Abductive State Grounding For Neuro-Symbolic Reinforcement Learning

Min-Le Su[*1, 2], **Kai-Yu Chen**[*2], **Jiong-Da Wang**[1, 2], **YiKuan Hu**[1, 2], **Wang-Zhou Dai**[1, 2],

[1]National Key Laboratory for Novel Software Technology, Nanjing University, China
[2]School of Intelligence Science and Technology, Nanjing University, China
{suml, kaiyuchen, jiongdawang}@smail.nju.edu.cn
kilgrim@foxmail.com
daiwz@lamda.nju.edu.cn

Abstract

In reinforcement learning, the use of symbolic state representations usually offers higher training efficiency and better generalization. However, converting sensory raw inputs into symbolic states usually requires pre-trained models and manually annotated datasets, leading to significant human labor. This paper proposes a state grounding method based on abductive learning, which can automatically accomplish the conversion from subsymbolic states to symbolic states by utilizing some general, environment-related background knowledge while exploring the environment. This method does not require pre-annotated data or pre-trained models; instead, it leverages first-order logic knowledge to provide supervised information for training the grounding model. Meanwhile, within the abductive learning framework, it can realize end-to-end training for both the visual grounding model and the deep reinforcement learning model.

Introduction

Reinforcement learning (Sutton and Barto 1998) has developed rapidly over the past two decades. Benefiting from the advancement of deep learning, a series of excellent and widely applied algorithms such as Deep Q-Network (DQN) (Mnih et al. 2013) and Proximal Policy Optimization (PPO) (Schulman et al. 2017) have emerged.

In reinforcement learning, state representations can be divided into symbolic representations and subsymbolic representations. Most of the deep reinforcement learning methods that have achieved remarkable success recently are based on subsymbolic representations. However, these methods still have many limitations, which are mainly reflected in low learning efficiency, poor generalization ability, and lack of interpretability (Núñez Molina, Mesejo, and Fernández-Olivares 2024).

On the other hand,reinforcement learning based on symbolic representations exhibits higher learning efficiency, stronger generalization ability, and greater interpretability. For example,we have conducted a series of comparative experiments in FrozenLake (Towers et al. 2024) and the Results are shown in Figure 1 . Furthermore, some reinforcement learning tasks accomplished by relying on large language models can only handle those based on symbolic representations (Ye, Arenz, and Kersting 2025).

While reinforcement learning based on symbolic representations offers numerous advantages, it still has certain drawbacks, including the requirement for a symbolic environment or pre-trained models that can extract symbols from inputs. For instance, to convert raw inputs (such as images) into symbolic data, a large number of annotated image datasets in the environment must be collected, and then a mapper like a Convolutional Neural Network (CNN) needs to be trained to achieve this goal.

To address such challenges, we propose the abductive state grounding method. By leveraging exploration in the subsymbolic environment, this method enables the agent to automatically ground task-relevant symbols. It reduces the requirement for labeled samples in state space grounding, enabling reinforcement learning (RL) to automatically learn how to abstract subsymbolic states into symbolic states during the decision-making learning process, thereby improving RL performance.

We formalize the state grounding problem as an abductive learning problem. Abductive learning (Dai et al. 2019) is a framework that can integrate machine learning and symbolic reasoning, similar to the dual-system in the human decision-making process (Shane and Frederick 2005) in neuroscience: System 1 outputs rapid intuition, which is generally undertaken by neural networks; System 2 outputs slow reasoning, which is generally undertaken by knowledge bases. Abductive learning has also been applied in many works (Hu et al. 2025; Huang et al. 2023; Yang et al. 2024).

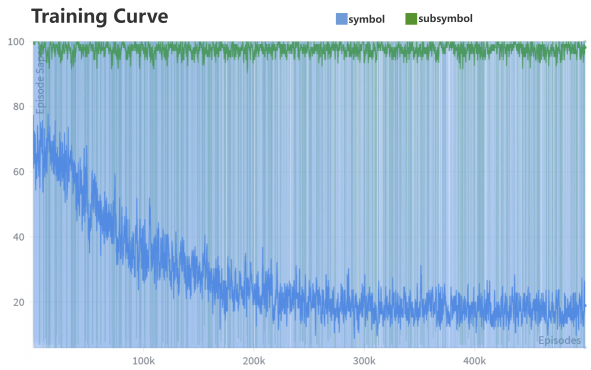
In our task,the knowledge refers to environment-related symbolic conceptual knowledge, which can be transferred across domains and tasks—for example, the agent can move, and walls block the agent's movement. The machine learning component consists of a visual grounding model and a deep reinforcement learning model; both can be unified under the ABL framework for end-to-end training.

Formulation

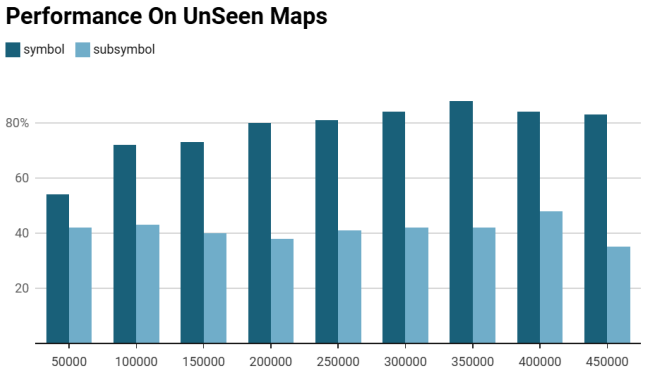
State Grounding

A state grounding system can be abstracted as a triple $\langle R, S, h \rangle$, where R denotes the original subsymbolic space

[*]Co-first authors.



(a) Training curve. When stepping into a hole, it is regarded as having executed 100 steps, and the next round starts.



(b) Completion rate of models (trained for a certain number of rounds) on 100 unseen maps.

Figure 1: Comparison of Training Efficiency and Generalization Between Symbol Representation and Subsymbol Representation in the FrozenLake Environment

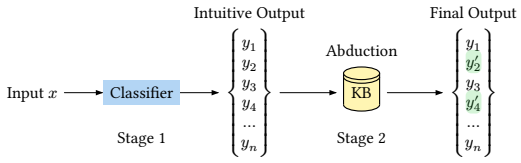


Figure 2: Abductive Learning Framework.

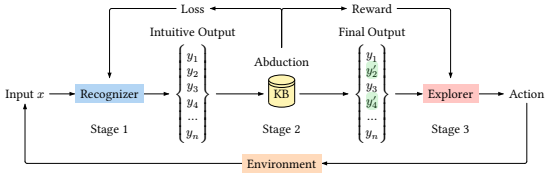


Figure 3: ABL-Explorer Framework.

where $R \subseteq \mathbb{R}^n$ (here n represents the dimension of the subsymbolic space), S denotes the symbolic space and satisfies $S = S_1 \times S_2 \times \cdots \times S_m$ (where $S_i = \{S_i^1, S_i^2, \ldots, S_i^{k_i}\}$ is a finite set whose elements stand for symbols, and k_i denotes the upper limit of the number of possible symbols in the i-th dimension), and $h : R \to S$ represents the grounding function to be learned, which is responsible for mapping instances in the subsymbolic space R to the symbolic space S.

For example, in our experiment, the subsymbolic space R is the pixel space. That means the input is an image. The image shows a square map. This map is made up of grid blocks. Let's assume both its length and width are 8 blocks. So the size of the symbolic space S should be 8 x 8 x K. Here, K stands for the number of possible block types.

Abductive Learning

Abductive learning has typically been used in supervised training classification tasks. Its general process is shown in Figure2. In the first stage, a machine learning model (such as a neural network) provides an intuitive output y, which is then sent to the knowledge base KB for consistency optimization. During this process, parts of y are modified to obtain the final output y' that has the best consistency with the knowledge base.

Symbol Abduction By Exploration

For a symbol S_i^j, we divide it into two components: $\langle \alpha_i^j, \beta_i^j \rangle$, where α_i^j denotes the direct representation of S_i^j . For instance, the α_i^j of the symbol 'key' is the pixels of the block where the 'key' is located, with a size of $G \times G \times 3$ (G represents the size of a block, and 3 corresponds to the RGB channels). In contrast, β_i^j represents both the intrinsic function of S_i^j and its interaction relationships with other symbols. For example, the β_i^j of the symbol 'key' can be expressed as rules in first-order logic:

$$\text{face\_to}(\text{player}(Y), X) \wedge \text{pick\_up}(\text{Action}) \wedge \text{pick\_able}(X) \rightarrow \text{key}(X)$$

assuming that the 'key' is the only pickable item in the environment.

Since β_i^j is more universal and generalizable than α_i^j: for example, the α_i^j values of 'Player' in different environments are usually different, and even the probability of their spatial dimensions being equal is very low. However, the β_i^j values of 'Player' in different environments have high similarity (e.g., all 'Player' entities possess the movement function).

We thus assume that α_i^j is unknown, while β_i^j is known and incorporated into the Knowledge Base (KB) of the abductive learning framework.

We perform abduction by β_i^j and environmental exploration to learn the α_i^j values of all symbols. When the α_i^j values of all symbols are learned, h is naturally acquired. For example, when the agent executes the "move forward" command: if a difference is detected between two adjacent frames, and the differing region contains a block that appears only once in the entire image, this block can be abduced as the player. In first-order logic, this is expressed as:

$$\text{movable}(X) \wedge \text{single}(X) \rightarrow \text{player}(X)$$

In some cases, multiple causes may lead to the same result. For example, after the agent executes "move forward", no difference is observed between two adjacent frames. This implies that the agent may be blocked by a wall, a key, or a door. In such scenarios, additional exploration is required to distinguish these possibilities and obtain features of the symbol Y that the player is facing (e.g., $\text{pick\_able}(Y) \wedge \neg\text{open\_able}(Y)$), thereby identifying $\text{Key}(Y)$.

The symbols obtained through abduction follow a certain order. For example, only after determining which block represents the player can we identify which symbols correspond to the remaining blocks.

Method

In this section, we will introduce the architecture and training process of ABL-Explorer.

Architecture

Traditional reinforcement learning models only have an end-to-end model that takes subsymbolic inputs and outputs actions, and are unable to convert subsymbols into symbols. Therefore, we add a visual grounding model and a knowledge base to them, enabling them to possess this conversion function. The overall flow chart of ABL-Explorer is shown in Figure 3.

Therefore, ABL-Explorer can be decomposed into 3 main stages. The first stage is what we call the Recognizer. In this stage, the Recognizer splits the raw environmental data X into smaller segments $(x_1, x_2, \ldots, x_n)$, and simultaneously provides an intuitive mapping $h\ (x_i \rightarrow Probs(S_i))$ where $X \in R$ and $Probs$ is probability distribution. We define $X = g(x_1, x_2, \ldots, x_n)$, where g depends on the prior knowledge of the environment. For example, in a grid-based environment, g functions to divide the environment into multiple small blocks.

The second stage is referred to as the abductive stage. The knowledge base KB will abduct which x_i can be definitely mapped to s_j based on the historical state X and action a. Based on this, it will update the mappings stored in KB, revise the mapping h provided in the first stage, and generate the final mapping h' at this moment.

The third stage is called the exploration stage. We fuse the decoupled raw environmental data $(x_1, x_2, \ldots, x_n)$ from the first stage with the final mapping h' obtained from the second stage to get the input X' for the third stage, which is $\{h'(x_1), h'(x_2), h'(x_3), \ldots, h'(x_n)\}$. We then input X' into the explorer neural network to obtain an action a, which is designed to enable the Agent to execute it such that the number of steps to explore all symbols is minimized.

Compared to traditional abductive learning, the addition of the explorer module enables it to actively interact with the environment, thereby updating the knowledge base and aligning the prior knowledge and the real environment. In contrast to traditional reinforcement learning, the integration of abductive learning makes it more convenient to introduce prior knowledge, abstract symbols with interpretability, and bridge the neural system and the symbolic system. Meanwhile, when encountering data x_i that has never triggered certain rules, the Explorer can also make predictions on it.

Training Paradigm

There are a total of 2 neural networks in the entire model that need to be trained: namely the Recognizer in Stage 1 and the Explorer in Stage 3. This section will explain how each of them is trained respectively.

First, for the Recognizer in Phase 1, its loss consists of two components: the system 1 loss and the system 2 loss (Shane and Frederick 2005). The system 1 loss represents some fast, intuitive losses. For example, if we over-predict a symbol $s_i \in S_{small}$ where S_{small} is a subset consisting of all symbols in the symbol set that are expected to appear infrequently. when it exceeds a certain threshold, the absolute value of the sum of probabilities for this symbol in the probability matrix minus that threshold is taken as part of the system 1 loss. That is,

$$\begin{aligned} L_{\text{sys1}}(X) = &\sum_{s_i \in S_{\text{small}}} \max(0, \sum_{x_j \in g'(X)} f(x_j)[i] - \tau_{small}) \\ &+ \sum_{s_i \in S_{\text{large}}} \max(0, \tau_{large} - \sum_{x_j \in g'(X)} f(x_j)[i]) \end{aligned}$$

where f is the symbol prediction function of the Recognizer for x_j, i.e., $f(x_j) \in \mathbb{R}^n$. The purpose of designing the system 1 loss is that it can provide an intuitively predicted result with good confidence when certain symbols have not been determined.

The system 2 loss, on the other hand, simply calculates the sum of squared errors between the intuitive results of the Recognizer and the results corrected by the KB.

$$L_{sys2}(X) = \frac{1}{n} \sum_{x_i \in g'(X)} ||f(x_i) - f'(x_i)||^2$$

To summarize, the total loss of the Recognizer is as follows

$$L_{\text{Rec}} = \alpha \cdot L_{\text{sys1}} + \beta \cdot L_{\text{sys2}}$$

The Explorer is a reinforcement learning model. We mainly use PPO (Proximal Policy Optimization) (Schulman et al. 2017) for its implementation, with the focus being on its reward design. The reward for each step is set as follows:

$$R = N(g'(X_t), a_t, g'(x_{t-1}), a_{t-1}, ...) - p$$

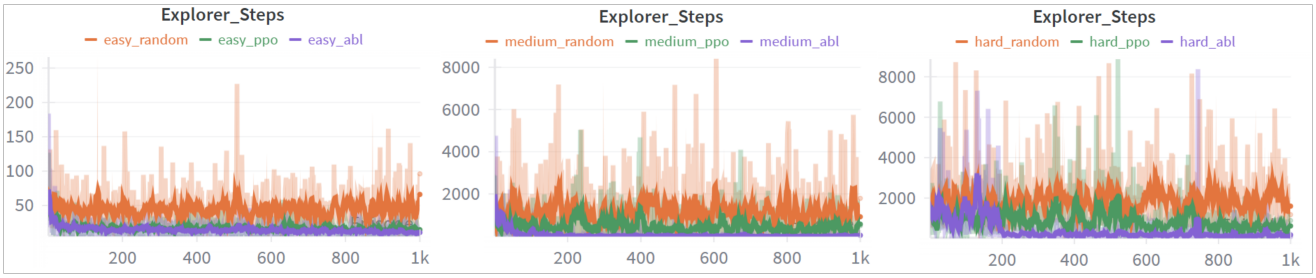


Figure 4: Experiments in Minigrid Environment. The difficulty increases from left to right, and a curve that lies lower indicates better performance.

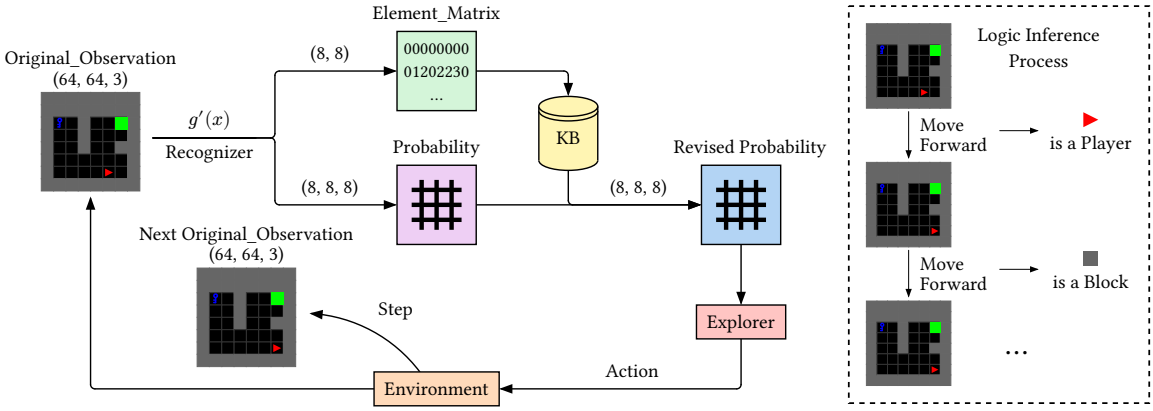


Figure 5: The Specific Flow Chart of ABL-Explorer in the Minigrid Environment

Here, N represents the number of newly discovered mapping relationships of the action in this step, determined based on historical information. Meanwhile, p is a constant term, which denotes a penalty incurred for each step executed. This penalty mechanism helps the model achieve efficient exploration within a shorter time frame.

Experiments

In this section, we will test our method in the Minigrid (Chevalier-Boisvert et al. 2023) environment. Within this environment, there are numerous symbols that can be explored through interaction with the environment. However, some of these symbols can only be explored by performing a series of combined actions, making this task highly challenging.

Furthermore, we will conduct additional tests of our method in the MiniHack (Samvelyan et al. 2021) environment to verify the reliability of our approach across different settings. This environment offers a more diverse set of rules, with various symbols that also need to be identified through interaction with the environment. Notably, the mutual occlusion of symbols in images and the requirement that certain symbols can only be recognized via specific combinations of actions render this task equally challenging.

Minigrid Symbol Exploration

Environment Setting We customized an 8×8 grid map, where each grid cell is an 8×8 RGB pixel, resulting in an original input dimension of 64×64×3. The action space has a size of 6, including turning left, turning right, moving forward, picking up, putting down, and switching. We also defined three difficulty levels: easy, medium, and hard, to facilitate multi-dimensional comparison of the advantages and disadvantages of ABL-Explorer.

In the easy levels, a total of 6 types of symbols need to be discovered: walls, floors, and players (in 4 directions). The medium levels require discovering 7 types of symbols, adding target points to those in the easy levels. The hard levels require discovering 8 types of symbols, adding keys to those in the medium levels. We ensure that all symbols

Method	Training Steps		
	Easy	Medium	Hard
Random	45735	1168486	1893838
PPO	18205	492785	922411
ABL-Explorer(ours)	**15476**	**91384**	**439845**

Table 1: Training Steps(1000 episodes in total) Across Different Difficulty Levels in Minigrid Environment.

that need to be discovered will definitely appear in the map.

Compared Methods To demonstrate the performance of ABL-Explorer, we compared it with two methods: (1) Random Action, where each output action is completely random; (2) PPO (Schulman et al. 2017), one of the most mainstream reinforcement learning algorithms currently, which outputs policies directly based on the original environment.

We present the training curves in Figure 4, and the total number of training steps for each method to complete 1000 episodes is detailed in Table 1. In the easy setting, random actions take an average of approximately 50 steps to complete one episode. The performance of ABL-Explorer is comparable to that of PPO, with ABL-Explorer requiring slightly fewer total steps than PPO. In the medium setting, random actions take an average of around 1000 steps to finish one episode. Here, ABL-Explorer significantly outperforms PPO in training: it only takes about 18.54% of the steps that PPO needs to complete 1000 episodes. Moreover, in terms of final performance, ABL-Explorer can consistently finish one episode within an average of 50 steps, while PPO still requires over 300 steps per episode. In the hard setting, random actions take an average of roughly 2000 steps to complete one episode. ABL-Explorer remains more effective than PPO, as it only uses approximately 47.68% of the steps required by PPO to complete 1000 episodes. Additionally, ABL-Explorer achieves a final performance of finishing one episode within an average of 50 steps, whereas PPO still needs an average of 500 steps per episode.

MiniHack Symbol Exploration

Environment Setting We customized a 7×7 grid level for learning the skill of eating food, where recognizing food essentially equates to mastering this eating skill. Each grid cell is a 16×16 RGB pixel, resulting in an original input dimension of 112×112×3. The action space has 5 actions, including moving up, moving down, moving left, moving right, and eating food. Within this level, a total of five types of symbols need to be identified: walls, floors, the player, food, and the starting point. The starting point is initially covered by the player.

Compared Methods We present our training curves in Figure 6 and the total number of steps required to complete 1000 episodes in Table 2.Similarly, we adopt the two aforementioned methods: Random Action and PPO. In the MiniHack-eat task level, random actions take an average of 245 steps to finish one episode. Here, ABL-Explorer also

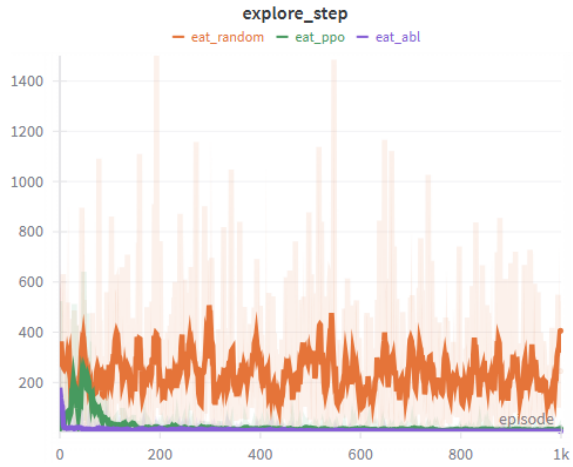


Figure 6: Experiments in MiniHack Environment. A curve that lies lower indicates better performance.

Method	Training Steps
Random	236373
PPO	25333
ABL-Explorer(ours)	**10409**

Table 2: Training Steps (1000 episodes in total) in MiniHack Environment.

achieves substantial improvements over PPO in training: it converges faster and only takes about 41.08% of the steps that PPO needs to complete 1000 episodes. Moreover, in terms of final performance, ABL-Explorer can consistently finish one episode within an average of 7 steps, while PPO still requires an average of 13 steps per episode.

Conclusion

This article proposes an automated state grounding method based on abductive learning. Specifically, we formalize state grounding as an abductive learning problem, conduct exploration in the environment, collect feedback, and then abduce symbols based on a general set of prior knowledge. Without relying on pre-trained models or annotated datasets, this method can convert subsymbolic states into symbolic states, which greatly reduces the cost of manual annotation and significantly improves the efficiency of reinforcement learning tasks.

Future Work

The experiments in this article are conducted in a closed environment, which means the knowledge base is guaranteed to contain all relevant symbols present in the environment. In the future, we aim to extend this method to an open environment—specifically, enabling it to explore and identify unknown symbols. Therefore, Inductive Logic Programming (ILP) (Muggleton 1999) is one of the techniques worth considering, as it can induct rules for unknown symbols from

partial examples. Additionally, large language models could also be attempted to fulfill this function.

References

Chevalier-Boisvert, M.; Dai, B.; Towers, M.; Perez-Vicente, R.; Willems, L.; Lahlou, S.; Pal, S.; Castro, P. S.; and Terry, J. 2023. Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. In *Advances in Neural Information Processing Systems 36, New Orleans, LA, USA*.

Dai, W. Z.; Xu, Q.; Yu, Y.; and Zhou, Z. H. 2019. Bridging Machine Learning and Logical Reasoning by Abductive Learning *. In *NeurIPS 2019*.

Hu, W.-C.; Dai, W.-Z.; Jiang, Y.; and Zhou, Z.-H. 2025. Efficient Rectification of Neuro-Symbolic Reasoning Inconsistencies by Abductive Reflection. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI)*, 17333–17341.

Huang, Y.-X.; Dai, W.-Z.; Jiang, Y.; and Zhou, Z.-H. 2023. Enabling Knowledge Refinement upon New Concepts in Abductive Learning. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI'23)*, 7928–7935.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602.

Muggleton, S. 1999. Inductive Logic Programming: Issues, results and the challenge of Learning Language in Logic. *Artificial Intelligence*, 114(1): 283–296.

Núñez Molina, C.; Mesejo, P.; and Fernández-Olivares, J. 2024. A Review of Symbolic, Subsymbolic and Hybrid Methods for Sequential Decision Making. *ACM Comput. Surv.*, 56(11).

Samvelyan, M.; Kirk, R.; Kurin, V.; Parker-Holder, J.; Jiang, M.; Hambro, E.; Petroni, F.; Kuttler, H.; Grefenstette, E.; and Rocktäschel, T. 2021. MiniHack the Planet: A Sandbox for Open-Ended Reinforcement Learning Research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347.

Shane; and Frederick. 2005. Cognitive Reflection and Decision Making. *Journal of Economic Perspectives*.

Sutton, R. S.; and Barto, A. G. 1998. Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, 9(5): 1054.

Towers, M.; Kwiatkowski, A.; Terry, J.; Balis, J. U.; De Cola, G.; Deleu, T.; Goulão, M.; Kallinteris, A.; Krimmel, M.; KG, A.; et al. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. *arXiv preprint arXiv:2407.17032*.

Yang, X.-W.; Shao, J.-J.; Tu, W.-W.; Li, Y.-F.; Dai, W.-Z.; and Zhou, Z.-H. 2024. Safe abductive learning in the presence of inaccurate rules. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press. ISBN 978-1-57735-887-9.

Ye, Z.; Arenz, O.; and Kersting, K. 2025. Learning from Less: Guiding Deep Reinforcement Learning with Differentiable Symbolic Planning. arXiv:2505.11661.