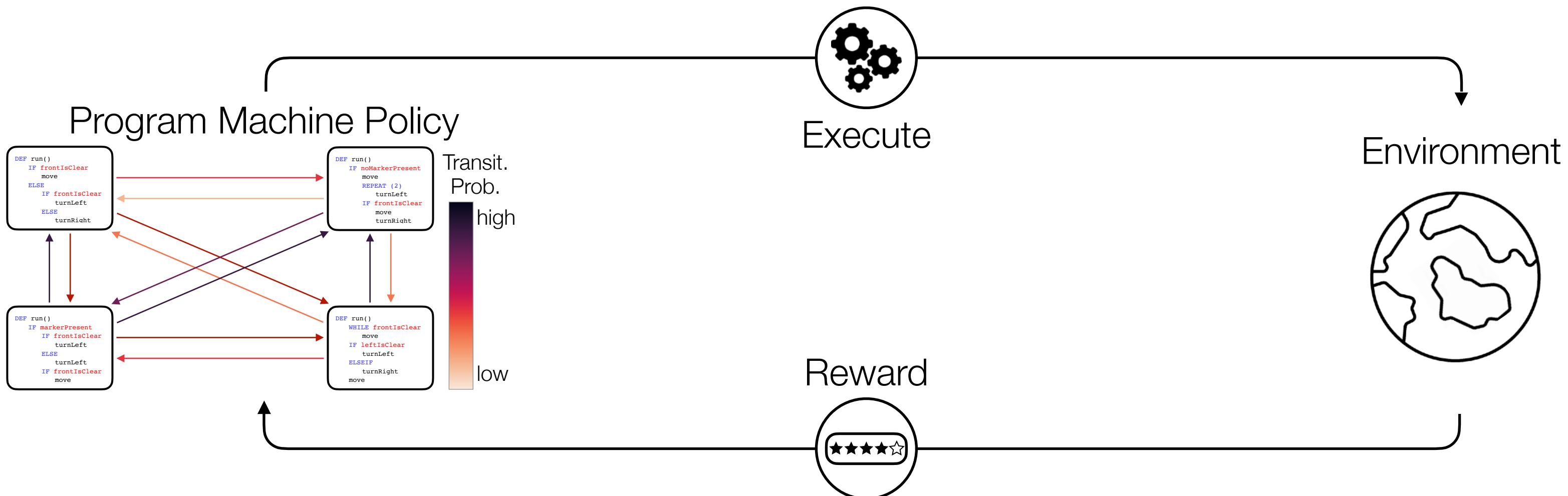


Addressing Long-Horizon Tasks by Integrating Program Synthesis and State Machines

NeurIPS 2023 Workshop on Generalization in Planning



Yu-An Lin*



Chen-Tao Lee*



Guan-Ting Liu*



Pu-Jen Cheng

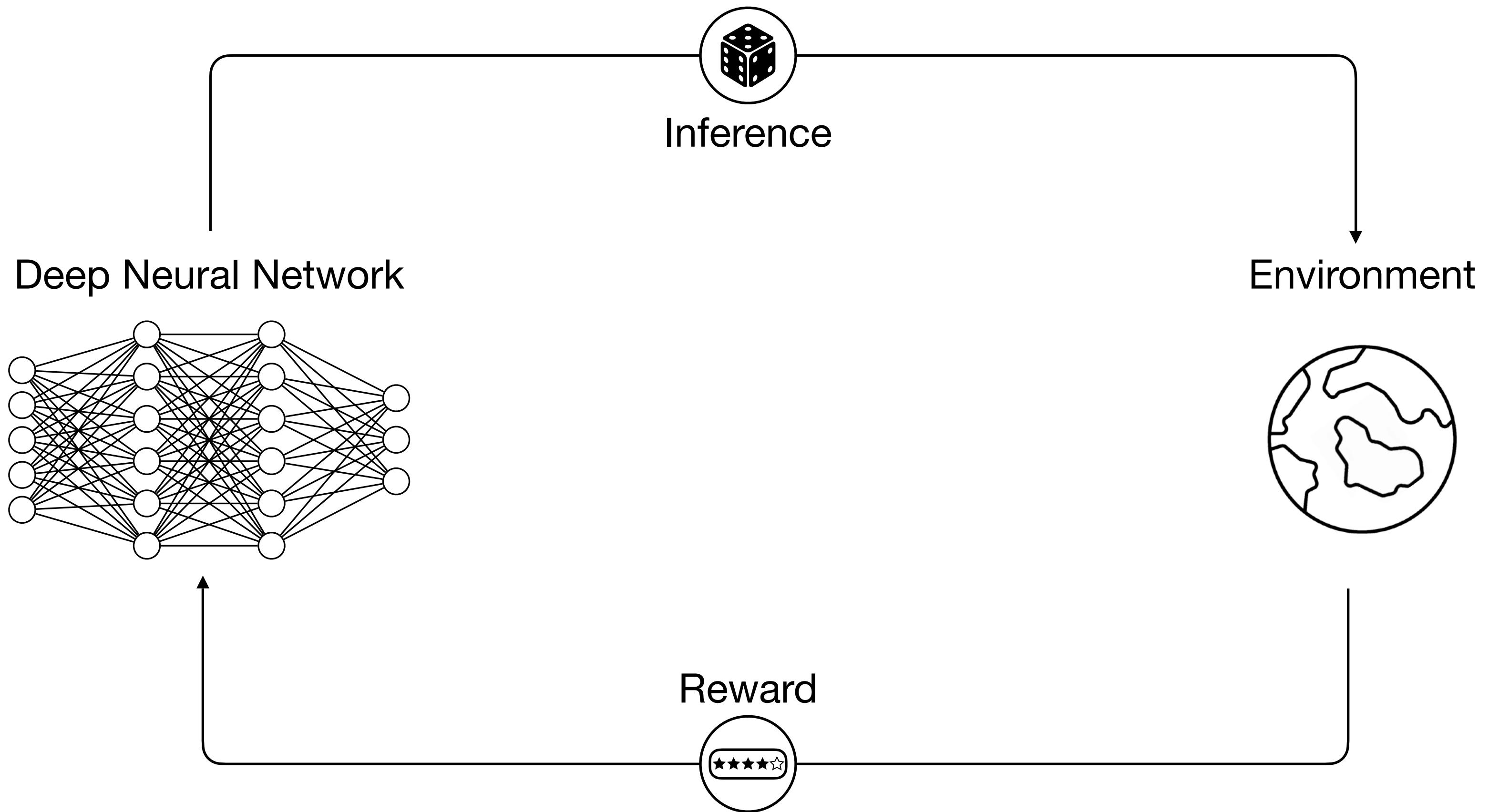


Shao-Hua Sun

National
Taiwan
University

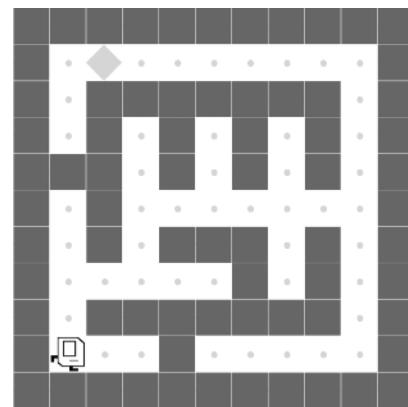


Deep Reinforcement Learning

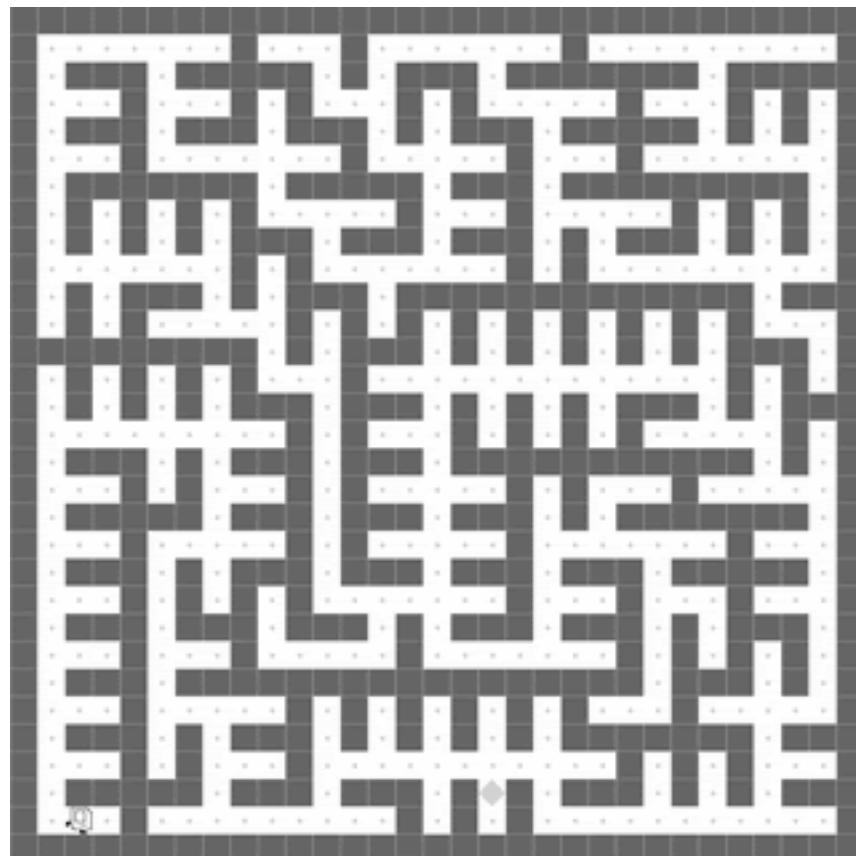
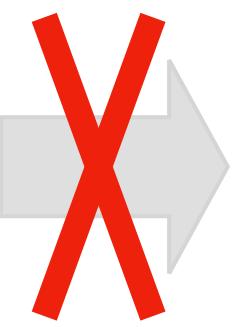


Deep Reinforcement Learning - Issues

Generalization



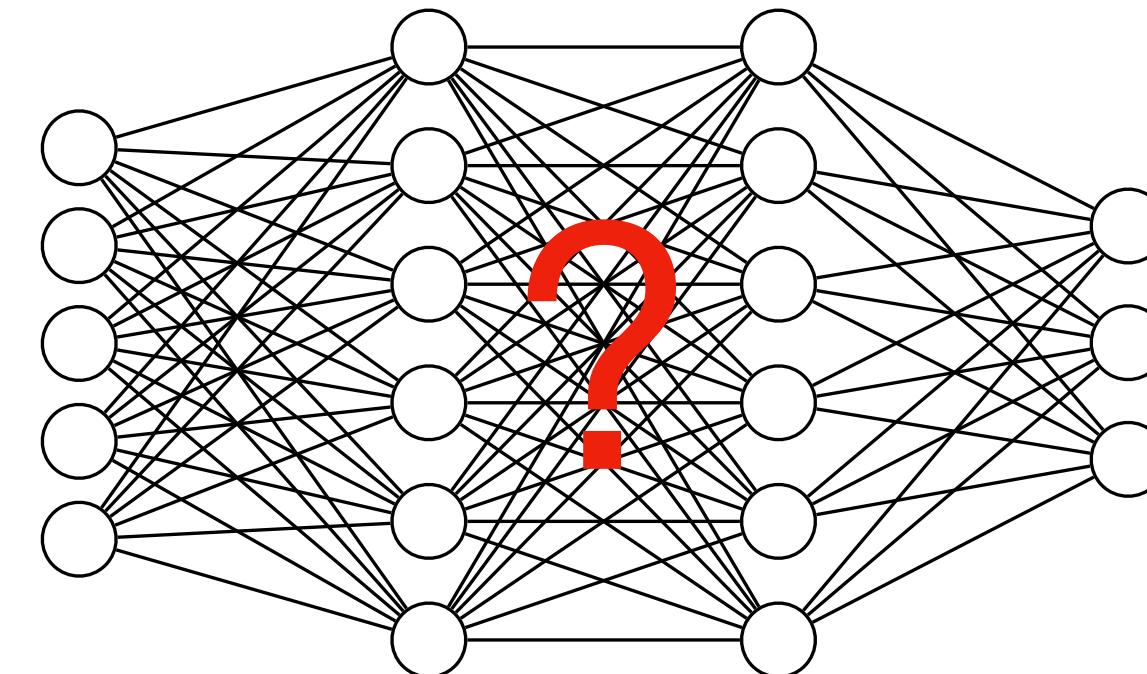
Simple task



Complex task

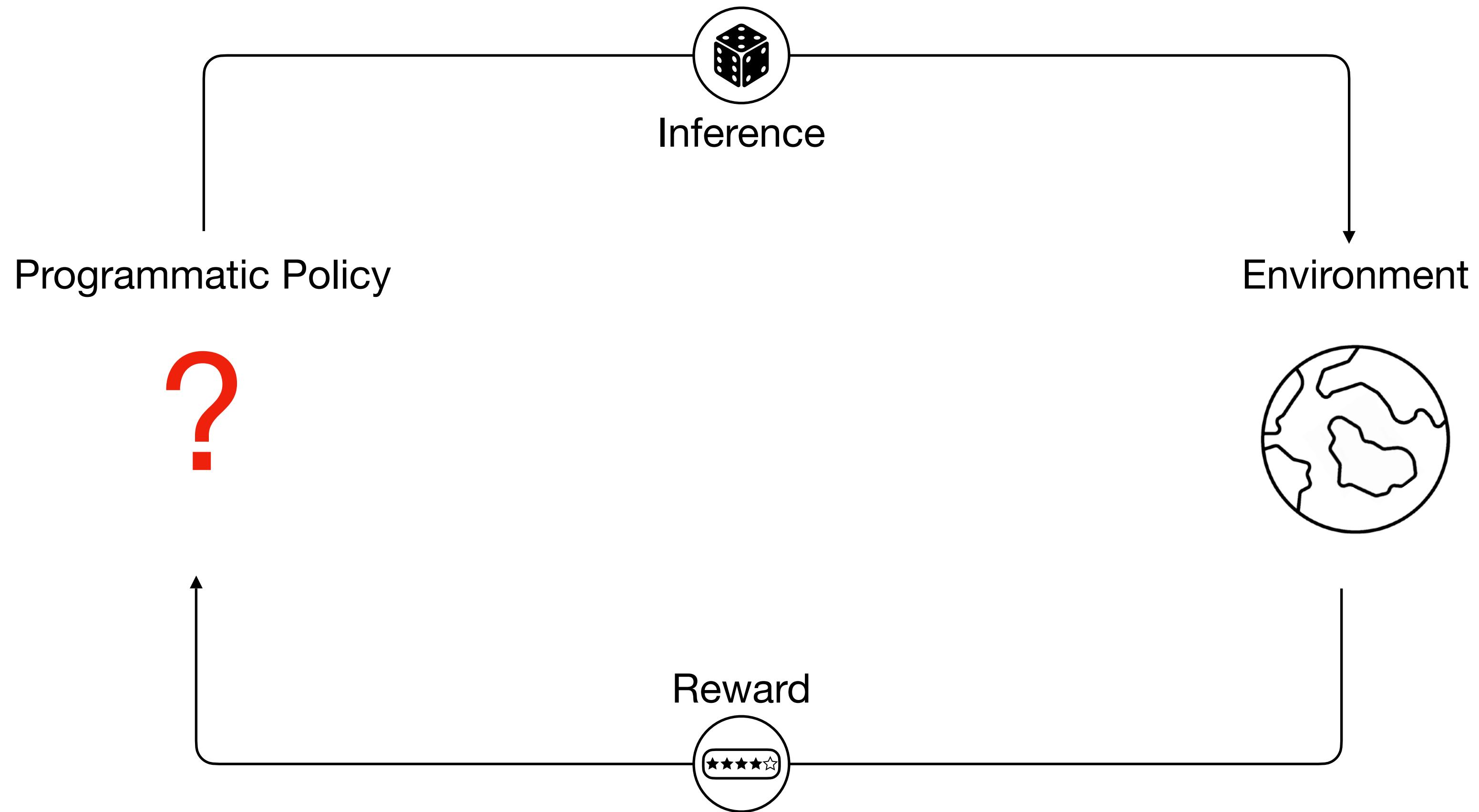
Interpretability

Trust, Safety, and Contestability

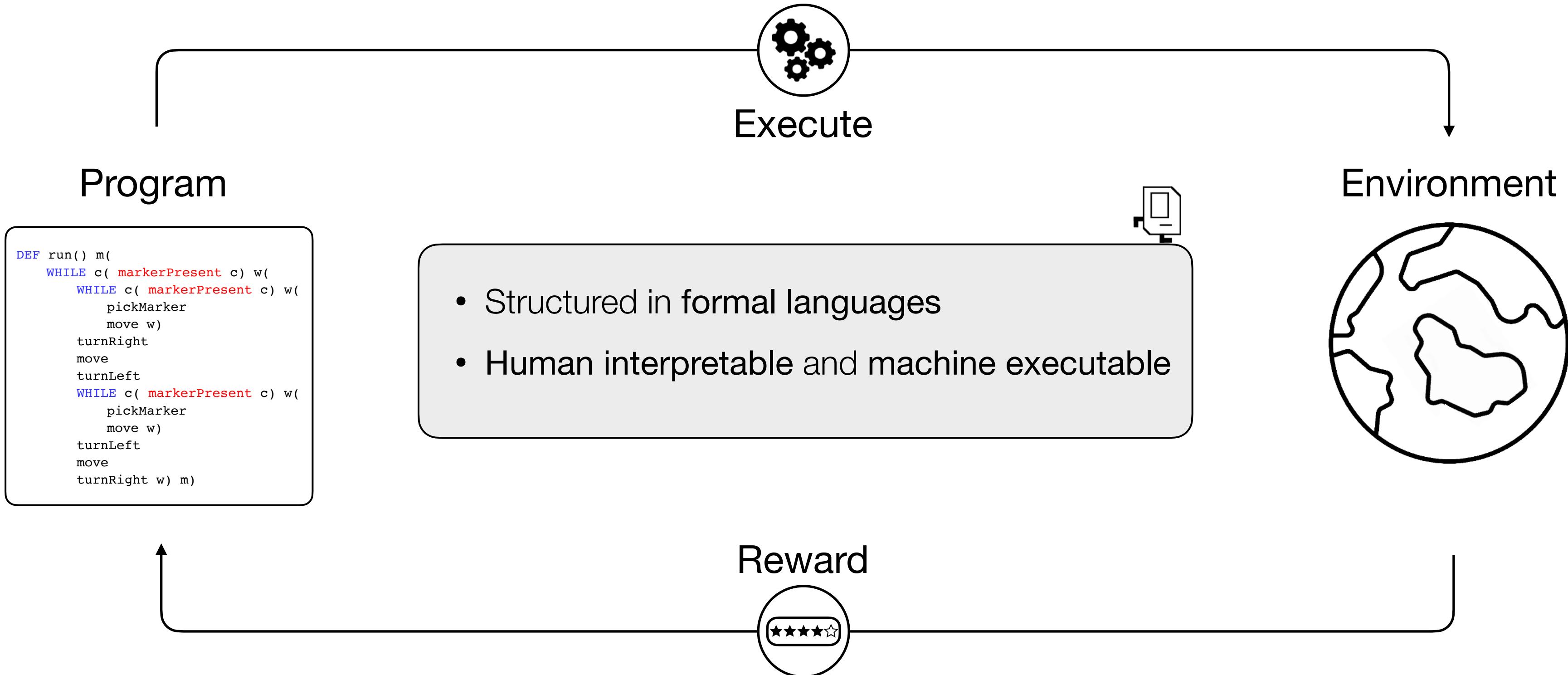


Deep neural network policy

Programmatic Reinforcement Learning



Programmatic Reinforcement Learning - Program Synthesis



Sun et al. "[Neural program synthesis from diverse demonstration videos](#)." ICML 2018.

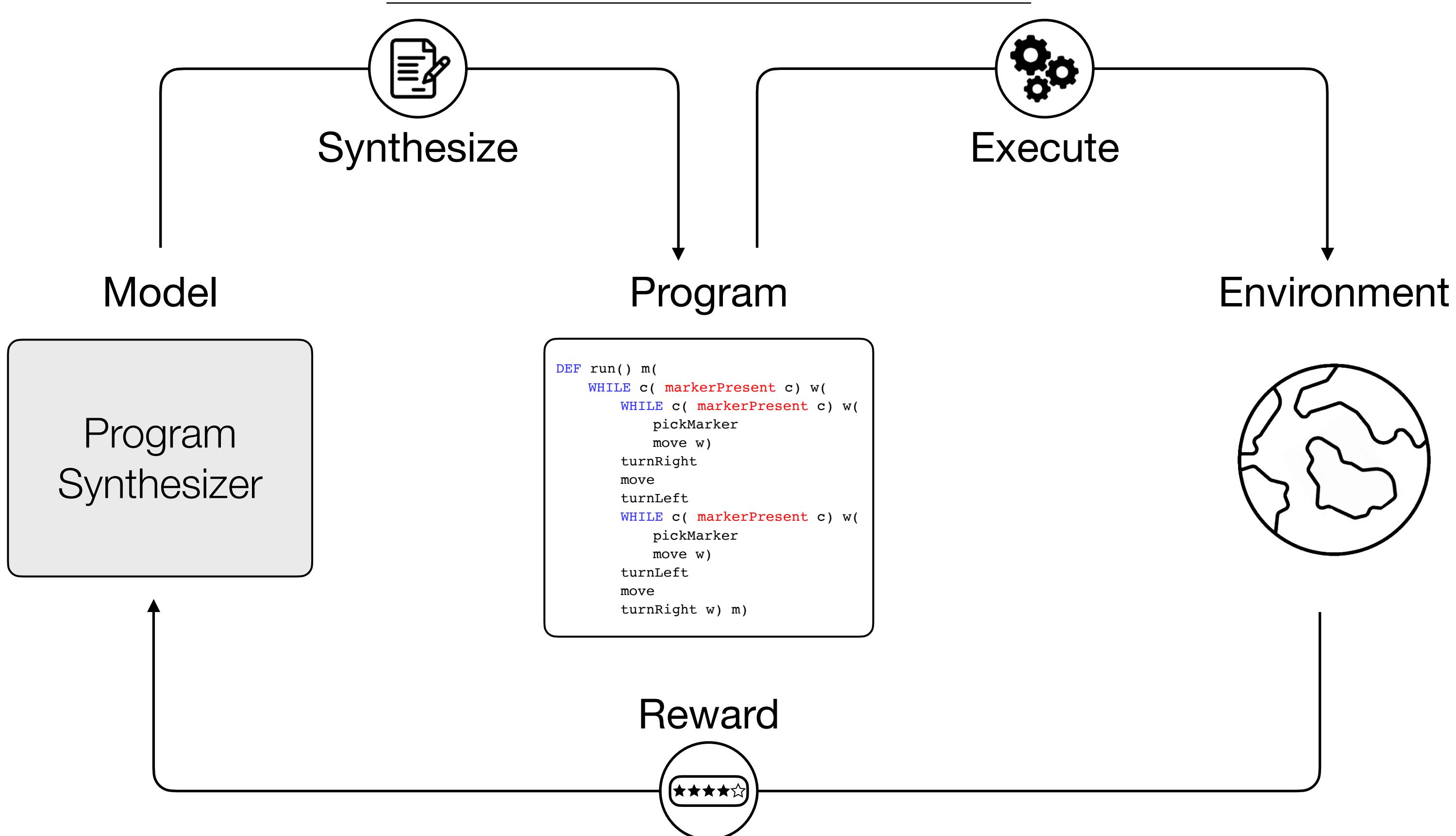
Sun et al. "[Program Guided Agent](#)." ICLR 2020.

Dang-Nhu "[PLANS: Neuro-symbolic program learning from videos](#)." NeurIPS 2020.

Trivedi et al. "[Learning to Synthesize Programs as Interpretable and Generalizable Policies](#)." NeurIPS 2021.

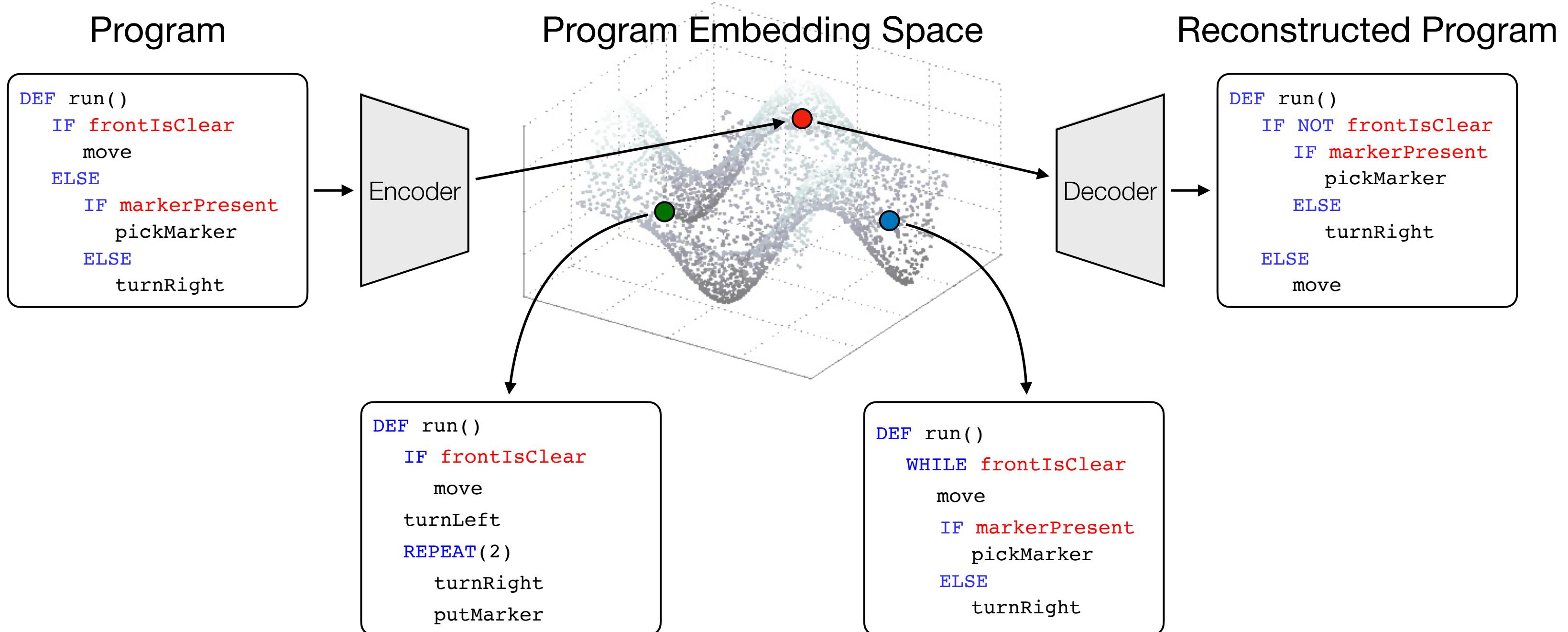
Liu et al. "[Hierarchical Programmatic Reinforcement Learning via Learning to Compose Programs](#)." ICML 2023.

LEAPS: Learning Embeddings for IAtent Program Synthesis



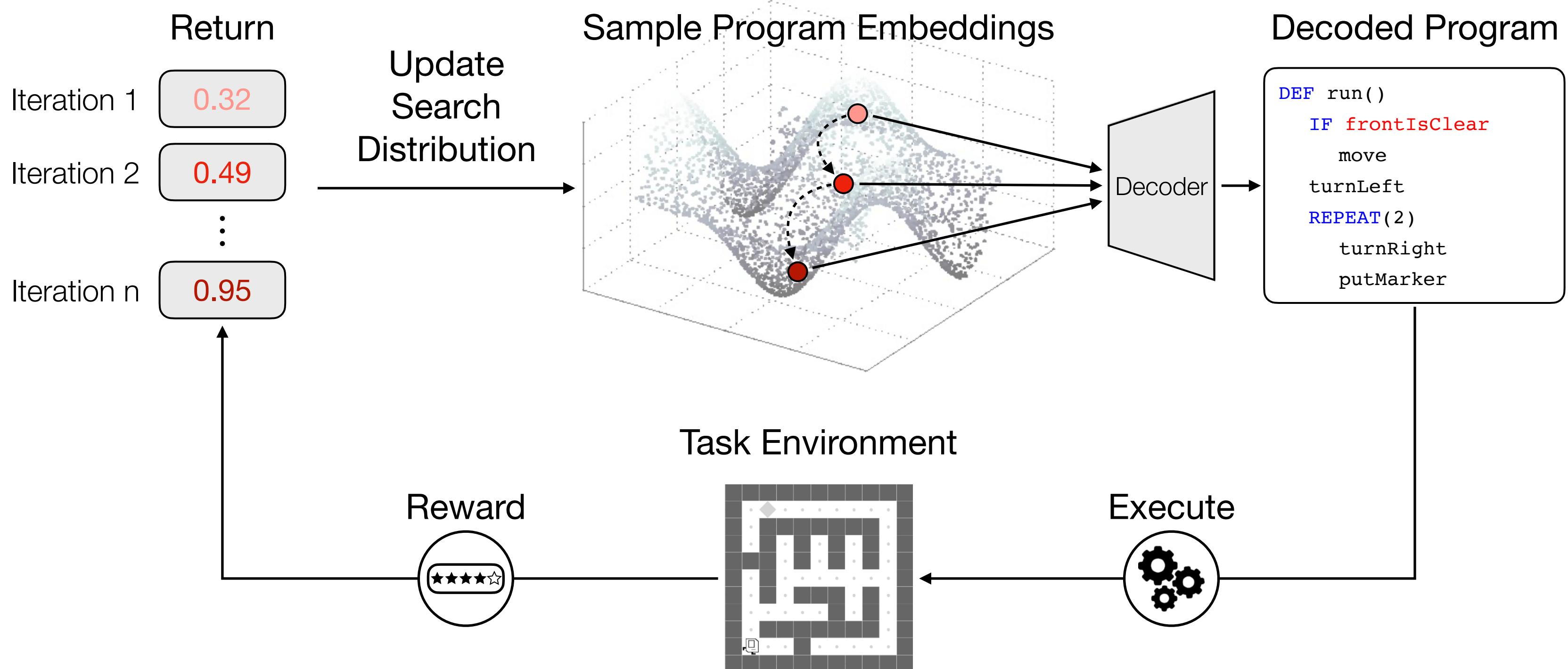
LEAPS - Learning a Program Embedding Space

Goal: Learn the grammar and the environment dynamics



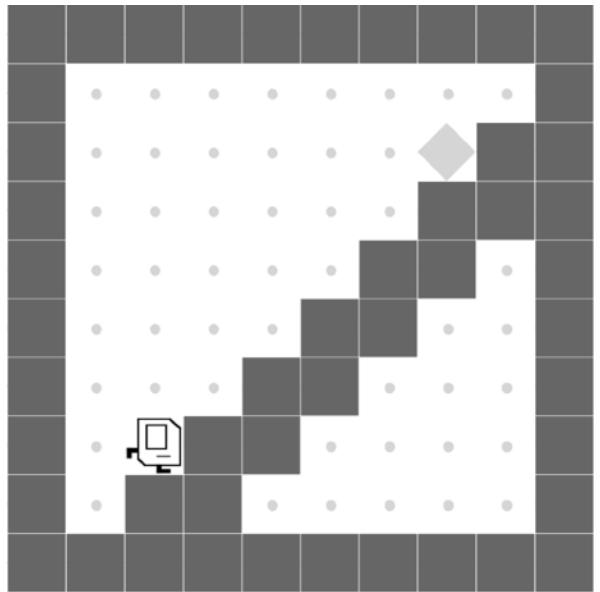
LEAPS - Latent Program Search

Search for a task-solving program using the cross-entropy method (CEM)

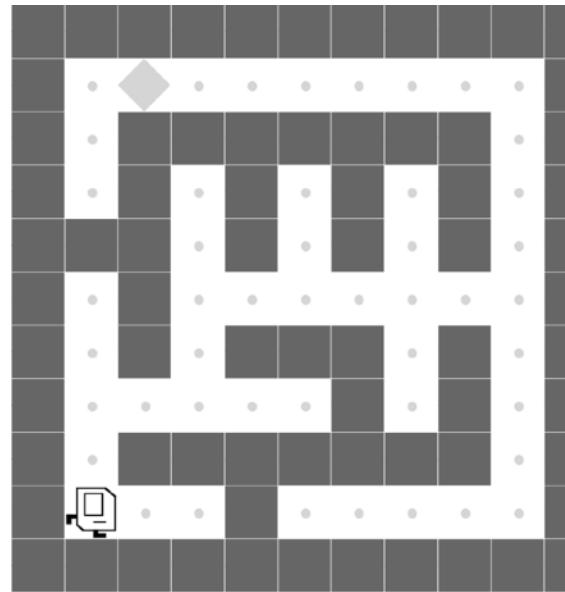


Karel Tasks

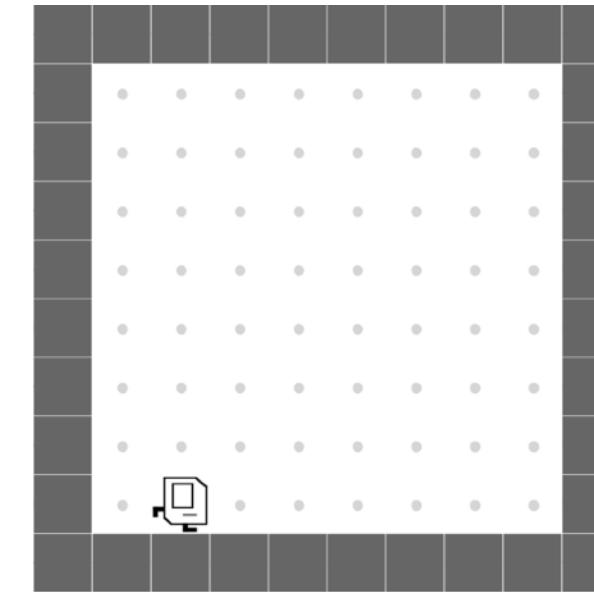
StairClimber



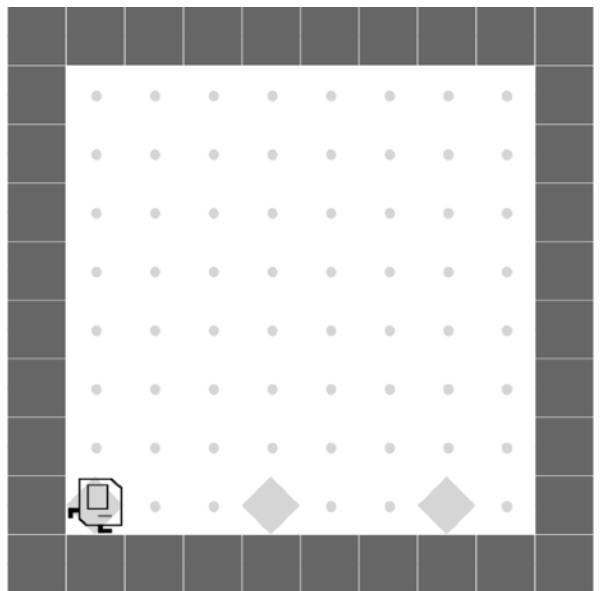
Maze



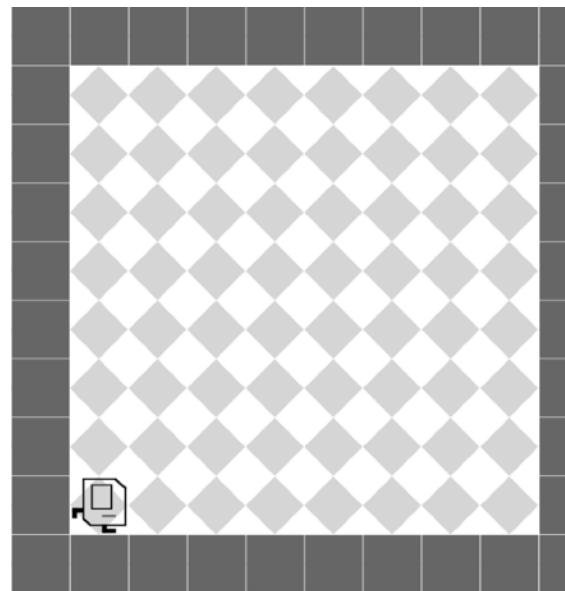
FourCorners



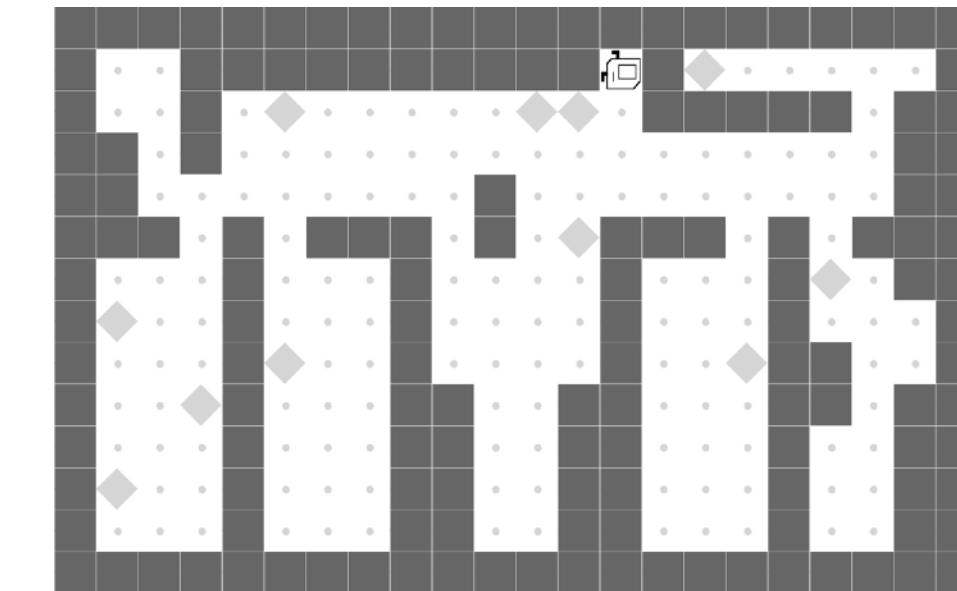
TopOff



Harvester

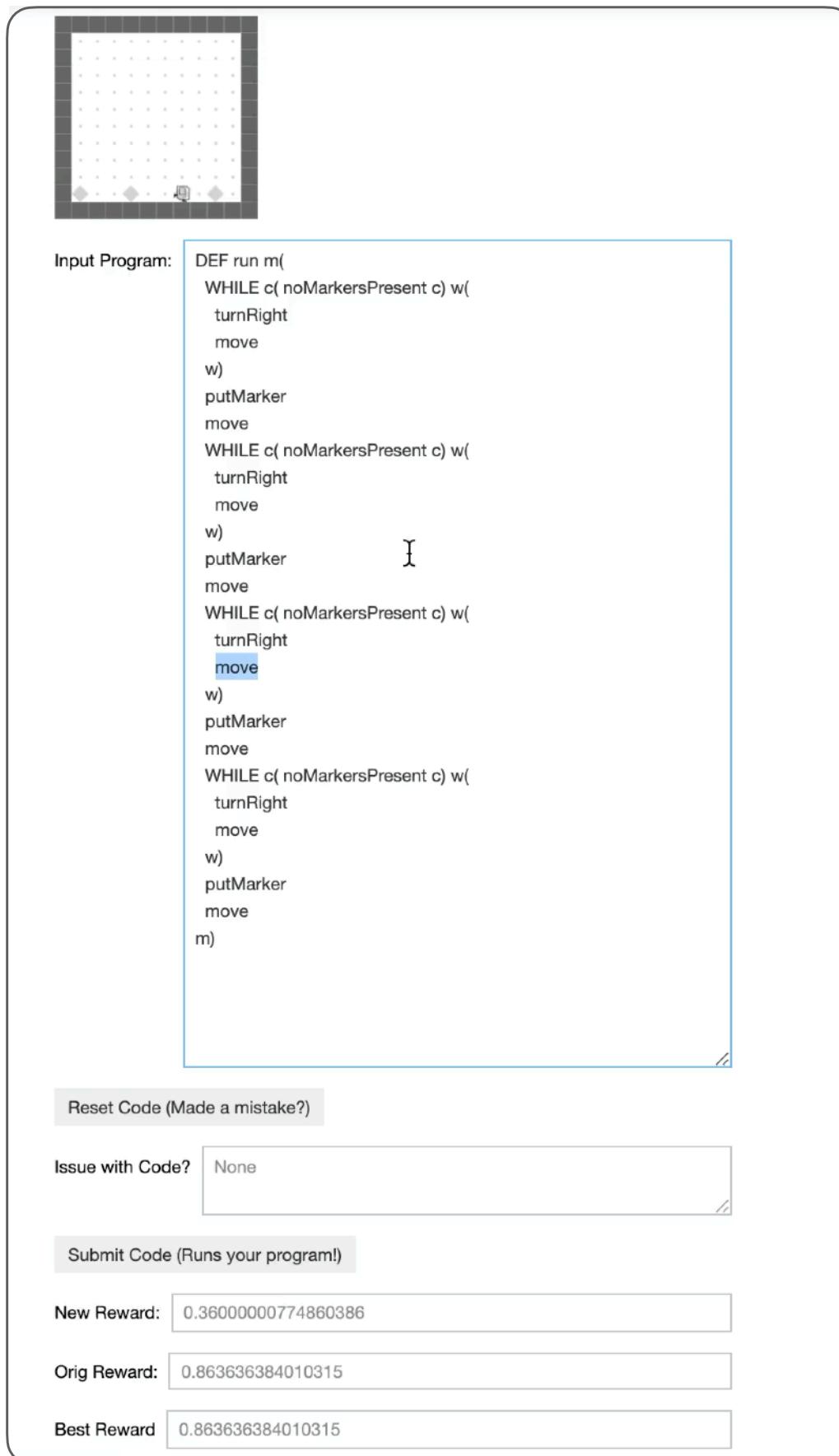


CleanHouse

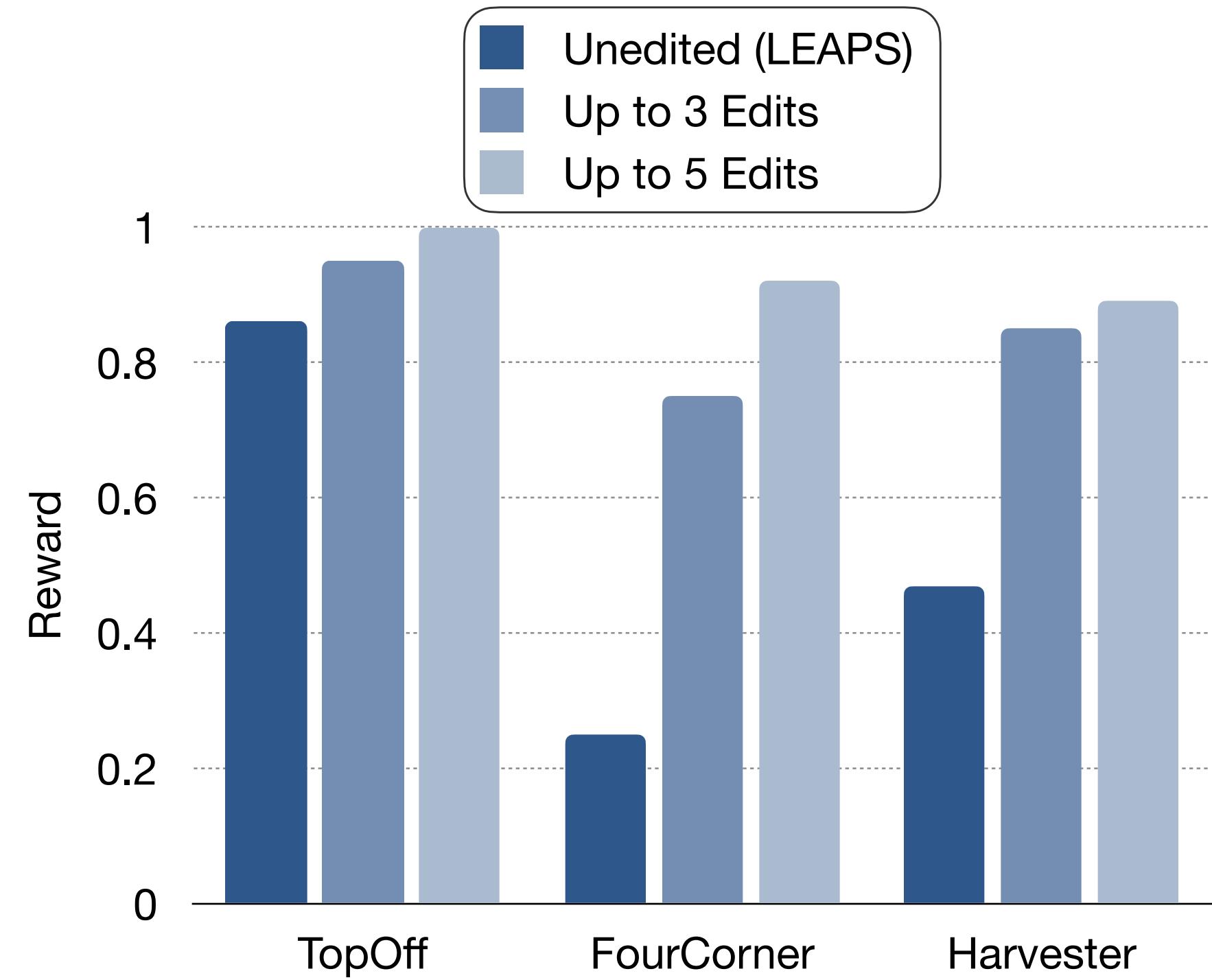


Interpretability & Interactivity

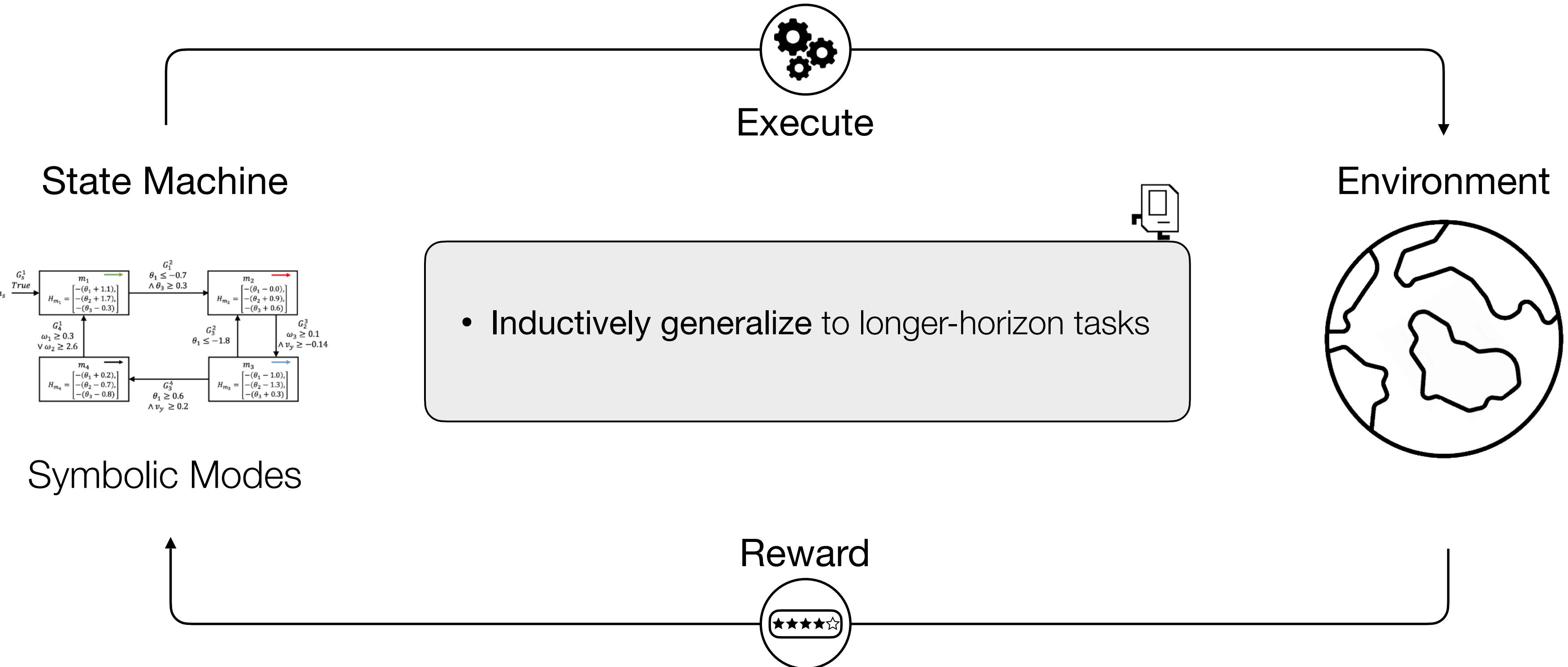
Interactive Debugging Interface



Performance Improvement



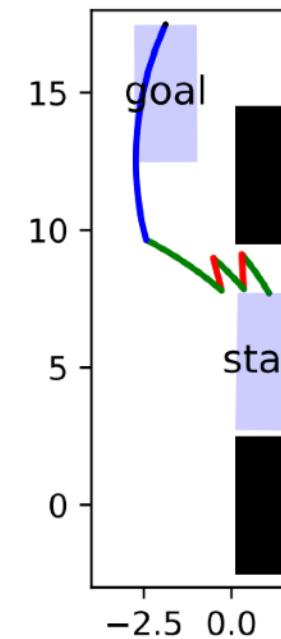
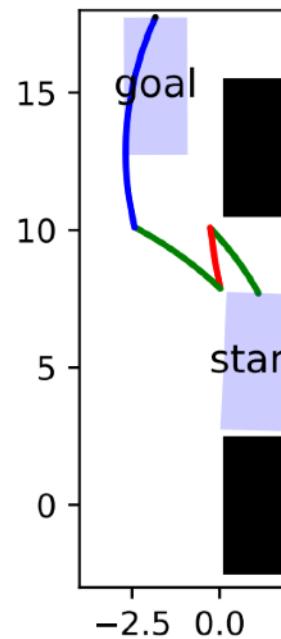
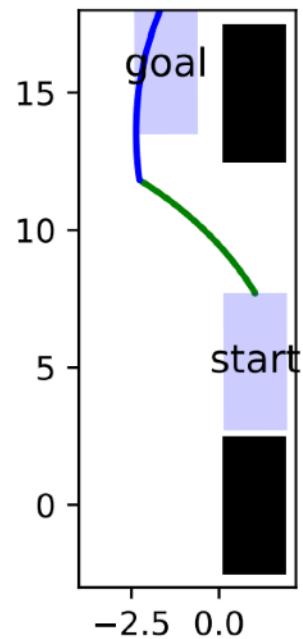
Programmatic Reinforcement Learning - State Machine



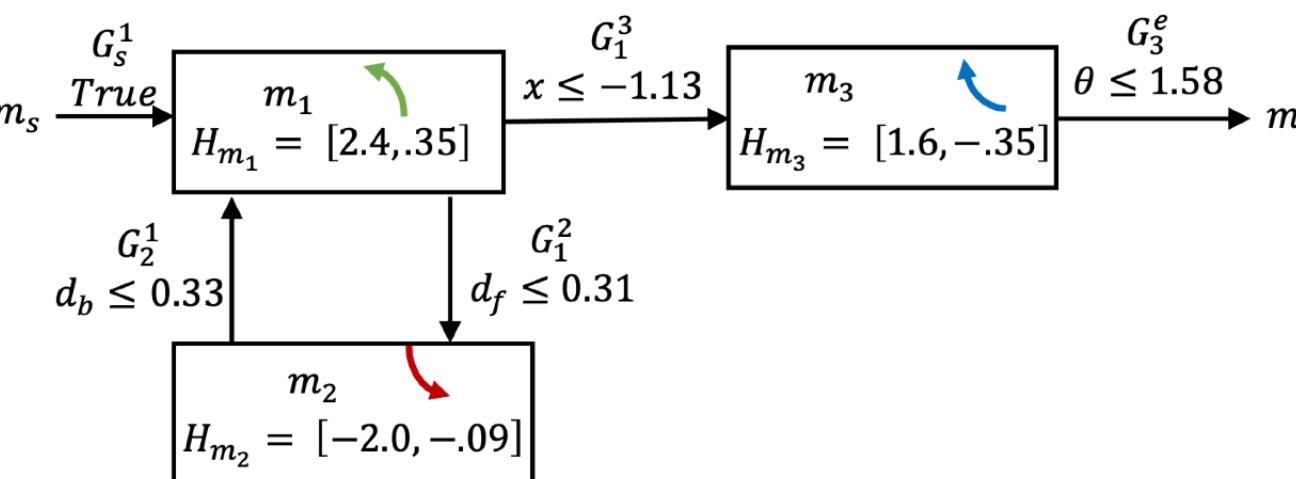
Inductive Generalization

Task: Retrieve a car from tight parking spots

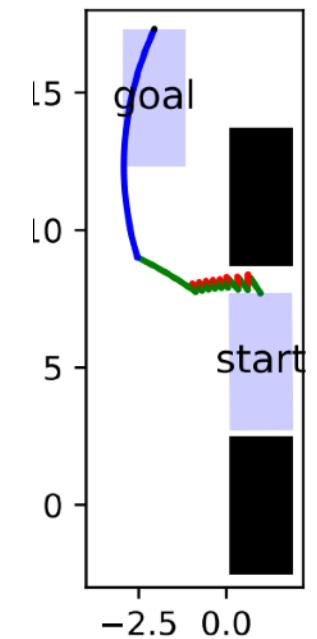
Training Tasks



Learned State Machine Policy
Symbolic Modes (i.e., actions) +
Transition Function



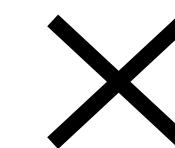
Testing Task



This Work: Integrating Program Synthesis and State Machine

Program

```
DEF run() m(
  WHILE c( markerPresent c) w(
    WHILE c( markerPresent c) w(
      pickMarker
      move w)
    turnRight
    move
    turnLeft
    WHILE c( markerPresent c) w(
      pickMarker
      move w)
    turnLeft
    move
    turnRight w) m)
```



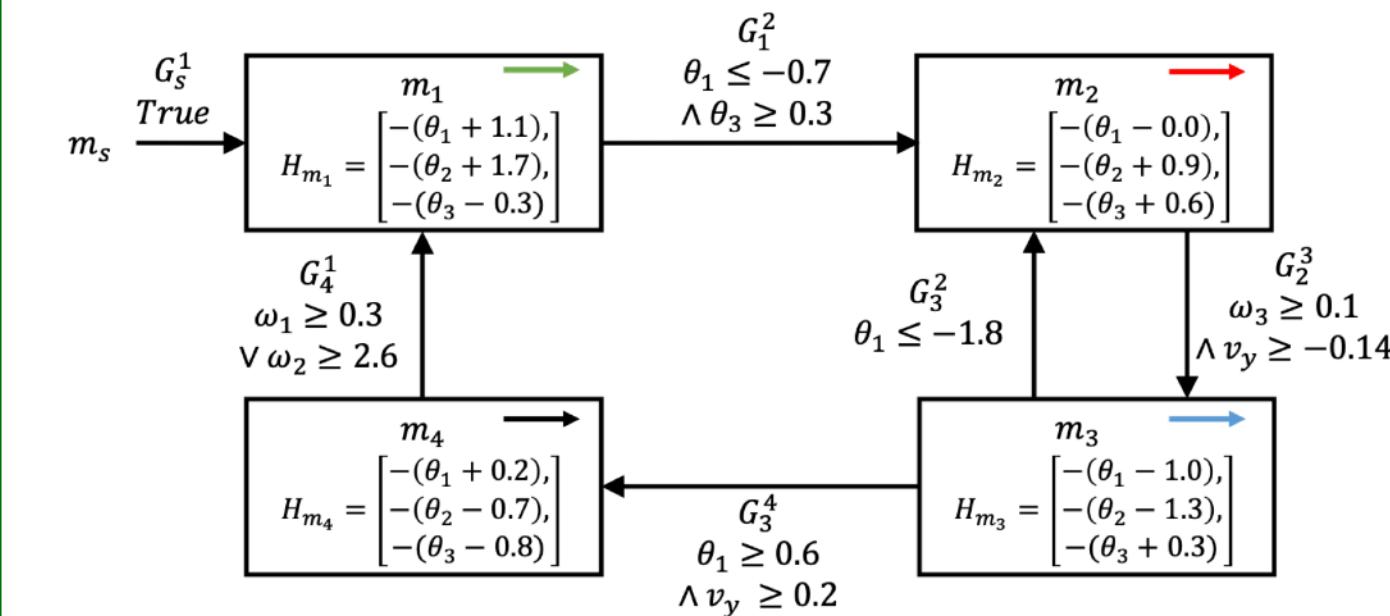
Pros

- Interpretable

Cons

- Often fail to generalize to longer-horizon tasks

State Machine



Pros

- Inductively generalizable

Cons

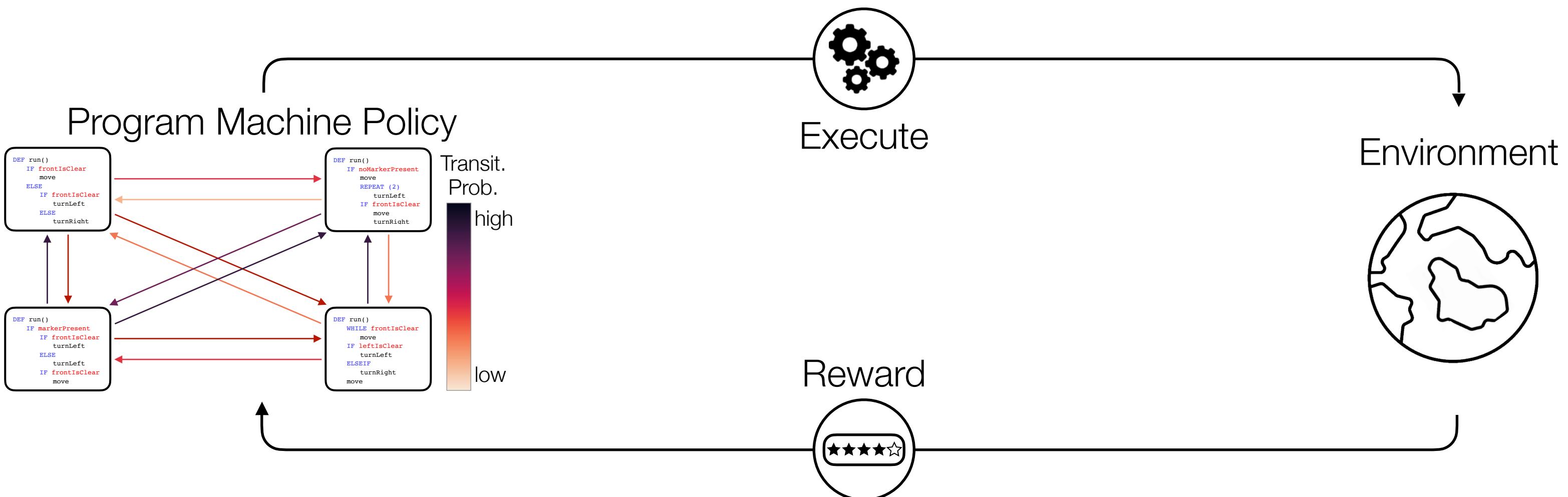
- Difficult to interpret

PrOgram Machine Policy (POMP)

Modes: A set of programs that can be executed

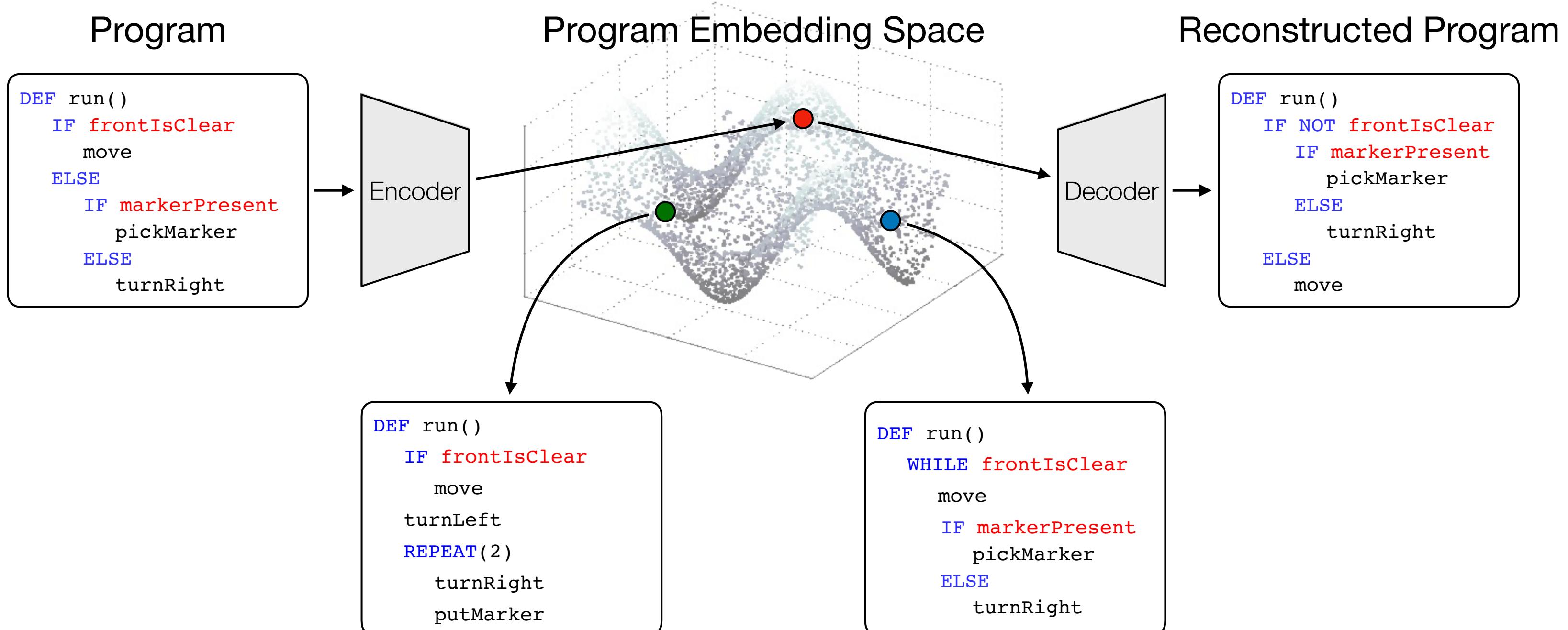
Transition function: Alter between a set of program modes

- Mapping: current environment state \times current mode \rightarrow next mode



Step 1 - Learning a Program Embedding Space

Goal: Learn the grammar and the environment dynamics



Step 2 - Retrieving Effective, Diverse, Compatible Programs

Goal: Retrieve a set of programs as state machine modes

Effectiveness: Retrieved programs should (partially) solve the task

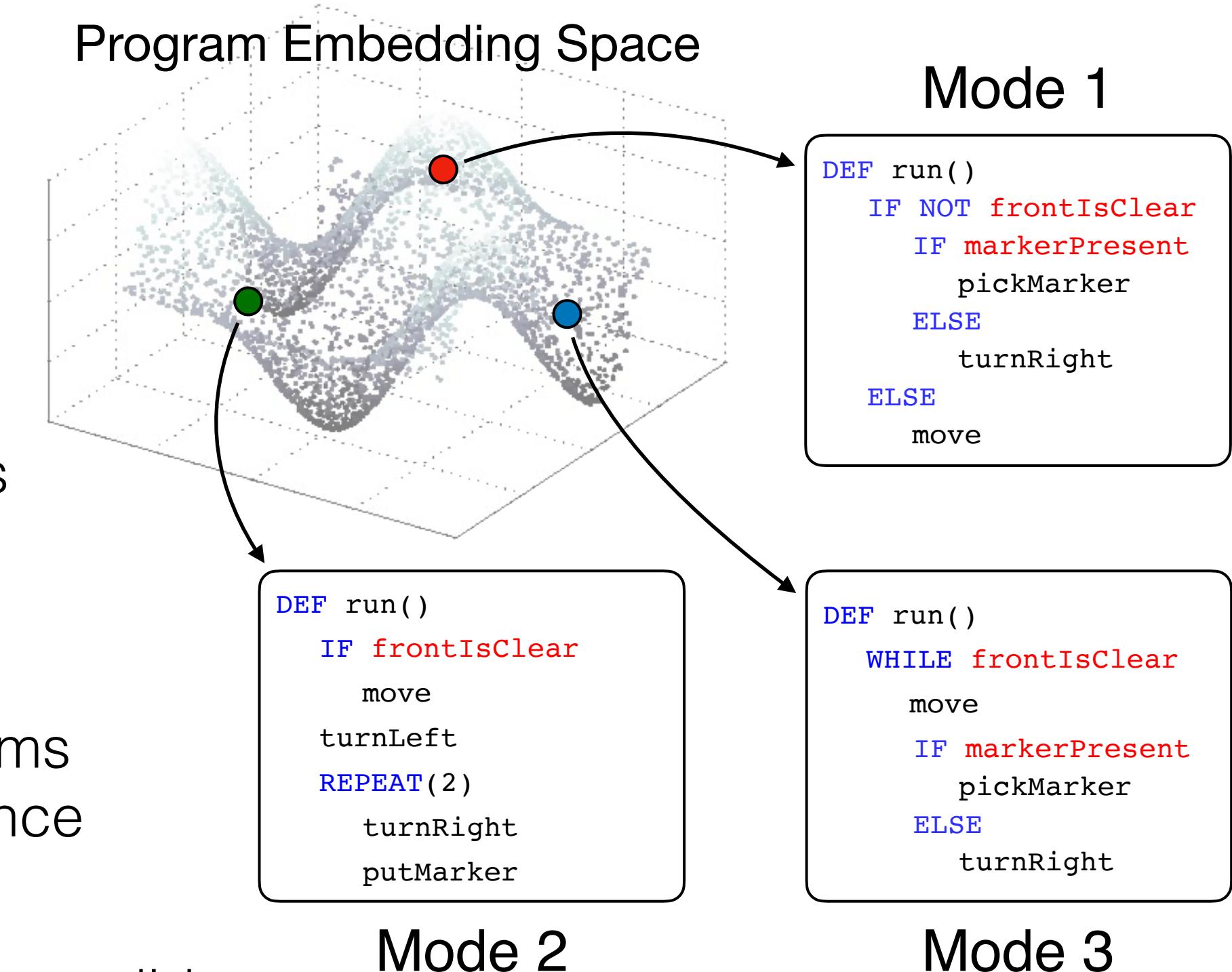
$$\rightarrow \text{Task reward: } \sum_{t=0}^T \gamma^t \mathbb{E}_{(s_t, a_t) \sim EXEC(\rho_z)}[r_t]$$

Diversity: Retrieved programs should induce non-overlapping, diverse behaviors

$$\rightarrow \text{Diversity bonus: } - \max_{z_i \in Z} \frac{z \cdot z_i}{\|z\| \|z_i\|}$$

Compatibility: Composing retrieved programs in some order should yield good performance

\rightarrow Randomly execute retrieved programs before/after executing the current program candidate



Step 3 - Learning Transition Function

Goal: Learn a transition function, i.e., current environment state \times current mode \rightarrow next mode

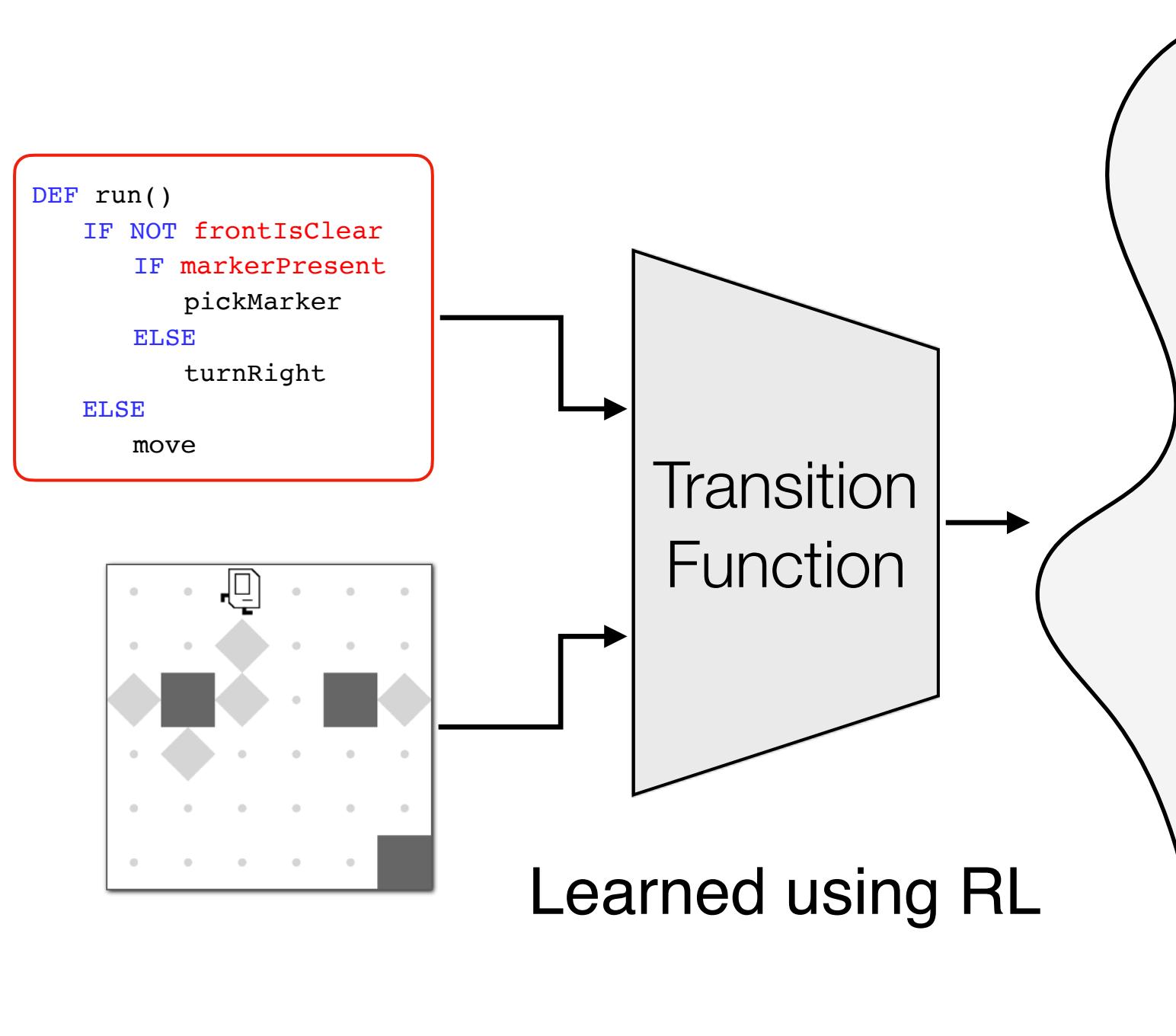
Modes (Retrieved Programs)

```
DEF run()
  IF NOT frontIsClear
    IF markerPresent
      pickMarker
    ELSE
      turnRight
  ELSE
    move
```

```
DEF run()
  WHILE frontIsClear
    move
    IF markerPresent
      pickMarker
    ELSE
      turnRight
```

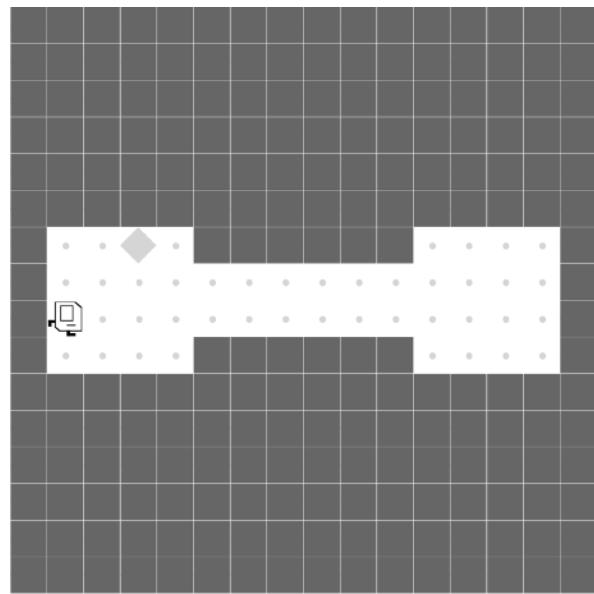
```
DEF run()
  IF frontIsClear
    move
    turnLeft
  REPEAT(2)
    turnRight
    putMarker
```

Transition Probability

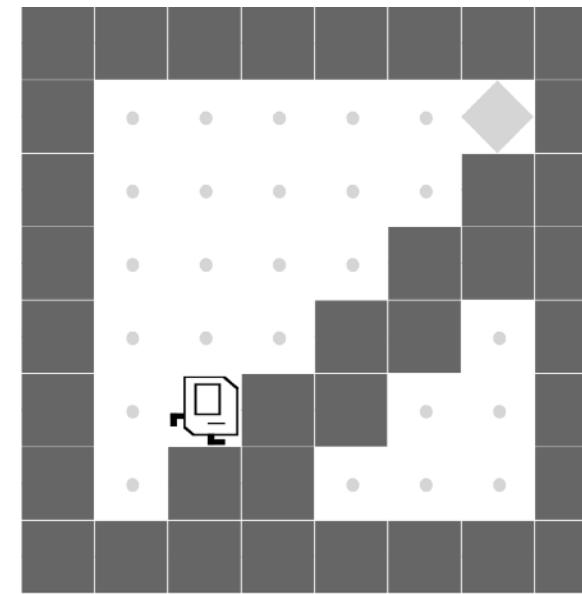


Long-Horizon Karel Tasks

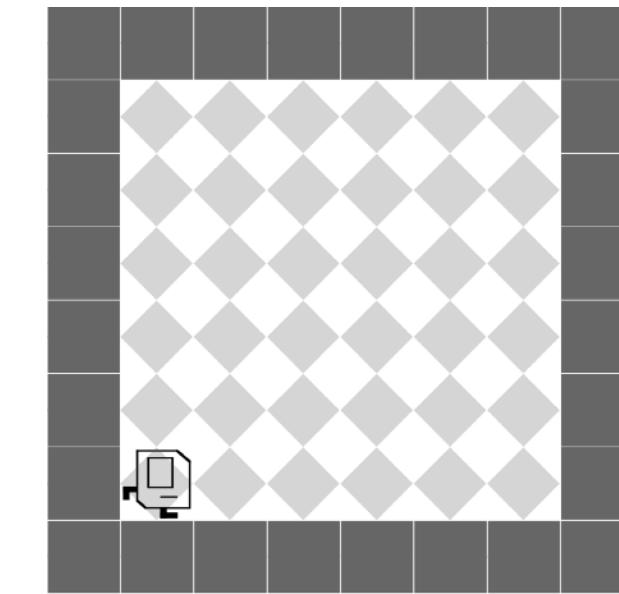
Seesaw



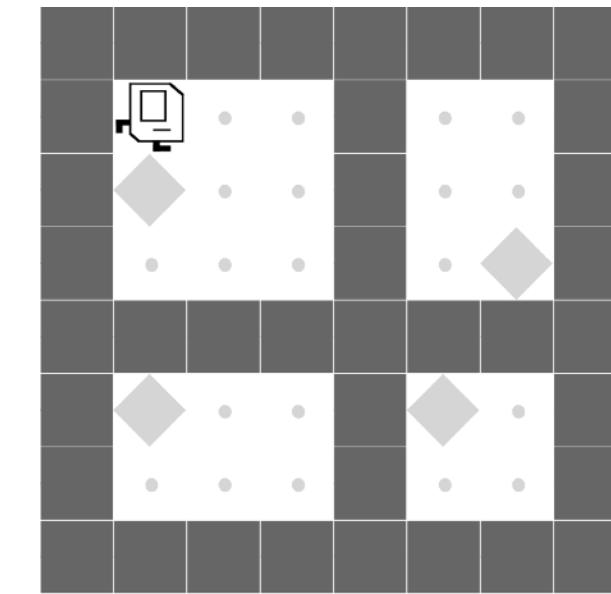
Up-N-Down



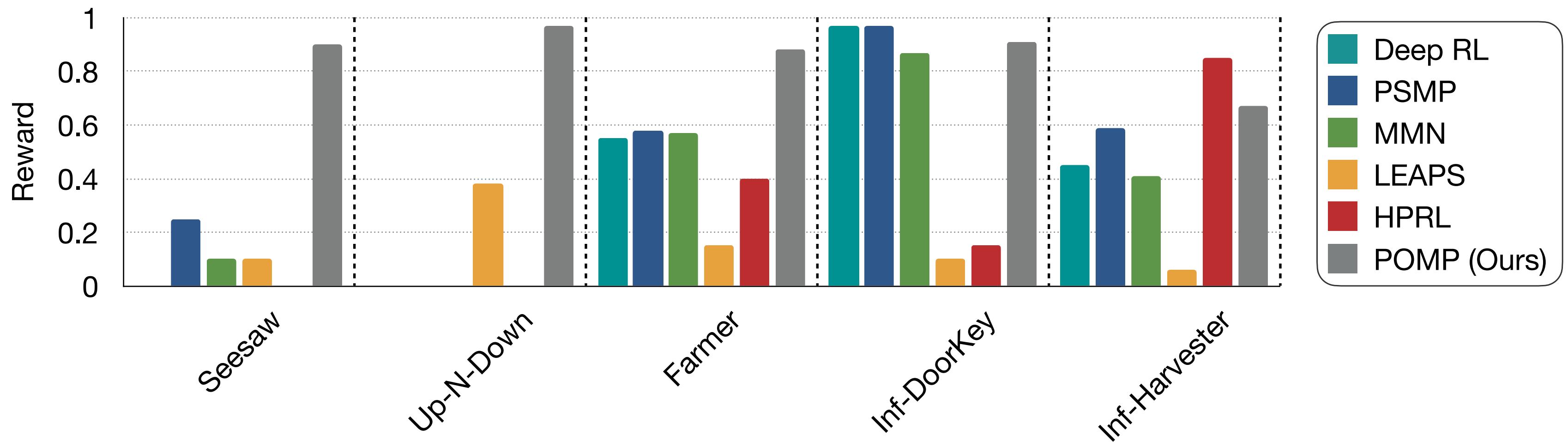
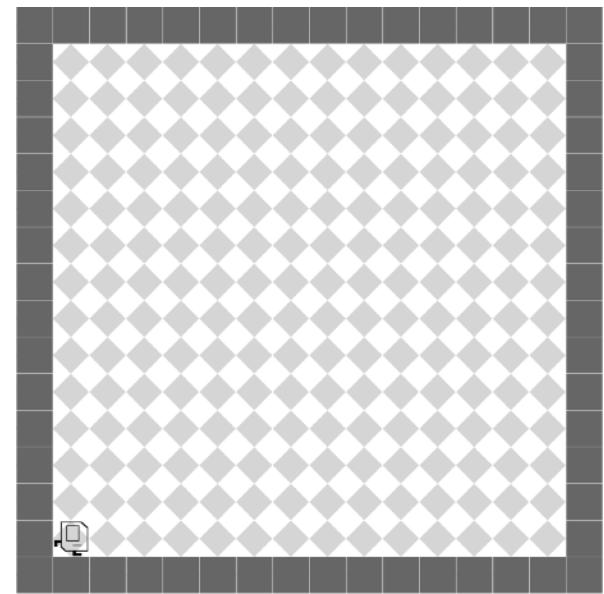
Farmer



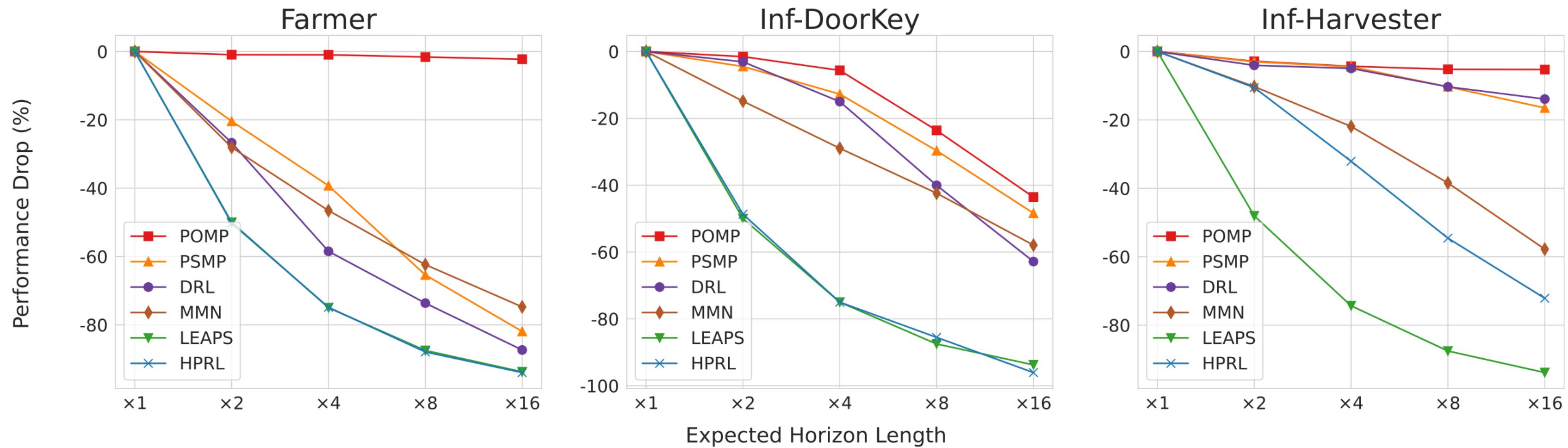
Inf-DoorKey



Inf-Harvester



Inductive Generalization



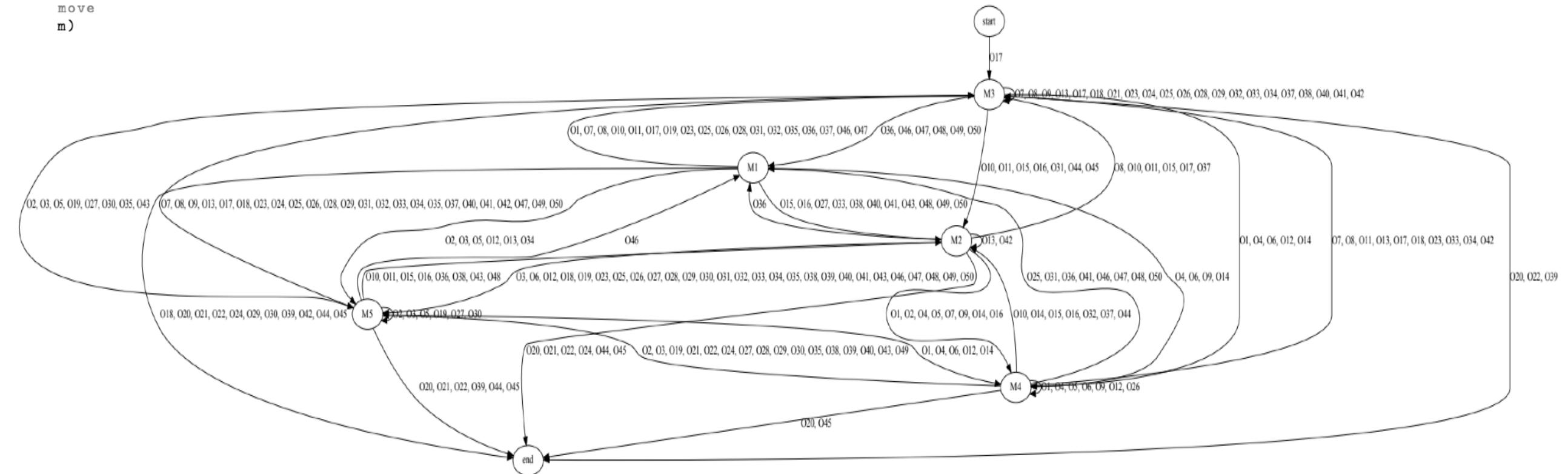
Learned Program Machine Policy

Task: Inf-DoorKey

Retrieved Programs

Mode 1	Mode 2	Mode 3	Mode 4	Mode 5
<pre>DEF run m(WHILE c(noMarkersPresent c) w(WHILE c(noMarkersPresent c) w(turnRight move w) pickMarker putMarker w) WHILE c(rightIsClear c) w(move w) WHILE c(rightIsClear c) w(move w) pickMarker putMarker move m)</pre>	<pre>DEF run m(WHILE c(noMarkersPresent c) w(move turnLeft w) WHILE c(noMarkersPresent c) w(move turnLeft m)</pre>	<pre>DEF run m(move turnLeft move turnLeft move turnLeft m)</pre>	<pre>DEF run m(pickMarker IF c(not c(leftIsClear c) c) i(move i) IF c(not c(leftIsClear c) c) i(turnRight i) m)</pre>	<pre>DEF run m(putMarker REPEAT R=13 r(pickMarker turnRight move putMarker move pickMarker move turnRight move r) move pickMarker move move m)</pre>

Extracted State Machine

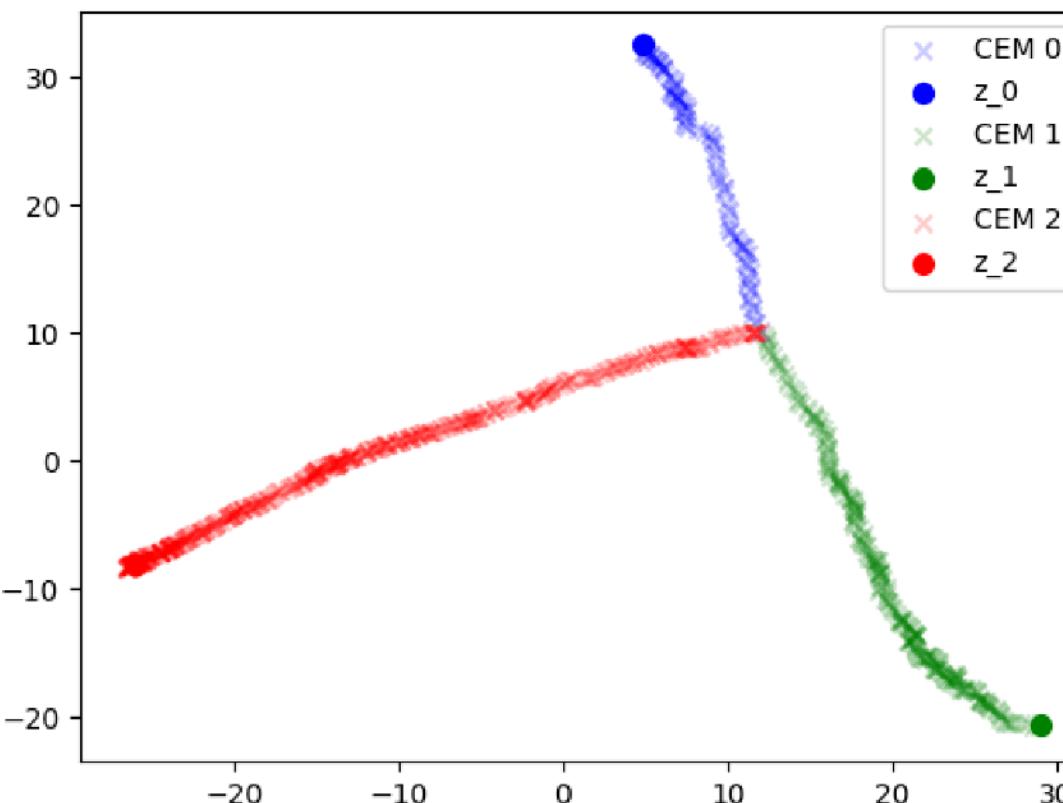


Ablation Study

Diversity & Compatibility

Method	SEESAW	UP-N-DOWN	FARMER	INF-DOORKEY	INF-HARVESTER
CEM $\times M $	0.31 ± 0.18	0.88 ± 0.12	0.22 ± 0.00	0.39 ± 0.46	0.56 ± 0.12
CEM+diversity top k , $k = M $	0.09 ± 0.11	0.72 ± 0.36	0.23 ± 0.00	0.92 ± 0.01	0.71 ± 0.02
CEM+diversity $\times M $	0.47 ± 0.39	0.76 ± 0.31	0.24 ± 0.03	0.89 ± 0.02	0.66 ± 0.07
POMP (Ours)	0.90 ± 0.02	0.97 ± 0.00	0.88 ± 0.01	0.91 ± 0.01	0.67 ± 0.03

CEM Search Trajectories with Diversity Bonus



Addressing Long-Horizon Tasks by Integrating Program Synthesis and State Machines

Yi-Chen Lin¹, Chen-Jui Lin¹, Gang-Lin Lin¹, Pei-Yang Lin¹, Ming-Chuan Lin¹¹National Taiwan University

✉Correspondence:

yiclin@csie.ntu.edu.tw

Abstract

Deep reinforcement learning (DRL) has been widely applied to solve tasks in various domains. Despite encouraging results, these methods are limited in due to their lack of explicit knowledge representation and reasoning. To address this issue, we propose two novel methods, namely *Karel*-based program synthesis and state machines, to effectively handle long-horizon tasks. The first method, *Karel*-based program synthesis, is able to learn a programmatic policy directly from raw sensor data. The second method, state machines, is able to learn a state transition function to represent the environment dynamics. These two methods are integrated to form a hybrid system, which can effectively handle long-horizon tasks.

1 Introduction

Deep reinforcement learning (DRL) has been widely applied to solve tasks in various domains. Despite encouraging results, these methods are limited in due to their lack of explicit knowledge representation and reasoning. To address this issue, we propose two novel methods, namely *Karel*-based program synthesis and state machines, to effectively handle long-horizon tasks. The first method, *Karel*-based program synthesis, is able to learn a programmatic policy directly from raw sensor data. The second method, state machines, is able to learn a state transition function to represent the environment dynamics. These two methods are integrated to form a hybrid system, which can effectively handle long-horizon tasks.

2 Related Work

Program Synthesis. Program synthesis techniques involve solving program generation to convert high-level specifications into executable programs. There are two main categories of program synthesis: deductive synthesis and inductive synthesis.

Programmatic Reinforcement Learning. Programmatic reinforcement learning (PRL) is a technique that uses learned programs to solve reinforcement learning problems. It has been applied to various domains, such as robotics, games, and control systems. PRL has shown promising results in solving complex tasks, such as navigation, planning, and decision-making.

Long-Horizon Tasks. Long-horizon tasks are tasks that require agents to make decisions over extended time periods. Solving long-horizon tasks is challenging due to the difficulty of predicting future states and rewards. Various approaches have been proposed to handle long-horizon tasks, such as hierarchical reinforcement learning, programmatic reinforcement learning, and state machine learning.

3 Problem Formulation

We define a task as a function $T: \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$, where \mathcal{S} is the state space, \mathcal{A} is the action space, and \mathcal{R} is the reward space. A programmatic policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$ is a function that maps a state to an action. We denote the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π .

4.1 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.2 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.3 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.4 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.5 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.6 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.7 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.8 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.9 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.10 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.11 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.12 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.13 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.14 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.15 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.16 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.17 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.18 Learning Function

We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

$$\pi_t(a|s) = \pi_{t-1}(a|s) + \alpha \nabla_{\theta} V_{\pi_{t-1}}(s)$$

where α is the learning rate.

4.19 Learning Function

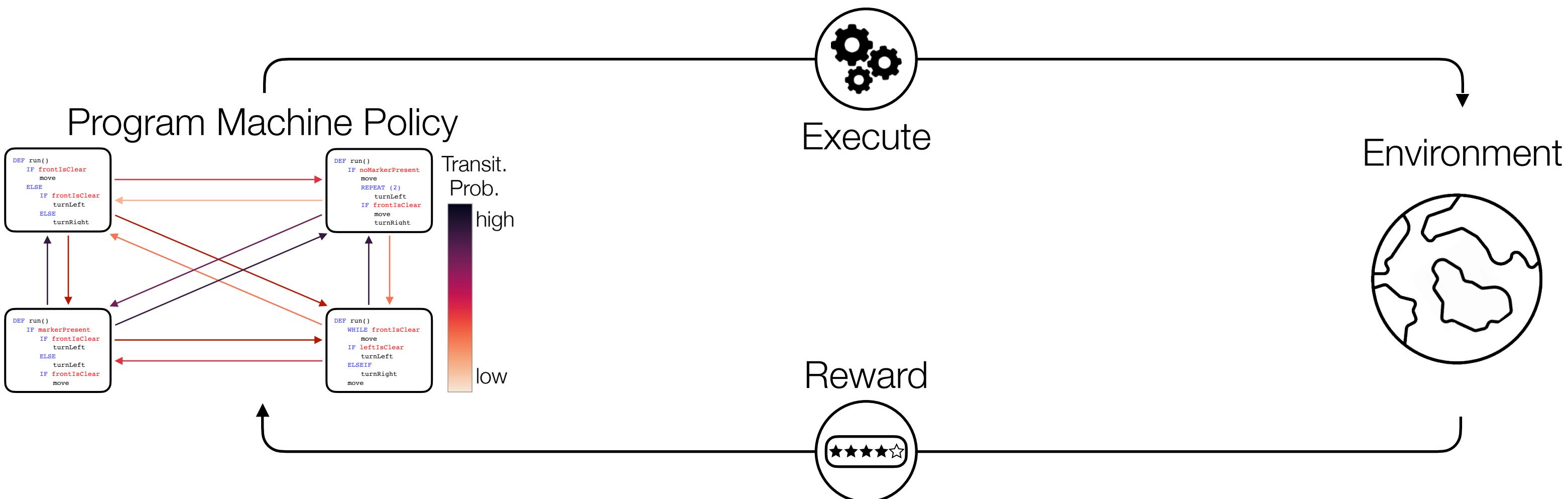
We follow the approach of the programmatic policy π to learn a programmatic policy π . We first define the value function of a programmatic policy π as $V_\pi(s)$, which represents the expected return starting from state s under policy π . We then define the policy gradient as $\nabla_{\theta} V_\pi(s)$, where θ is the parameter of the programmatic policy π . The policy gradient is used to update the programmatic policy π as follows:

<math

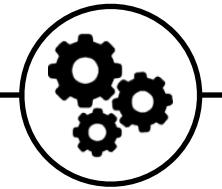
Takeaway

Program Synthesis \times State Machine =

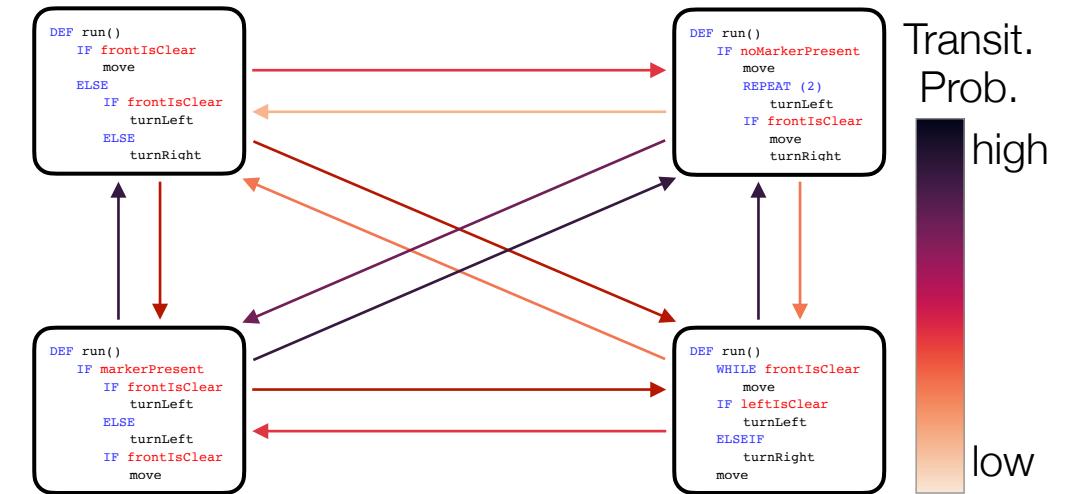
Interpretable and Inductively Generalizable Policies



Execute



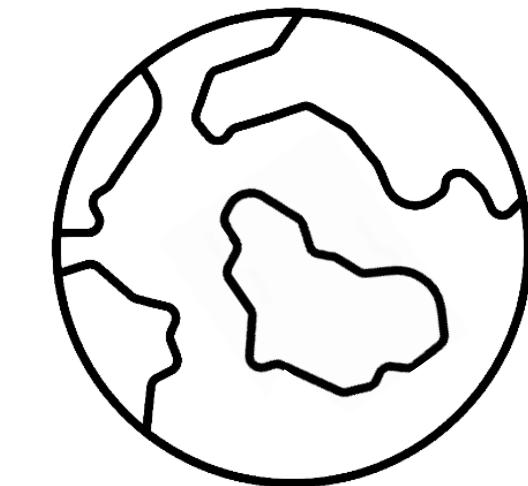
Program Machine Policy



Thank You

Questions?

Environment



Reward