Formal Languages and Automata for Reward Function Specification and Efficient Reinforcement Learning

Sheila A. McIlraith
Department of Computer Science, University of Toronto
Vector Institute for Artificial Intelligence
Schwartz Reisman Institute for Technology and Society

July 29, 2021

Computer Science UNIVERSITY OF TORONTO   VECTOR INSTITUTE | INSTITUT VECTEUR   CIFAR   SCHWARTZ REISMAN INSTITUTE FOR TECHNOLOGY AND SOCIETY

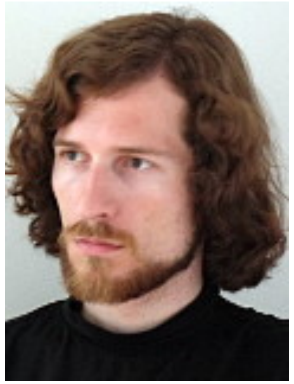
Photo credit dreamstime.com

Acknowledgements

Rodrigo Toro Icarte

Acknowledgements

Rodrigo Toro Icarte



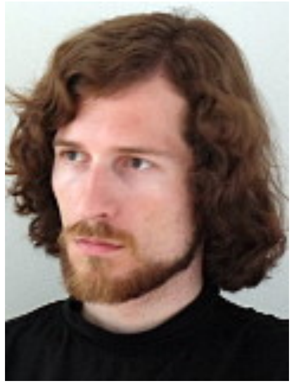
Toryn Klassen

Richard Valenzano

Alberto Camacho

Acknowledgements

Rodrigo Toro Icarte



Toryn Klassen

Richard Valenzano

Alberto Camacho

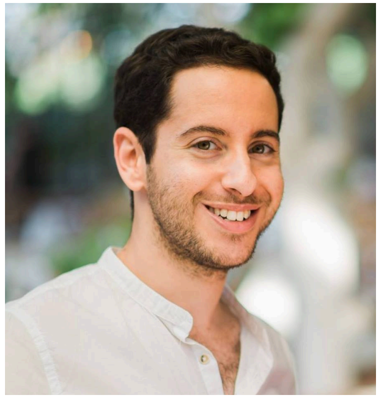
Léon Illanes

Ethan Waldie

Margarita Castro

Andrew Li

Pashootan Vaezipoor



Maayan Shvo

Phillip Christoffersen

Xi Yan

Sequential Decision Making

How do we decide how to act?

Sequential Decision Making

How do we decide how to act?

… and what informs this decision making?

Sequential Decision Making

How do we decide how to act?

Environment Behaviour

State

Agent Actions

Effects of Actions

Objectives of the Agent

Partially/Fully observable

Sequential Decision Making

How do we decide how to act?

Environment Behaviour

State

Agent Actions

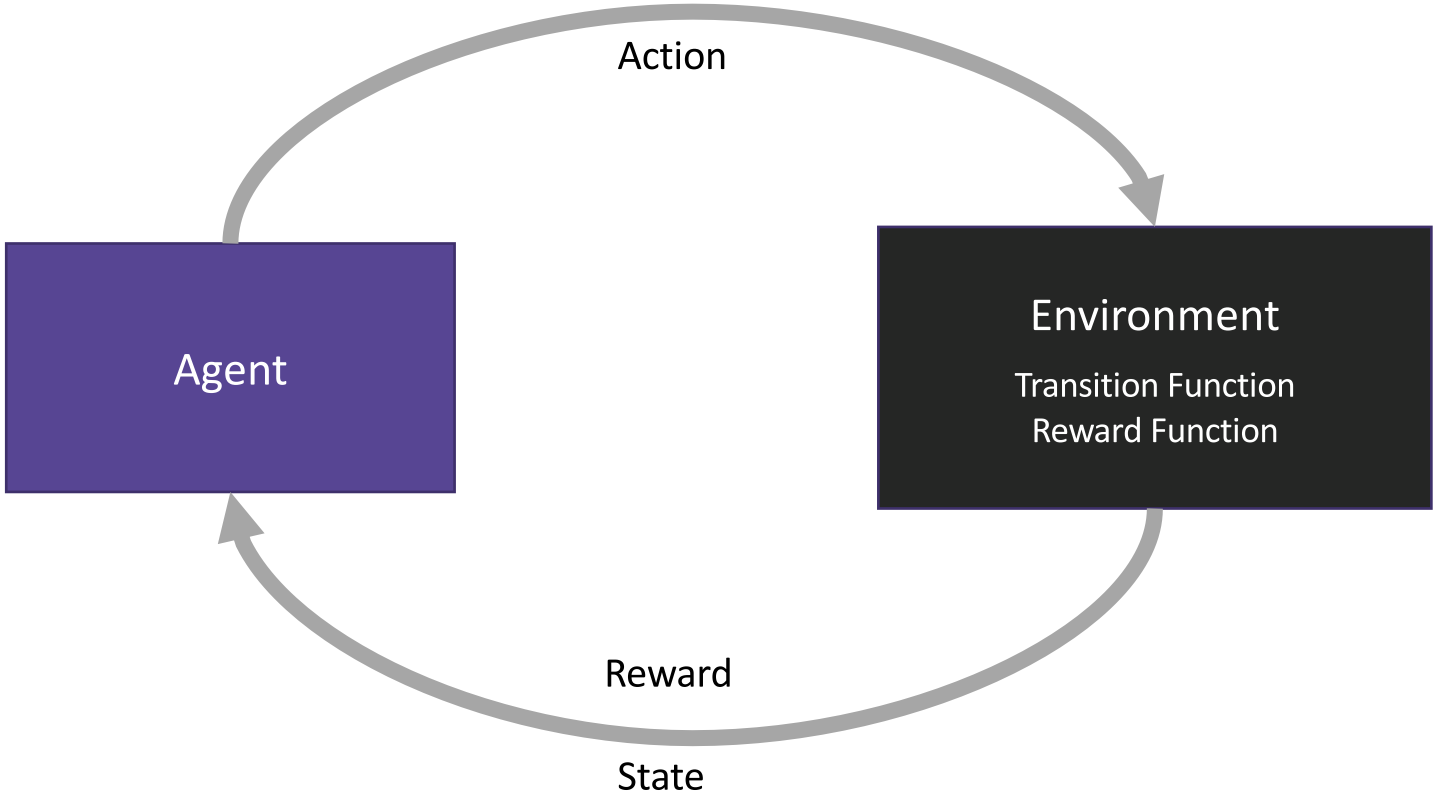
Effects of Actions

Objectives of the Agent

Partially/Fully observable

Perception

Reinforcement Learning (RL)



Following Sutton and Barto, 2018

Sequential Decision Making

How do we decide how to act?

$$\pi : S \to A$$

Environment Behaviour

State

Agent Actions

Effects of Actions

Objectives of the Agent

Partially/Fully observable

Perception

Sequential Decision Making

How do we decide how to act?

$$\pi : S \rightarrow A$$

Environment Behaviour

State

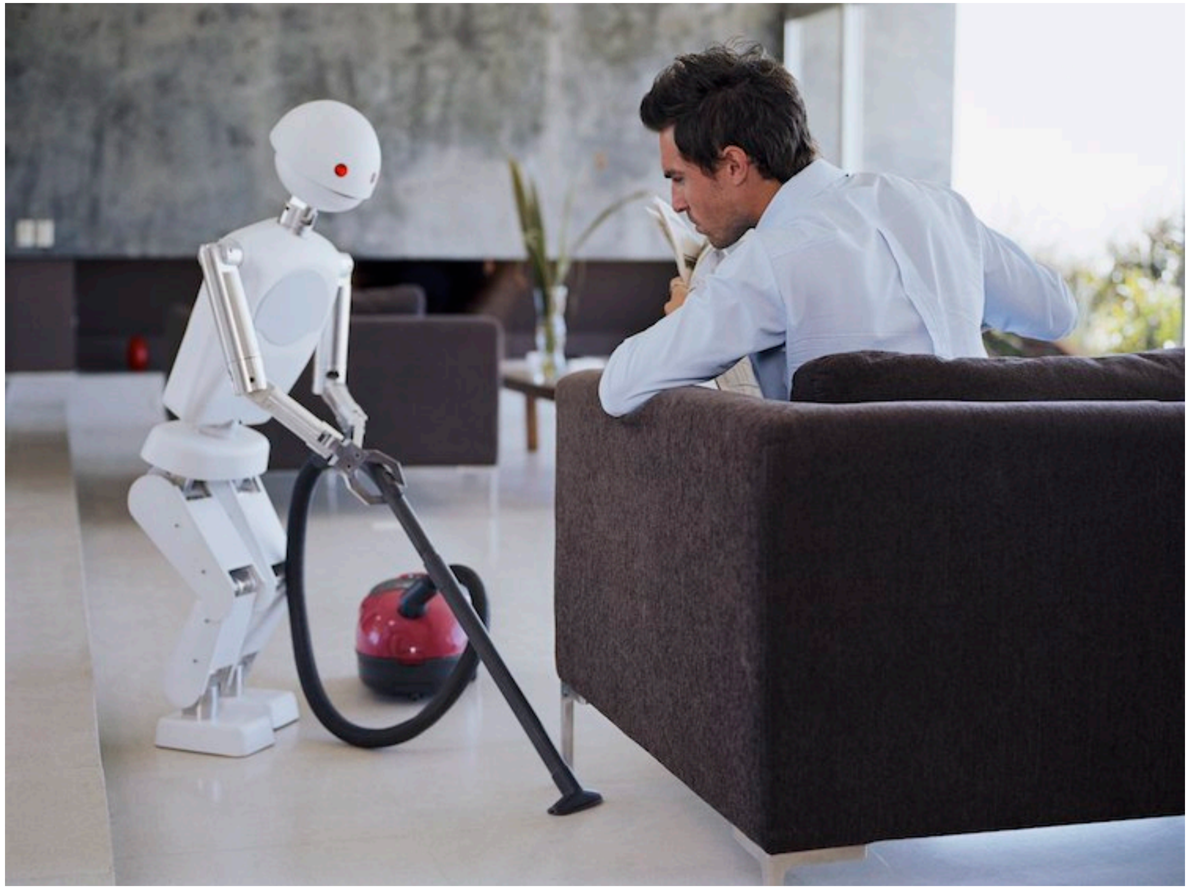
Agent Actions

Effects of Actions

Objectives of the Agent

Partially/Fully observable

Perception

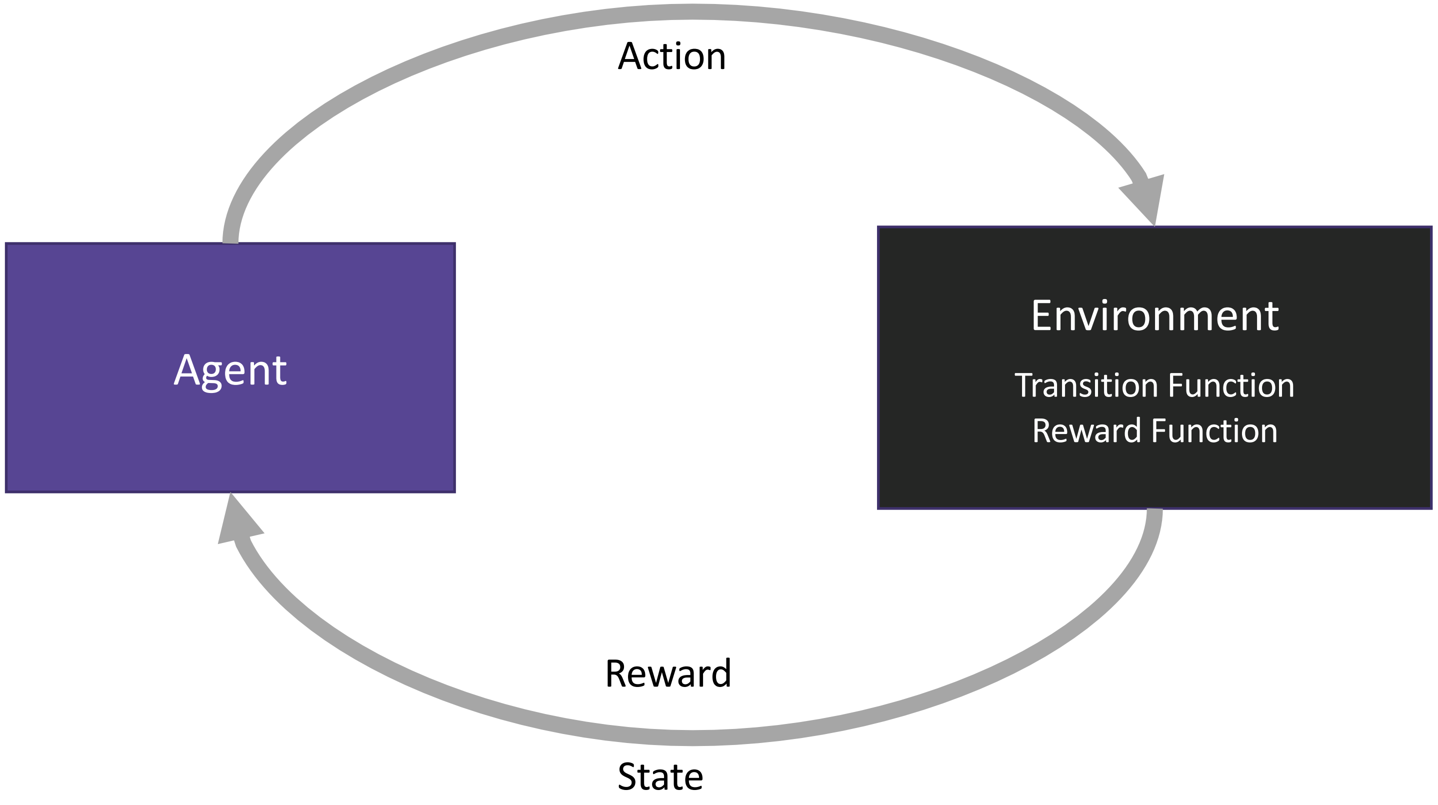


How do we advise, instruct, task, … and impart knowledge to our AI that learns?

Photo: Javier Pierin (Getty Images)

… and how do they use that knowledge to learn?

Reinforcement Learning (RL)



Following Sutton and Barto, 2018

Q-Learning

$$Q^{new}(s_t, a_t) \leftarrow \boxed{Q(s_t, a_t)} + \alpha * (r_t + \gamma * max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Image credit

Q-Learning

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \boxed{\alpha} * \boxed{(r_t + \gamma * max_a Q(s_{t+1}, a) - Q(s_t, a_t))}$$

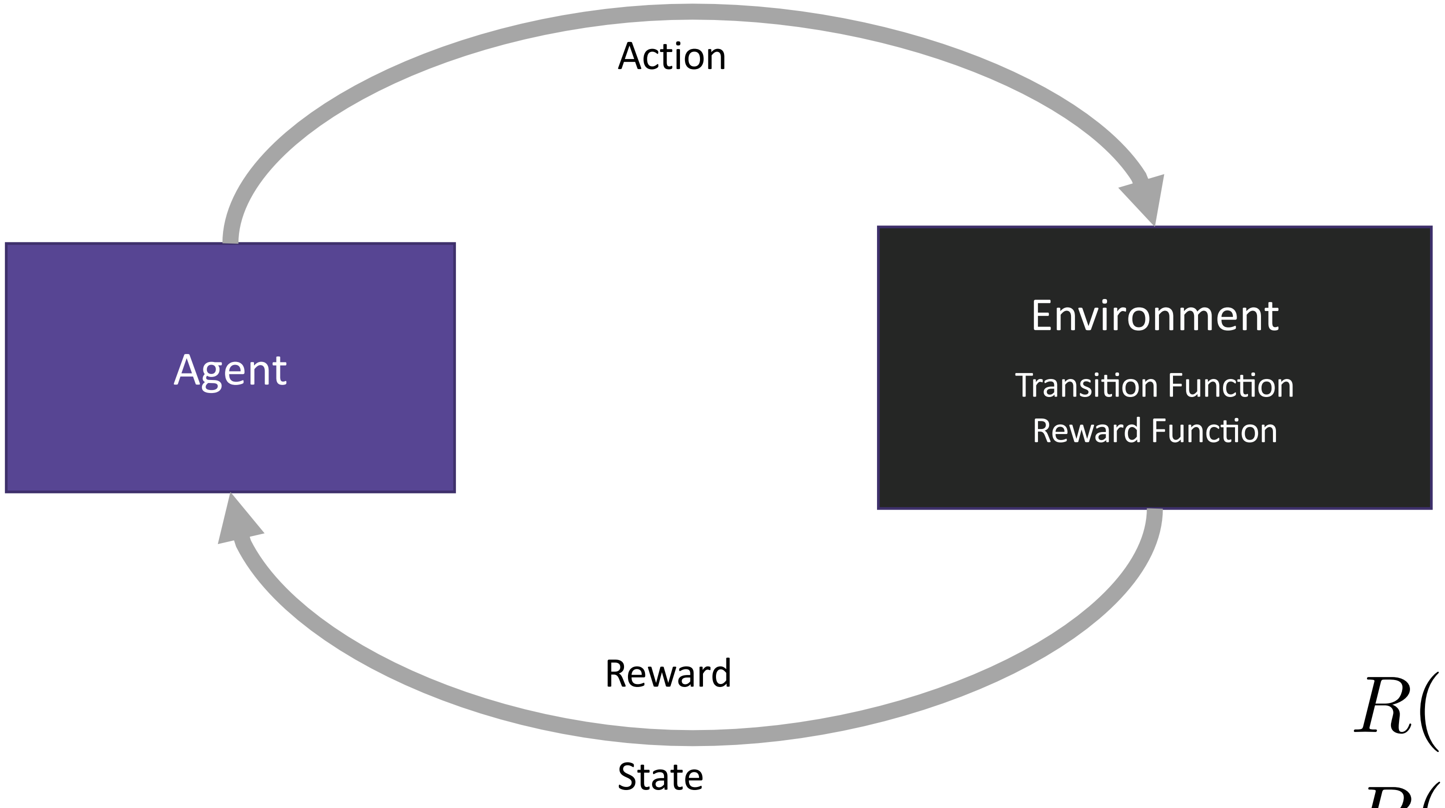
Image credit

Q-Learning

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * \boxed{(r_t} + \boxed{\gamma * max_a Q(s_{t+1}, a) - Q(s_t, a_t))}$$

Image credit

Reinforcement Learning (RL)



$$R(s) \to \mathbb{R}$$
$$R(s, a, s') \to \mathbb{R}$$

Following Sutton and Barto, 2018

Challenges to RL

- **Reward Specification:** It’s hard to define reward functions for complex tasks.

- **Sample Efficiency:** RL agents might require billions of interactions with the environment to learn good policies.

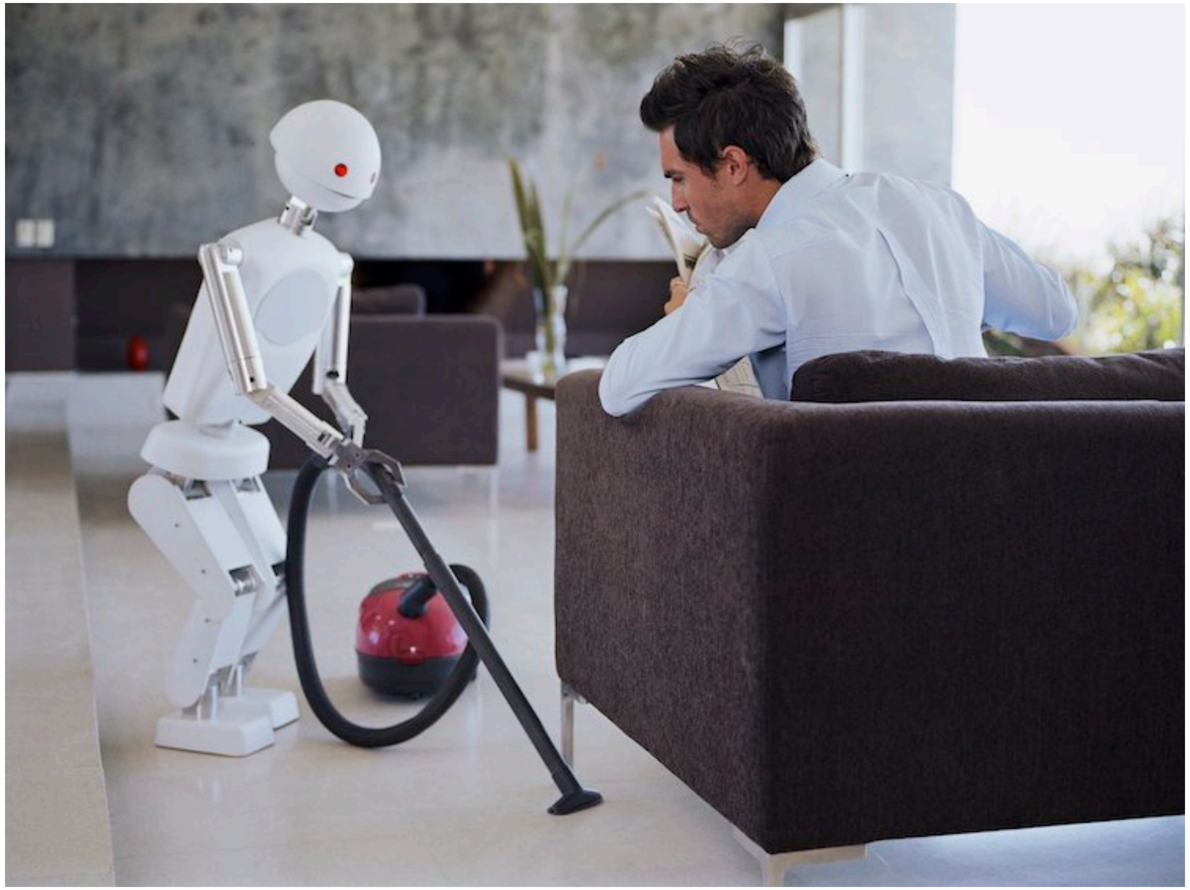


Photo: Javier Pierin (Getty Images)

Goals and Preferences

- Run the dishwasher when it’s full or when dishes are needed for the next meal.
- Make sure the bath temperature is between 38 – 43 celcius immediately before letting someone enter the bathtub.
- Do not vacuum while someone in the house is sleeping.

How do we communicate this to our RL agent?

Linear Temporal Logic (LTL)

A compelling logic to express temporal properties of traces.

Syntax

Logic connectives: $\wedge, \vee, \neg$

LTL **basic operators:**

- next: $\bigcirc\varphi$
- weak next: $\CIRCLE\varphi$
- until: $\psi \,\mathsf{U}\, \chi$

Other LTL **operators:**

- eventually: $\Diamond\varphi \stackrel{\text{def}}{=} \text{true}\,\mathsf{U}\,\varphi$
- always: $\Box\varphi \stackrel{\text{def}}{=} \neg\Diamond\neg\varphi$
- release: $\psi \,\mathsf{R}\, \chi \stackrel{\text{def}}{=} \neg(\neg\psi \,\mathsf{U}\, \neg\chi)$

Properties

- Interpreted over **finite** or **infintite** traces.
- Can be transformed into **automata**.

Linear Temporal Logic (LTL)

A compelling logic to express temporal properties of traces.

Syntax

Logic connectives: $\wedge, \vee, \neg$

LTL **basic operators:**

- next: $\bigcirc \varphi$
- weak next: $\bullet \varphi$
- until: $\psi \, \mathsf{U} \, \chi$

Other LTL **operators:**

- eventually: $\Diamond \varphi \stackrel{\text{def}}{=} \text{true} \, \mathsf{U} \, \varphi$
- always: $\Box \varphi \stackrel{\text{def}}{=} \neg \Diamond \neg \varphi$
- release: $\psi \, \mathsf{R} \, \chi \stackrel{\text{def}}{=} \neg(\neg \psi \, \mathsf{U} \, \neg \chi)$

Properties

- Interpreted over **finite** or **infintite** traces.
- Can be transformed into **automata**.

Remember this!

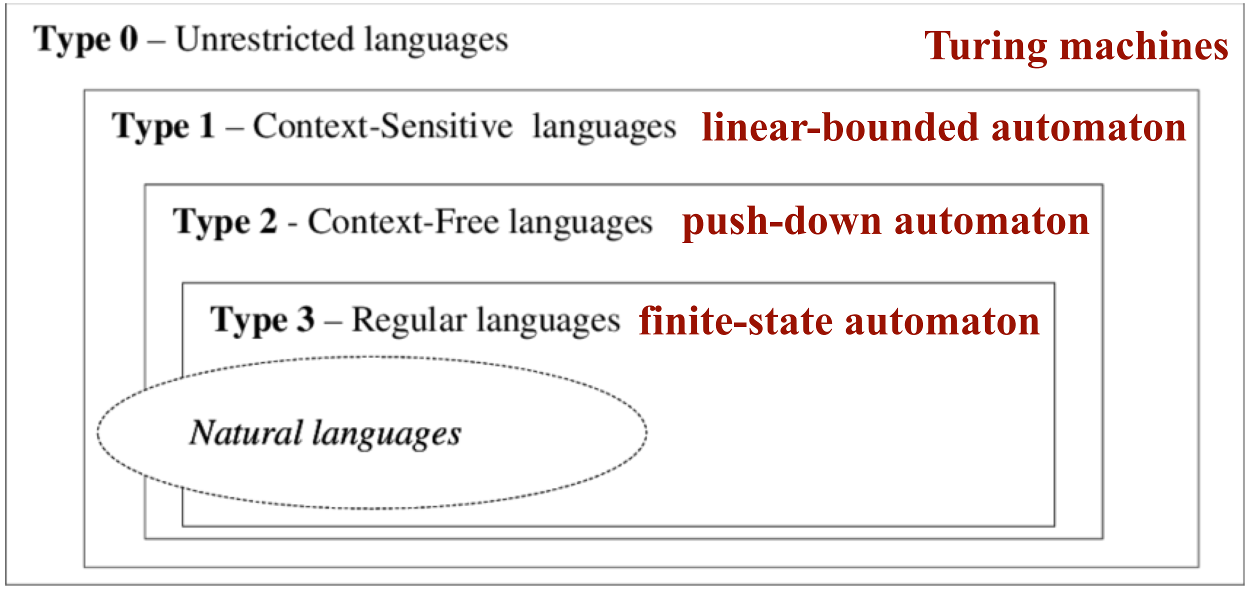
Goals and Preferences

- Do not vacuum while someone is sleeping

always[¬ (vacuum ∧ sleeping)]

How do we communicate this to our RL agent?

Remember Chomsky Hierarchy?



Noam Chomsky

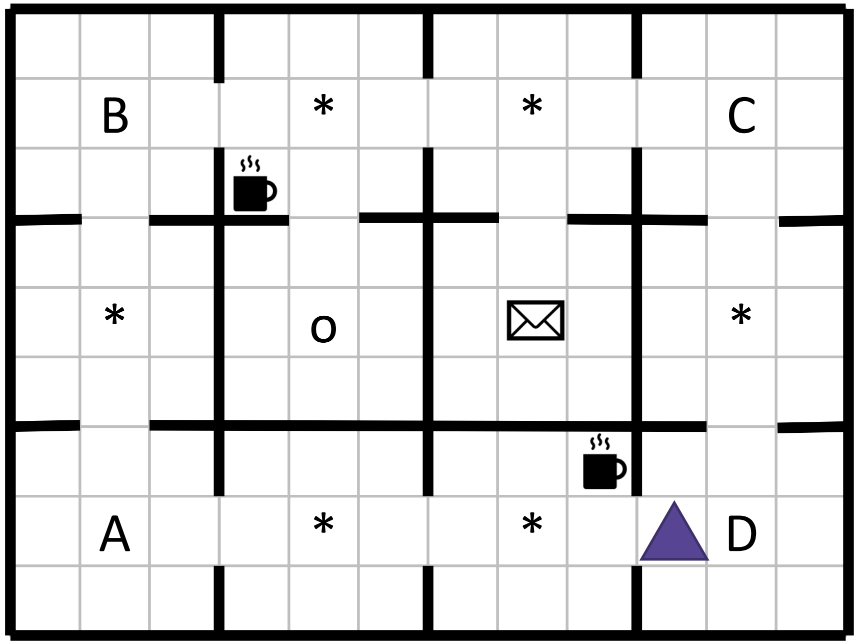
Automata

REWARD MACHINES

The Rest of the Talk

- ▶ Reward Machines (RM)
- Exploiting RM Structure in Learning
- Experiments
- Creating Reward Machines
- Recap

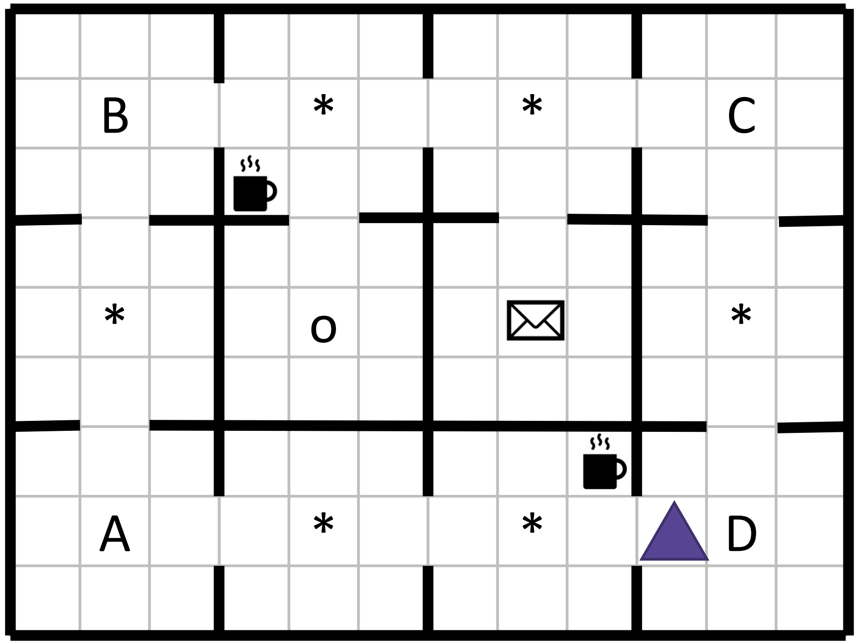
Running Example



Symbol	Meaning
▲	Agent
*	Furniture
☕	Coffee Machine
✉	Mail Room
o	Office
A, B, C, D	Marked Locations

Task: Visit A, B, C, and D, in order.

Reward Function

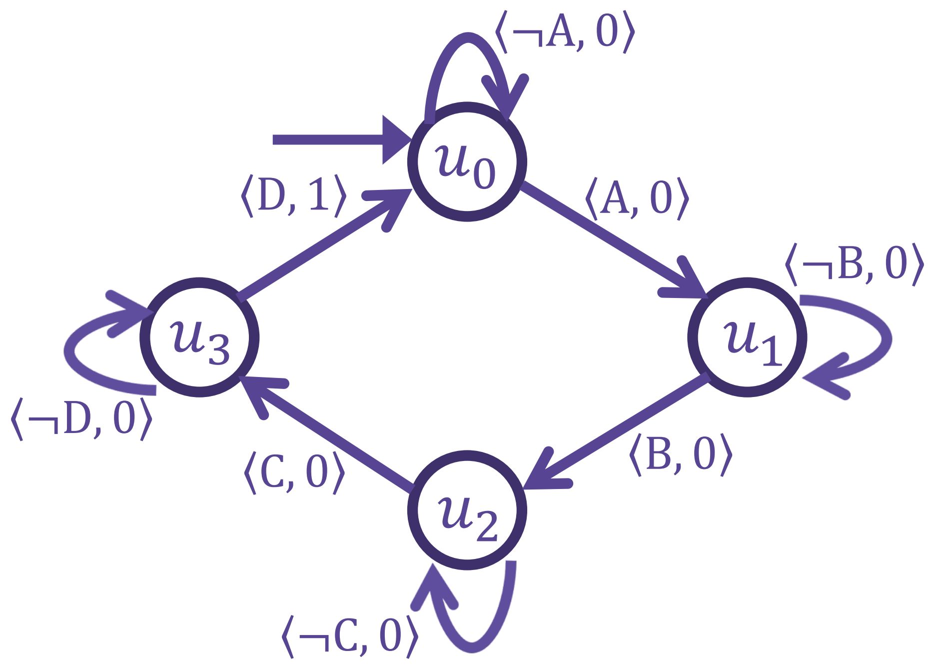



```
count = 0  # global variable

def get_reward(s):
    if count == 0 and state.at("A"):
        count = 1
    if count == 1 and state.at("B"):
        count = 2
    if count == 2 and state.at("C"):
        count = 3
    if count == 3 and state.at("D"):
        count = 0
        return 1
    return 0
```

Task: Visit A, B, C, and D, in order.

Define a Reward Function using a Reward Machine

```
count = 0  # global variable

def get_reward(s):
    if count == 0 and state.at("A"):
        count = 1
    if count == 1 and state.at("B"):
        count = 2
    if count == 2 and state.at("C"):
        count = 3
    if count == 3 and state.at("D"):
        count = 0
        return 1
    return 0
```



Encode reward function in an automata-like structure

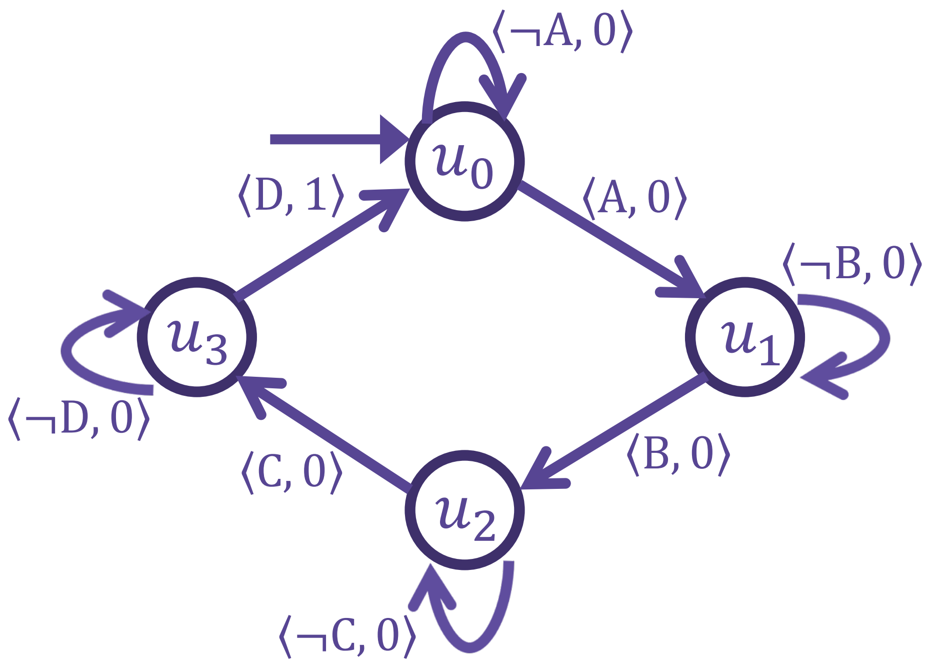
using a vocabulary $P = \{$☕, ✉, $o, *, A, B, C, D\}$

Reward Function Vocabulary

Vocabulary can comprise **human-interpretable events/properties** realized via detectors over the environment state, or it can (conceivably) be **learned**.

Reward Machine

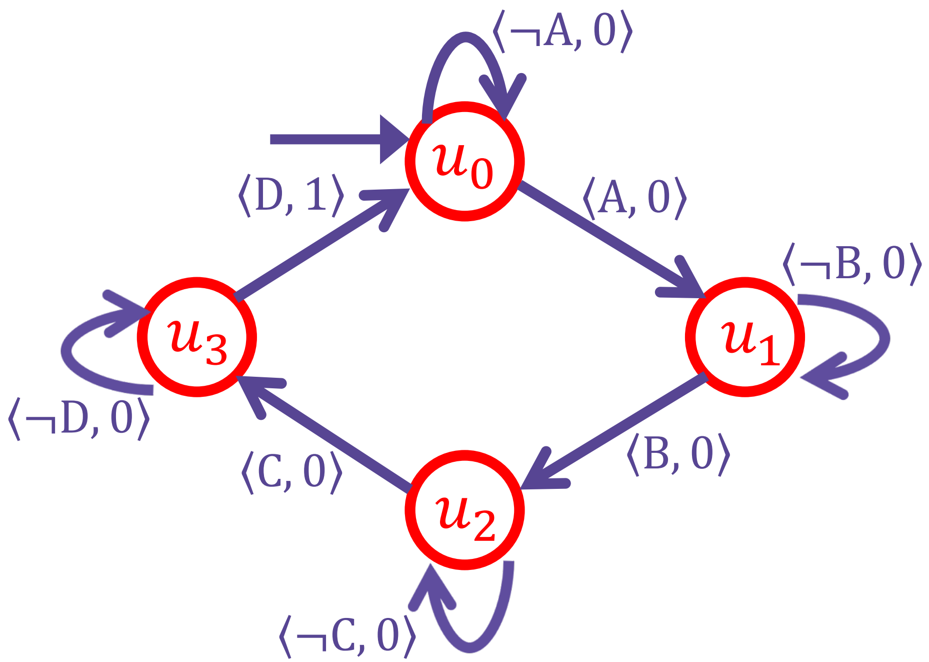
Reward Machine



Reward Machine

Reward Machine

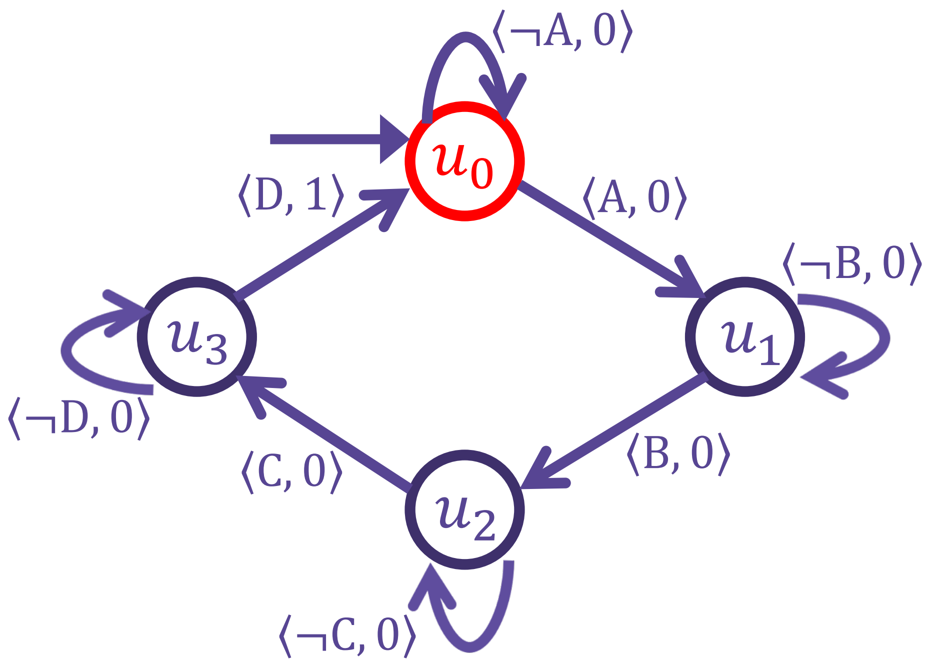
- finite set of states U



Reward Machine

Reward Machine

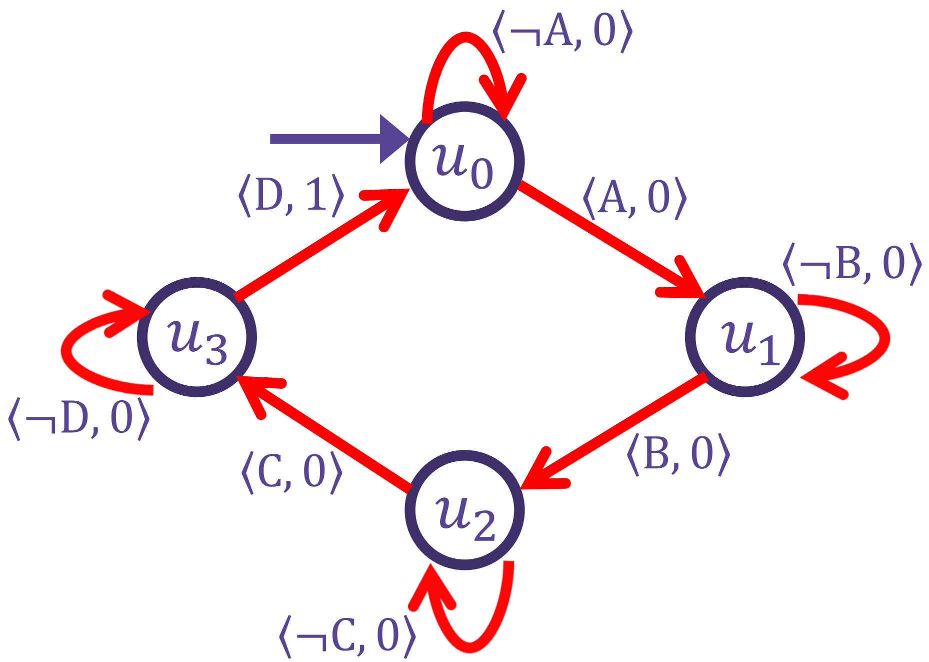
- finite set of states U
- initial state $u_0 \in U$



Reward Machine

Reward Machine

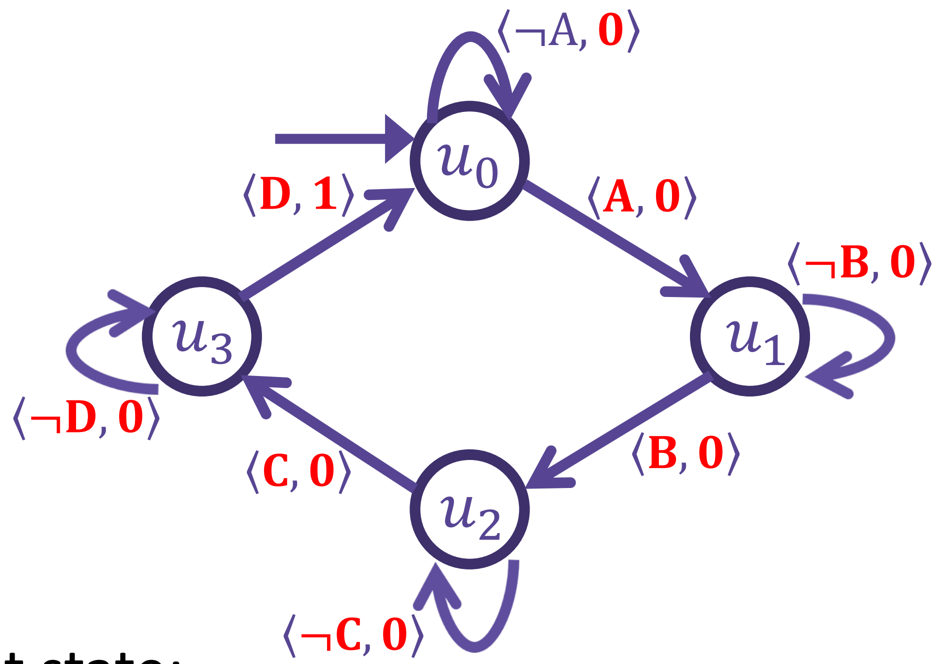
- finite set of states U
- initial state $u_0 \in U$
- set of transitions labelled by:



Reward Machine

Reward Machine

- finite set of states U
- initial state $u_0 \in U$
- set of transitions labelled by:
  - A logical condition (guards)
  - A reward function (or constant)



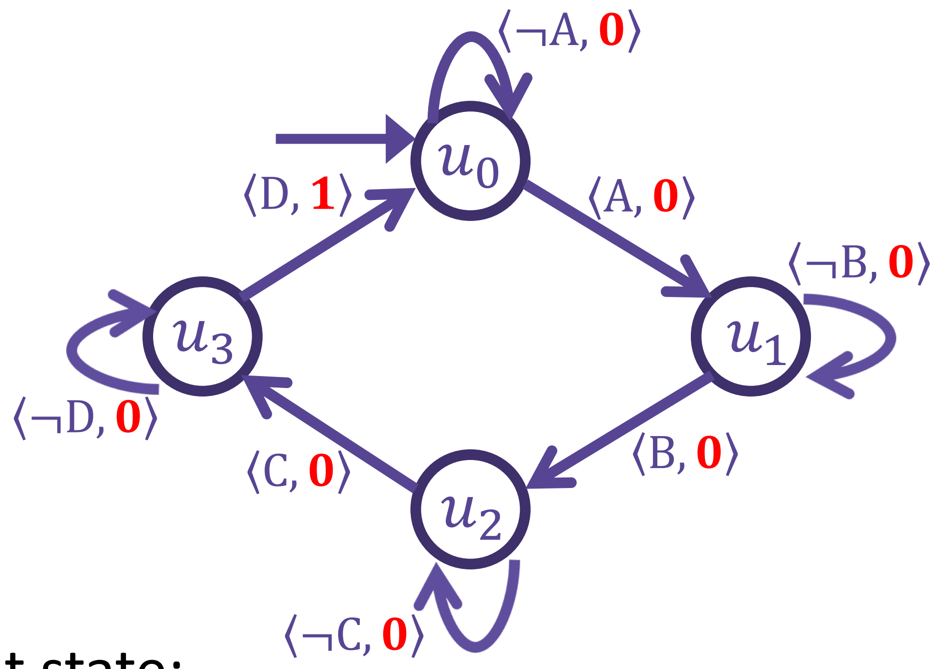
Conditions are over properties of the current state:

$$P = \{☕, ✉, o, *, A, B, C, D\}$$

Reward Machine

Reward Machine

- finite set of states U
- initial state $u_0 \in U$
- set of transitions labelled by:
  - A logical condition (guards)
  - A reward function (or constant)



Conditions are over properties of the current state:

$$P = \{☕, ✉, o, *, A, B, C, D\}$$

A Reward Machine is a **Mealy Machine** over the input alphabet $\Sigma = 2^P$, whose output alphabet is a set of Markovian reward functions.

Reward Machine

Definition 3.1 (reward machine). *Given a set of propositional symbols $\mathcal{P}$, a set of (environment) states S, and a set of actions A, a reward machine (RM) is a tuple $\mathcal{R}_{\mathcal{P}SA} = \langle U, u_0, F, \delta_u, \delta_r \rangle$ where U is a finite set of states, $u_0 \in U$ is an initial state, F is a finite set of terminal states (where $U \cap F = \emptyset$), δ_u is the state-transition function, $\delta_u : U \times 2^{\mathcal{P}} \to U \cup F$, and δ_r is the state-reward function, $\delta_r : U \to [S \times A \times S \to \mathbb{R}]$.*

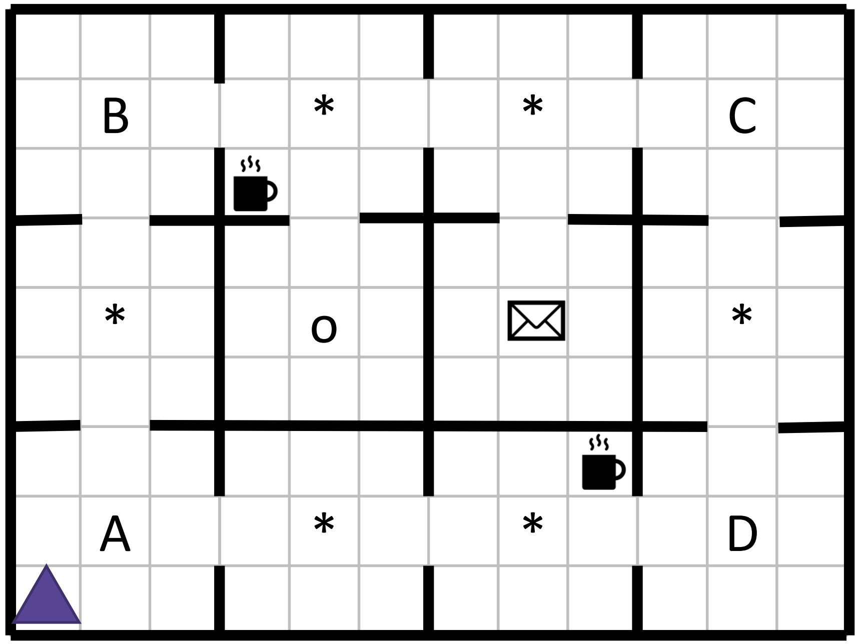
[Toro Icarte et al., ICML18]
[Camacho et al., IJCAI19]
[Toro Icarte et al., forthcoming]

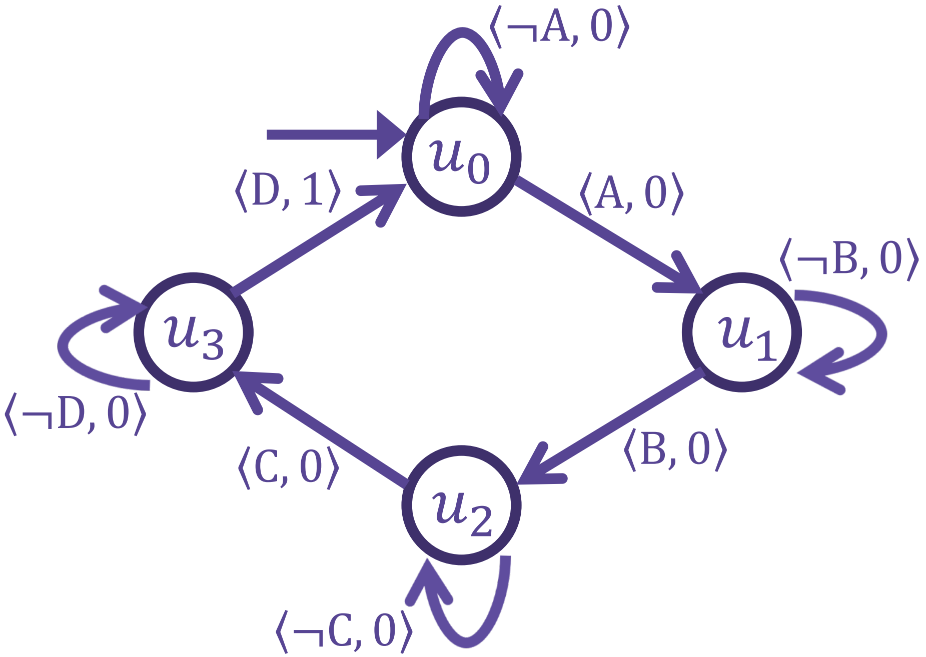
Simple Reward Machine

Definition 3.2 (simple reward machine). *Given a set of propositional symbols $\mathcal{P}$, a simple reward machine is a tuple $\mathcal{R} = \langle U, u_0, F, \delta_u, \delta_r \rangle$ where U, u_0, F, and δ_u are defined as in a standard reward machine, but the state-reward function $\delta_r : U \times 2^{\mathcal{P}} \to \mathbb{R}$ depends on $2^{\mathcal{P}}$ and returns a number instead of a function.*

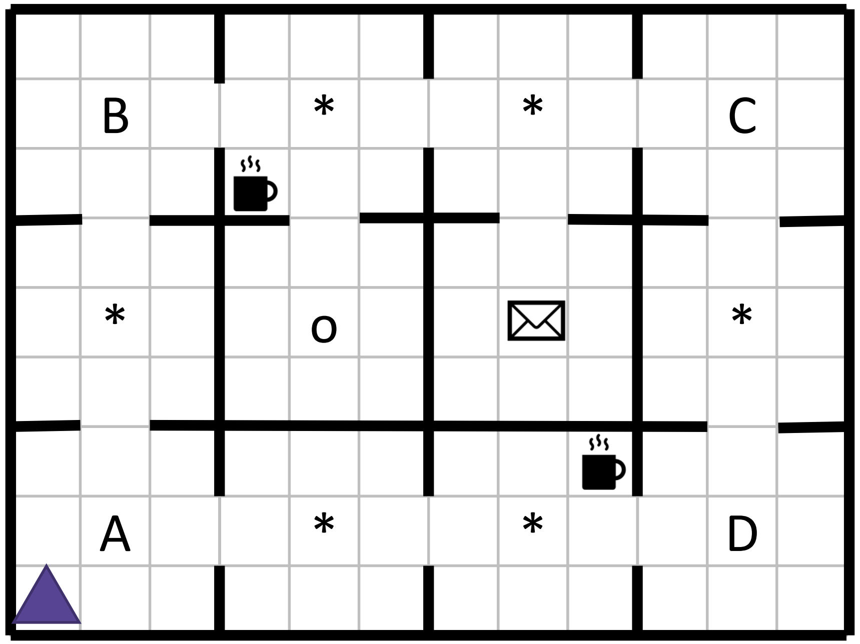
[Toro Icarte et al., ICML18]
[Camacho et al., IJCAI19]
[Toro Icarte et al., forthcoming]

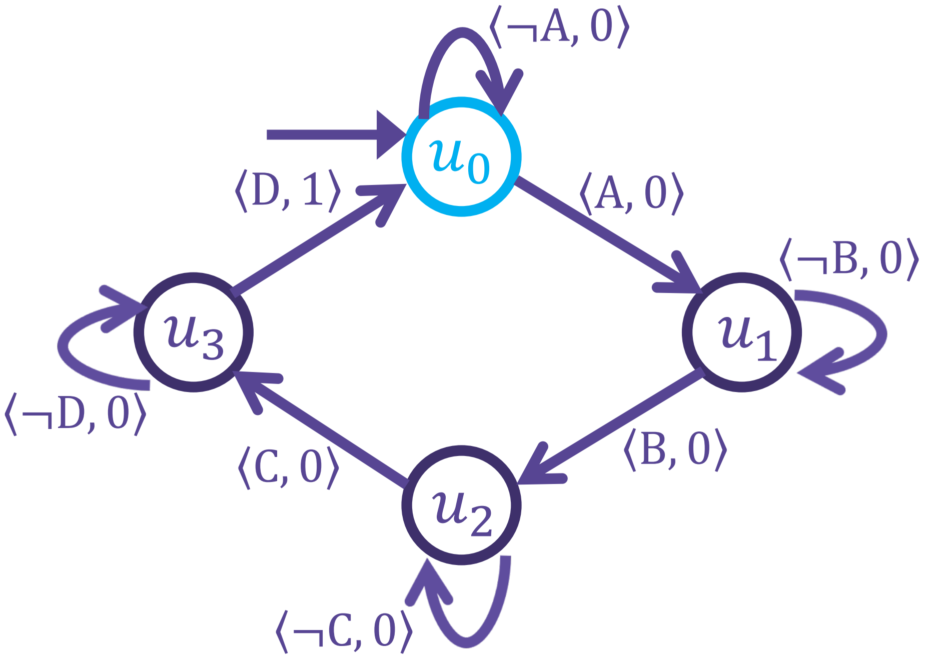
Reward Machines in Action



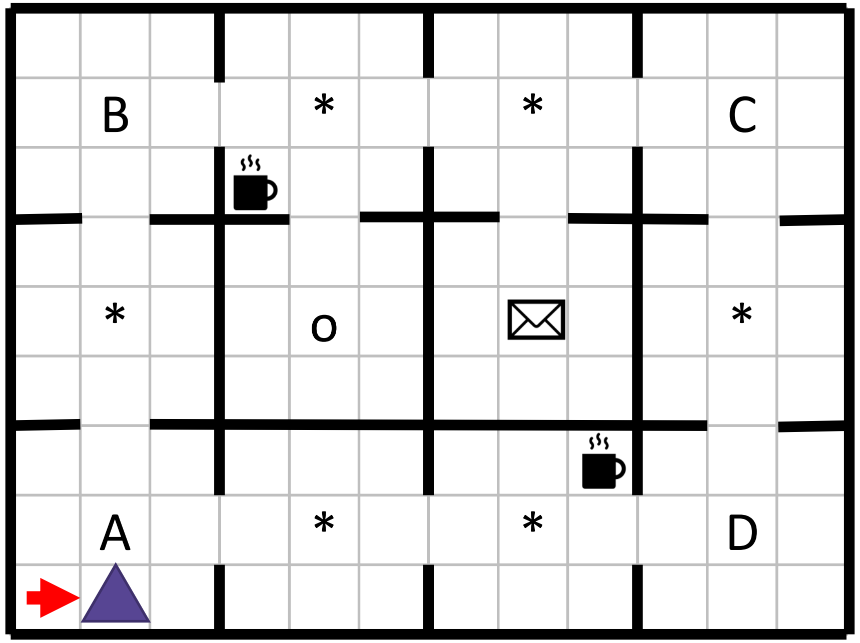


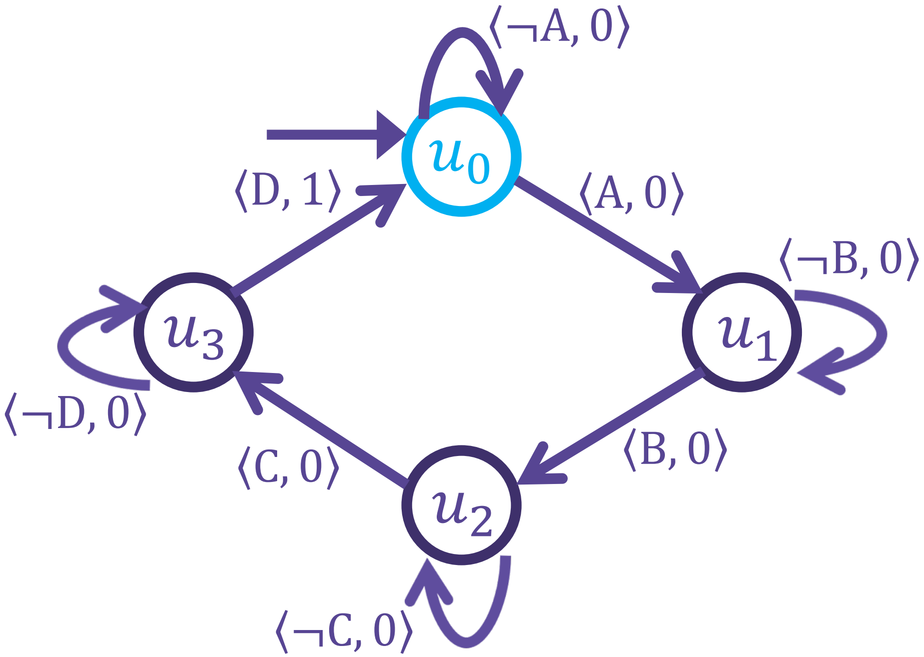
Reward Machines in Action



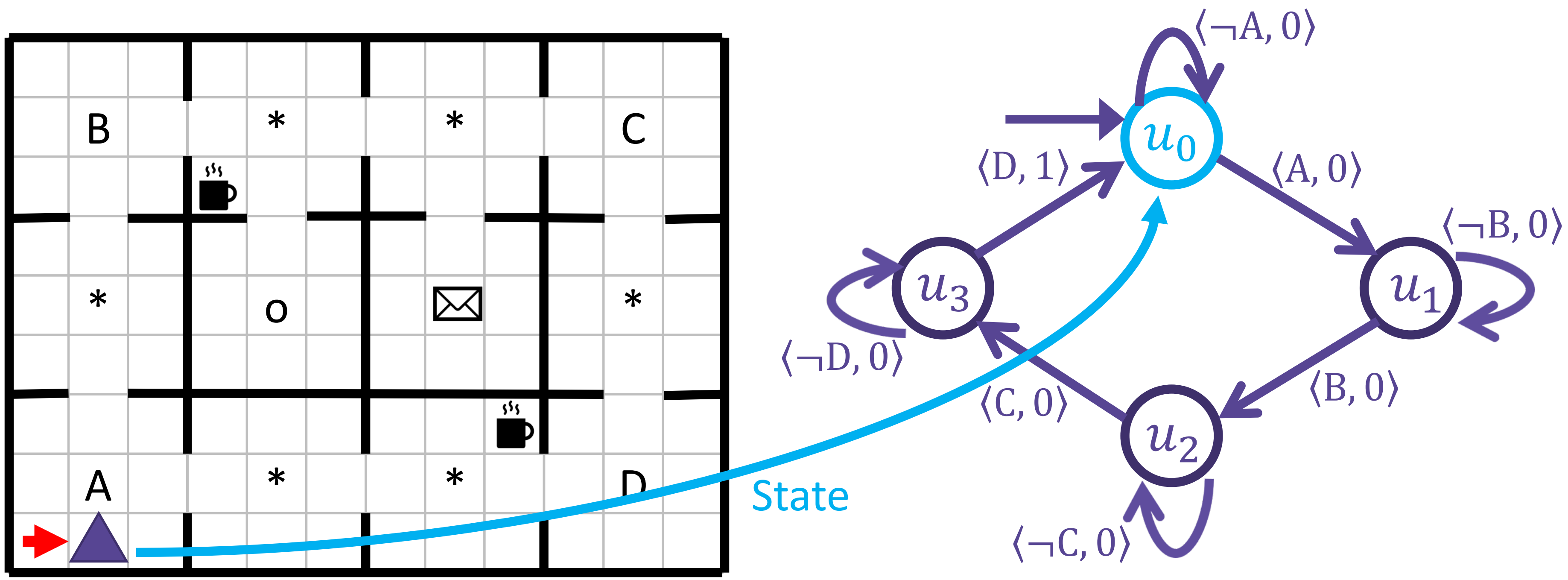


Reward Machines in Action

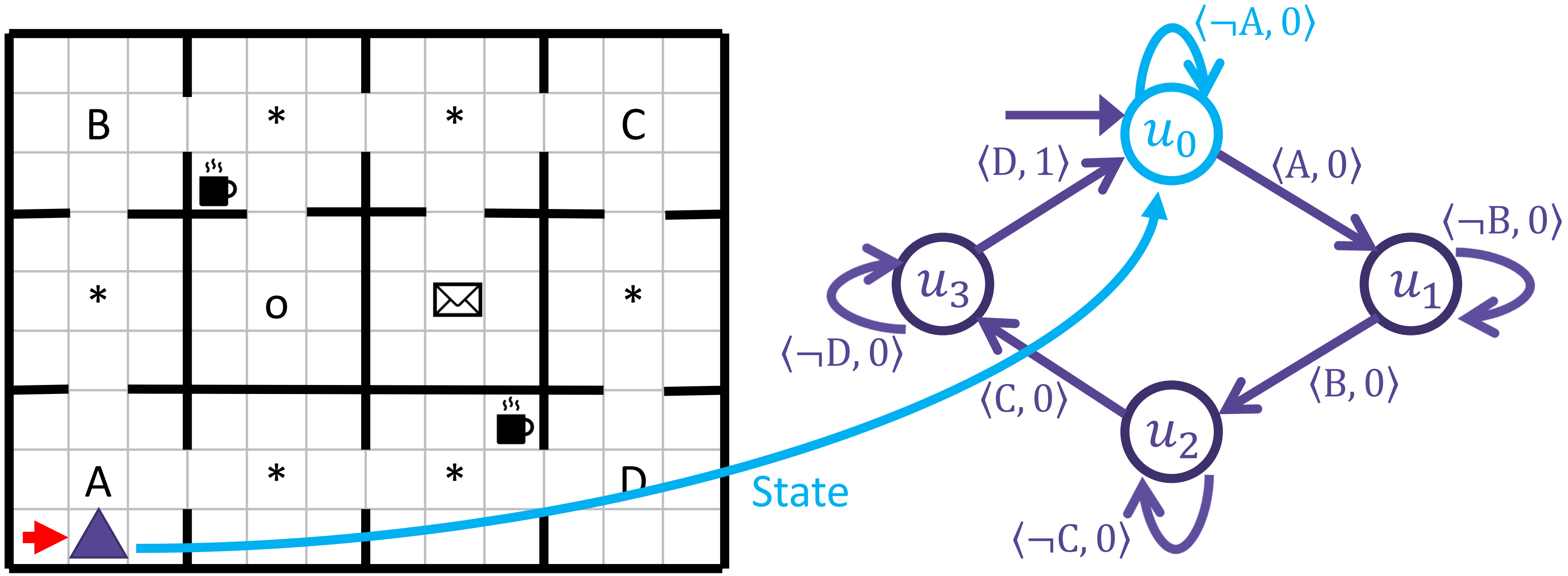




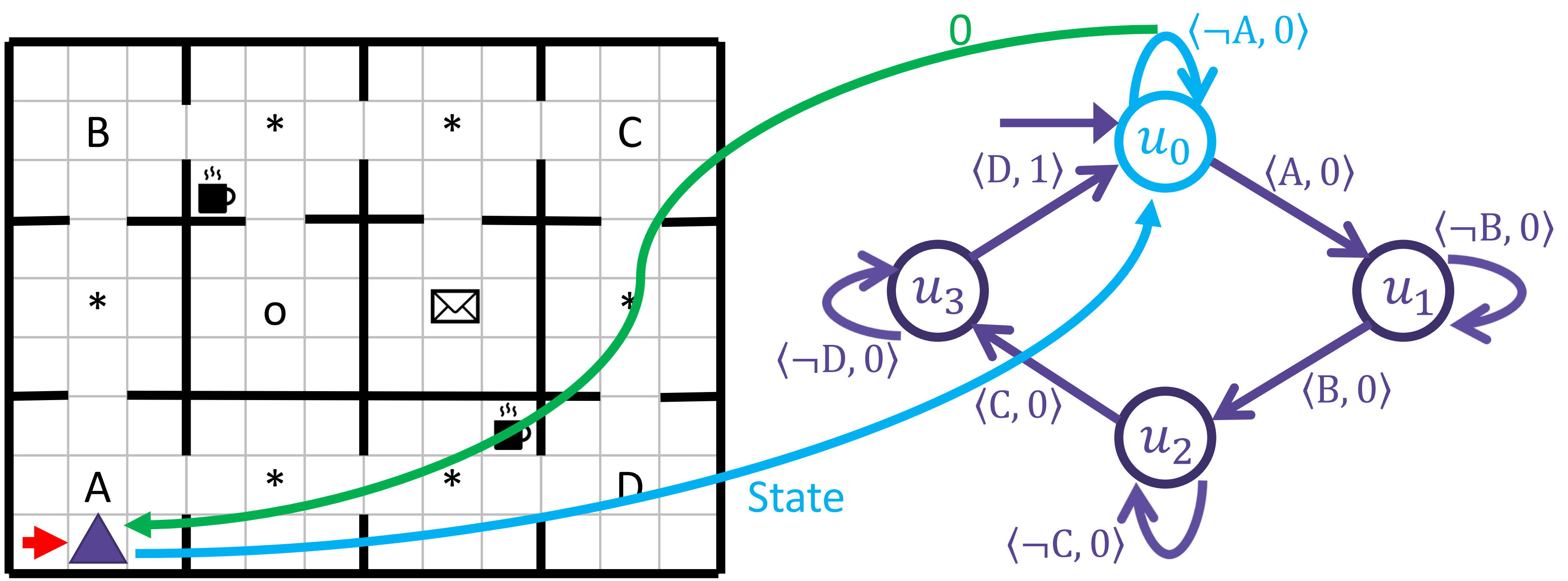
Reward Machines in Action



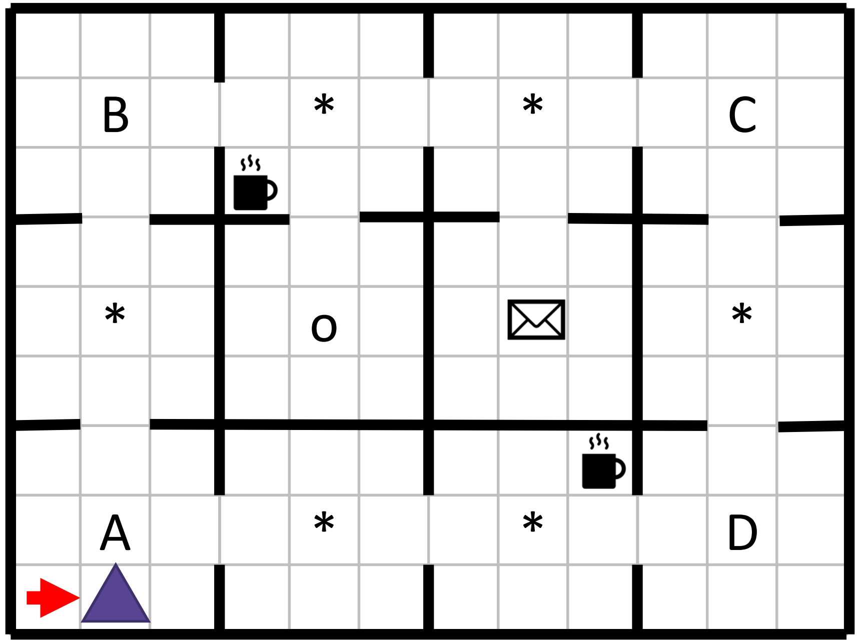
Reward Machines in Action

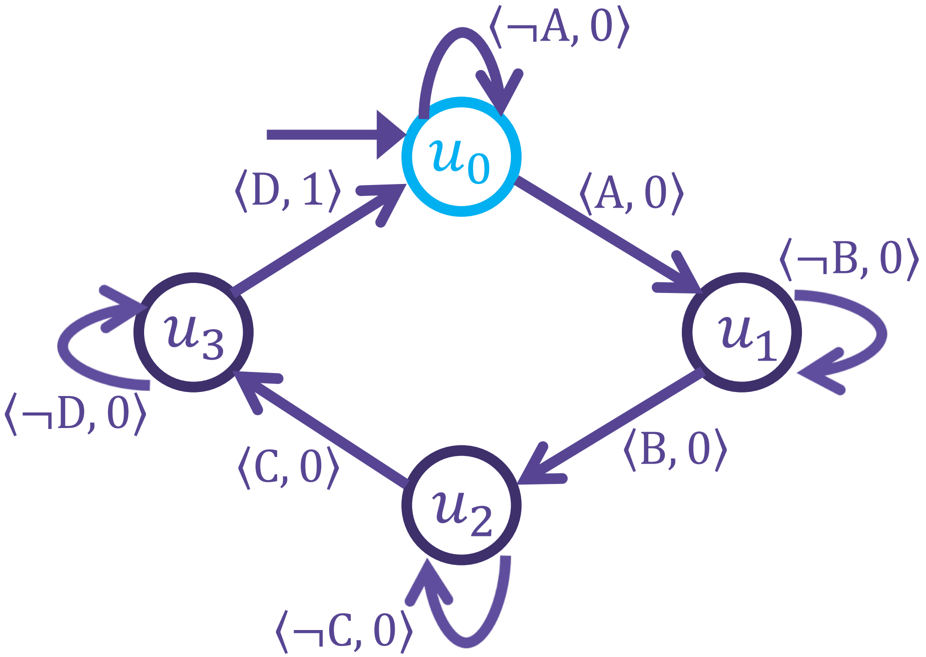


Reward Machines in Action

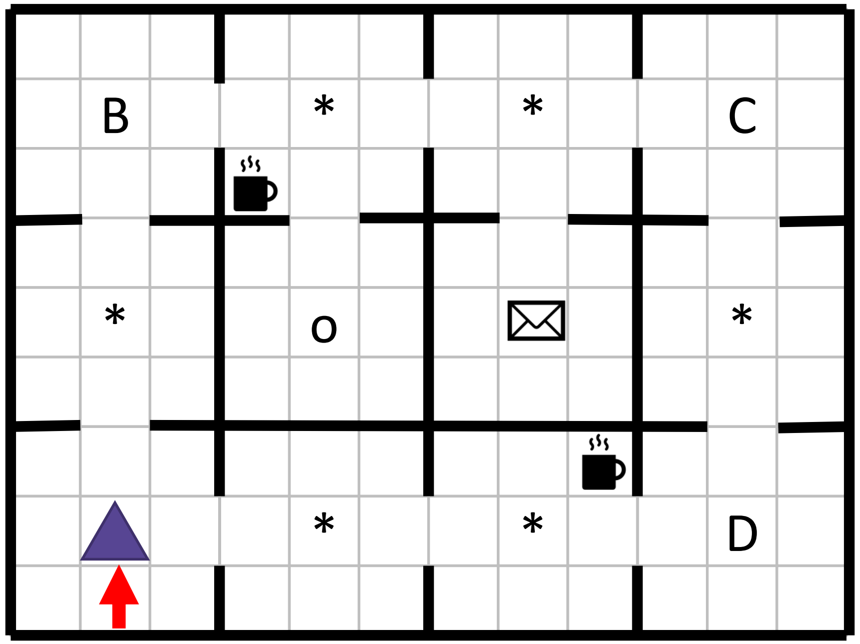


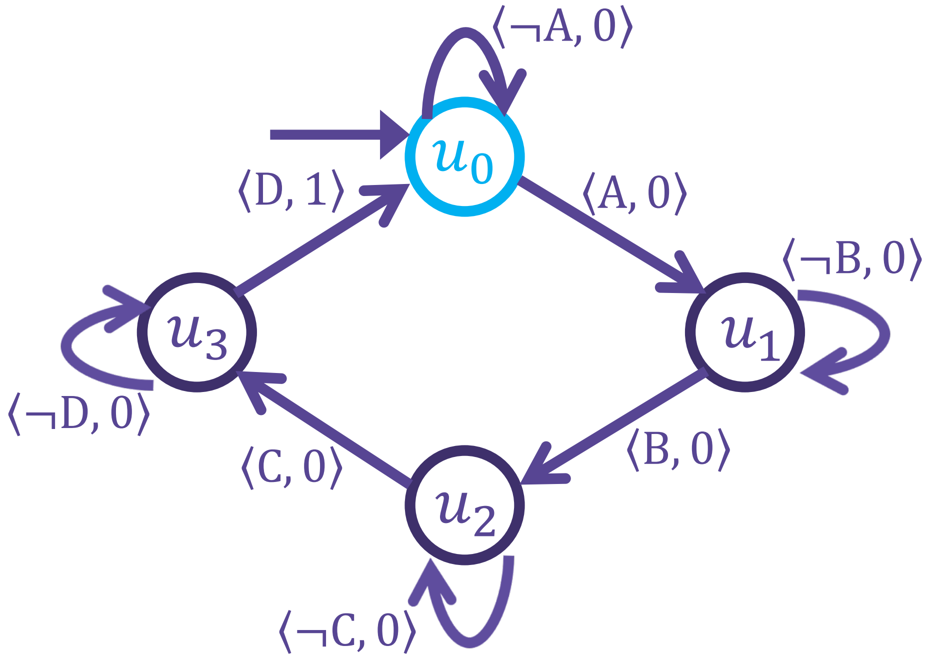
Reward Machines in Action



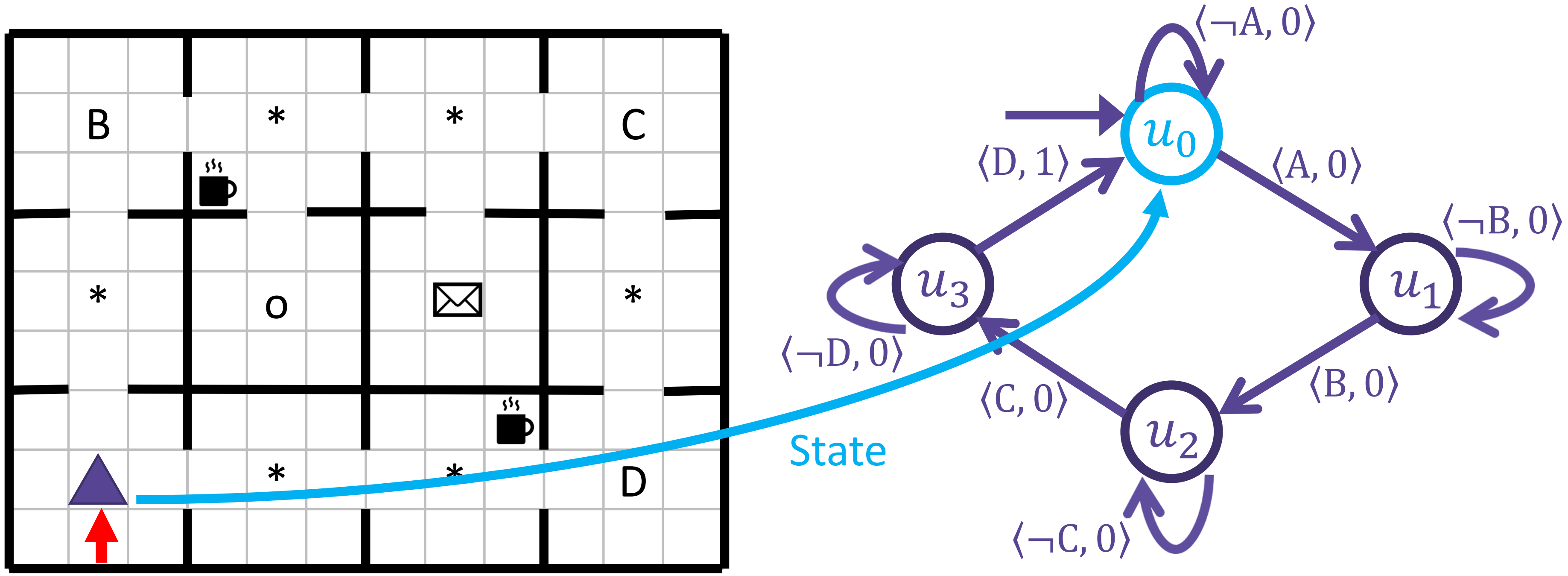


Reward Machines in Action

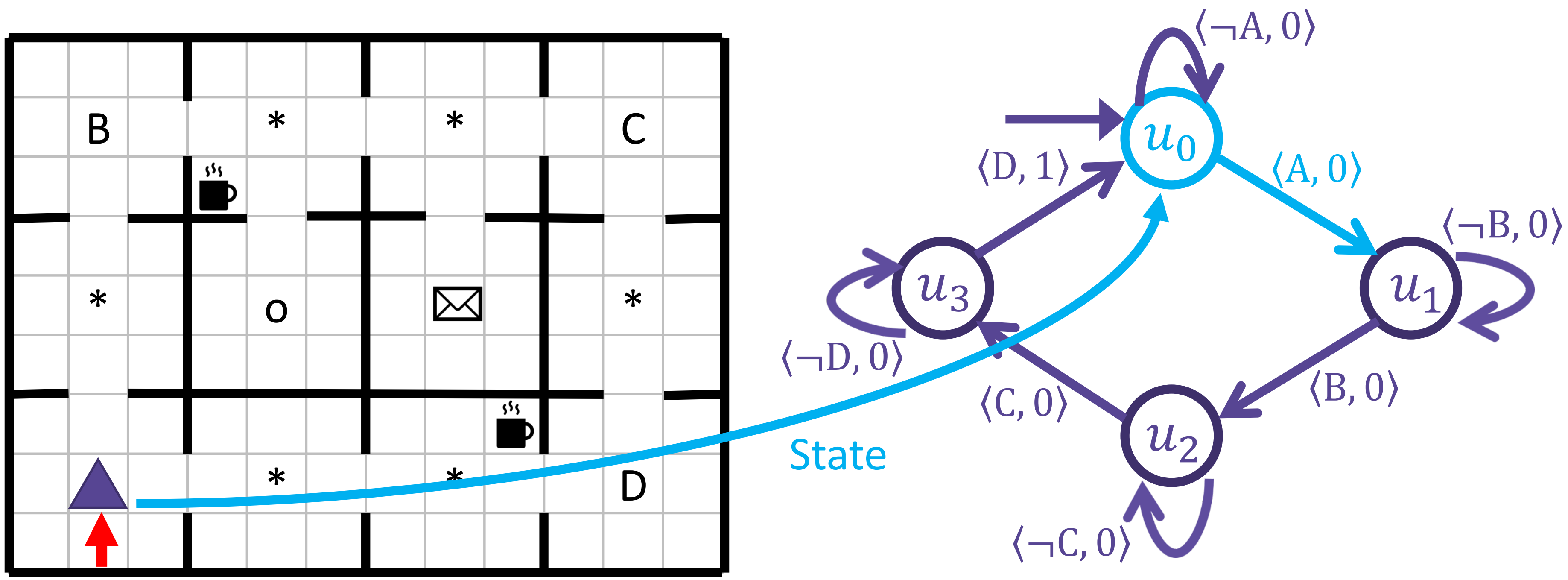




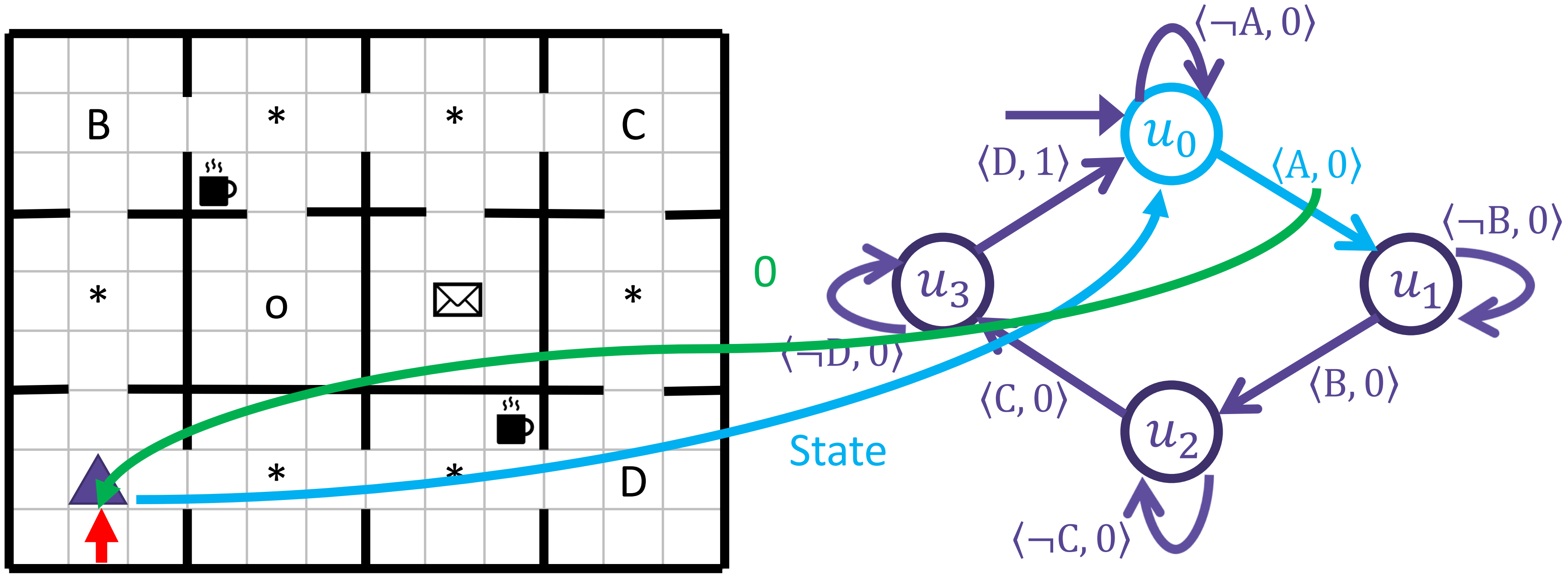
Reward Machines in Action



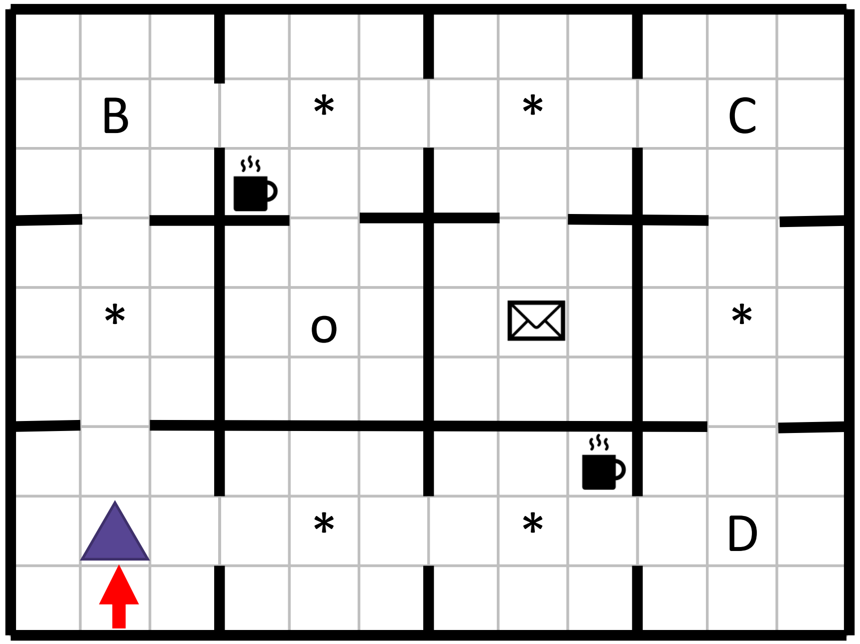
Reward Machines in Action

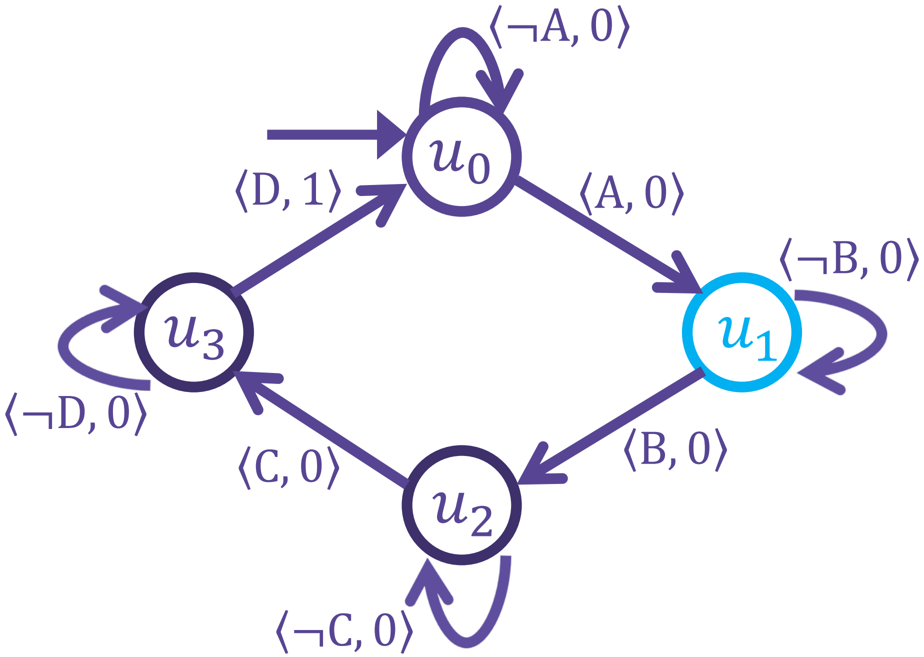


Reward Machines in Action

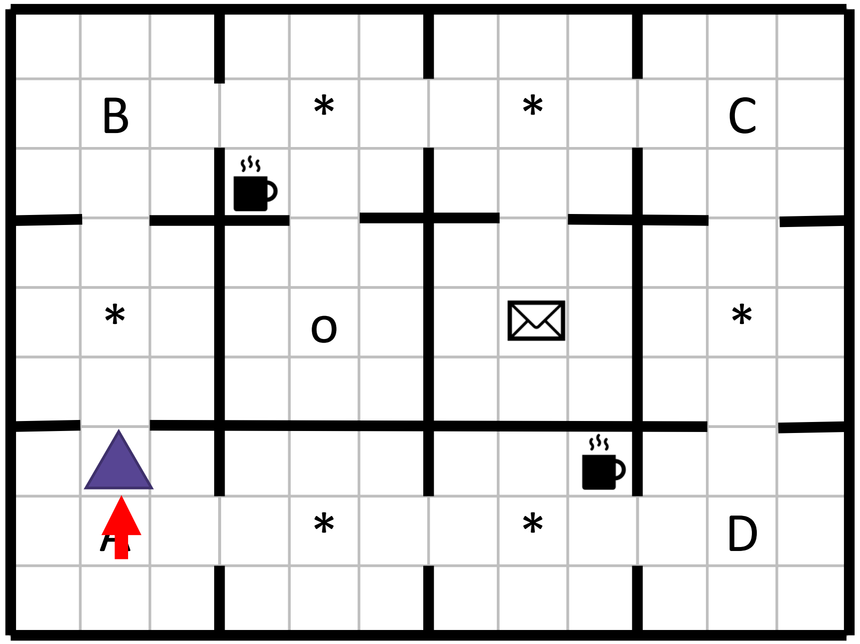


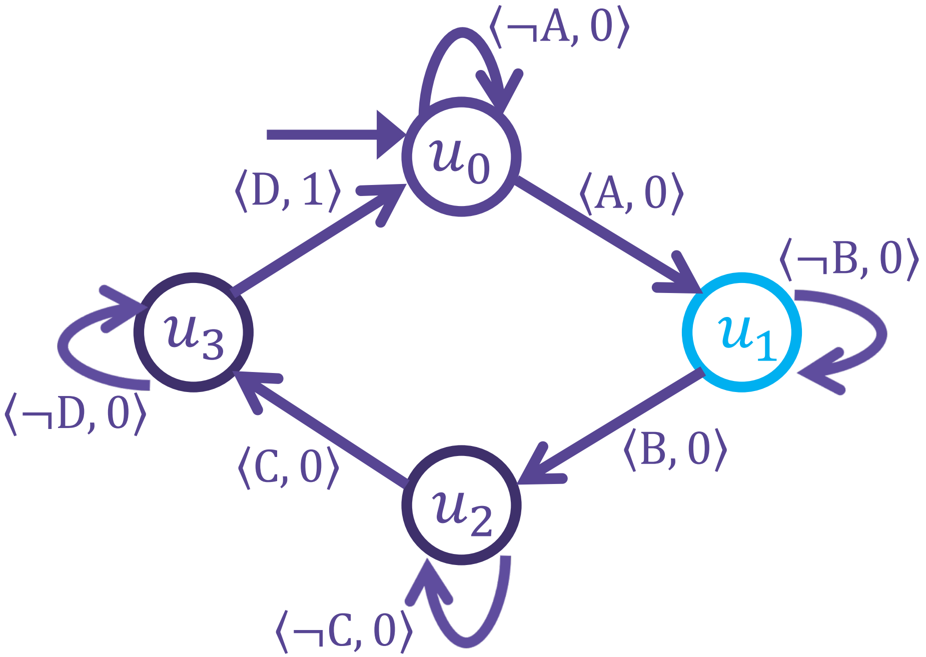
Reward Machines in Action



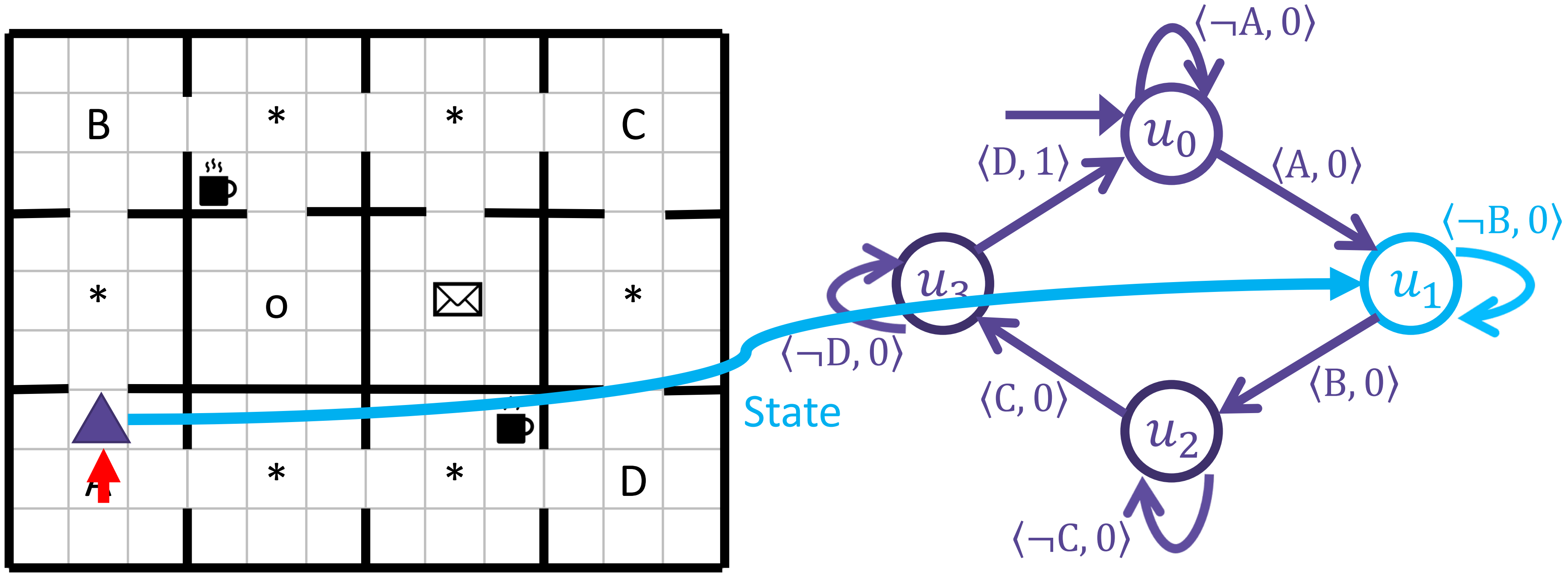


Reward Machines in Action

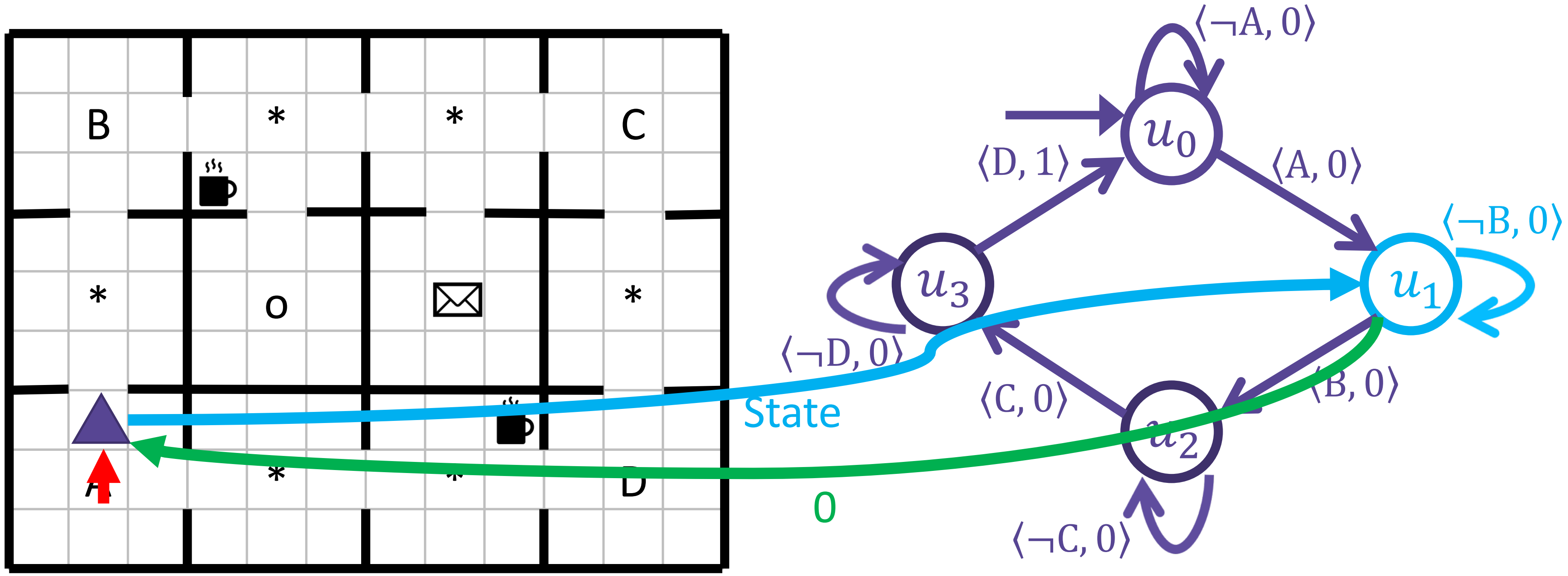




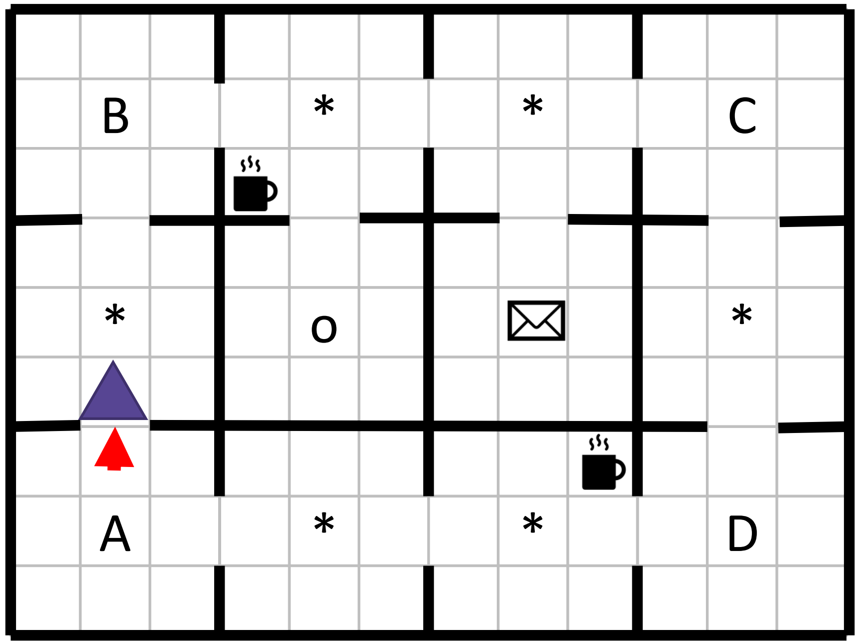
Reward Machines in Action

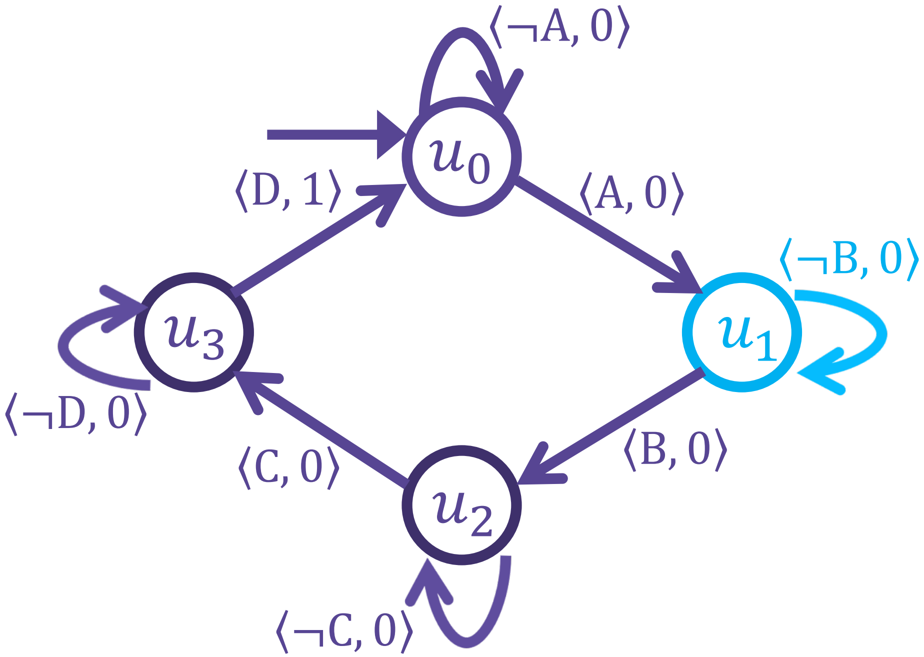


Reward Machines in Action

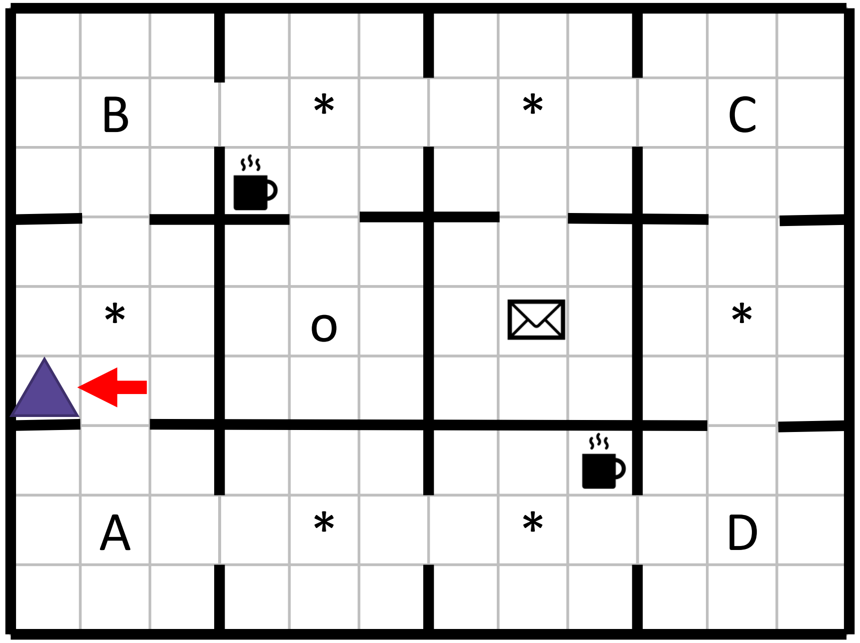


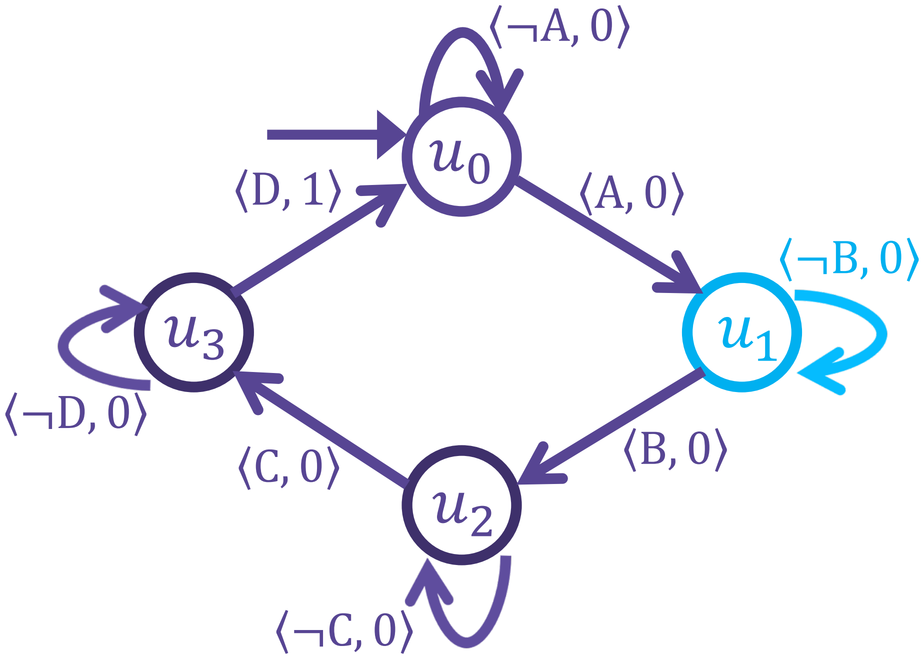
Reward Machines in Action



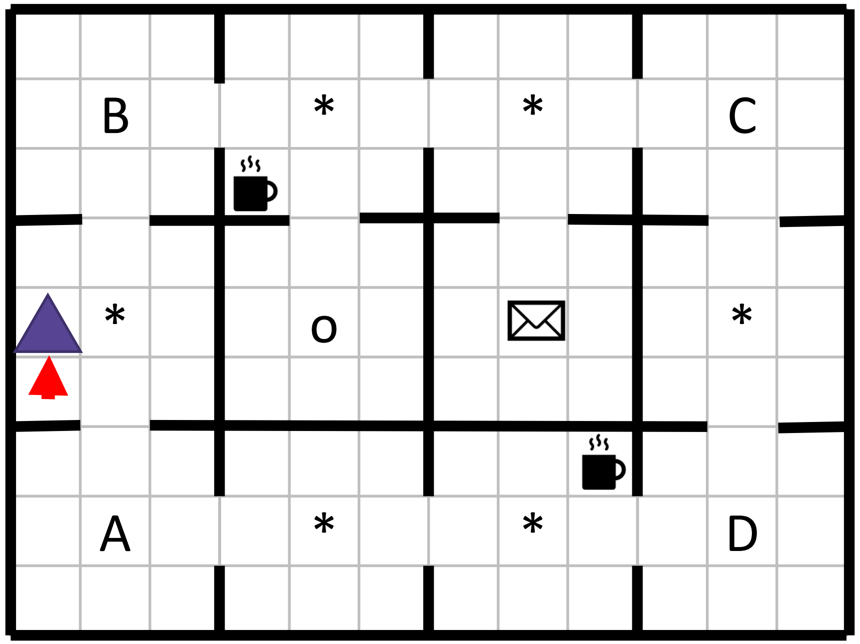


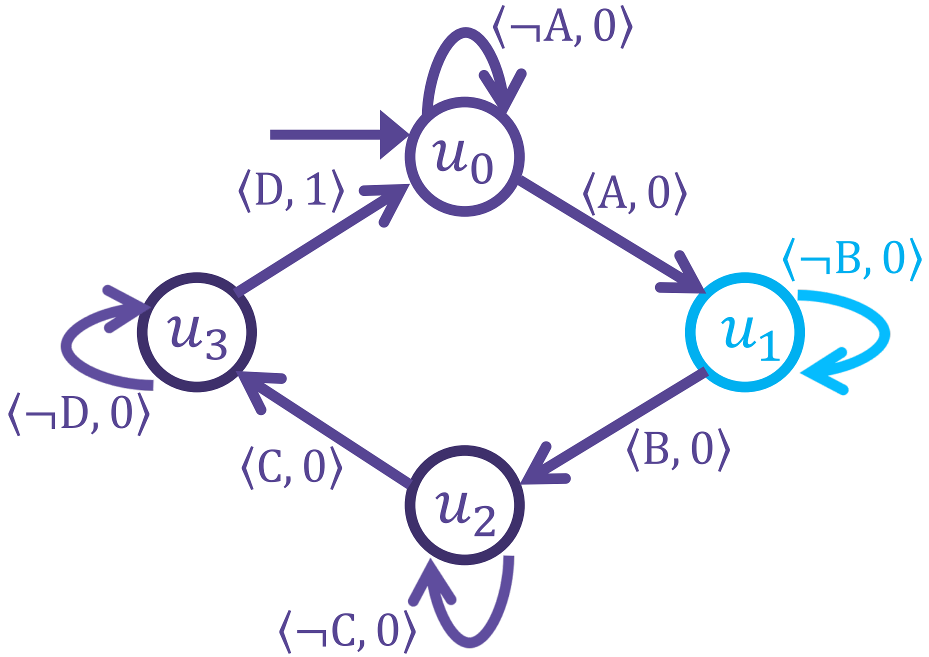
Reward Machines in Action



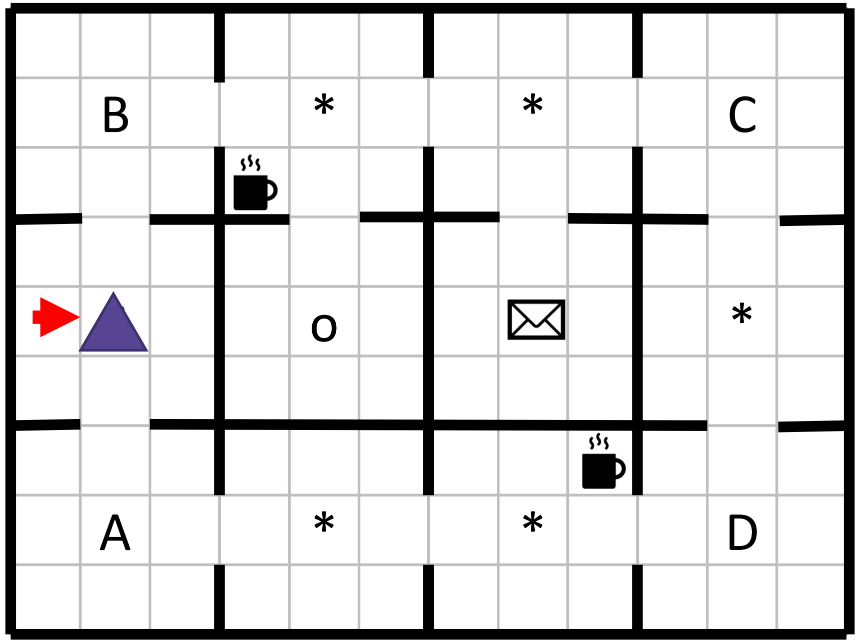


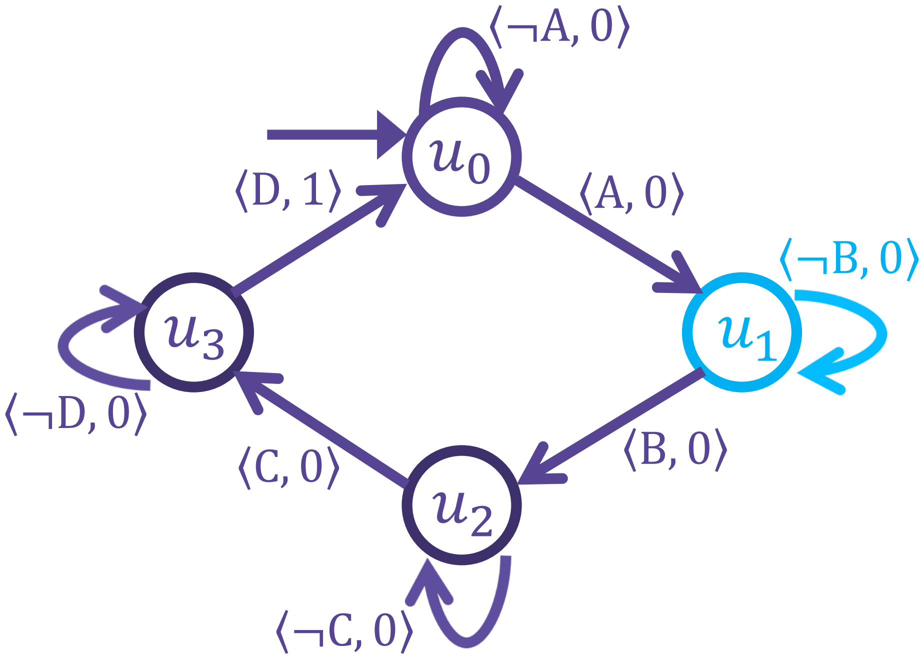
Reward Machines in Action



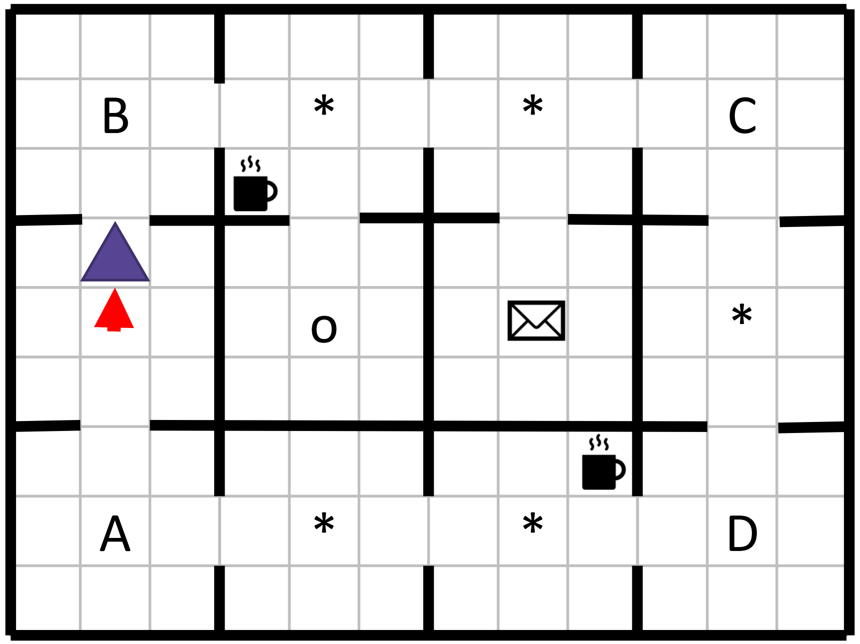


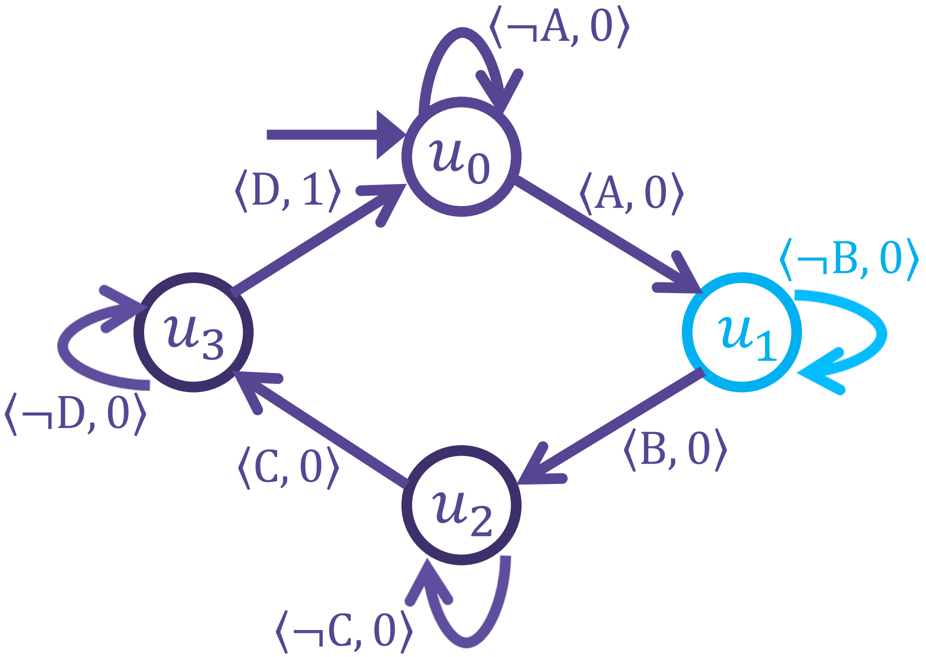
Reward Machines in Action



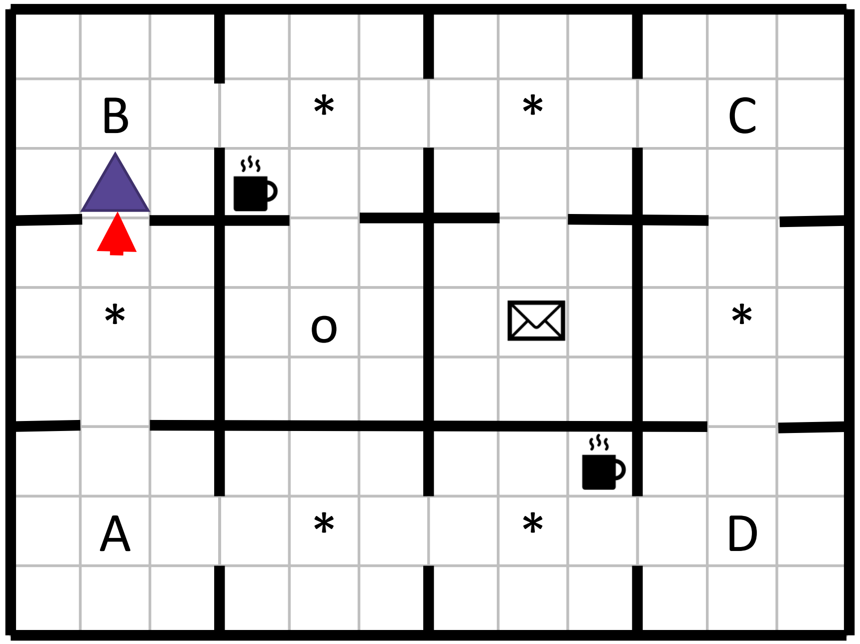


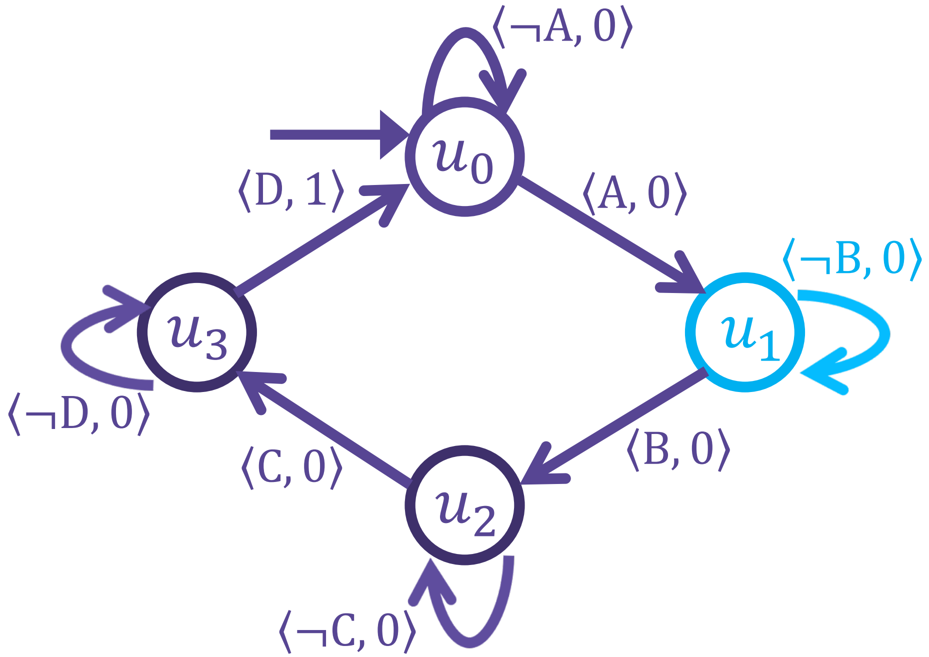
Reward Machines in Action



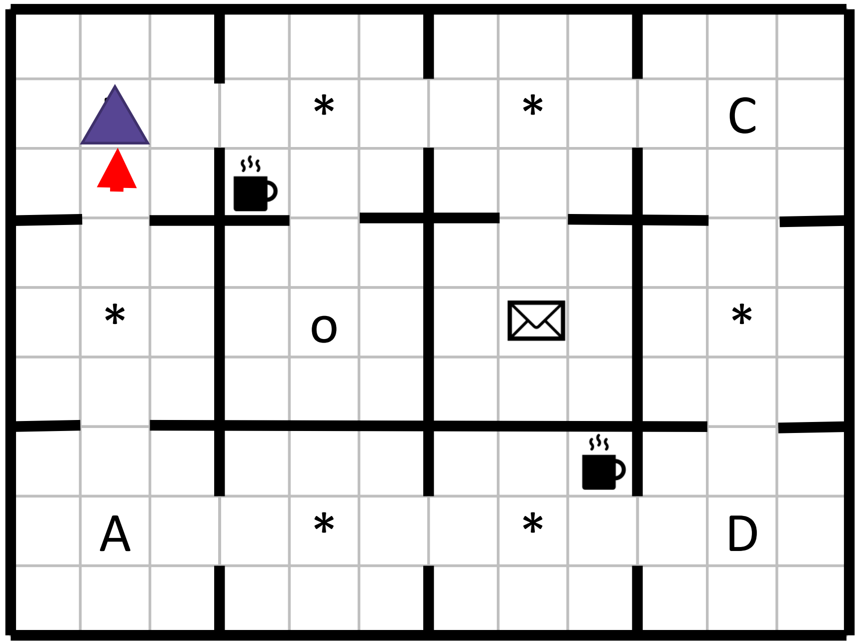


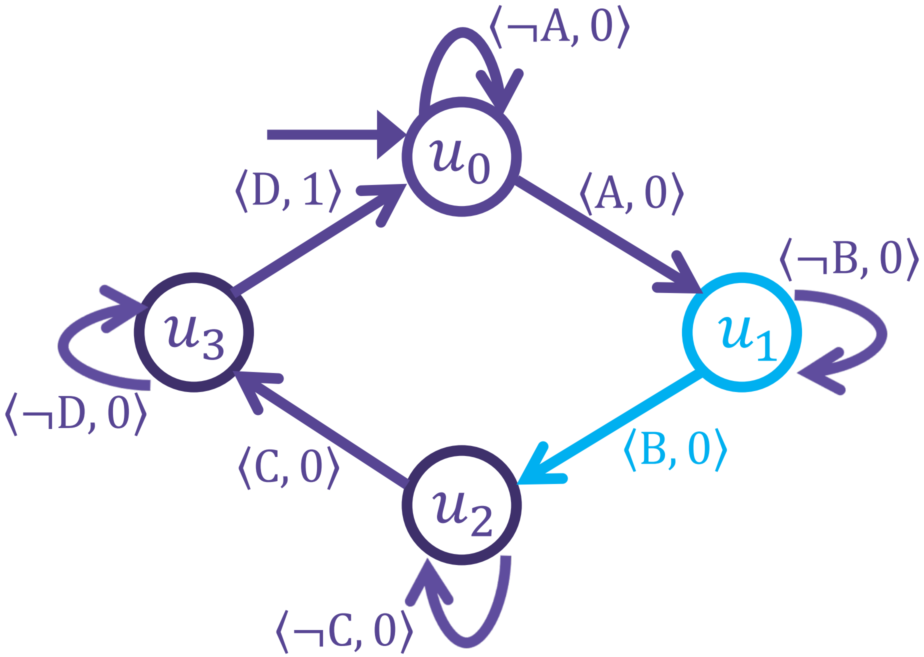
Reward Machines in Action



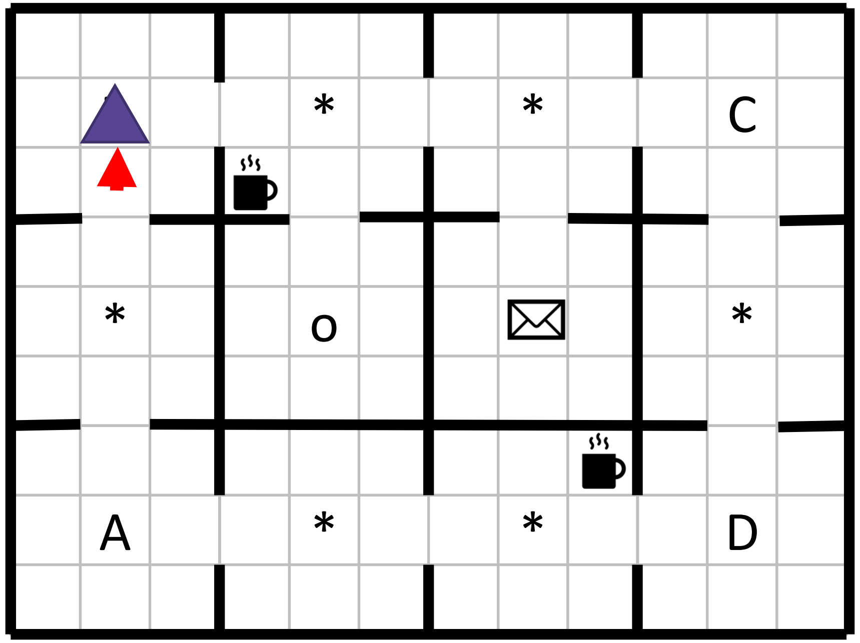


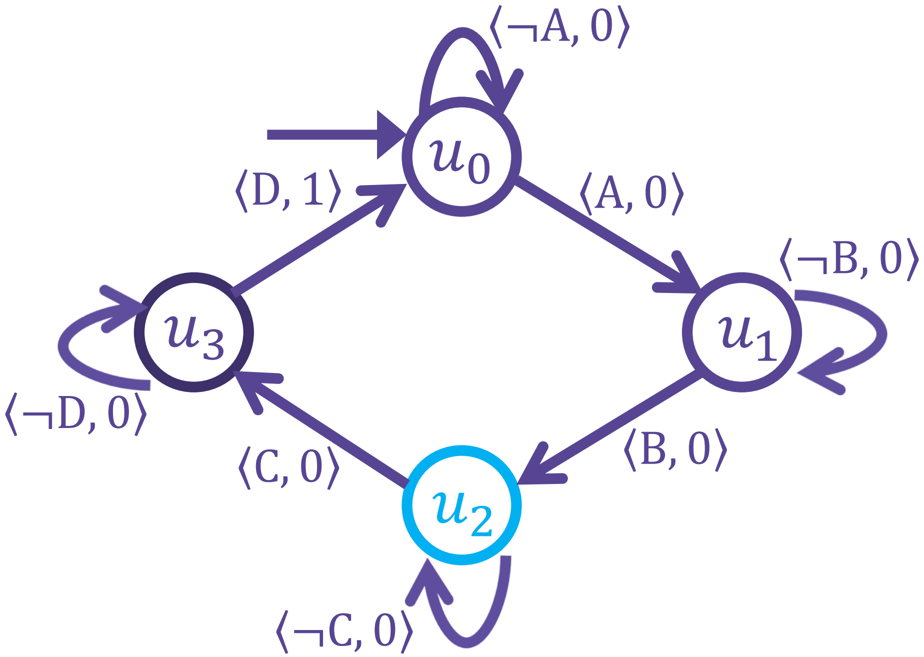
Reward Machines in Action



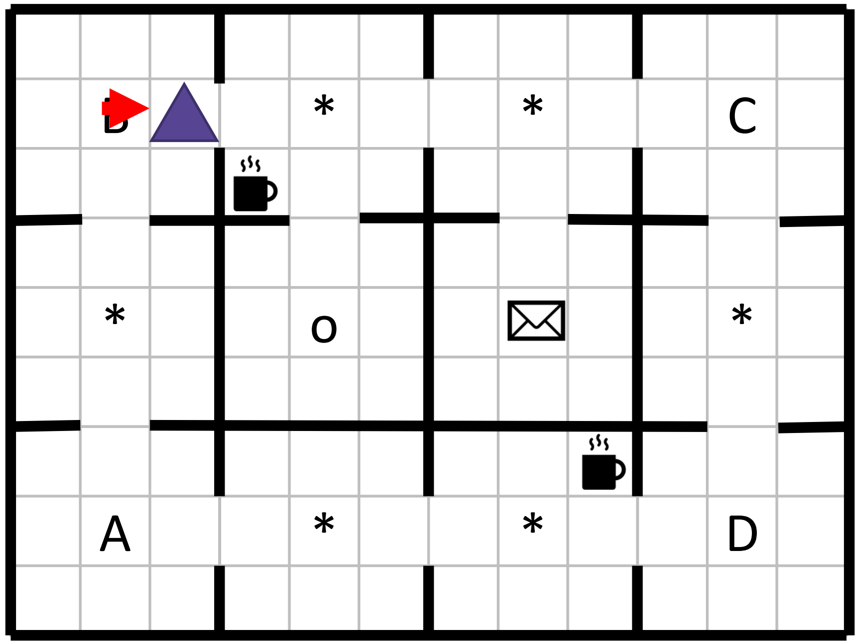


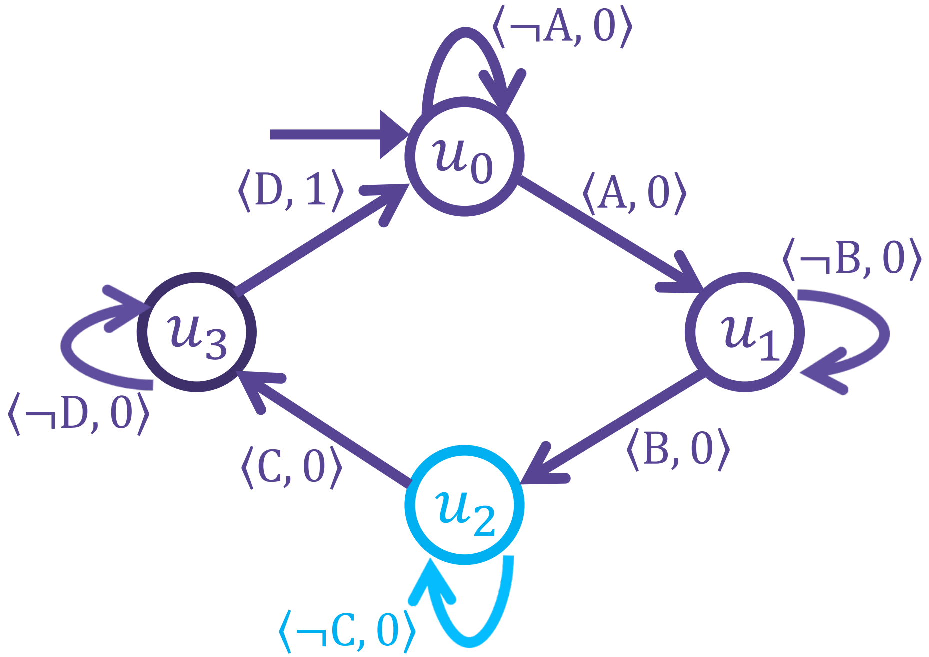
Reward Machines in Action



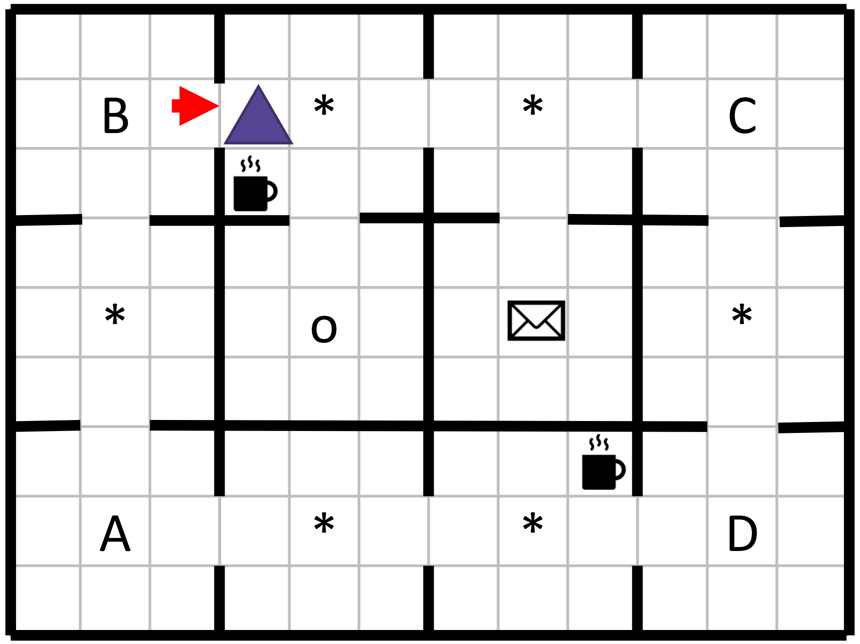


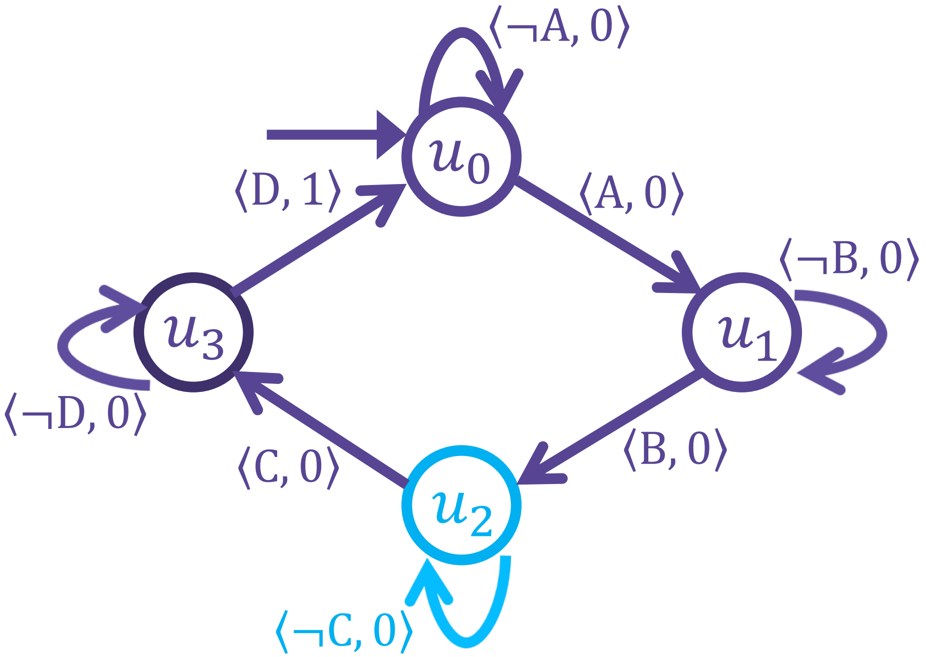
Reward Machines in Action



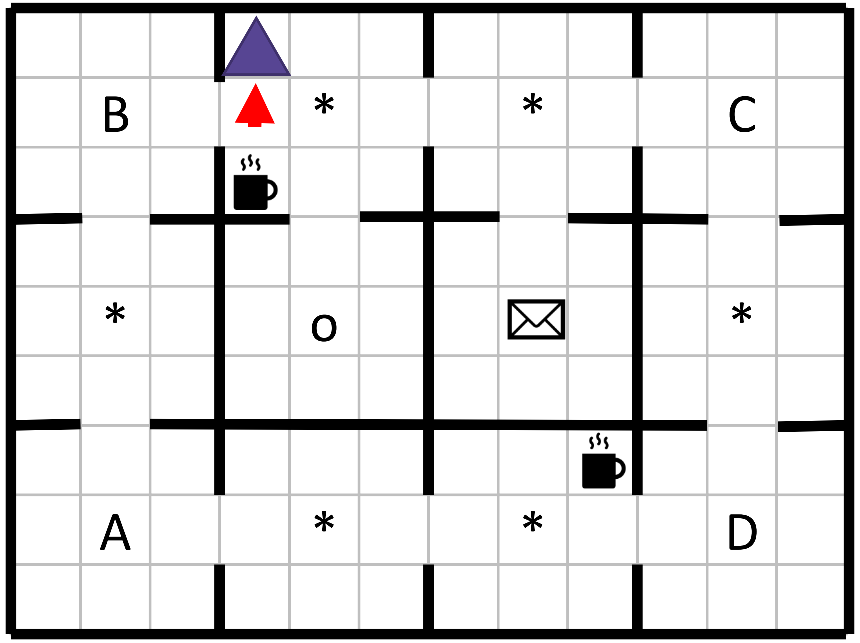


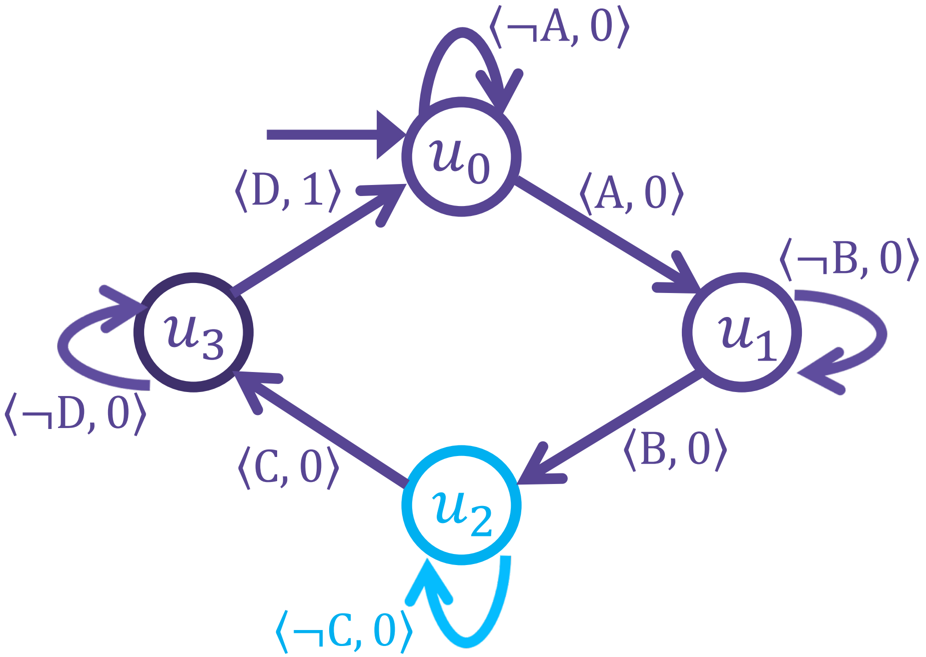
Reward Machines in Action



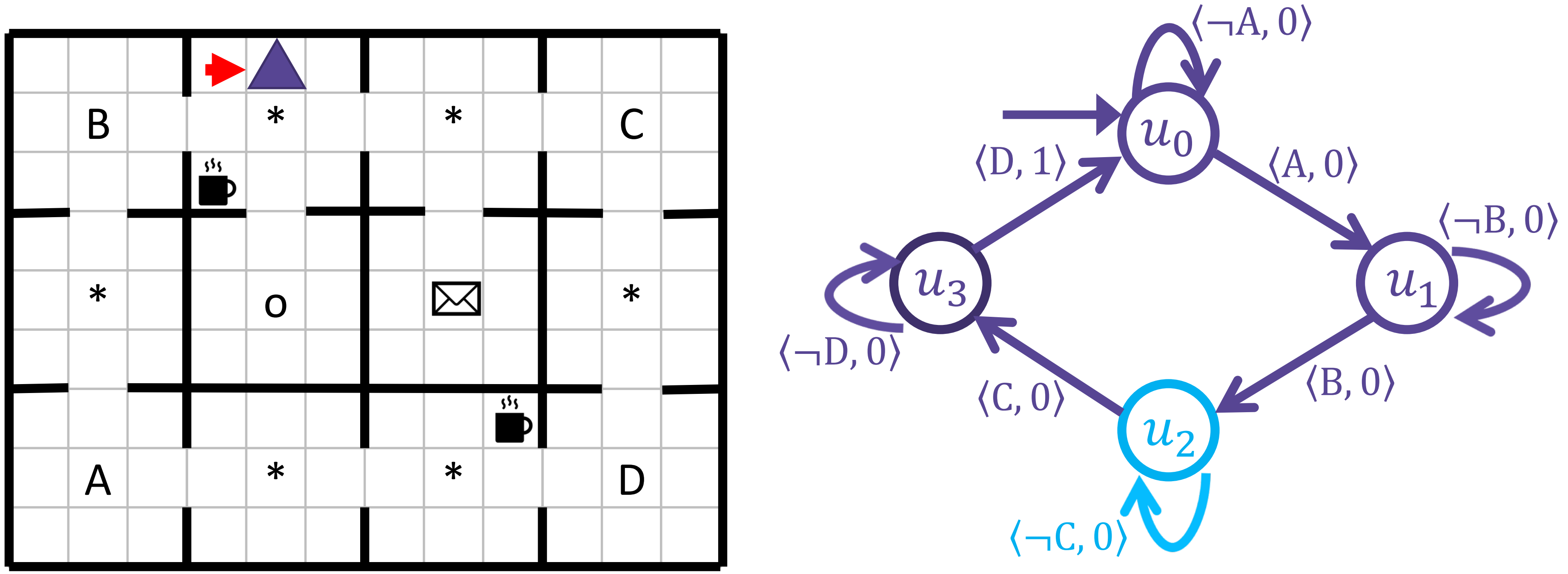


Reward Machines in Action

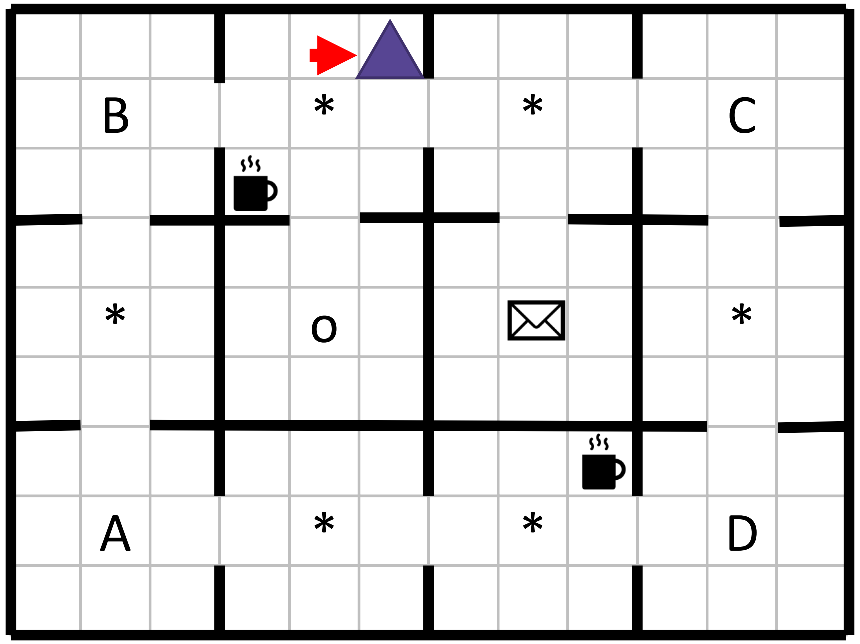


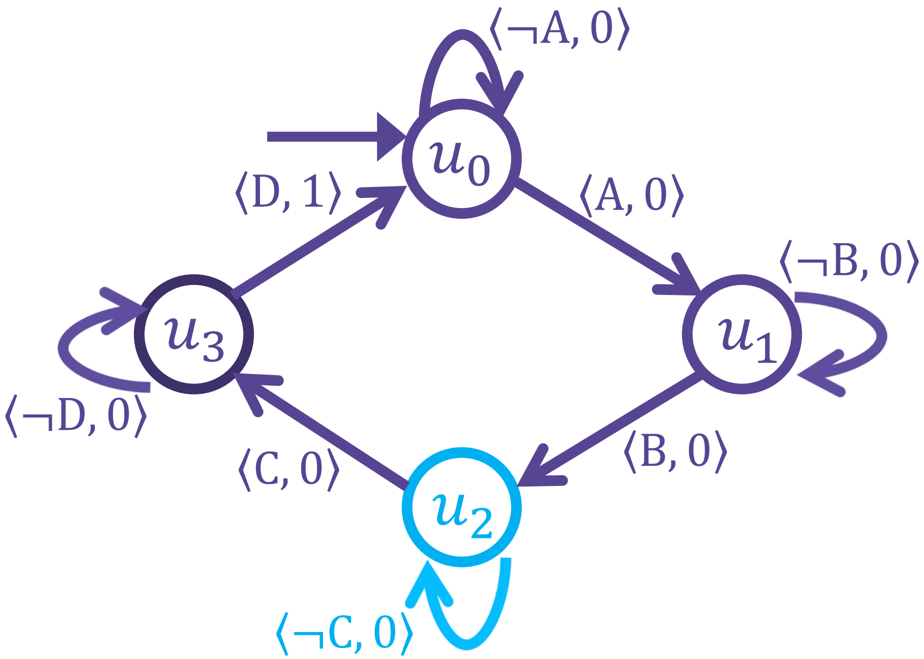


Reward Machines in Action

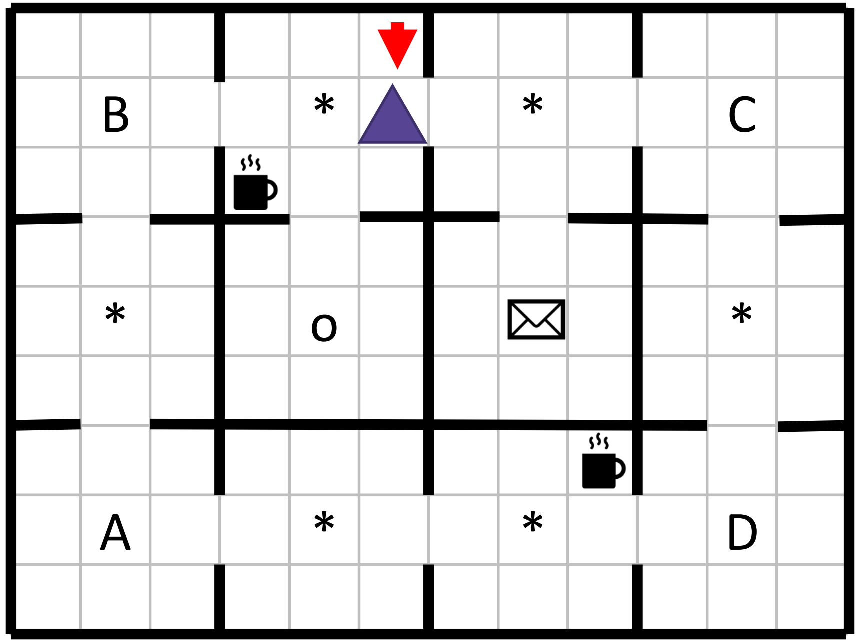


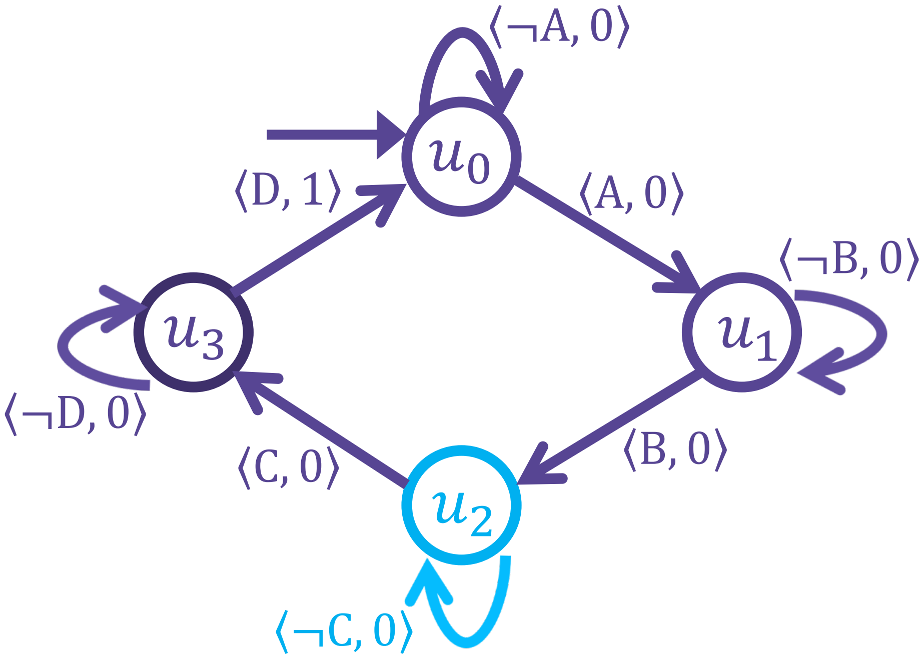
Reward Machines in Action



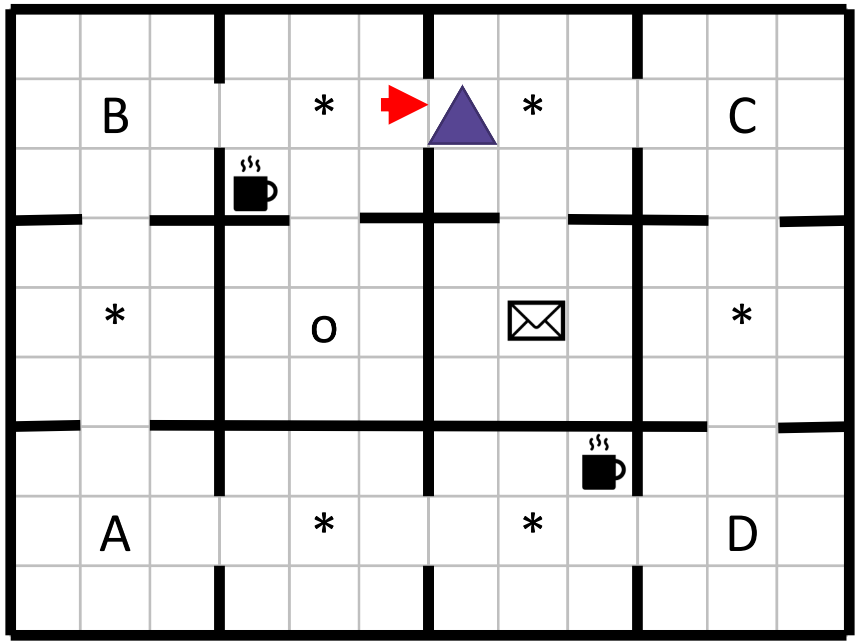


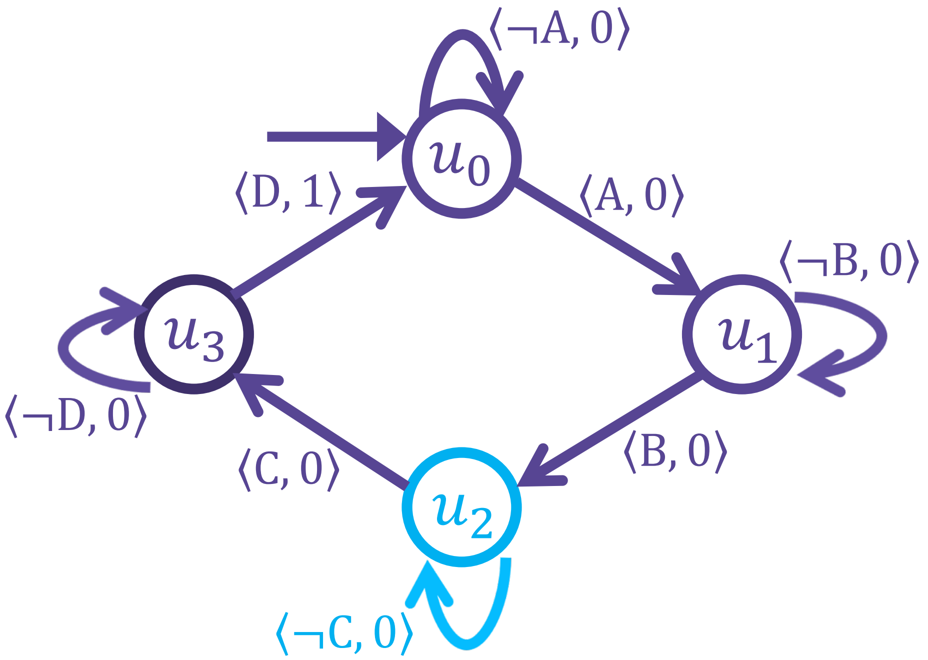
Reward Machines in Action



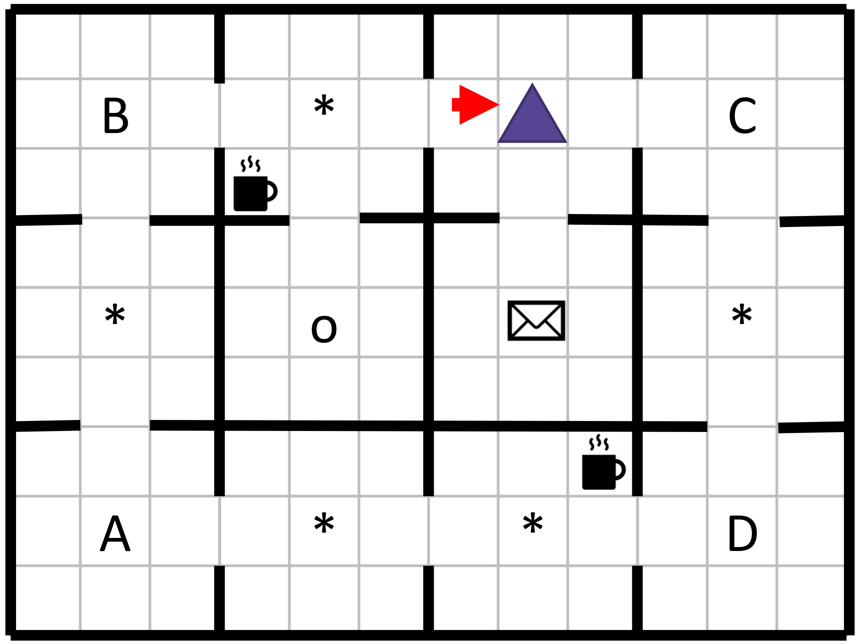


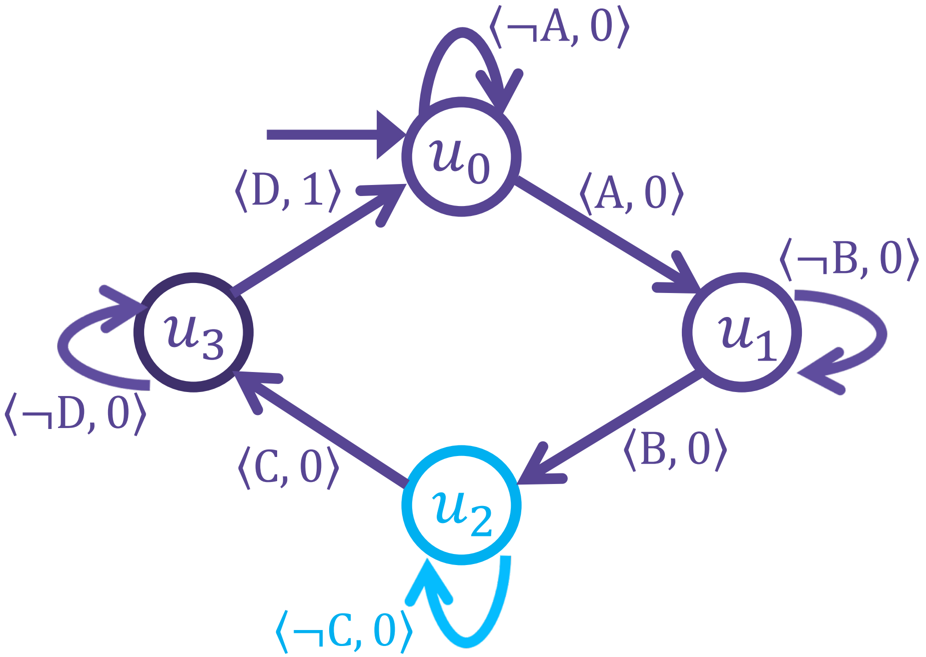
Reward Machines in Action



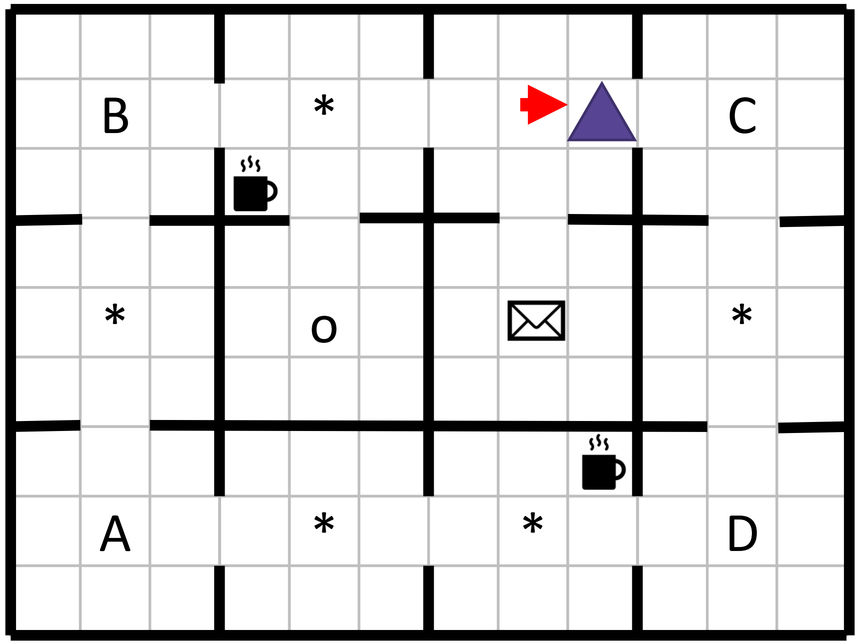


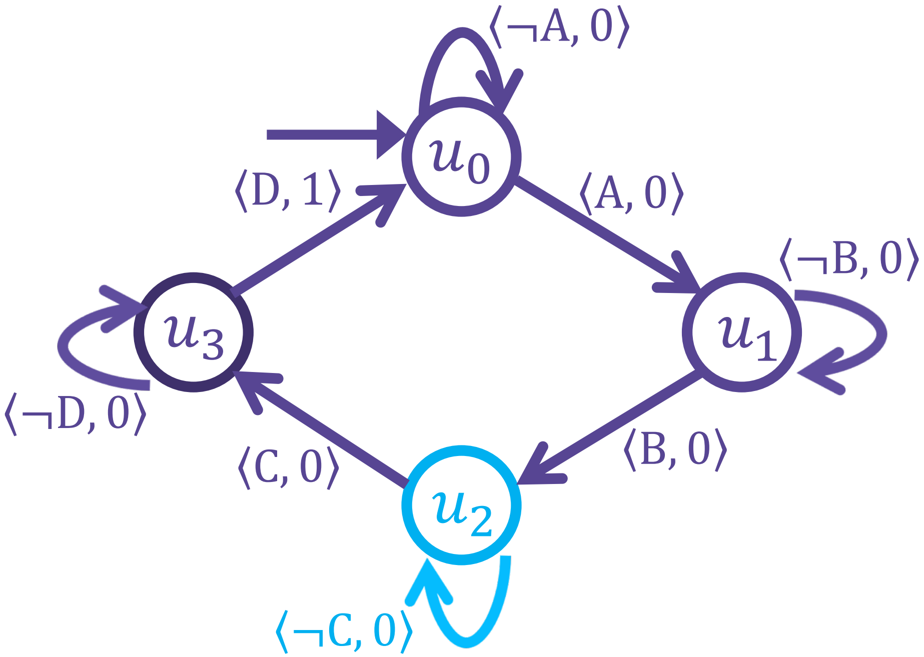
Reward Machines in Action



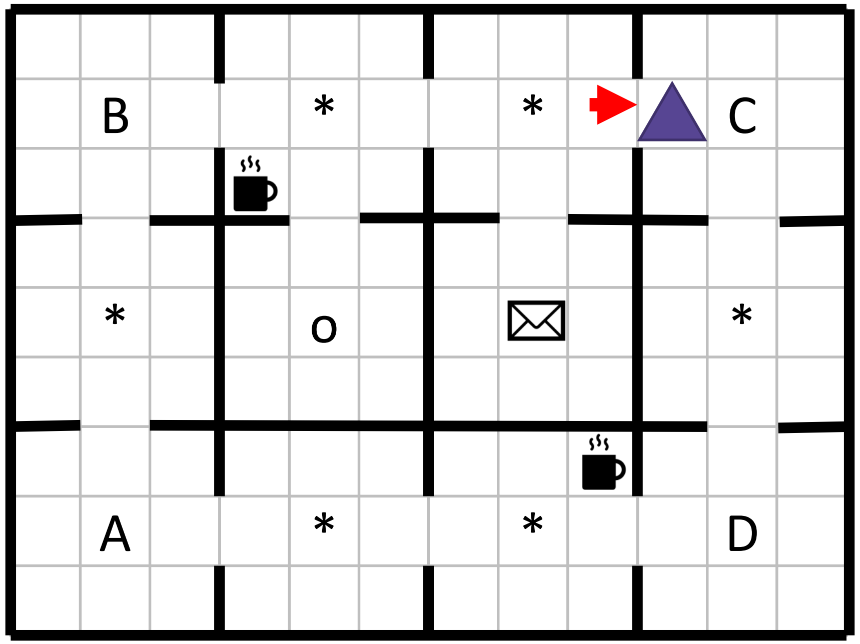


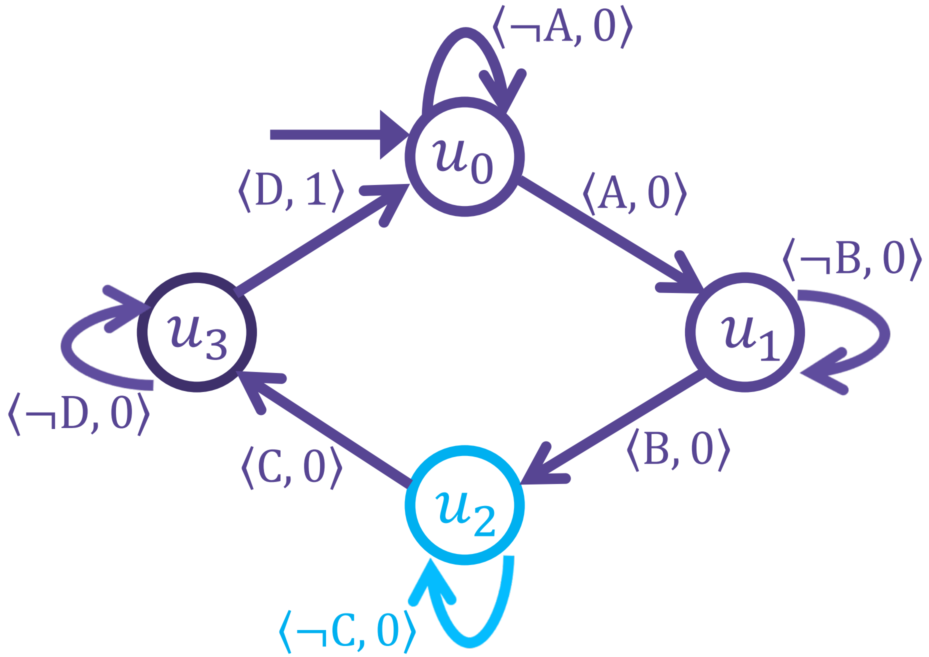
Reward Machines in Action



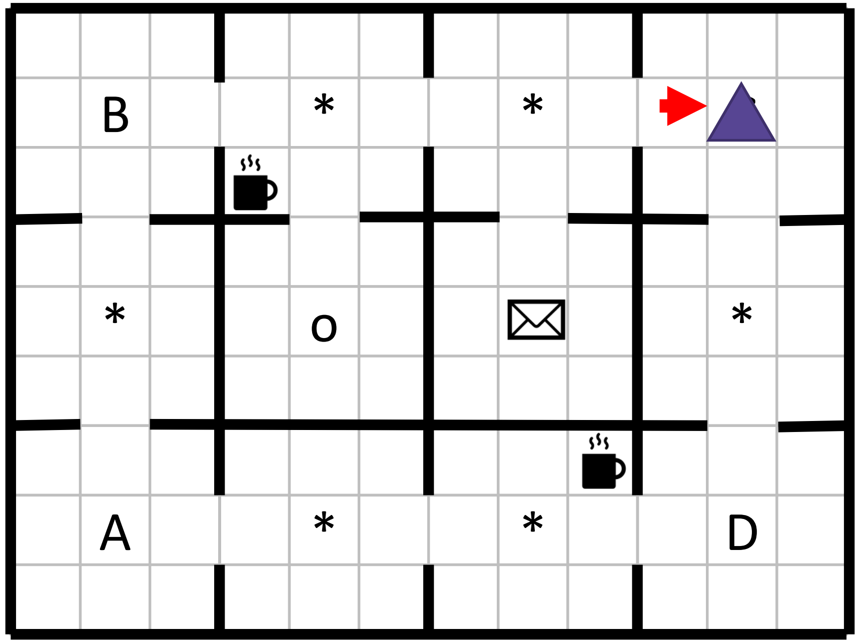


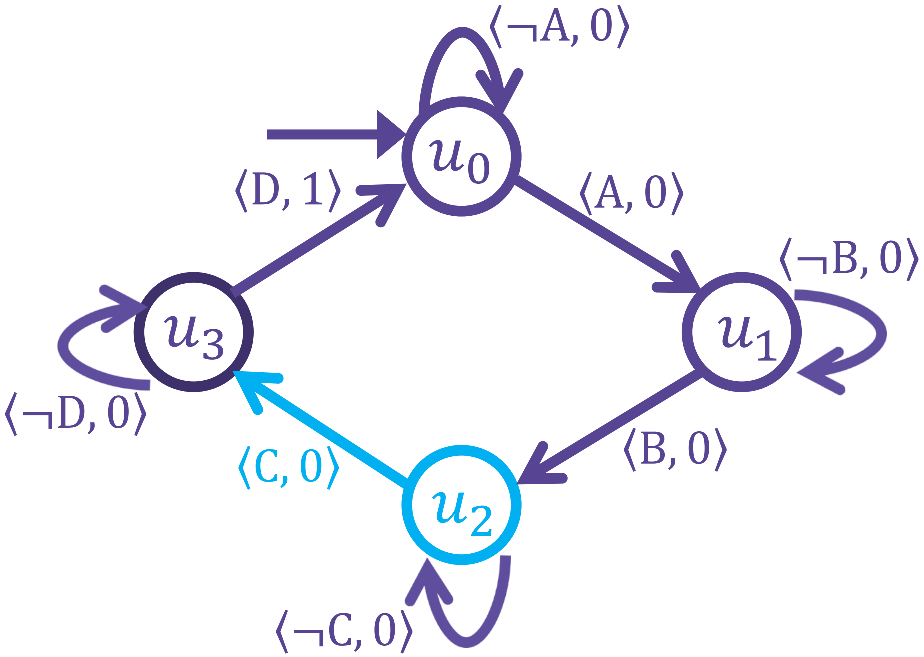
Reward Machines in Action



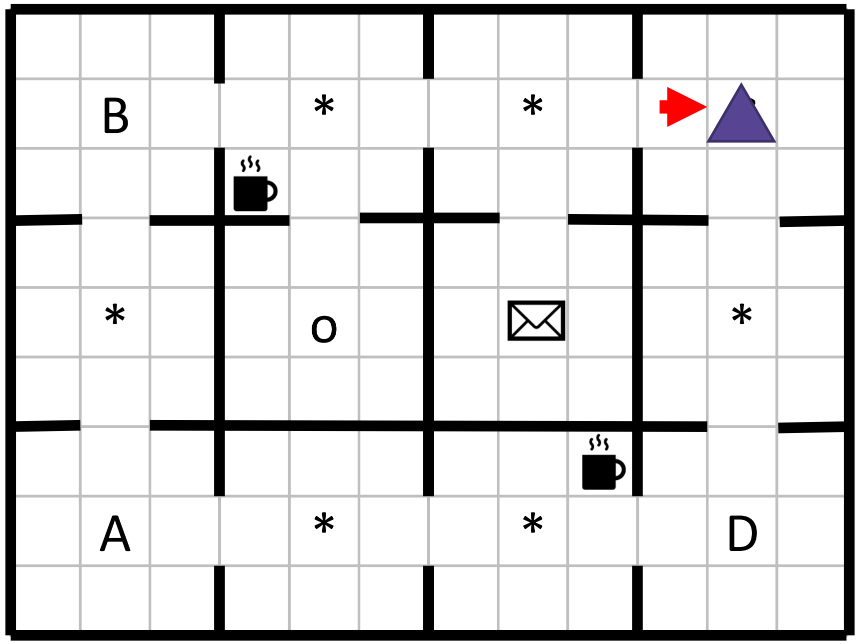


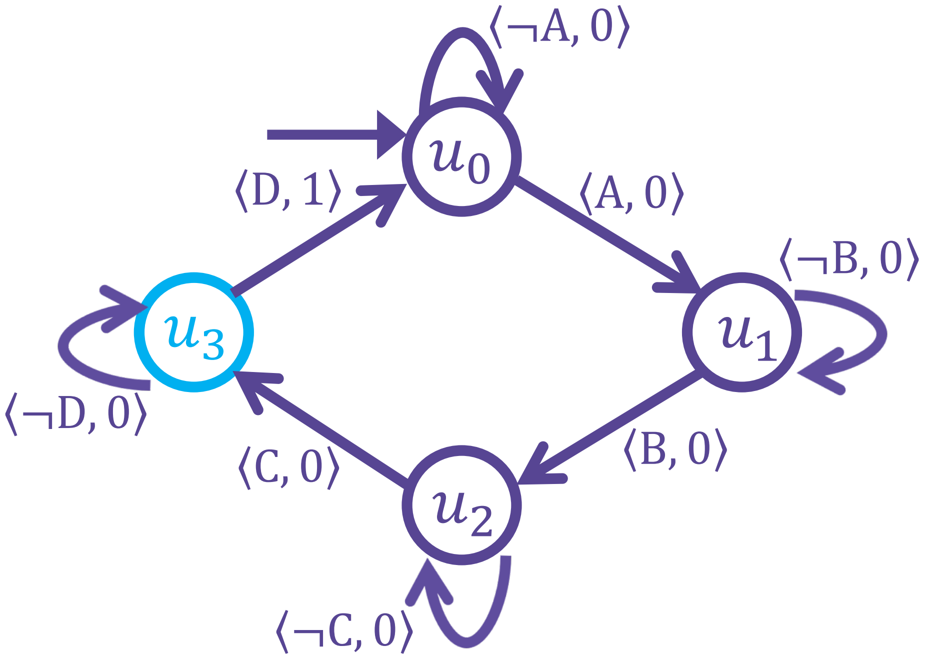
Reward Machines in Action



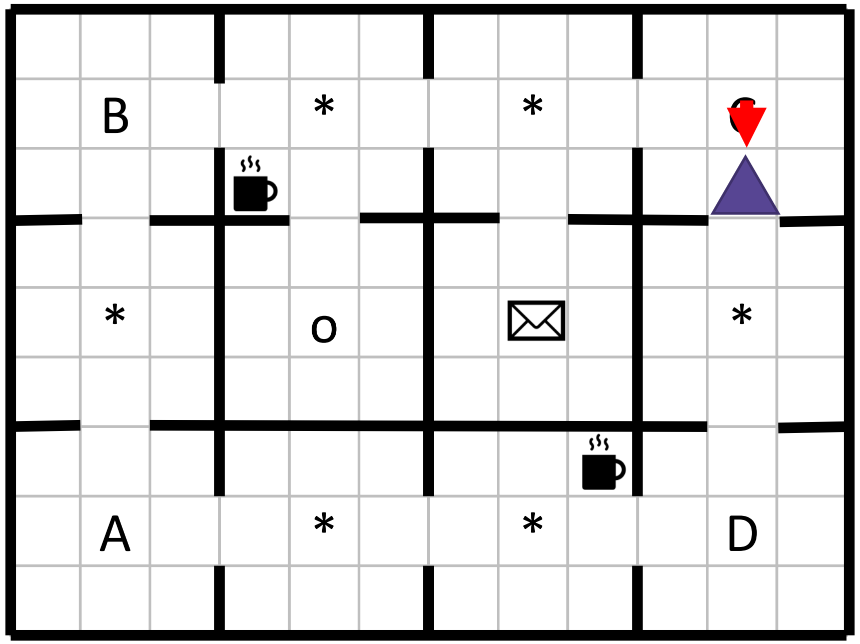


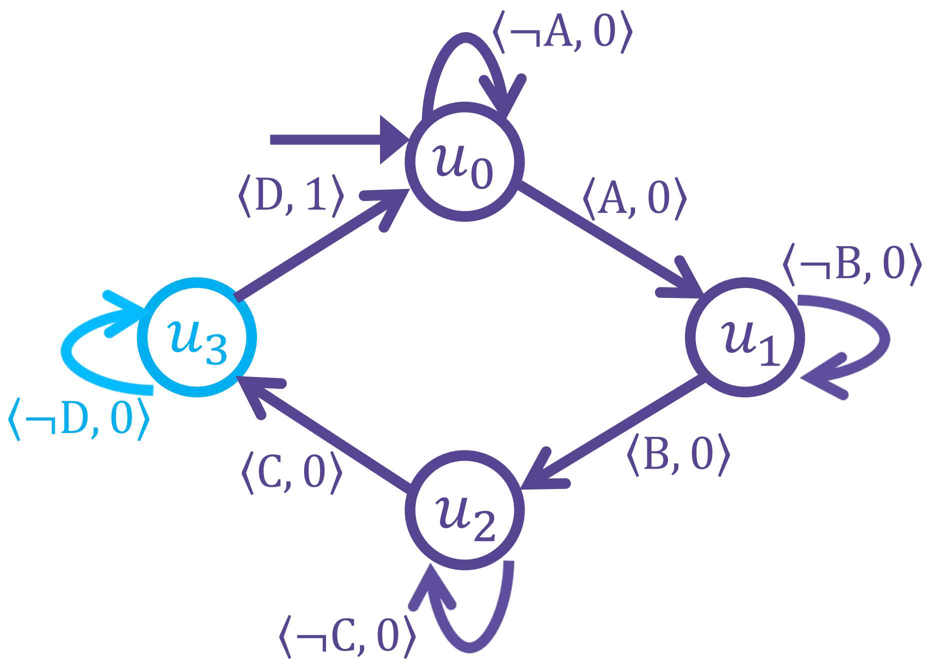
Reward Machines in Action



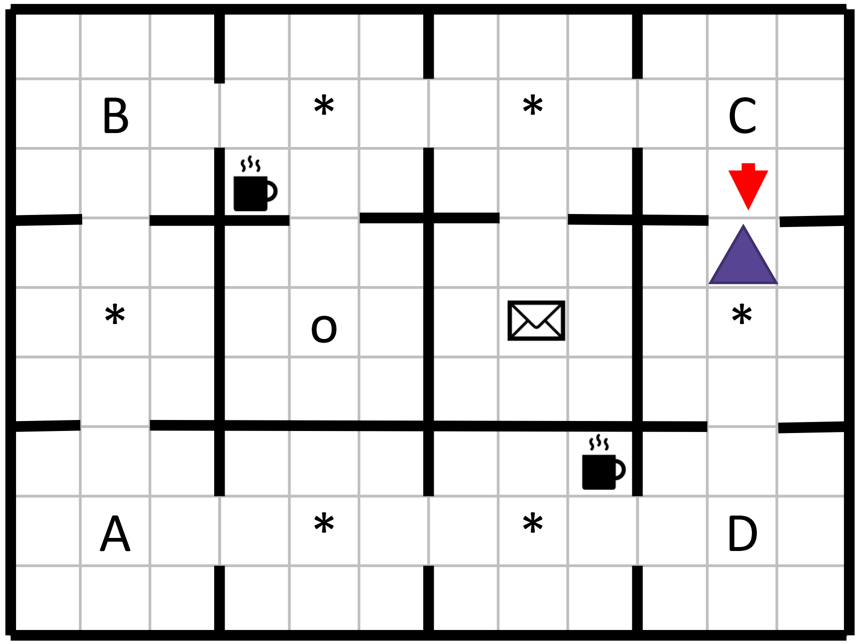


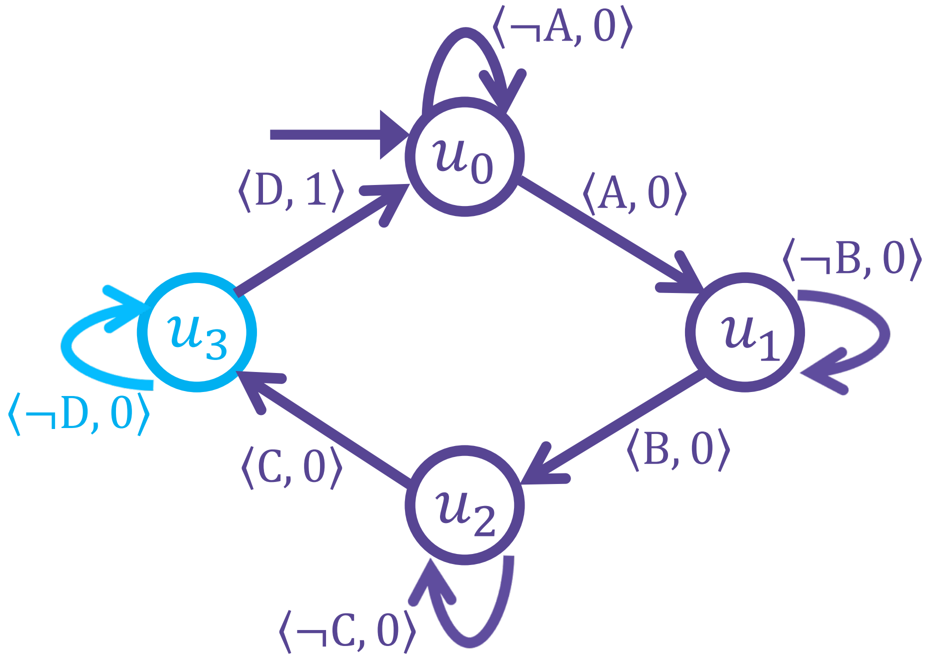
Reward Machines in Action



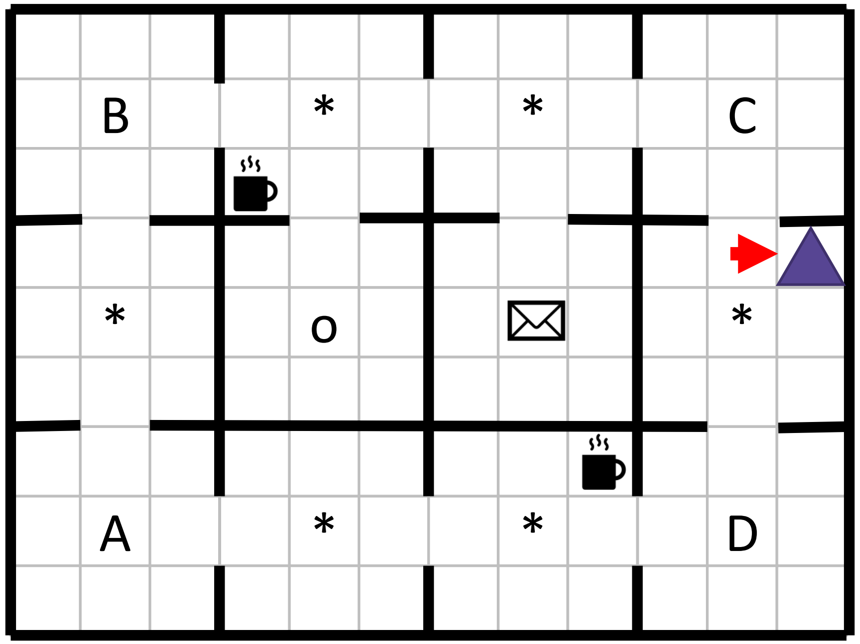


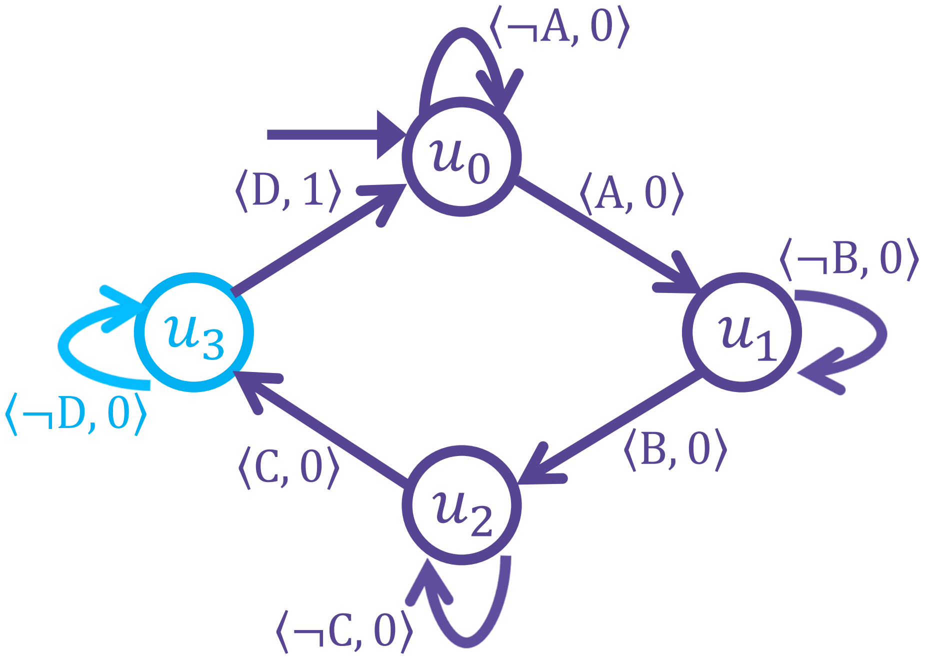
Reward Machines in Action



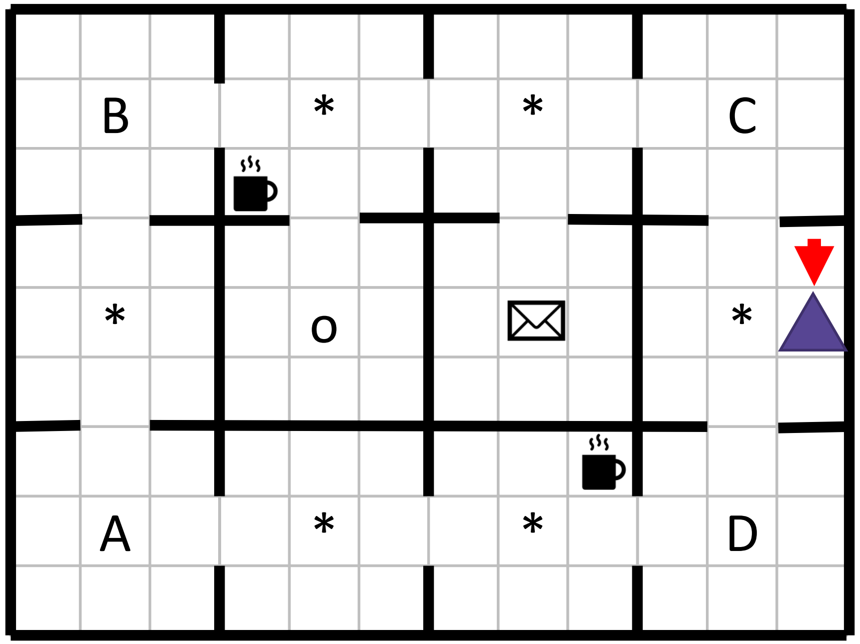


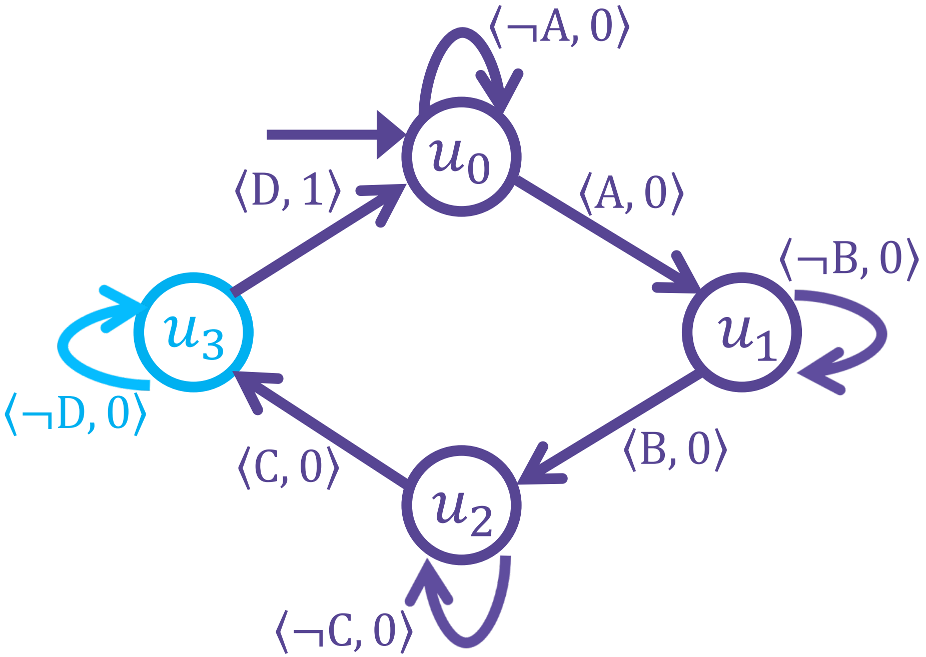
Reward Machines in Action



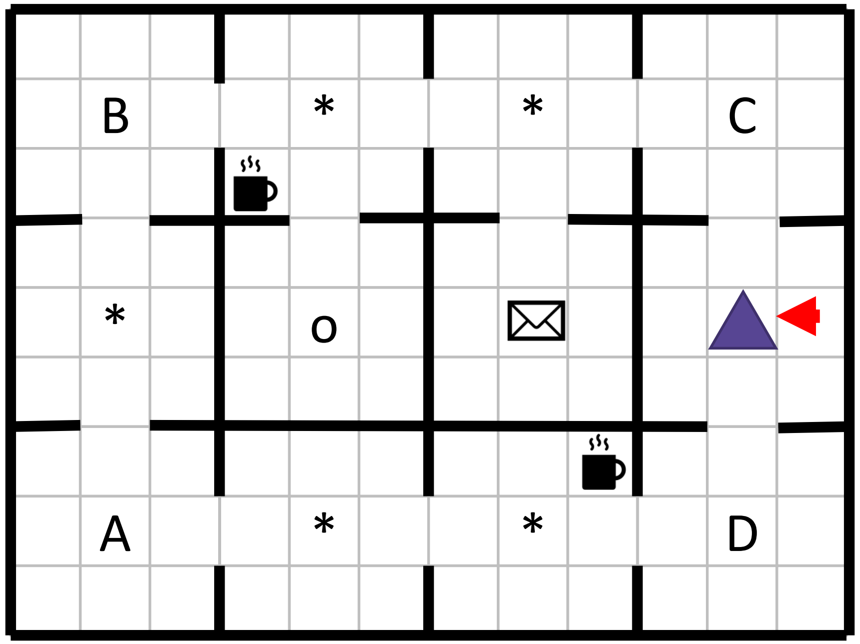


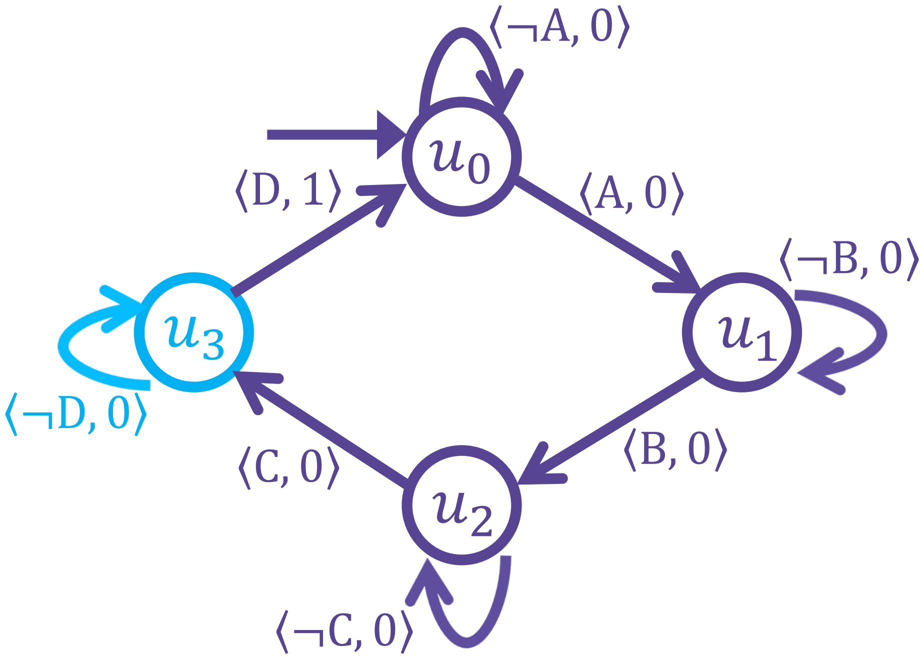
Reward Machines in Action



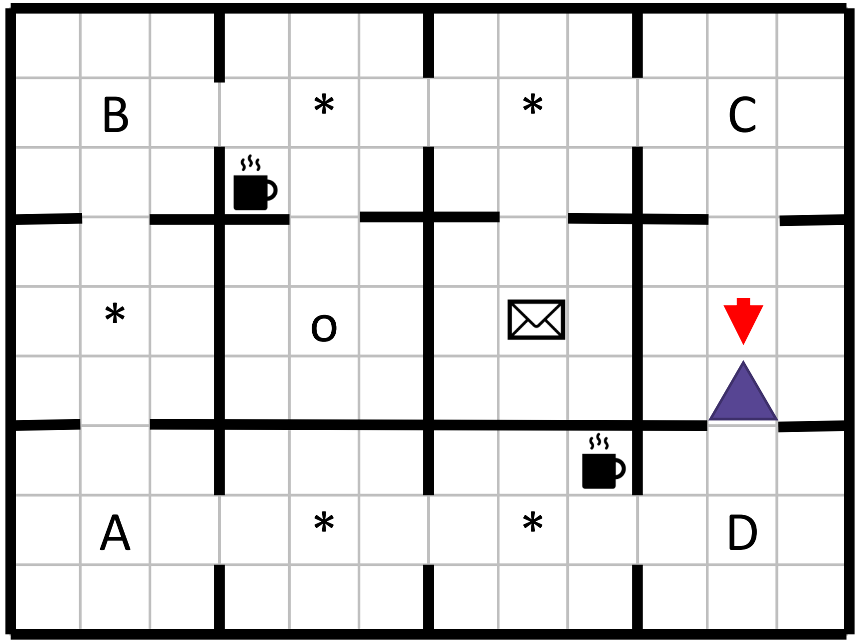


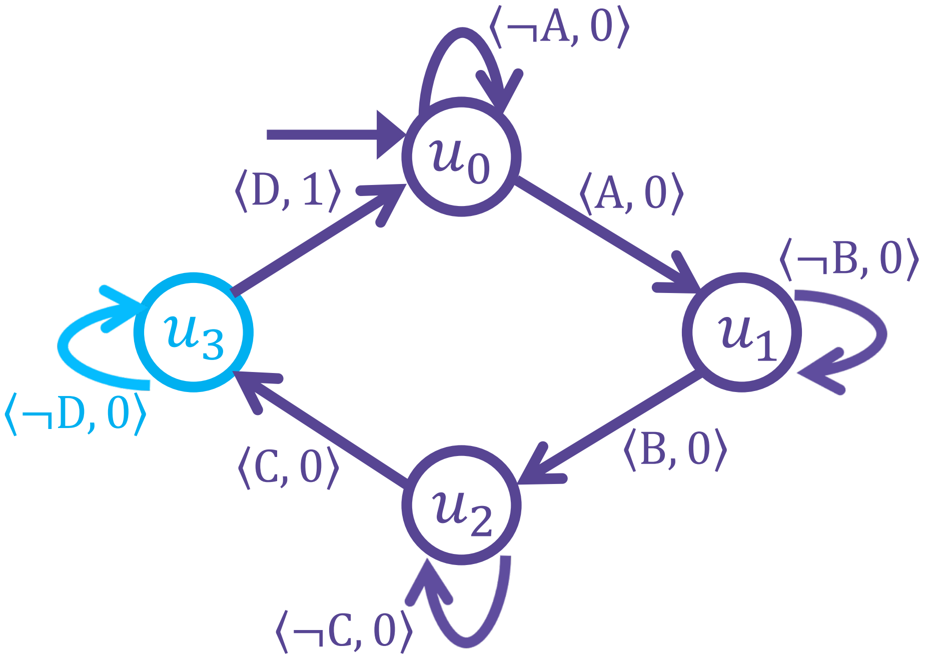
Reward Machines in Action



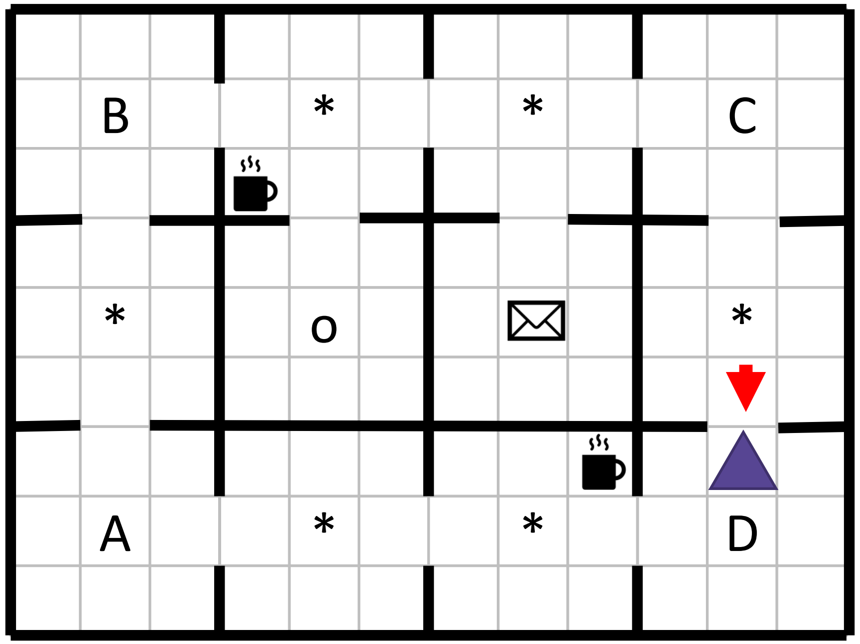


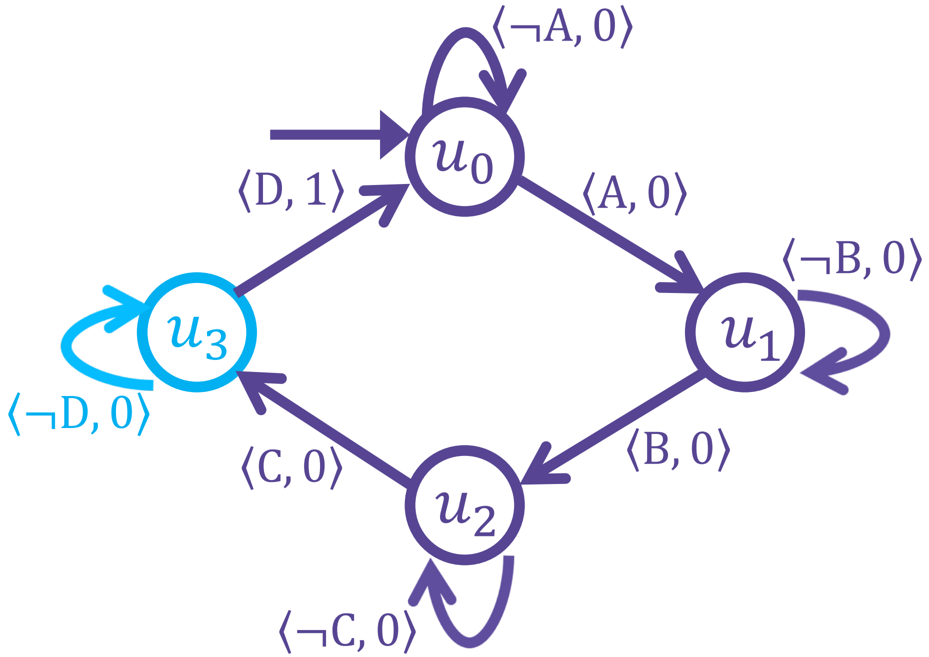
Reward Machines in Action



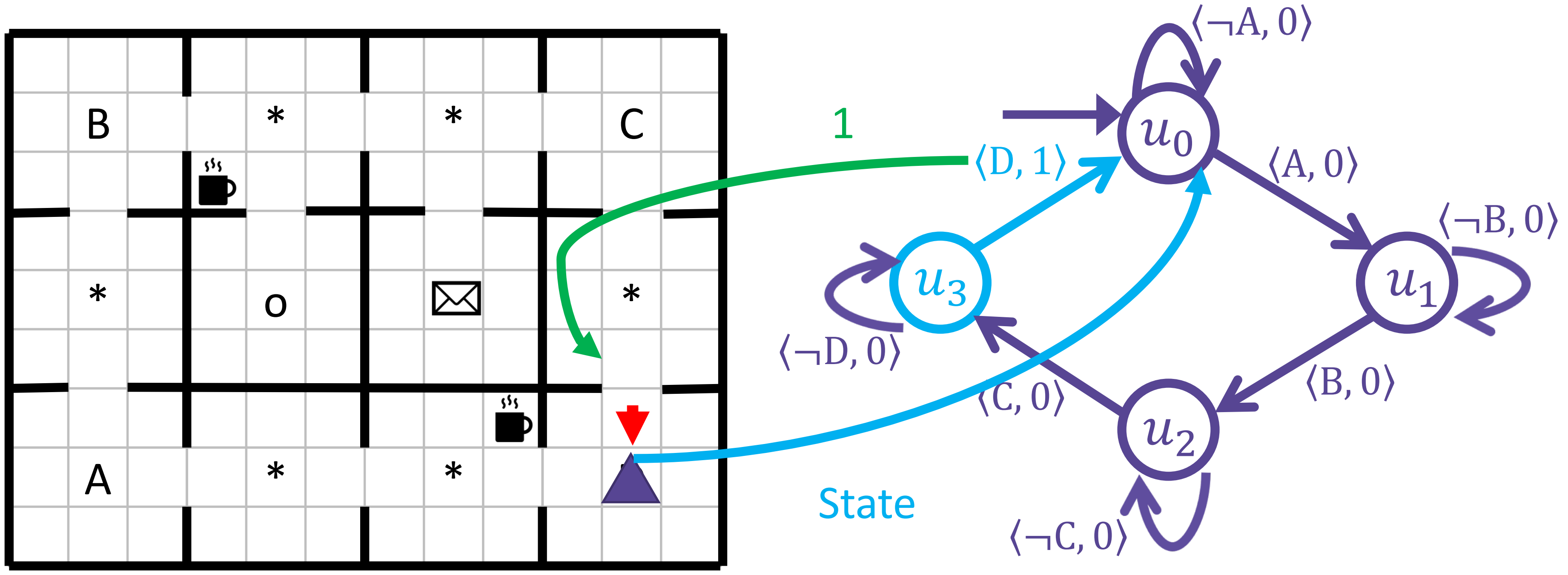


Reward Machines in Action

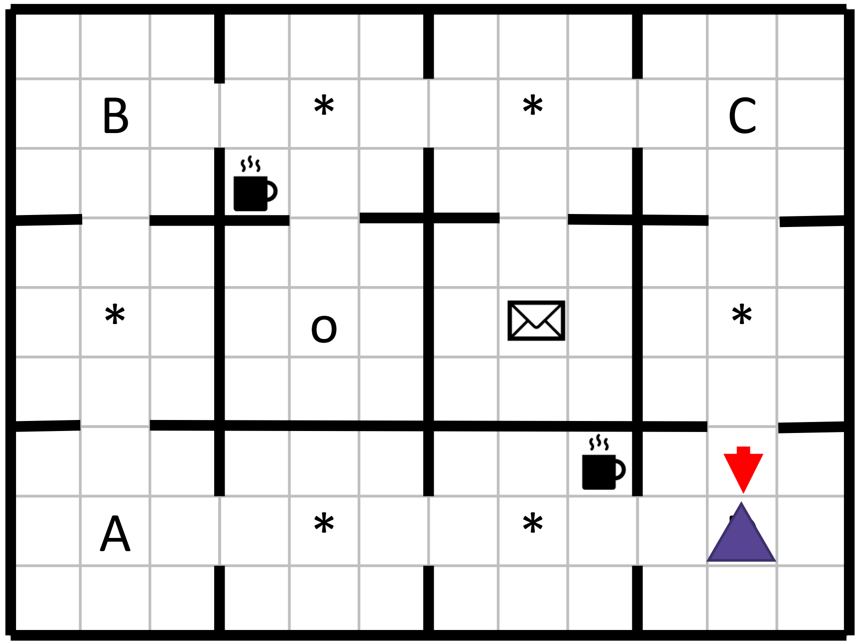


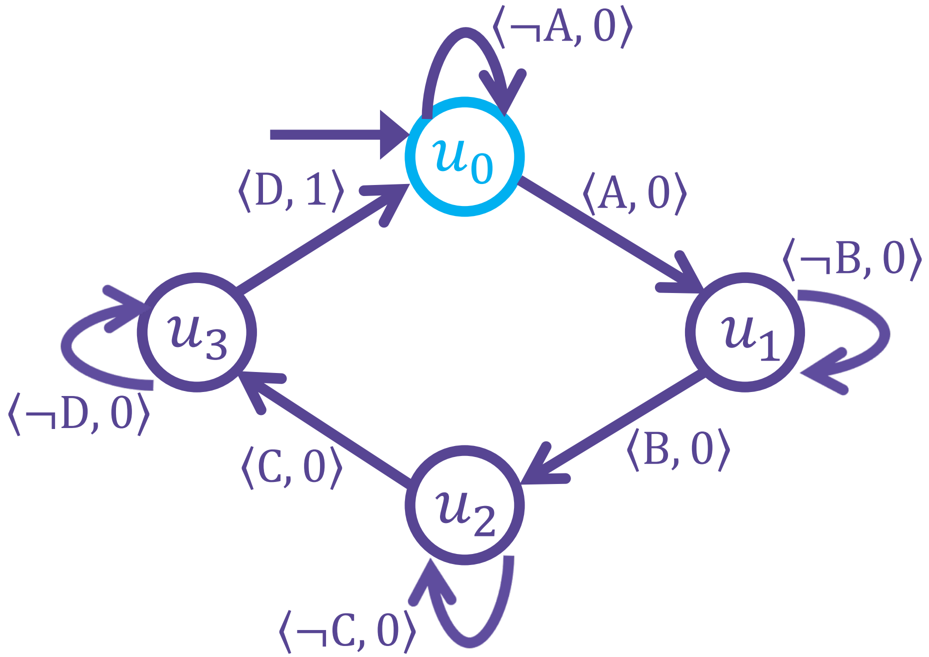


Reward Machines in Action



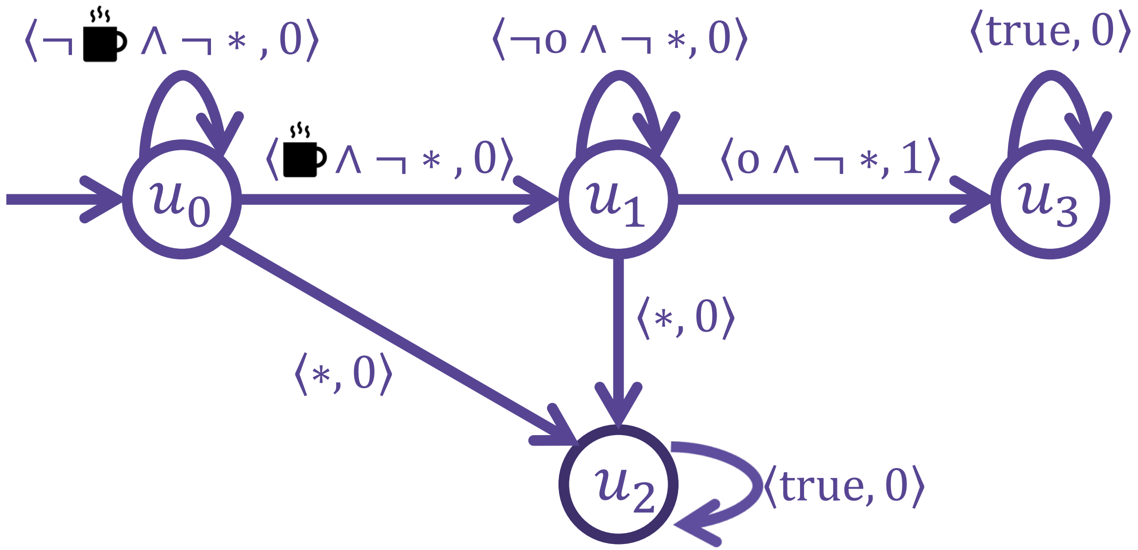
Reward Machines in Action





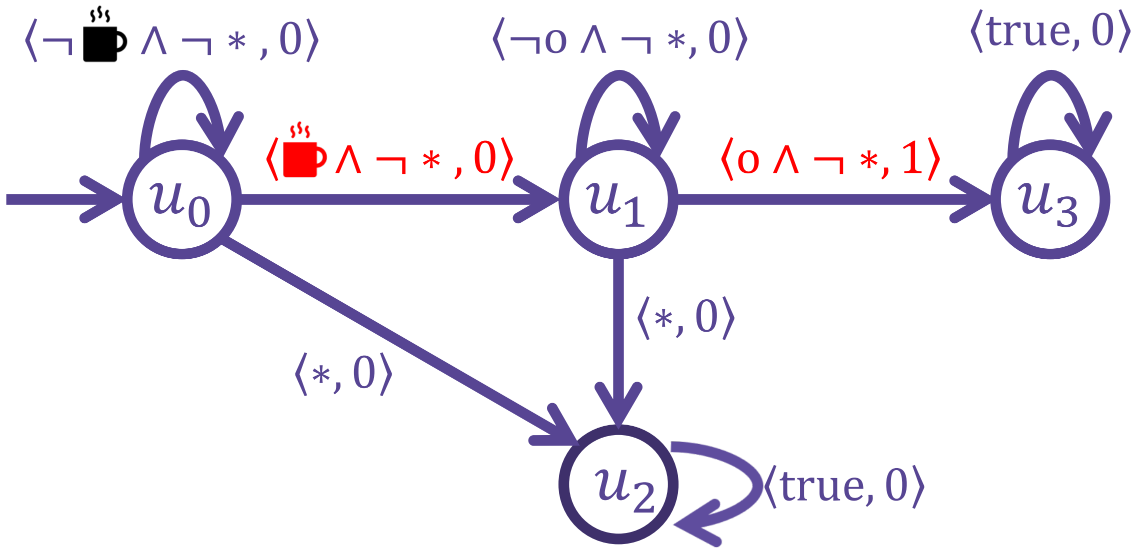
Other Reward Machines

Task: Deliver coffee to the office, while avoiding furniture.



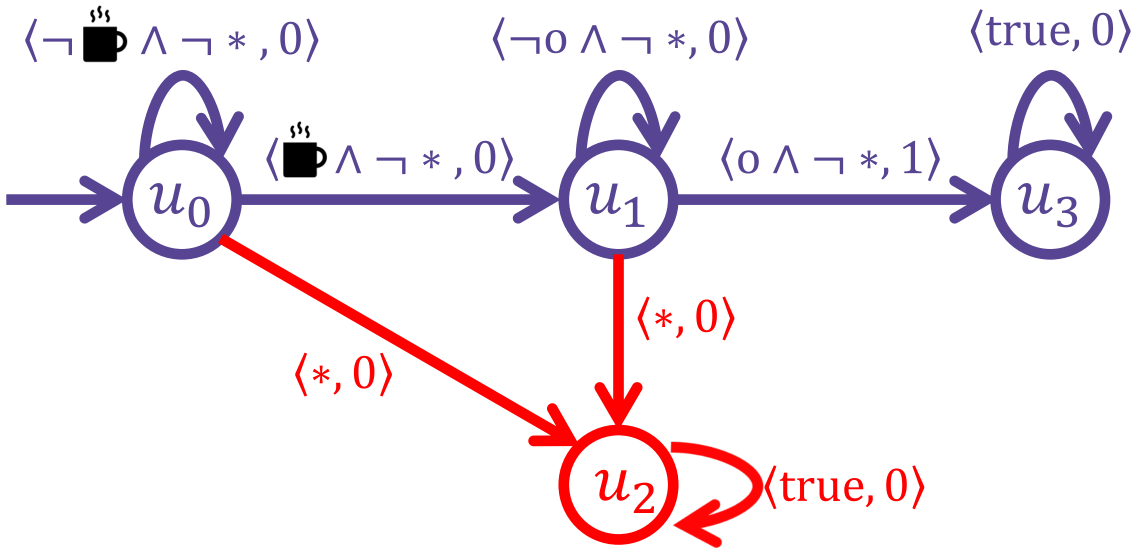
Other Reward Machines

Task: Deliver coffee to the office, while avoiding furniture.



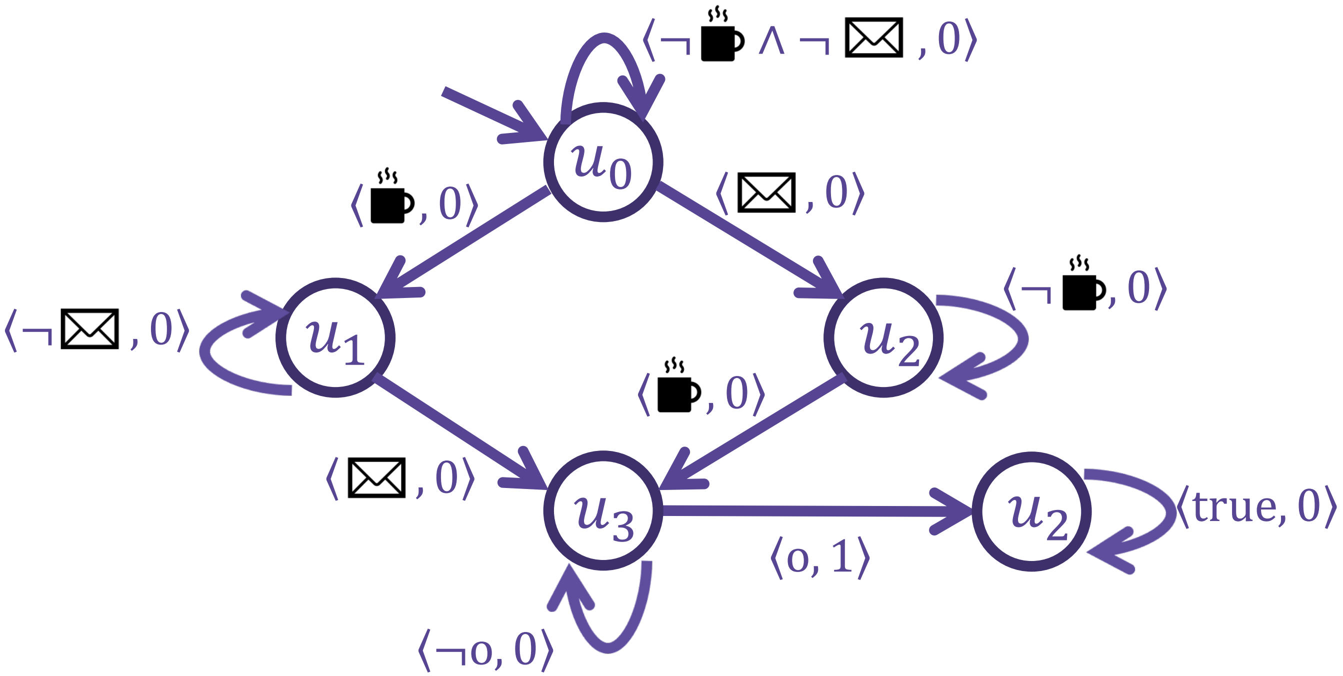
Other Reward Machines

Task: Deliver coffee to the office, while avoiding furniture.



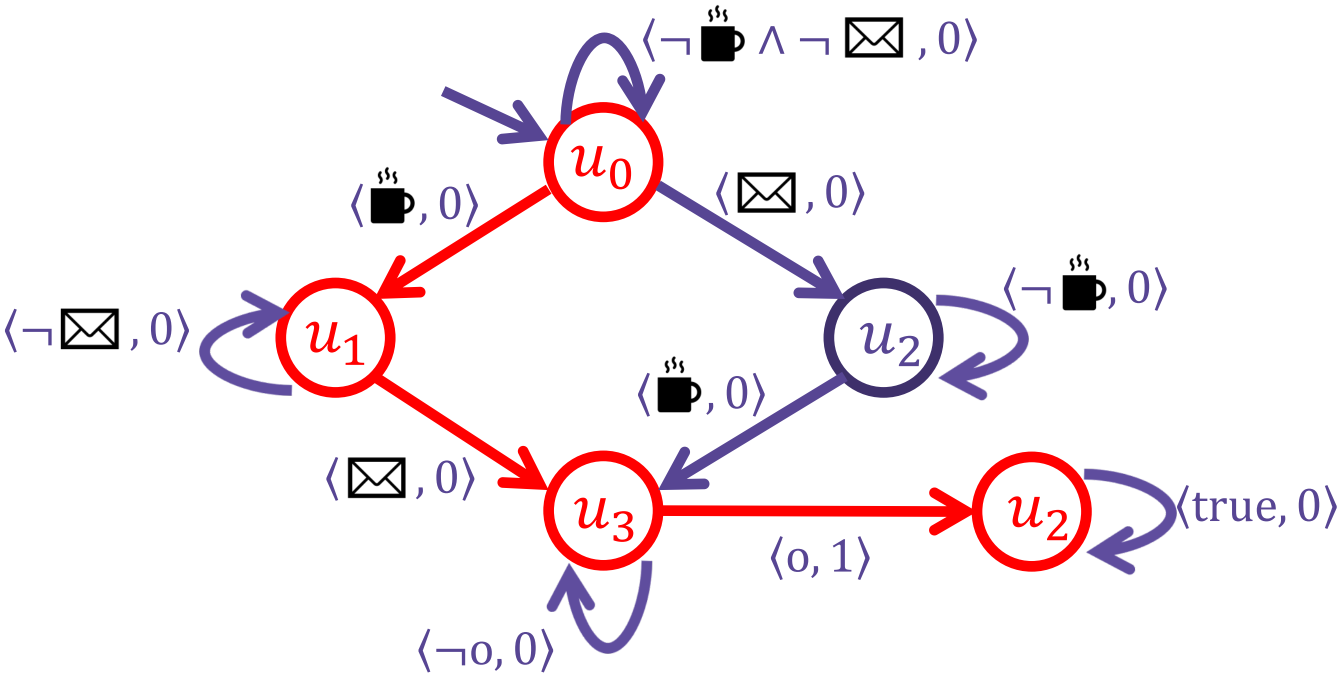
Other Reward Machines

Task: Deliver coffee and mail to the office.



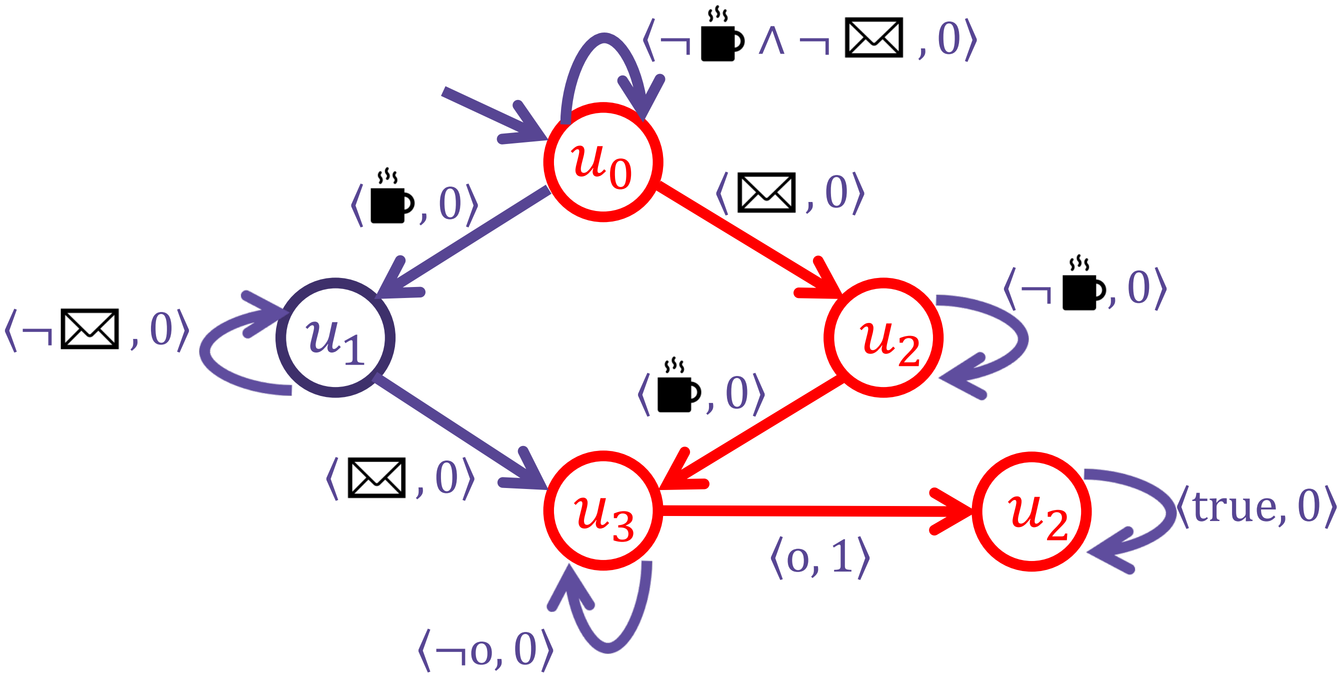
Other Reward Machines

Task: Deliver coffee and mail to the office.



Other Reward Machines

Task: Deliver coffee and mail to the office.



The Rest of the Talk

- Reward Machines (RM)

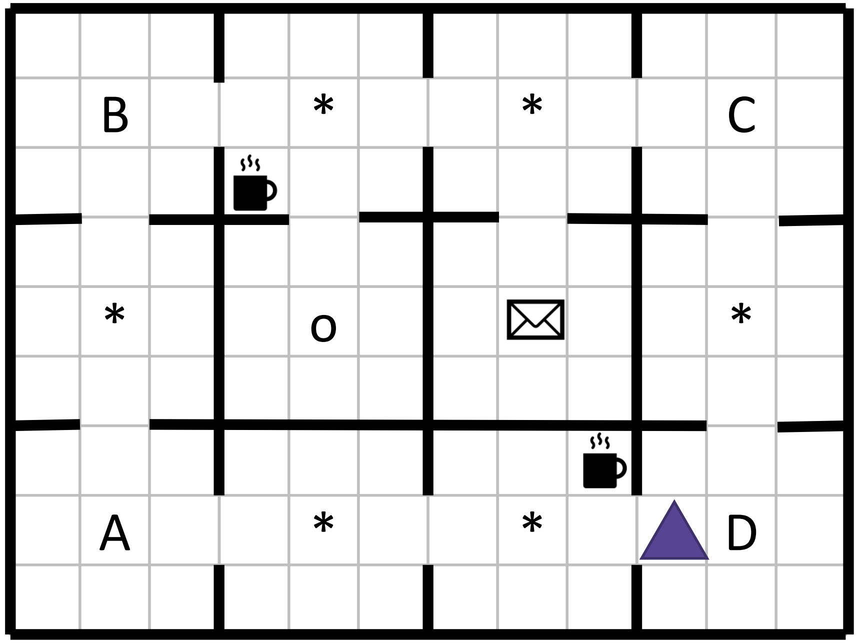
▶ Exploiting RM Structure in Learning

- Experiments
- Creating Reward Machines
- Recap

EXPLOITING RM STRUCTURE IN LEARNING

A simple idea …

Someone has to program the reward function

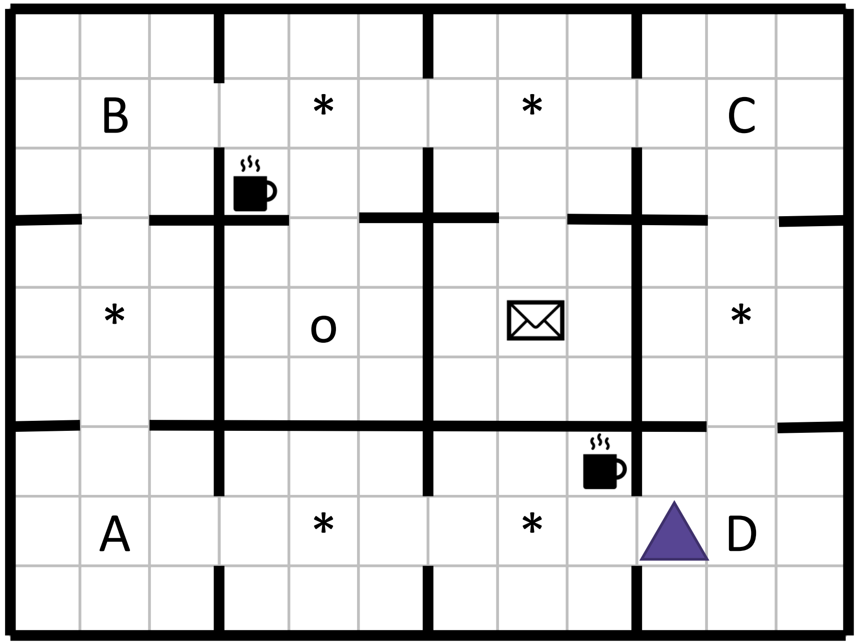



```
count = 0  # global variable

def get_reward(s):
    if count == 0 and state.at("A"):
        count = 1
    if count == 1 and state.at("B"):
        count = 2
    if count == 2 and state.at("C"):
        count = 3
    if count == 3 and state.at("D"):
        count = 0
        return 1
    return 0
```

Task: Visit A, B, C, and D, in order.

… even when the environment is the real world!

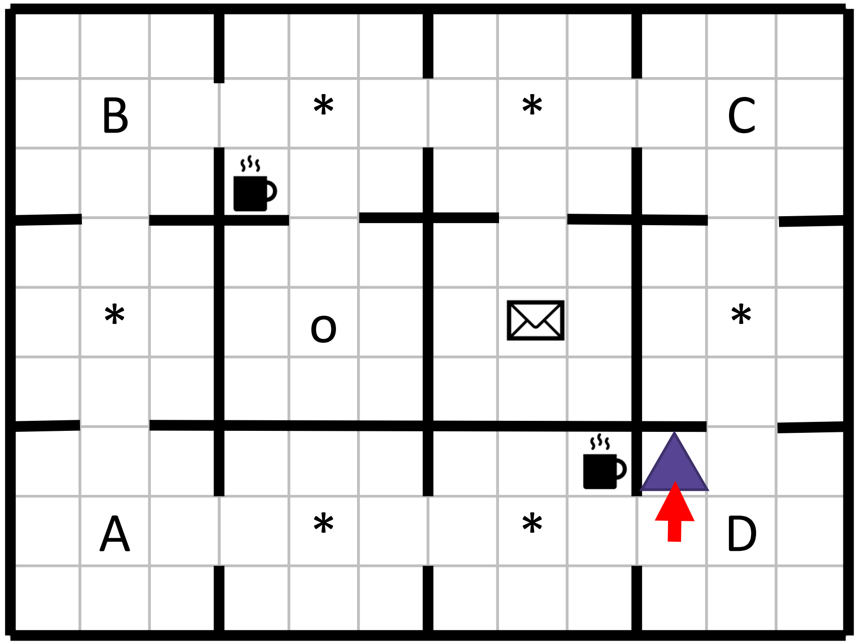
But the Reward Function is a Black Box



Reward Function
(as part of environment)

Task: Visit A, B, C, and D, in order.

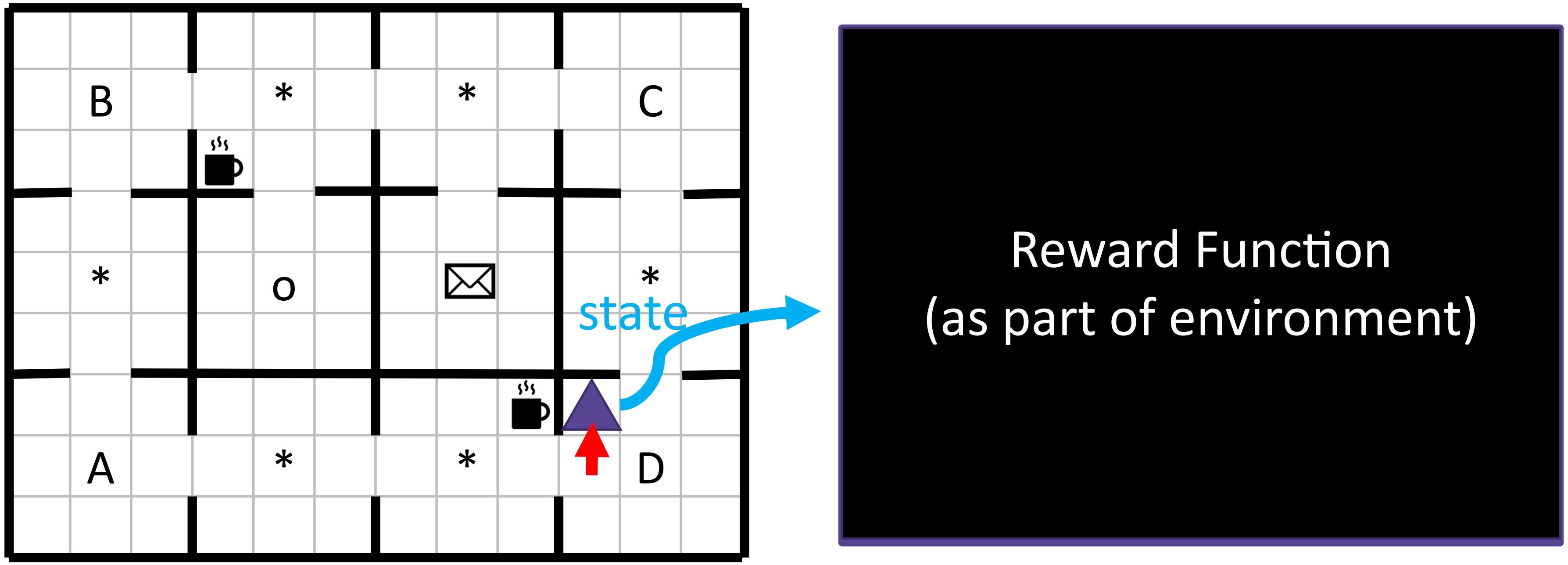
But the Reward Function is a Black Box



Reward Function
(as part of environment)

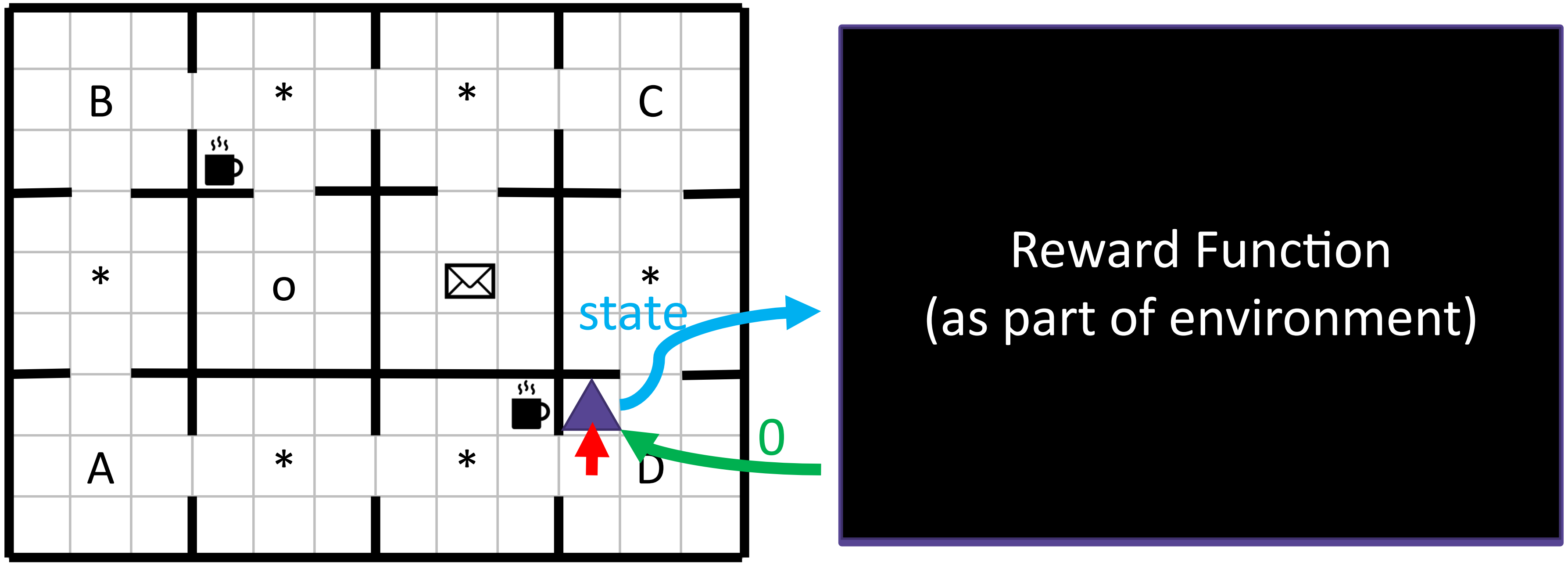
Task: Visit A, B, C, and D, in order.

But the Reward Function is a Black Box



Task: Visit A, B, C, and D, in order.

But the Reward Function is a Black Box



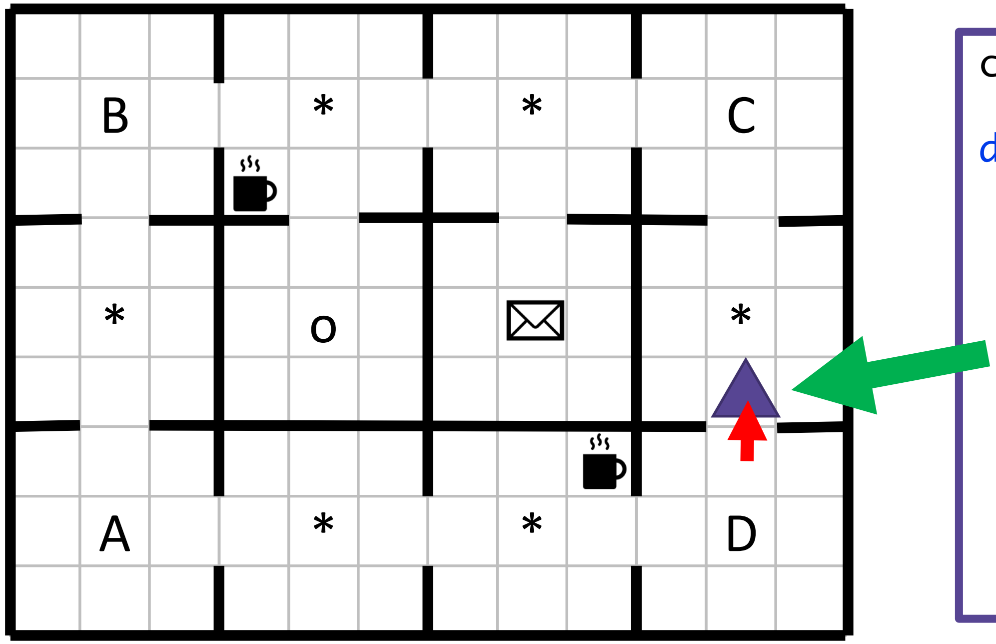
Task: Visit A, B, C, and D, in order.

Remember this!

Simple Idea:

- Give the agent access to the reward function
- Exploit reward function structure in learning

Running Example




```
count = 0  # global variable

def get_reward(s):
    if count == 0 and state.at("A"):
        count = 1
    if count == 1 and state.at("B"):
        count = 2
    if count == 2 and state.at("C"):
        count = 3
    if count == 3 and state.at("D"):
        count = 0
        return 1
    return 0
```

The agent can exploit structure in the reward function.

Methods for Exploiting RM Structure

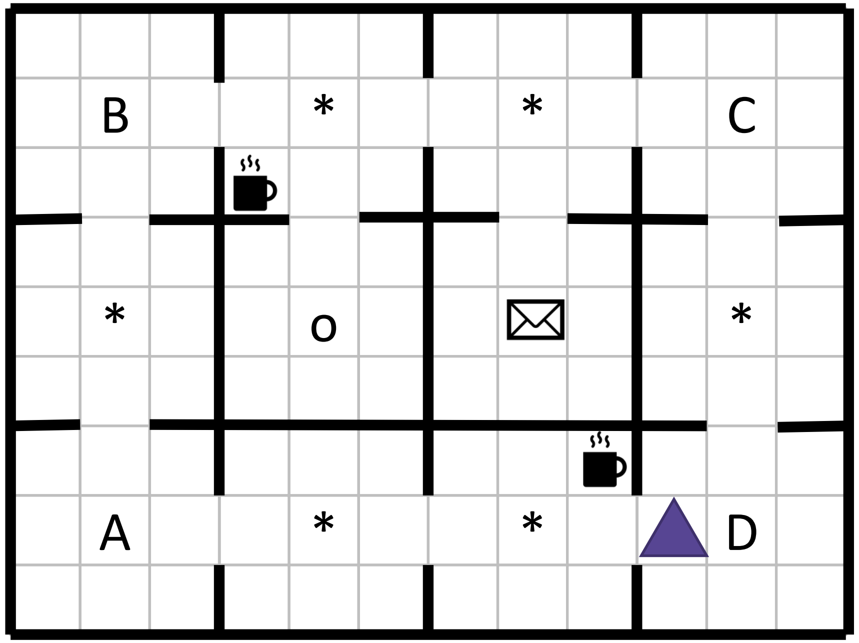
Baselines based on existing methods:

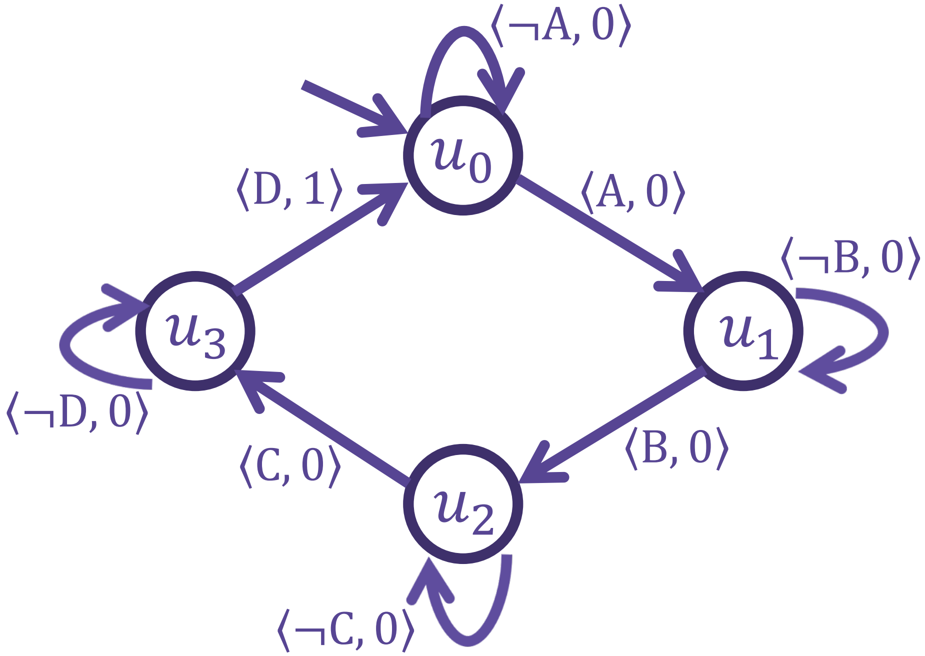
1. Q-learning over an equivalent MDP (Q-learning)
2. Hierarchical RL based on options (HRL)
3. HRL with RM-based pruning (HRL-RM)

Our approaches:

4. Q-learning for Reward Machines (QRM)
5. QRM + Reward Shaping for Reward Machine (QRM + RS)

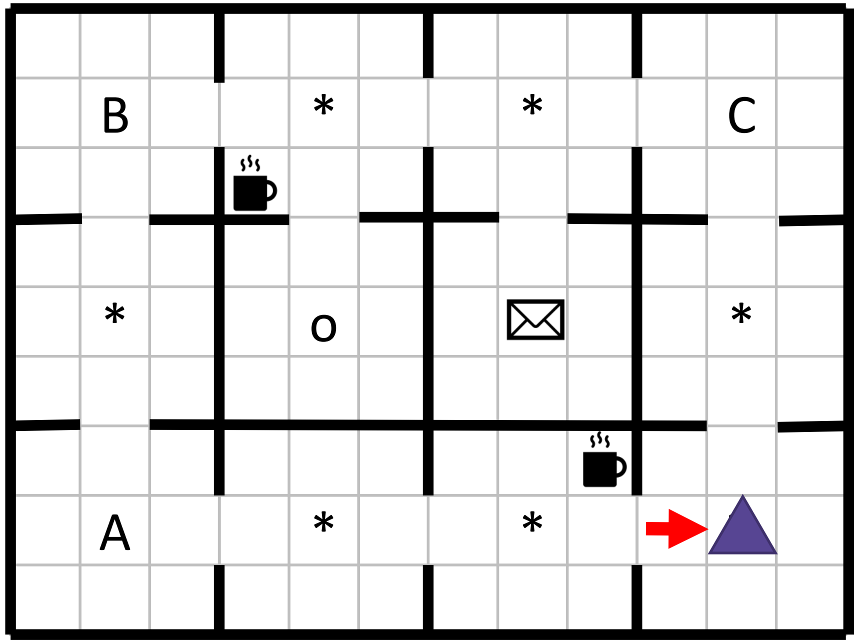
1. Q-Learning Baseline

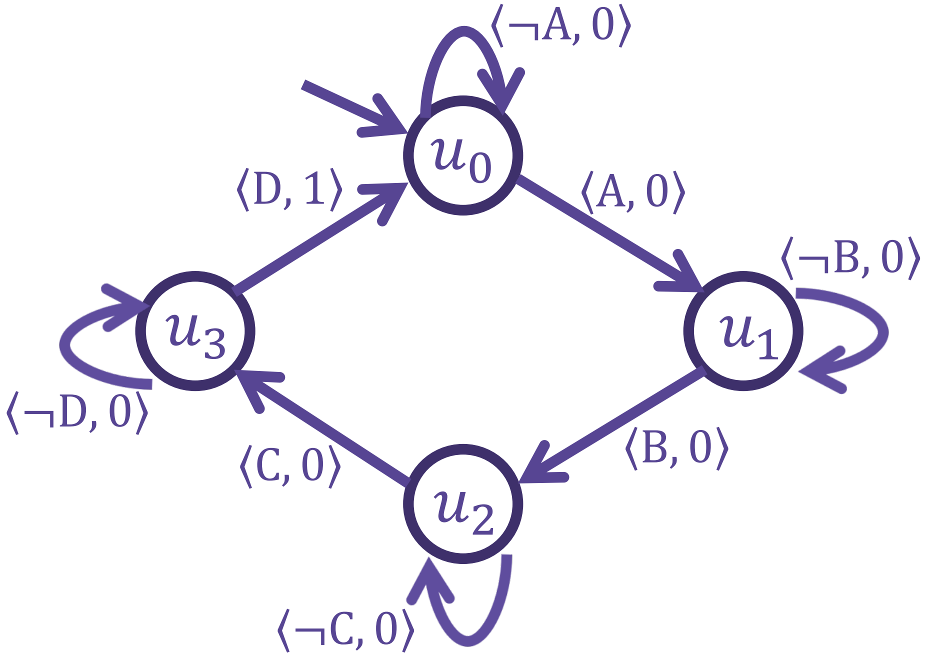




A Reward Machine may define a non-Markovian reward function.

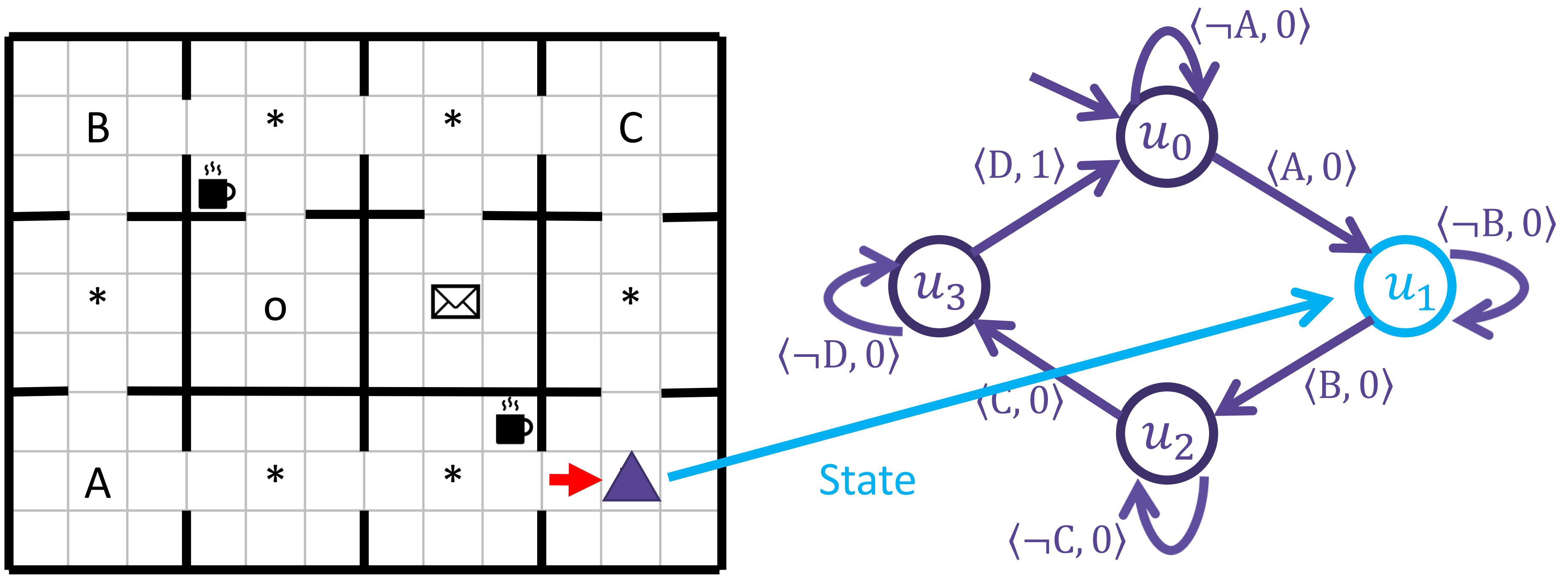
1. Q-Learning Baseline





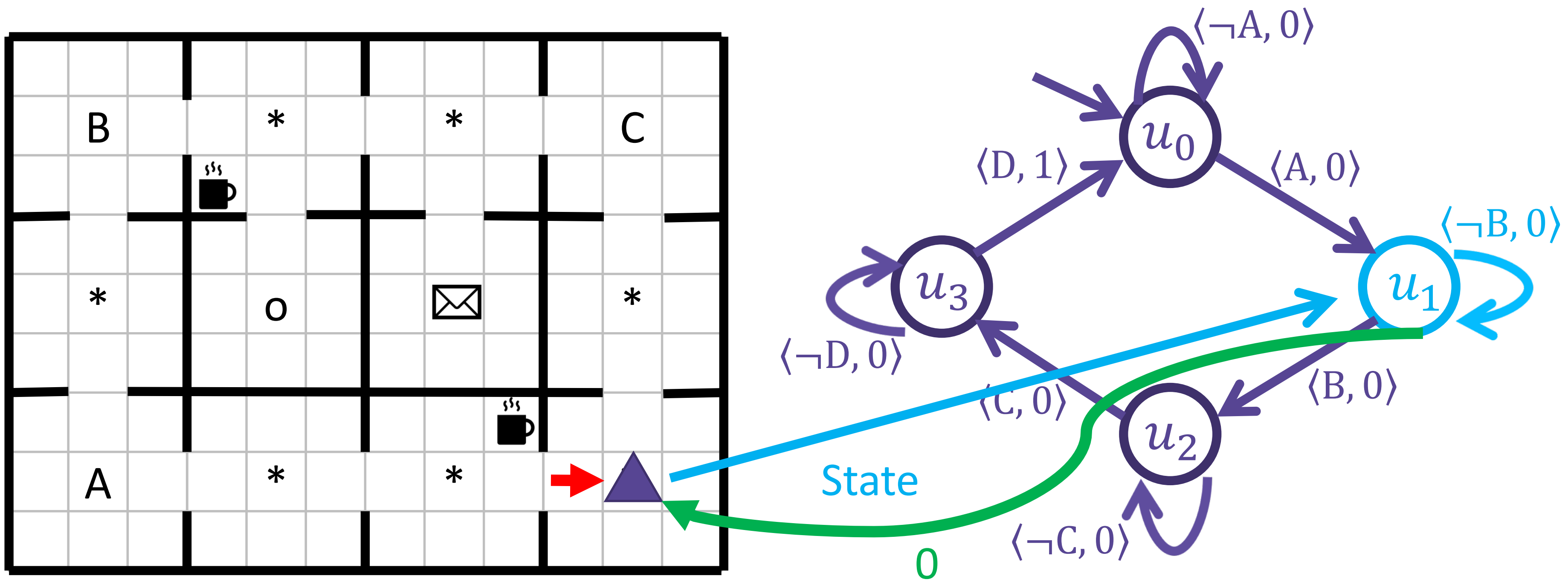
A Reward Machine may define a non-Markovian reward function.

1. Q-Learning Baseline



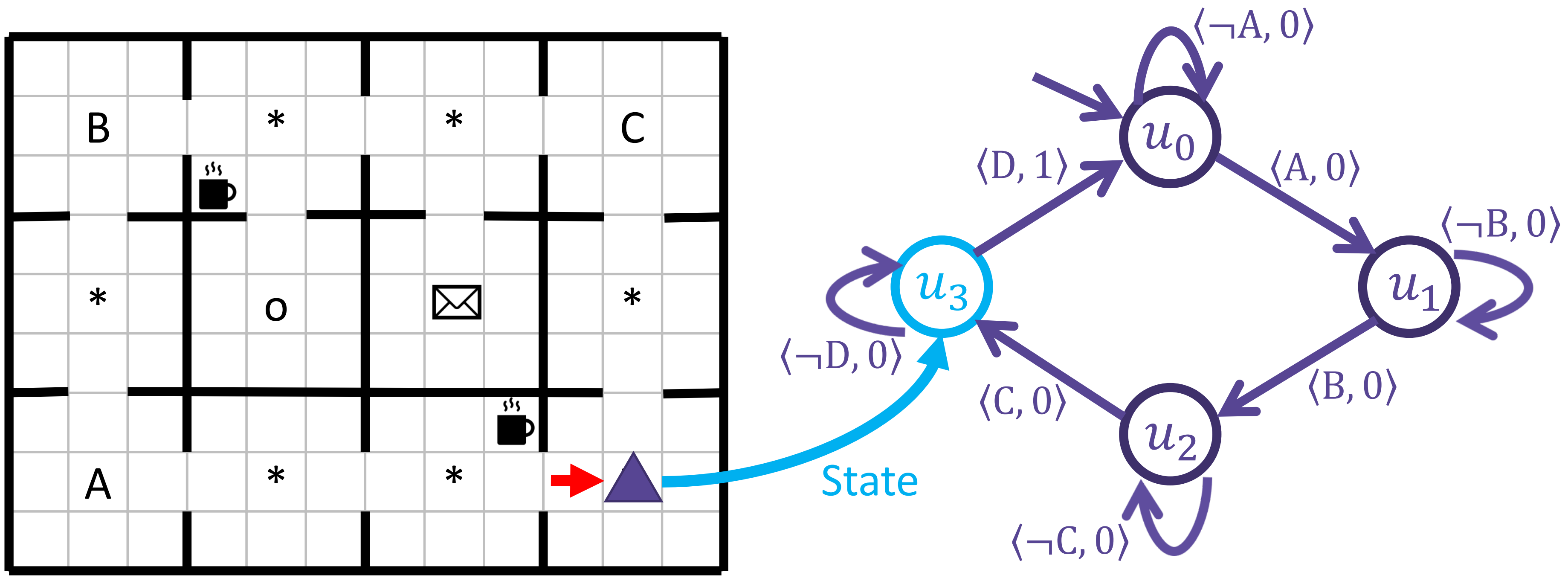
A Reward Machine may define a non-Markovian reward function.

1. Q-Learning Baseline



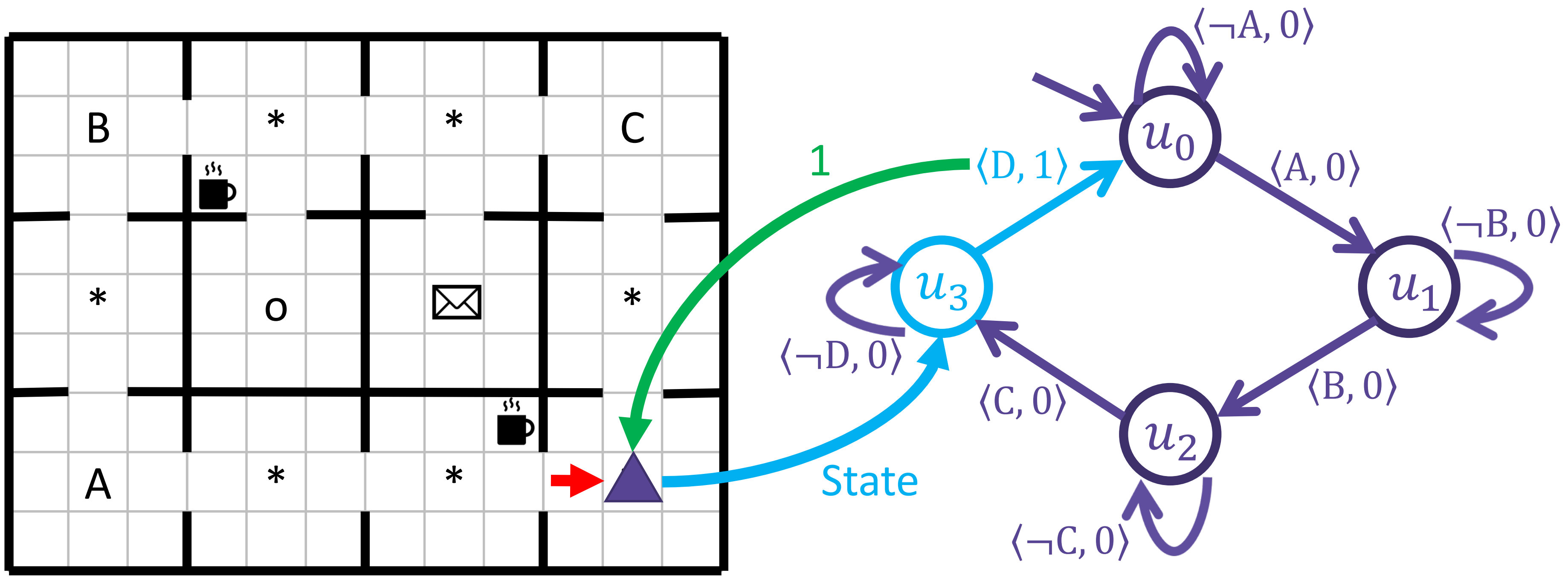
A Reward Machine may define a non-Markovian reward function.

1. Q-Learning Baseline



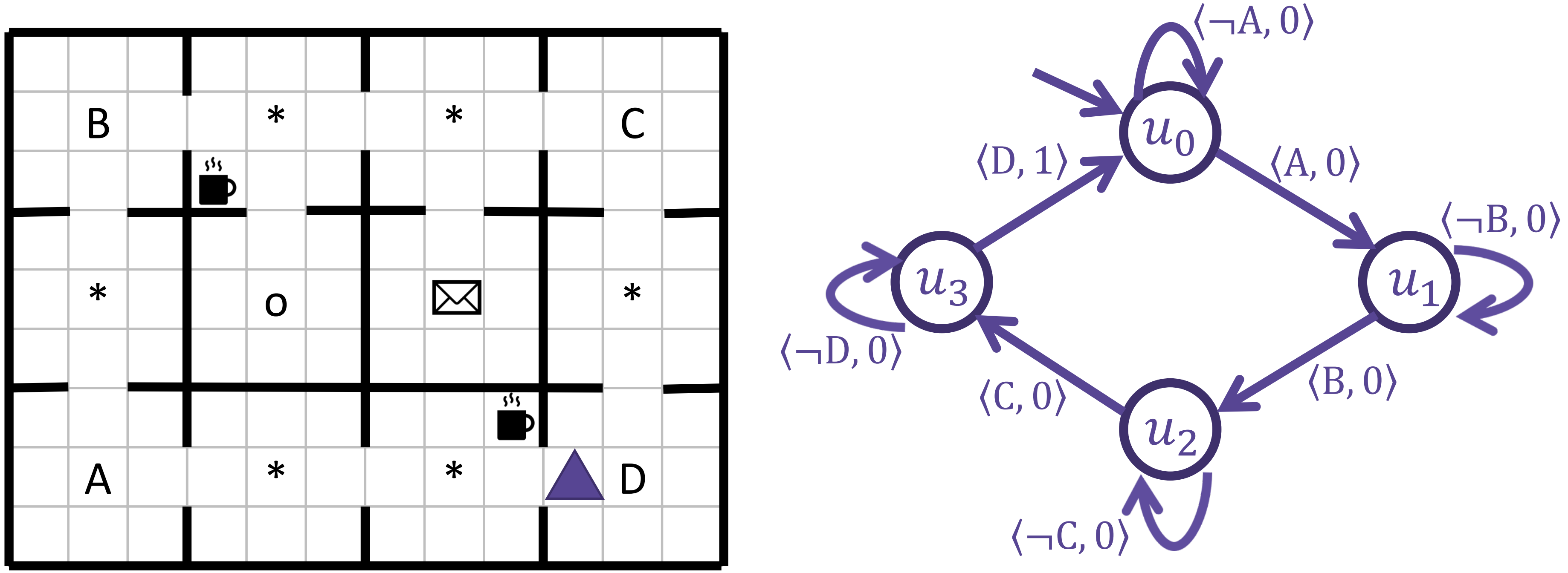
A Reward Machine may define a non-Markovian reward function.

1. Q-Learning Baseline



A Reward Machine may define a non-Markovian reward function.

1. Q-Learning Baseline

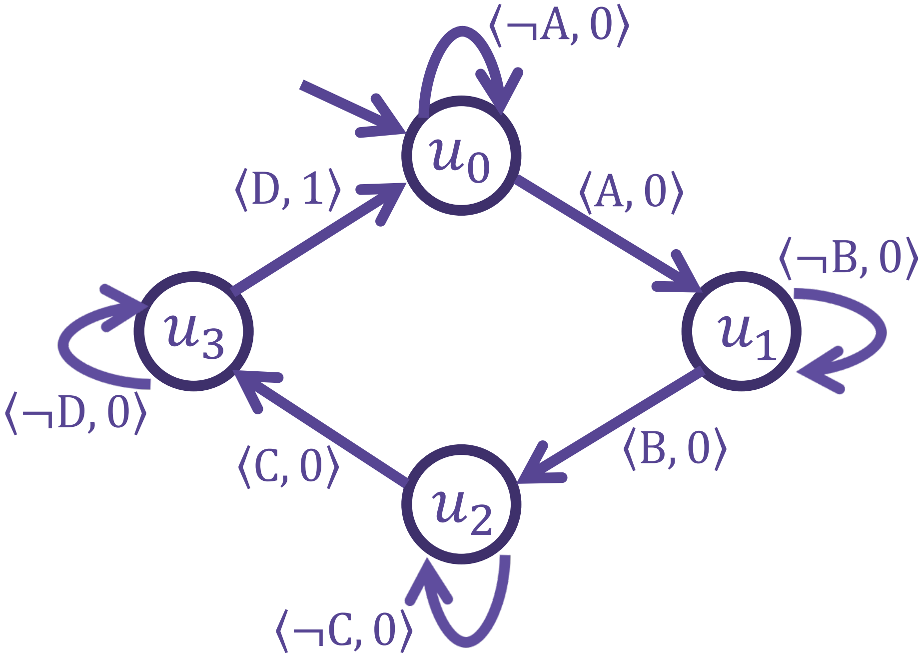


Solution: Include RM state as part of agent's state representation.

Use standard Q-learning on resulting MDP.

2. Option-Based Hierarchical RL (HRL)

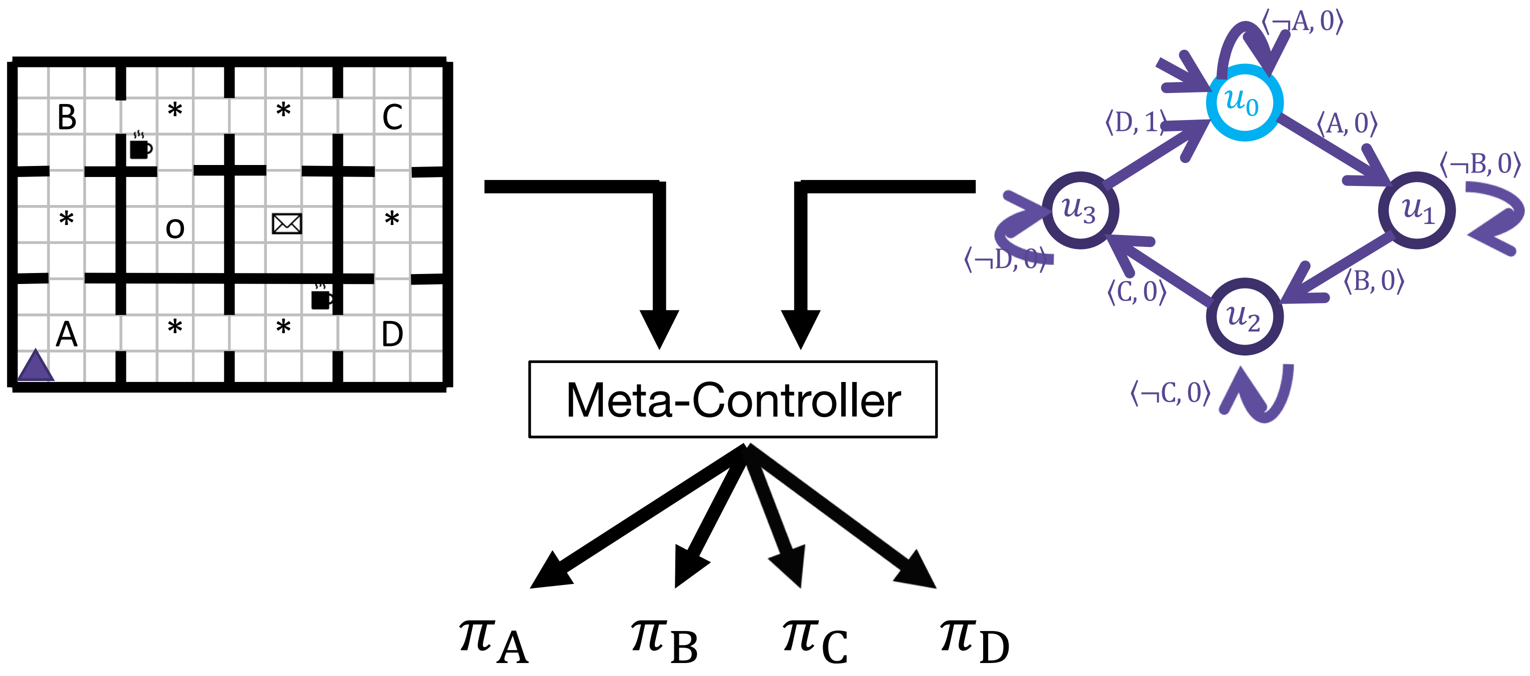
Learn one **option policy** for each proposition mentioned in the RM



- RM refers to A, B, C, and D
- Learn policies π_{A}, π_{B}, π_{C}, and π_{D}
- Optimize π_i, to satisfy i optimally

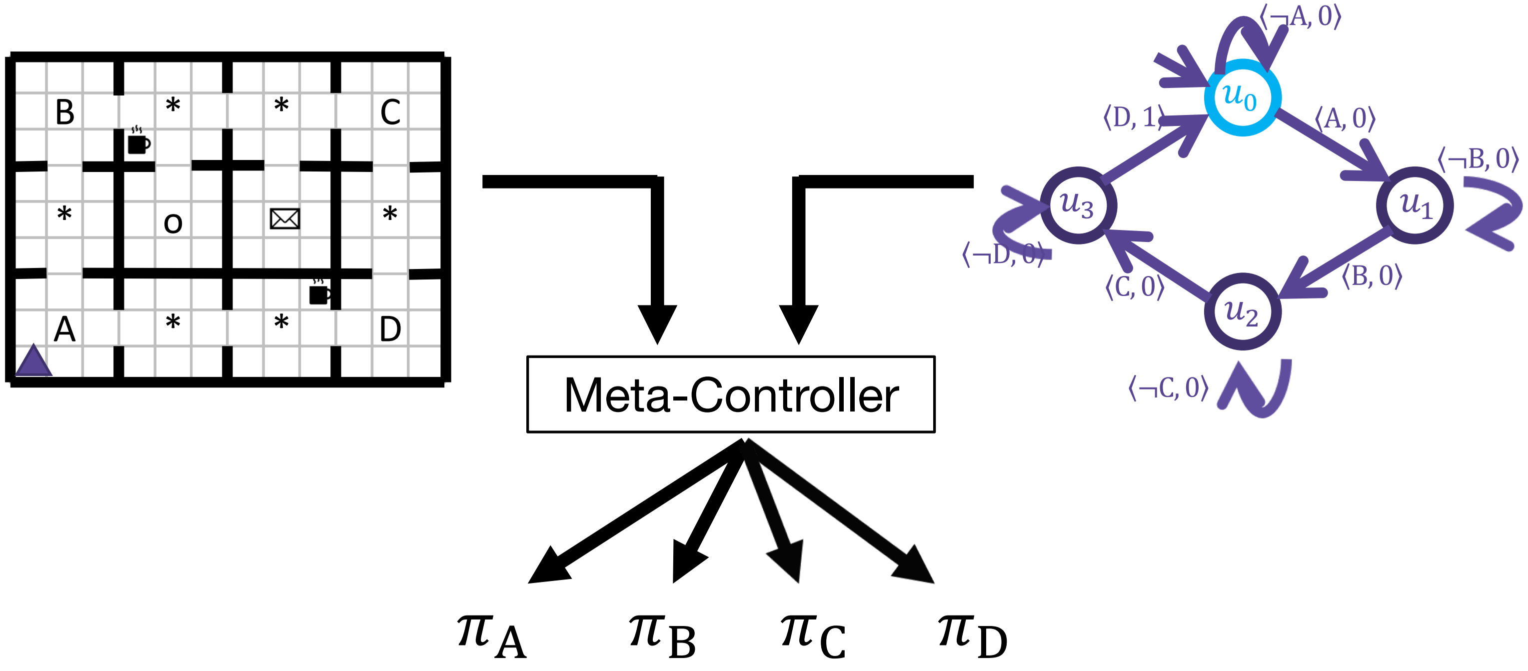
2. Option-Based Hierarchical RL (HRL)

Simultaneously learn when to use each option policy



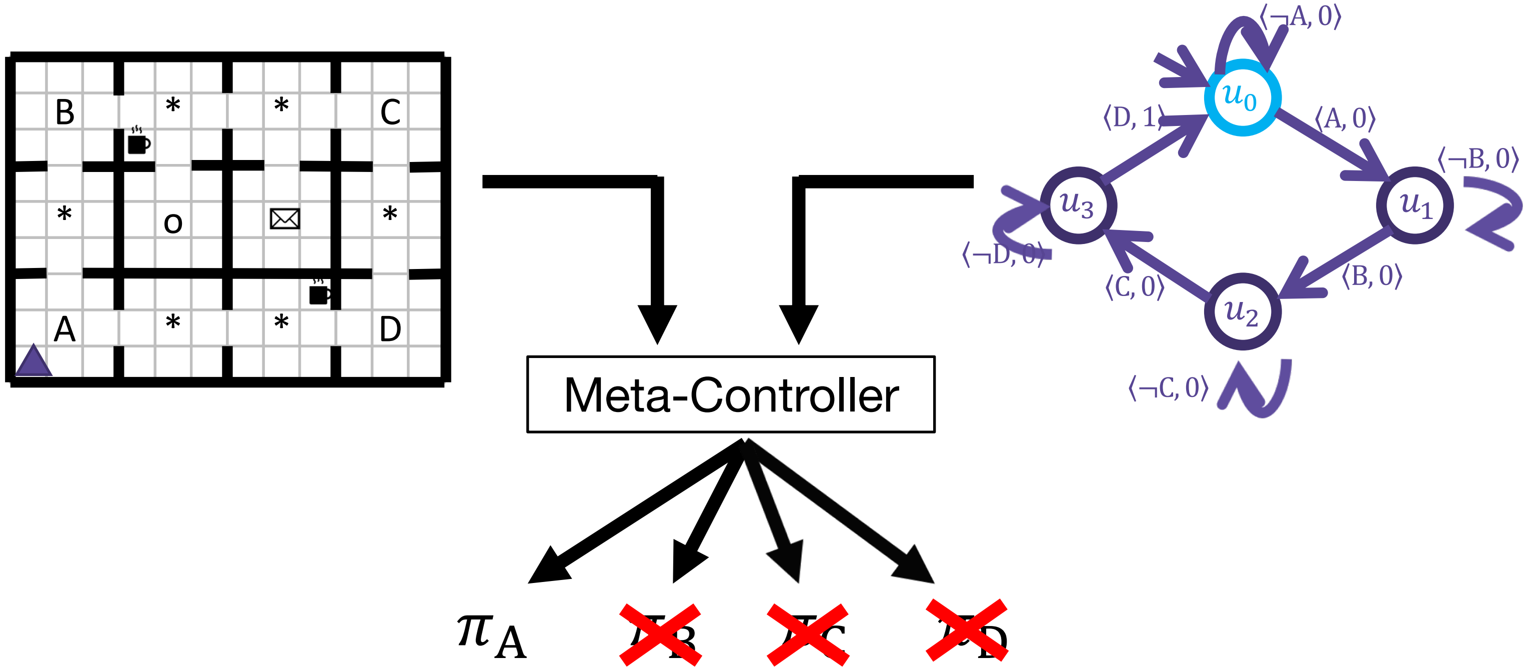
3. HRL with RM-Based Pruning (HRL-RM)

Prune irrelevant options using current RM state

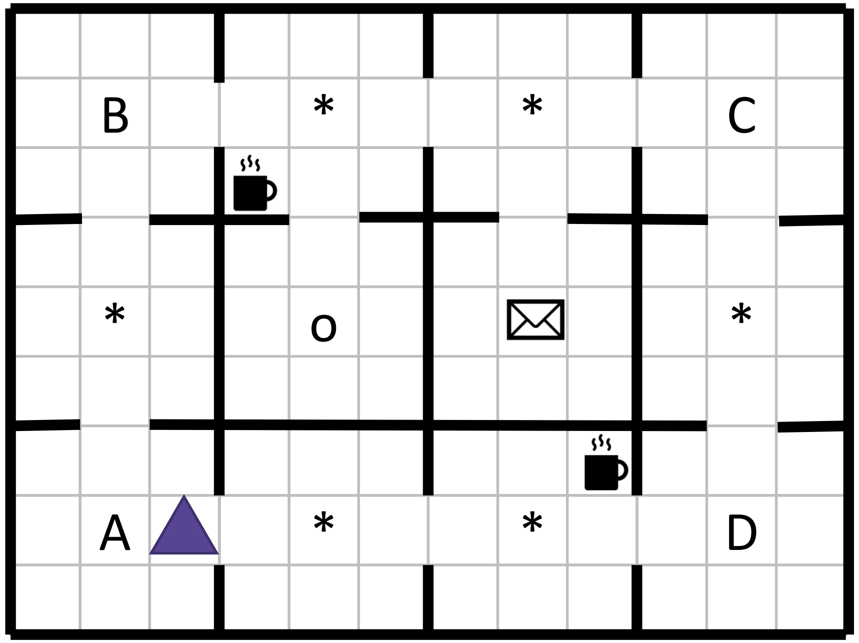


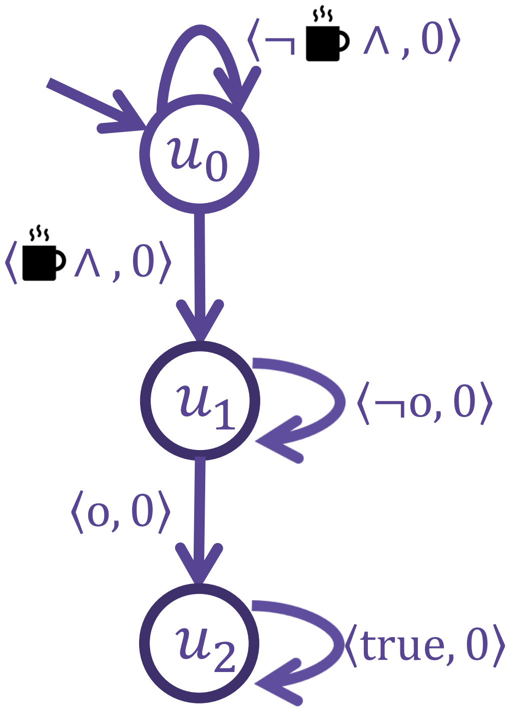
3. HRL with RM-Based Pruning (HRL-RM)

Prune irrelevant options using current RM state



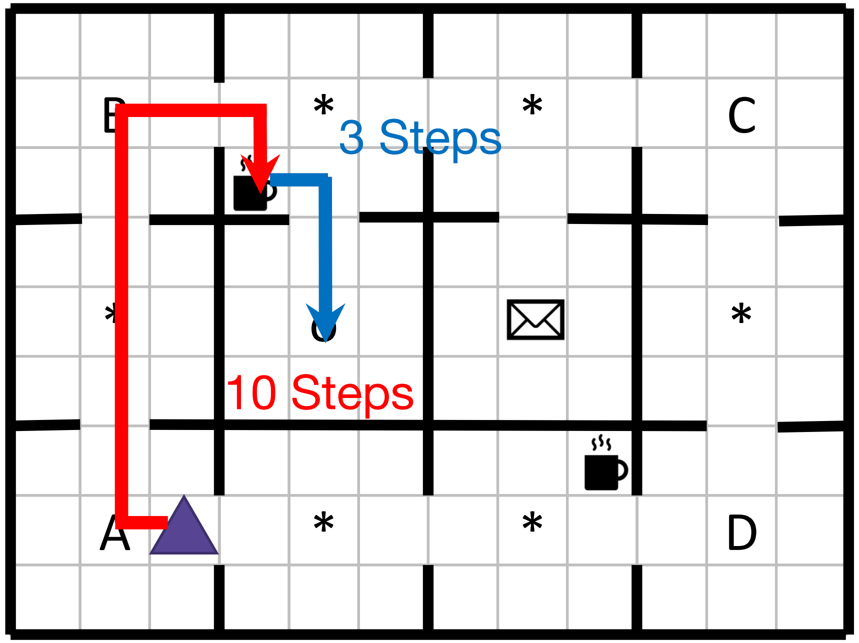
HRL Methods Can Find Suboptimal Policies

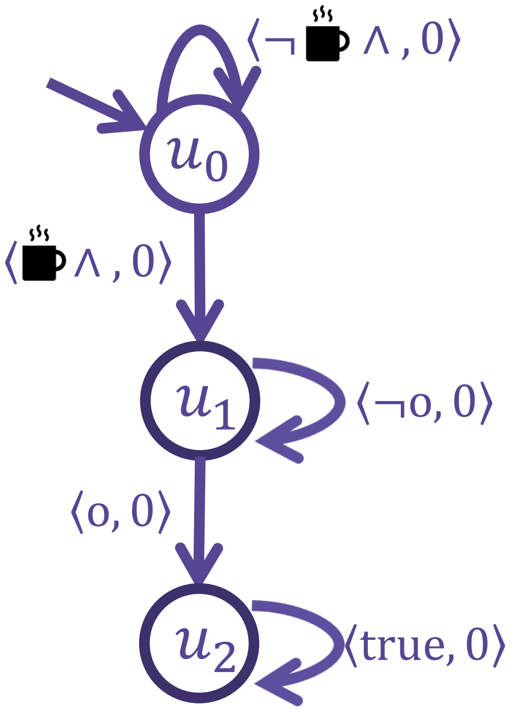




HRL approaches find "locally" optimal solutions.

HRL Methods Can Find Suboptimal Policies



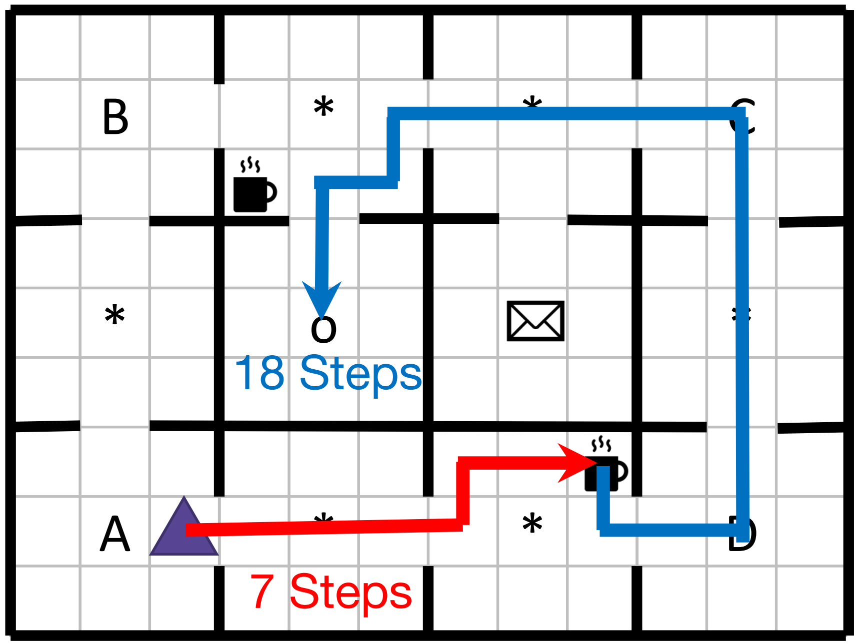


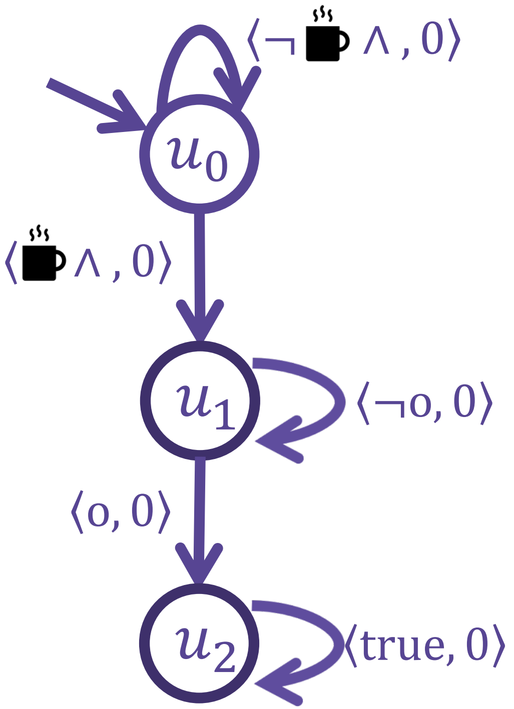
Optimal solution ($\gamma < 1$)

- 13 total steps

HRL approaches find "locally" optimal solutions.

HRL Methods Can Find Suboptimal Policies





Learns two options:

1. Getting
2. Getting to "o"

HRL approaches find "locally" optimal solutions.

Recall: Methods for Exploiting RM Structure

Baselines based on existing methods:

1. Q-learning over an equivalent MDP (Q-learning)
2. Hierarchical RL based on options (HRL)
3. HRL with RM-based pruning (HRL-RM)

Our approaches:

4. Q-learning for Reward Machines (QRM)
5. QRM + Reward Shaping for Reward Machine (QRM + RS)

Recall: Methods for Exploiting RM Structure

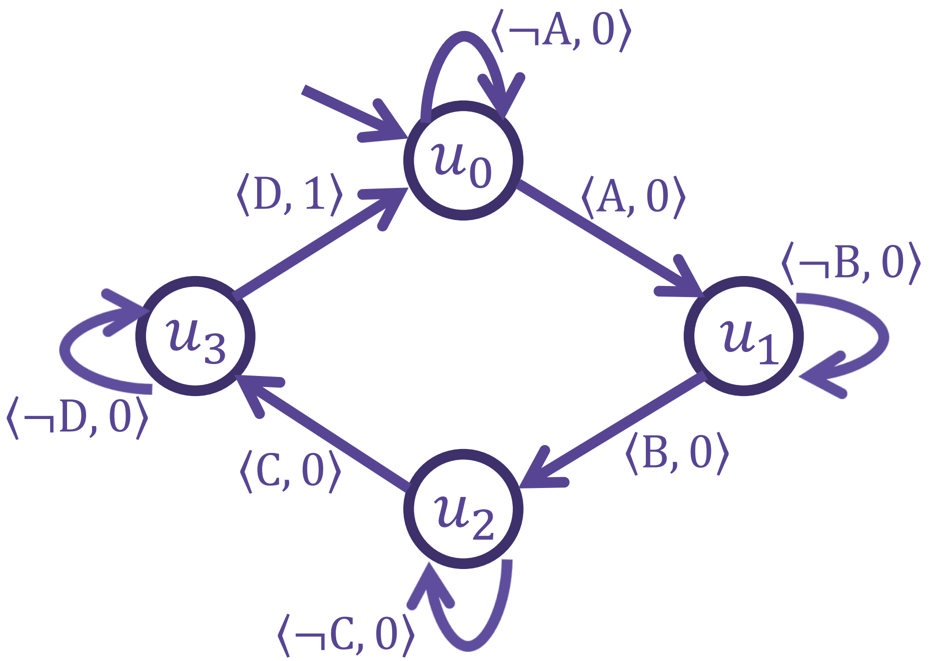
Baselines based on existing methods:

1. Q-learning over an equivalent MDP (Q-learning)
2. Hierarchical RL based on options (HRL)
3. HRL with RM-based pruning (HRL-RM)

Our approaches:

4. Q-learning for Reward Machines (QRM)
5. QRM + Reward Shaping for Reward Machine (QRM + RS)

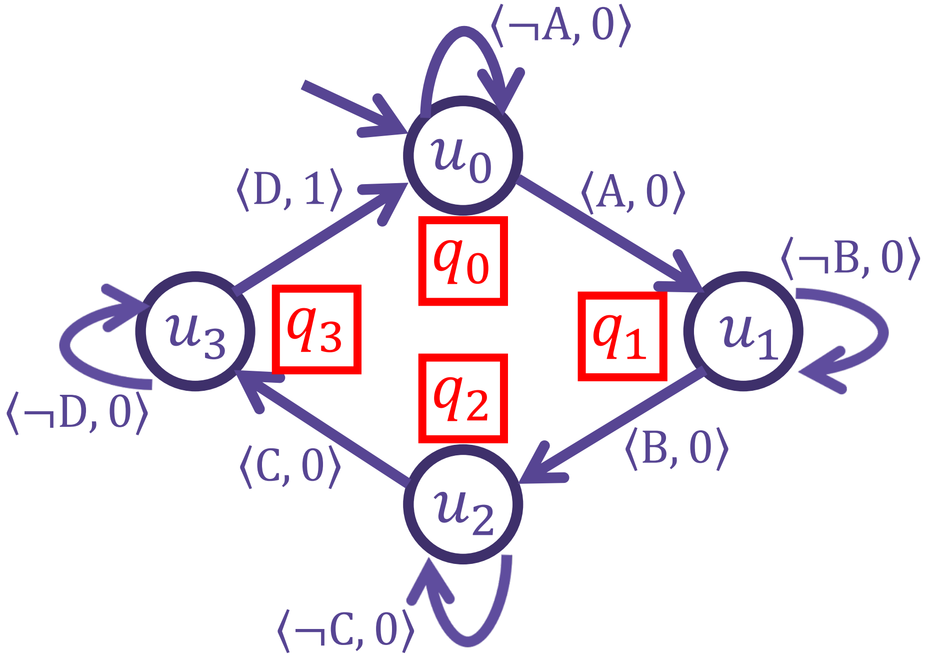
4. Q-Learning for Reward Machines (QRM)



4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

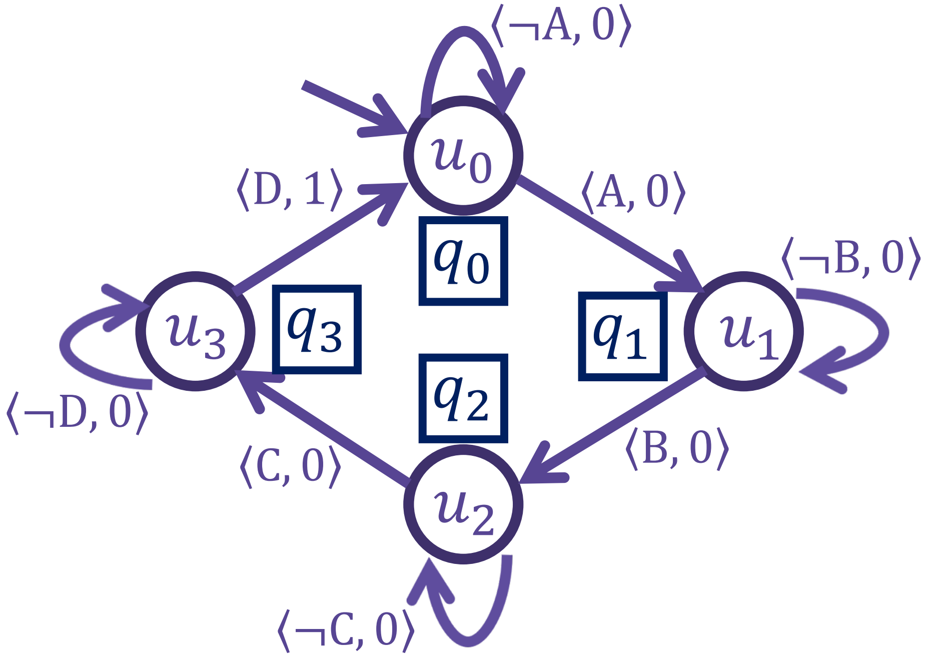
1. Learn one policy (q-value function) per state in the Reward Machine.



4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

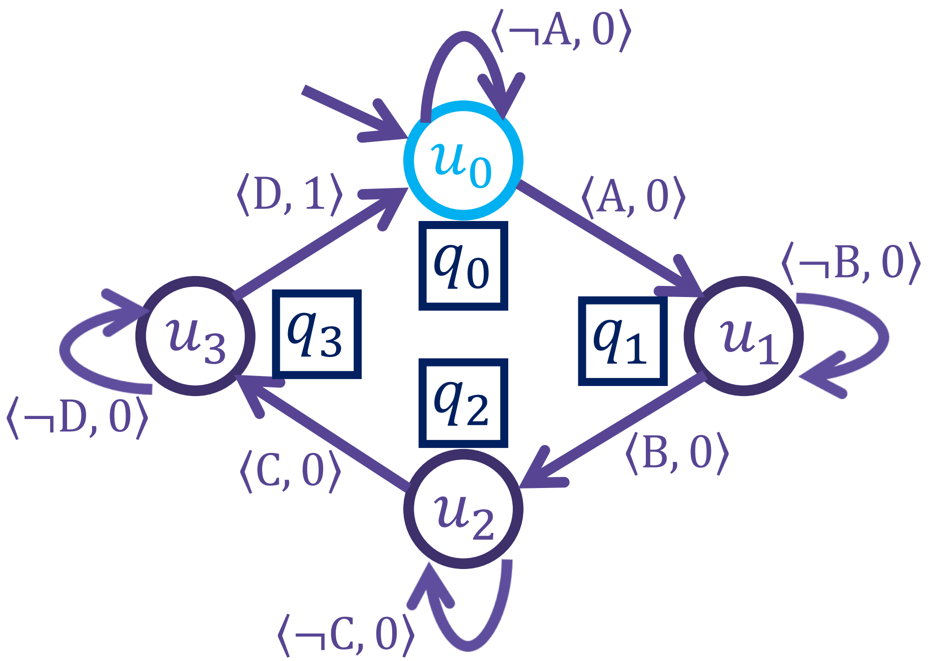
1. Learn one policy (q-value function) per state in the Reward Machine.
2. Select actions using the policy of the current RM state.



4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

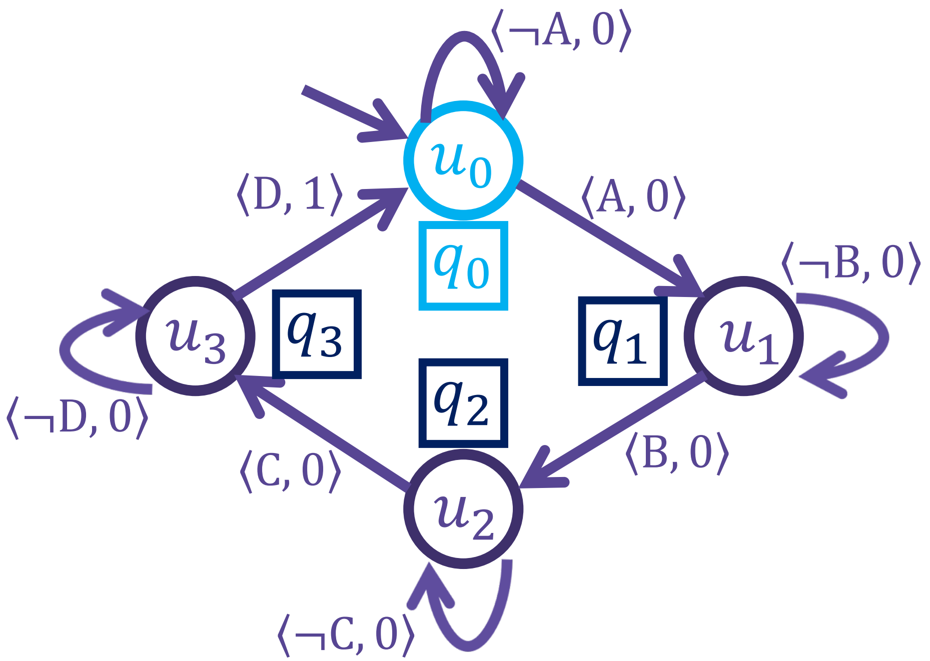
1. Learn one policy (q-value function) per state in the Reward Machine.
2. Select actions using the policy of the current RM state.



4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

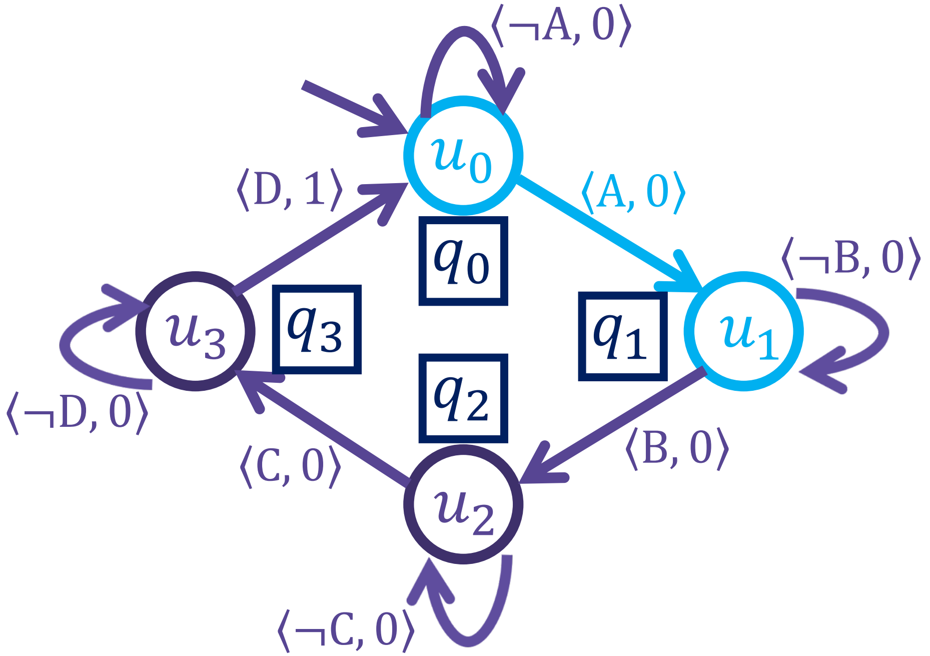
1. Learn one policy (q-value function) per state in the Reward Machine.
2. Select actions using the policy of the current RM state.



4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

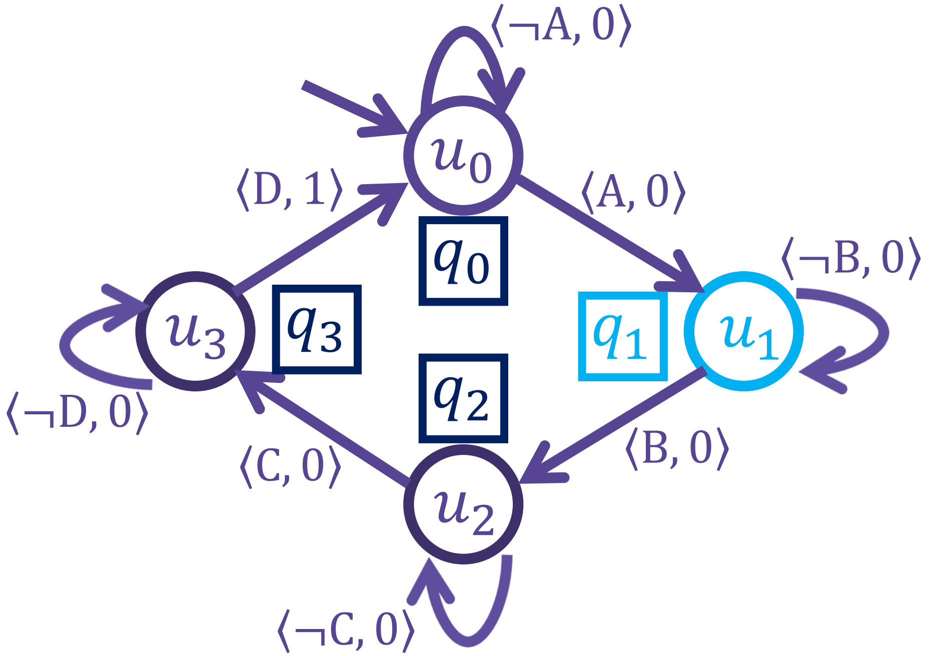
1. Learn one policy (q-value function) per state in the Reward Machine.
2. Select actions using the policy of the current RM state.



4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

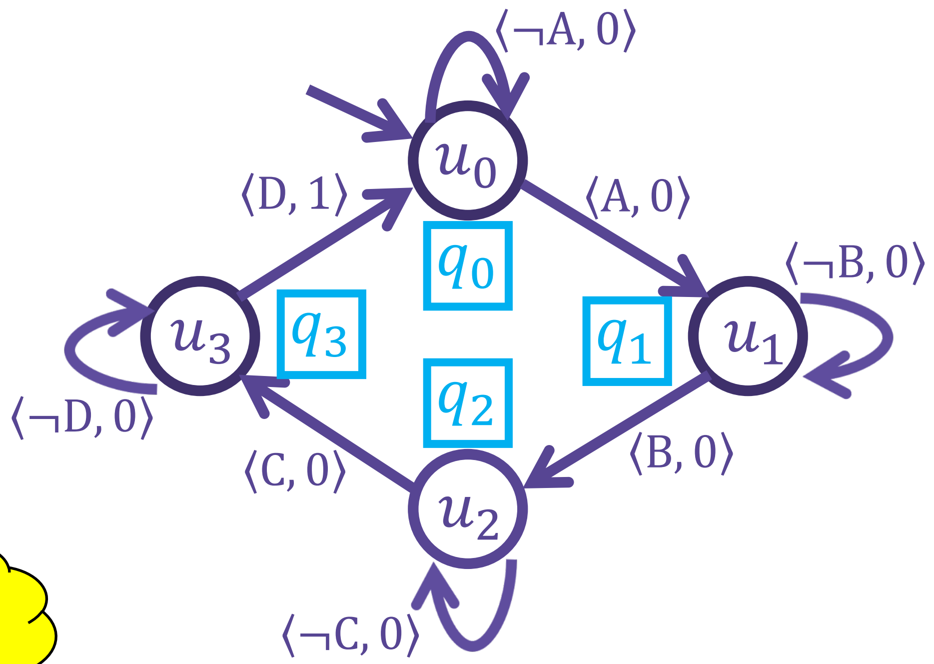
1. Learn one policy (q-value function) per state in the Reward Machine.
2. Select actions using the policy of the current RM state.



4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

1. Learn one policy (q-value function) per state in the Reward Machine.
2. Select actions using the policy of the current RM state.
3. Reuse experience to update all q-value functions on every transition via off-policy reinforcement learning.



This is a form of **Counterfactual Reasoning**

Recall: Methods for Exploiting RM Structure

Baselines based on existing methods:

1. Q-learning over an equivalent MDP (Q-learning)
2. Hierarchical RL based on options (HRL)
3. HRL with RM-based pruning (HRL-RM)

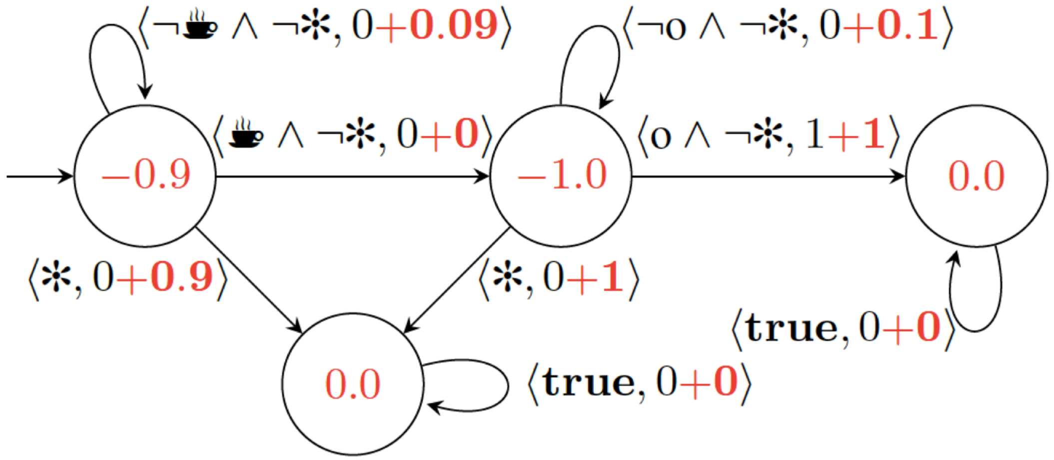
Our approaches:

4. Q-learning for Reward Machines (QRM)
5. QRM + Reward Shaping for Reward Machine (QRM+RS)

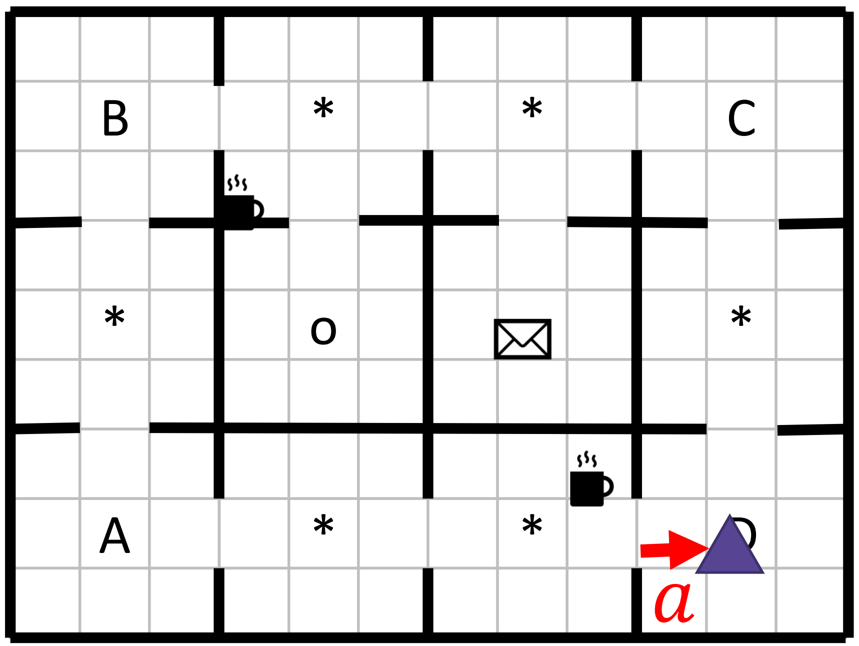
5. QRM + Reward Shaping (QRM + RS)

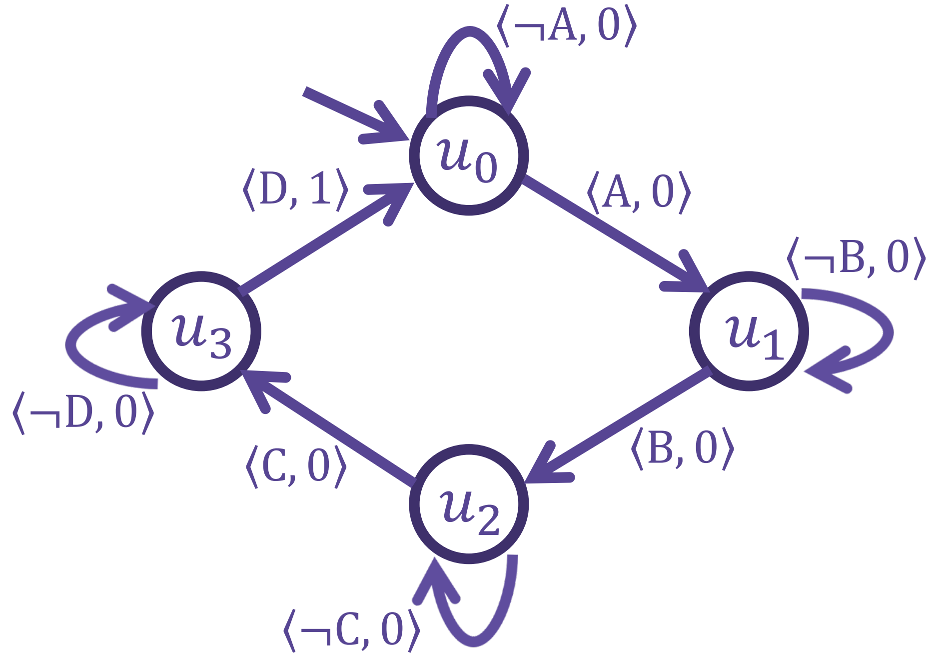
QRM + RS (our approach)

1. Treat the RM itself as an MDP and perform value iteration over the RM.
2. Apply QRM to the shaped RM



Optimality of QRM and QRM+RS





Theorem: QRM and QRM+RS converge to the optimal policy in the limit.

The Rest of the Talk

- Reward Machines (RM)
- Exploiting RM Structure in Learning
- ▶ Experiments
- Creating Reward Machines
- Concluding Remarks

EXPERIMENTS

Test Domains

- Two domains with a discrete action and state-space
  - Office domain (4 tasks)
  - Craft domain (10 tasks)
- One domain with a continuous state-space
  - Water World domain (10 tasks)

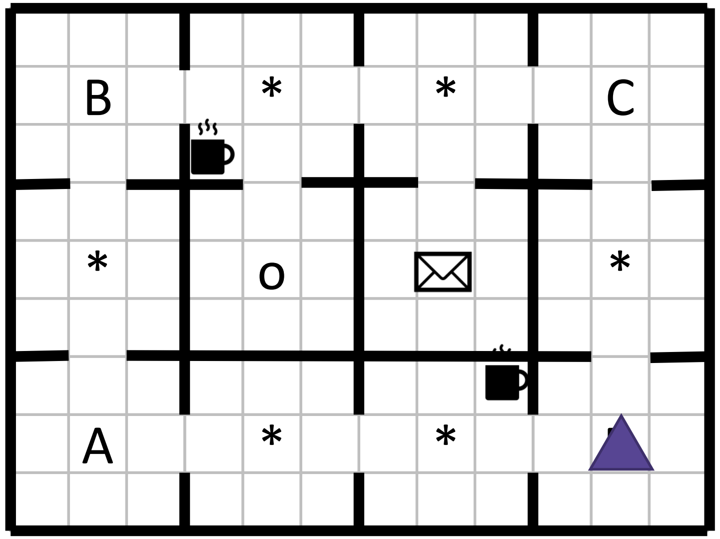
Test in Discrete Domains

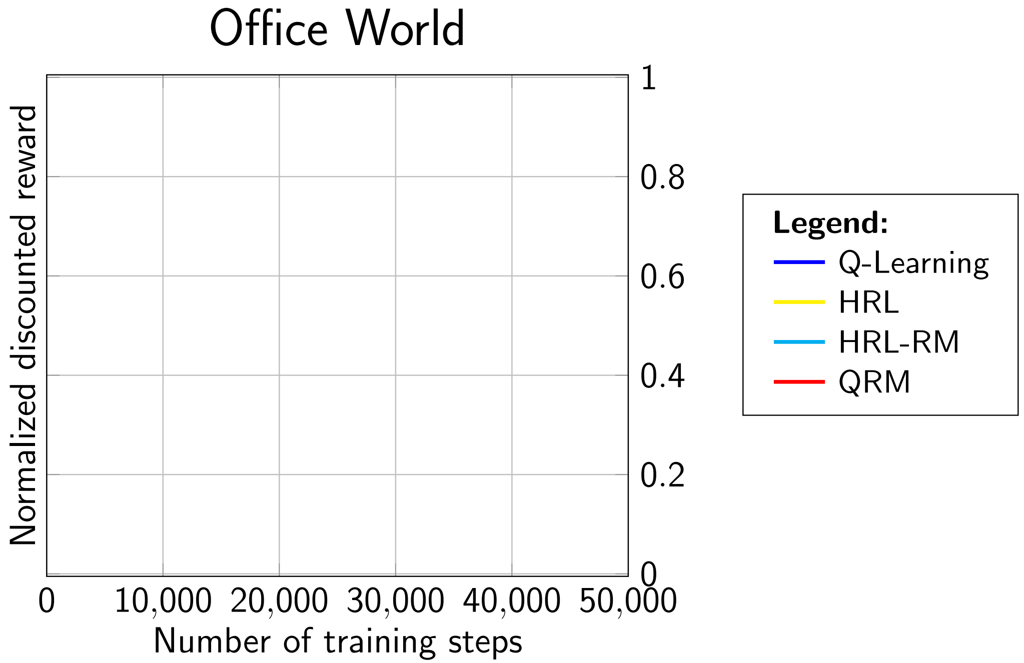
Tested all five approaches

1. Q-learning over an equivalent MDP (Q-learning)
2. Hierarchical RL based on options (HRL)
3. HRL with RM-based pruning (HRL-RM)
4. Q-learning for Reward Machines (QRM)
5. QRM + Reward Shaping (QRM + RS)

Method	**Optimality?**	**Decomposition?**
Q-Learning	✓	
HRL		✓
HRL-RM		✓
QRM	✓	✓
QRM + RS	✓	✓

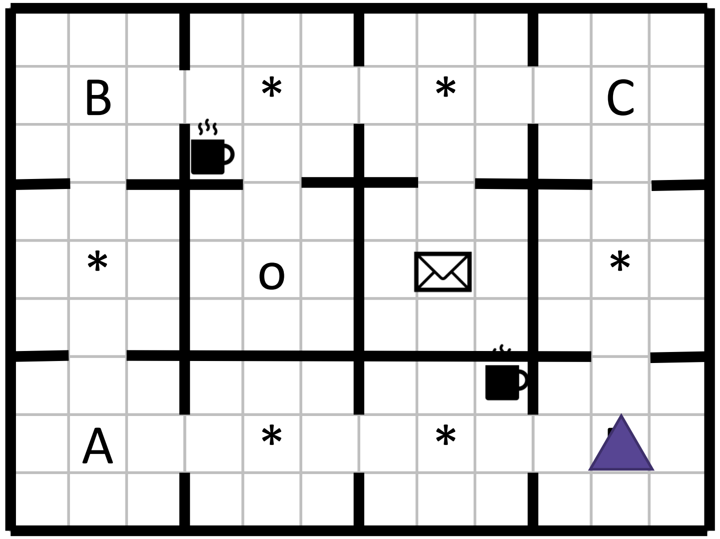
Office World Experiments

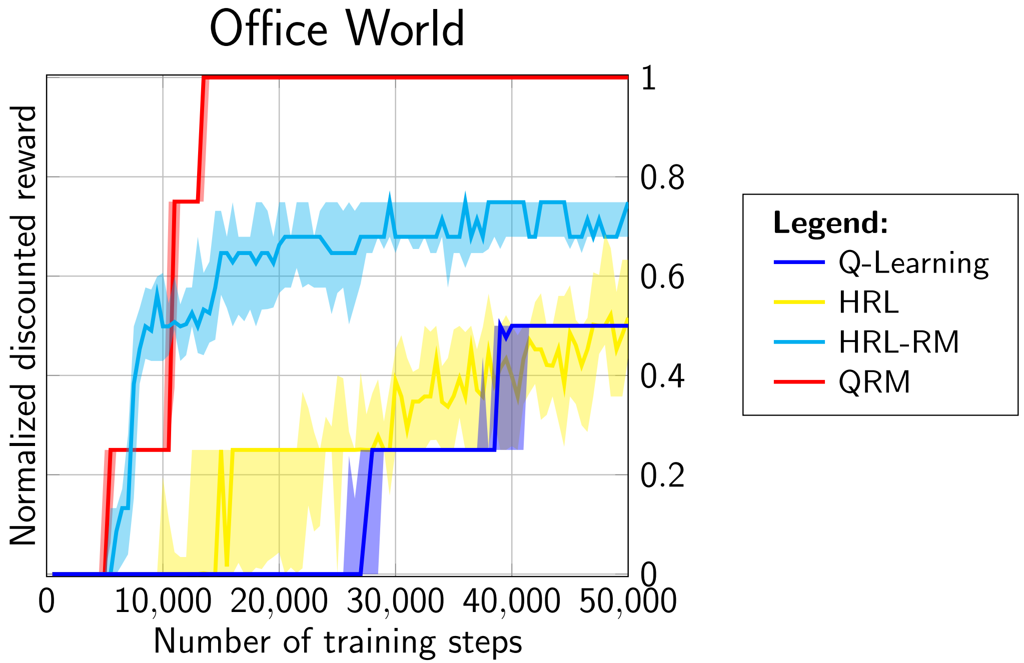




4 tasks, 30 independent trials per task

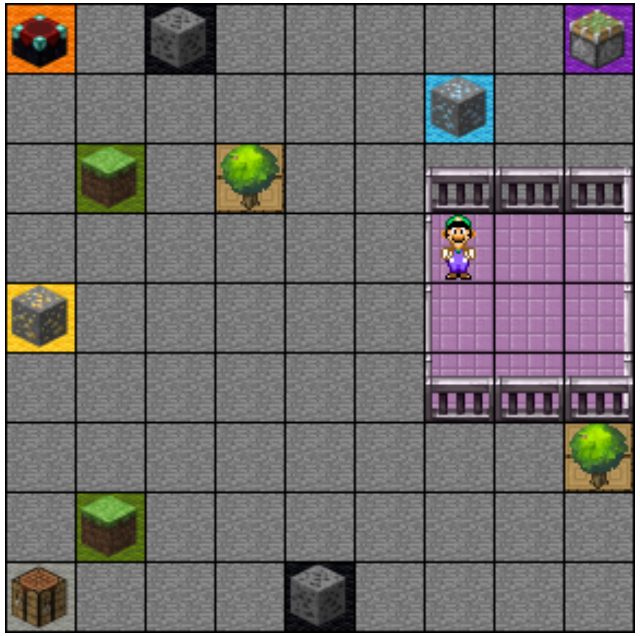
Office World Experiments

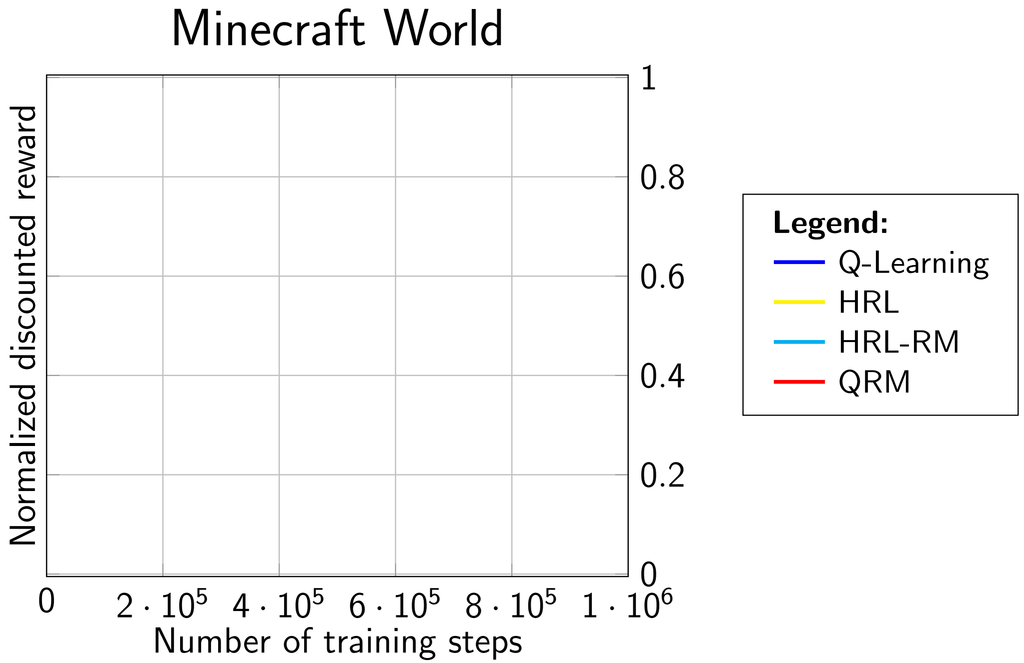




4 tasks, 30 independent trials per task

Minecraft World Experiments

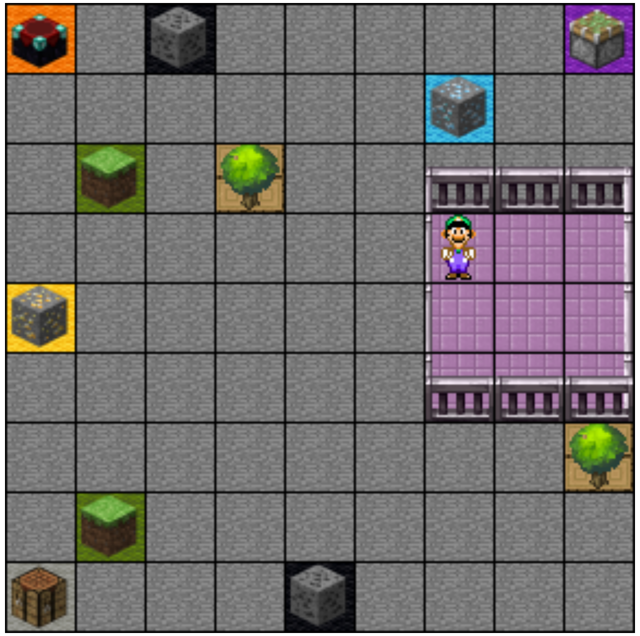


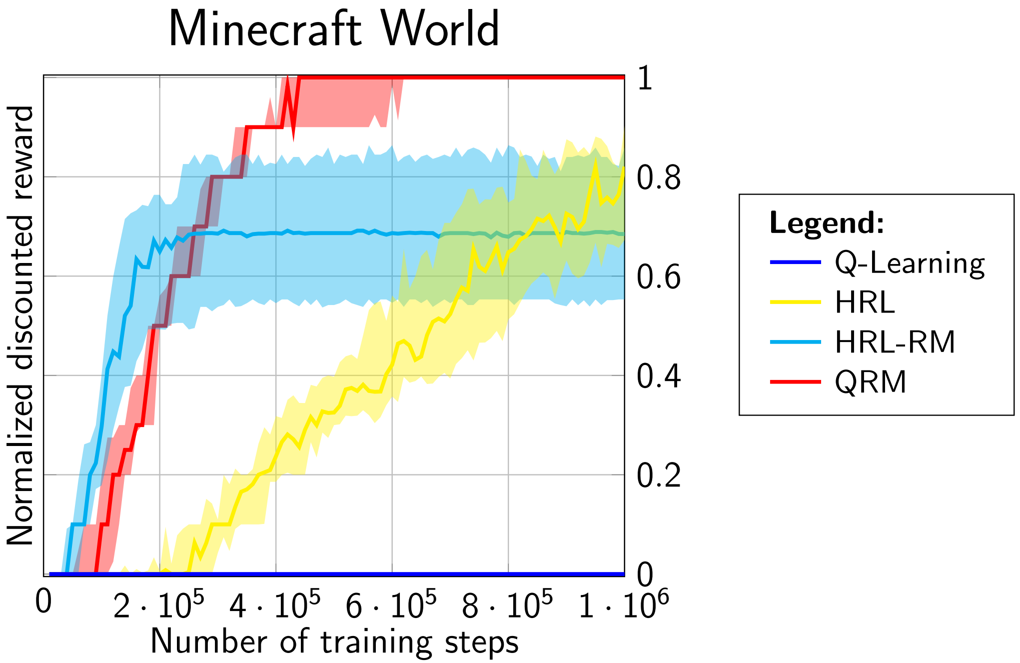


10 tasks over 10 random maps, 3 independent trials per combination

Tasks from Andreas *et al.* (ICML 2017)

Minecraft World Experiments





10 tasks over 10 random maps, 3 independent trials per combination

Tasks from Andreas *et al.* (ICML 2017)

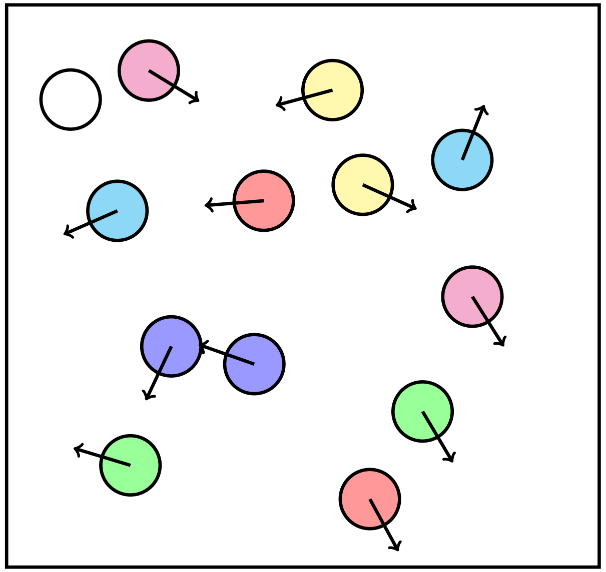
Function Approximation with QRM

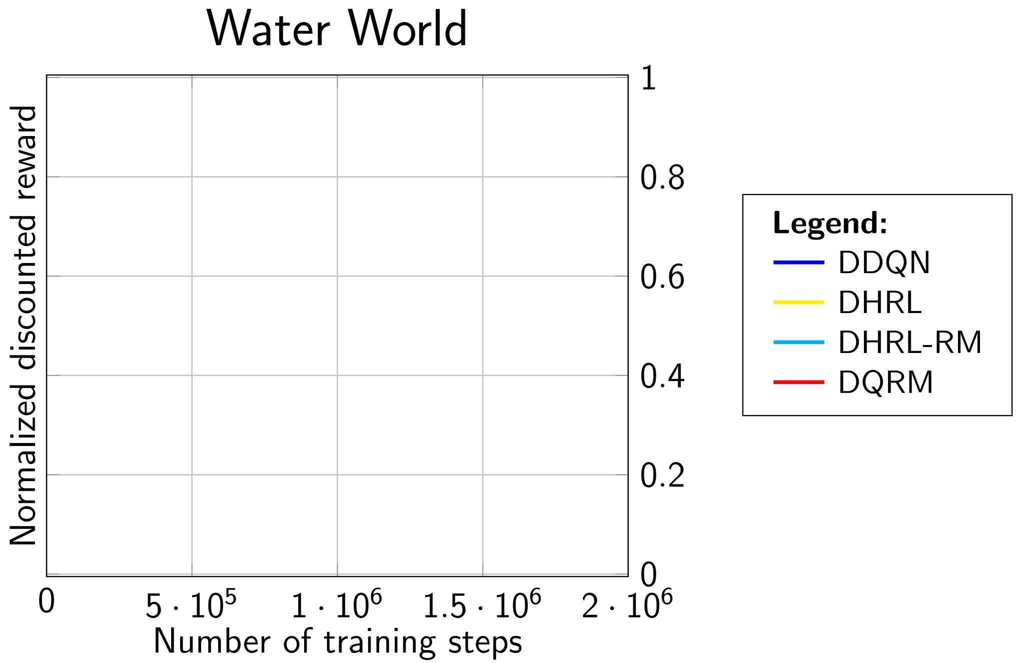
From tabular QRM to Deep QRM

- Replace Q-learning by Double DQN (DDQN) with prioritized experience replays

Method	Optimality?	Decomposition?
Q-Learning		
HRL		✓
HRL-RM		✓
QRM		✓
QRM + RS		✓

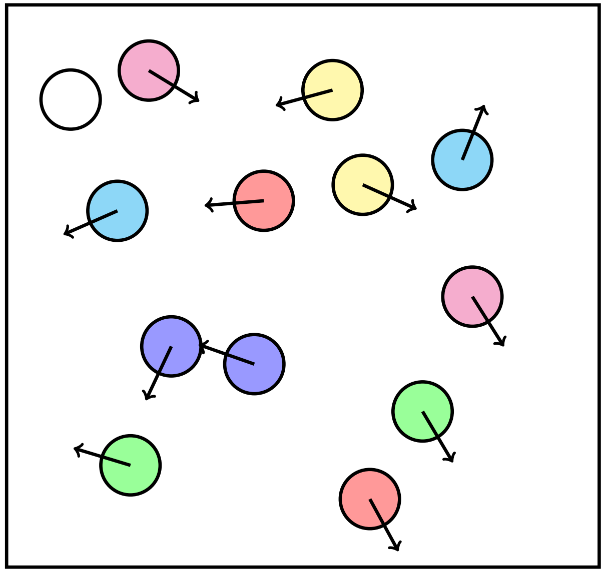
Water World Experiments

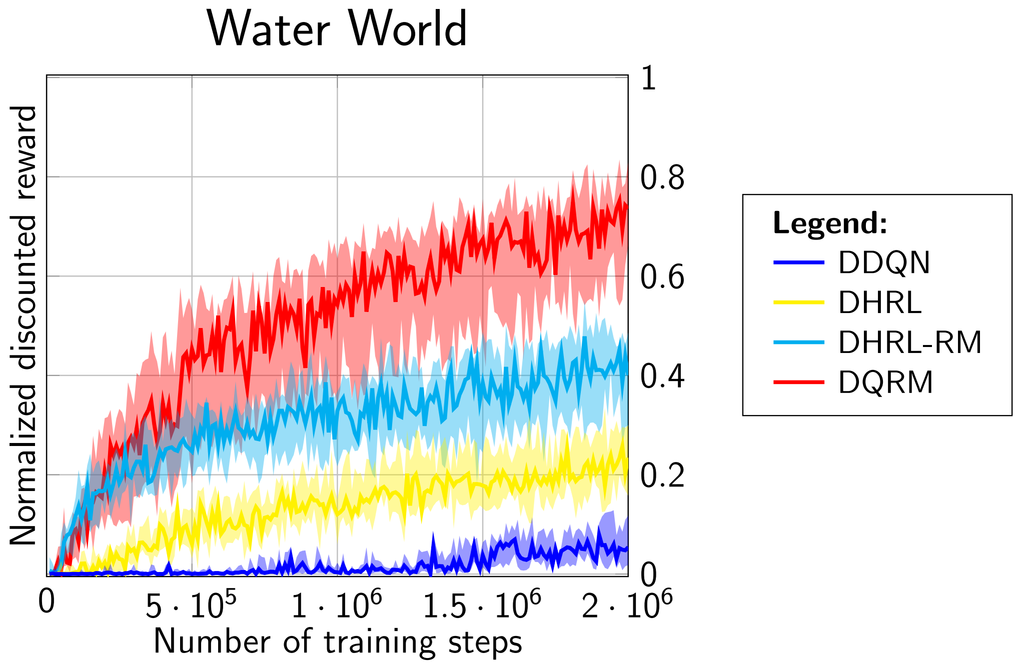




10 tasks over 10 random maps, 3 independent trials per combination

Water World Experiments





10 tasks over 10 random maps, 3 independent trials per combination

The Rest of the Talk

- Reward Machines (RM)
- Exploiting RM Structure in Learning
- Experiments
- ▶ Creating Reward Machines
- Recap

CREATING REWARD MACHINES

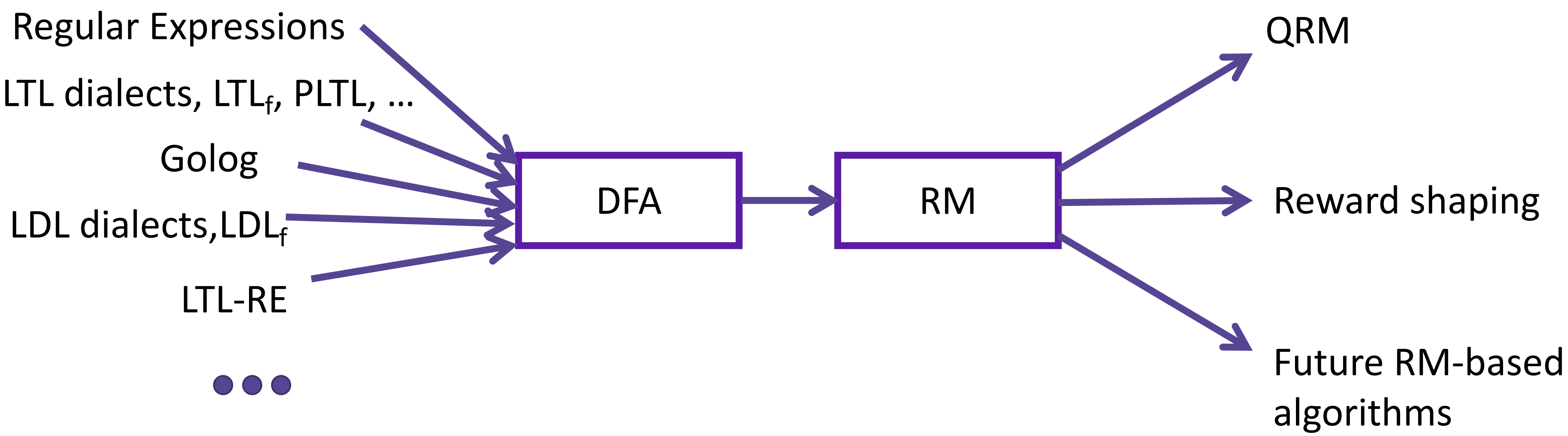
Creating Reward Machines

Where do Reward Machines come from?

1. Specify
2. Generate
3. Learn

1. Construct Reward Machine from Formal Languages

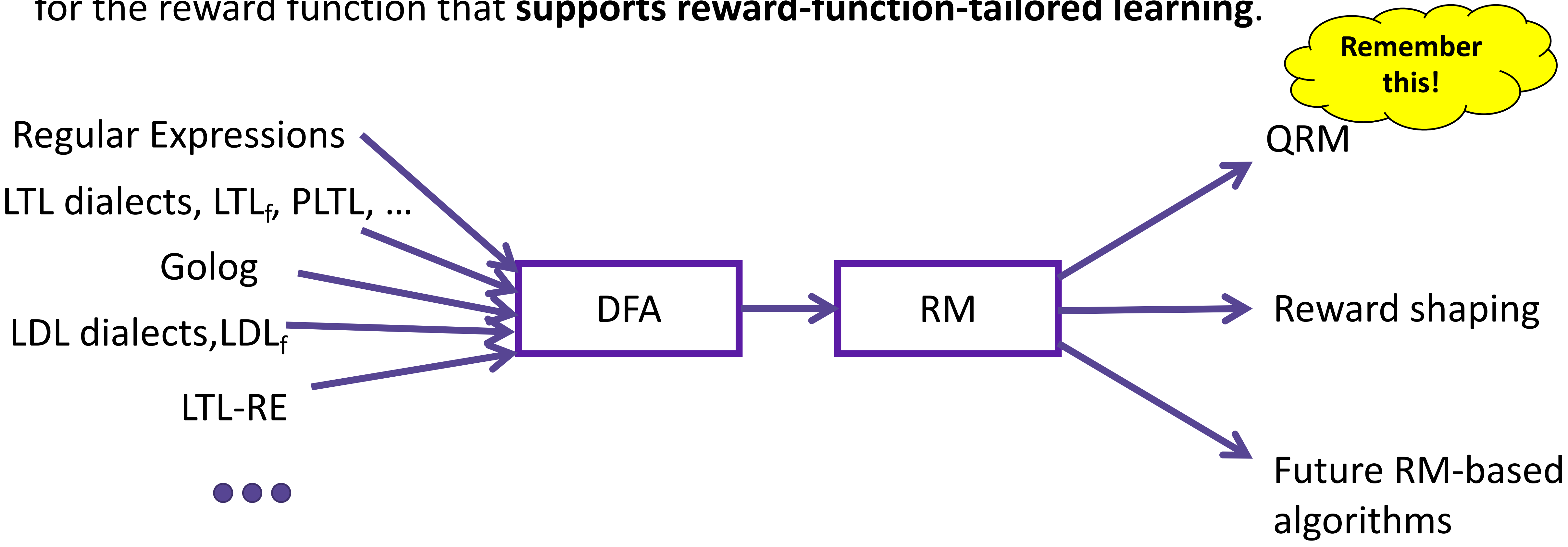
Reward Machines serves as a **lingua franca** and provide a **normal form representation** for the reward function that **supports reward-function-tailored learning**.



[Camacho, Toro Icarte, Klassen, Valenzano, M., IJCAI19]
[Middleton, Klassen, Baier, M, ICAPS2020 Systems Demo]

1. Construct Reward Machine from Formal Languages

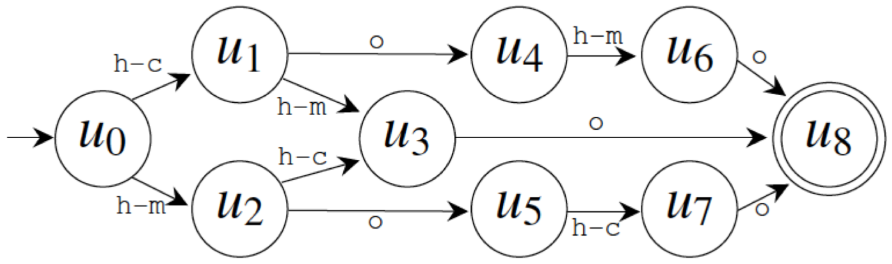
Reward Machines serves as a **lingua franca** and provide a **normal form representation** for the reward function that **supports reward-function-tailored learning**.



[Camacho, Toro Icarte, Klassen, Valenzano, M., IJCAI19]
[Middleton, Klassen, Baier, M, ICAPS2020 Systems Demo]

2. Generate RM using a Symbolic Planner

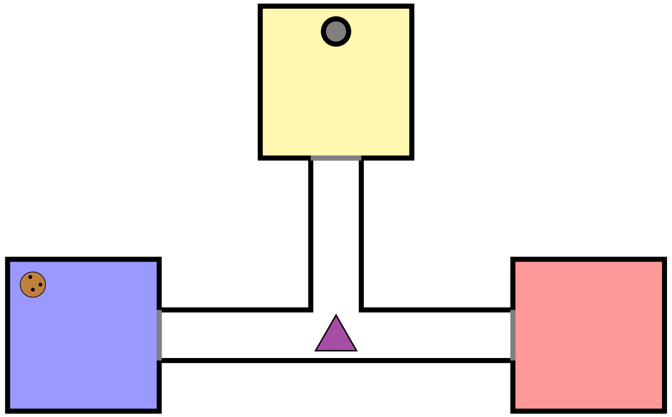
- ✓ high-level **model** to describe **abstract actions (options)**
- ✓ **symbolic planning to generate RMs** corresponding to high- level partial-order plans
- ✓ use **these abstract solutions to guide an RL agent**

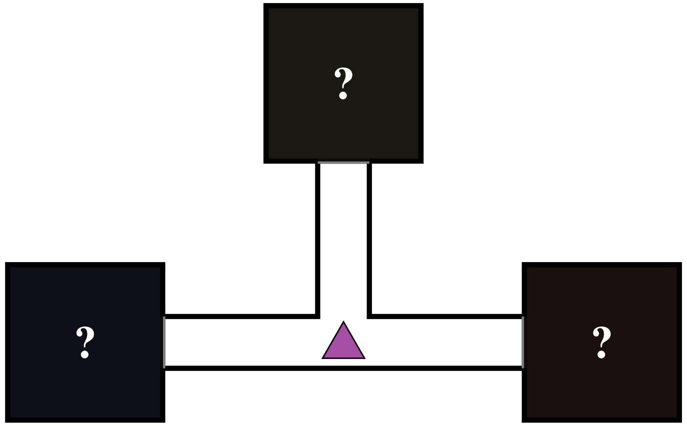


u_0: $\varnothing$
u_1: {get-coffee}
u_2: {get-mail}
u_3: {get-coffee, get-mail}
u_4: {get-coffee, deliver-coffee}
u_5: {get-mail, deliver-mail}
u_6: {get-coffee, get-mail, deliver-coffee}
u_7: {get-mail, get-coffee, deliver-mail}
u_8: {get-coffee, get-mail, deliver-coffee, deliver-mail}

[Illanes, Yan, Toro Icarte, M., RLDM19, ICAPS20, KR2ML@NeurIPS20]

3. Learn RMs for Partially-Observable RL





Problem: Find a policy that maximizes the external reward given by a partially observable environment

Assumptions: Agent has a set of high-level binary classifiers/event detectors (e.g., button-pushed, cookies, etc.)

Key Insight: Learn an RM such that **its internal state can be effectively used as external memory** by the agent to solve the task.

Approach: Discrete Optimization via Tabu Search

3. Learn RMs for Partially-Observable RL

These "toy problems" cannot be solved by A3C, PPO, and ACER with LSTMs

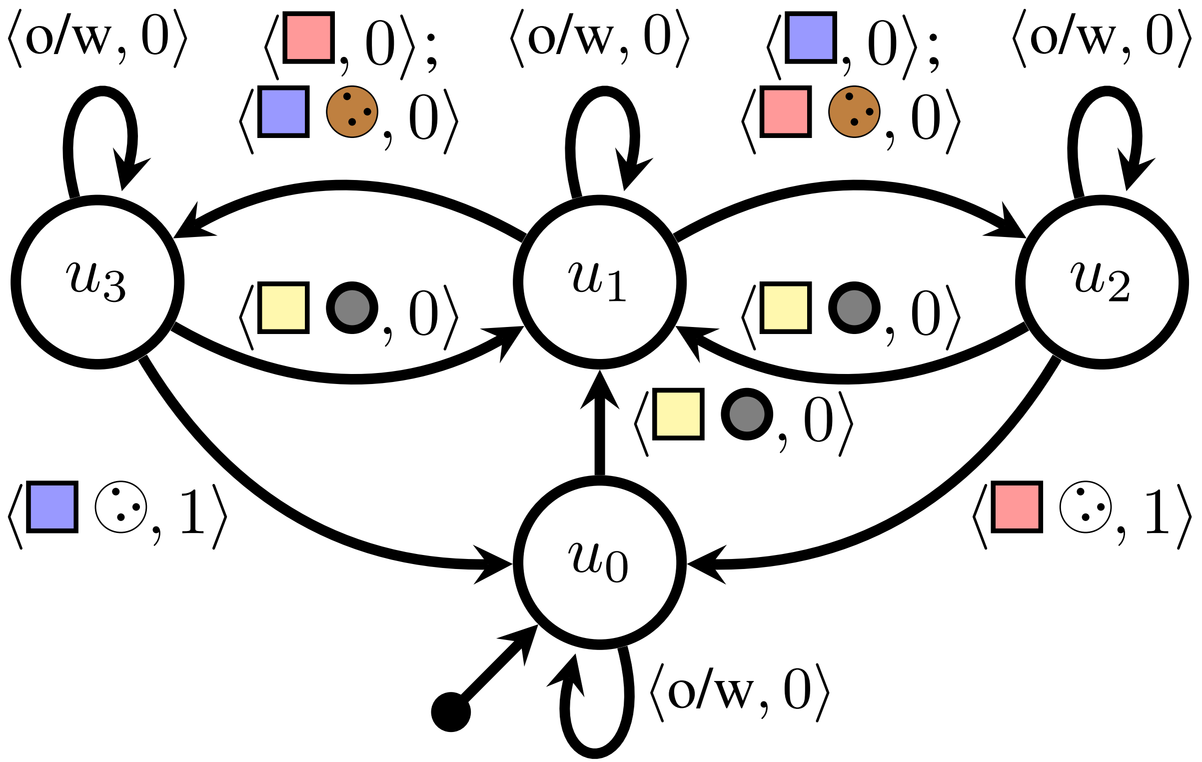
Problem: Find a policy that maximizes the external reward given by a partially observable environment

Assumptions: Agent has a set of high-level binary classifiers/event detectors (e.g., button-pushed, cookies, etc.)

Key Insight: Learn an RM such that **its internal state can be effectively used as external memory** by the agent to solve the task.

Approach: Discrete Optimization via Tabu Search

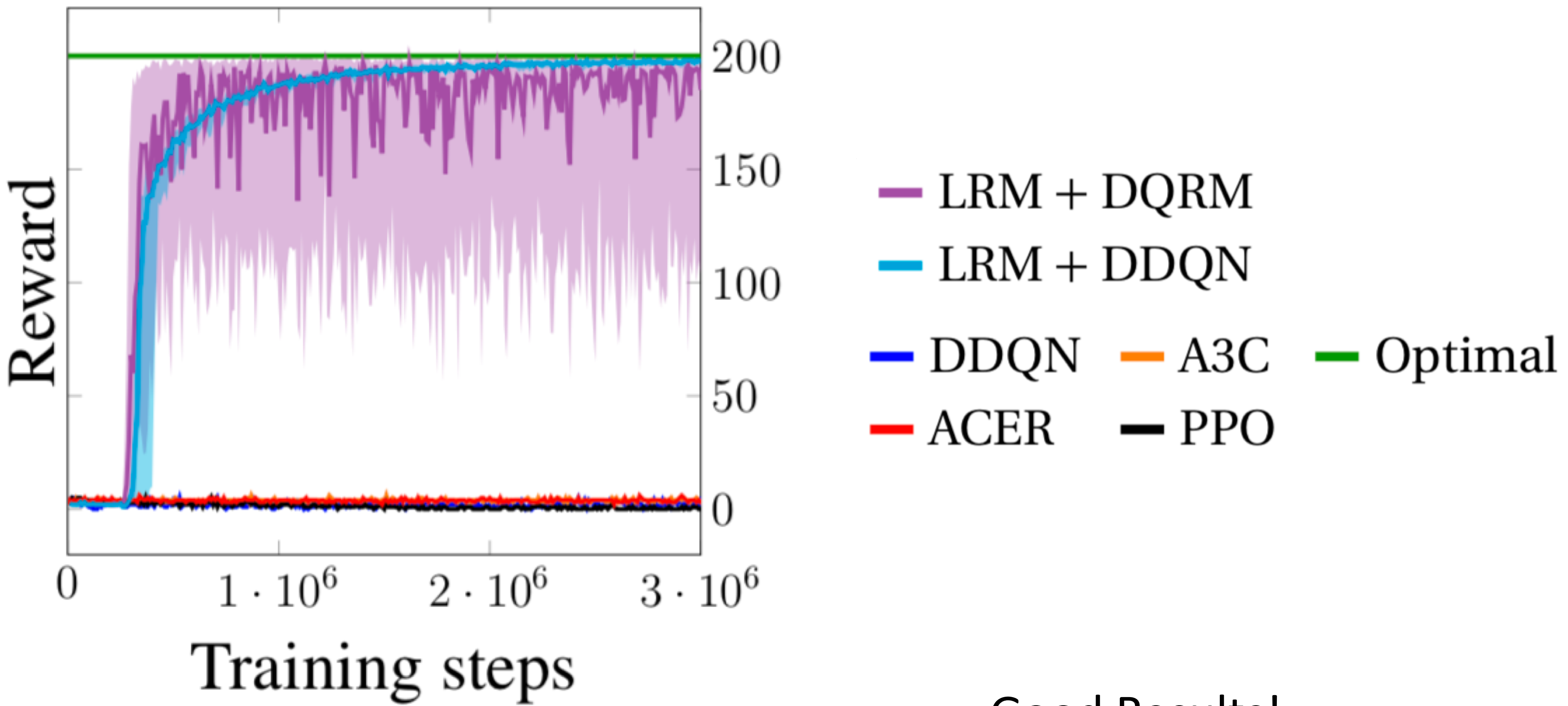
3. Learn Reward Machines (LRM)



More **human interpretable** concept of what the agent is trying to do

[Toro Icarte; Waldie; Klassen; Valenzano; Castro; M, NeurIPS 2019]

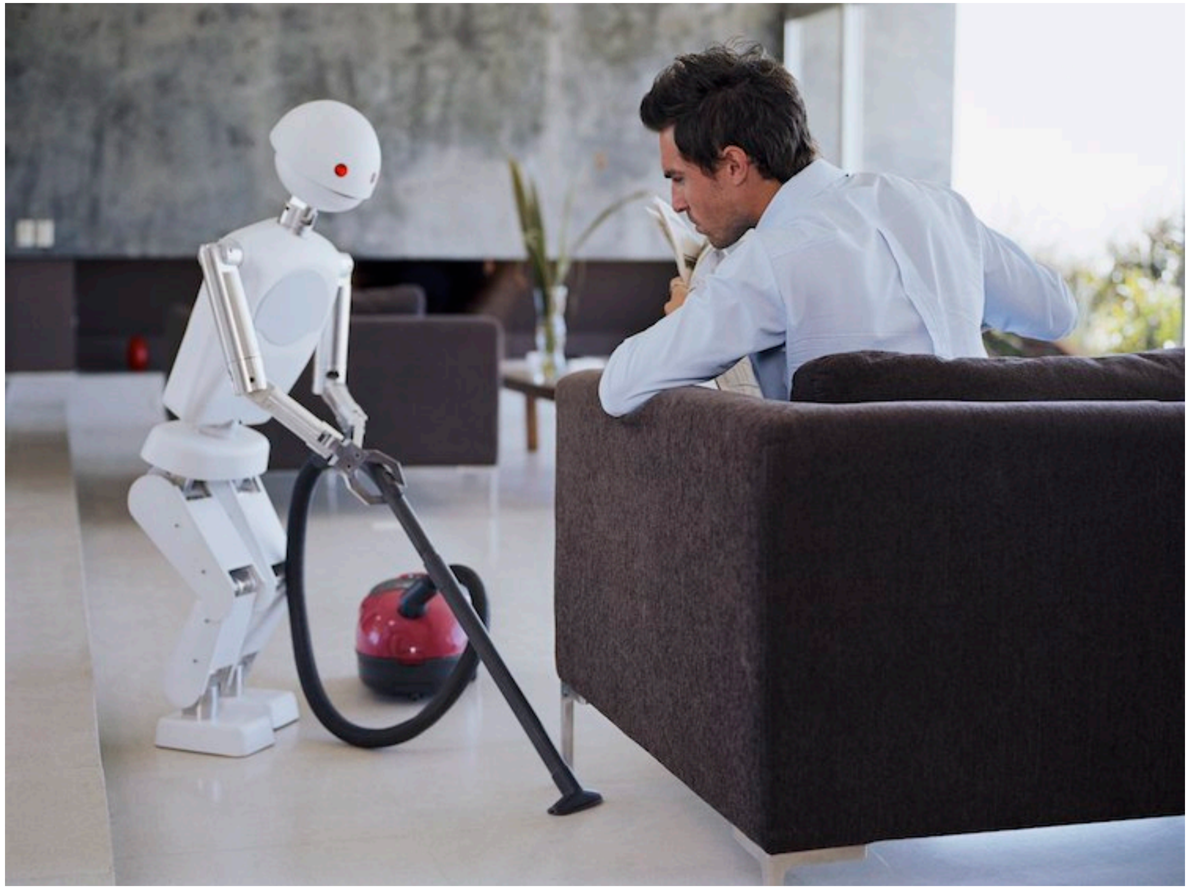
3. Learn Reward Machines (LRM)



Good Results!

[Toro Icarte, Waldie, Klassen, Valenzano, Castro, M, NeurIPS 2019]

RECAP

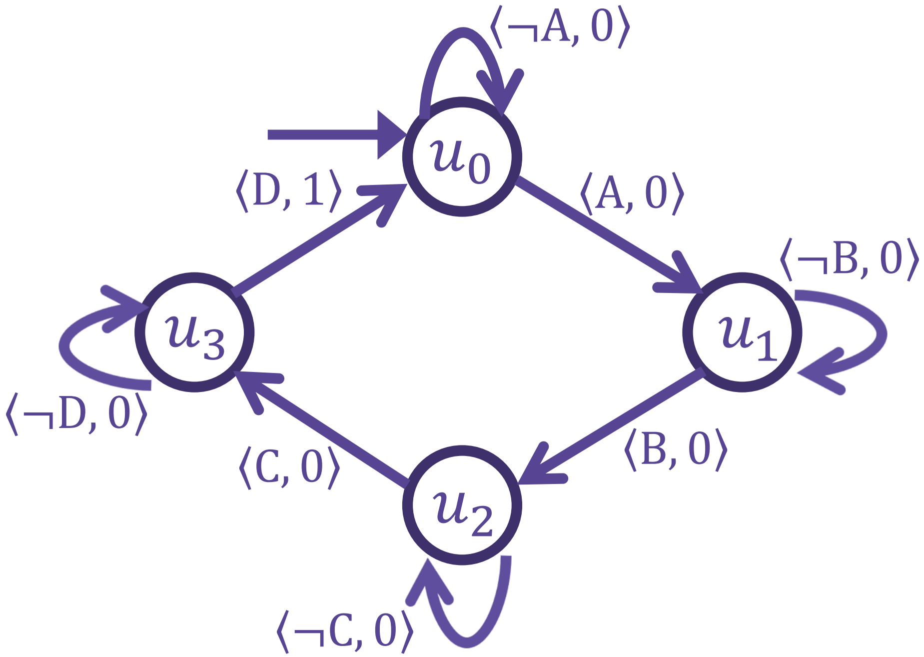


How do we advise, instruct, task, … and impart knowledge to AI that learns?

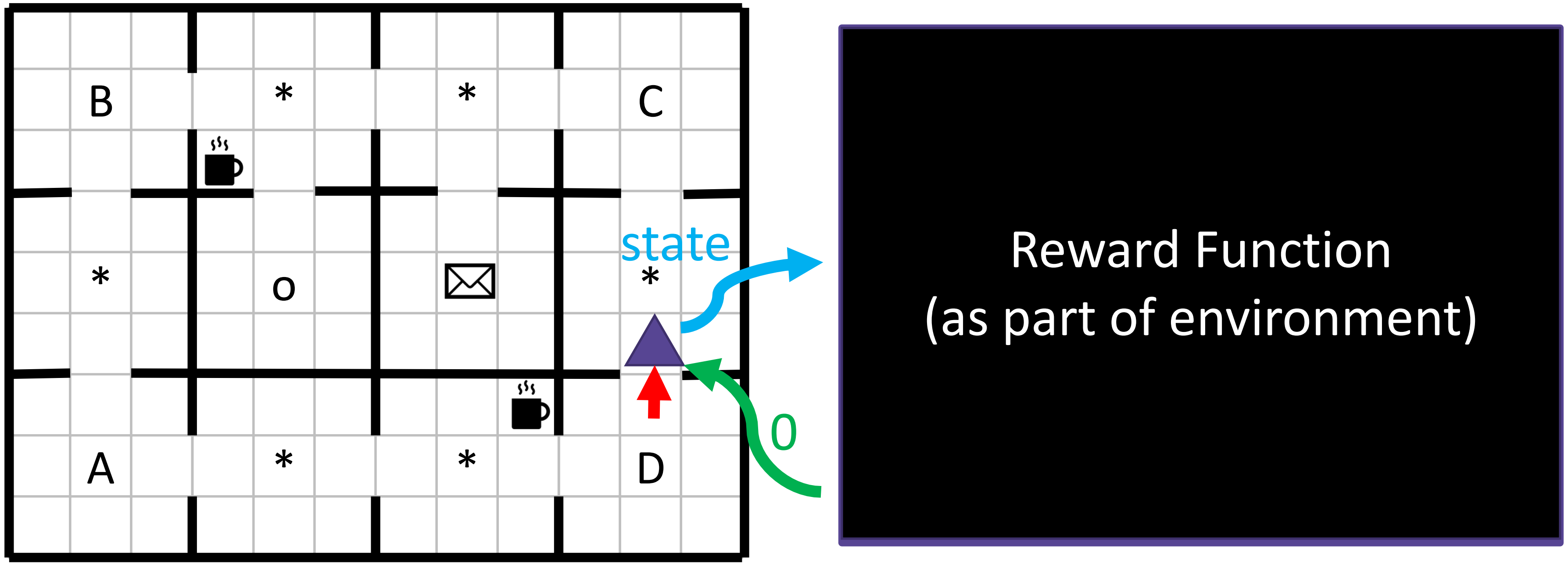
Photo: Javier Pierin (Getty Images)

Big Idea: Reward Machines

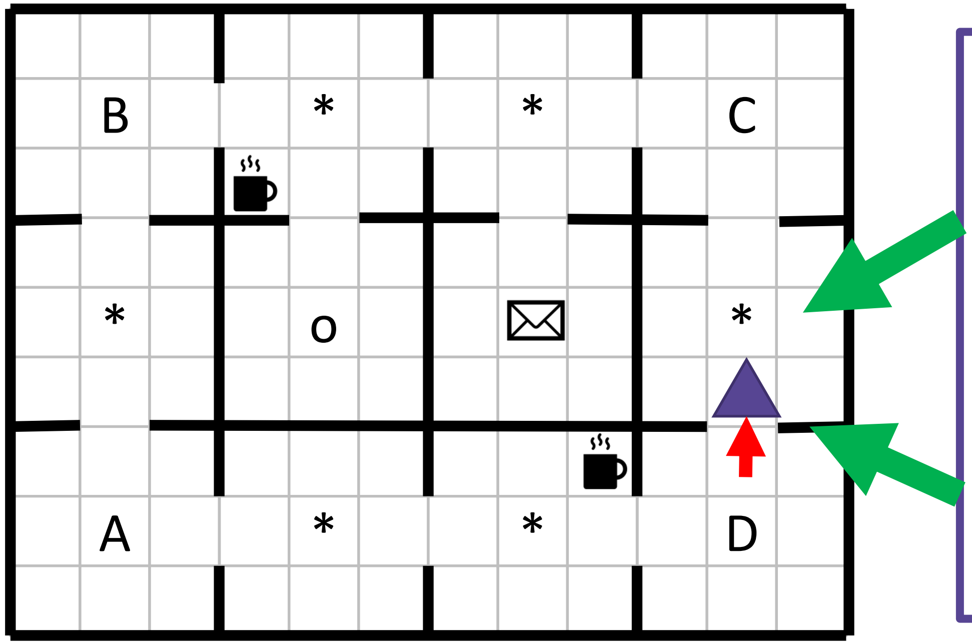
```
count = 0  # global variable

def get_reward(s):
    if count == 0 and state.at("A"):
        count = 1
    if count == 1 and state.at("B"):
        count = 2
    if count == 2 and state.at("C"):
        count = 3
    if count == 3 and state.at("D"):
        count = 0
        return 1
    return 0
```



Key Insight: Reveal Reward Function to the Agent

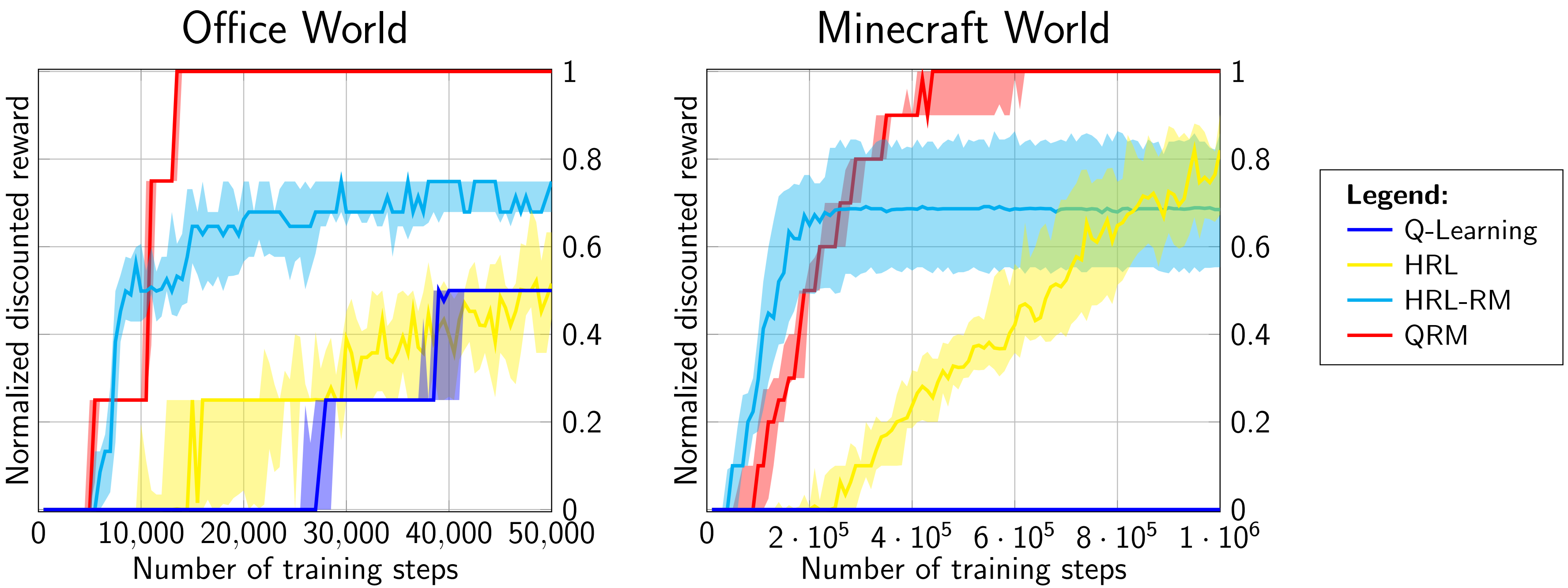


Key Insight: Reveal Reward Function to the Agent



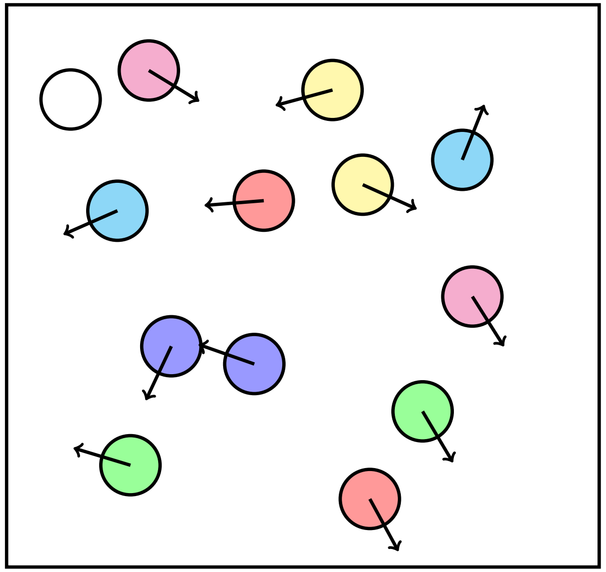

```
count = 0  # global variable

def get_reward(s):
    if count == 0 and state.at("A"):
        count = 1
    if count == 1 and state.at("B"):
        count = 2
    if count == 2 and state.at("C"):
        count = 3
    if count == 3 and state.at("D"):
        count = 0
        return 1
    return 0
```

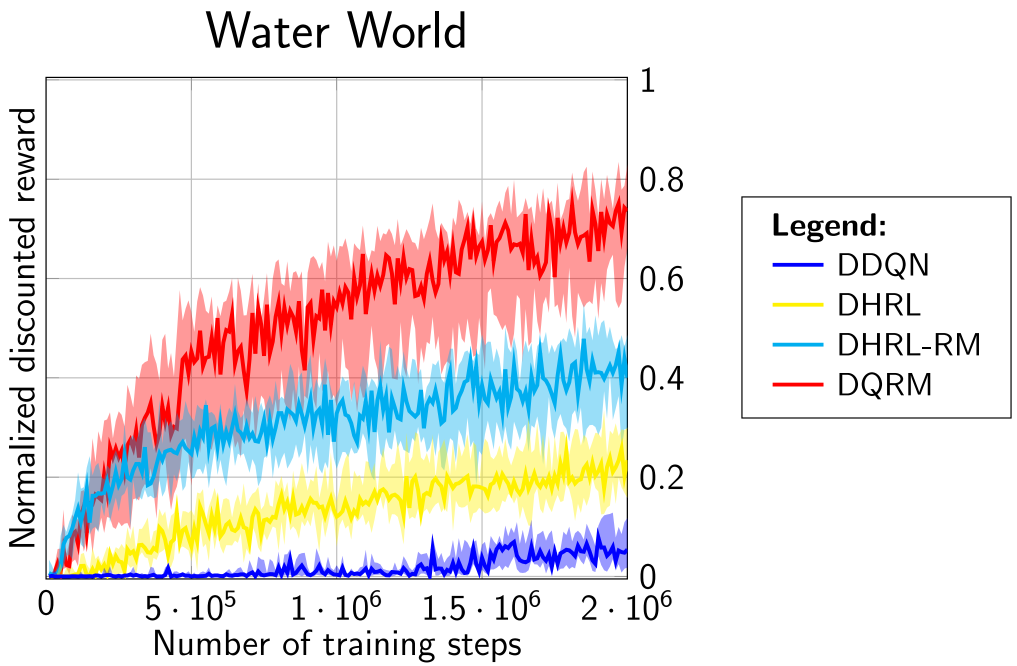
Great Results in Discrete Domains



QRM outperforms HRL and standard Q-learning in two domains

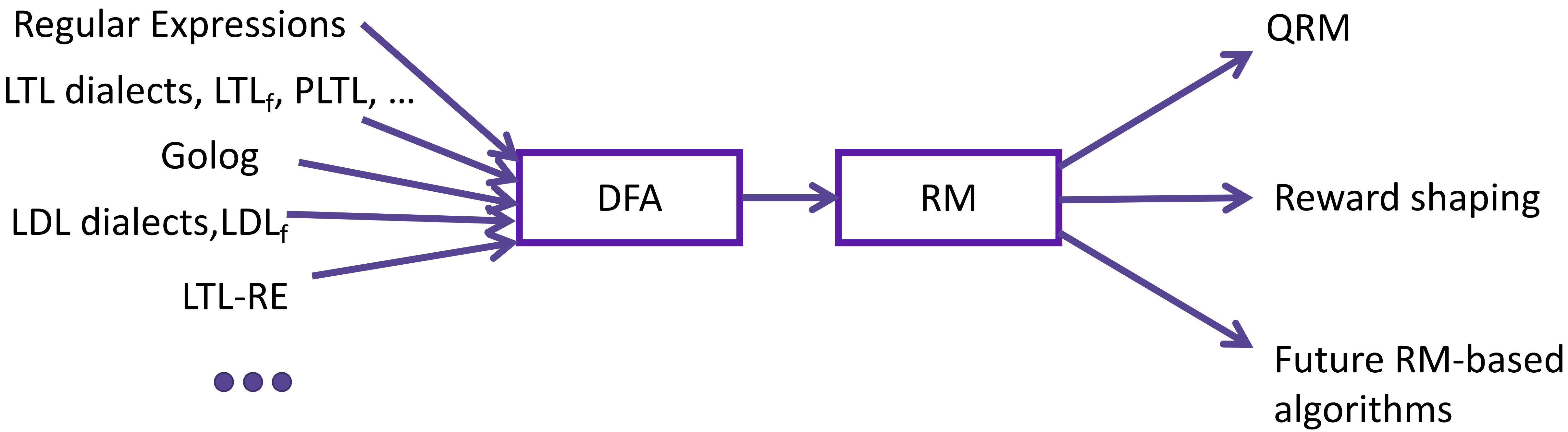
...and in Continuous Domains



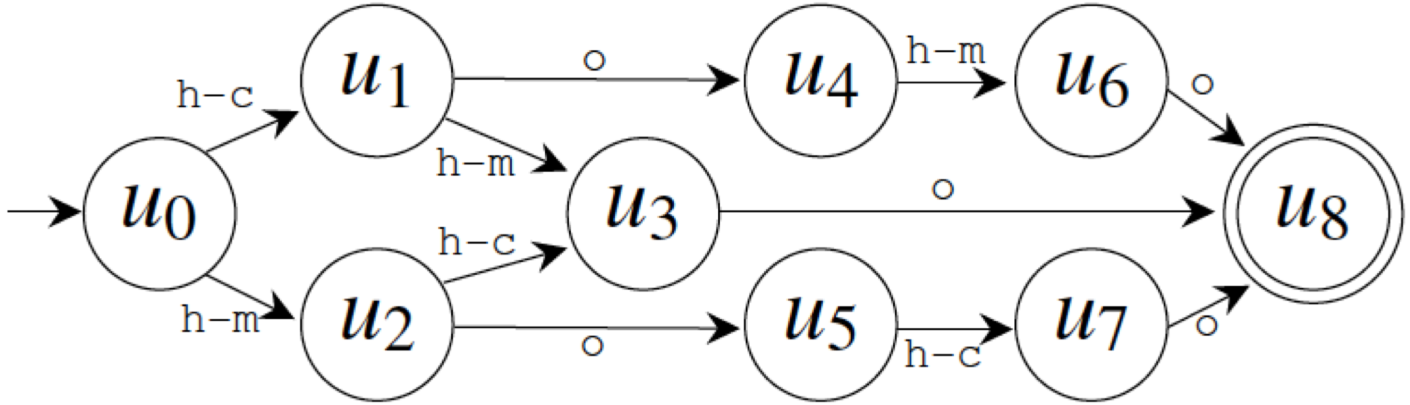


... and is also effective when combined with deep learning

We can construct RMs from a diversity of formal languages …

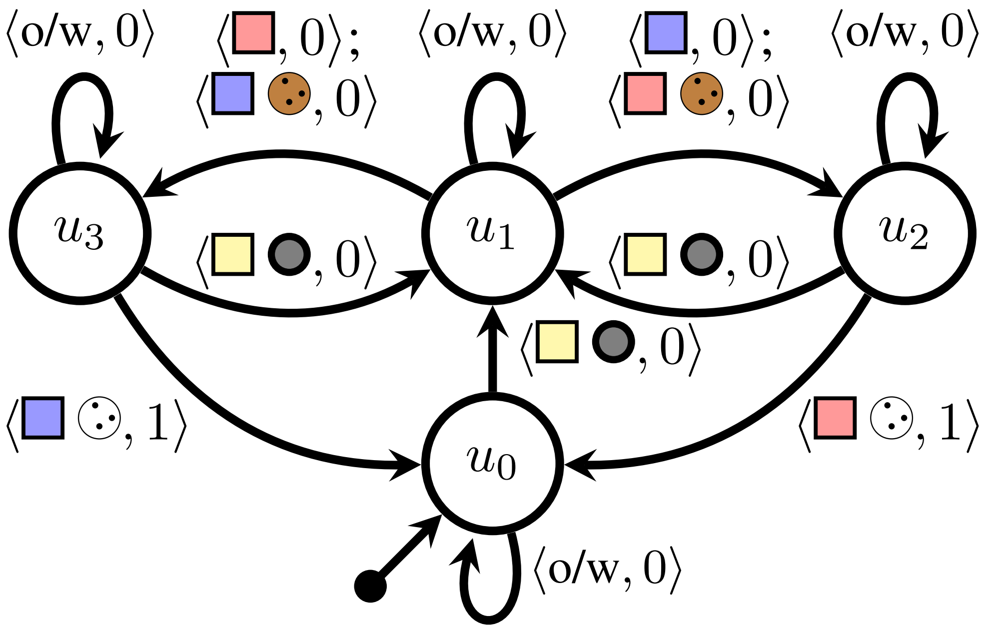


We can generate them using a Symbolic Planner



u_0: $\varnothing$
u_1: {get-coffee}
u_2: {get-mail}
u_3: {get-coffee, get-mail}
u_4: {get-coffee, deliver-coffee}
u_5: {get-mail, deliver-mail}
u_6: {get-coffee, get-mail, deliver-coffee}
u_7: {get-mail, get-coffee, deliver-mail}
u_8: {get-coffee, get-mail, deliver-coffee, deliver-mail}

…and they can be learned in partially observable environments to solve hard problems



Play with the code, read the papers, …

Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning
Toro Icarte, Klassen, Valenzano, McIlraith
ICML 2018
Code: https://bitbucket.org/RToroIcarte/qrm

Teaching Multiple Tasks to an RL Agent using LTL
Toro Icarte, Klassen, Valenzano, McIlraith
AAMAS 2018 & NeurIPS 2018 Workshop (Learning by Instructions)
Code: https://bitbucket.org/RToroIcarte/lpopl

LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning
Camacho, Toro Icarte, Klassen, Valenzano, McIlraith
IJCAI 2019

Learning Reward Machines for Partially Observable Reinforcement Learning
Toro Icarte, Waldie, Klassen, Valenzano, Castro, McIlraith
NeurIPS 2019

Symbolic Plans as High-Level Instructions for Reinforcement Learning
Illanes, Yan, Toro Icarte, McIlraith
ICAPS 2020/RLDM 2019

Latest Work

Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning
Toro Icarte, Klassen, Valenzano, McIlraith
Forthcoming
*An update of our original ICML 2018 Reward Machines paper. With **QRM replaced by CRM**.*
Code and paper available at http://www.cs.toronto.edu/~rntoro

Note this!

LTL2Action: Generalizing LTL Instructions for Multi-Task RL
Vaezipoor, Li, Toro Icarte, McIlraith
ICML 2021.
RL agent learns the language of LTL and how to follow and generalize instructions for multi-task RL.

New

Other related work

Advice-Based Exploration in Model-Based Reinforcement Learning.
Toro Icarte, Klassen, Valenzano, McIlraith
Canadian AI 2018.
Linear temporal logic (LTL) formulas and a heuristic were used to guide exploration during reinforcement learning.

Non-Markovian Rewards Expressed in LTL: Guiding Search Via Reward Shaping (Extended Version)
Camacho, Chen, Sanner, McIlraith
Extended Abstract: SoCS 2017, RLDM 2017
Full Paper: First Workshop on Goal Specifications for Reinforcement Learning, collocated with ICML/IJCAI/AAMAS, 2018.
Linear temporal logic (LTL) formulas are used to express non-Markovian reward in fully specified MDPs. LTL is translated to automata and reward shaping is used over the automata to help solve the MDP.

Learning Interpretable Models in Linear Temporal Logic
Camacho, McIlraith
ICAPS, 2019

FL-AT: A Formal Language–Automaton Transmogrifier.
Middleton, Klassen, Baier, McIlraith
ICAPS 2020 Systems Demo

Past work on Planning with Formal Languages & Automata

Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces
Camacho, Triantafillou, Muise, Baier and McIlraith
AAAI 2017

Planning with First-Order Temporally Extended Goals Using Heuristic Search
Baier and McIlraith AAAI 2006

Planning with Temporally Extended Goals Using Heuristic Search
Baier and McIlraith, ICAPS 2006

Exploiting Procedural Domain Control Knowledge in State-of-the-Art Planners
Baier Fritz and McIlraith, ICAPS 2007

Beyond Classical Planning: Procedural Control Knowledge and Preferences in State-of-the-Art Planners
Baier Fritz Bienvenu and McIlraith, AAAI 2008

A Heuristic Search Approach to Planning with Temporally Extended Preference
Baier, Bacchus and McIlraith Artificial Intelligence Journal, 2009

Specifying and Computing Preferred Plans
Fritz, Bienvenu and McIlraith, Artificial Intelligence Journal, 2011 (See also KR2006 paper)

...

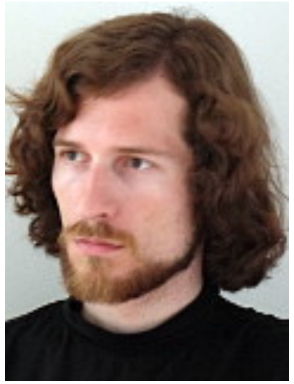
Past work on Planning with Formal Languages & Automata

For work on LTL FOND Planning, LTL Synthesis & their relationship
see work by Alberto Camacho

http://www.cs.toronto.edu/~acamacho/publications

Alberto Camacho

Acknowledgements



Toryn Klassen

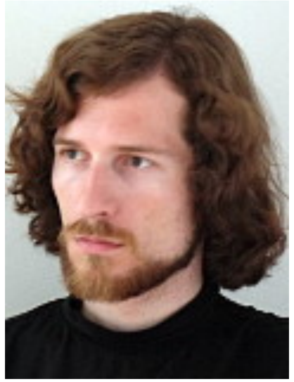
Richard Valenzano

Alberto Camacho

Rodrigo Toro Icarte

Acknowledgements

Rodrigo Toro Icarte



Toryn Klassen

Richard Valenzano

Alberto Camacho

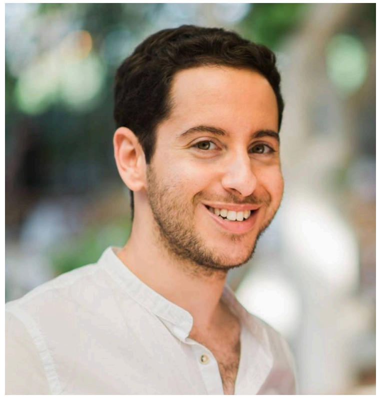
Léon Illanes

Ethan Waldie

Margarita Castro

Andrew Li

Pashootan Vaezipoor



Maayan Shvo

Phillip Christoffersen

Xi Yan