

GALOIS: Boosting Deep Reinforcement Learning via Generalizable Logic Synthesis

Yushi Cao^{2,*}, Zhiming Li^{2,*}, Tianpei Yang^{1,3,†}, Hao Zhang¹, Yan Zheng^{1,†}
 Yi Li², Jianye Hao¹, Yang Liu²

¹College of Intelligence and Computing, Tianjin university, Tianjin, China

²Nanyang Technological University, Singapore, ³University of Alberta, Canada

{yushi002,zhiming001}@e.ntu.edu.sg

{tpyang,3018216216,yanzheng,jianye.hao}@tju.edu.cn

{yi_li,yangliu}@ntu.edu.sg

A Synthesized Programs

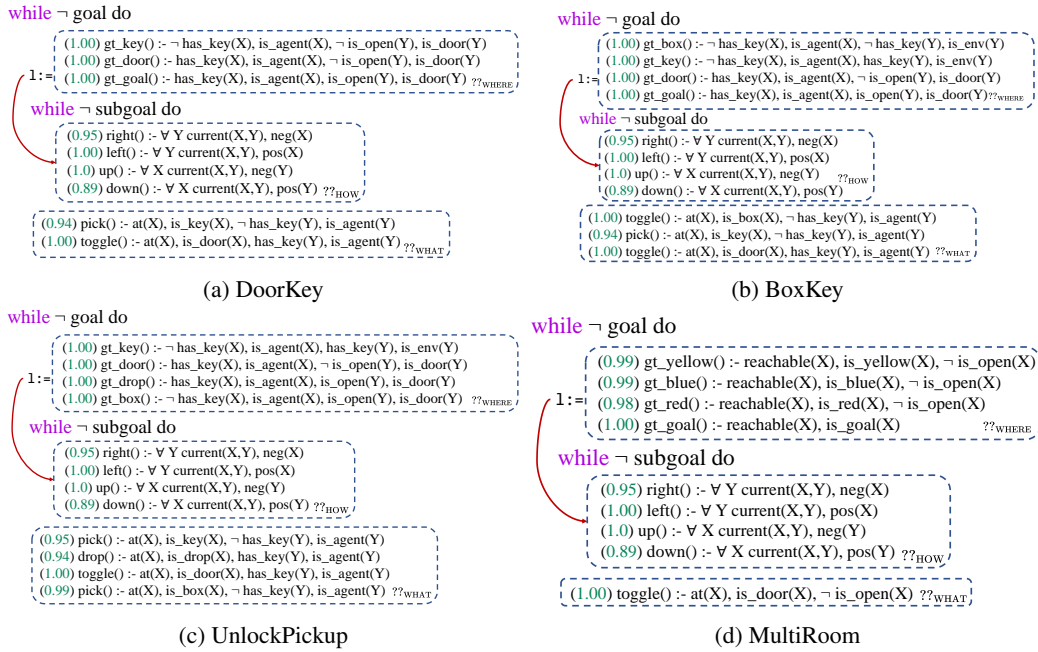


Figure 1: Detailed synthesized programs.

In this section, we present the detailed synthesized programs for the four evaluated environments in Figure 1.

DoorKey As shown in Figure 1a, regarding the hole function $??_{\text{WHERE}}$, the agent goes to the key if it verifies that the grounded fact to be true: the agent has no key and the door is closed; goes to door if it has key and the door is closed; goes to target location if it has key and the door is already open. And for hole function, $??_{\text{WHAT}}$, the agent picks up the key if it is adjacent to the key and it has no key; toggles the door if it is adjacent to the door and it has the key.

*Equal contribution.

†Corresponding authors: Yan Zheng (yanzheng@tju.edu.cn) and Tianpei Yang (tpyang@tju.edu.cn).

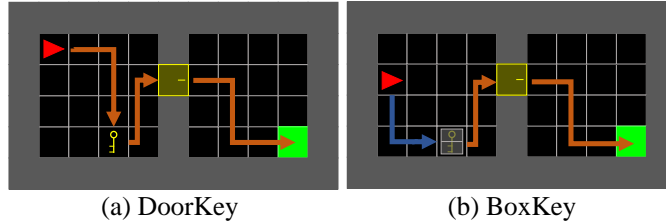


Figure 2: The illustration of knowledge reused from DoorKey to BoxKey. The orange path in (b) represents the reusable knowledge from (a). The blue path in (b) denotes the new knowledge need to learn for BoxKey.

BoxKey As shown in Figure 1b, different from DoorKey, it has to open the box to get the key. Thus the newly learned sub-program for $??_{\text{WHERE}}$ would be: agent should go to the box if there is no key in the environment and the agent does not have a key. Accordingly, for $??_{\text{WHAT}}$, the agent would toggle (open) the box when it arrives at the box.

UnlockPickup As shown in Figure 1c, since it has to pick up an object located at the target location to accomplish the task, the newly learned sub-program $??_{\text{WHERE}}$ would be: agent would drop the key when it has the key and the door is already open. And for $??_{\text{WHAT}}$, whenever the agent arrives at the goal position, it would pick up the box if it does not has a key and the door is already open.

MultiRoom As shown in Figure 1d, regarding $??_{\text{WHERE}}$, agent would head for a door of a specific color if it is reachable and closed, e.g., $\text{gt_red}() :- \text{reachable}(X), \text{is_red}(X), \neg \text{is_open}(X)$. Thus the learned program is color-agnostic (i.e., the agent’s policy would remain robust no matter how the colored doors sequence changes).

B Implementation Details

In this section we present the implementation details of all the baseline methods and our approach³, along with the information of the hardware on which these models are trained.

Environment settings. The valuation vector representations are fed to all the methods as input. Specifically, for the MPPS model, we directly provide the oracle program to save time for synthesizing programs, which could be considered as the upper bound performance of MPPS [3]. For the h-DQN method, we keep the high-level goal the same as MPPS and GALOIS, (i.e., {key, door, goal} for DoorKey; {box, key, door, goal} for BoxKey; {box, key, door, drop} for UnlockPickup; {red door, yellow door, blue door, goal} for MultiRoom).

The reward from the MiniGrid environment is sparse (i.e., only a positive reward will be given after completing the task), thus in order to motivate the agent to learn to solve the task using the least amount of actions, we follow previous works [1, 3] and apply the reward shaping mechanism. For every action taken, a negative reward of -0.01 will be given, a small positive reward of $+0.20$ will be given for achieving each sub-task (e.g., picking up the key), and a reward of $+1.00$ will be given for completing the task.

Training settings. For all DNN-based baseline methods, we use a 2-layer multilayer perceptron (MLP) with 128 neurons in each layer. We train all the baseline methods and GALOIS with the Adam optimizer [2] with a 0.001 learning rate. We use a batch size of 256. The entropy coefficient γ_ϵ starts at 5 and decays at a factor of 10 for every 50 episodes. We conducted all experiments on a Ubuntu 16.04 server with 24 cores of Intel(R) Xeon(R) Silver 4214 2.2GHz CPU, 251GB RAM, and an NVIDIA GeForce RTX 3090 GPU.

³The code is available at: <https://github.com/caoysh/GALOIS>

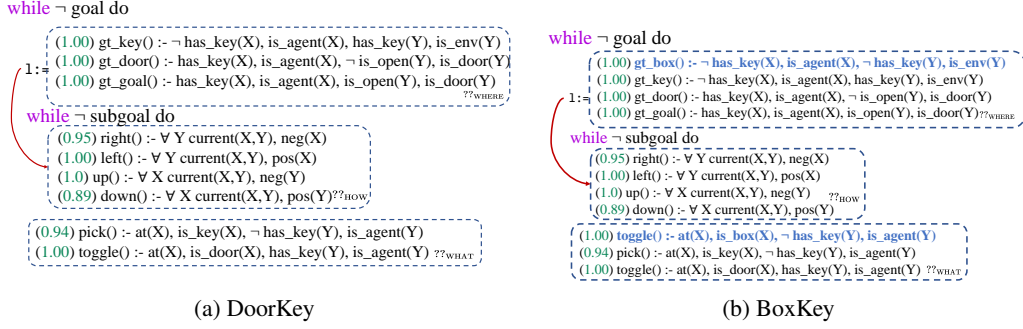


Figure 3: Reused program from DoorKey to BoxKey. The subprograms marked in blue (in (b)) need to be learned for BoxKey, while other subprograms are directly reused from (a).

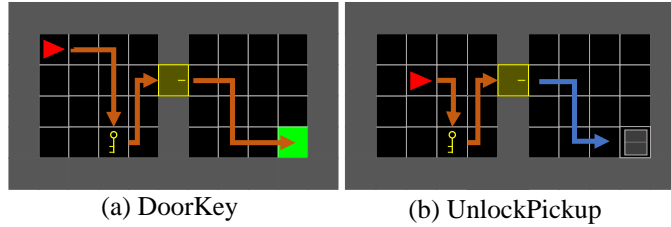


Figure 4: The illustration of knowledge reused from DoorKey to UnlockPickup. The orange path in (a) means knowledge learned from scratch. The orange path in (b) represents the reusable knowledge. The blue path in (b) means the new knowledge learned for UnlockPickup.

C Knowledge Reusability

In this section, we illustrate how the programs are reused from one environment to another in detail.

C.1 DoorKey to BoxKey

Figure 2 illustrates the DoorKey and BoxKey environments and their optimal policies, respectively. Different from DoorKey, to complete the BoxKey, the agent has to open the box to get the key first, while the agent in DoorKey is able to get the key directly. Afterward, the optimal policies would be the same, i.e., pick up the key, open the door and then go to the goal. Concretely, the BoxKey agent will only need to learn how to go to the box and open it, and thus all the knowledge can be reusable from DoorKey. As shown in Figure 3b, the subprograms marked in blue are the new knowledge to learn to solve the BoxKey while other sub-programs can be directly reused from DoorKey.

C.2 DoorKey to UnlockPickup

Figure 4 shows the DoorKey and UnlockPickup environments and their respective optimal policies. After opening the door, to solve UnlockPickup, the agent has to drop the key and then pick up the box. Specifically, the knowledge before dropping the key is reusable and the UnlockPickup agent continues to learn the rest. Due to the white-box nature of GALOIS, instead of reusing all the knowledge, the sub-program learned in DoorKey (i.e., `gt_goal() :- has_key(X), is_agent(X), is_open(Y), is_door(Y)`) can be removed when reusing. As shown in Figure 5, the sub-programs marked in blue are the new knowledge to learn while others are directly reused from DoorKey.

C.3 BoorKey to UnlockPickup

Figure 6 shows the BoorKey and UnlockPickup environments and their respective optimal policies. In specific, the agent in BoxKey learns the clause: `toggle() :- at(X), is_box(X), ¬has_key(Y), is_agent(Y)` while in UnlockPickup, the clause should be: `pick() :- at(X), is_box(X), ¬has_key(Y), is_agent(Y)`, as shown in Figure 7. This leads to a negative impact of reusing

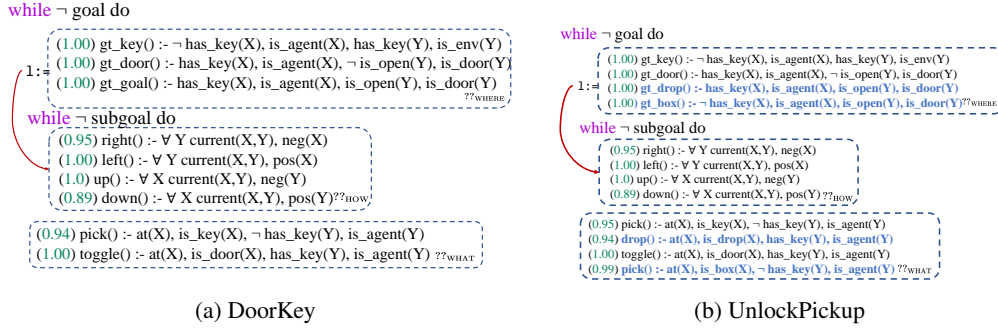


Figure 5: Reused program from DoorKey to UnlockPickup. The sub-programs marked in blue (in (b)) are needed to be learned for UnlockPickup while other sub-programs are directly reused from (a).

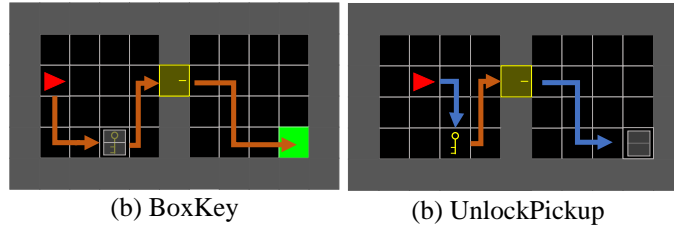


Figure 6: The illustration of knowledge reused from BoorKey to UnlockPickup. The orange path in (a) means knowledge learned from scratch. The orange path in (b) represents the reusable knowledge. The blue path in (b) means the new knowledge learned for UnlockPickup.

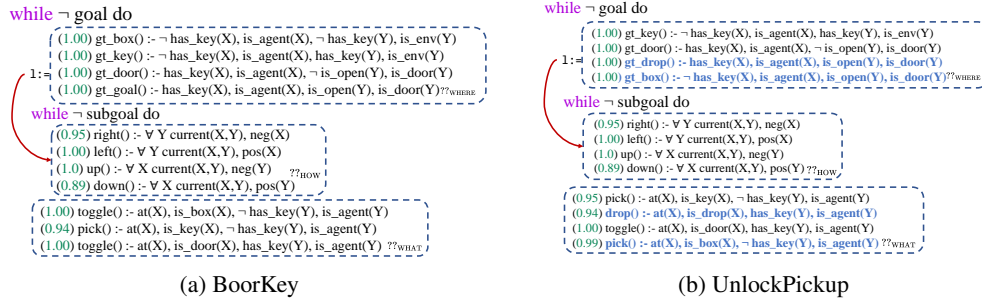


Figure 7: Reused program from BoorKey to UnlockPickup. The sub-programs marked in blue (in (b)) are needed to be learned for UnlockPickup while other sub-programs are directly reused from (a).

knowledge as in UnlockPickup environment, the box will disappear if toggled and the task will never be completed. The experiments demonstrate the ability of GALOIS to handle nonreusable knowledge. Additionally, thanks to its white-box nature, this rule can also be identified and removed manually to avoid negative effects.

References

- [1] Zhengyao Jiang and Shan Luo. Neural logic reinforcement learning. In *International Conference on Machine Learning*, pages 3110–3119. PMLR, 2019.
- [2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [3] Yichen Yang, Jeevana Priya Inala, Osbert Bastani, Yewen Pu, Armando Solar-Lezama, and Martin Rinard. Program synthesis guided reinforcement learning for partially observed environments. *Advances in Neural Information Processing Systems*, 34, 2021.