, a neurally simulated reward function f_θ and a sampler q_φ that generates a multivariate Gaussian distribution. The actor-critic networks of π_ϕ have two versions, a non-recurrent CNN version and an LSTM version. In most tasks we only report the results from the versions with higher performance. The reward function f_θ is simulated by an LSTM network. For fair comparisons, we use identical hyperparameters and the same actor-critics and neurally simulated reward functions, if applicable, when comparing our approach with PPO, GAN-GCL, GAIL and AGAC. To measure training efficiency, we show how the average return, i.e. the average default reward achieved over a series of consecutive episodes by the agent policy, changes as the number



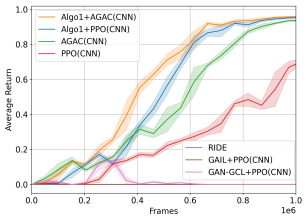
(a) DoorKey-16x16



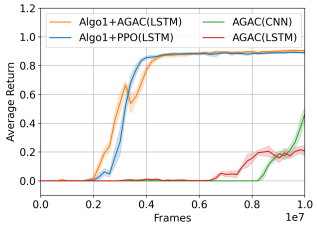
(b) KeyCorridorS6R3



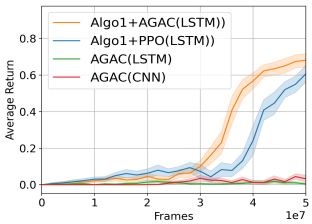
(c) ObstructedMaze-Full



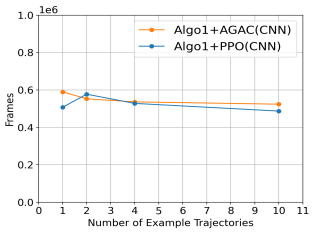
(d) DoorKey-8x8-v0



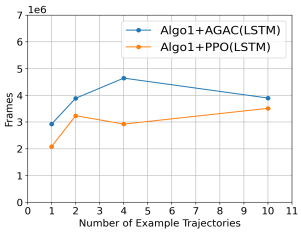
(e) KeyCorridorS3R3



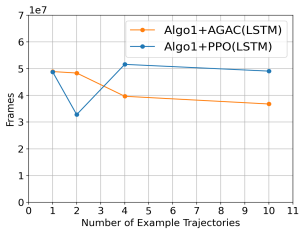
(f) ObstructedMaze-2Dlhb



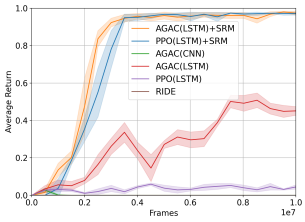
(g) DoorKey-8x8-v0



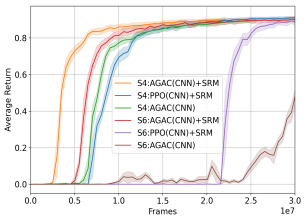
(h) KeyCorridorS3R3



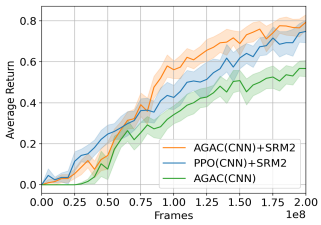
(i) ObstructedMaze-2Dlhb



(j) DoorKey-16x16-v0



(k) KeyCorridorS4/S6R3



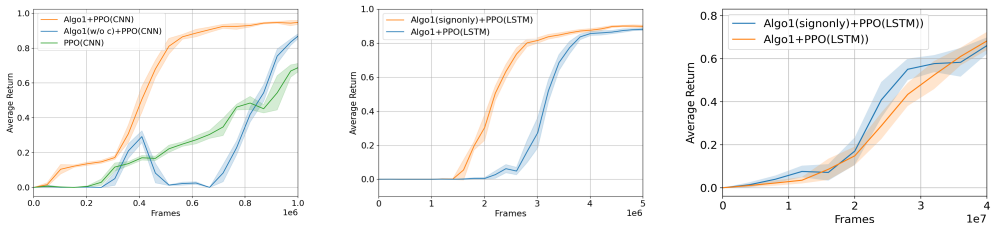
(l) ObstructedMaze-Full

Figure 3: Algo1+AGAC/PPO indicates using AGAC or PPO as the policy learning algorithm in line 4 of Algorithm 1. AGAC/PPO+SRM indicates training an AGAC or PPO agent with the concretized SRM. CNN and LSTM in the parentheses indicate the versions of the actor-critic networks. S4 and S6 in (k) indicate respectively the results for KeyCorridorS4R3 and KeyCorridorS6R3.

of frames, i.e. the number of total interactions between the agent and the environment, increases. To measure demonstrated example efficiency, we show how many frames that an algorithm takes to pass a certain level of average return for different number of demonstrations.

DoorKey. We use 10 example trajectories for the DoorKey-8x8 environment shown in Fig.2a. In Fig.3d, running Algorithm 1 by using PPO and AGAC in line 4 respectively produces policies with higher performance and needing fewer frames than by training PPO or AGAC with the default reward. RIDE, GAN-GCL+PPO(CNN) and GAIL+PPO(CNN) fail with close-to-zero returns. In Fig.3g we reduce the number of examples from 10 to 1 and it does not affect the number of frames that Algorithm 1 needs to produce a policy with average return of at least 0.8, regardless of whether PPO or AGAC is used in line 4. We use the concretized SRM to train PPO and AGAC agents in a 16x16 DoorKey environment as shown in Fig.3a and achieve higher performances with significantly fewer frames than training PPO, AGAC or RIDE with the default reward as shown in Fig.3k.

KeyCorridor. We use 10 example trajectories for a 6×6 KeyCorridorS3R3 environment. By using PPO and AGAC in line 4 of Algorithm 1, we respectively obtain higher performance with



(a) DoorKey-8x8-v0 (b) KeyCorridorS3R3 (c) ObstructedMaze-2Dlhb

Figure 4: Algo1+PPO(CNN) indicates using PPO as the policy learning algorithm in line 4 of Algorithm 1 with symbolic constraints including relational predicates; Algo1(w/o c)+PPO(CNN) indicates running Algorithm 1 without symbolic constraint; Algo1(signonly)+PPO(CNN) indicates running Algorithm 1 with symbolic constraint that only concerns the signs of the holes

significantly fewer frames than by training AGAC with the default reward as shown in Fig.3e. GAIL and GAN-GCL fail again in the KeyCorridor task. Hence we omit their results in the plot. As shown in Fig.3h, reducing the number of examples (to 1) does not affect performance of Algorithm 1 for producing a policy with average return of at least 0.8. In Fig.3k, we use the concretized SRM to train RL agents in the 10×10 KeyCorridorS4R3 and 16×16 KeyCorridorS6R3 environments, and achieve higher performances with fewer frames than training with the default rewards. We omit the results from other baselines in this task since AGAC(CNN) is the current SOTA for this task.

ObstructedMaze. We use 10 example trajectories in a two-room ObstructedMaze-2Dhlb environment to concretize three differently designed SRMs, i.e., SRM1$\sim$3, via Algorithm 1. We show the results for concretizing SRM1 in Fig.3f where Algorithm 1 produces policies with higher performance and needing fewer frames than AGAC trained with the default reward. When training the PPO agent, we discount PPO by episodic state visitation counts as in many exploration-driven approaches including AGAC. GAIL and GAN-GCL fail again in this task. As shown in Fig.3i, reducing the number of examples (to 1) does not affect the performance of Algorithm 1 for producing a policy with average return of at least 0.7. We also use all the concretized SRMs to train RL agents in a 9-room ObstructedMaze-Full environment. We show in Fig.3l that the RL agent trained with SRM2 attains high performance more efficiently than that with the default reward. The concretized SRM1 does not generalize as well (with performance similar to that of AGAC with default reward) as SRM2 does in this larger environment. The main difference between SRM1 and SRM2 is that SRM1 implements a hindsight reward modification functionality similar to that in Andrychowicz et al. (2017) but SRM2 does not. Since SRM1 modifies the hindsight reward to 0, it may make the reward signals too sparse for the RL agent to learn efficiently for the larger environment. We provide more details on the experimental comparison of all three SRMs in the Appendix.

6.2 Ablation Study

In all the previous tests, the symbolic constraints include *relational predicates*, i.e., ones that concern the relations between holes, such as the $?_4 + ?_5 \leq 0$ mentioned in the motivating example. To investigate whether Algorithm 1 can work with weaker symbolic constraint, we run Algorithm 1 *without symbolic constraint* for the DoorKey-8x8 task. For the KeyCorridorS3R3 and ObstructedMaze-2Dhlb tasks, we remove all the relational predicates and keep those concerning *only the signs of the holes*, e.g. $?_{\texttt{id}} \leq 0$. As shown in Fig.4, Algorithm 1 retains its high performance albeit needing more interactions in the DoorKey environment.

7 Conclusion

We propose symbolic reward machines to represent reward functions in RL tasks. SRMs complement policy learning methods by providing a structured way to capture high-level task knowledge. In addition, we develop a hierarchical Bayesian framework to concretize SRMs by learning from expert demonstrations. Experimental comparison with SOTA baselines on challenging benchmarks validates our approach. Future works will focus on reducing human efforts in the design of SRMs.

REFERENCES

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pp. 1–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: 10.1145/1015330.1015430. URL http://doi.acm.org/10.1145/1015330.1015430. 2

Mohammed Alshiekh, Roderick Bloem, Ruediger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. *CoRR*, abs/1708.08611, 2017. URL http://arxiv.org/abs/1708.08611. 2

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *CoRR*, abs/1606.06565, 2016. URL http://arxiv.org/abs/1606.06565. 1

David Andre and Stuart J Russell. Programmable reinforcement learning agents. In *Advances in neural information processing systems*, pp. 1019–1025, 2001. 2

David Andre and Stuart J Russell. State abstraction for programmable reinforcement learning agents. In *AAAI/IAAI*, pp. 119–125, 2002. 2

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pp. 5048–5058, 2017. 9

Loris Antoni and Margus Veanes. The power of symbolic automata and transducers. In *Computer Aided Verification, 29th International Conference (CAV'17)*. Springer, July 2017. URL https://www.microsoft.com/en-us/research/publication/power-symbolic-automata-transducers-invited-tutorial/. 1

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29:1471–1479, 2016. 2

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018. 3

Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *CoRR*, abs/1611.03852, 2016. URL http://arxiv.org/abs/1611.03852. 1, 2

Yannis Flet-Berliac, Johan Ferret, Olivier Pietquin, Philippe Preux, and Matthieu Geist. Adversarially guided actor-critic. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=_mQp5cr_iNy. 2, 7, 18, 19

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rkHywl-A-. 2, 3, 7

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014. 2, 20

Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. Inverse reward design. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 2

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pp. 4565–4573, 2016. 2, 3, 7

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *CoRR*, abs/2010.03950, 2020. URL https://arxiv.org/abs/2010.03950. 1, 2

Wonseok Jeon, Seokin Seo, and Kee-Eung Kim. A bayesian approach to generative adversarial imitation learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. 1, 2, 3

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 6

Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pp. 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-707-2. URL http://dl.acm.org/citation.cfm?id=645529.657801. 2, 5

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pp. 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. URL http://dl.acm.org/citation.cfm?id=645528.657613. 2

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017. 2

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008. 6

Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 1st edition, 2002. ISBN 0262162091. 3

Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkg-TJBFPB. 7

Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. *Urbana*, 51 (61801):1–4, 2007. 2

Nathan D. Ratliff, J. Andrew Bagnell, and Martin A. Zinkevich. Maximum margin planning. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pp. 729–736, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143936. URL http://doi.acm.org/10.1145/1143844.1143936. 2

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347. 3, 7

Lucas Tian, Kevin Ellis, Marta Kryven, and Josh Tenenbaum. Learning abstract structure for drawing by efficient motor program induction. *Advances in Neural Information Processing Systems*, 33, 2020. 2

Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. Learning reward machines for partially observable reinforcement learning. *Advances in Neural Information Processing Systems*, 32:15523–15534, 2019. 2

Margus Veanes, Pieter Hooimeijer, Benjamin Livshits, David Molnar, and Nikolaj Bjorner. Symbolic finite state transducers: Algorithms and applications. *SIGPLAN Not.*, 47(1):137–150, January 2012. ISSN 0362-1340. doi: 10.1145/2103621.2103674. URL https://doi.org/10.1145/2103621.2103674. 2, 3

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pp. 5045–5054. PMLR, 2018. 2

Yichen Yang, Jeevana Priya Inala, Osbert Bastani, Yewen Pu, Armando Solar-Lezama, and Martin Rinard. Program synthesis guided reinforcement learning. *CoRR*, abs/2102.11137, 2021. URL https://arxiv.org/abs/2102.11137. 2

Weichao Zhou and Wenchao Li. Safety-aware apprenticeship learning. In *International Conference on Computer Aided Verification*, pp. 662–680. Springer, 2018. 2

He Zhu, Zikang Xiong, Stephen Magill, and Suresh Jagannathan. An inductive synthesis framework for verifiable reinforcement learning. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 686–701, 2019. 2

Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pp. 1433–1438. AAAI Press, 2008. ISBN 978-1-57735-368-3. URL http://dl.acm.org/citation.cfm?id=1620270.1620297. 2

A APPENDIX

In this appendix, we will present additional experimental results; the design details of the SRMs and the symbolic constraints used in the experiments; a detailed experimental setup including the hyperparameters.

A.1 ADDITIONAL RESULTS

We show some addition experimental results in this section to answer the following questions.

E. Can arbitrarily concretized SRM effectively train RL agents?

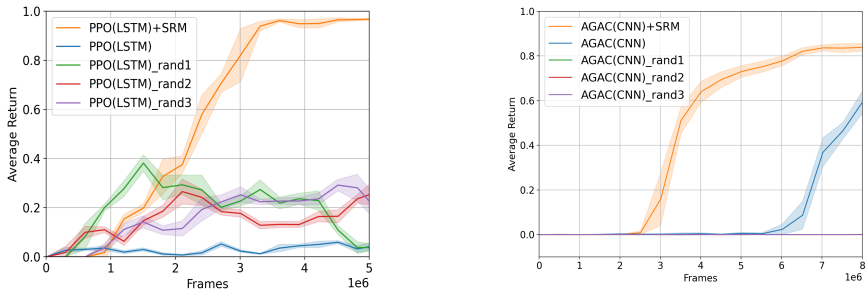
F. How much do the performance of Algorithm 1 depend on the designs of the SRMs?

For question **E**, we randomly generate hole assignments that satisfy the symbolic constraints for the SRMs of the DoorKey and KeyCorridor tasks. The SRMs are shown in Fig.8 and 9. The symbolic constraints contain the relational predicates as shown in Table.1 and 2. Those SRMs and symbolic constraints produce the main results in the main text. Now the assignments are generated by only optimizing the supervised objective J_{con} mentioned in the main text. The concretized SRMs are used for training RL policies in the following large DoorKey and KeyCorridor environments.

- **DoorKey-16x16** . In Fig.5a, we test three 3 randomly generated hole assignments for the SRM, each annotated by PPO(LSTM)_rand# . The PPO(LSTM) agents trained with those SRMs achieve certain level of performance than that trained with the default reward. However, the SRM concretized with a learned hole assignment, annotated by PPO(LSTM)+SRM, enables the agent to attain much higher performance with much lower amount of frames.
- **KeyCorridorS4R4** . we test 3 randomly generated hole assignments for the SRMs, each annotated by AGAC(CNN)_rand#. As in Fig.5b, the agents trained with the SRMs with random assignments do not perform at all. In contrast, the agent trained with the SRM that is concretized with a learned hole assignment achieves high performance with comparable amount of frames to that trained with the default reward.

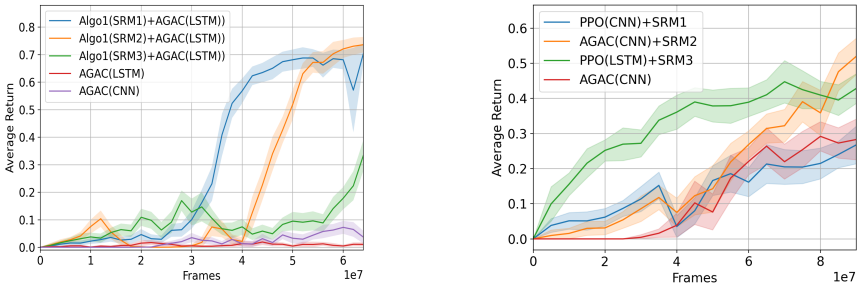
For question **F**, as mentioned in the main text we design three SRMs for the ObstructedMaze task. We will describe the difference between these SRMs in the next section. We run Algorithm 1 with those SRMs in the ObstructedMaze-2Dhlb environment and compare the results in Fig.6. In Fig.7b, we use those concretized SRMs to train RL agents in ObstructedMaze-Full. However, the SRM1 that achieves highest performance in Fig.7b is outperformed by two others.

Besides answering those two questions, we recall that we run Algorithm 1 in DoorKey and KeyCorridor tasks without symbolic constraint and with weaker symbolic constraint in the ablation study



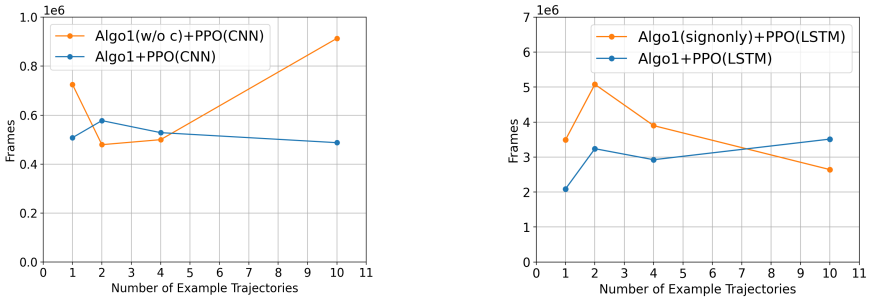
(a) DoorKey-16x16 (b) KeyCorridorS4R3

Figure 5: AGAC/PPO+SRM indicates that the hole assignments are learned via Algorithm 1; AGAC/PPO_rand# with an index # indicates that the holes are randomly assigned with some values that satisfy the symbolic constraint for that task. CNN and LSTM indicate the versions of the actor-critic networks.



(a) ObstructedMaze-2Dhlb (b) ObstructedMaze-Full

Figure 6: Algo1(SRM#)+AGAC(LSTM) with an index $\# = 1 \sim 3$ indicates running Algorithm 1 with those three designed SRMs and by using AGAC in line 4 of Algorithm 1. PPO/AGAC+SRM# indicates training RL agents with SRM# by using PPO or AGAC algorithm. CNN and LSTM indicate the versions of the actor-critic networks.



(a) DoorKey-8x8 (b) KeyCorridorS4R3

Figure 7: Algo1+PPO(CNN) indicates using PPO as the policy learning algorithm in line 4 of Algorithm 1; Algo1(w/o c)+PPO(CNN) indicates that running Algorithm 1 without symbolic constraint while using PPO(CNN) in line 4; Algo1(signonly)+PPO(CNN) indicates that running Algorithm 1 without symbolic constraint while using PPO(CNN) in line 4;CNN indicates CNN version of the actor-critic networks.

of the main text. Under the same conditions, we vary the number of demonstrations and check the number of frames needed for π_A to attain high performance. In Fig.7a and Fig.7b, we show that when the number of examples is reduced from 10 to 1, number of frames that Algorithm 1 needs to produce a policy with average return of at least 0.8 are not severely influenced.

A.2 Design Details of the SRMs

In this section, we show the diagrams of the SRMs as well as the symbolic constraints designed for the tasks. We will explain the design patterns in those SRMs in detail.

A.2.1 DoorKey Task

For readers convenience, we show the diagram of the SRM for DooKey in Fig.8. This SRM implicitly identifies an unlocking-door sub-task with two internal states "`Before Unlocking`" and "`After Unlocking`". The transitions are designed mostly based on high level human insights represented in first order logic: a) $(\texttt{Reach\_Goal@t}) \mapsto \exists \texttt{t}_1 < \texttt{t}. \exists \texttt{t}_2 < \texttt{t}_1. (\texttt{Unlock\_Door@t}_1) \wedge (\texttt{Pick\_up\_Key@t}_2)$ where `@t` indicates that the predicate preceding it, e.g., `Reach_Goal`, operates on the time step t of the trajectory τ; b) $\forall \texttt{t} \in [\texttt{t}_1, \texttt{t}_2]. (\texttt{Drop\_Key@t}_1, \texttt{Before Unlocking}) \wedge (\neg \texttt{Pick\_up\_Key@t}) \wedge (\texttt{Pick\_up\_Key@t}_2) \mapsto \forall \texttt{t}' \in [\texttt{t}_1, \texttt{t}_2]. (\neg \texttt{Unlock\_Door@t}')$ where we additionally integrate the internal state, i.e., "`Before Unlocking`", next to $\texttt{@t}_1$, to indicate the internal

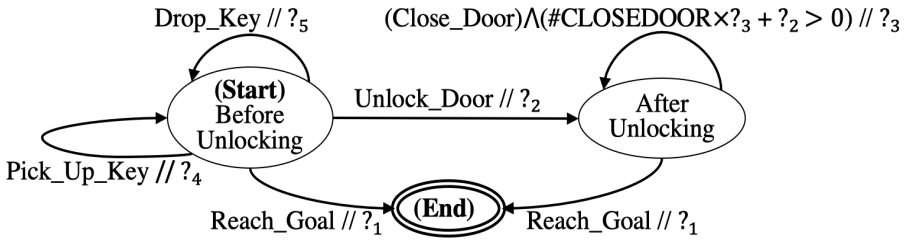


Figure 8: The diagram of the SRM designed for the DoorKey task.

Properties	**Predicates**
$[\mu_1]$Reward reaching the goal	$\bigwedge_{\texttt{id}=1}^{5} (?_{\texttt{id}} \leq ?_1)$
$[\mu_2]$Penalize dropping unused key	$?_5 + ?_4 \leq 0$
$[\mu_3]$Reward unlocking door	$\bigwedge_{\texttt{id}=2}^{5} (?_{\texttt{id}} \leq ?_2)$
$[\mu_4]$Penalty for closing door	$?_3 \leq 0$
$[\mu_5]$Mildly penalize door toggling	$?_3 + ?_2 \leq 0$

Table 1: The correspondence between properties and atomic predicates for the DoorKey SRM in Fig.1b

state at the time step t_1. The predicate $\#\texttt{CLOSEDOOR} \times ?_3 + ?_2 > 0$ in Fig.1b is introduced with due consideration of avoiding overly penalizing the agent for closing the door, which behavior is redundant for the task. The underlying idea is: *if the reward function penalized an under-trained RL agent for every door closing behavior with some high penalty* $?_3 < 0$ *for a total of* $\#\texttt{CLOSEDOOR}$ *amount of times, and the accumulated penalty* $\#\texttt{CLOSEDOOR} \times ?_3$ *outweighed the reward* $?_2 > 0$ *for unlocking the door, then the agent in practice might be inclined to reside away from the door for good.* The SRM in Fig.1b simply upper-bounds the accumulated penalty to avoid negative effects in practice. Then we show the atomic predicates in the symbolic constraint for this task in Table.1. The final symbolic constraint is $c = \bigwedge_{i=1}^{5} \mu_i$. We omit the explanation for the symbolic constraint since the atomic predicates are self-explanatory.

A.2.2 KeyCorridor Task

We depict in Fig.9 the diagram of the SRM designed for this task. Due to the added complexity in this task in comparison with the DoorKey task, two sub-tasks, finding-key and unlocking-door, are implicitly established by using three internal states "`Before Finding Key`", "`Before Unlocking`" and "`After_Unlocking`". Some important first order logic formulas that hold in most situations in the KeyCorridor task include: a) $(\texttt{Pick\_Up\_Target@t}_1) \wedge (\texttt{Unlock\_Door@t}_2) \mapsto \exists \texttt{t} \in [\texttt{t}_2, \texttt{t}_1].(\texttt{Drop\_Key@t})$;

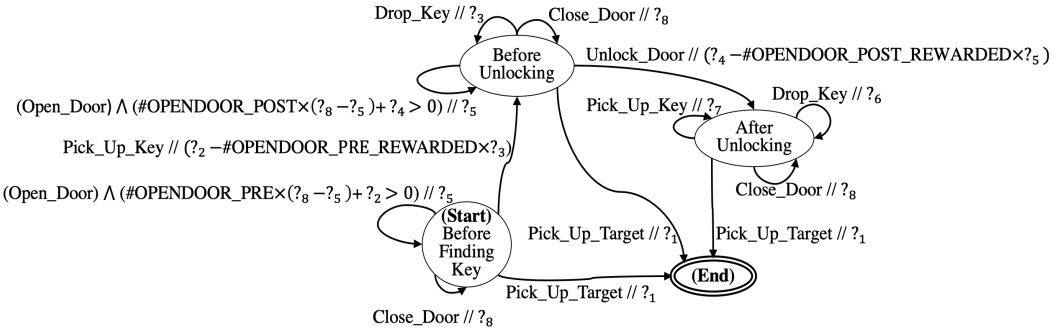


Figure 9: The diagram of the SRM designed for the KeyCorridor task.

Properties	**(Retaional) Predicates**	**(Non-Relational) Predicates**
$[\mu_1]$Reward picking up ball	$\bigwedge_{\texttt{id}=2}^{8}(?_{\texttt{id}} \leq ?_1)$	$?_1 \geq 0$
$[\mu_2]$Reward 1st time picking up key	$?_2 \geq 0$	$?_2 \geq 0$
$[\mu_3]$Reward dropping used key	$?_3 \geq 0$	$?_3 \geq 0$
$[\mu_4]$Reward unlocking door	$?_4 \geq 0$	$?_4 \geq 0$
$[\mu_5]$Encourage opening door	$?_5 \geq 0$	$?_5 \geq 0$
$[\mu_6]$Penalize meaningless move	$?_8 \leq 0$	$?_8 \leq 0$
$[\mu_7]$Moderately reward opening door	$?_5 - ?_8 \leq ?_2$	
$[\mu_8]$Penalize dropping unused key	$?_2 + ?_6 \leq 0$	$?_6 \leq 0$
$[\mu_9]$Penalize picking up used key	$?_3 + ?_7 \leq 0$	$?_7 \leq 0$

Table 2: The correspondence between properties and the relational and non-relational atomic predicates for the SRM of KeyCorridor in Fig.9

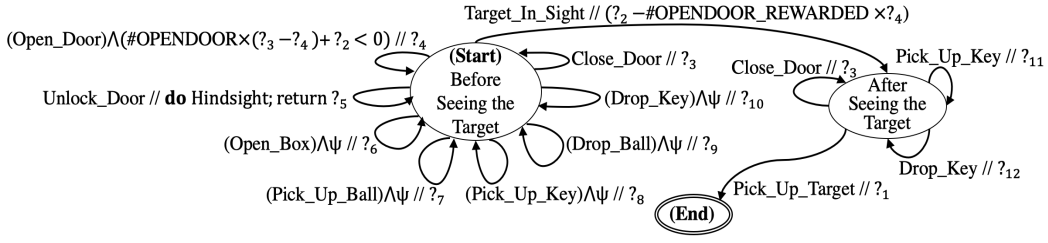
b) $\forall \texttt{t}' < \texttt{t}.(\texttt{Pick\_Up\_Key@t}) \wedge (\neg\texttt{Pick\_Up\_Key@t}') \mapsto \exists \texttt{t}'' < \texttt{t}.(\texttt{Open\_a\_Door@t}'')$; c) $(\texttt{Unlock\_Door@t}) \mapsto \exists \texttt{t}' < \texttt{t}.(\texttt{Open\_a\_Door@t}'')$. Regarding the implication a, two predicates `Pick_Up_Key@t` and `Drop_Key@t` are added at the internal state "`After Unlocking`" to govern the rewards returned for their respectively concerned behaviors after the door is unlocked. As for the implications b and c, the caveat is to determine the utility of each door opening behavior. A designer may go to one extremity by rewarding every door opening behavior with some constant, which, however, either represses exploration by penalizing opening door, or oppositely raises reward hacking, i.e., agent accumulates reward by exhaustively searching for doors to open. Alternatively, the designer may go to another extremity by carrying out a motion planning and specify the solution in the SRM, which, however, is cumbersome and cannot be generalized. In this paper, we highlight a economical design pattern to circumvent such non-determinism.

As shown in Fig.9, before the agent accomplishes the finding-key sub-task, i.e., in the "`Before Finding Key`" internal state, once the agent opens a door, the predicate $\#\texttt{OPENDOOR\_PRE} \times (?_8 - ?_5) + ?_2 > 0?$ checks whether the total reward gained from opening doors is about to exceed a threshold. The counter $\#\texttt{OPENDOOR\_PRE}$ counts the number of times that agent opens doors prior to the agent finding the key; the variable $?_8$ is expected to be a penalty for the agent closing a door, which is redundant. By introducing $?_8$, we specify that even if the agent closed doors instead of opening doors for equal number $\#\texttt{CLOSEDOOR\_PRE} \equiv \#\texttt{OPENDOOR\_PRE}$ of times, the agent could still gain positive net reward by finishing the finding-key sub-task, i.e., $\#\texttt{CLOSEDOOR\_PRE} \times ?_8 + ?_2 \geq \#\texttt{OPENDOOR\_PRE} \times ?_5$. When the agent accomplishes the finding-key sub-task, i.e., transitioning to the "`Before Unlocking`" internal state, the reward $?_2 - \#\texttt{OPENDOOR\_PRE\_REWARDED} \times ?_3$ subtracts the reward hitherto gained from opening doors with $\#\texttt{OPENDOOR\_PRE\_REWARDED} \leq \#\texttt{OPENDOOR\_PRE}$ counting the number of times that door opening behaviors are indeed awarded prior to the agent finding the key. In some sense, this approach amortizes the reward $?_2$ for finishing the finding-key sub-task over the door opening behaviors. The amortized reward $\#\texttt{OPENDOOR\_PRE\_REWARDED} \times ?_3$ cannot exceed $?_2$ and should be deducted from $?_2$. The same idea is adopted to award the door opening behaviors prior to the agent unlocking the door. The counter $\#\texttt{OPENDOOOR\_POST}$ in Fig.9 counts the number of times that agent opens doors after the agent finding the key prior to the agent unlocking the door; $\#\texttt{OPENDOOR\_PRE\_REWARDED}$ counts the number of times that door opening behaviors are awarded within that time interval. Apparently, such design pattern is convenient enough to be implemented via symbolic means. The challenge, however, remains to properly determine values for $?_{\texttt{id}}$'s. Then we show the atomic predicates in the symbolic constraint for this task in Table.2. The final symbolic constraint is $c = \bigwedge_{i=1}^{9} \mu_i$.

A.2.3 OBSTRUCTEDMAZE TASK

Fig.10 shows the diagram of the reward function designed for the ObstructedMaze task. Despite of the complexity of task, there are only three internal states, "**(Start)**", "`After Seeing the Target`" and "**(End)**". This is because only specifying the the sub-tasks is far from adequate for this task.

Once the "`After Seeing the Target`" state is reached, the reward function only concerns whether the agent drops or picks up a key or the target. In the "**(Start)**" state, the reward function views each door unlocking behavior as a milestone. If the agent does not unlock a



ψ: =?5 +#OPENBOX×(?3 −?6)+#PICKUPBALL ×(?3 −?7) + #PICKUPKEY×(?3 −?8)+#DROPBALL×(?3 −?9)+#DROPKEY×(?3 −?10) > 0

```
Hindsight :=  let t0 = Last_Time_Unlock_Door          in  Replay_Buffer.Rewards[t0:t]= 0  ;
              let t1 = Last_Time_Open_Right_Box       in  Replay_Buffer.Rewards[t1] = ?6  ;
              let t2 = Last_Time_Pick_Up_Right_Ball   in  Replay_Buffer.Rewards[t2] = ?7  ;
              let t3 = Last_Time_Pick_Up_Key          in  Replay_Buffer.Rewards[t3] = ?8  ;
              let t4 = Last_Time_Drop_Ball            in  Replay_Buffer.Rewards[t4] = ?9  ;
              let t5 = Last_Time_Drop_Wrong_Ball      in  Replay_Buffer.Rewards[t5] = ?10 ;
```

Figure 10: The diagram of the SRM designed for the ObstructedMaze task.

door at the present time step, the SRM awards the following agent behaviors: opening a box, picking up a ball, picking up a key, dropping a ball, dropping a key through a proposition $?_5 + \#\texttt{OPENBOX} \times (?_3 - ?_6) + \#\texttt{PICKUPBALL} \times (?_3 - ?_7) + \ldots > 0$ which bounds the number of times that the those behaviors are awarded. The counters $\#\texttt{OPENBOX}, \#\texttt{PICKUPBALL}\ldots$ only count the number of respectively concerned behaviors between two successive door unlocking behaviors by resetting themselves to 0 once the agent unlocks a door. Thus far the design pattern is still similar to that adopted in the KeyCorridor tasks. What makes a difference here is that we assume the SRM to have access to the replay buffer of the agent policy, annotated as `Replay_Buffer`. Suppose that in some time step `t` the agent unlocks a `C` colored door located at coordinate `X`, the SRM locates the last time step when the agent unlocked a door. Then it reassigns the rewards to 0 for all the opening a box, picking up a ball, picking up a key, dropping a ball, dropping a key behaviors stored in `Replay_Buffer` ever since that last door locking time step till the present time step. Then it identifies the time steps of four milestone behaviors based on the following human insights represented in first order logic: a) $(\texttt{Unlock\_C\_Colored\_Door@t}) \mapsto \exists \texttt{t}_1 < \texttt{t}.(\texttt{Open\_Box@t}_1) \wedge (\texttt{Find\_C\_Colored\_Key@t}_1 + 1)$, i.e., in time step $\texttt{t}_1$ the agent opened the box that contains the key for this C colored door; b) $(\texttt{Unlock\_X\_Located\_Door@t}) \mapsto \exists \texttt{t}_2 < \texttt{t}.\forall \texttt{t}_2' > \texttt{t}_2.(\texttt{Pick\_Up\_X\_Located\_Ball@t}_2) \wedge (\neg \texttt{Pick\_Up\_X\_Located\_Ball@t}_2')$, i.e., in time step $\texttt{t}_2$ the agent picked up the ball obstructing this door at position X for the last time; c) $(\texttt{Unlock\_C\_Colored\_Door@t}) \mapsto \exists \texttt{t}_3 < \texttt{t}.\forall \texttt{t}_3' \in [\texttt{t}_3, \texttt{t}].(\texttt{Pick\_Up\_C\_Colored\_Key@t}_3) \wedge (\neg \texttt{Pick\_Up\_C\_Colored\_Key@t}_3')$, i.e., in time step $\texttt{t}_3$ the agent picked up the key for this C colored door for the last time; d) $(\texttt{Unlock\_Door@t}) \mapsto \exists \texttt{t}_4 < \texttt{t}.\forall \texttt{t}_4' \in [\texttt{t}_4, \texttt{t}].(\texttt{Drop\_Ball@t}_4) \wedge (\neg \texttt{Drop\_Ball@t}_4')$, i.e., in time step $\texttt{t}_4$ the agent dropped a ball for the last time. After identifying those milestone time steps, the SRM rewards the behaviors at the corresponding time steps. The intuition behind such design pattern is that the reward function simply encourages all those behaviors if it is unclear what outcome those behavior will lead to; once the agent unlocks a door, the reward function is able to identify the milestone behaviors that are most closely related to the door unlocking outcome. Then we show the atomic predicates in the symbolic constraint for this task in Table.3. The final symbolic constraint is $c = \bigwedge_{i=1}^{12} \mu_i$.

Note that the stored reward is not to be confused with the reward output at the present time. The syntax of sequencing in the `Hindsignt` code block depends on the language of the r term specified in the background theory. For the other two SRMs annotated by SRM2 and SRM3 as mentioned earlier, we remove the `Hindsight` block. Especially, in SRM2, we restrict that door unlocking and door opening behaviors are rewarded if only the accumulated rewards gained from those two behaviors do not exceed $?_2$. Otherwise, none of the behaviors correlated with the self-looping transitions at state "`Before_Seeing_the_Target`" in Fig.10 will ever be rewarded. A possible reason for the policies trained by SRM1 do not generalize well in larger environment is that due to the hindsight reward modification, the reward output is too sparse in the large environment for the agent to learn. As shown by the experimental results of SRM2 and SRM3, once the `Hindsight` block is removed, the training performance in large environment is improved.

Properties	(Retaional) Predicates	(Non-Relational) Predicates
$[\mu_1]$Reward picking up target	$\bigwedge_{\texttt{id}=2}^{12}(?_{\texttt{id}} \leq ?_1)$	$?_1 \geq 0$
$[\mu_2]$Reward finding target	$?_2 \geq ?_4 + ?_5 - 2?_3$	$?_2 \geq 0$
$[\mu_3]$Reward opening door	$?_3 \leq 0$	$?_3 \leq 0$
$[\mu_4]$Reward opening door	$?_4 \geq 0$	$?_4 \geq 0$
$[\mu_5]$Reward unlocking door	$?_5 \geq \sum_{\texttt{id}=6}^{10} ?_{\texttt{id}}$	$?_5 \geq 0$
$[\mu_6]$Penalize meaningless move	$?_3 \leq 0$	$?_3 \geq 0$
$[\mu_7]$Penalize picking up used key	$?_{11} + ?_{12} \leq 0$	$?_{11} \leq 0$
$[\mu_8]$Reward opening box	$?_6 \geq 0$	$?_6 \geq 0$
$[\mu_9]$Reward picking up ball	$?_7 \geq 0$	$?_7 \geq 0$
$[\mu_10]$Reward picking up key	$?_8 \geq 0$	$?_8 \geq 0$
$[\mu_{11}]$Reward dropping ball	$?_9 \geq 0$	$?_9 \geq 0$
$[\mu_{12}]$Reward dropping used key	$?_{12} \geq 0$	$?_{12} \geq 0$

Table 3: The correspondence between properties and predicates for the SRM of ObstructedMaze task in Fig.10

A.3 Training details

- **Training Overhead**. We note that all the designed SRMs require checking hindsight experiences, or maintaining memory or other expensive procedures. However, line 5 of Algorithm 1 requires running all K candidate programs on all m sampled trajectories, which may incur a substantial overhead during training. Our solution is that, before sampling any program as in line 5 of Algorithm 1, we evaluate the result of $[\![\mathcal{L}]\!](\tau_{A,i})$, which keeps holes **?** unassigned, for all the m trajectories. By doing this, we only need to execute the expensive procedures that do not involve the holes once, such as the counter #OPENDOOR and the reward modification steps in the Hindsight block in Fig.10. Then we use q_φ to sample K hole assignments $\{\mathbf{h_k}\}_{k=1}^K$ from $\mathbf{H}$ and feed them to $\{[\![\mathcal{L}]\!](\tau_{A,i})\}_{i=1}^m$ to obtain $\{\{[\![l_k := \mathcal{L}[\mathbf{h}_k/\mathbf{?}]]\!](\tau_{A,i})\}_{i=1}^m\}_{k=1}^K$. By replacing line 2 and line 5 with those two steps in Algorithm 1, we significantly reduce the overhead.

- **Supervised Learning Loss**. In Algorithm 1, a supervised learning objective J_{cons} is used to penalize any sampled hole assignment for not satisfying the symbolic constraint. In practice, since our sampler q_φ directly outputs the mean and log-variance of a multivariate Gaussian distribution for the candidate hole assignments, we directly evaluate the satisfaction of the mean. Besides, as mentioned earlier, in our experiments we only consider symbolic constraint as a conjunction of atomic predicates, e.g., $c = \wedge_{i=1}^n \mu_i$ with each μ_i only concerning linear combinations of the holes, we reformulated each μ_i into a form $u_i(\mathbf{?}) \leq 0$ where u_i is some linear function of the holes **?**. We make sure that $(u_i(\mathbf{h}) \leq 0) \leftrightarrow ([\![\mu_i]\!](\mathbf{h}) = \top)$ for any hole assignment $\mathbf{h}$. After calculating each $u_i(\mathbf{h})$, which is now a real number, we let $J_{cons}(q_\varphi)$ be a negative binary cross-entropy loss for $Sigmoid(ReLU([u_i(\mathbf{h}), \ldots, u_n(\mathbf{h})]^T)))$ with 0 being the ground truth. This loss penalizes any $\mathbf{h}$ that makes $u_i(\mathbf{h}) > 0$. In this way $J_{cons}(q_\varphi)$ is differentiable w.r.t φ. Besides, we retain the entropy term $\mathcal{H}(q_\varphi)$ extracted from the KL-divergence to regularize the variance output by q_φ.

  Network Architectures. Algorithm 1 involves an agent policy π_ϕ, a neural reward function f_θ and a sampler q_φ. Each of the three is composed of one or more neural networks.

  - **Agent policy** π_ϕ. We prepare two versions of actor-critic networks, a CNN version and an LSTM version. For the CNN version, we adopt the actor-critic network from the off-the-shelf implementation of AGAC Flet-Berliac et al. (2021). It has 3 convolutional layers each with 32 filters, 3×3 kernel size, and a stride of 2. A diagram of the CNN layers can be found in Flet-Berliac et al. (2021). For the LSTM version, we concatenate 3 identically configured convolutional layers with a LSTM cell of 32-size state vector. The LSTM cell is then followed by multiple fully connected layers each to simulate the policy, value and advantage functions. While AGAC contains other

Parameter	**Value**
# Epochs	4
# minibatches (π_ϕ)	8
# batch size (f_θ, q_φ)	128
# frames stacked (CNN π_ϕ)	4
# reccurence (LSTM π_ϕ)	1
# recurrence (f_θ)	8
Discount factor γ	0.99
GAE parameter λ	0.95
PPO clipping parameter ϵ	0.2
K	16
α	0.001
β	0.0003
η	1.e8
$Entropy$	1.e-2

Table 4: Hyperparameters used in the training processes

components Flet-Berliac et al. (2021), the PPO agent solely consists of the actor-critic networks.

– **Neural reward function** f_θ. The network is recurrent. It has 3 convolutional layers each with 16, 32 and 64 filters, 2×2 kernel size and a stride of 1. The last convolutional layer is concatenated with an LSTM cell of which the state vector has a size of 128. The LSTM cell is then followed by a 3-layer fully connected network where each hidden layer is of size 64. Between each hidden layer we use two $tanh$ functions and one Sigmoid function as the activation functions. The output of the Sigmoid function is the logit for each action in the action space $\mathcal{A}$. Finally, given an action in a state, we use softmax and a Categorical distribution output the log-likelihood for the given action as the reward.

– **Sampler** q_φ. The input to q_φ is a constant $[1, \ldots, 1]^T$ of size 20. The sampler is a fully-connected network with 2 hidden layers of size 64. The activation functions are both $tanh$. Suppose that there are $|\mathbf{?}|$ holes in the SRM. Then the output of q_φ is a vector of size no less than $2|\mathbf{?}|$. The $|\mathbf{?}|$ most and the $|\mathbf{?}|$ least significant elements in the output vector will respectively be used as the mean of the Gaussian and constitute a diagonal log-variance matrix. Besides, we let q_φ to output a value as the constant reward for the dummy transitions. While we still return 0 instead of this constant as the reward to the agent, we subtract every sampled $\mathbf{h}$ with this constant to compute $[\![l]\!](\tau)$ for J_{soft}. This subtraction simulates normalizing $[\![l]\!](\tau)$ in order to match the outputs of f_θ, which, as mentioned earlier, is always non-positive in order to match $\log \pi_E$.

- **Hyperparameters**. Most of the hyperparameters that appear in Algorithm 1 are summarized as in Table.4. All hyperparameters relevant to AGAC are identical as those in Flet-Berliac et al. (2021) although we do not present all of them in Table.4 in order to avoid confusion. The hyperparameter η is made large to heavily penalize q_φ when its output violates the symbolic constraint c. Besides, we add an entropy term $\mathcal{H}(q_\varphi)$ multiplied by $1e-2$ in addition to J_{soft} and J_{con} to regularize the variance output by q_φ. The item $Entropy$ refers to the multiplier for the entropy term $\mathcal{H}(q_\varphi)$ as introduced earlier.

A.4 Derivation Of the Objective Functions

First, we derive the lower-bound of $\log p(0_A, 1_E|\pi_A, E, l)$ in Eq.5 as follows.

$$
\begin{aligned}
& \log p(0_A, 1_E|\pi_A, E, l) \\
= \; & \log \sum_{\tau_A, \tau_E} p(\tau_A|\pi_A)p(\tau_E|E) \iint\limits_{f_{\tau_A}, f_{\tau_E}} p(0_A|\tau_A; \pi_A, f_{\tau_A})p(1_E|\tau_E; \pi_A, f_{\tau_E}) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad p(f_{\tau_E}|\tau_E; l)p(f_{\tau_A}|\tau_A; l) \\
\geq \; & \mathop{\mathbb{E}}_{\substack{\tau_A \sim \pi_A \\ \tau_E \sim E}} \Big[\log \iint\limits_{f_{\tau_A}, f_{\tau_E}} p(0_A|\tau_A; \pi_A, f_{\tau_A})p(1_E|\tau_E; \pi_A, f_{\tau_E})p(f_{\tau_E}|\tau_E; l)p(f_{\tau_A}|\tau_A; l) \Big] \\
= \; & \mathop{\mathbb{E}}_{\substack{\tau_A \sim \pi_A \\ \tau_E \sim E}} \Big[\log \iint\limits_{f_{\tau_A}, f_{\tau_E}} p(0_A|\tau_A; \pi_A, f_{\tau_A})p(1_E|\tau_E; \pi_A, f_{\tau_E})p(f_{\tau_A}|\tau_A; l)p(f_{\tau_E}|\tau_E; l) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \frac{p(f_{\tau_A}|\tau_A; f)p(f_{\tau_E}|\tau_E; f)}{p(f_{\tau_A}|\tau_A; f)p(f_{\tau_E}|\tau_E; f)} \Big] \\
\geq \; & \max_f \mathop{\mathbb{E}}_{\substack{\tau_A \sim \pi_A \\ \tau_E \sim E}} \Big[\mathop{\mathbb{E}}_{\substack{f_{\tau_A} \sim p(\cdot|\tau_A; f) \\ f_{\tau_E} \sim p(\cdot|\tau_E; f))}} \Big(\log p(0_A|\tau_A; \pi_A, f_{\tau_A})p(1_E|\tau_E; \pi_A, f_{\tau_E}) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \frac{p(f_{\tau_A}|\tau_A; l)p(f_{\tau_E}|\tau_E; l)}{p(f_{\tau_A}|\tau_A; f)p(f_{\tau_E}|\tau_E; f)} \Big) \Big] \\
= \; & \max_f \mathop{\mathbb{E}}_{\epsilon \sim \mathcal{N}} \Big[J_{adv}(D_\epsilon) \Big] - \mathop{\mathbb{E}}_{\tau \sim \pi_A, E} \Big[D_{KL}(p_{f(\tau)} || p_{l(\tau)}) \Big]
\end{aligned}
$$

We justify the usage of the stochastic version $\mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)}[J_{adv}(D_\epsilon)]$, rather than the conventional generative adversarial objective $J_{adv}(D)$, by showing that one of the saddle point of $\min_{\pi_A} \max_f \mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)}[J_{adv}(D_\epsilon)]$ is attained when $f \equiv \log \pi_E \equiv \log \pi_A$ where we write π_E in proxy of E by assuming that the distribution of state-action pairs satisfies $p(s, a|\pi_E) \equiv p(s, a|E)$.

Theorem 1. *Given a* π_E, $\min_{\pi_A} \max_f \mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)} \big\{ \mathbb{E}_{(s,a) \sim \pi_E} \big[\log D_\epsilon(s, a) \big] + \mathbb{E}_{(s,a) \sim \pi_A} \big[\log(1 - D_\epsilon(s, a) \big] \big\}$, *where* $D_\epsilon(s, a) := \frac{\exp(f(s,a)+\epsilon)}{\exp(f(s,a)+\epsilon)+\pi_A(a|s))}$, *is optimal when* $f \equiv \log \pi_E \equiv \log \pi_A$.

Proof. Firstly, we consider optimizing f under the condition of $\pi_A \equiv \pi_E$. Inspired by the proof of optimality condition of Generative Adversarial Nets in Goodfellow et al. (2014), we introduce two variables $x_{s,a} = y_{s,a} \in (0, 1]$ to simulate $p(s, a|\pi_A) = p(s, a|\pi_E)$ for any $s, a \in \mathcal{S} \times \mathcal{A}$. Then we prove that $\mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)} \big[x_{s,a} \log \frac{\hat{x}_{s,a} \cdot \exp(\epsilon)}{\hat{x}_{s,a} \cdot \exp(\epsilon) + y_{s,a}} + y_{s,a} \log \frac{y_{s,a}}{\hat{x}_{s,a} \cdot \exp(\epsilon) + y_{s,a}} \big]$ as a function of $\hat{x}_{s,a}$ has x stationary point at $\hat{x}_{s,a} = x_{s,a}$ by computing its gradient w.r.t $\hat{x}_{s,a}$ as follows.

$$
\begin{aligned}
& \nabla_{\hat{x}_{s,a}} \mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)} \big[x_{s,a} \log \frac{\hat{x}_{s,a} \cdot \exp(\epsilon)}{\hat{x}_{s,a} \cdot \exp(\epsilon) + y_{s,a}} + y_{s,a} \log \frac{y_{s,a}}{\hat{x}_{s,a} \cdot \exp(\epsilon) + y_{s,a}} \big] \\
= \; & \mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)} \big[x_{s,a} \cdot \frac{\hat{x}_{s,a} \cdot \exp(\epsilon) + y_{s,a}}{\hat{x}_{s,a} \cdot \exp(\epsilon)} \cdot \frac{\exp(\epsilon)(\hat{x}_{s,a} \cdot \exp(\epsilon) + y_{s,a}) - \hat{x}_{s,a} \exp(2\epsilon)}{(\hat{x}_{s,a} \cdot \exp(\epsilon) + y_{s,a})^2} + \\
& \qquad\qquad y_{s,a} \cdot \frac{\hat{x}_{s,a} \cdot \exp(\epsilon) + y_{s,a}}{y_{s,a}} \cdot \frac{-y_{s,a} \exp(\epsilon)}{(\hat{x}_{s,a} \cdot \exp(\epsilon) + y_{s,a})^2} \big] \\
= \; & \mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)} \big[\frac{x_{s,a} y_{s,a} / \hat{x}_{s,a} - y_{s,a} \exp(\epsilon)}{\hat{x}_{s,a} \cdot \exp(\epsilon) + y_{s,a}} \big] \qquad (7)
\end{aligned}
$$

When $\hat{x}_{s,a} = x_{s,a} = y_{s,a}$, Eq.7 equals $\mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)} \big[\frac{1-\exp(\epsilon)}{1+\exp(\epsilon)} \big]$. Note that the probabilities of sampling ϵ and $-\epsilon$ equal each other, and $\frac{1-\exp(\epsilon)}{1+\exp(\epsilon)} = -\frac{1-\exp(-\epsilon)}{1+\exp(-\epsilon)}$. Hence, $\mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)} \big[\frac{1-\exp(\epsilon)}{1+\exp(\epsilon)} \big] = 0$. It can trivially proved that the gradient of Eq.7 w.r.t $\hat{x}$ is non-positive. Therefore, $f \equiv \log \pi_E$ is a local maximum.

Next, we consider optimizing π_A under the condition of $f \equiv \log \pi_E$. We denote $p(s,a|\pi_E)$ and $p(s,a|\pi_A)$ for any $(s,a) \in \mathcal{S} \times \mathcal{A}$ as $x_{s,a}, y_{s,a} \in (0,1]$ for short. Then we show that $\mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)}\big[\sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} x_{s,a} \log \frac{x_{s,a}\cdot\exp(\epsilon)}{x_{s,a}\cdot\exp(\epsilon)+y_{s,a}} + \frac{y_{s,a}}{x_{s,a}+y_{s,a}} \log \frac{y_{s,a}}{x_{s,a}\cdot\exp(\epsilon)+y_{s,a}}\big]$ as a function of $\{y_{s,a}|(s,a) \in \mathcal{S}\times\mathcal{A}\}$ s.t. $\sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} y_{s,a} = 1$ has a stationary point at $x_{s,a} \equiv y_{s,a}$ by computing the gradient of the Lagrangian of this constrained function as follows.

$$
\begin{aligned}
L &= \mathop{\mathbb{E}}_{\epsilon\sim\mathcal{N}(0,1)}\Big[\sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} x_{s,a} \log \frac{x_{s,a}\cdot\exp(\epsilon)}{x_{s,a}\cdot\exp(\epsilon)+y_{s,a}} + y_{s,a} \log \frac{y_{s,a}}{x_{s,a}\cdot\exp(\epsilon)+y_{s,a}}\Big] - \\
&\qquad\qquad \lambda\Big(\sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} y_{s,a} - 1\Big) \\
\nabla_{y_{s,a}} L &= \mathop{\mathbb{E}}_{\epsilon\sim\mathcal{N}(0,1)}\Big[\log \frac{y_{s,a}}{x_{s,a}\cdot\exp(\epsilon)+y_{s,a}} + \frac{x_{s,a}(\exp(\epsilon)-1)}{x_{s,a}\cdot\exp(\epsilon)+y_{s,a}}\Big] - \lambda = 0 \\
&\Rightarrow \quad \forall (s,a) \in \mathcal{S}\times\mathcal{A}.\ \mathop{\mathbb{E}}_{\epsilon\sim\mathcal{N}(0,1)}\Big[\log \frac{y_{s,a}}{x_{s,a}\cdot\exp(\epsilon)+y_{s,a}} + \frac{x_{s,a}(\exp(\epsilon)-1)}{x_{s,a}\cdot\exp(\epsilon)+y_{s,a}}\Big] = \lambda \quad (8) \\
\nabla_\lambda L &= \sum_{(s,a)\in\mathcal{S}\times\mathcal{A}} y_{s,a} - 1 = 0 \quad (9)
\end{aligned}
$$

Suppose that $\lambda = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}[\log \frac{1}{1+\exp(\epsilon)}]$ and $x_{s,a} = y_{s,a}$ holds for any $(s,a) \in \mathcal{S}\times\mathcal{A}$. Then both Eq.8 and Eq.9 hold. Hence, $x_{s,a} \equiv y_{s,a}$ is a stationary point. It is also trivially provable that the gradient of Eq.8 w.r.t $y_{s,a}$ is non-negative. Therefore, $\pi_A \equiv \pi_E$ is a local minimum. In conclusion, $f \equiv \log \pi_E \equiv \log \pi_A$ is a saddle point. □

We derive the lower-bound of the ELBO in Eq.6 as follows.

$$
\begin{aligned}
& ELBO(q) = D_{KL}\Big[q(l)||p(l)\Big] + \mathop{\mathbb{E}}_{l\sim q}\Big[\log p(0_A, 1_E|\pi_A, E, l)\Big] \\
= \ & \mathop{\mathbb{E}}_{l\sim q}\Big\{\log \mathop{\mathbb{E}}_{\substack{\tau_A\sim\pi_A \\ \tau_E\sim E}}\Big[\iint_{f_{\tau_A}, f_{\tau_E}} p(0_A|\tau_A;\pi_A,f_{\tau_A}) p(1_E|\tau_E;\pi_A,f_{\tau_E}) p(f_{\tau_E}|\tau_E;l) p(f_{\tau_A}|\tau_A;l)\Big]\Big\} - \\
& \qquad\qquad D_{KL}\Big[q(l)||p(l)\Big] \\
\geq \ & \max_f \mathop{\mathbb{E}}_{\epsilon\sim\mathcal{N}}\Big[J_{adv}(D_\epsilon)\Big] - \mathop{\mathbb{E}}_{\substack{l\sim q \\ \tau\sim\pi_A, E}}\Big[D_{KL}(p_{f(\tau)}||p_{l(\tau)})\Big] - D_{KL}\Big[q(l)||p(l)\Big] \\
= \ & J_{soft}(q,f) + J_{con}(q)
\end{aligned}
$$