APPENDIX

A EXTENDED RELATED WORK

Program Synthesis. Program synthesis techniques revolve around program generation to convert given inputs into desired outputs. These methods have demonstrated notable successes across diverse domains such as array and tensor manipulation(Balog et al., 2017; Ellis et al., 2020), string transformation(Devlin et al., 2017; Hong et al., 2021; Zhong et al., 2023), generating computer command (Lin et al., 2018) and code (Chen et al., 2021; Li et al., 2022), graphics and 3D shape modeling(Wu et al., 2017; Liu et al., 2019; Tian et al., 2019), and describing agent behaviors(Bunel et al., 2018; Sun et al., 2018; Chen et al., 2019; Silver et al., 2020; Liang et al., 2022). Most program synthesis methods focus on task specifications such as input/output pairs, demonstrations, or language descriptions; in contrast, this work aims to synthesize human-readable programs as policies to solve reinforcement learning tasks.

Hierarchical Reinforcement Learning and Semi-Markov Decision Processes. HRL frameworks (Sutton et al., 1999; Barto & Mahadevan, 2003; Vezhnevets et al., 2017; Bacon et al., 2017) focus on learning and operating across different levels of temporal abstraction, enhancing the efficiency of learning and exploration, particularly in sparse-reward environments. In this work, Our proposed Program Machine Policy shares the same spirit and some ideas with HRL frameworks if we view the transition function as a “high-level” policy and the set of mode programs as “low-level” policies or skills. While most HRL frameworks either pre-define and pre-learned low-level policies, or jointly learn the high-level and low-level policies from scratch, our proposed framework first retrieves a set of effective, diverse, and compatible modes (*i.e.*, programs), and then learns the mode transition function.

The POMP framework also resembles the Option framework Sutton et al. (1999); Bacon et al. (2017); Klissarov & Precup (2021). More specifically, one can characterise POMP as using interpretable options as sub-policies since there is a high-level neural network being used to pick among retrieved programs as described in Section 4.3. Note that besides interpretable options, our work still differs from the option frameworks in the following aspects. Our work first retrieves a set of mode programs and then learns a transition function; this differs from most option frameworks that jointly learn options and a high-level policy that chooses options. Also, the transition function in our work learns to terminate, while the high-level policy option frameworks do not.

On the other hand, based on the definition of the recursive optimality described in Dietterich (2000), POMP can be categorized as recursively optimal since it is locally optimal given the policies of its children. Specifically, one can view the mode program retrieval process of POMP as solving a set of subtasks based on the proposed CEM-based search method that considers effectiveness, diversity, and compatibility. Then, POMP learns a transition function according to the retrieved programs, resulting in a policy as a whole. We have revised the paper to include this point of view.

Symbolic Planning for Long-Horizon Tasks. Another line of research uses symbolic operators (Yang et al., 2018; Guan et al., 2022; Cheng & Xu, 2023) for long-horizon planning. The major difference between POMP and Cheng & Xu (2023) and Guan et al. (2022) is the interpretability of the skill or option. In POMP, each learned skill is represented by a human-readable program. On the other hand, neural networks used in Cheng & Xu (2023) and tabular approaches used in Guan et al. (2022) are used to learn the skill policies. In Yang et al. (2018), the option set is assumed as input without learning and cannot be directly compared with Cheng & Xu (2023), Guan et al. (2022) and POMP.

Another difference between the proposed POMP framework, Cheng & Xu (2023), Guan et al. (2022), and Yang et al. (2018) is whether the high-level transition abstraction is provided as input. In Cheng & Xu (2023), a library of skill operators is taken as input and serves as the basis for skill learning. In Guan et al. (2022), the set of “landmarks” is taken as input to decompose the task into different combinations of subgoals. In PEORL (Yang et al., 2018), the option set is taken as input, and each option has a 1-1 mapping with each transition in the high-level planning. On the other hand, the proposed POMP framework utilized the set of retrieved programs as modes, which is conducted based on the reward from the target task without any guidance from framework input.

B Details of the Cross Entropy Method

B.1 CEM

The pseudo-code of CEM is as follow:

Algorithm 1 Cross Entropy Method

```
procedure CEM($G$, $g$, $\mathcal{P} = \mathcal{N}$, $N_s = 1000$, $n = 64$, $\sigma = 0.1$, $e = 0.1$)
    $z_r \leftarrow [z_0, z_1, ..., z_i, ..., z_{255}], z_i \sim \mathcal{P}$
    $step \leftarrow 0$
    while $step < N_s$ do
        $Z \leftarrow [\,]$
        $L_G \leftarrow [\,]$
        for $i \leftarrow 1$ to $n$ do
            $\varepsilon \leftarrow [\varepsilon_0, \varepsilon_1, ..., \varepsilon_i, ..., \varepsilon_{255}], \varepsilon_i \sim \mathcal{N}(0, \sigma_n)$
            $Z \leftarrow Z + [z_r + \varepsilon]$
            $L_G \leftarrow L_G + [G(Z[i-1], g)]$
        end for
        $R^{kl} \leftarrow$ KthLargest$(L_G, n \cdot e)$
        $Z^{kl} \leftarrow [\,]$
        for $i \leftarrow 0$ to $n - 1$ do
            if $L_G[i] \leq R^{kl}$ then
                $Z^{kl} \leftarrow Z^{kl} + [Z[i]]$
            end if
        end for
        $z_r \leftarrow mean(Z^{kl})$
        $step \leftarrow step + 1$
    end while
end procedure
```

G is the evaluation function, g is the input of G, $\mathcal{P}$ is the distribution initial vector sampled from, N_s is the maximum number of the iteration, n is the population size, σ is the standard deviation of the noise added to z_r, and e is the percent of the population elites.

B.2 CEM+Diversity

The procedure of running CEM+diversity N times is as follows:

(1) Search the $1st$ program embedding z_1 by $CEM(G, g = (\{\} \cup Z_k, \Psi_{\text{before}}, \Psi_{\text{after}}))$

(2) Search the $2nd$ program embedding z_2 by $CEM(G, g = (\{z_1\} \cup Z_k, \Psi_{\text{before}}, \Psi_{\text{after}}))$
...

(N) Search the Nth program embedding z_N by $CEM(G, g = (\{z_1, ..., z_{N-1}\} \cup Z_k, \Psi_{\text{before}}, \Psi_{\text{after}}))$

In the simplest case, Z_k is a empty set, Ψ_{before} and Ψ_{after} are empty sequences. It will then reduce to the process described in Section 4.2.2. An example of the searching trajectories can be seen in Figure 7.

B.3 CEM+Diversity+Compatibility

B.3.1 Sample Program Sequence

The procedure $H(z_1, z_2, z_3, ..., z_k)$ for sampling program embedding sequences SQ_1 and SQ_2 in Figure 3d from a total of k programs is as follows:

(1) Randomly choose from the following options:

a. With $(1 - \frac{1}{k+1})$ probability, sample a program embedding from Z_y with replacement. Decode it to a program and append this program to Ψ.

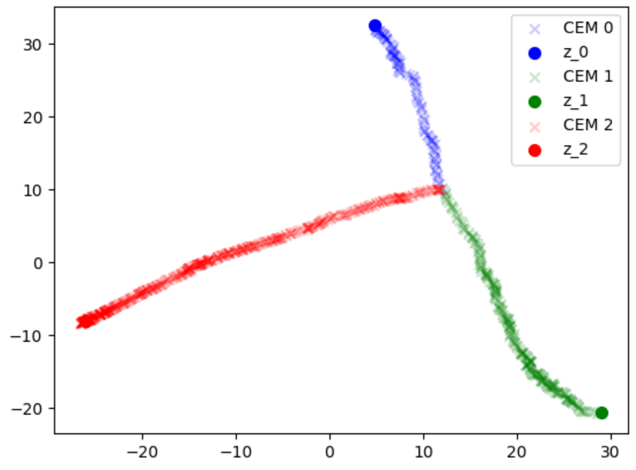


Figure 7: **CEM+Diversity Searching Trajectories.** It shows the trajectories of the procedure of running CEM+diversity 3 times. The program embeddings searched during the CEM are reduced to 2-dimensional embeddings using PCA. Since the diversity design, the $2nd$ CEM is forced to explore the opposite direction related to the searching path of the $1st$ CEM, and the $3rd$ CEM is urged to search a path that is perpendicular to the $1st$ and $2nd$ searching paths.

b. With $\frac{1}{k+1}$ probability, stop sampling.

(2) Repeat (1) until the stop option is chosen.

After SQ_1 and SQ_2 are sampled, they are then be decoded to be Ψ_{before} and Ψ_{after}, respectively.

B.3.2 WHOLE PROCEDURE

The procedure of running CEM+diversity+Compatibility $|M|$ times in order to retrieve $|M|$ mode programs is as follows:

(1) Retrieve $1st$ mode program z_1.

a. Run CEM+diversity N times with $Z_k = \{\}$, $\Psi_{\text{before}} = []$ and $\Psi_{\text{after}} = []$ to get N program embeddings.

b. Choose the program embedding with the highest $G(z, \{\}, \Psi_{\text{before}}, \Psi_{\text{after}})$ among the N program embeddings as z_1.

(2) Retrieve $2nd$ mode program z_2.

a. Sample SQ_1 and SQ_2 from $H(z_1)$, and deocde SQ_1 and SQ_2 to Ψ_{before} and Ψ_{after}, respectively.

b. Run CEM+diversity N times with $Z_k = \{z_1\}$, to get N program embeddings.

c. Choose the program embedding with the highest $G(z, \{\}, \Psi_{\text{before}}, \Psi_{\text{after}})$ among the N program embeddings as z_2.

...

($|M|$) Retrieve $|M|th$ mode program $z_{|M|}$.

a. Sample SQ_1 and SQ_2 from $H(z_1, z_2, ..., z_{|M|-1})$, and decode SQ_1 and SQ_2 to Ψ_{before} and Ψ_{after}, respectively.

b. Run CEM+diversity N times with $Z_k = \{z_1, z_2, ..., z_{|M|-1}\}$, to get N program embeddings.

c. Choose the program embedding with the highest $G(z, \{\}, \Psi_{\text{before}}, \Psi_{\text{after}})$ among the N program embeddings as $z_{|M|}$.

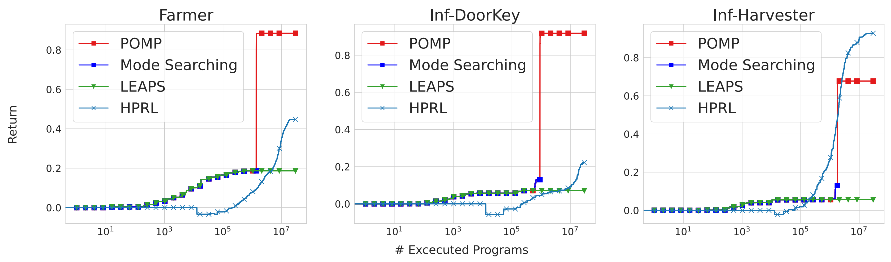


Figure 8: **Program Sample Efficiency.** Results of different programmatic RL approaches in FARMER, INF-DOORKEY, and INF-HARVESTER.

C PROGRAM SAMPLE EFFICIENCY

During the training of programmatic RL approaches, programs will be synthesized and executed in the environment of a given task to evaluate whether these programs are good enough. This three-step procedure (synthesis, execution, and evaluation) will be repeatedly done until the return converges or the maximum training steps are reached. The purpose of the analysis of the program sample efficiency is to figure out how many times this procedure needs to be done to achieve a certain return. As shown in Figure 8, POMP has the best program sample efficiencies in FARMER and INF-DOORKEY, but lower program sample efficiency than HPRL in INF-HARVESTER. The detail of the return calculation for each approach is described below.

C.1 POMP

During the mode program searching process of POMP, a total of 265 CEMs are done for searching 5 mode programs. In each CEM, a maximum of 1000 iterations will be done, and in each iteration of the CEM, n (population size of the CEM) times of the three-step procedure are done. The return of a certain number of executed programs in the first half of the figure is recorded as the maximum return obtained from executing the previously searched programs solely.

During the transition function training process, the three-step procedure is done once in each PPO training step. The return of a certain number of executed programs in the remainder of the figure is recorded as the maximum validation return obtained by POMP.

C.2 LEAPS

During the program searching process of LEAPS, the CEM is used for searching the program, and the hyperparameters of the CEM are tuned. A total of 216 CEMs are done. In each CEM, a maximum of 1000 iterations will be done, and in each iteration of the CEM, n (population size of the CEM) times of the three-step procedure are done. The return of a certain number of executed programs in the figure is recorded as the maximum return obtained from executing the previously searched programs solely.

C.3 HPRL

During the meta-policy training process of HPRL, the three-step procedure is done once in each PPO training step. Therefore, with the setting of the experiment described in Section F.6, as the training is finished, the three-step procedure will be done 25M times. The return of a certain number of executed programs in the figure is recorded as the maximum return obtained from the cascaded execution of 10 programs, which are decoded from latent programs output by the meta-policy.

D INDUCTIVE GENERALIZATION

To test the ability of the inductive generalization of different methods, we scale up the expected horizon of the environment by increasing the upper limit of the target for each KAREL-LONG task. To elaborate, using INF-DOORKEY as an example, the upper limit number of marker-picking and marker-placing is 16 under a training environment setting. Therefore, all policies are trained to

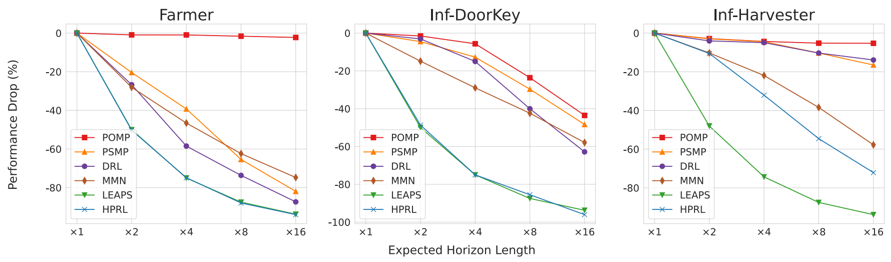


Figure 9: **Inductive Generalization.** Experiment Results on different baselines in FARMER, INF-DOORKEY, and INF-HARVESTER.

terminate after 16 markers are picked and placed. However, the upper limit number is set to 32, 64, etc, in the testing environment.

Since most of the baselines don't perform well on SEESAW and UP-N-DOWN, we do the inductive generalization experiments mainly on FARMER, INF-DOORKEY, and INF-HARVESTER. The expected horizon lengths of the testing environments will be 2, 4, 8, and 16 times longer compared to the training environment setting. Also, rewards gained from picking or placing markers and the penalty for actions are divided by 2, 4, 8, and 16 to normalize the maximum total reward of the tasks to 1. The detailed setting and the experiment result for each of these three tasks are shown as follows.

D.1 FARMER

During the training phases of our and other baseline methods, we set the maximum iteration number to 2. However, we adjust this number for the testing phase to 4, 8, 16, and 32. As shown in Figure 9, when the expected horizon length grows, the performances of all the baselines except POMP drop dramatically, which means that our method has a much better inductive generalization property on this task.

D.2 INF-DOORKEY

During the training phases of our and other baseline methods, we set the upper limit number of marker-picking and marker-placing to 16. However, we adjust this number for the testing phase to 32, 64, 128, and 256. As shown in Figure 9, when the expected horizon length grows, the performances of all the baselines drop considerably. Nevertheless, POMP has a minor performance drop compared to other baselines on this task.

D.3 INF-HARVESTER

During the training phases of our and other baseline methods, we set the emerging probability to $\frac{1}{2}$. However, we adjust this number for the testing phase to $\frac{3}{4}$, $\frac{7}{8}$, $\frac{15}{16}$ and $\frac{31}{32}$. As shown in Figure 9, when the expected horizon length grows, the performances of POMP, PSMP, and DRL drop slightly, but the performances of MMN, LEAPS and HPRL drop extensively. Overall, POMP has a minimum performance drop among all baselines on this task.

E STATE MACHINE EXTRACTION

In our approach, since we employ the neural network transition function, the proposed program machine policies are only partially or locally interpretable – once the transition function selects a mode program, human users can read and understand the following execution of the program.

To further increase the interpretability of the trained mode transition function f, we extracted the state machine structure by the approach proposed in (Koul et al., 2019). In this setup, since POMP utilizes the previous mode as one of the inputs and predicts the next mode, we focus solely on encoding the state observations. Each state observation is extracted by convolutional neural networks and fully connected layers to a 1×128 vector, which is then quantized into a 1×5 vector. We can construct a state-transition table using these quantized vectors and modes. The final step involves minimizing these quantized vectors, which allows us to represent the structure of the state machine effectively. Examples of extracted state machine are shown in Figure 10, Figure 11 and Figure 12.

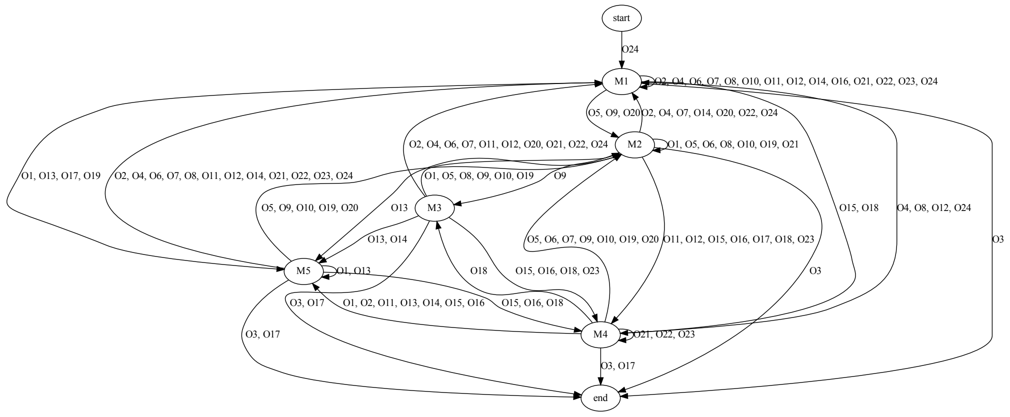

start
O24
M1
O2, O4, O6, O7, O8, O10, O11, O12, O14, O16, O21, O22, O23, O24
O5, O9, O20
O2, O4, O7, O14, O20, O22, O24
O2, O4, O6, O7, O11, O12, O20, O21, O22, O24
M2
O1, O5, O6, O8, O10, O19, O21
O1, O13, O17, O19
O2, O4, O6, O7, O8, O11, O12, O14, O21, O22, O23, O24
O1, O5, O8, O9, O10, O19
O9
O5, O9, O10, O19, O20
O13
M3
O15, O18
O4, O8, O12, O24
O3
O13, O14
O5, O6, O7, O9, O10, O19, O20
O11, O12, O15, O16, O17, O18, O23
M5
O1, O13
O18
O15, O16, O18, O23
O3
O3, O17
O1, O2, O11, O13, O14, O15, O16
O15, O16, O18
O3, O17
M4
O21, O22, O23
O3, O17
end

Figure 10: **Example of extracted state machine on FARMER**. $O1$ to $O24$ represent the unique quantized vectors encoded from observations. The corresponding mode programs of $M1$ to $M5$ are displayed in Figure 22.

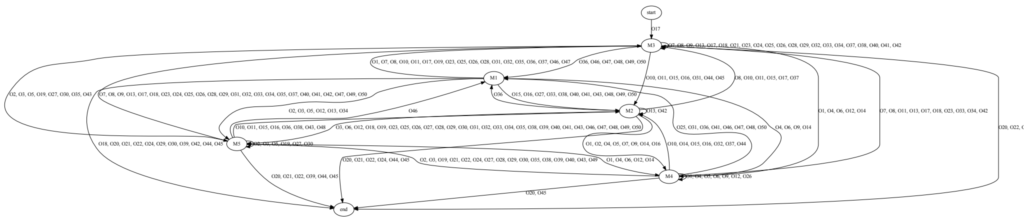


Figure 11: **Example of extracted state machine on INF-DOORKEY**. $O1$ to $O50$ represent the unique quantized vectors encoded from observations. The corresponding mode programs of $M1$ to $M5$ are displayed in Figure 22.

F Hyperparameters and Settings of Experiments

F.1 POMP

Encoder & Decoder. We follow the training procedure and the model structure proposed in Trivedi et al. (2021), which uses recurrent networks to implement both the encoder q_ϕ and the decoder p_θ with hidden dimensions of 256 and trains them on programs randomly sampled from the Karel DSL. The model are updated through PPO (Schulman et al., 2017) algorithm and trained to optimize the β-VAE Higgins et al. (2016), the program behavior reconstruction loss, and the latent behavior reconstruction loss described in Trivedi et al. (2021).

The program dataset consists of 35,000 programs for training and 7,500 programs for validation and testing that were randomly sampled from the Karel DSL. We sequentially sample program tokens for each random program based on defined probabilities until an ending token or when a maximum program length is reached. The defined probability of each kind of token are listed below:

- `WHILE`: 0.15
- `REPEAT`: 0.03
- `STMT_STMT`: 0.5

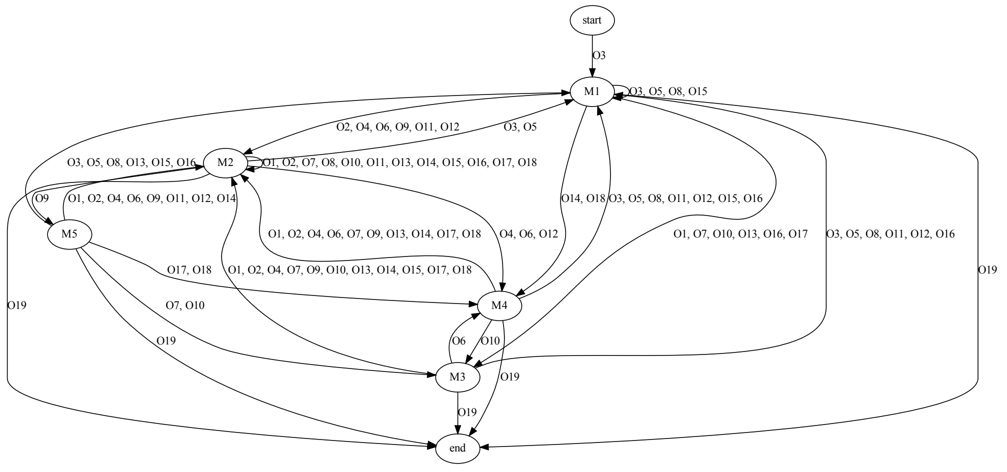

start
O3
M1
O3, O5, O8, O15
O2, O4, O6, O9, O11, O12
O3, O5
O3, O5, O8, O13, O15, O16
M2
O1, O2, O7, O8, O10, O11, O13, O14, O15, O16, O17, O18
O9
O1, O2, O4, O6, O9, O11, O12, O14
O14, O18
O3, O5, O8, O11, O12, O15, O16
M5
O1, O2, O4, O6, O7, O9, O13, O14, O17, O18
O4, O6, O12
O1, O7, O10, O13, O16, O17
O3, O5, O8, O11, O12, O16
O17, O18
O1, O2, O4, O7, O9, O10, O13, O14, O15, O17, O18
O19
O19
M4
O19
O7, O10
O6
O10
M3
O19
O19
O19
end

Figure 12: **Example of extracted state machine on INF-HARVESTER**. $O1$ to $O19$ represent the unique quantized vectors encoded from observations. The corresponding mode programs of $M1$ to $M5$ are displayed in Figure 23.

- `ACTION`: 0.2
- `IF`: 0.08
- `IFELSE`: 0.04

where `STMT_STMT` represents dividing the current token into two separate tokens, each chosen based on the same probabilities defined above. This token primarily dictates the length of the programs, as well as the quantity and complexity of nested loops and statements.

Mode Program Synthesis. To increase the diversity of the set of mode programs, we conduct 10 CEM searches in CEM+diversity to synthesize each mode program, and each program machine policy uses 5 modes from 5 CEM+diversity searches in the experiments. Only the hyperparameter set of the first CEM in the whole process is tuned. The rest of the CEMs use the same hyperparameter set as the first. The range of the hyperparameters is the same as Section F.5.

Mode Transition Function. The mode transition function f consists of convolutional layers (Fukushima & Miyake, 1982; Krizhevsky et al., 2017) to derive features from the Karel states and the fully connected layers to predict the transition probabilities among each mode. Meanwhile, we utilize one-hot encoding to represent the current mode index of the program machine policy. The detail setting of the convolutional layers is the same as those described in Section F.4. The training process of the mode transition function f can be optimized using the PPO (Schulman et al., 2017) algorithm. The hyperparameters are listed below:

- Maximum program number: 1000
- Batch size : 128
- Clipping: 0.05
- α: 0.99
- γ: 0.99
- GAE lambda: 0.95
- Value function coefficient: 0.5
- Entropy coefficient: 0.1
- Number of updates per training iteration: 4
- Number of environment steps per set of training iterations: 32
- Number of parallel actors: 32

- Optimizer : Adam
- Learning rate: {0.1, 0.01, 0.001, 0.0001, 0.00001}

F.2 PSMP

It resembles the setting in Section F.1. The input, output, and structure of the mode transition function f remain the same. However, the 5 modes programs are replaced by 5 primitive actions (`move`, `turnLeft`, `turnRight`, `putMarker`, `pickMarker`).

F.3 DRL

DRL training on the Karel environment uses the PPO (Schulman et al., 2017) algorithm with 20 million timesteps. Both the policies and value networks share a convolutional encoder that interprets the state of the grid world. This encoder comprises two layers: the initial layer has 32 filters, a kernel size of 4, and a stride of 1, while the subsequent layer has 32 filters, a kernel size of 2, and maintains the same stride of 1. The policies will predict the probability distribution of primitive actions (move, turnLeft, turnRight, putMarker, pickMarker) and termination. During our experiments with DRL on KAREL-LONG tasks, we fixed most of the hyperparameters and did hyperparameter grid search over learning rates. The hyperparameters are listed below:

- Maximum horizon: 10000
- Batch size : 128
- Clipping: 0.05
- α: 0.99
- γ: 0.99
- GAE lambda: 0.95
- Value function coefficient: 0.5
- Entropy coefficient: 0.1
- Number of updates per training iteration: 4
- Number of environment steps per set of training iterations: 32
- Number of parallel actors: 32
- Optimizer : Adam
- Learning rate: {0.1, 0.01, 0.001, 0.0001, 0.00001}

F.4 MMN

Aligned with the approach described in Koul et al. (2019), we trained and quantized a recurrent policy with a GRU cell and convolutional neural network layers to extract information from gird world states. During our experiments with MMN on KAREL-LONG tasks, we fixed most of the hyperparameters and did a hyperparameter grid search over learning rates. The hyperparameters are listed below:

- Hidden size of GRU cell: 128
- Number of quantized bottleneck units for observation: 128
- Number of quantized bottleneck units for hidden state: 16
- Maximum horizon: 10000
- Batch size : 128
- Clipping: 0.05
- α: 0.99
- γ: 0.99
- GAE lambda: 0.95
- Value function coefficient: 0.5

- Entropy coefficient: 0.1
- Number of updates per training iteration: 4
- Number of environment steps per set of training iterations: 32
- Number of parallel actors: 32
- Optimizer : Adam
- Learning rate: {0.1, 0.01, 0.001, 0.0001, 0.00001}

F.5 LEAPS

In line with the setup detailed in Trivedi et al. (2021), we conducted experiments over various hyperparameters of the CEM to optimize rewards for LEAPS. The hyperparameters are listed below:

- Population size (n): {8, 16, 32, 64}
- σ: {0.1, 0.25, 0.5}
- e: {0.05, 0.1, 0.2}
- Exponential σ decay: {True, False}
- Initial distribution $\mathcal{P}$: $\{\mathcal{N}(1, 0), \mathcal{N}(0, \sigma), \mathcal{N}(0, 0.1\sigma)\}$

F.6 HPRL

In alignment with the approach described in Liu et al. (2023), we trained the meta policy for each task to predict a program sequence. To adapt this method to tasks with longer horizons, we increased the number of programs from 5 to 10. The hyperparameters are listed below:

- Max subprogram: 10
- Max subprogram Length: 40
- Batch size : 128
- Clipping: 0.05
- α: 0.99
- γ: 0.99
- GAE lambda: 0.95
- Value function coefficient: 0.5
- Entropy coefficient: 0.1
- Number of updates per training iteration: 4
- Number of environment steps per set of training iterations: 32
- Number of parallel actors: 32
- Optimizer : Adam
- Learning rate: 0.00001
- Training steps: 25M

G Details of Karel Problem Set

The KAREL problem set is presented in Trivedi et al. (2021), consisting of the following tasks: STAIRCLIMBER, FOURCORNER, TOPOFF, MAZE, CLEANHOUSE and HARVESTER. Figure 13 and Figure 14 provide visual depictions of a randomly generated initial state, an internal state sampled from a legitimate trajectory, and the desired final state for each task. The experiment results presented in Table 1 and Table 3 are evaluated by averaging the rewards obtained from 32 randomly generated initial configurations of the environment.

G.1 STAIRCLIMBER

This task takes place in a 12×12 grid environment, where the agent's objective is to successfully climb the stairs and reach the marked grid. The marked grid and the agent's initial location are both randomized at certain positions on the stairs, with the marked grid being placed on the higher end of the stairs. The reward is defined as 1 if the agent reaches the goal in the environment, -1 if the agent moves off the stairs, and 0 otherwise.

G.2 FOURCORNER

This task takes place in a 12×12 grid environment, where the agent's objective is to place a marker at each of the four corners. The reward received by the agent will be 0 if any marker is placed on the grid other than the four corners. Otherwise, the reward is calculated by multiplying 0.25 by the number of corners where a marker is successfully placed.

G.3 TOPOFF

This task takes place in a 12×12 grid environment, where the agent's objective is to place a marker on every spot where there's already a marker in the environment's bottom row. The agent should end up in the rightmost square of this row when the rollout concludes. The agent is rewarded for each consecutive correct placement until it either misses placing a marker where one already exists or places a marker in an empty grid on the bottom row.

G.4 MAZE

This task takes place in an 8×8 grid environment, where the agent's objective is to find a marker by navigating the grid environment. The location of the marker, the initial location of the agent, and the configuration of the maze itself are all randomized. The reward is defined as 1 if the agent successfully finds the marker in the environment, 0 otherwise.

G.5 CLEANHOUSE

This task takes place in a 14×22 grid environment, where the agent's objective is to collect as many scattered markers as possible. The initial location of the agent is fixed, and the positions of the scattered markers are randomized, with the additional condition that they will only randomly be scattered adjacent to some wall in the environment. The reward is defined as the ratio of the collected markers to the total number of markers initially placed in the grid environment.

G.6 HARVESTER

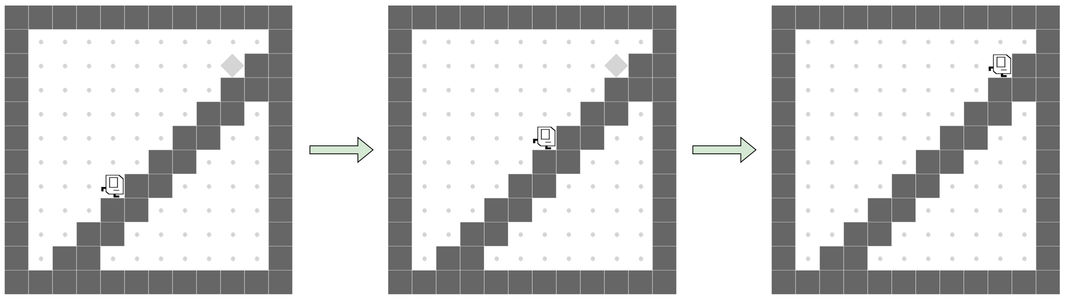
This task takes place in an 8×8 grid environment, where the environment is initially populated with markers appearing in all grids. The agent's objective is to pick up a marker from each location within this grid environment. The reward is defined as the ratio of the picked markers to the total markers in the initial environment.

H DETAILS OF KAREL-HARD PROBLEM SET

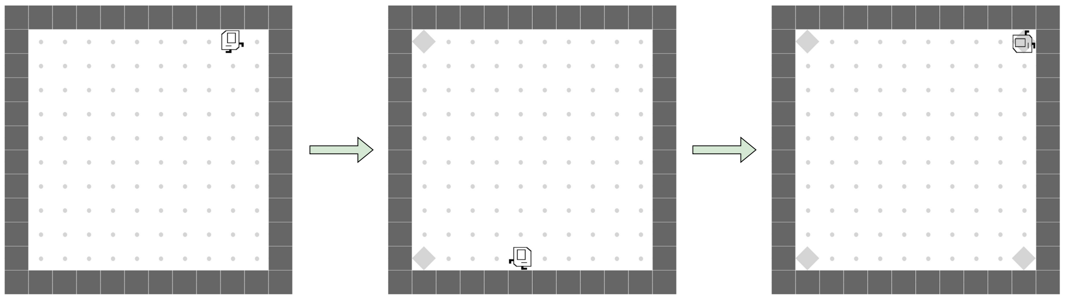
The KAREL-HARD problem set proposed by Liu et al. (2023) consists of the following tasks: DOORKEY, ONESTROKE, SEEDER and SNAKE. Each task in this benchmark is designed to have more constraints and be more structurally complex than tasks in the KAREL problem set. Figure 15 provides a visual depiction of a randomly generated initial state, some internal state(s) sampled from a legitimate trajectory, and the desired final state for each task. The experiment results presented in Table 1 and Table 3 are evaluated by averaging the rewards obtained from 32 randomly generated initial configurations of the environment.

H.1 DOORKEY

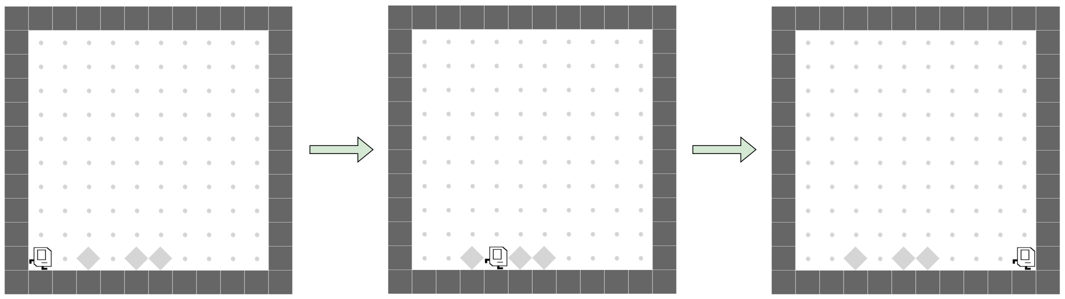
This task takes place in an 8×8 grid environment, where the grid is partitioned into a 6×3 left room and a 6×2 right room. Initially, these two rooms are not connected. The agent's objective is to collect a key (marker) within the left room to unlock a door (make the two rooms connected) and



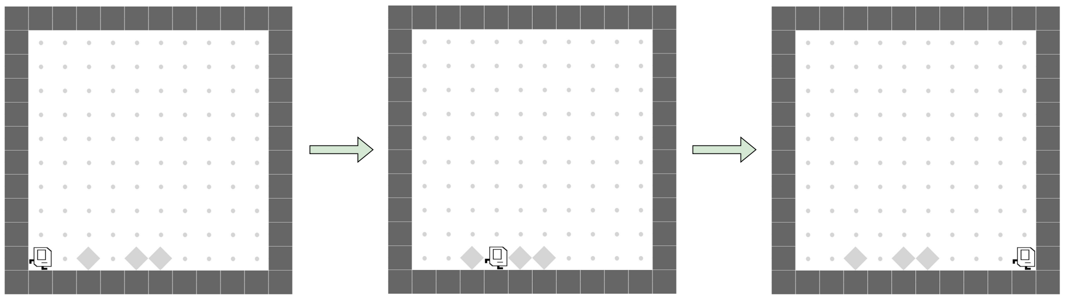
(a) STAIRCLIMBER



(b) FOURCORNER



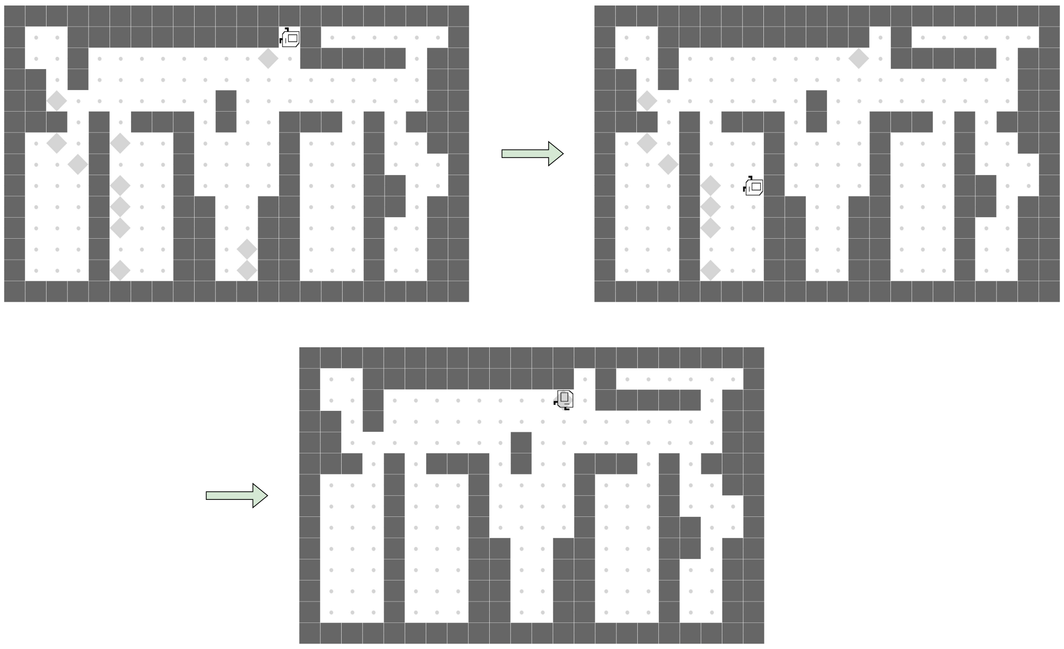
(c) TOPOFF



(d) MAZE

Figure 13: Visualization of STAIRCLIMBER, FOURCORNER, TOPOFF and MAZE in the KAREL problem set presented in Trivedi et al. (2021). For each task, a random initial state, a legitimate internal state, and the ideal end state are shown. In most tasks, the position of markers and the initial location of the Karel agent are randomized. More details of the KAREL problem set can be found in Section G.

precisely position the collected key atop a target (marker) situated in the right room subsequently. The agent's initial location, the key's location, and the target's location are all randomized. The agent receives a 0.5 reward for collecting the key and another 0.5 reward for putting the key on top of the target.


(a) CLEANHOUSE

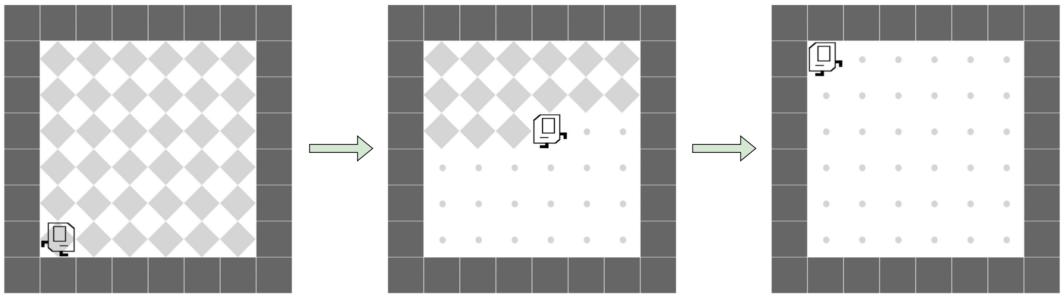

(b) HARVESTER

Figure 14: Visualization of CLEANHOUSE and HARVESTER in the KAREL problem set presented in Trivedi et al. (2021). For each task, a random initial state, a legitimate internal state, and the ideal end state are shown. More details of the KAREL problem set can be found in Section G.

H.2 ONESTROKE

This task takes place in an 8×8 grid environment, where the agent's objective is to navigate through all grid cells without revisiting any of them. Once a grid cell is visited, it transforms into a wall. If the agent ever collides with these walls, the episode ends. The reward is defined as the ratio of grids visited to the total number of empty grids in the initial environment.

H.3 SEEDER

This task takes place in an 8×8 grid environment, where the agent's objective is to place a marker on every single grid. If the agent repeatedly puts markers on the same grid, the episode will then terminate. The reward is defined as the ratio of the number of markers successfully placed to the total number of empty grids in the initial environment.

H.4 SNAKE

This task takes place in an 8×8 grid environment, where the agent plays the role of the snake's head and aims to consume (pass through) as much food (markers) as possible while avoiding colliding with its own body. Each time the agent consumes a marker, the snake's body length grows by 1, and

Table 3: **KAREL and KAREL-HARD Performance.** Mean return and standard deviation of all methods across the KAREL and KAREL-HARD problem set, evaluated over five random seeds. As the table shows, POMP outperforms LEAPS and HPRL on most KAREL and KAREL HARD tasks, except ONESTROKE and SNAKE, which need good long-term planning abilities. One thing to notice is that, unlike KAREL-LONG, KAREL and KAREL-HARD do not have per-action cost in design, so for POMP here, we did not let it learn to terminate but gave a maximum mode program execution number. Therefore, the whole policy will stop when this number is reached, or the task is solved.

Method	Four Corner	Top Off	Clean House	Stair Climber	Harvester	Maze	Door Key	One Stroke	Seeder	Snake
DRL	0.29 ± 0.05	0.32 ± 0.07	0.00 ± 0.00	$\mathbf{1.00} \pm 0.00$	0.90 ± 0.10	$\mathbf{1.00} \pm 0.00$	0.48 ± 0.03	$\mathbf{0.89} \pm 0.04$	0.96 ± 0.02	$\mathbf{0.67} \pm 0.17$
LEAPS	0.45 ± 0.40	0.81 ± 0.07	0.18 ± 0.14	$\mathbf{1.00} \pm 0.00$	0.45 ± 0.28	$\mathbf{1.00} \pm 0.00$	0.50 ± 0.00	0.65 ± 0.19	0.51 ± 0.21	0.21 ± 0.15
HPRL (5)	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	0.50 ± 0.00	0.80 ± 0.02	0.58 ± 0.07	0.28 ± 0.11
POMP	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	0.62 ± 0.01	$\mathbf{0.97} \pm 0.02$	0.36 ± 0.02

a new marker emerges at a different location. Before the agent successfully consumes 20 markers, there will consistently exist exactly one marker in the environment. The reward is defined as the ratio of the number of markers consumed by the agent to 20.

I DETAILS OF KAREL-LONG PROBLEM SET

Since none of the tasks in the KAREL and KAREL-HARD problem sets are truly long-horizon tasks, it is inadequate to use any of them as the environment when investigating the ability of our proposed framework. Hence, we introduce a newly designed KAREL-LONG problem set as a benchmark to evaluate the capability of POMP. Each task is designed to possess long-horizon properties based on the Karel states. Besides, we design the tasks in our KAREL-LONG benchmark to have a constant per-action cost (i.e., 0.0001). Figure 16, Figure 17, Figure 18, Figure 19 and Figure 20 provide visual depictions of all the tasks within the KAREL-LONG problem set. For each task, a randomly generated initial state and some internal states sampled from a legitimate trajectory are provided.

I.1 EXPERIMENT RESULT DISCUSSION

As shown in Table 2, all approaches except ours struggle at the tasks in the KAREL-LONG problem set that require long-horizon exploration with sparse rewards (i.e., SEESAW and UP-N-DOWN). On the other hand, most approaches can achieve satisfactory performance on the tasks that provide dense rewards (i.e., FARMER, INF-DOORKEY, and INF-HARVESTER), even when the episode horizon is long.

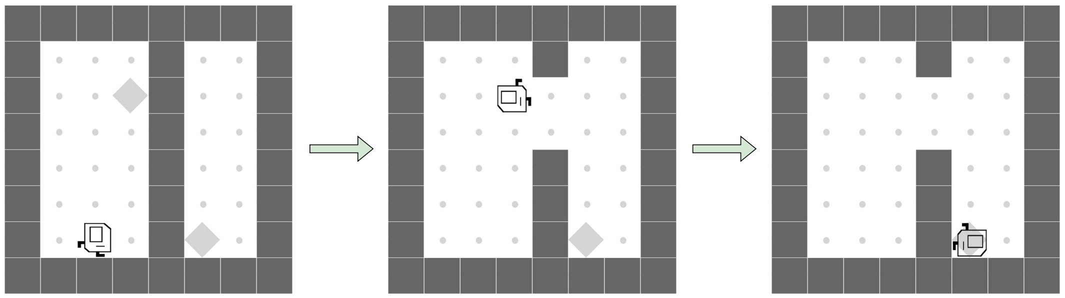
In INF-DOORKEY, the environment consists of four small chambers (i.e., one 3x3, two 2x3, and one 2x2) connected. Therefore, each stage of INF-DOORKEY requires only moderate exploration compared to FARMER and INF-HARVESTER, allowing DRL to perform best. Besides, in INF-HARVESTER, the rewards provided are very dense, which is suitable for HPRL for optimizing the meta-policy and allowing it to perform best.

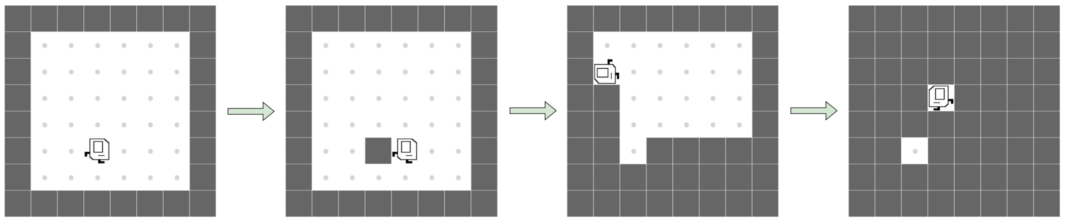
I.2 SEESAW

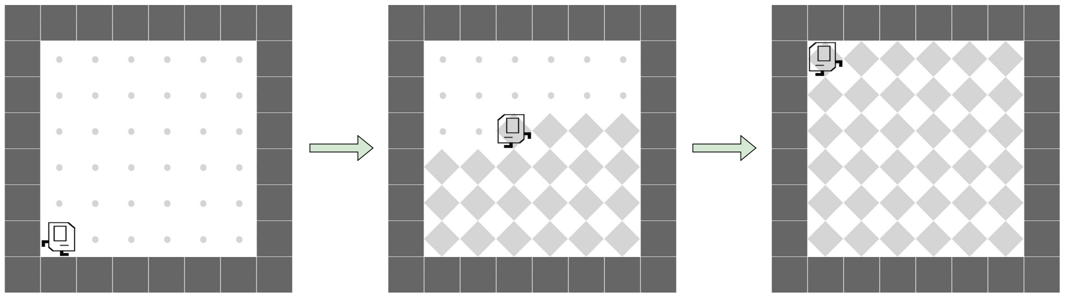
This task takes place in a 16×16 grid environment, where the agent's objective is to move back and forth between two 4×4 chambers, namely the left chamber and the right chamber, to continuously collect markers. To facilitate movement between the left and right chambers, the agent must traverse through a middle 2×6 corridor. Initially, exactly one marker is randomly located in the left chamber, awaiting the agent to collect. Once a marker is picked in a particular chamber, another marker is then randomly popped out in the other chamber, further waiting for the agent to collect. Hence, the agent must navigate between the two chambers to pick markers continuously. The reward is defined as the ratio of the number of markers picked by the agent to the total number of markers that the environment is able to generate (emerging markers).

I.3 UP-N-DOWN

This task takes place in an 8×8 grid environment, where the agent's objective is to ascend and descend the stairs repeatedly to collect markers (loads) appearing both above and below the stairs.


(a) DOORKEY


(b) ONESTROKE


(c) SEEDER

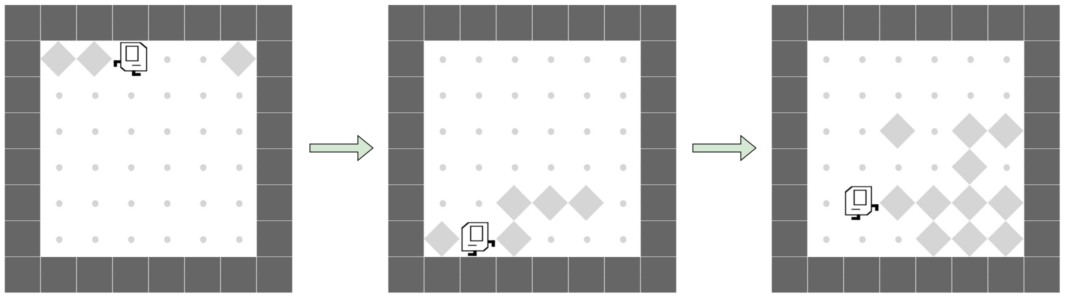

(d) SNAKE

Figure 15: Visualization of each task in the KAREL-HARD problem set proposed by Liu et al. (2023). For each task, a random initial state, some legitimate internal state(s), and the ideal end state are shown. More details of the KAREL-HARD problem set can be found in Section H.

Once a marker below (above) the stairs is picked up, another marker will appear above (below) the stairs, enabling the agent to continuously collect markers. If the agent moves to a grid other than those right near the stairs, the agent will receive a constant penalty (i.e., 0.005). The reward is defined as the ratio of the number of markers picked by the agent to the total number of markers that the environment is able to generate (emerging loads).

I.4 FARMER

This task takes place in an 8×8 grid environment, where the agent's objective is to repeatedly fill the entire environment layout with markers and subsequently collect all of these markers. Initially, all grids in the environment are empty except for the one in the upper-right corner. The marker in the upper-right corner is designed to be a signal that indicates the agent to start populating the environment layout with markers. After most of the grids are placed with markers, the agent is then asked to pick up markers as much as possible. Then, the agent is further asked to fill the environment again, and the whole process continues in this back-and-forth manner. We have set a maximum iteration number to represent the number of the filling-and-collecting rounds that we expect the agent to accomplish. The reward is defined as the ratio of the number of markers picked and placed by the agent to the total number of markers that the agent is theoretically able to pick and place (max markers).

I.5 INF-DOORKEY

This task takes place in an 8×8 grid environment, where the agent's objective is to pick up a marker in certain chambers, place a marker in others, and continuously traverse between chambers until a predetermined upper limit number of marker-picking and marker-placing that we have set is reached. The entire environment is divided into four chambers, and the agent can only pick up (place) markers in one of these chambers. Once the agent does so, the passage to the next chamber opens, allowing the agent to proceed to the next chamber to conduct another placement (pick-up) action. The reward is defined as the ratio of markers picked and placed by the agent to the total number of markers that the agent can theoretically pick and place (max keys).

I.6 INF-HARVESTER

This task takes place in a 16×16 grid environment, where the agent's objective is to continuously pick up markers until no markers are left and no more new markers are further popped out in the environment. Initially, the environment is entirely populated with markers. Whenever the agent picks up a marker from the environment, there is a certain probability (emerging probability) that a new marker will appear in a previously empty grid within the environment, allowing the agent to collect markers both continuously and indefinitely. The reward is defined as the ratio of the number of markers picked by the agent to the expected number of total markers that the environment can generate at a certain probability.

J DESIGNING DOMAIN-SPECIFIC LANGUAGES

Our program policies are designed to describe high-level task-solving procedures or decision-making logics of an agent. Therefore, our principle of designing domain-specific languages (DSLs) considers a general setting where an agent can perceive and interact with the environment to fulfill some tasks. DSLs consist of control flows, perceptions, and actions. While control flows are domain-independent, perceptions and actions can be designed based on the domain of interest, which would require specific expertise and domain knowledge.

Such DSLs are proposed and utilized in various domains, including ViZDoom (Kempka et al., 2016), 2D MineCraft (Sun et al., 2020), and gym-minigrid (Chevalier-Boisvert et al., 2023). Recent works (Liang et al., 2023; Wang et al., 2023) also explore describing agents' behaviors using programs with functions taking arguments.

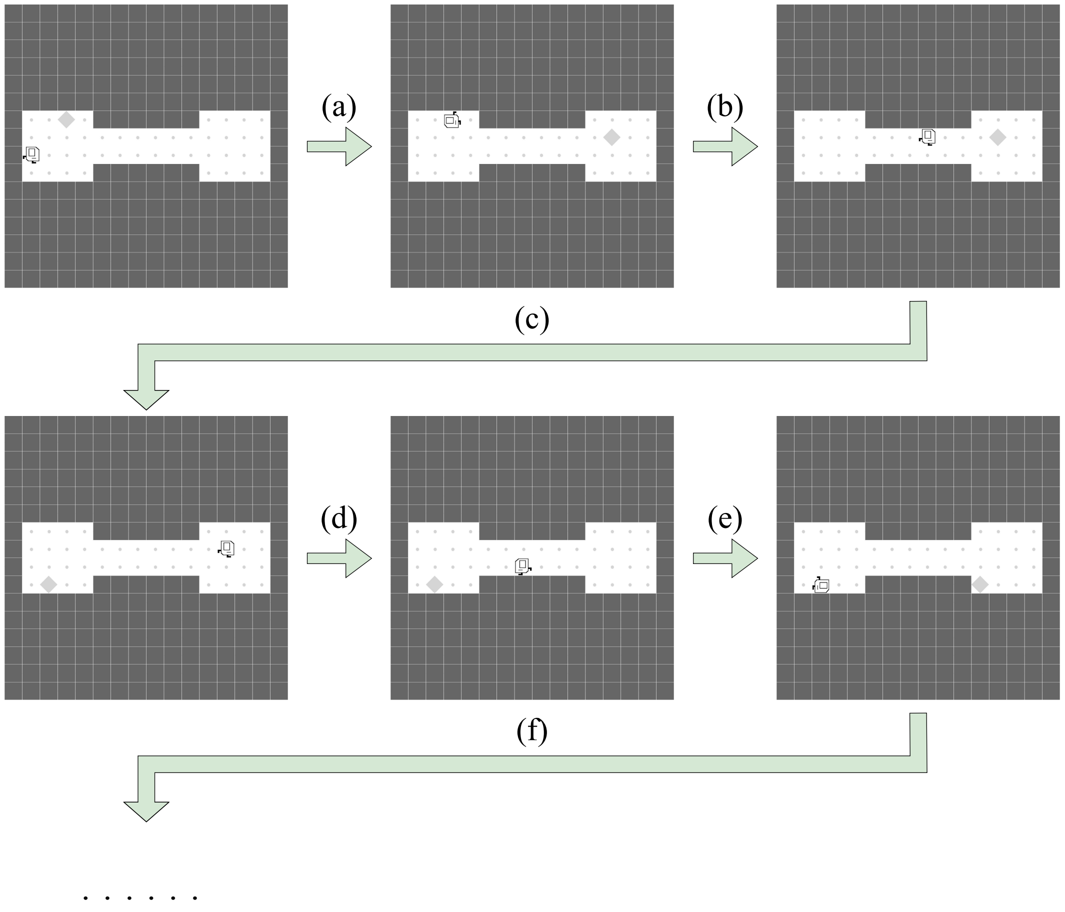


Figure 16: **Visualization of SEESAW in the KAREL-LONG problem set.** This partially shows a typical trajectory of the Karel agent during the task SEESAW. (a): Once the Karel agent collects a marker in the left chamber, a new marker will appear in the right chamber. (b): The agent must navigate through a middle corridor to collect the marker in the right chamber. (c): Upon the Karel agent collecting a marker in the right chamber, a new marker further appears in the left chamber. (d): Once again, the agent is traversing through the corridor to the left chamber. (e): A new marker appears in the right chamber again after the agent picks up the marker in the left chamber. (f): The agent will move back and forth between the two chambers to collect the emerging markers continuously. Note that the locations of all the emerging markers are randomized. Also, note that we have set the number of emerging markers to 10 during the training phase, meaning the agent has to pick up 10 markers to fully complete the task. More details of the task SEESAW can be found in Section I.

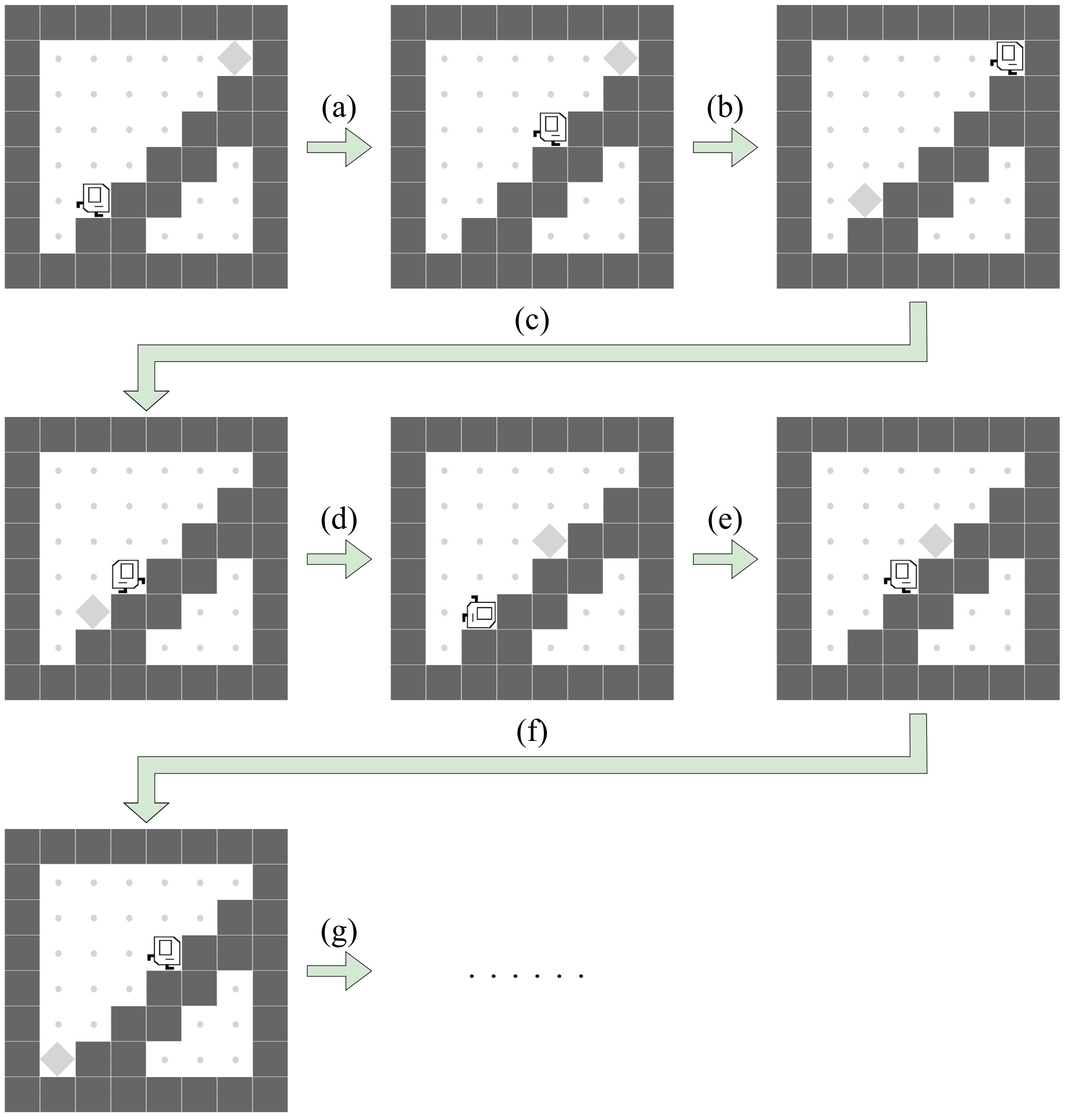


Figure 17: **Visualization of UP-N-DOWN in the KAREL-LONG problem set.** This partially shows a typical trajectory of the Karel agent during the task UP-N-DOWN. (a): The Karel agent is ascending the stairs to collect a load located above the stairs. Note that the agent can theoretically collect the load without directly climbing up the stairs, but it will receive some penalties if it does so. (b): Once the agent collects the load, a new load appears below the stairs. (c): The agent is descending the stairs to collect a load located below the stairs. Still, note that the agent can theoretically collect the load without directly climbing down the stairs, but it will receive some penalties if it does so. (d): Upon the agent collecting the load, a new load appears above the stairs. (e): The agent is again ascending the stairs to collect a load. (f): A new load appears below the stairs again after the agent collects the load located above the stairs. (g): The agent will then descend and ascend the stairs repeatedly to collect the emerging loads. Note that the locations of all the emerging loads are randomized right near the stairs, and they will always appear above or below the stairs depending on the position of the agent. Also, note that we have set the number of emerging loads to 10 during the training phase, meaning the agent has to collect 10 loads to fully complete the task. More details of the task UP-N-DOWN can be found in Section I.

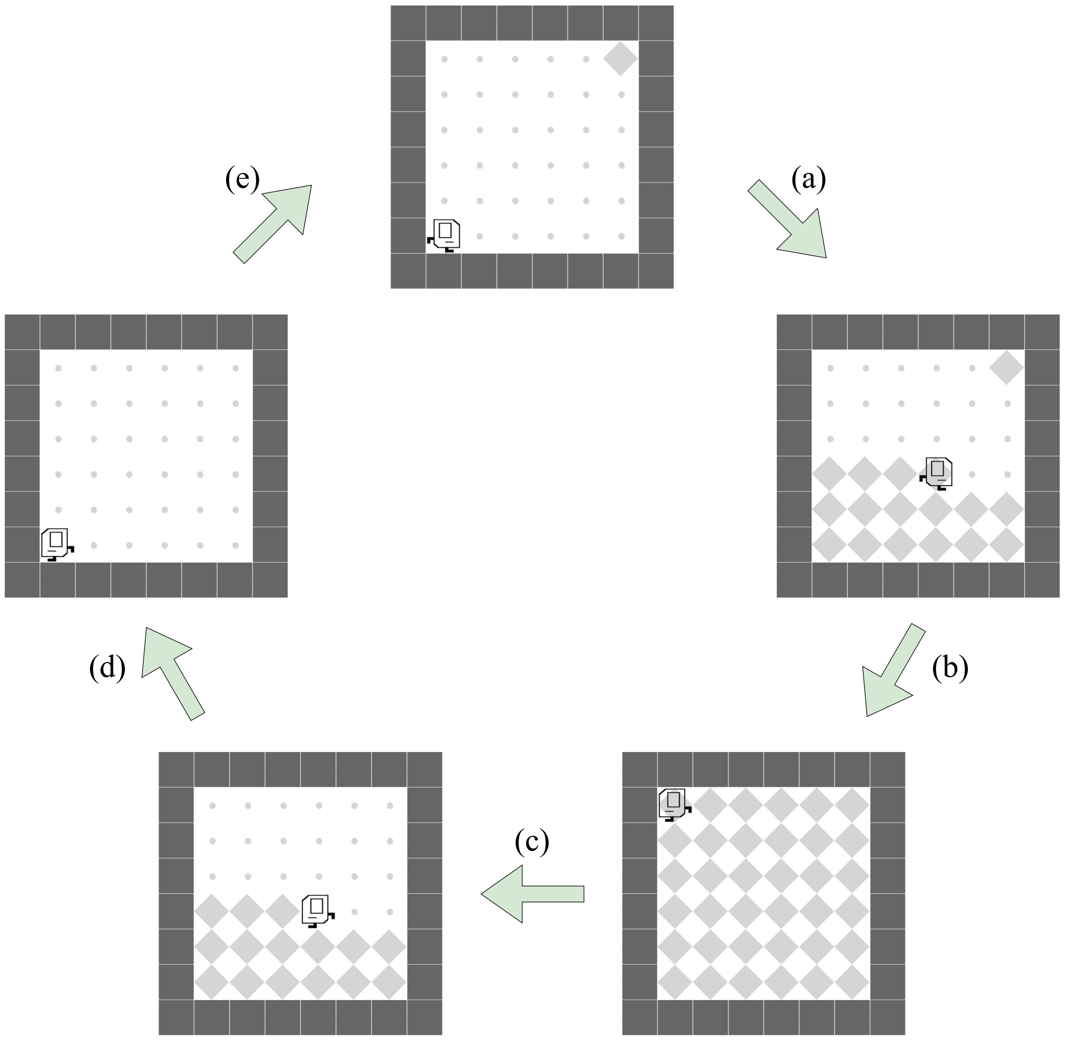


Figure 18: **Visualization of FARMER in the KAREL-LONG problem set.** This partially shows a typical trajectory of the Karel agent during the task FARMER. (a): The Karel agent is filling (placing) the entire environment layout with markers. Note that, in the initial state, there is a single marker located in the upper-right corner. The marker is designed to be a signal indicating the agent to start filling the environment layout. (b): The agent successfully populates the entire environment. (c): The agent is then asked to pick up markers as much as possible. (d): The agent successfully picks all markers up, leaving the environment empty. (e): If there is another filling-and-collecting round, a marker will appear in the upper-right corner to indicate that the agent should start the filling process again. Otherwise, the agent completes the entire task, and no further marker will appear. For simplicity, here, we only show the former case. Note that we have set the number of max markers to 144 during the training phase, meaning the agent has to both fill the entire environment layout with markers and pick up all markers twice to fully complete the task. More details of the task FARMER can be found in Section I.

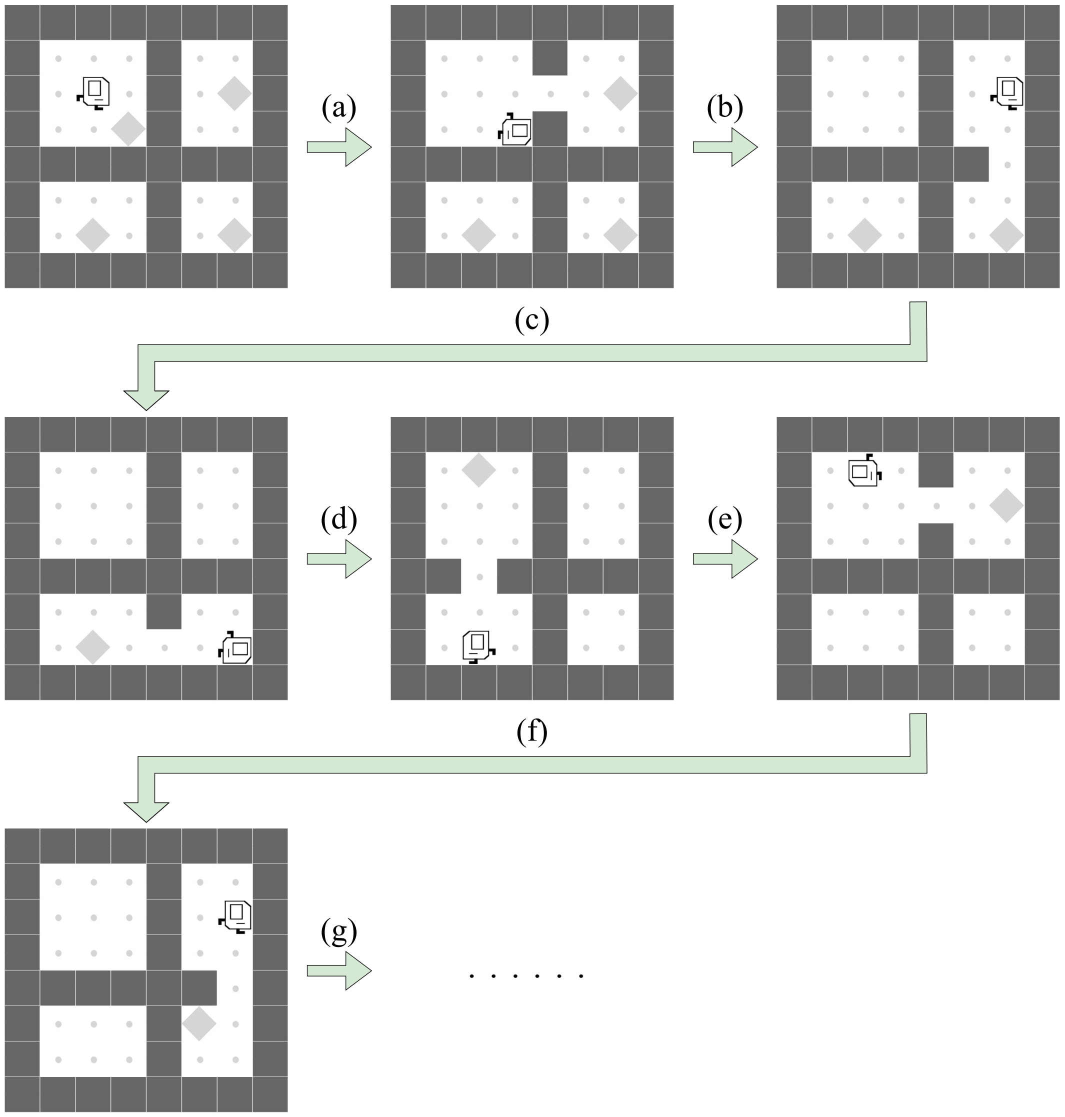


Figure 19: **Visualization of INF-DOORKEY in the KAREL-LONG problem set.** This partially shows a typical trajectory of the Karel agent during the task INF-DOORKEY. (a): The Karel agent picks up a marker in the upper-left chamber. Then, a passage to the upper-right chamber opens, allowing the agent to traverse through. (b): The agent successfully places a marker at a marked grid located in the upper-right chamber. Subsequently, a passage to the lower-right chamber opens, allowing the agent to traverse through. (c): After the agent collects a marker in the lower-right chamber, a passage to the lower-left chamber opens, allowing the agent to traverse through. (d): The agent properly places a marker at a marked grid located in the lower-left chamber. After that, a passage to the upper-left chamber opens and a new marker appears in the upper-left chamber. (e): Upon the agent picking up a marker in the upper-left chamber, the passage to the upper-right chamber opens again and a grid is marked randomly in the upper-right chamber. (f): The agent accurately places a marker at a marked grid located in the upper-right chamber. Afterward, the passage to the lower-right chamber opens again, and a new marker emerges in the lower-right chamber. (g): The agent will repeatedly pick up and place markers in this fashion until the number of max keys is reached. We have set the number of max keys to 16 during the training phase, meaning the agent has to pick up and place 16 markers in total to fully complete the task. More details of the task INF-DOORKEY can be found in Section I.

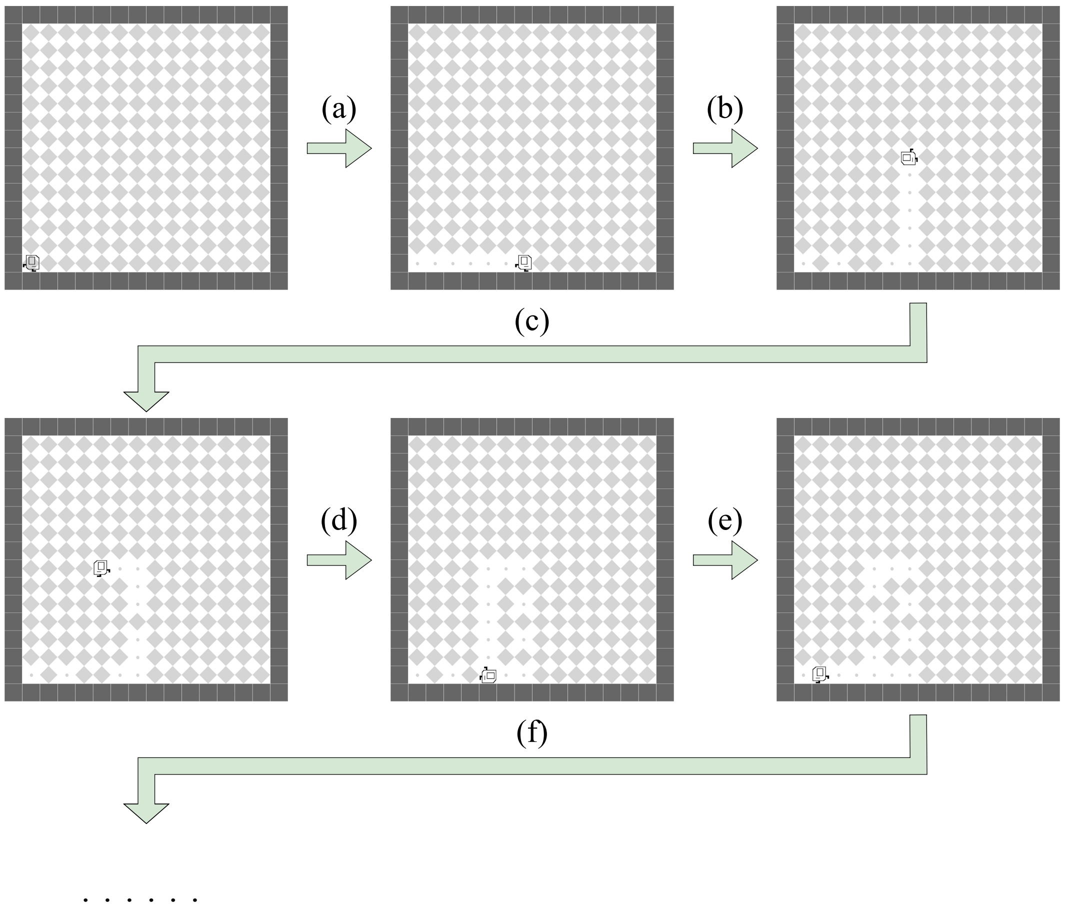


Figure 20: **Visualization of INF-HARVESTER in the KAREL-LONG problem set.** This partially shows a legitimate trajectory of the Karel agent during the task INF-HARVESTER. (a): The Karel agent is picking up markers in the last row. Meanwhile, no new markers are popped out in the last row. (b): The agent turns left and picks up 6 markers in the 7^{th} column. During this picking-up process, 3 markers appeared in 3 previously empty grids in the last row. (c): The agent is collecting markers in the 8^{th} row. During this picking-up process, 1 marker appeared in a previously empty grid in the 7^{th} column. (d): The agent picks up 6 markers in the 5^{th} column. During this picking-up process, 2 markers appeared in 2 previously empty grids in the 7^{th} column. (e): The agent picks up 2 more markers in the last row. During this picking-up process, 2 markers appeared in 2 previously empty grids in the 5^{th} column. (f): Since markers will appear in previously empty grids based on the emerging probability, the agent will continuously and indefinitely collect markers until no markers are left and no more new markers are further popped out in the environment. Note that we have set the emerging probability to $\frac{1}{2}$ during the training phase. More details of the task INF-HARVESTER can be found in Section I.

Karel Programs

SEESAW

Mode 1

```
DEF run m(
    IFELSE c( frontIsClear c)
         i(
        turnLeft
        i)
    ELSE e(
        turnLeft
        e)
    m)
```

Mode 2

```
DEF run m(
    IF c( leftIsClear c) i(
        move
        turnLeft
        WHILE c( frontIsClear
             c) w(
            move
            pickMarker
            w)
        i)
    m)
```

Mode 3

```
DEF run m(
    IF c( markersPresent c) i(
        REPEAT R=2 r(
            turnLeft
            r)
        putMarker
        i)
    m)
```

Mode 4

```
DEF run m(
    WHILE c( noMarkersPresent
         c) w(
        turnLeft
        move
        w)
    IFELSE c(frontIsClear c) i
         (
        pickMarker
        i)
    ELSE e(
        move
        e)
    turnRight
    m)
```

Mode 5

```
DEF run m(
    WHILE c( frontIsClear c) w
         (
        IF c( markersPresent c
             ) i(
            move
            i)
        w)
    IFELSE c( frontIsClear c)
         i(
        pickMarker
        move
        i)
    ELSE e(
        move
        turnRight
        e)
    m)
```

UP-N-DOWN

Mode 1

```
DEF run m(
    WHILE c( markersPresent c)
         w(
        turnRight
        turnLeft
        w)
    IF c( frontIsClear c) i(
        turnRight
        move
        i)
    m)
```

Mode 2

```
DEF run m(
    WHILE c( rightIsClear c) w
         (
        turnRight
        move
        w)
    IF c( frontIsClear c) i(
        WHILE c(
             markersPresent c
             ) w(
            IF c( leftIsClear
                 c) i(
                pickMarker
                i)
            w)
        i)
    m)
```

Mode 3

```
DEF run m(
    WHILE c( noMarkersPresent
         c) w(
        turnRight
        move
        IFELSE c( leftIsClear
             c) i(
            move
            i)
        ELSE e(
            move
            e)
        w)
    m)
```

Mode 4

```
DEF run m(
    REPEAT R=16 r(
        move
        turnRight
        r)
    REPEAT R=0 r(
        move
        move
        r)
    m)
```

Mode 5

```
DEF run m(
    WHILE c( frontIsClear c) w
         (
        turnRight
        w)
    WHILE c( leftIsClear c) w(
        turnLeft
        move
        turnRight
        move
        w)
    REPEAT R=17 r(
        turnLeft
        move
        turnRight
        move
        r)
    m)
```

Figure 21: **Example programs on Karel-Long tasks: SEESAW and UP-N-DOWN.** The programs with best rewards out of all random seeds are shown.

Karel Programs

FARMER

Mode 1

```
DEF run m(
    IFELSE c( markersPresent c
         ) i(
        move
        i)
    ELSE e(
        move
        move
        move
        putMarker
        e)
    m)
```

Mode 2

```
DEF run m(
    IF c( frontIsClear c) i(
        move
        turnRight
        i)
    IF c( noMarkersPresent c)
         i(
        turnLeft
        move
        i)
    m)
```

Mode 3

```
DEF run m(
    REPEAT R=0 r(
        turnRight
        move
        turnLeft
        turnRight
        r)
    m)
```

Mode 4

```
DEF run m(
    REPEAT R=9 r(
        REPEAT R=9 r(
            move
            pickMarker
            r)
        turnRight
        r)
    REPEAT R=0 r(
        pickMarker
        r)
    turnRight
    move
    turnRight
    pickMarker
    turnRight
    move
    m)
```

Mode 5

```
DEF run m(
    REPEAT R=16 r(
        turnRight
        move
        putMarker
        move
        putMarker
        move
        putMarker
        move
        putMarker
        move
        putMarker
        move
        turnRight
        putMarker
        move
        r)
    m)
```

INF-DOORKEY

Mode 1

```
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)
```

Mode 2

```
DEF run m(
    WHILE c( noMarkersPresent
         c) w(
        move
        turnLeft
        w)
    WHILE c( noMarkersPresent
         c) w(
        move
        w)
    m)
```

Mode 3

```
DEF run m(
    move
    turnLeft
    move
    turnLeft
    move
    turnLeft
    m)
```

Mode 4

```
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)
```

Mode 5

```
DEF run m(
    putMarker
    REPEAT R=13 r(
        pickMarker
        turnRight
        move
        putMarker
        move
        pickMarker
        move
        turnRight
        move
        r)
    move
    pickMarker
    move
    move
    m)
```

Figure 22: **Example programs on Karel-Long tasks: FARMER and INF-DOORKEY.** The programs with best rewards out of all random seeds are shown.

Karel Programs

INF-HARVESTER

Mode 1

```
DEF run m(
    WHILE c( markersPresent c)
          w(
        WHILE c( frontIsClear
             c) w(
            pickMarker
            move
            w)
        turnLeft
        w)
    turnLeft
    m)
```

Mode 2

```
DEF run m(
    REPEAT R=9 r(
        turnRight
        REPEAT R=9 r(
            move
            pickMarker
            r)
        r)
    m)
```

Mode 3

```
DEF run m(
    turnRight
    turnRight
    move
    move
    move
    move
    move
    move
    m)
```

Mode 4

```
DEF run m(
    move
    REPEAT R=13 r(
        turnRight
        REPEAT R=13 r(
            move
            pickMarker
            r)
        r)
    m)
```

Mode 5

```
DEF run m(
    REPEAT R=8 r(
        turnRight
        move
        pickMarker
        WHILE c( frontIsClear
             c) w(
            move
            pickMarker
            w)
        r)
    m)
```

Figure 23: **Example programs on Karel-Long tasks: INF-HARVESTER.** The programs with best rewards out of all random seeds are shown.