CONTRASTIVE REINFORCEMENT LEARNING FOR SYMBOLIC PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Symbolic planning is a type of sequential decision-making problem for goal reaching modeled in a symbolic language like Planning Domain Definition Language (PDDL). The mainstream approaches for solving symbolic planning problems are search-based methods. Recently, along with the development of machine learning techniques, there is a growing interest in exploiting learning-based approaches in symbolic planning. In spite of the similarity between symbolic planning and Markov Decision Processes, most learning-based approaches for symbolic planning are supervised (imitation) learning methods. Model-free reinforcement learning methods, which can explore the environment freely, are under-explored in this area. In this paper, we study how to customize contrastive reinforcement learning (CRL) for symbolic planning, which results in a model-free approach for symbolic planning with competitive performance.

1 INTRODUCTION

Symbolic planning is concerned with sequential decision-making problems for goal reaching that are described in a symbolic language like Planning Domain Definition Language (PDDL) (Haslum et al., 2019). In general, symbolic planning has been proved to be $\mathbb{PSPACE}$-hard (Bylander, 1994). Real-world applications of symbolic planning include Mars exploration rover missions (Ai-Chang et al., 2004) and robotics (in particular, task and motion planning (Garrett et al., 2021)) (Liang et al., 2025; 2026; Liu et al., 2026; Silver et al., 2023). Standard approaches for solving planning problems are based on heuristic search algorithms (Bonet & Geffner, 2001; Hoffmann & Nebel, 2001; Helmert & Domshlak, 2009; Pommerening et al., 2014). Recently, a growing number of machine learning-based methods have been proposed to improve the scalability over standard approaches (Toyer et al., 2018; Ståhlberg et al., 2022; 2023; Müller et al., 2026). The majority of these methods are supervised (imitation) learning methods.

Another paradigm for solving sequential decision-making tasks is reinforcement learning (RL) However, RL approaches have so far not been very successful when applied to symbolic planning, especially in the model-free setting (i.e., without knowing the underlying symbolic planning model). Recently, Ståhlberg & Geffner (2026) have proposed an approach in the model-free setting which does not require any offline training, however only performs well on limited domains.

As a step towards exploring model-free RL methods for symbolic planning, in this paper, we study how to adapt contrastive reinforcement learning (CRL) (Eysenbach et al., 2022; Liu et al., 2025) in symbolic planning, which has demonstrated promising performance in solving goal-reaching tasks. The main idea of CRL is to estimate the likelihood $f(s, a, s')$ of reaching a future state s' from the current state s by taking action a and following the current policy. The policy is then optimized to take actions that maximize the likelihood of reaching a commanded goal state. This aligns perfectly with the goal-reaching objective of symbolic planning.

Despite the underlying connection between the objective of CRL and symbolic planning, it is non-trivial to deploy CRL directly. One obstacle is that, as we will show in Sec. 2, the goal in a symbolic planning problem is usually *not* a concrete state of the environment captured by the symbolic model, but a set of binary indicators to indicate whether certain tasks are completed. As a consequence, a concrete goal state of the environment cannot be inferred from the goal specification of a symbolic planning problem, while CRL requires such a state to contrast against.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

There is another less obvious but more severe obstacle to applying CRL, caused by the combinatorial nature of symbolic planning. Intuitively speaking, the exponential number of avaliable actions and states together with the fact that every action only modify a small portion of a state and that reaching a state can be viewed as a joint consequence of many independent action chains makes the estimation of the likelihood $f(s, a, s')$ completely ignore the input action a. We will analyze this phenomenon both theoretically and empirically in Sec. 3 and 4.

Contributions We quantitatively analyze the difficulties of deploying CRL to symbolic planning. The insights gained from this analysis can also be generalized to other RL methods. To address those difficulities, we adapt CRL to estimates the likelihood of completing a task specified in the goal instead of reaching a concrete goal state. We could then extract a policy which leads to completing a commanded task. An entire planning problem can be solved by commanding one task after another that are specified in the goal. This is in line with the notion of unit planning width (Lipovetzky & Geffner, 2012; Bonet & Geffner, 2024). Notably, our adaptation of CRL can be generalized to other goal-conditioned RL methods like Goal-conditioned Behavior Cloning (GCBC) (Ding et al., 2019; Ghosh et al., 2021) (as shown in our experiments) and, unlike most existing learning-based methods for symbolic planning, our approach does not require any supervision and remains *model-free*.

2 BACKGROUND

Symbolic Planning Framework We consider symbolic planning modeled in PDDL. A symbolic planning domain $\mathcal{D}$ is a tuple $(\mathcal{T}, \mathcal{P}, \mathcal{A})$ where $\mathcal{T}$ is a finite set of *types*, $\mathcal{P}$ is a finite set of *predicates*, and $\mathcal{A}$ is a finite set of *action schemas*. Here, a predicate is defined similarly as in First-Order Logic. Concretely, a predicate $\mathtt{p}$ is of the form $p(x_1|_{t_1}, x_2|_{t_2}, \ldots, x_n|_{t_n})$ where $n \geq 0$ is an integer, x_i is a *variable* with type $t_i \in \mathcal{T}$ for all $1 \leq i \leq n$, and p is the name of the predicate. A variable with a certain type can be instantiated to any object of the same type. We say that each x_i is a *parameter* of $\mathtt{p}$. Intuitively, the predicate $\mathtt{p}$ captures a relation between objects of types $t_1, \ldots, t_n$. An action schema $\mathtt{a}$ is characterized by its *precondition*, *positive effect*, and *negative effect*, each of which is a set of predicates, written $\mathtt{a} = (\mathtt{pre}[\![\mathtt{a}]\!], \mathtt{eff}^+[\![\mathtt{a}]\!], \mathtt{eff}^-[\![\mathtt{a}]\!])$. It describes how it can manipulate different objects (its effects) and under what conditions (its precondition) it can be applied.

A planning instance $(\mathcal{D}, T)$ consists of a planning domain $\mathcal{D}$ and a planning task $T = (\mathcal{O}, s_I, g)$ where $\mathcal{O}$ is a finite set of *objects* each of which is associated with a type, s_I is the *initial state*, and g is the *goal*. A predicate in the domain can be instantiated to an *atom* by substituting its parameters with objects in $\mathcal{O}$ of the same type. Similarly, an action schema can be instantiated to an *action* by instantiating the predicates in its precondition and effects. Let a be an instantiated action. We abuse the notation to let $\mathtt{pre}[\![a]\!]$, $\mathtt{eff}^+[\![a]\!]$, and $\mathtt{eff}^-[\![a]\!]$ denote the instantiated precondition, positive effect, and negative effect of a. A state, e.g., the initial state s_I, in a planning instance is a set of *atoms*. An action a is said to be *applicable* in a state s if $\mathtt{pre}[\![a]\!] \subseteq s$, i.e., the precondition of a is satisfied in s. Applying an action a in a state s results in a new state $s' = (s \setminus \mathtt{eff}^-[\![a]\!]) \cup \mathtt{eff}^+[\![a]\!]$, that is, the atoms in a's negative effect will be removed from s while those in the positive effect will be added. The goal g in a planning instance is again a set of atoms. A *solution* to a planning instance is a sequence of actions called a *plan* which turns the initial state s_I into another state s^* such that $g \subseteq s^*$, i.e., the goal is satisfied in s^*.

Example Fig. 1 illustrates a planning instance under the Mars Rovers domain.[1] Here, *rovers* need to move to different *waypoints* to collect *rock samples*, *soil samples*, and *images* of certain *objects*. An image of an object can only be taken from a waypoint where the object is visible. The rovers need to transmit the analyzed data about the samples and images to *landers*. A predicate `at[r, w]` is used to indicate the rover's position. The variable `r` can be instantiated to a concrete rover (i.e., it is of *type* `rover`), and similarly, `w` can be instantiated to a concrete waypoint. In the same way, there are predicates `at_soil_sample`, `at_rock_sample` and `visable_from` which indicate the position of soil samples, rock samples, and which objects are visible from which waypoints. Rovers can move to different waypoints via the action schema `navigate[r, x, y]` with r being the variable that can be instantiated to a rover, and x and y being the variables that can be instantiated to concrete waypoints. The goal is to transmit all the analyzed data to the landers. Whether an analyzed sample (or an image) is transmitted is indicated by a predicate, as shown in

[1] For simplicity, some details of the actual domain are omitted here.

Fig. 1. Notice that the conditions specified in the goal have nothing to do with a concrete state, which is the key obstacle to be addressed when adapting CRL to this setting.

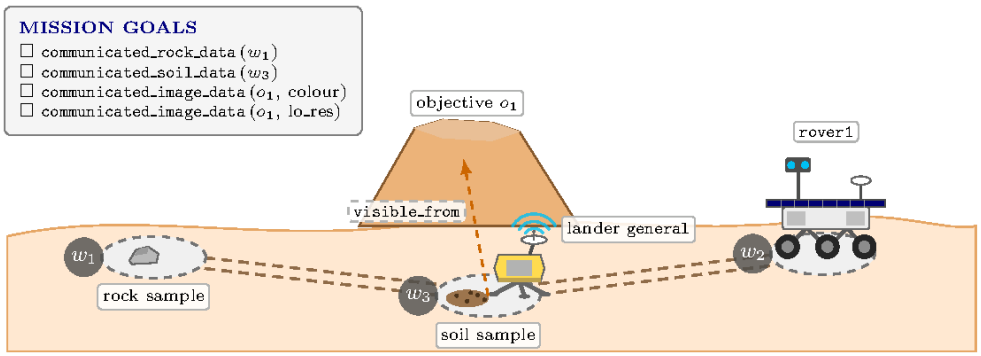

Figure 1: A simple example illustrating a symbolic planning instance in the Mars Rovers domain. Notably, the goal (mission) is specified in terms of atoms that are irrelevant to an actual state.

Contrastive Reinforcement Learning. Contrastive reinforcement learning (CRL) (Eysenbach et al., 2022) addresses goal-reaching Markov decision processes (MDP) with a state space $\mathcal{S}$, an action space $\mathcal{A}$, the dynamics $\mathbb{P}(s_{t+1}|s_t, a_t)$, an initial state distribution $\mathbb{P}_I$, a goal state distribution $\mathbb{P}_G$, and a discount factor γ. The objective is to find a policy such that

$$\pi = \arg\max_{\pi} \mathbb{E}_{s_g \sim \mathbb{P}_G} \left[(1-\gamma) \sum_{t=0}^{\infty} \gamma^t \mathbb{P}_\pi(s_{t+1} = s_g) \right]$$

where $\mathbb{P}_\pi(s_{t+1} = s)$ is the probability of visiting the state s at timestep $t+1$ by following the policy π from the initial state distribution $\mathbb{P}_I$. The core of CRL is to learn a critic function (Q-function) $f(s, a, s_f)$, which estimates the likelihood of reaching a future state s_f from a current state s by taking an action a. One way to estimate this critic function is to parameterize it as an inner product $f(s, a, s_f) = \phi^T(s, a)\, \psi(s_f)$, where $\phi(s, a)$ and $\psi(s_f)$ are learned representations of the state-action pair (s, a) and the future state s_f, respectively. Both are learned via contrastive learning. For a state-action pair (s, a), a positive example is a future state s_f^+ sampled according to the discounted state occupancy $\mathbb{P}_\pi^f(s_f^+|s, a) = (1-\gamma) \sum_{t=0}^{\infty} \gamma^t \mathbb{P}_\pi(s_{t+1} = s_f^+|s, a)$ with respect to the policy π, and a negative example is a state s_f^- sampled from the marginal distribution $\mathbb{P}_f$ of future states. We then maximize the NCE-binary objective:

$$\mathbb{E}_{(s,a)\sim\mathbb{P}_{(s,a)}, s_f^+ \sim \mathbb{P}_\pi^f(\cdot|s,a), s_f^- \sim \mathbb{P}_f} \left[\log \sigma(f(s, a, s_f^+)) + \log(1 - \sigma(f(s, a, s_f^-))) \right] \tag{1}$$

where $\mathbb{P}_{(s,a)}$ is the marginal distribution of state-action pairs. Practically, we sample an anchor state-action pair (s, a) from a replay buffer. A future state s_t (a positive example) is sampled from the policy trajectory starting from the anchor by sampling t from the geometric distribution $\mathrm{Geom}(1-\gamma)$. Negative examples are future states sampled in the same way from other trajectories. The policy is then updated by maximizing the likelihood of reaching goal states:

$$\max_{\pi} \mathbb{E}_{s\sim\mathbb{P}_s, s_g \sim \mathbb{P}_G, a\sim\pi(\cdot|s,s_g)} \left[f(s, a, s_g) \right]$$

Following the result by Eysenbach et al. (2022), the fixed point of the critic is

$$f^*(s, a, s_g) = \log \mathbb{P}_\pi^f(s_g|s, a) - \log \mathbb{P}_G(s_g) \tag{2}$$

3 CRL FOR SYMBOLIC PLANNING

Adapting CRL to planning is nontrivial for two reasons. First, as explained previously, goals in symbolic planning are *not* concrete states, so CRL cannot be applied directly. The second is less obvious but more severe. The combinatorial nature of symbolic planning causes the learned critic to be dominated by an action-irrelevant, so that the action-dependent signal it is meant to capture is overwhelmed by training noise.

Quantitative Anlysis Eq. 2 can be rewritten as:

$$f^*(s, a, s_g) = \underbrace{(\log \mathbb{P}^f_\pi(s_g|s, a) - \log \mathbb{P}^f_\pi(s_g|s))}_{\text{action-relevant term}} + \underbrace{(\log \mathbb{P}^f_\pi(s_g|s) - \log \mathbb{P}_G(s_g))}_{\text{action-irrelevant term}} \tag{3}$$

where $\mathbb{P}^f_\pi(s_g|s)$ is the probability that by starting from state s and following the policy π, the state s_g can be visited in the future. Only the action-relevant term is used by the policy for selecting good actions. In practice, the critic will tend to fit to the action-irrelevant term because it is usually much larger than the first term. This frequently occurs in symbolic planning. Given a symbolic planning instance with M instantiated atoms, the entire reachable state space of the planning instance could consist of up to 2^M different states while an action only modifies a small portion of the state. Consequently, the ratio between the action-relevant and action-irrelevant terms is $\mathcal{O}(M/2^M)$, making the former term become negligible. This combinatorial nature along with the following proposition make critic learning extremely challenging in symbolic planning.

Proposition 1. *Let π be a deterministic policy in a deterministic MDP with a discount factor γ, s_g a fixed goal state, and f^* the fixed point of the binary NCE loss. For simplicity, assume that the policy π will stay in s_g once it is reached. Given a state s and two actions a^* and a' such that s^* and s' are the successor states of applying a^* and a' in s, respectively, if π will reach s_g from s^* in t_1 steps and from s' in t_2 steps, then, we have that*

$$f^*(s, a^*, s_g) - f^*(s, a', s_g) = (t_2 - t_1)\log(1/\gamma)$$

Proof. See Appendix A for the proof. □

Notably, the premise of the proposition matches the setting of symbolic planning. Intuitively, it says that the advantage of taking a good action over a bad one that can delay reaching a goal state by d steps is only $d\log(1/\gamma)$. For many symbolic domains, the result of applying an action can be reversed by taking another few more actions. In these domains, Prop. 1 means that the advantage of taking a good action over a bad one is negligible. So, the policy extracted from a learned critic is very sensitive to training noise. To illustrate this, consider two actions a^* and a' in a state s where a^* is a good action and a' is a bad one with respect to a goal state s_g. Ideally, $f^*(s, a^*, s_g) > f^*(s, a', s_g)$ with their difference being $d\log(1/\gamma)$. Having this inequality hold is crucial because when the policy is updated, it will select a^* over a' based on this advantage. However, due to training noise, $f(s, a', s_g)$ could be overestimated with an error $\epsilon > d\log(1/\gamma)$. As a consequence, the policy would regard a' as a better action than a^*.

Moreover, unlike typical RL environments with a discrete action space, which usually only have a handful of actions, a symbolic planning instance has an exponential number of instantiated actions, many of them applicable in any given state. The resulting training noise is spread across all these actions and can easily drown out the small advantage $d\log(1/\gamma)$ of taking a good action. We demonstrate this phenomenon in our experiments (Section 4).

Naive Adaption Our first attempt to make CRL viable for planning is to let the critic approximate the likelihood of achieving one single *atom* in the future instead of a concrete state. By doing so, we can 1) command on one single atom specified in the goal, and 2) reduce the order of the magnitude of the action-irrelevant term. Intuitively, we view achieving an atom p as a task to be completed, and $f(s, a, p)$ becomes the likelihood of completing this task by taking the action a in the state s.

This critic is used to construct a policy that can achieve one atom in the goal at a time and complete the entire goal by sequentially achieving all the atoms in the goal. Because symbolic planning has a discrete action space, for any atom p_g we want to achieve and any action s, the policy π is simply:

$$\pi(a|s, p_g) = \arg\max_a f(s, a, p_g)$$

During training, we again sample state-action pairs (s, a) from the replay buffer. Then, for a positive example with respect to a pair (s, a), we run the policy π commanded on an atom $p_g \in g$ starting from (s, a) and sample an achieved atom p^+_f from a future state s_t (i.e., $p^+_f \in s_t$ but $p^+_f \notin s$) in the trajectory with $t \sim \text{Geom}(1-\gamma)$. For a negative example, we sample an atom p^-_f in the same way

Algorithm 1 Customized CRL for Symbolic Planning (One Training Round)

$\left\{\langle s_0^i, a_0^i, \cdots, s_{T_i-1}^i, a_{T_i-1}^i, s_{T_i}^i \rangle\right\}_i \leftarrow$ Sample trajectories following $\pi(\cdot|\cdot, p_g^i)$ for some p_g^i
for all i **do**
  if $\exists\, p' \in s_{T_i}^i$ with the same predicate as p_g^i **then** $q_g^i \leftarrow p'$
  else $q_g^i \leftarrow p_{\texttt{fail}}$ ▷ Set p' as the achieved atom if exists, otherwise, set it as $p_{\texttt{fail}}$
  $w_j^i \leftarrow \gamma^{T_i-j}(1-\gamma)$ for all $0 \le j < T_i$
  $\mathbb{T} \leftarrow \mathbb{T} \cup \left\{(w_j^i, s_j^i, a_j^i, q_g^i)\right\}_{j=0}^{T_i-1}$ ▷ Create positive tuples with weights w_j^i
$\mathbb{T} \leftarrow$ Filter $\mathbb{T}$ to remove irrelevant actions (our refinement 2)
$\mathbb{B} \leftarrow$ Sample a training batch from $\mathbb{T}$ according to the weights w_j^i
▷ q^- sampled uniformly from the atom set with $q^- \neq q^+$ but sharing the same predicate as q^+
Create K negative examples (s, a, q^-) for each $(s, a, q^+) \in \mathbb{B}$
Update the critic f with contrastive learning

but from a different trajectory. The critic $f(s, a, p)$ is then optimized via the same objective as Eq. 1 but with the future state s_f replaced by the future atom p_f.

This approach is not perfect either, because the sampled atom p_f^+ need not be *temporally correlated* with the action a, since a need not be part of the causal chain of actions that eventually made p_f^+ true. Concretely, although a state in a planning domain describes relations among many objects, some of those objects might be independent of each other. A transition between states can be viewed as a joint consequence of multiple independent chain of actions. Within a chain, the actions and the atoms they modify are strongly correlated in time since each atom is made true by an action earlier in the same chain. Across different chains, no such correlation exists. As a result, the sampled atom p_f^+ might be a consequence of an action chain that is independent of the one starting from a.

We propose two refinements which mitigate this issue by ensuring that the atom sampled as a positive example actually possesses a temporal correlation with the previous actions in the trajectory.

Refinement 1 When sampling a trajectory by running the policy π conditioned on an atom p_g to be achieved , we terminate the trajectory at the *first* time step t where an atom p_g' is achieved which shares the same predicate as p_g, i.e., either $p_g' = p_g$ or p_g' and p_g are defined by the same predicate but instantiated to different atoms. Then, p_g' is a positive example for *every* state-action pair (s_i, a_i) in this trajectory. Furthermore, for each positive tuple (s_i, a_i, p_g'), we assign a weight $w_i = \gamma^{t-i}(1-\gamma)$ which decides the chance of it being sampled into a training batch. In other words, in contrast to the standard CRL where a state-action pair serves as an anchor for sampling *one* future state, we now treat the achieved atom p_g' as the anchor, and all state-action pairs in the trajectory have the chance weighted by their weights of entering the training batch.

The intuition behind the above procedure is that when a trajectory is terminated when a certain atom is first achieved, then, actions in this trajectory are more likely to have a temporal correlation with this achieved atom, i.e., those actions contribute to achieving the atom. In particular, the actions that are closer to the achieved atom usually have a stronger correlation and will have a higher chance of being sampled into the training batch due to the geometric weights.

For negative examples, instead of using atoms sampled from different trajectories, for each positive example (s_i, a_i, p_g), we manually create K negative examples by sampling p_g^- from the set of all atoms with $p_g^- \neq p_g$ but sharing the same predicate as p_g. The latter is particularly important because otherwise, the learned critic cannot distinguish between different atoms that share the same predicate. In practice, we set a max horizon H for each sampled trajectory. If there are no atoms that share the same predicate as p_g achieved within H steps, we use a special atom $p_{\texttt{fail}}$ as the positive example to indicate that the policy fails to achieve any meaningful atom. If a trajectory ends at a certain atom p_g, then, $p_{\texttt{fail}}$ is used as a negative example for all state-action pairs in this trajectory.

Refinement 2 The described refinement could already significantly reduce the trainig noise caused by actions that are irrelevant to commanded atoms. We could further improve on this direction by exploiting the state transitions in each sampled trajectory. Concretely, an atom's instantiated param-

eters indicate the objects on which the corresponding relation holds while an aciton's instantiated parameters indicate the objects that it will manipulate. Consequently, it is safe to say that an action a possesses a temporal correlation with an atom p if they share at least one object in their instantiated parameters. For instance, consider a step (s_t, a_t, s_{t+1}) in a sampled trajectory and an action a_j taken in a future step ($j > t$). If there exists an atom $p \in s_{t+1}$ and $p \notin s_t$ which shares some parameters with a_j, we can conclude that a_j is temporally correlated with p and thus with a_t.

Based on this observation, we could approximate the actions in a sampled trajectory that posses a temporal correlation with the commanded atom. Only these relevant actions will be used as positive examples, and the irrelevant ones can be used as negative examples against the achieved atom to help the critic learn the temporal relation. Note that since this improvement only exploits the *observable* state information and action *names*, it does not require any model knowledge from actions' preconditions and effects. Therefore, it is still a model-free approach. The procedure of one training round of the approach is summarized in Alg. 1.

Details We represent an input tuple (s, a, p_g) to the critic as a graph. In learning-based methods for symbolic planning, it is common to represent a state as a graph Ståhlberg et al. (2022). Here, we use a different graph representation and include the action and the commanded atom in the graph. For more details, see Appendix B.

4 Experiments

Why Classical CRL Does Not Work Well? We ran the classical CRL training recipe on a small instance from the Mars Rovers domain. We fixed a *concrete* state as the goal state in which all goal conditions are satisfied and which is reachable from the initial state. This is the setting of single-goal CRL (Liu et al., 2025). The training outcome is disappointing. The policy learned cannot achieve the goal state. In fact, it can barely reach a state where some atoms of the goal are satisfied.

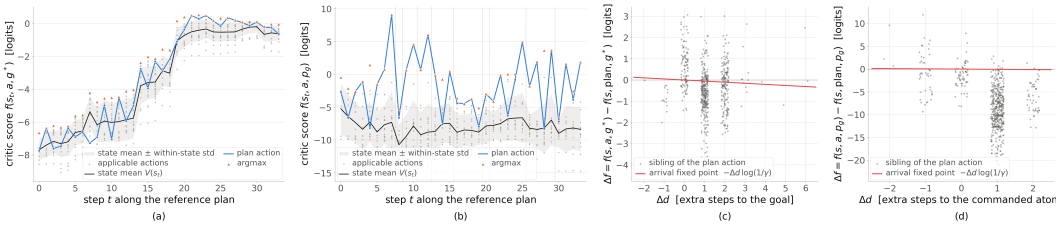

Figure 2: Analysis of the critic learned by the classical CRL training recipe.

To study the behavior of the learned policy more closely, we computed a trajectory which can reach the goal state from the initial state, i.e., a *plan*. For each state s_t in this trajectory, we recorded the critic value $f(s_t, a, s_g)$ for *each applicable action* a in s_t, including the one taken by the plan. These values are plotted in Fig. 2(a). The x-axis is the step index of the trajectory, and the y-axis is the critic value. The dots sitting on the same vertical line are all the applicable actions in that state and their corresponding critic values. The light shadow area indicates the variance of the critic values of all applicable actions in each state. The red triangles are the actions that will be taken by the *policy* (i.e., $\arg\max_a f(s_t, a, s_g)$) at each state. Each dot on the blue line is the action taken by the plan. Each dot on the black line is the *averaged critic value* over all applicable actions in that state. Notably, this averaged value is essentially the value of the *action-irrelevant* term in Eq. 3 for each state. We can observe that 1) the policy barely agrees with the plan, and 2) the variance of averaged critic value along with the trajectory (i.e., the black line) is much larger than the variance of the critic values of actions in each state. The second observation is particularly important because it supports the argument that critic learning is dominated by the action-irrelevant term.

In Fig. 2(c), each dot again represents an applicable action in a state along the plan except that now the x-axis is the additional number of steps needed to reach the goal state s_g when taking this action in the corresponding state, compared against taking the plan's action, and the y-axis is the difference between the critic value of the plan action and the critic value of this action. For instance, a dot sits

at the vertical line of $x = 2$ indicates that it is an applicable action a in a state s_t in the plan and taking this action will need 2 more steps to reach the s_g than taking the plan's action. The y-value of this dot represents $f(s_t, a, s_g) - f(s_t, a^*, s_g)$ where a^* is the plan's action in s_t. The red line is the true action advantages computed according to Prop. 1. What this figure can show is that 1) the variation of actions' advantages produced by the learned critic is much wider than the true ones, showing the vulnerability of the policy misranking actions, and 2) for those actions that need more additional steps (i.e., dots sitting on the vertical lines $x = 1, 2, \cdots$), the action advantages predicted by the learned critic are essentially uniformally distributed on both the postive and negative sides of the ideal advantages ($d \log(1/\gamma)$ by Prop. 1). The latter indicates that the learned critic is not able to distinguish good actions from bad ones.

What If We Command on One Atom? Fig. 2(b) and (d) show the same plots as (a) and (c) except that now the critic is trained with our recipe where the policy is commanded on a single atom. We can observe from Fig. 2(b) that the averaged critic value (the black line) along the trajectory is much more stable than that in Fig. 2(a). In particular, the ratio between the variance of the critic values among actions in each state and the variance of the averaged critic value along the trajectory is now more even, indicating that the action-irrelevant term is no longer dominating the critic learning. For Fig. 2(d), we can see that the advantages of bad actions predicted by the learned critic are now mostly negative, indicating that the learned critic is now able to identify good actions. This aligns with the blue line in Fig. 2(b) where for most states, the plan's action is the one taken by the policy.

Is CRL Better Than Other RL Methods? We first evaluated the performance of our approach in terms of the success rate of achieving commanded *atoms*. We ran the evaluation on domains from the International Planning Competition (IPC) learning track (Taitler et al., 2024). The learning track was initially designed for supervised learning. Hence, each domain contains a set of training instances together with their corresponding trajectories for training and a set of test instances. Notably, computing trajectories (in many cases, optimal trajectories) that can solve those training instances is expensive. The model-free nature of our approach is thus a significant advantage compared against supervised learning methods. We ran our customized CRL approach on 10 instances from the training set with varying instance sizes. The performance of the approach was measured by averaging the success rate of achieving commanded atoms along with training steps over all 10 instances (starting from state-action paris sampled from the replay buffer). As comparisons, we also ran Goal-conditioned behavioral cloning (GCBC) (Ding et al., 2019; Ghosh et al., 2021) and PPO (Schulman et al., 2017). The results are shown in Fig. 3, which are averaged over 3 different seeds.

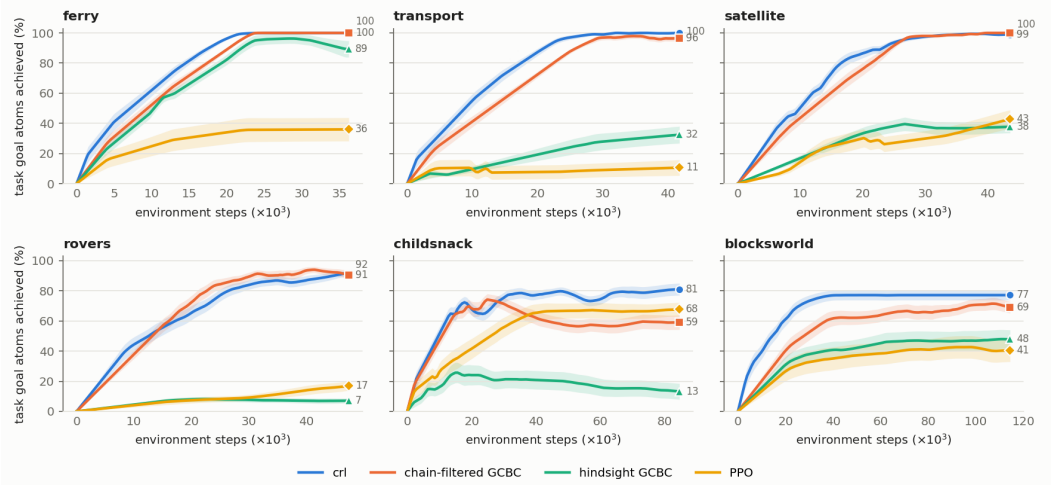

Figure 3: Comparison of CRL with Goal-conditioned Behavioral Cloning (GCBC) and PPO.

How we implement GCBC is identical to how we implement CRL. When sampling a trajectory, the policy is commanded on an atom to be achieved. The trajectory is terminated when the commanded atom is achieved or an atom that shares the same predicate as it is achieved. Then, this achieved atom

Algorithm 2 Deployment of the per-atom policy

$s_0 \leftarrow s_I$, $G^* \leftarrow G$ ▷ Starting from the initial state
while $G^* \neq \emptyset$ **do**
  $p_g \leftarrow$ Sample from G^*
  for $t = 0$ to $T - 1$ **do**
    $s_{t+1} \leftarrow$ The successor state of applying the action selected by $\pi(\cdot|s_t, p_g)$ in s_t
    ▷ If the action undoes some achieved goal atoms, mark them as unachieved again
    $G^* \leftarrow G \setminus s_{t+1}$
    if $p_g \in s_{t+1}$ **then**
      $G^* \leftarrow G^* \setminus \{p_g\}$
      break
  $s_0 \leftarrow s_{t+1}$ ▷ Always keep the newest state

is used as the hindsight relabelled target , named hindsight GCBC in Fig. 3. We also implemented a variant of it where for every sampled trajectory, we only consider the actions that are relevant to the relabelled target (i.e., our Refinement 2), named chain-filtered GCBC in Fig. 3. As can be seen from the plots, having the chain filtering is crucial for both our CRL and GCBC. Despite GCBC is competitive, it still underperforms CRL.

Can the Per-atom Policy Be Actually Deployed? Alg. 2 is a simple approach to deploy a learned per-atom policy to solve a symbolic planning instance: Every time we pick a not achieved goal atom and run the policy commanding on this atom with a fixed budget of steps. Notably, during execution, if some already achieved goal atoms are undone by an action, we mark them as not achieved again. In particular, the execution procedure is like an actor acting continuously. It does *not* feature any common techniques used in search-based methods like loop-elimination and look-ahead.

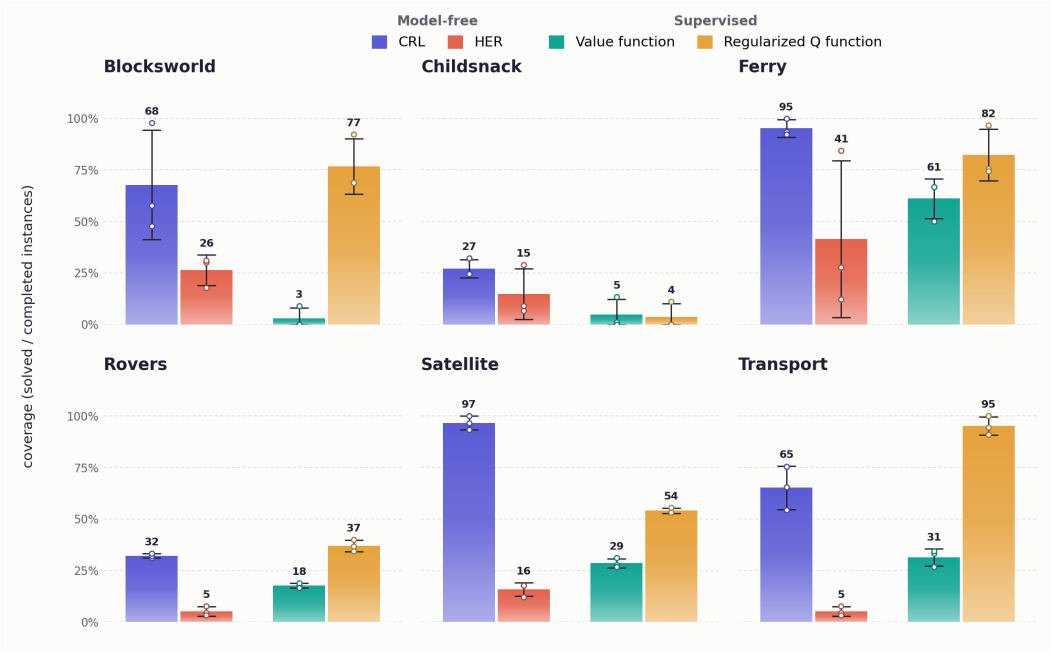

Figure 4: Coverage comparison of the CRL-based approach against other baselines.

We trained the per-atom policy *jointly* over 30 instances from the training set of each domain. By jointly, we mean that during every training round, we sample trajectories from each of the 30 instances with a *shared* policy. We evaluated the trained policy on all the *test* instances of each domain by recording the percentage of instances that can be successfully solved (i.e., the coverage) with Alg. 2 within 15-minutes timeout. Note that the training instances and test instances are disjoint,

and the size of test instances is usually larger than that of training instances. Hence, this evaluation not only measures how well the learned policy can be deployed to solve planning instances, but also how well it can be transferred and generalized to larger unseen instances. We compare against three other methods: 1) The other model-free method which exploits hindsight experience replay (HER) (Ståhlberg & Geffner, 2026). 2) A supervised learning method trained on optimal trajectories which learns a value function (Ståhlberg et al., 2023). 3) Another supervised learning method trained also on optimal trajectories which learns a Q-value function (Müller et al., 2026). Notably, for each state in each optimal trajectory, this method also needs to compute a heuristic value for *every* applicable action in that state as additional training information. All three methods were trained on the same 30 training instances as our method. The latter two were with the corresponding optimal trajectories and heuristic values. They were also evaluated on the same set of test instances with 15-minutes timeout. All evaluations were run with 3 different seeds. The averaged results as well as the variance are shown in Fig. 4. As can be seen, the CRL-based approach outperforms the other model-free method and the value-function-based supervised learning method on all 6 domains. It also shows competitive performance against the Q-function-based supervised learning method.

Limitations The most significant limitation of the CRL-based approach is that it does not work well on domains which have a strong global constraint or in which the order of achieving goal atoms matters, because it is trained to complete only one single atom. For instance, `Childsnack` is one such domain where there are two types of limited resources and two groups of targets that need to consume the resources. One group of target, say group A, can consume both types of resources while the other group, say group B, can only consume one type. A goal atom in this domain is to let one target consume one resource. When the policy is commanded on a goal atom which lets a target in group A consume a resource, it may assign a resource that is demanded by group B to this target. It might then exhaust the resource before all targets in group B can consume it, which makes the goal unreachable. The $p_{\texttt{fail}}$ atom we introduced can prevent this from happening to some extent because during training, we sample starting state-action pairs from the replay buffer. Some states we sampled might already be the ones where the type of resources required by group B is nearly exhausted. When starting from such states, any further action that miss-assigns the resources will lead to $p_{\texttt{fail}}$, and hence, such actions will be penalized. However, such a training signal is quite weak, and only around 27% of the instances can be solved in this domain with our approach.

Another limitation is that when deploying the policy to solve a planning instance, it can result in a long trajectory to solve the goal. For some domains, solving goal atoms in a particular order can lead to a shorter trajectory. One typical example is `Blocksworld` which is to stack blocks into specific towers. A good strategy is to layer blocks from the bottom to the top of a tower, which results in a trajectory of $\mathcal{O}(N)$ steps with N being the number of blocks. In contrast, our deployment procedure can result in a trajectory of $\mathcal{O}(N^2)$ steps because it picks a random order to layer blocks. Notably, there exist some method (Koehler & Hoffmann, 2000) for computing a good order of achieving goal atoms and hence can be combined with our policies. Since we are focusing on the model-free setting while those methods need to read the underlying symbolic model, we omit the discussion here.

5 Conclusion

In this paper, we study how to adapt contrastive reinforcement learning (CRL) in symbolic planning. We analyze, on both theoretical and empirical levels, the challenges of directly applying the original CRL training recipe. To address the issues we identified, instead of contrasting a state-action pair against a concrete future state, we propose to contrast it against a single atom that is achieved in the future and learn a policy that can achieve one single atom. The learned policy can then be deployed to solve a symbolic planning problem by solving each goal atom one by one. Unlike most existing learning-based methods for symbolic planning which are supervised learning methods, our method is model-free. Our empirical evaluation shows that our method outperforms the other model-free methods and has competitive performance with the state-of-the-art supervised learning methods.

In this paper, we only consider *deterministic* symbolic planning problems (i.e., the effects of actions are deterministic). However, it is trivial to extend our method to the non-deterministic setting, i.e., *probabilistic* symbolic planning problems. This remains as a future work. Another future work is to investigate whether we can overcome the obstacles identified in this paper and find a sophisticated approach to learn a policy that can achieve multiple goal atoms at the same time.

AI USE STATEMENT

In this work, we used generative AI tools for supporting qualitative and thematic data analysis, and for implementing code. We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, assist in the writing of proofs, propose or refine hypotheses, and design or provide feedback on research methodology or experiments, assist with translation, clean and reformat dataset, and interpret results. Additionally, we used generative AI tools for creating scientific figures or images. We have reviewed all AI-assisted work. We checked LLM-generated plots and figures for correctness and verified that they accurately represent the data and concepts discussed in the paper. We checked LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Mitchell Ai-Chang, John L. Bresina, Leonard Charest, Adam Chase, Jennifer Cheng-jung Hsu, Ari K. Jónsson, Bob Kanefsky, Paul H. Morris, Kanna Rajan, Jeffrey Yglesias, Brian G. Chafin, William C. Dias, and Pierre F. Maldague. MAPGEN: mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems*, 19:8–12, 2004.

Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

Blai Bonet and Hector Geffner. General policies, subgoal structure, and planning width. *Journal of Artificial Intelligence Research*, 80:475–516, 2024.

Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems, NeurIPS 2019*, pp. 15298–15309, 2019.

Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Ruslan Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems, NeurIPS 2022*, pp. 35603–35620, 2022.

Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1):265–293, 2021. ISSN 2573-5144.

Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Manon Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. In *Proceedings of the 9th International Conference on Learning Representations, ICLR 2021*, 2021.

Patrik Haslum, Christian Muise, Daniele Magazzeni, and Nir Lipovetzky. *An Introduction to the Planning Domain Definition Language*. Springer, 2019.

Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009*, 2009.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Jana Koehler and Jörg Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, 12:338–386, 2000.

Yichao Liang, Nishanth Kumar, Hao Tang, Adrian Weller, Joshua B. Tenenbaum, Tom Silver, João F. Henriques, and Kevin Ellis. Visualpredicator: Learning abstract world models with neuro-symbolic predicates for robot planning. In *Proceedings of the 13th International Conference on Learning Representations, ICLR 2025*, pp. 60416–60444, 2025.

Yichao Liang, Dat Nguyen, Cambridge Yang, Tianyang Li, Joshua B Tenenbaum, Carl Edward Rasmussen, Adrian Weller, Zenna Tavares, Tom Silver, and Kevin Ellis. Exopredicator: Learning abstract models of dynamic worlds for robot planning. In *Proceedings of the 14th International Conference on Learning Representations, ICLR 2026*, pp. 49174–49214, 2026.

Nir Lipovetzky and Hector Geffner. Width and serialization of classical planning problems. In *Proceedings of the 20th European Conference on Artificial Intelligence, ECAI 2012*, pp. 540–545. IOS, 2012.

Grace Liu, Michael Tang, and Benjamin Eysenbach. A single goal is all you need: Skills and exploration emerge from contrastive rl without rewards, demonstrations, or subgoals. In *Proceedings of The 13th International Conference on Learning Representations, ICLR 2025*, pp. 78599–78621, 2025.

Y Isabel Liu, Bowen Li, Benjamin Eysenbach, and Tom Silver. Slap: Shortcut learning for abstract planning. In *Proceedings of the 14th International Conference on Learning Representations, ICLR 2026*, pp. 68901–68926, 2026.

Nicola J Müller, Moritz Oster, Isabel Valera, Jörg Hoffmann, and Timo P Gros. Per-domain generalizing policies: On learning efficient and robust q-value functions. In *Proceedings of the 36th International Conference on Automated Planning and Scheduling, ICAPS 2026*, pp. 703–707, 2026.

Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet. Lp-based heuristics for cost-optimal planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling, ICAPS 2014*, pp. 226–234, 2014.

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

Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Kaelbling, and Joshua B Tenenbaum. Predicate invention for bilevel planning. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence, AAAI 2023*, pp. 12120–12129, 2023.

Simon Ståhlberg and Hector Geffner. First-order representation languages for goal-conditioned RL. In *Proceedings of the 40th AAAI Conference on Artificial Intelligence, AAAI 2026*, pp. 36394–36402, 2026.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling, ICAPS 2022*, pp. 629–637, 2022.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general policies with policy gradient methods. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023*, pp. 647–657, 2023.

Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. The 2023 international planning competition. *AI Magazine*, 45(2):280–296, 2024. ISSN 2371-9621. doi: 10.1002/aaai.12169.

Sam Toyer, Felipe W. Trevizan, Sylvie Thiébaux, and Lexing Xie. Action schema networks: Generalised policies with deep learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 6294–6301, 2018.

A Proof for Prop. 1

Proposition 1. *Let π be a deterministic policy in a deterministic MDP with a discount factor γ, s_g a fixed goal state, and f^* the fixed point of the binary NCE loss. For simplicity, assume that the policy π will stay in s_g once it is reached. Given a state s and two actions a^* and a' such that s^* and s' are the successor states of applying a^* and a' in s, respectively, if π will reach s_g from s^* in t_1 steps and from s' in t_2 steps, then, we have that*

$$f^*(s, a^*, s_g) - f^*(s, a', s_g) = (t_2 - t_1)\log(1/\gamma)$$

Proof. Following the result by Eysenbach et al. (2022), we have that

$$f(s, a, s_g) = \log \mathbb{P}^f_\pi(s_g|s, a) - \log \mathbb{P}_G(s_g)$$

for any state-action pair (s, a), where $\mathbb{P}^f_\pi(s_g|s, a) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \mathbb{P}_\pi(s_t = s_g|s, a)$ is the discounted state occupancy of s_g. By assumption, π will reach s_g from s^* (s^* is the successor state of applying a^* in s) in t_1 steps and will stay in s_g indefinitely. We thus have that $\mathbb{P}_\pi(s_t = s_g|s, a^*) = 0$ for all $t < t_1$ and $\mathbb{P}_\pi(s_t = s_g|s, a^*) = 1$ for all $t \geq t_1$. As a consequence,

$$\mathbb{P}^f_\pi(s_g|s, a^*) = (1-\gamma)\sum_{t=t_1}^{\infty}\gamma^t = (1-\gamma)\frac{\gamma^{t_1}}{1-\gamma} = \gamma^{t_1}$$

It follows that $f^*(s, a^*, s_g) = \log \gamma^{t_1} - \log \mathbb{P}_G(s_g)$. By applying the same arithmetic to a', we have that $f^*(s, a', s_g) = \log \gamma^{t_2} - \log \mathbb{P}_G(s_g)$. As a result,

$$f^*(s, a^*, s_g) - f^*(s, a', s_g) = \log \gamma^{t_1} - \log \gamma^{t_2} = (t_2 - t_1)\log(1/\gamma)$$

This completes the proof. □

B Implementation Details

Encoder We encode the input (i.e., a state-action pair and a commanded atom) to our critic as a graph. Representing a state as a graph is a common practice in learning-based methods for symbolic planning. Here, we include the action and the commanded atom in the graph representation as well. Note that, unlike the convention in CRL where there are two separated encoders, one for the state-action pair and the other for the commanded state, here we represent both two in a single graph and use a joint encoder. We do so because in our experiments, we found that using separated encoders could downgrade the performance.

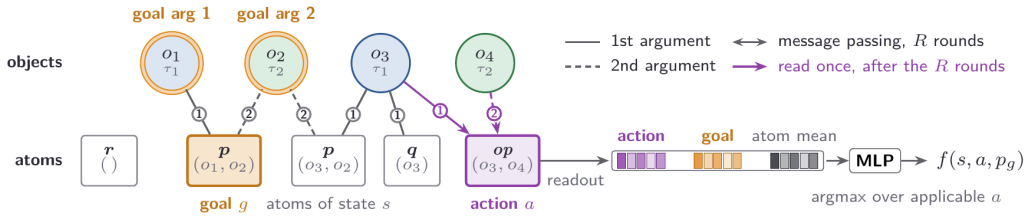

Figure 5: The architecture of our encoder.

In our architecture, we represent a state-action pair and a commanded atom as a *bipartite* graph. On one side of the graph, nodes are all the *objects* in the instance. On the other side, nodes are all the atoms that are true in the state, together with the input action and the commanded atom. An example is shown in Fig. 5. Nodes on the top side are objects in the planning instance, and those on the bottom are atoms in the state, and the input action and the commanded atom. For convenience, we name the commanded atom the goal atom in the figure. Each atom node, including the goal atom, is connected to the corresponding object nodes which are its arguments. The action node is connected to the corresponding object nodes in the same way.

Embeddings We initialize each object node according to its type, this is to say, at the beginning, all objects of the same type will have the same embedding. For each atom node, its embedding is a concatenation of the embeddings of its base predicate (predicate embedding) and a flag embedding indicating whether this atom is the goal atom or not. In other words, at the beginning, atoms that share the same predicate will have the same predicate embedding, all non-goal atoms also have the same flag embedding, and the goal atom will have a different one. The embedding of the action node is initialized according to its base action schema. Each edge connecting an object node and an atom (or action) node has a label indicating the position of the object in the argument list of the atom (or action). Edges with the same label will have the same weight matrix used in the message passing. For instance, in Fig. 5, all edges labeled with 1 mean that the objects on the one side of the edge are the first argument of the corresponding atoms they connect to and hence will be initialized with the same weight matrix. One remark is that, the action node does *not* take apart in the message passing. Its embedding is only updated from the object nodes it connects to after the message passing is done. The final embedding of the tuple (s, a, p_g) is a concatenation of the embeddings of the action node, the goal atom node, and the pool of atom nodes of the state.