Reward Machines:

Structuring reward function specifications and reducing sample complexity in reinforcement learning

Sheila A. McIlraith
Department of Computer Science
University of Toronto

MSR Reinforcement Learning Day 2019
New York, NY
October 3, 2019

Computer Science
UNIVERSITY OF TORONTO

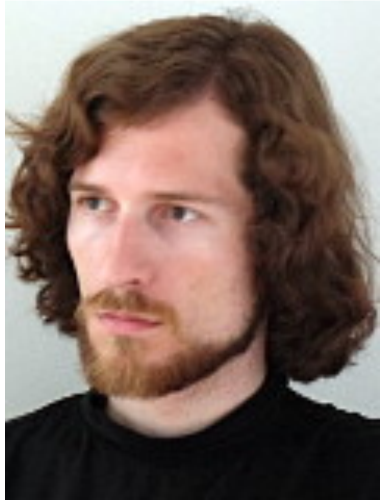
VECTOR INSTITUTE | INSTITUT VECTEUR

Acknowledgements

Rodrigo Toro Icarte

Acknowledgements

Rodrigo Toro Icarte



Toryn Klassen

Richard Valenzano

ELEMENT AI

Alberto Camacho

Ethan Waldie

Margarita Castro

LANGUAGE

LANGUAGE

Humans have evolved languages over thousands of years to provide useful abstractions for understanding and interacting with each other and with the physical world.

The claim advanced by some is that language influences what we think, what we perceive, how we focus our attention, and what we remember.

While psychologist continue to debate how (and whether) language shapes the way we think, there is some agreement that the alphabet and structure of a language can have a significant impact on learning and reasoning.

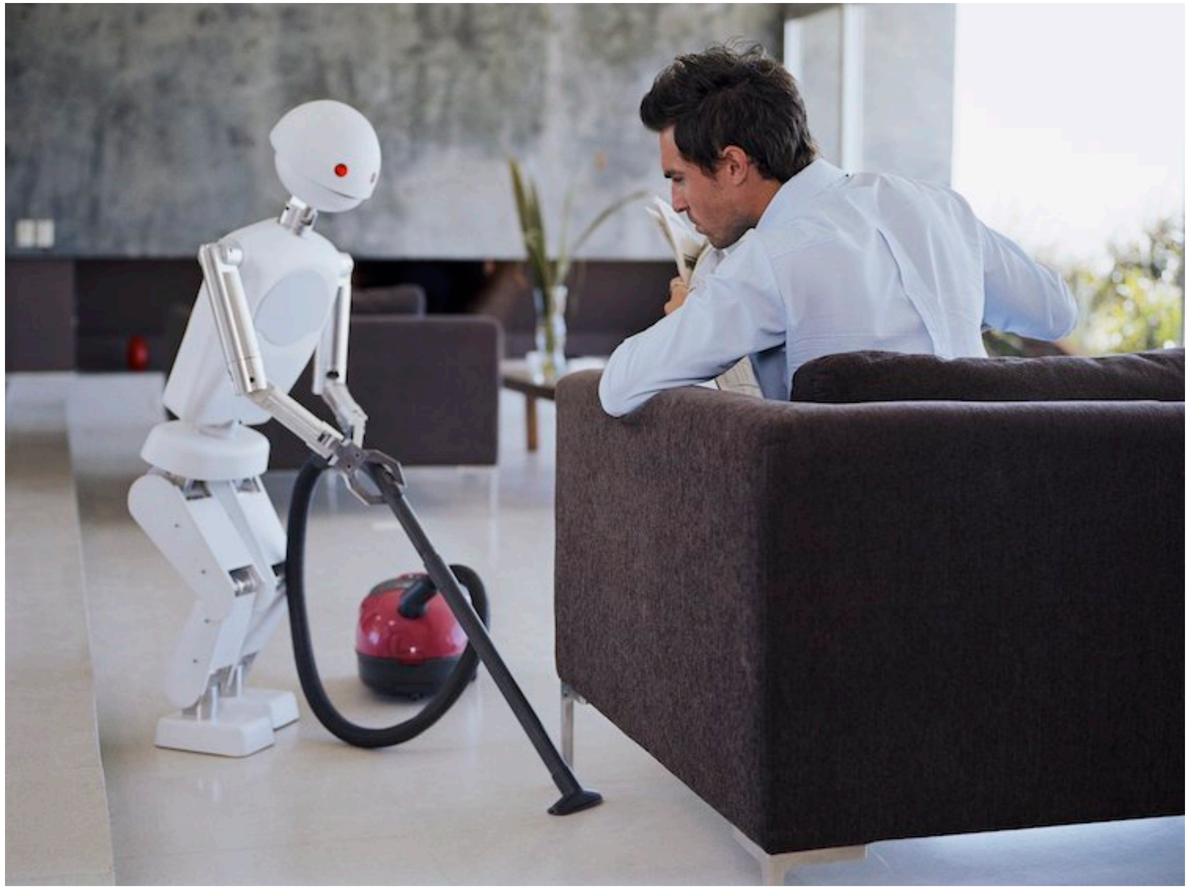
LANGUAGE

We use language to capture our understanding of the world around us, to communicate high-level goals, intentions and objectives,, and to support coordination with others.

We also use language to teach – to transfer knowledge.

Importantly, language can provide us with useful and purposeful abstractions that can help us to generalize and transfer knowledge to new situations.

Can exploiting the alphabet and structure of language help RL agents learn and think?



How do we advise, instruct, task, … and impart knowledge to our RL agents?

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.

Goals and Preferences

- When getting ice cream, please always open the freezer, take out the ice cream, serve yourself, put the ice cream back in the freezer, and close the freezer door.

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

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

Properties

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

Remember this!

Goals and Preferences

- Do not vacuum while someone is sleeping

$$\textbf{always}[\neg\,(\textbf{vacuum} \wedge \textbf{sleeping})]$$

Goals and Preferences

- Do not vacuum while someone is sleeping

always[¬ (vacuum ∧ sleeping)]

- When getting an ice cream for someone …

always[get(ice-cream) ->
eventually [open(freezer) ∧
next[remove(ice-cream,freezer) ∧
next[serve(ice-cream) ∧
next[replace(ice-cream,freezer) ∧
next[close(freezer)]]]]]]

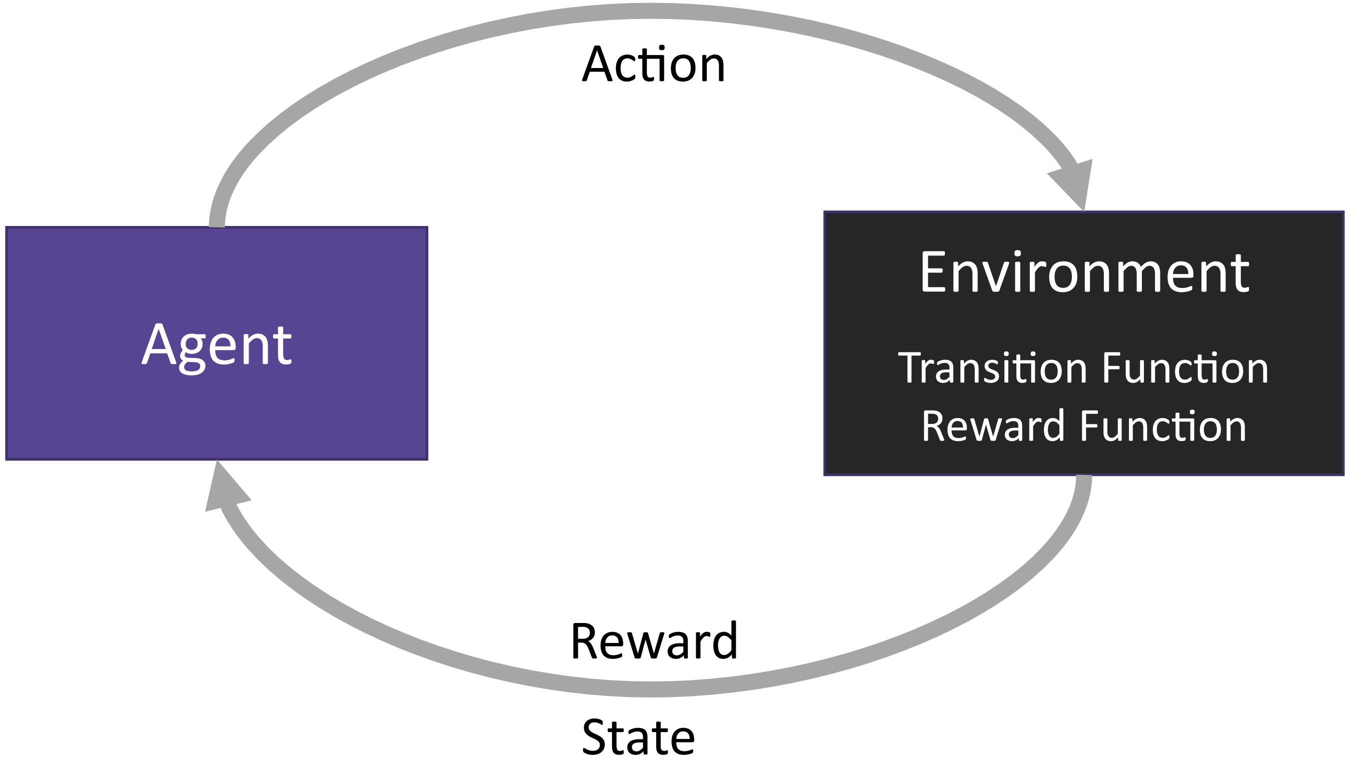
How do we communicate this to our RL agent?

MOTIVATION

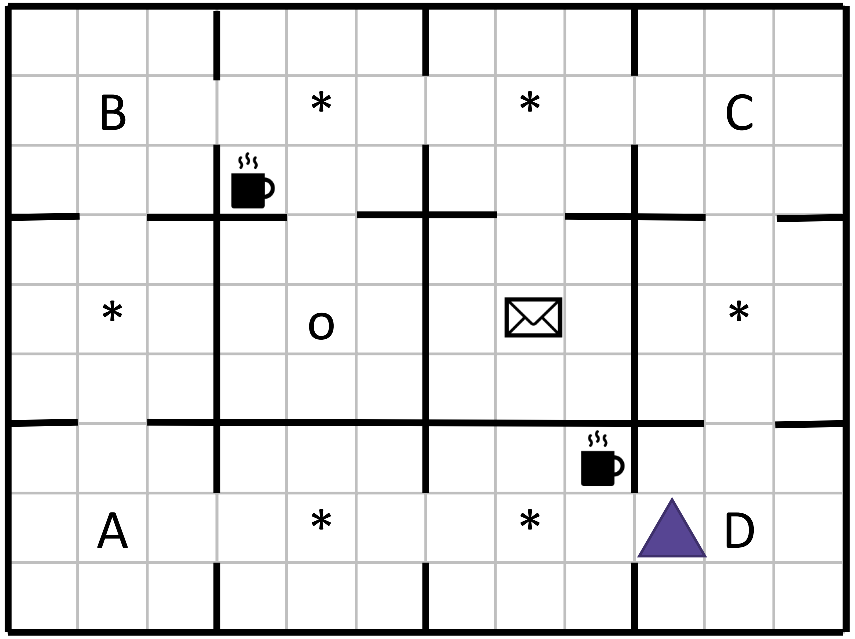
Challenges to RL

- **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.

Reinforcement Learning



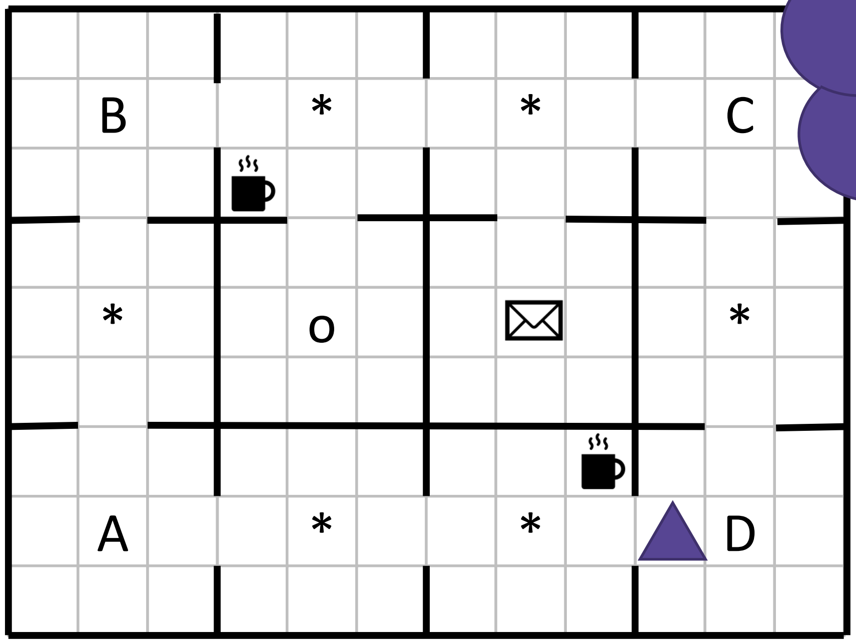
Running Example



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.

Toy Problem Disclaimer

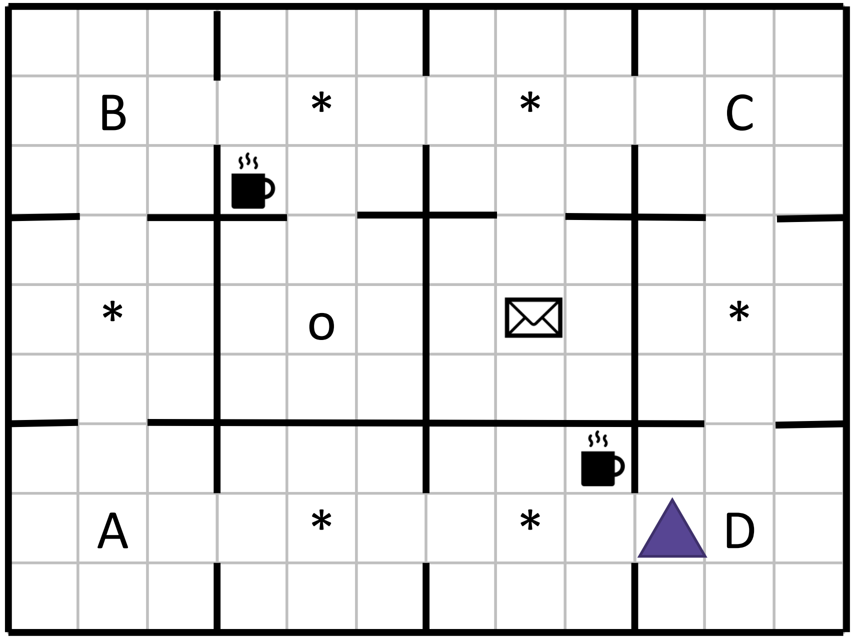


These "toy problems" challenge state-of-the-art RL techniques

	aning
	Agent
	Furniture
☕	Coffee Machine
✉	Mail Room
o	Office
A, B, C, D	Marked Locations

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

Running Example

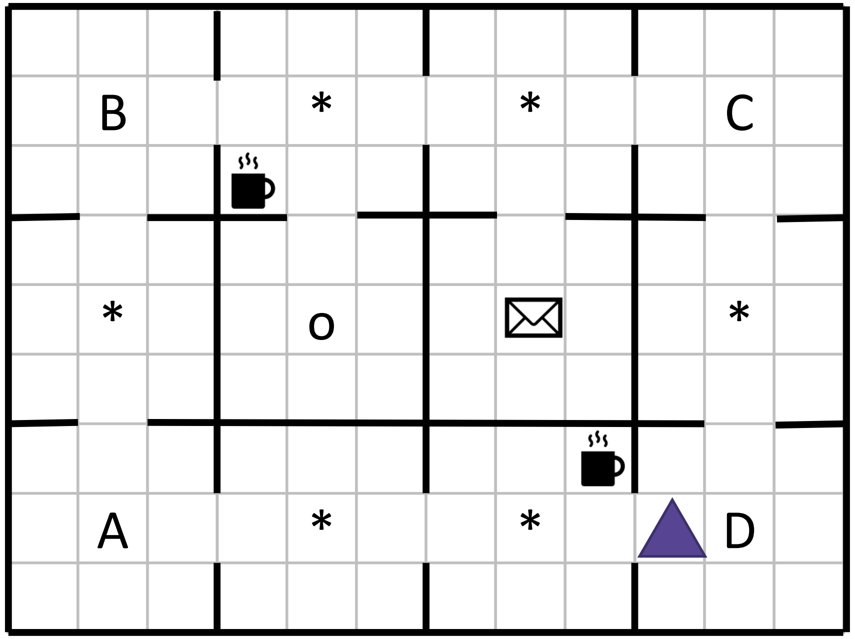


```
count = 0  # global variable

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
```

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

Observation: Someone always has to program the reward function … even when the environment is the real world!

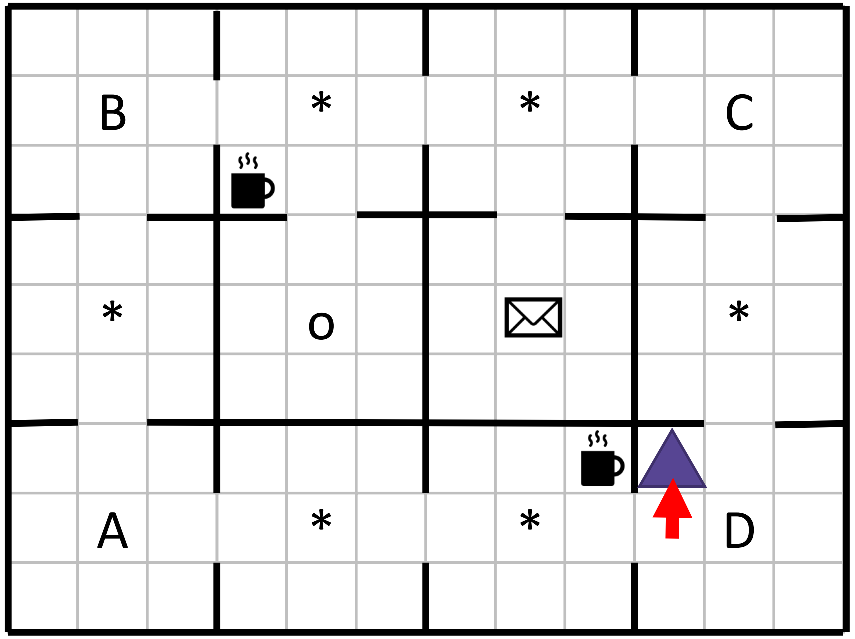
Running Example



Reward Function
(as part of environment)

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

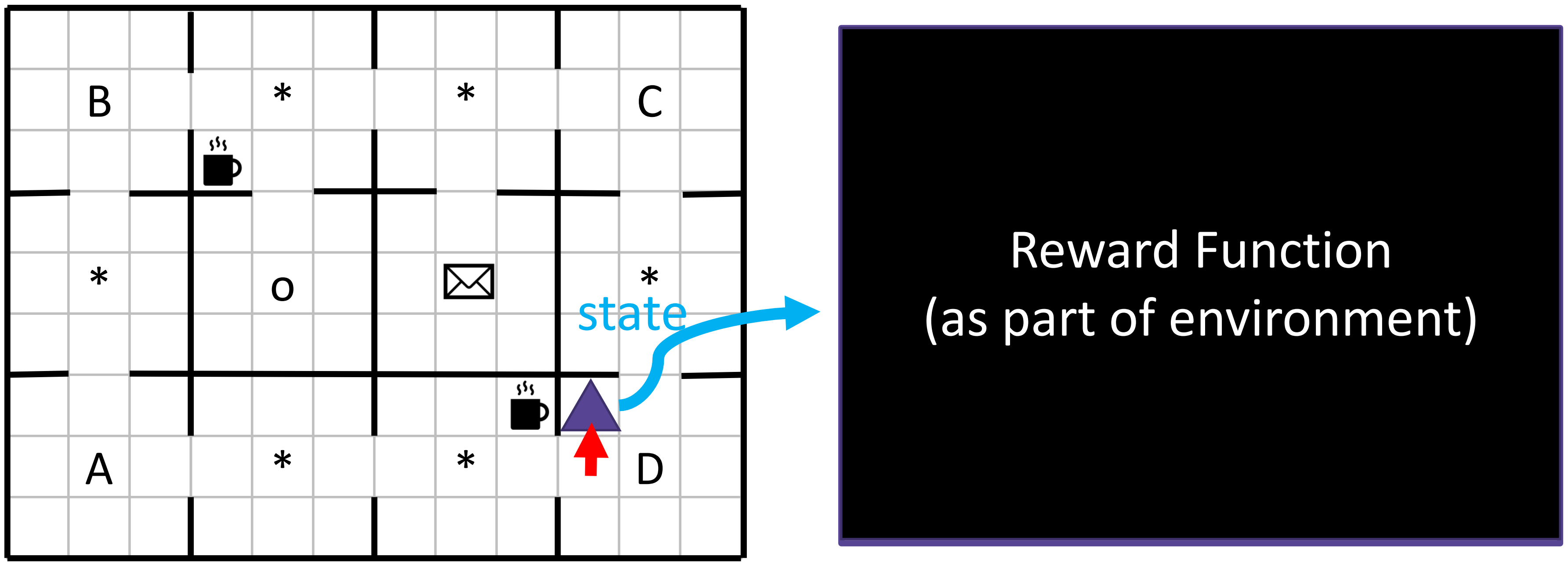
Running Example



Reward Function
(as part of environment)

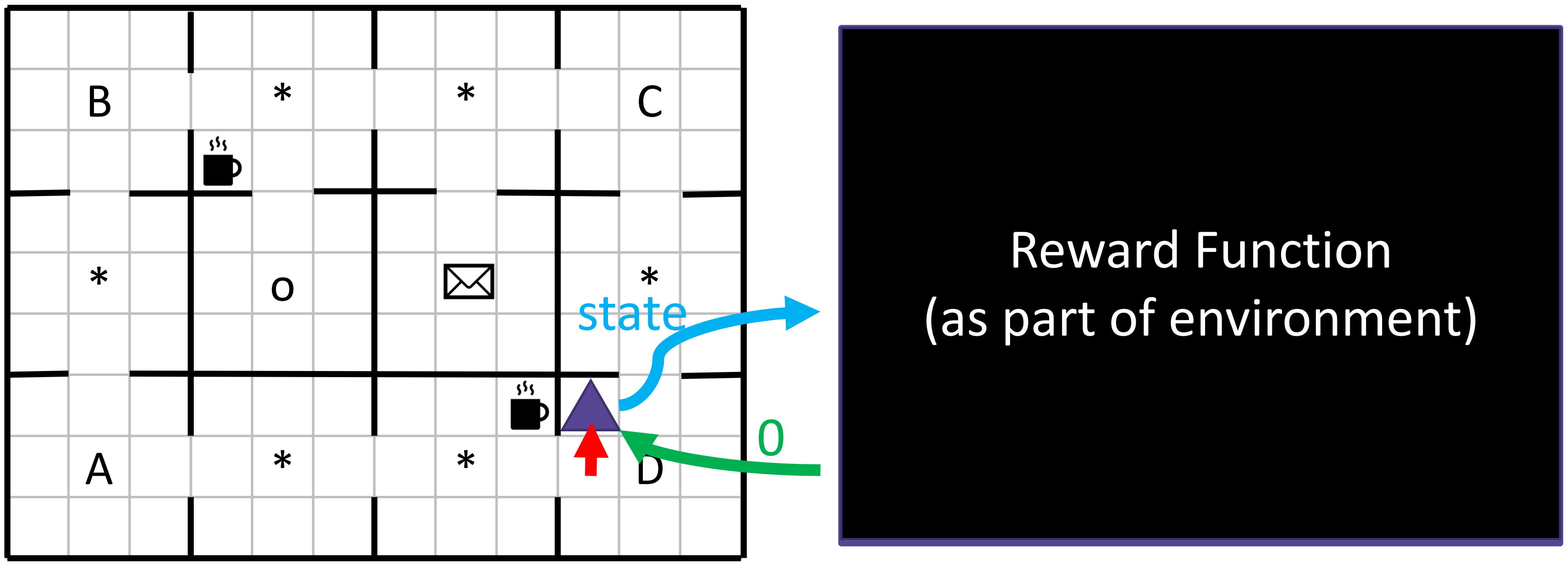
Task: Visit A, B, C, and D, in order.

Running Example



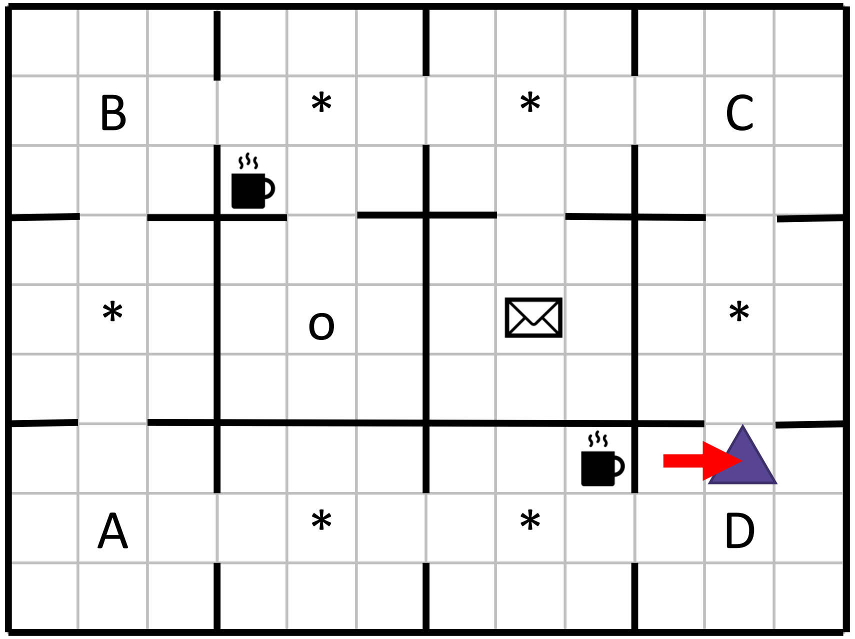
Task: Visit A, B, C, and D, in order.

Running Example



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

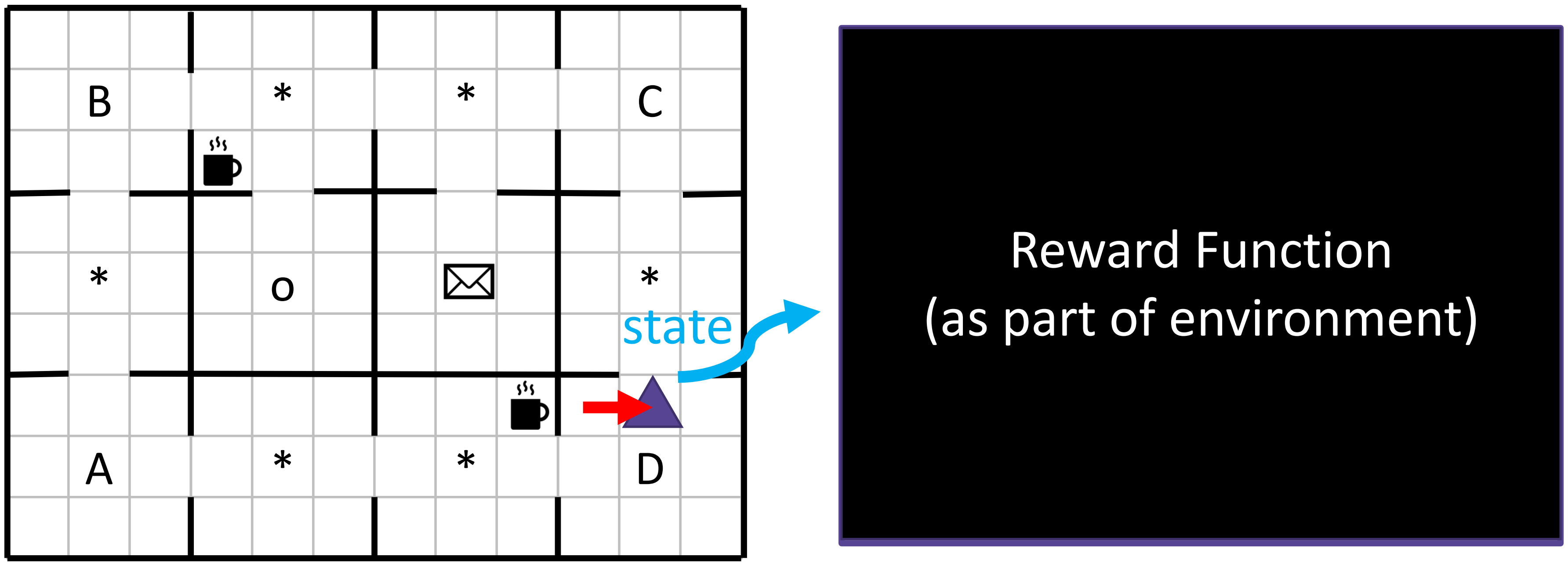
Running Example



Reward Function
(as part of environment)

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

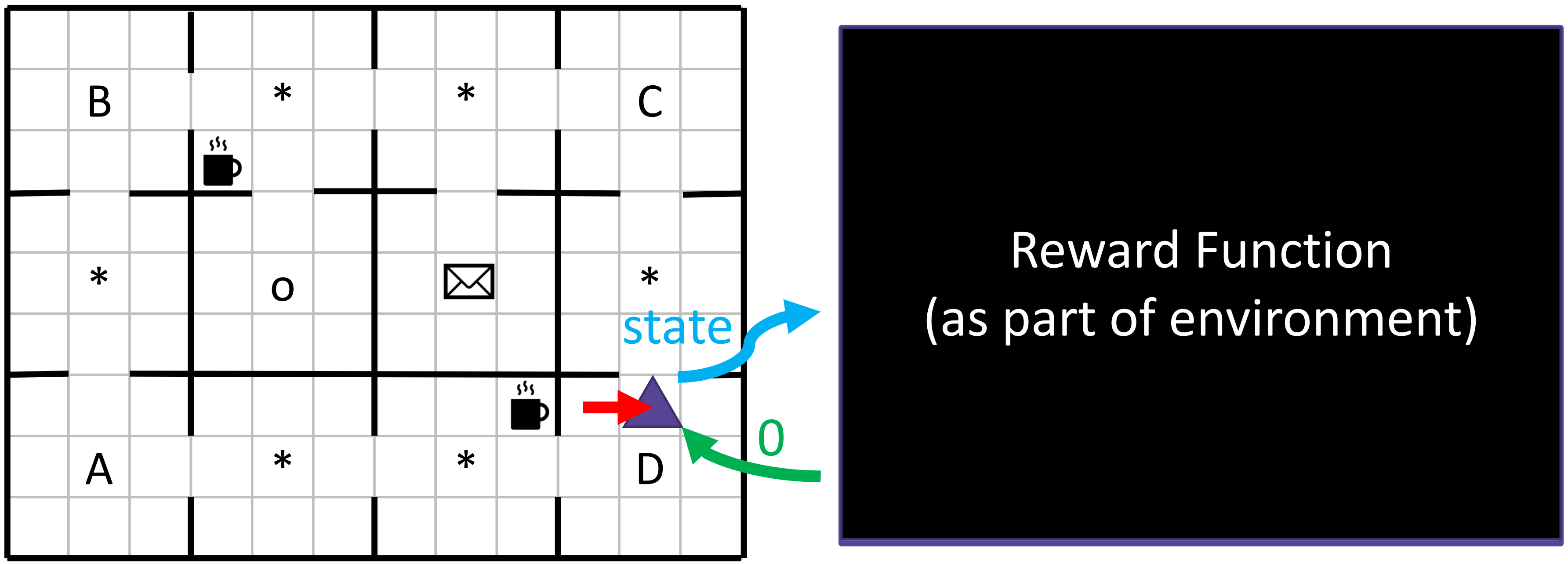
Running Example



Reward Function
(as part of environment)

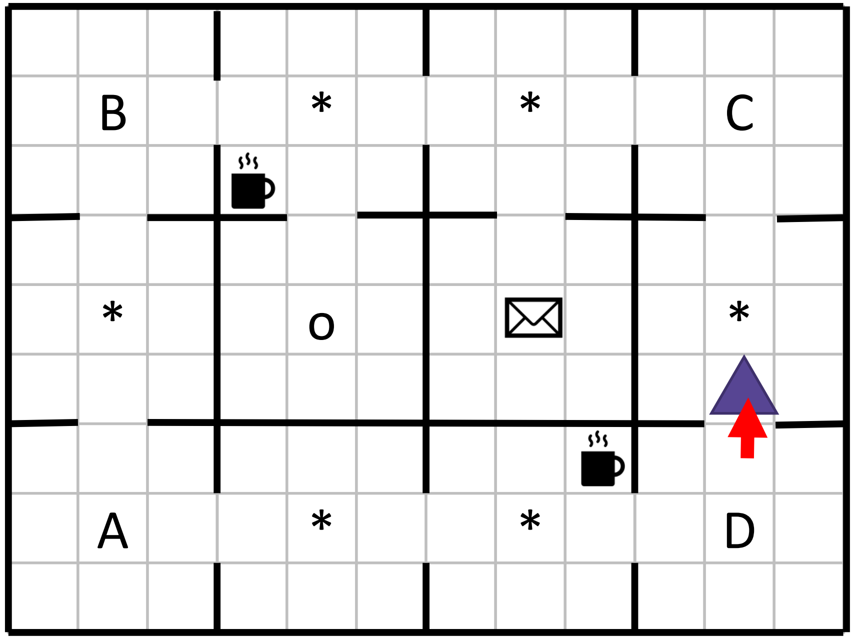
Task: Visit A, B, C, and D, in order.

Running Example



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

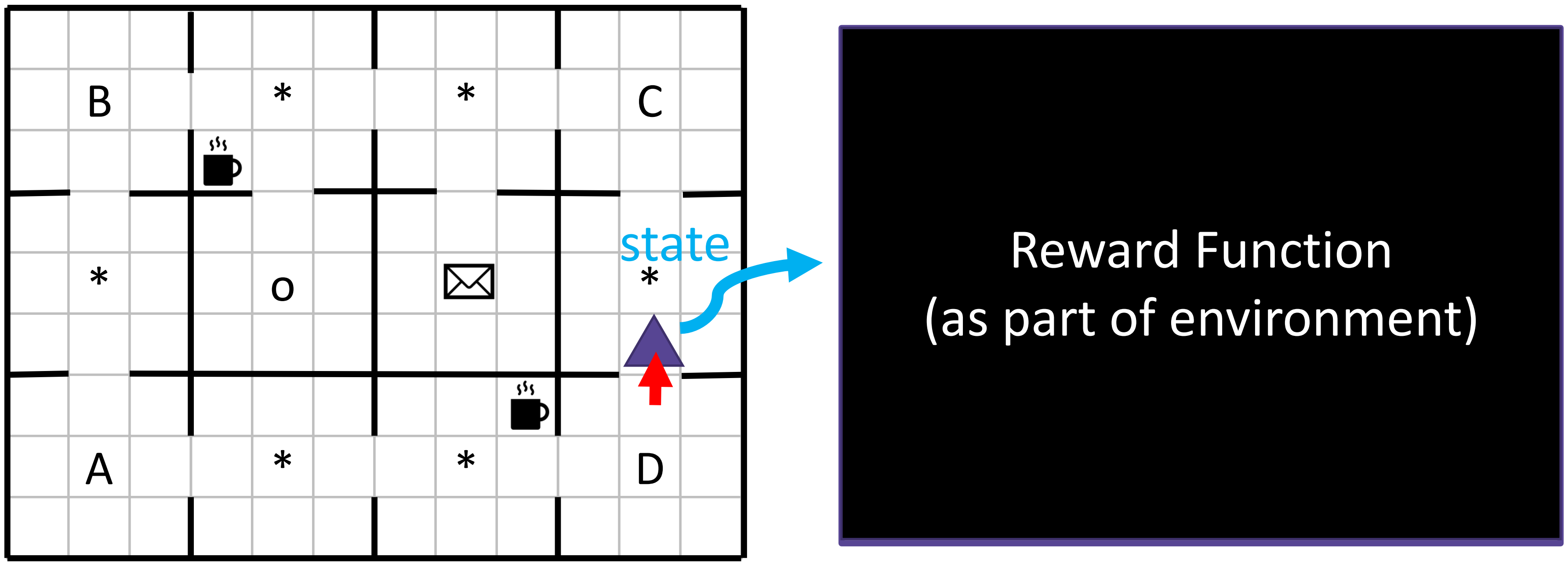
Running Example



Reward Function
(as part of environment)

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

Running Example

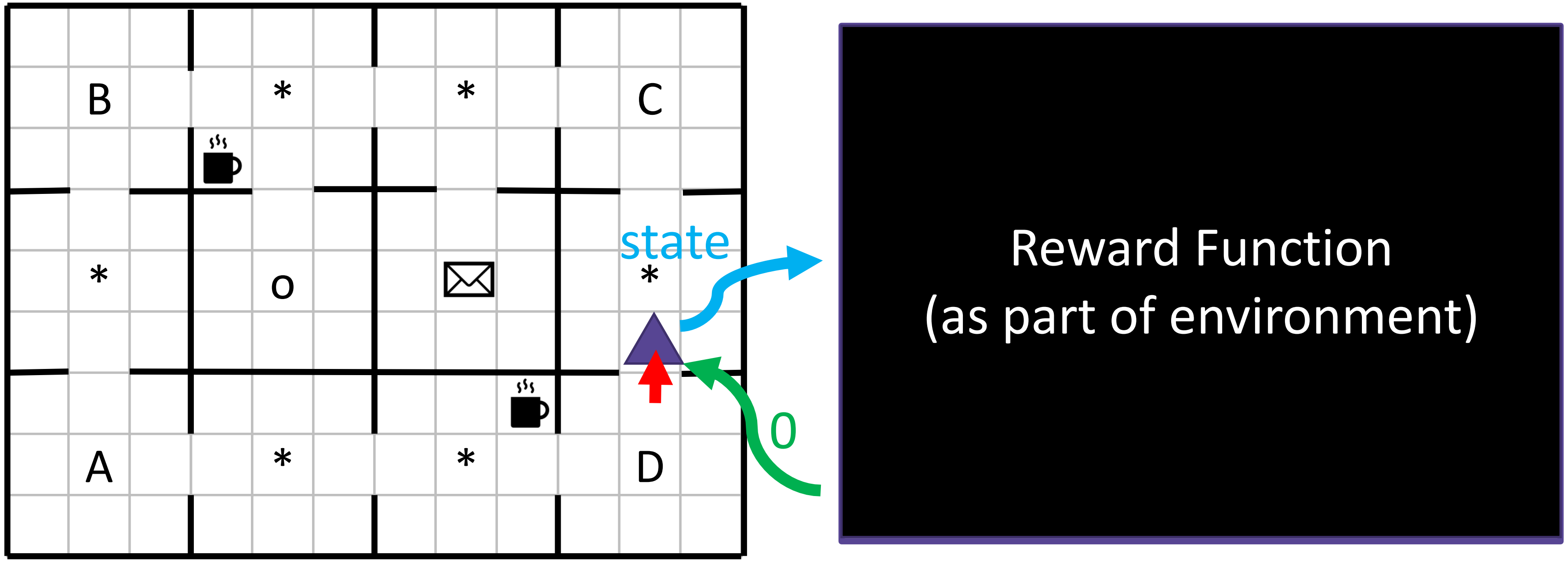


state

Reward Function
(as part of environment)

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

Running Example



Reward Function
(as part of environment)

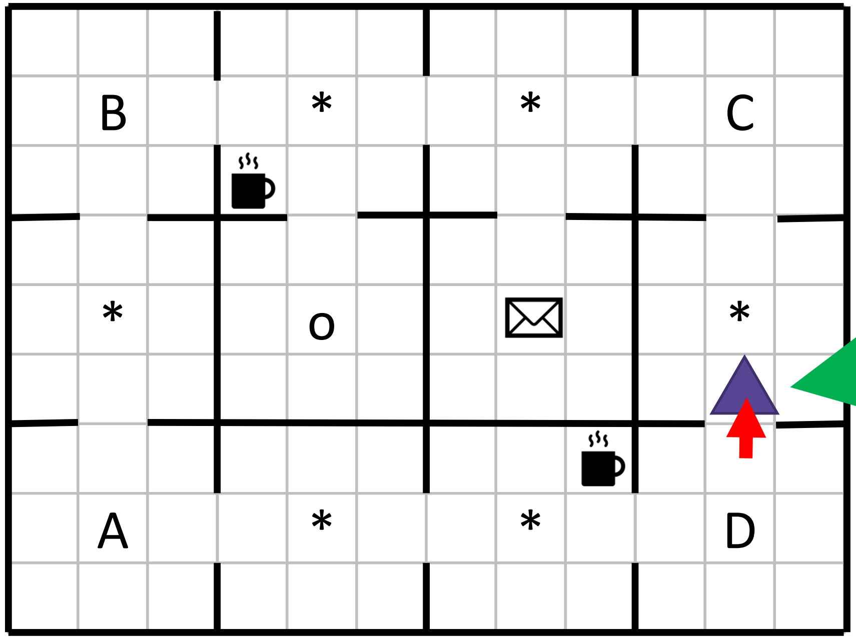
Task: Visit A, B, C, and D, in order.

Remember this!

Simple Idea:

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

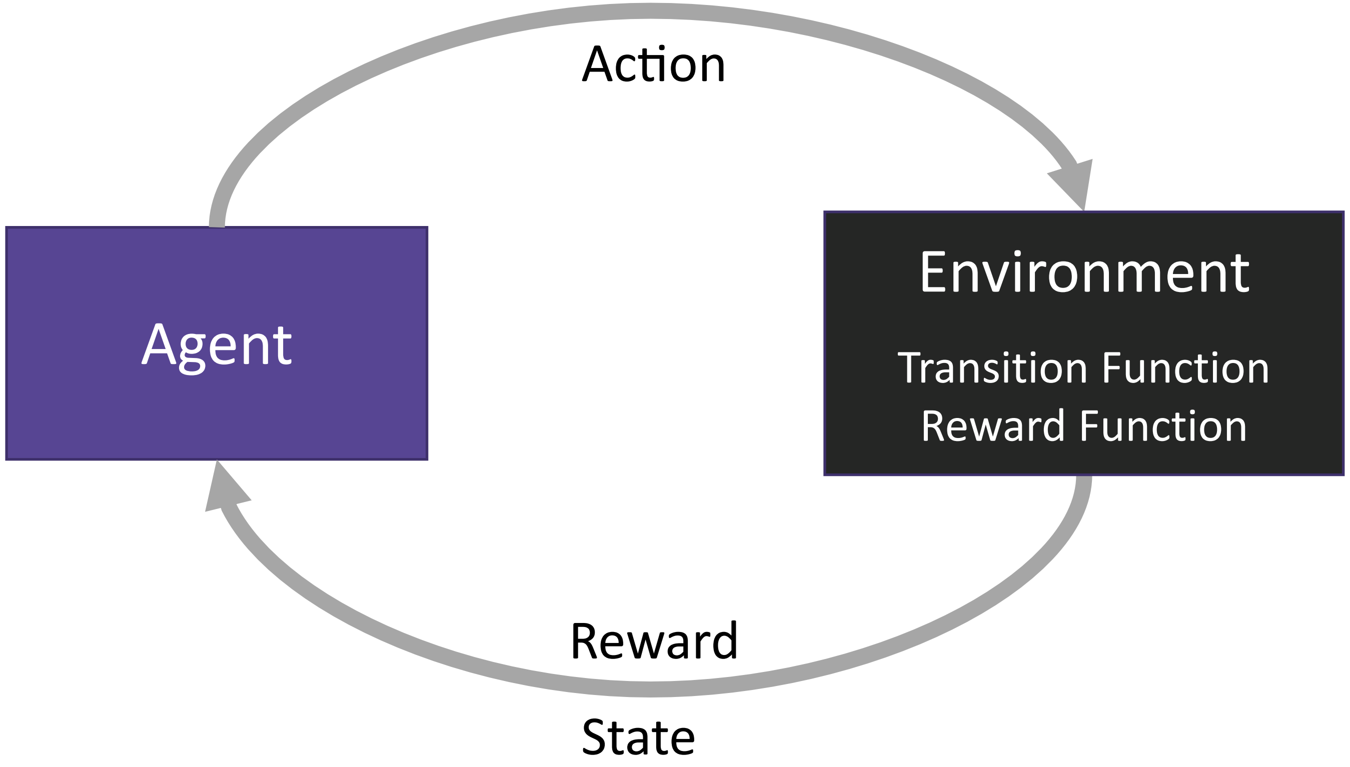
Running Example



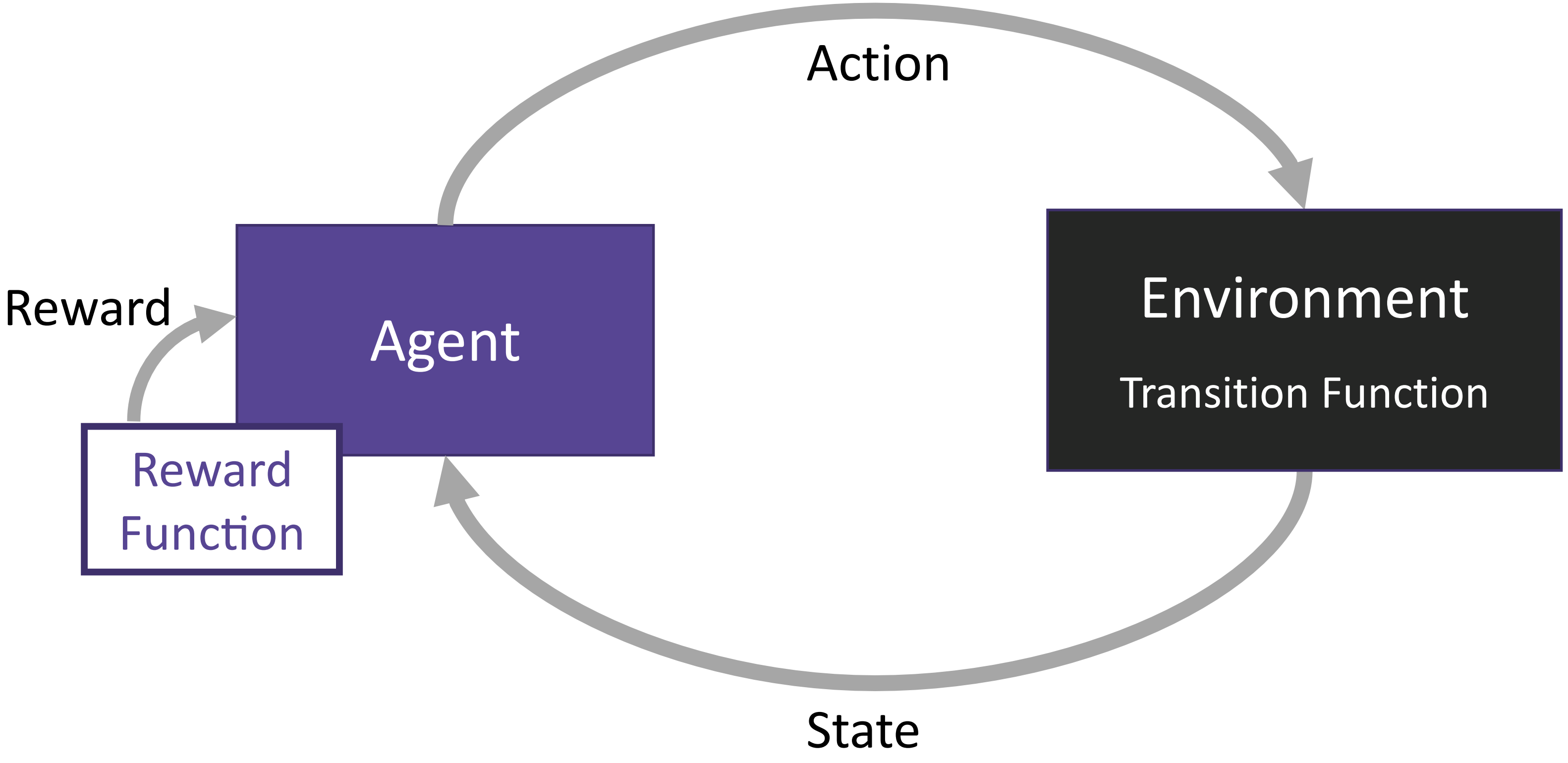
```
count = 0  # global variable

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.

Decoupling Transition and Reward Functions



Decoupling Transition and Reward Functions



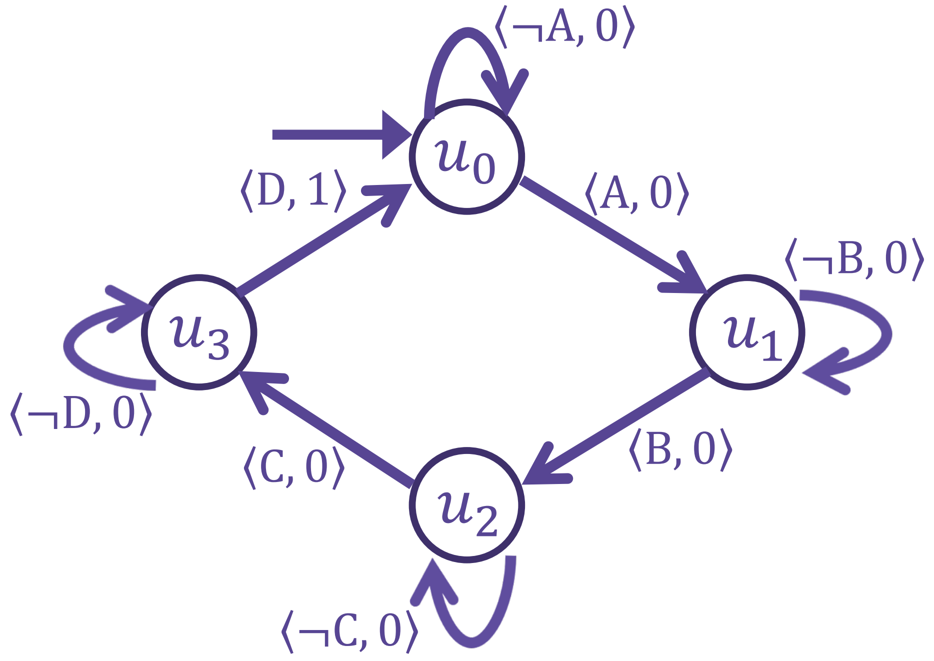
The Rest of the Talk

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

REWARD MACHINES

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
```

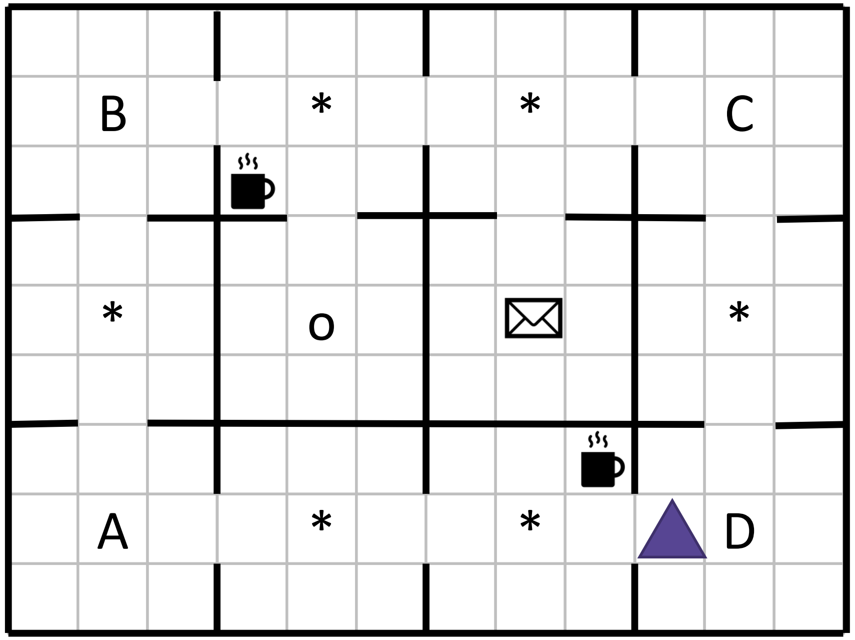


$\langle \neg A, 0 \rangle$
u_0
$\langle D, 1 \rangle$
$\langle A, 0 \rangle$
$\langle \neg B, 0 \rangle$
u_3
u_1
$\langle \neg D, 0 \rangle$
$\langle C, 0 \rangle$
$\langle B, 0 \rangle$
u_2
$\langle \neg C, 0 \rangle$

Encode reward function in an automata-like structure

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

Reward Function Vocabulary

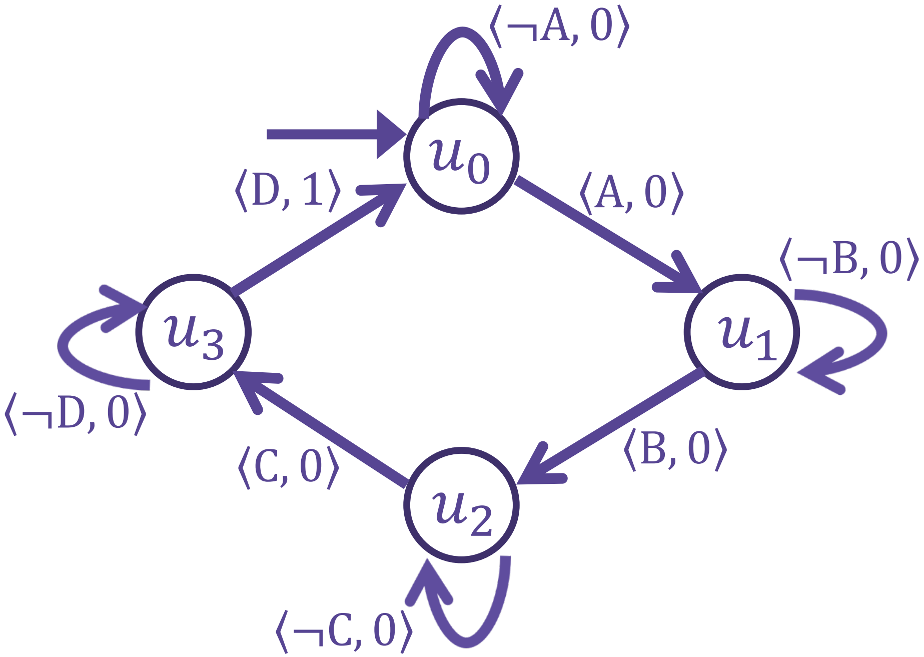


Symbol	Meaning
▲	Agent
*	Furniture
☕	Coffee Machine
✉	Mail Room
o	Office
A, B, C, D	Marked Locations

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

Reward Machine

Reward Machine

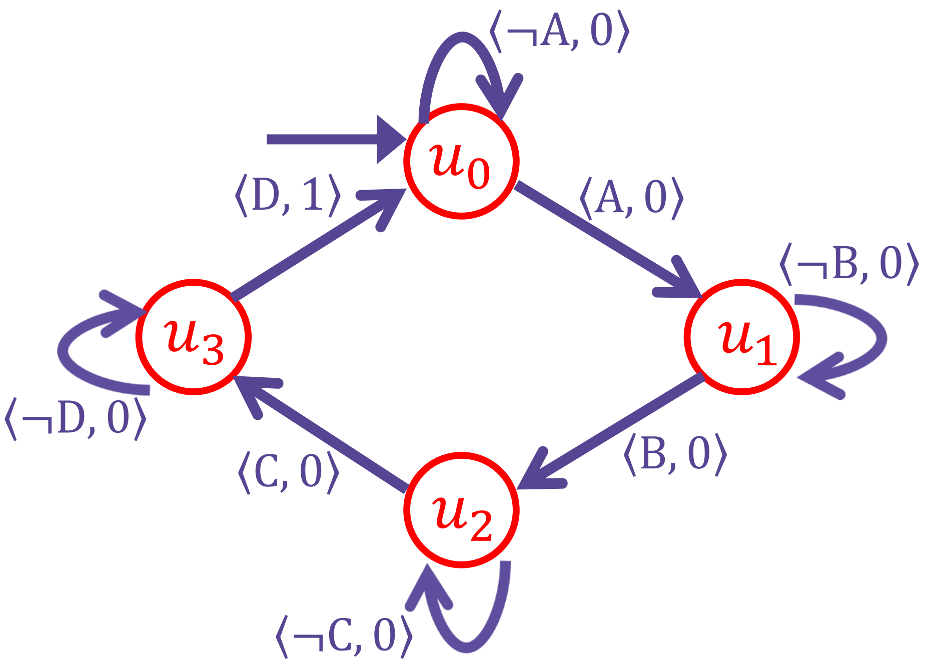


$\langle \neg A, 0\rangle$
u_0
$\langle D, 1\rangle$
$\langle A, 0\rangle$
$\langle \neg B, 0\rangle$
u_3
u_1
$\langle \neg D, 0\rangle$
$\langle C, 0\rangle$
$\langle B, 0\rangle$
u_2
$\langle \neg C, 0\rangle$

Reward Machine

Reward Machine

- finite set of states U

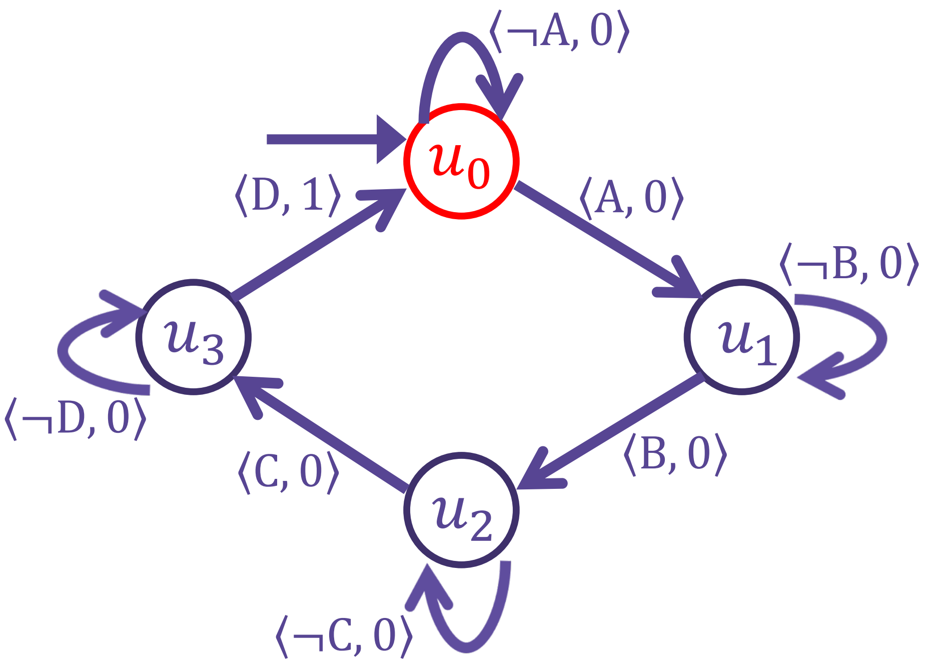


u_0
u_1
u_2
u_3
$\langle \neg A, 0 \rangle$
$\langle A, 0 \rangle$
$\langle \neg B, 0 \rangle$
$\langle B, 0 \rangle$
$\langle \neg C, 0 \rangle$
$\langle C, 0 \rangle$
$\langle \neg D, 0 \rangle$
$\langle D, 1 \rangle$

Reward Machine

Reward Machine

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

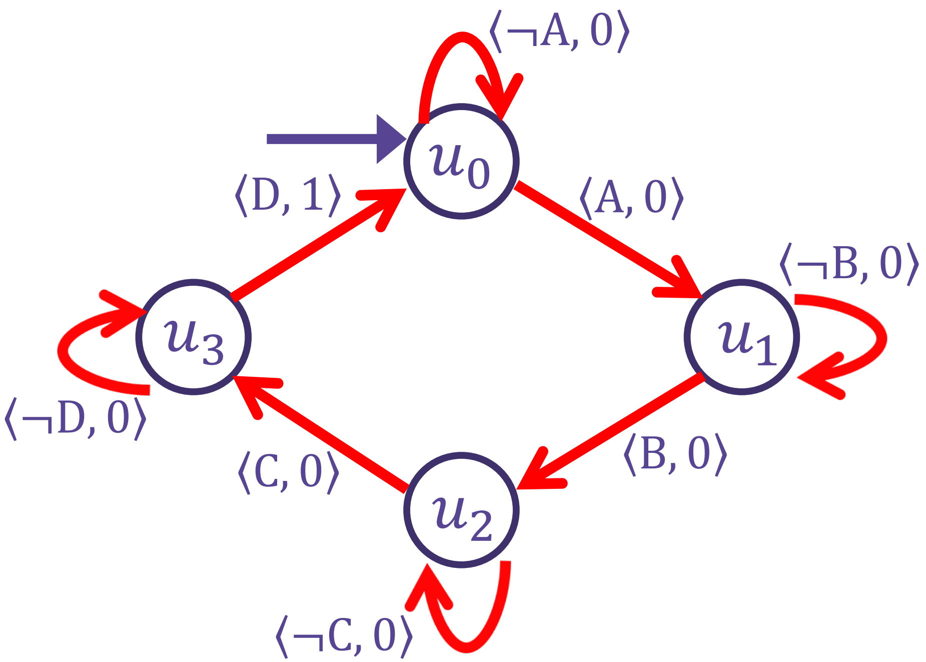


$\langle \neg A, 0 \rangle$
u_0
$\langle D, 1 \rangle$
$\langle A, 0 \rangle$
$\langle \neg B, 0 \rangle$
u_3
u_1
$\langle \neg D, 0 \rangle$
$\langle C, 0 \rangle$
$\langle B, 0 \rangle$
u_2
$\langle \neg C, 0 \rangle$

Reward Machine

Reward Machine

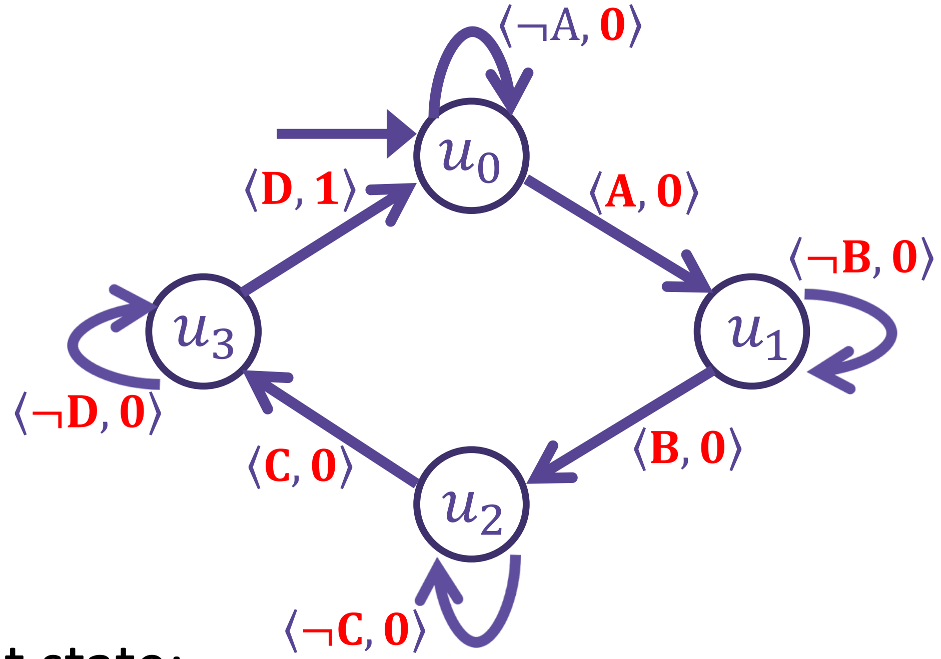
- finite set of states U
- initial state $u_0 \in U$
- set of transitions labelled by:


$\langle \neg A, 0\rangle$
u_0
$\langle D, 1\rangle$
$\langle A, 0\rangle$
$\langle \neg B, 0\rangle$
u_3
u_1
$\langle \neg D, 0\rangle$
$\langle C, 0\rangle$
$\langle B, 0\rangle$
u_2
$\langle \neg C, 0\rangle$

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)



$\langle \neg A, 0 \rangle$
u_0
$\langle A, 0 \rangle$
$\langle D, 1 \rangle$
$\langle \neg B, 0 \rangle$
u_3
u_1
$\langle \neg D, 0 \rangle$
$\langle C, 0 \rangle$
$\langle B, 0 \rangle$
u_2
$\langle \neg C, 0 \rangle$

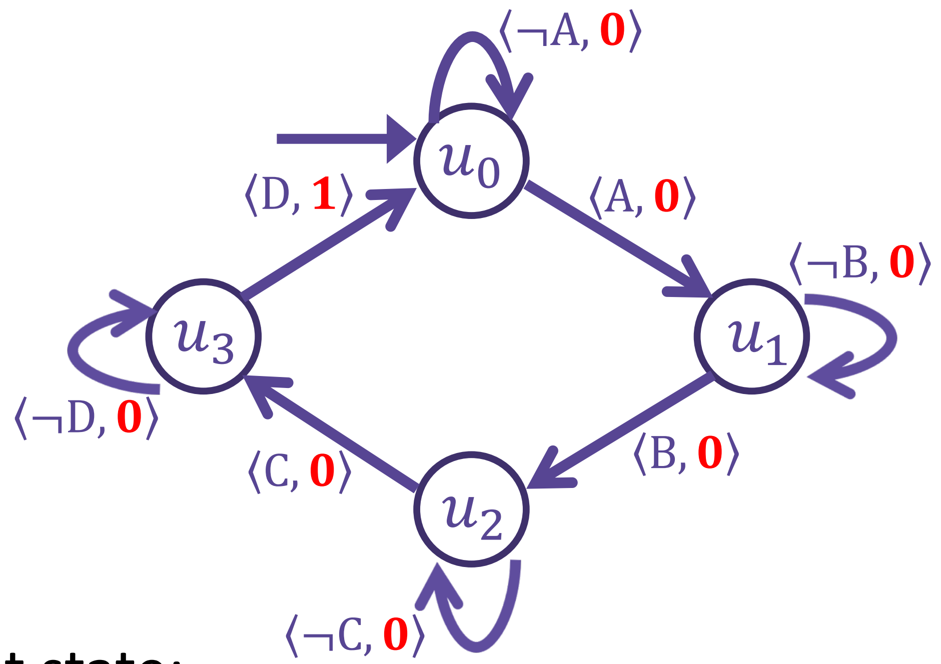
Conditions are over properties of the current state:

$$P = \{☕, ✉, o, *, A, B, C, D\}$$

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)



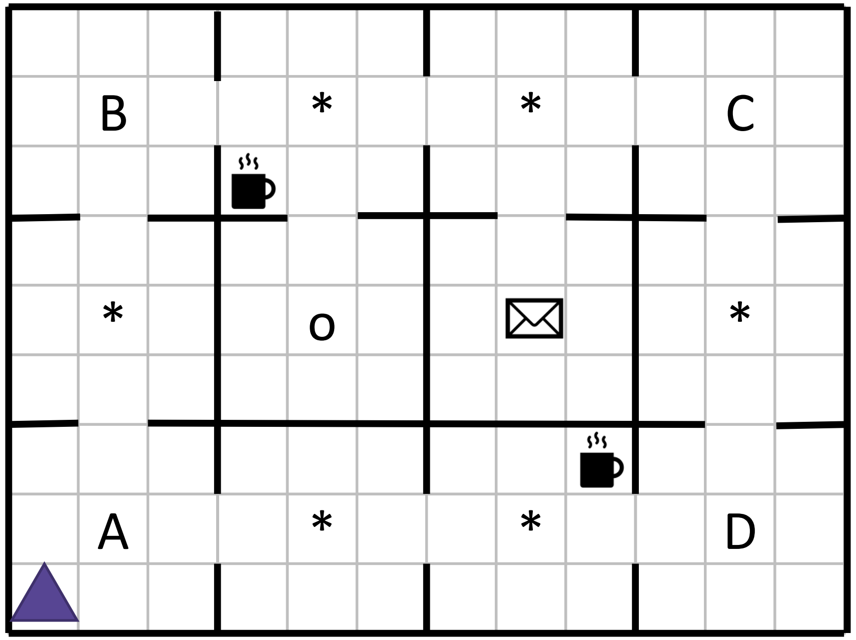
$\langle \neg A, \mathbf{0} \rangle$
u_0
$\langle D, \mathbf{1} \rangle$
$\langle A, \mathbf{0} \rangle$
$\langle \neg B, \mathbf{0} \rangle$
u_3
u_1
$\langle \neg D, \mathbf{0} \rangle$
$\langle B, \mathbf{0} \rangle$
$\langle C, \mathbf{0} \rangle$
u_2
$\langle \neg C, \mathbf{0} \rangle$

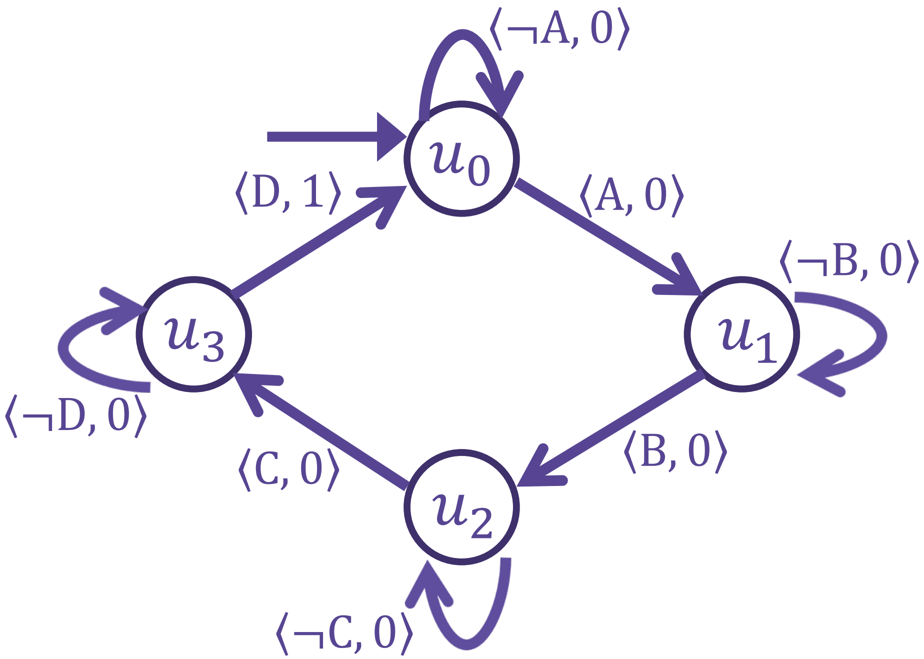
Conditions are over properties of the current state:

$P = \{$☕, ✉, $o, *, A, B, C, D\}$

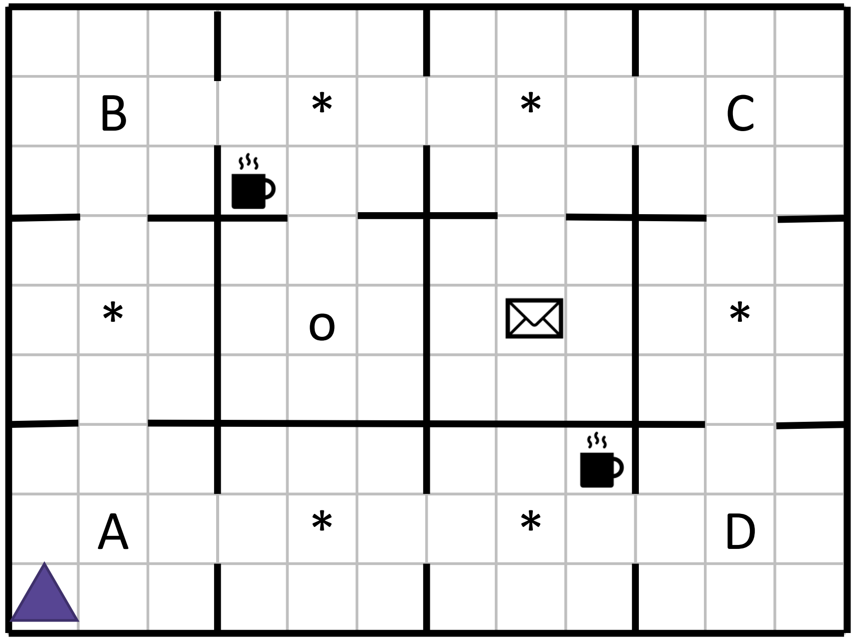
A Reward Machine is a **Mealy Machine** over the input alphabet $\Sigma = 2^P$, whose output alphabet is a set of Markovian reward functions.

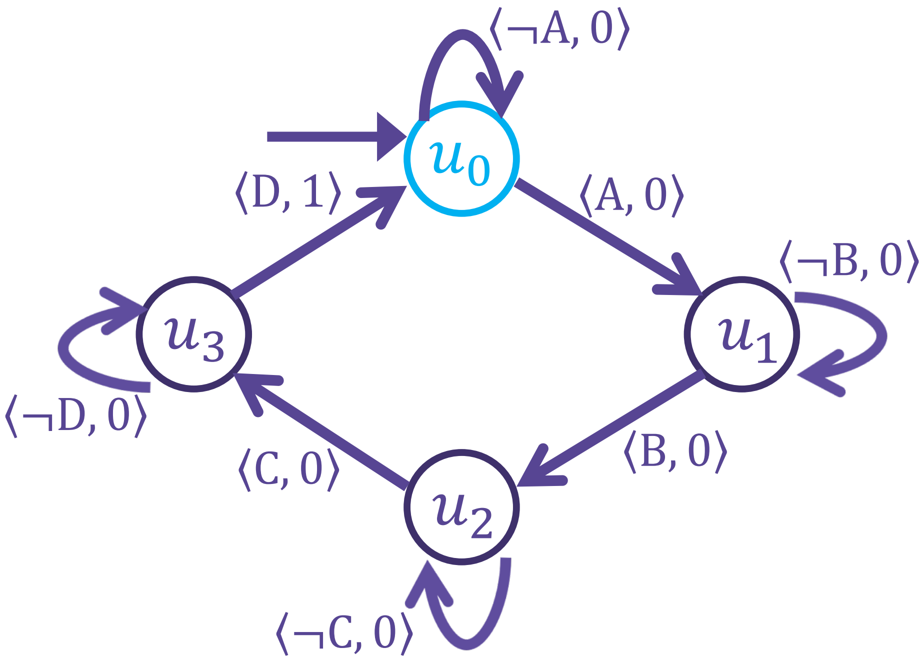
Reward Machines in Action



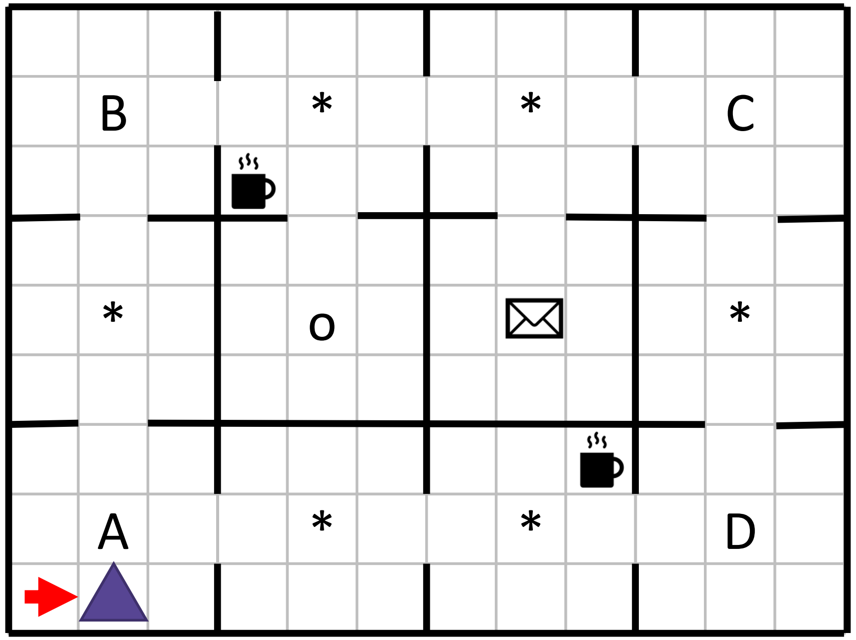

$\langle \neg A, 0\rangle$
u_0
$\langle D, 1\rangle$
$\langle A, 0\rangle$
$\langle \neg B, 0\rangle$
u_3
u_1
$\langle \neg D, 0\rangle$
$\langle C, 0\rangle$
$\langle B, 0\rangle$
u_2
$\langle \neg C, 0\rangle$

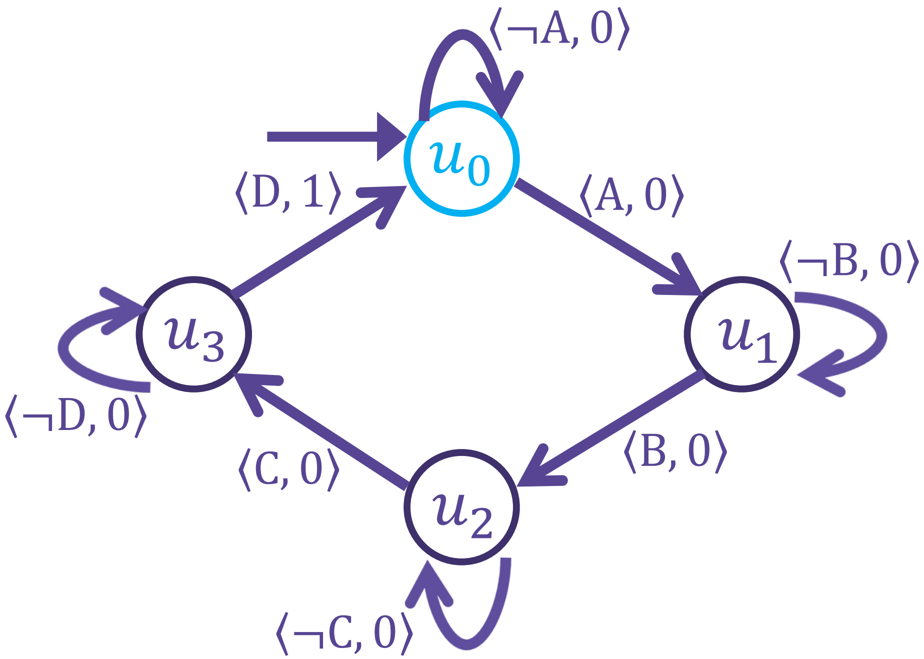
Reward Machines in Action



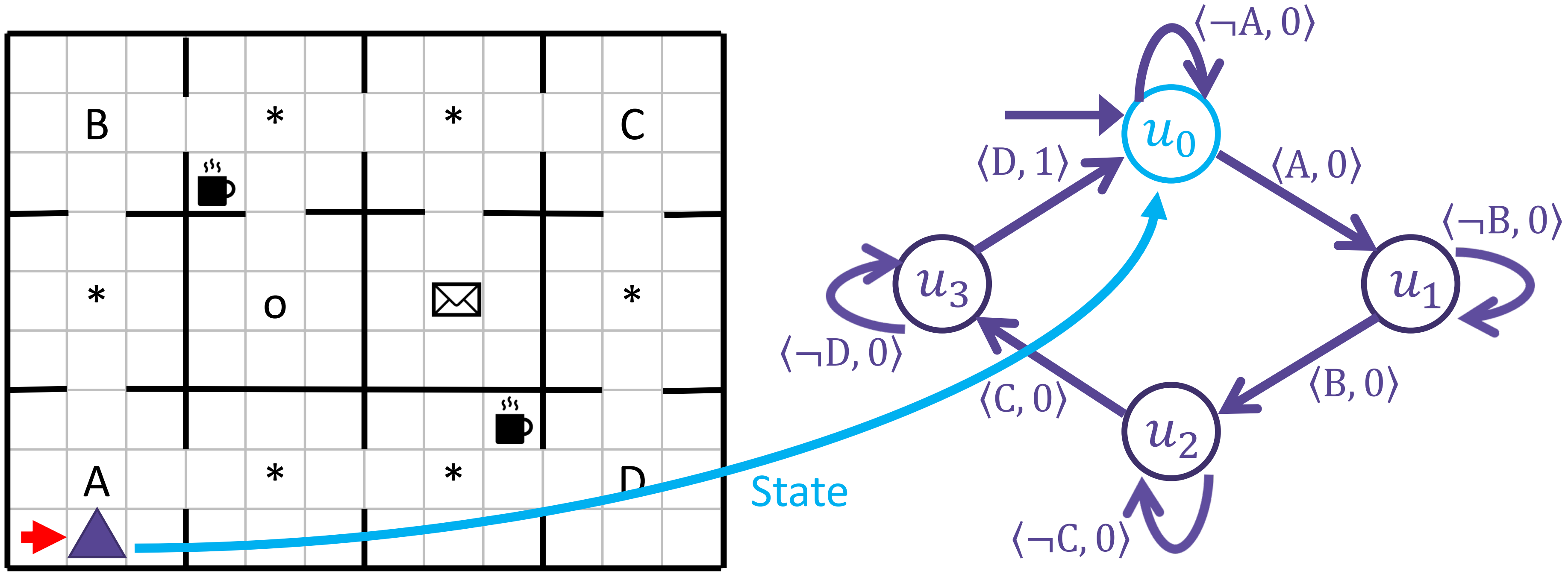


Reward Machines in Action

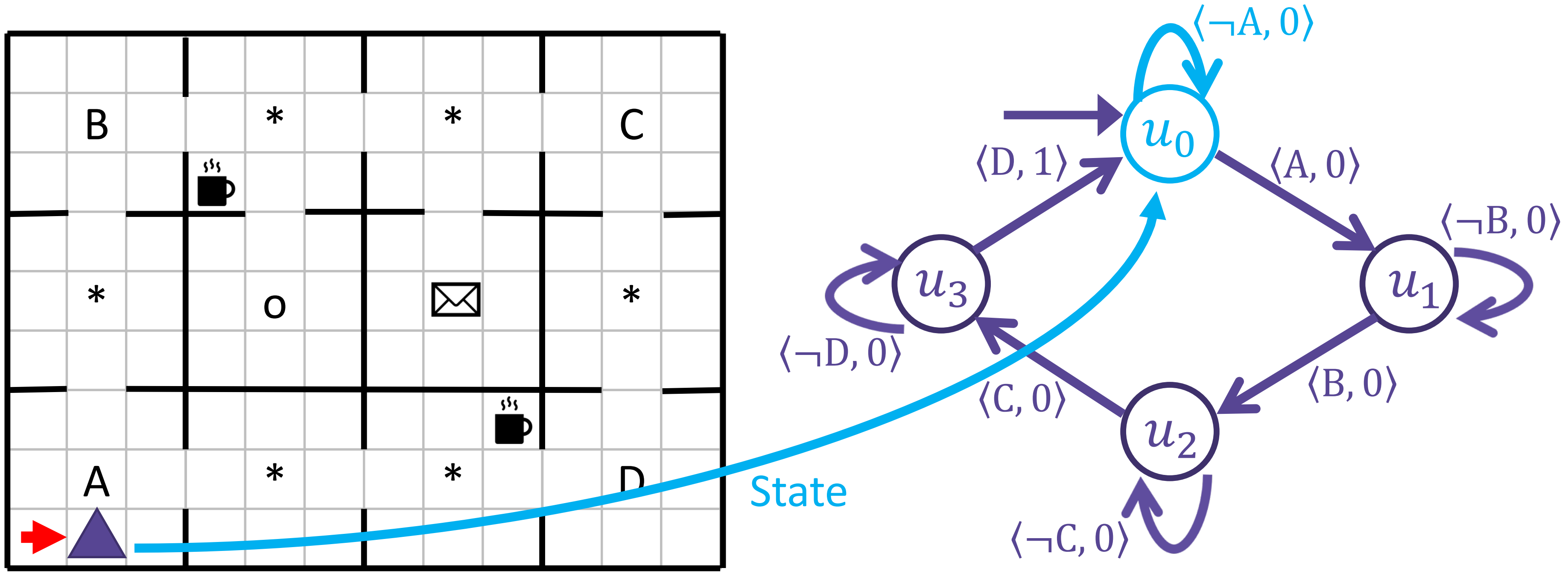




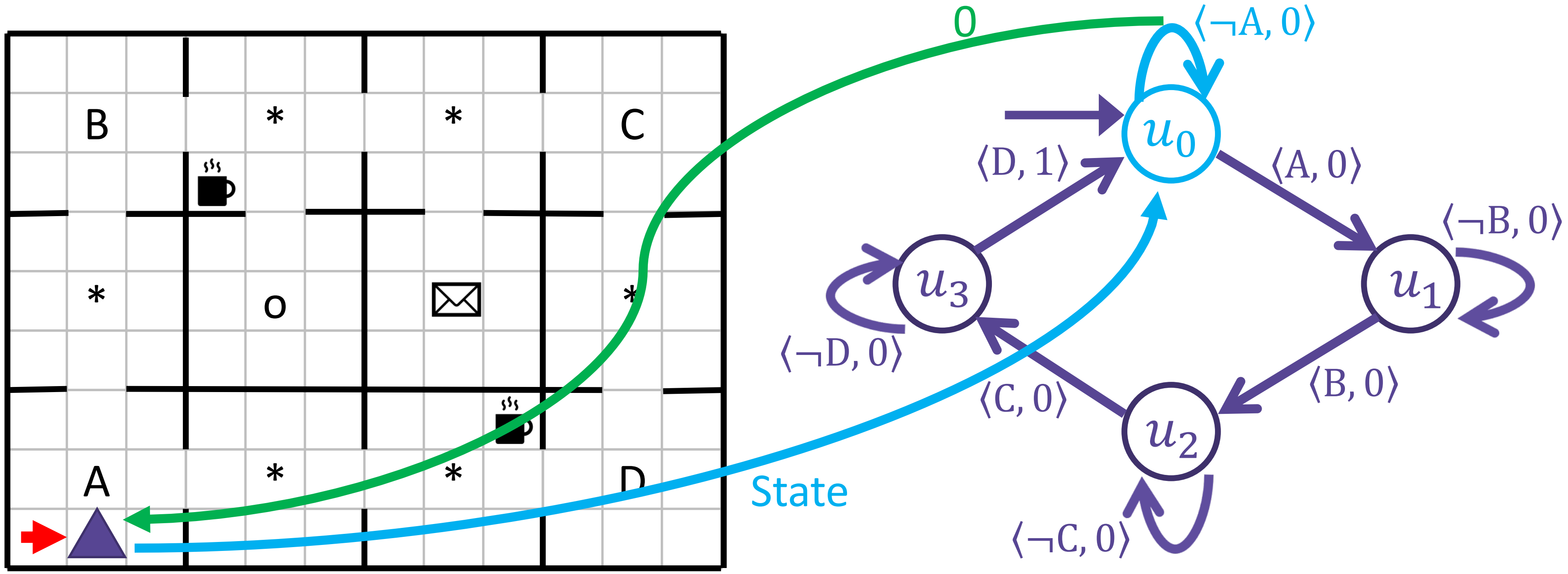
Reward Machines in Action



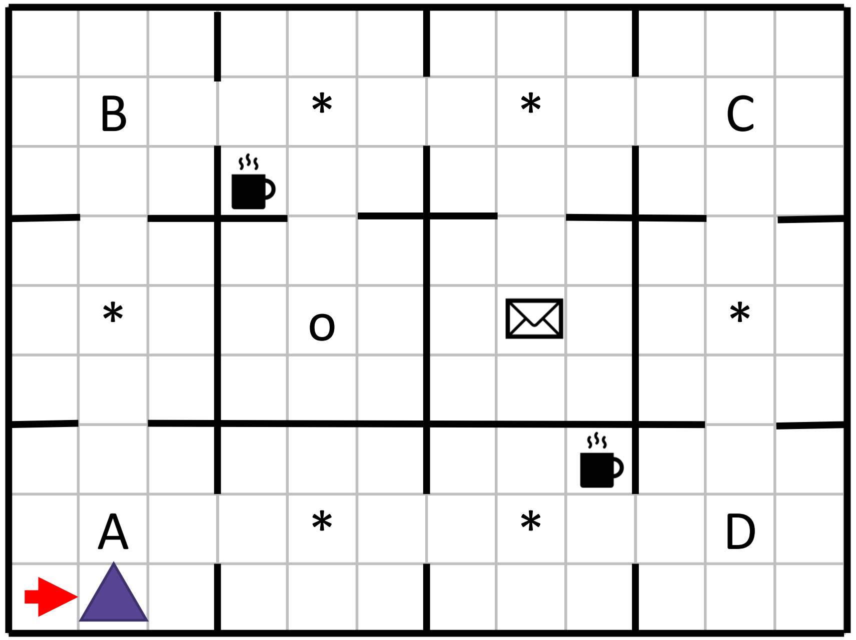
Reward Machines in Action

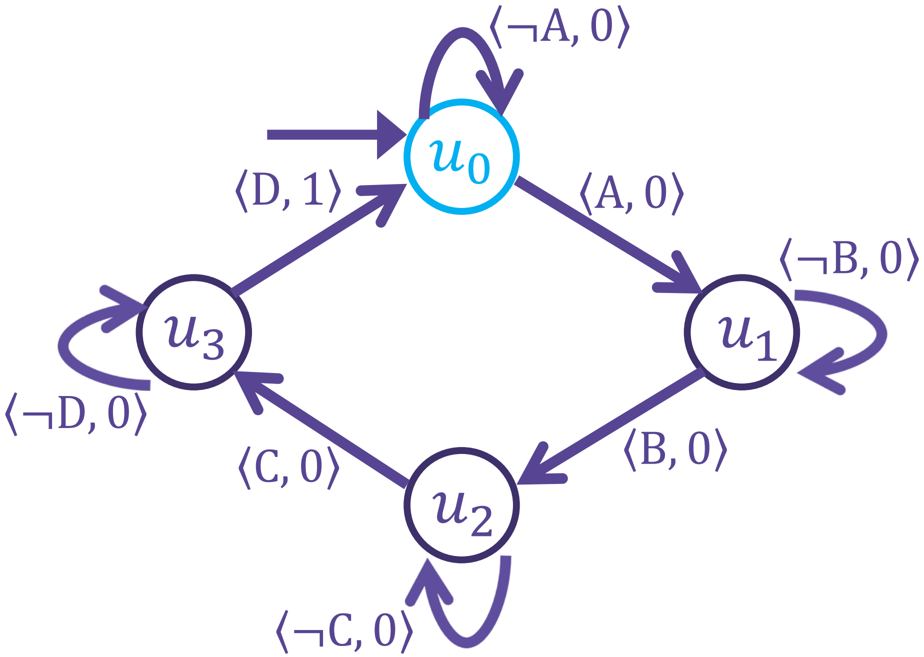


Reward Machines in Action

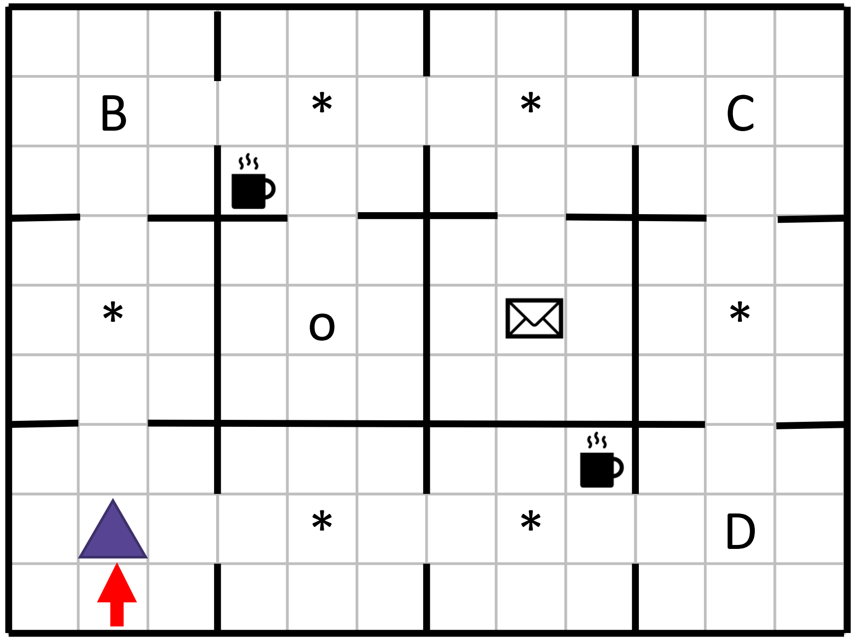


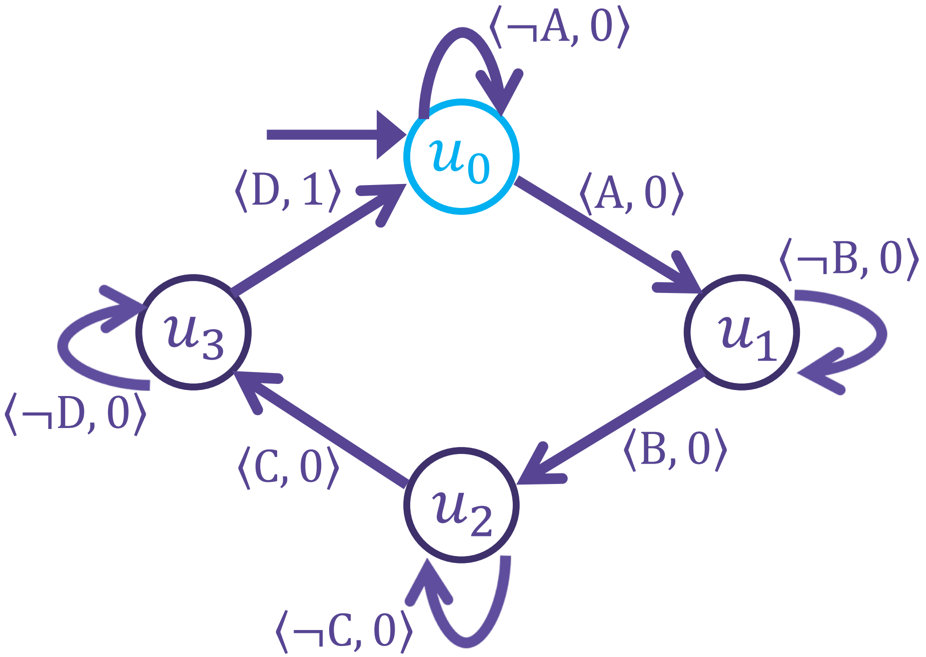
Reward Machines in Action



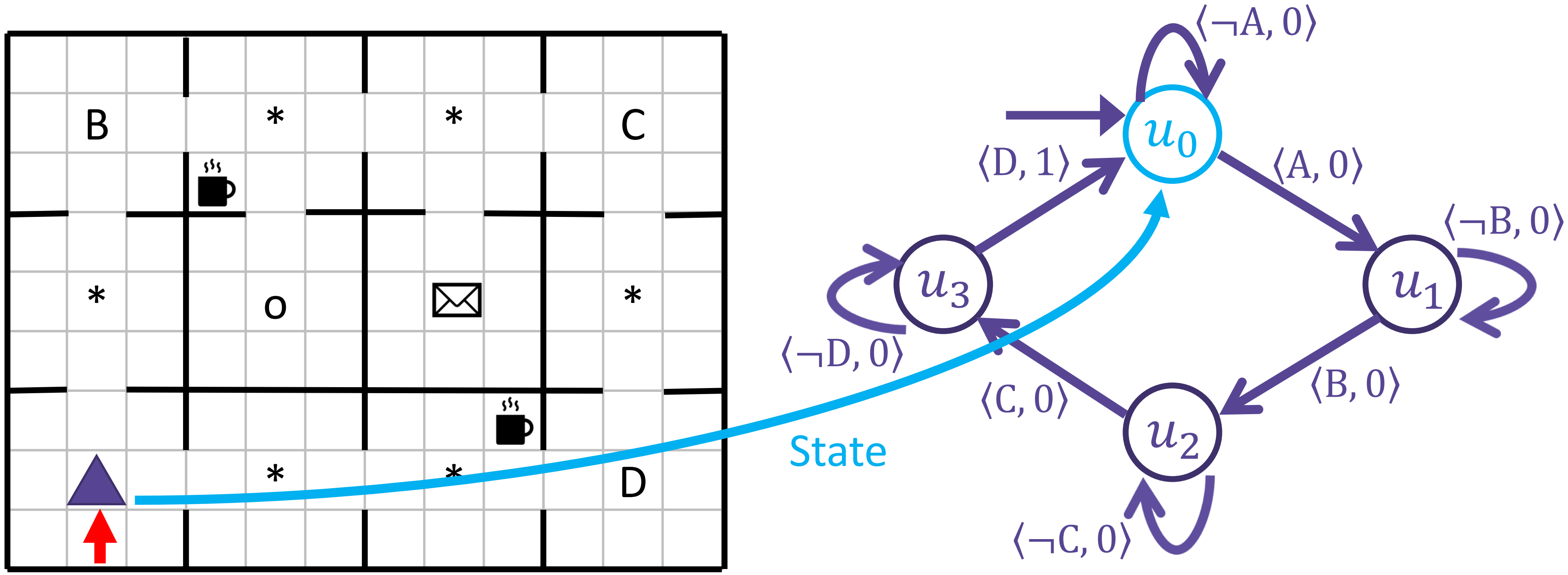

$\langle \neg A, 0 \rangle$
u_0
$\langle D, 1 \rangle$
$\langle A, 0 \rangle$
$\langle \neg B, 0 \rangle$
u_3
u_1
$\langle \neg D, 0 \rangle$
$\langle C, 0 \rangle$
$\langle B, 0 \rangle$
u_2
$\langle \neg C, 0 \rangle$

Reward Machines in Action

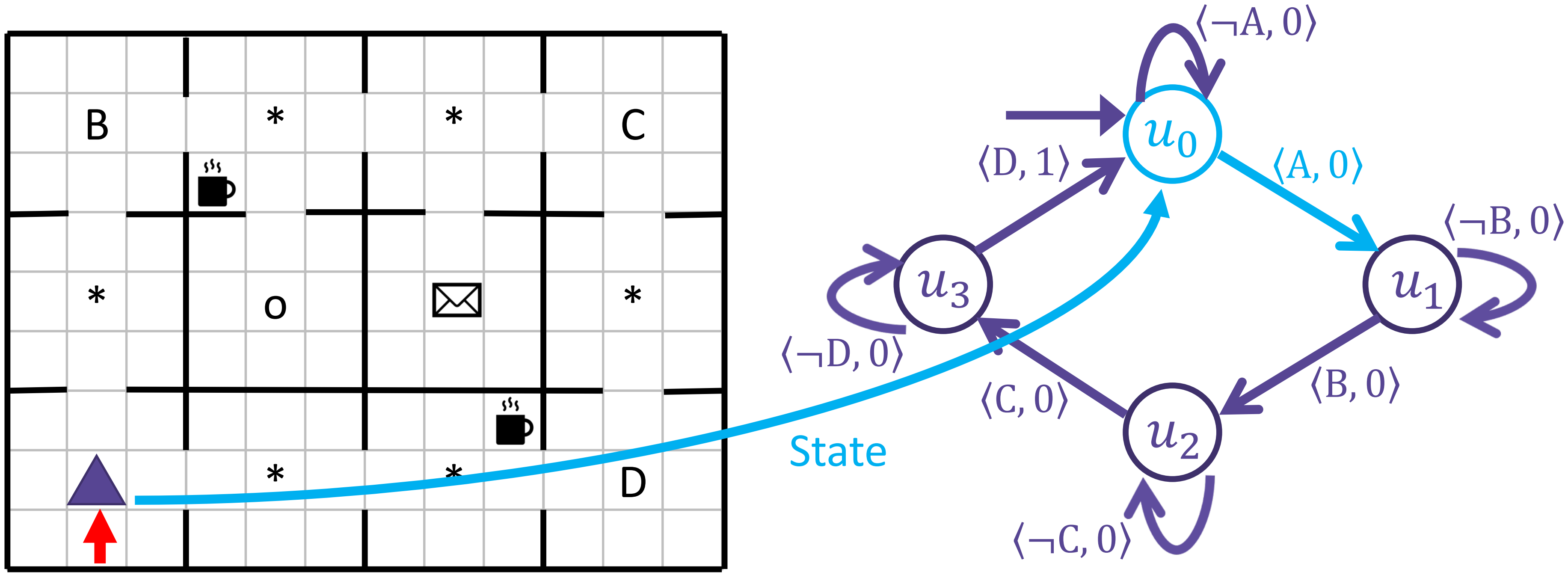




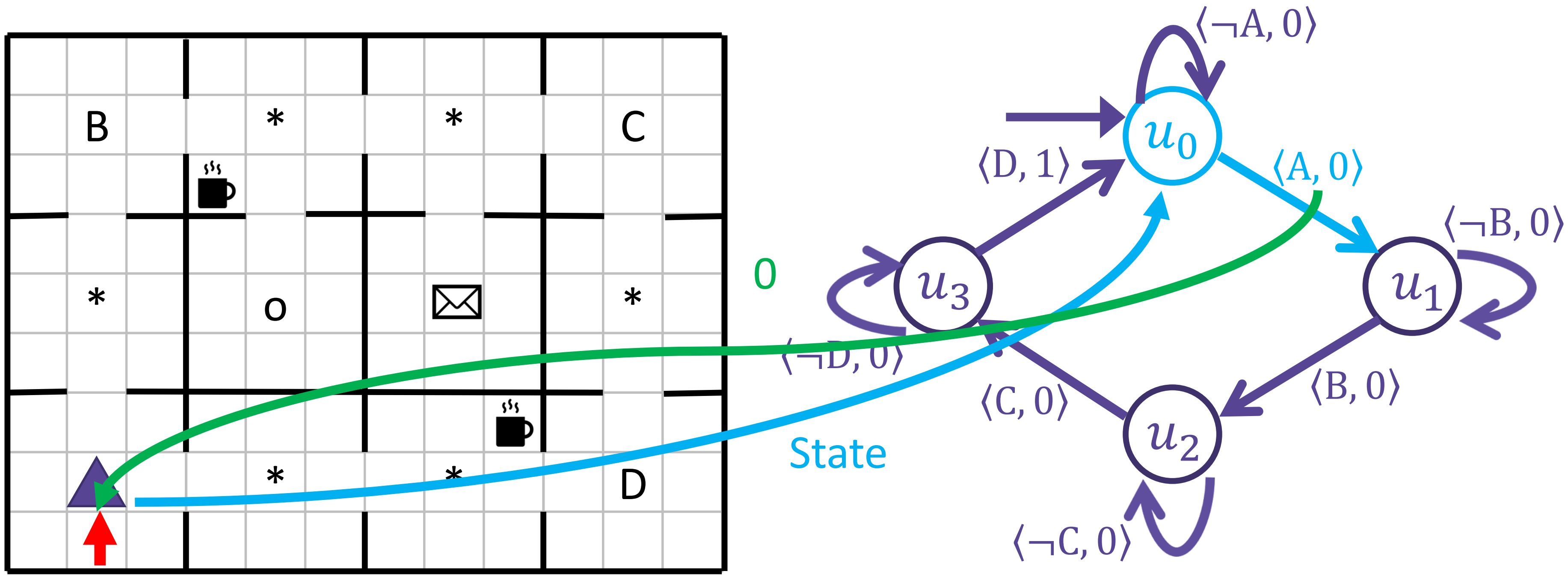
Reward Machines in Action



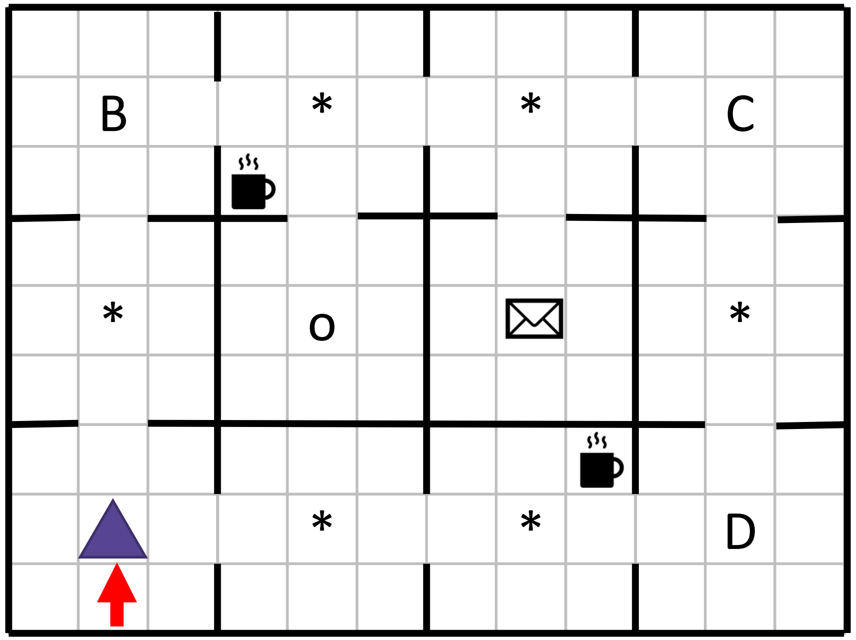
Reward Machines in Action

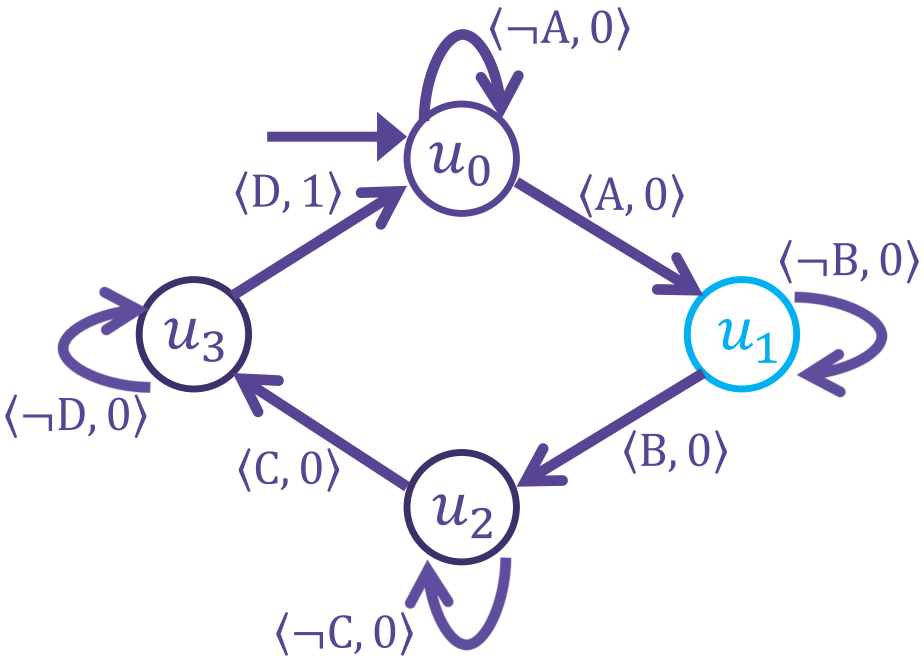


Reward Machines in Action

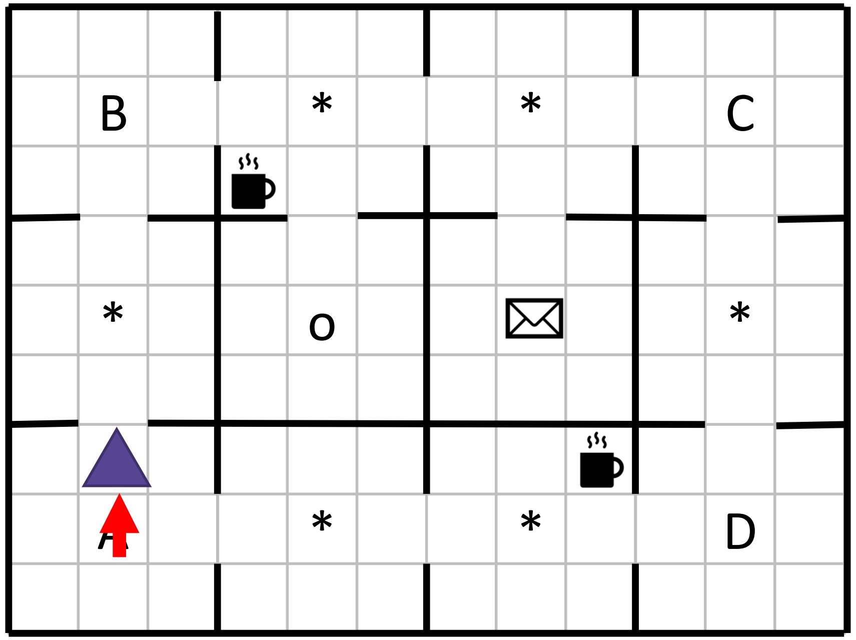


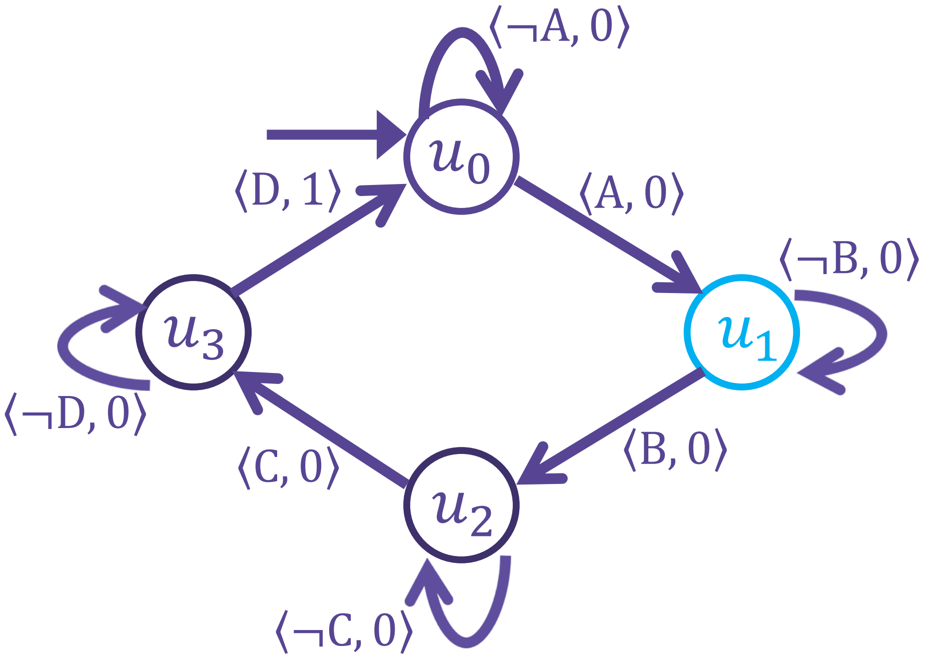
Reward Machines in Action



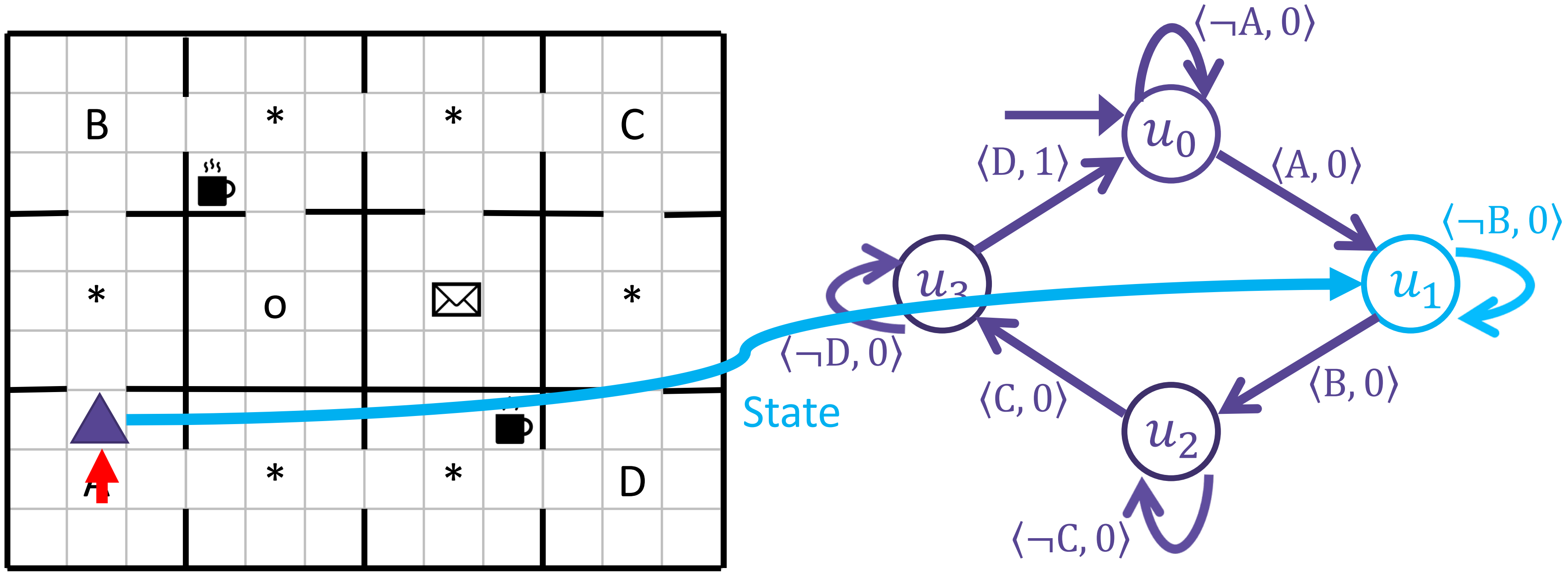


Reward Machines in Action

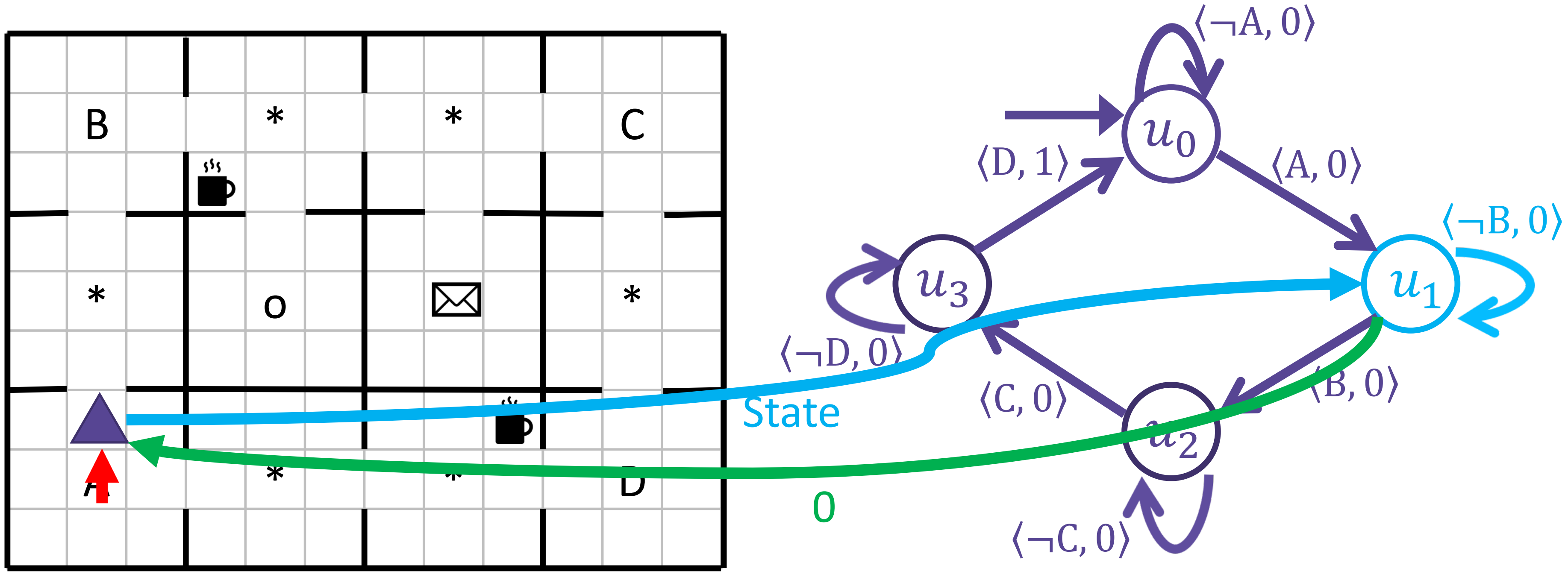



$\langle \neg A, 0 \rangle$
u_0
$\langle D, 1 \rangle$
$\langle A, 0 \rangle$
$\langle \neg B, 0 \rangle$
u_3
u_1
$\langle \neg D, 0 \rangle$
$\langle C, 0 \rangle$
$\langle B, 0 \rangle$
u_2
$\langle \neg C, 0 \rangle$

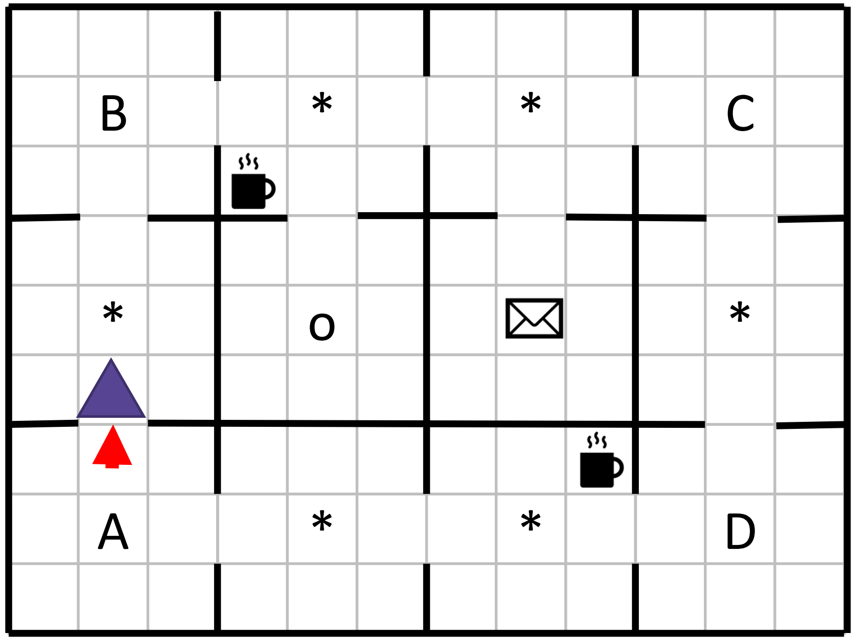
Reward Machines in Action

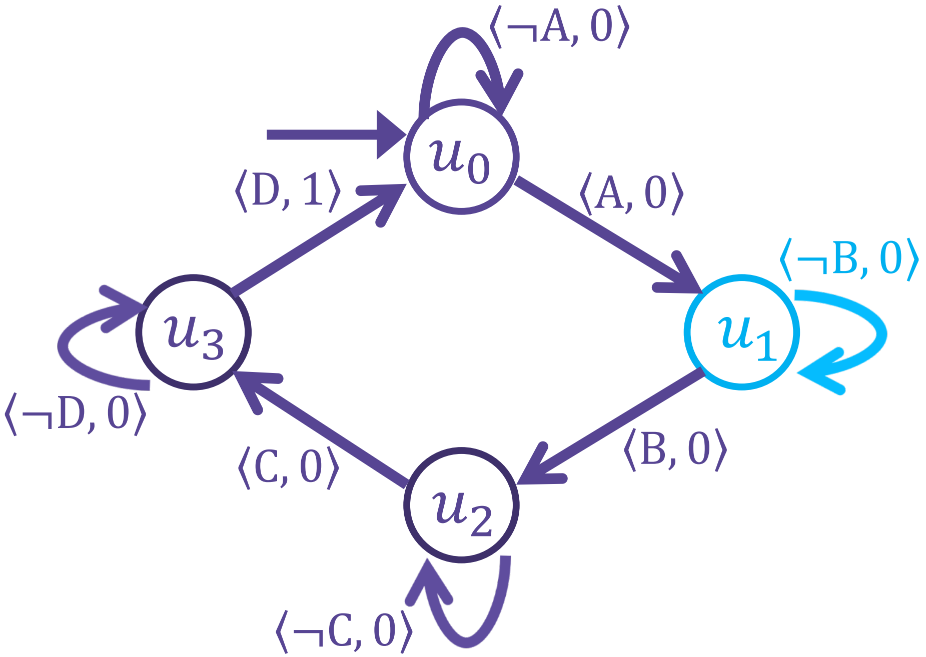


Reward Machines in Action

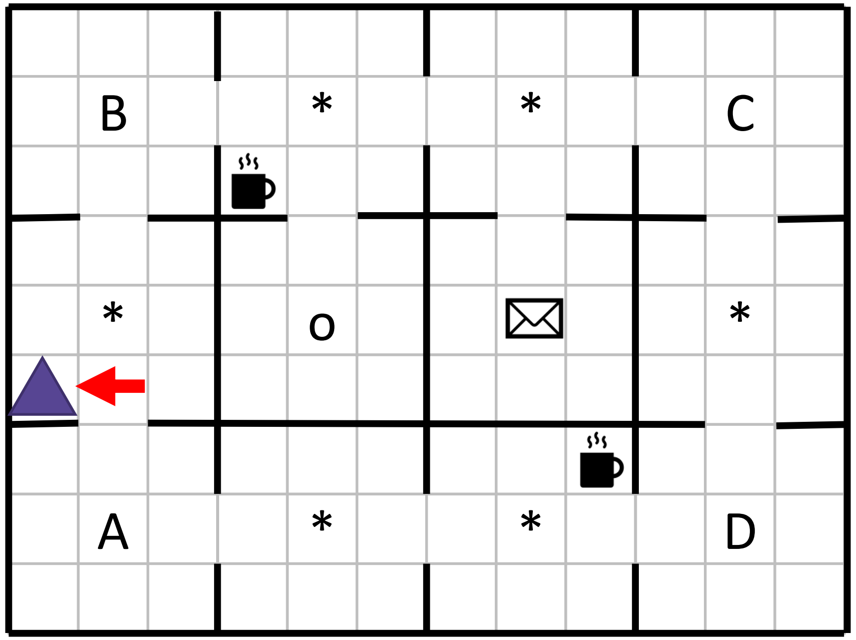


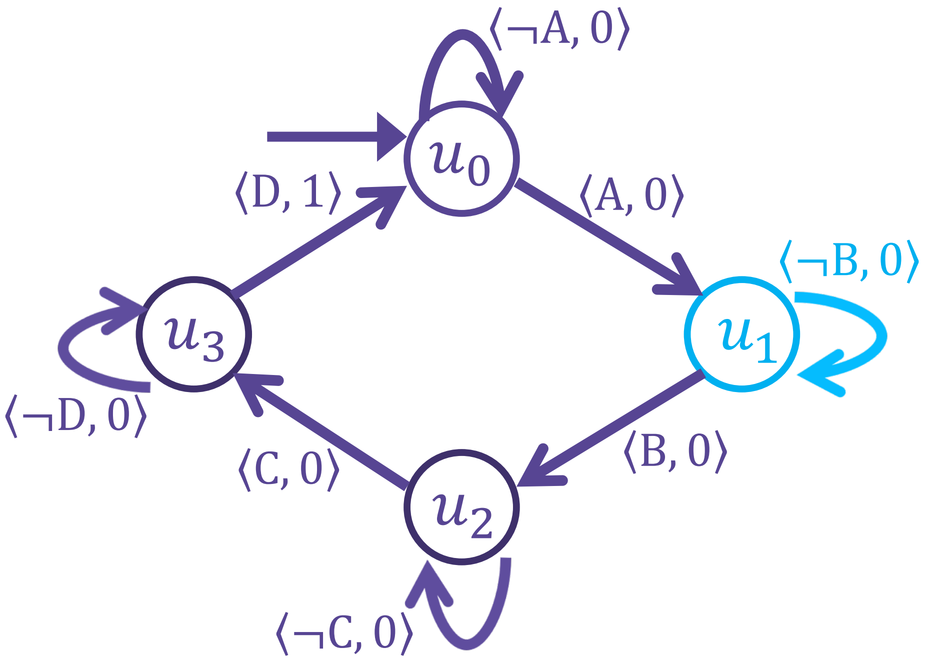
Reward Machines in Action



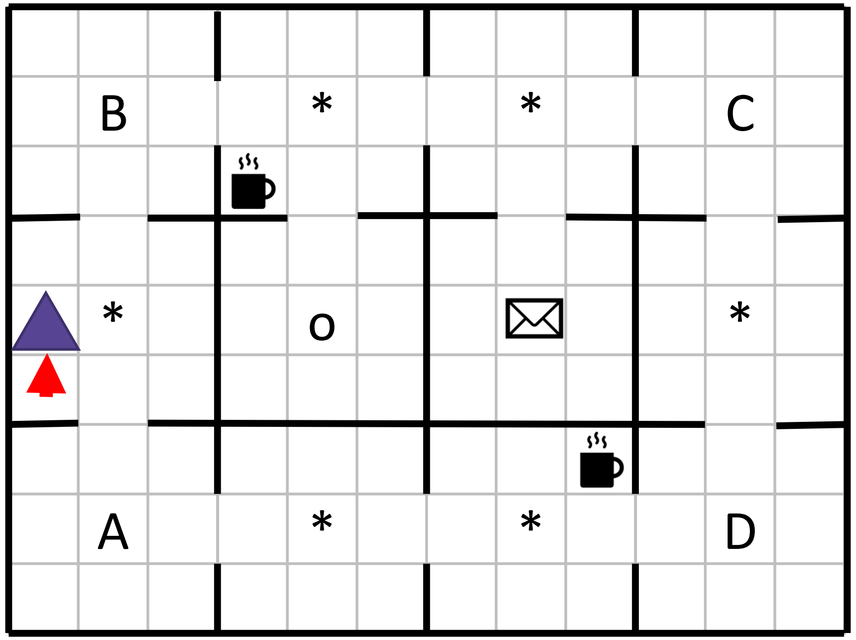

$\langle \neg A, 0 \rangle$
u_0
$\langle D, 1 \rangle$
$\langle A, 0 \rangle$
$\langle \neg B, 0 \rangle$
u_3
u_1
$\langle \neg D, 0 \rangle$
$\langle C, 0 \rangle$
$\langle B, 0 \rangle$
u_2
$\langle \neg C, 0 \rangle$

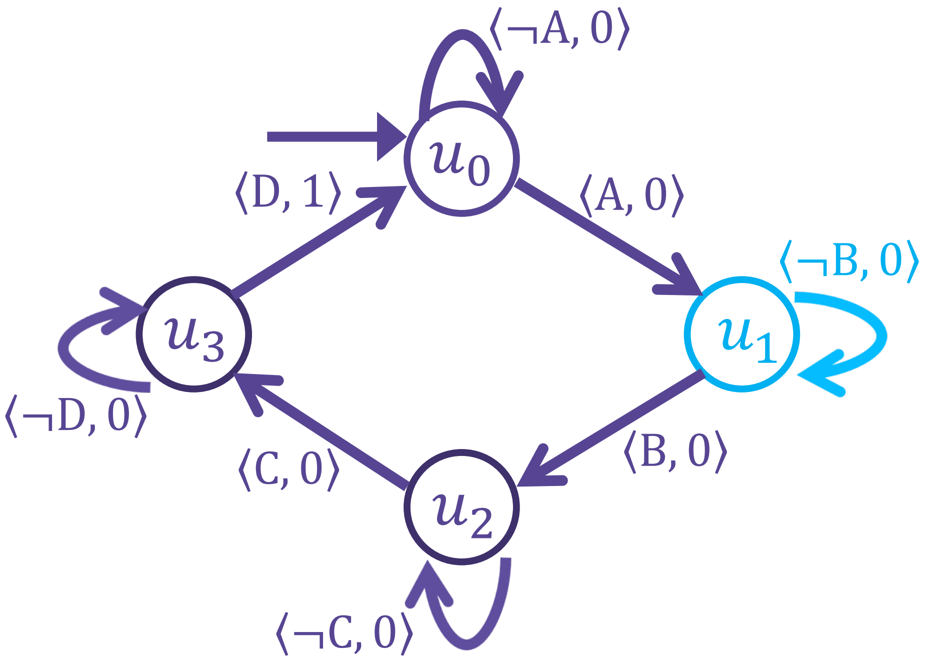
Reward Machines in Action



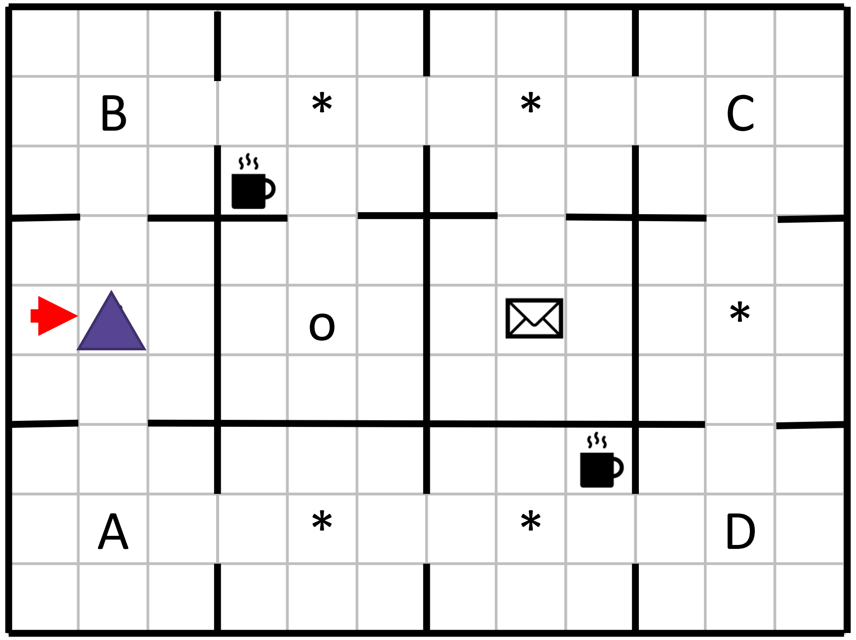


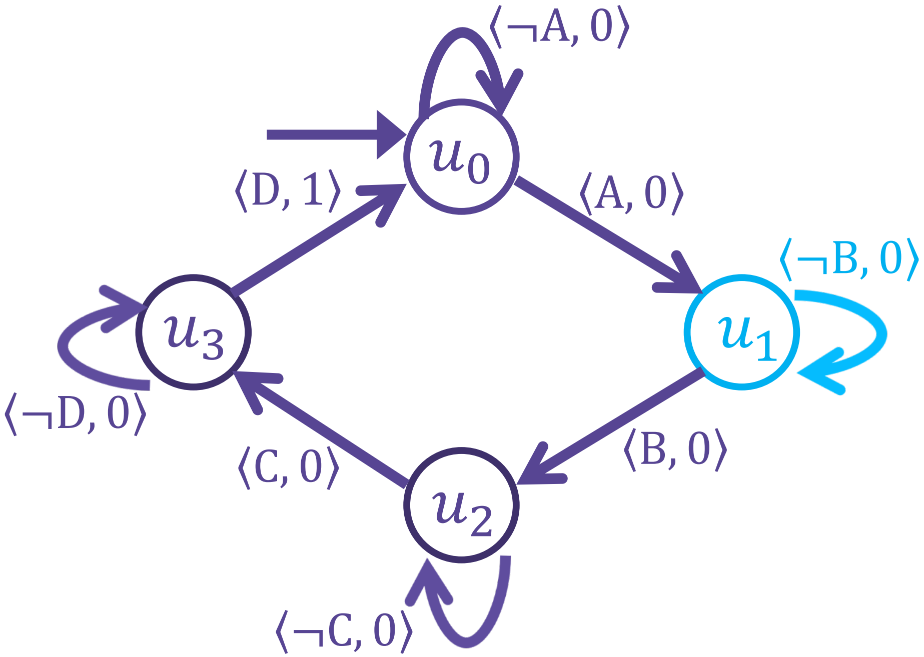
Reward Machines in Action



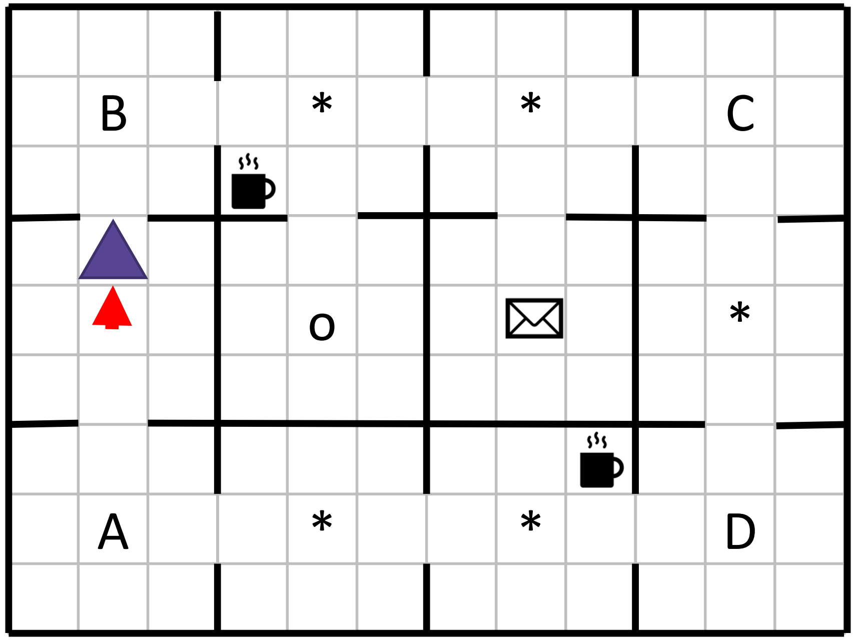


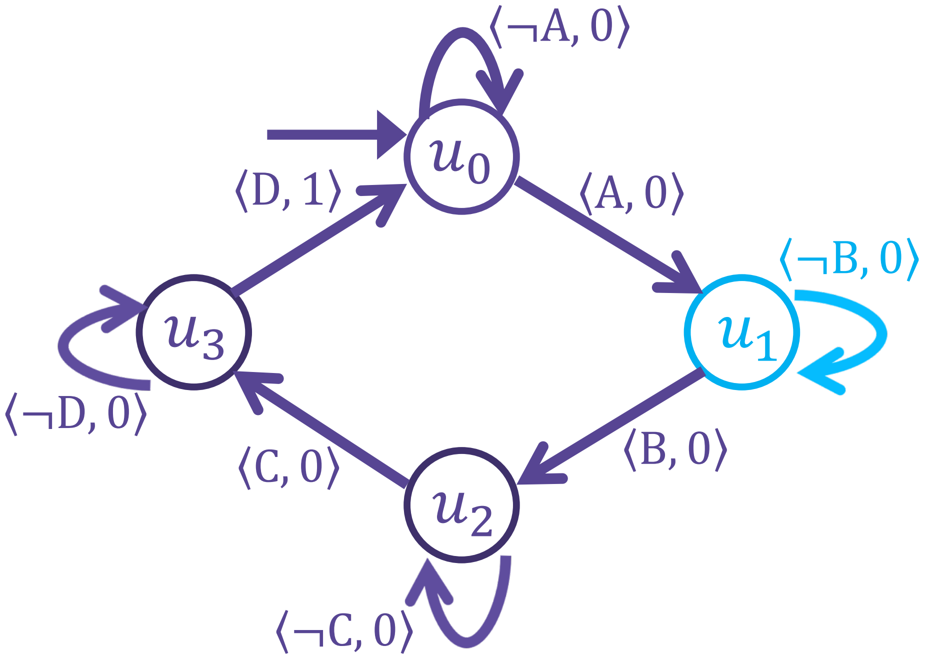
Reward Machines in Action



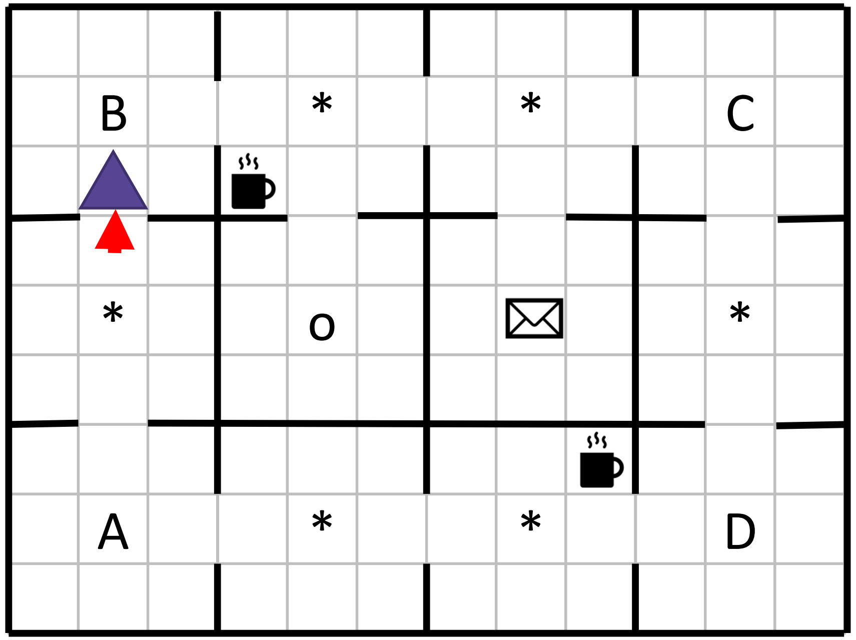


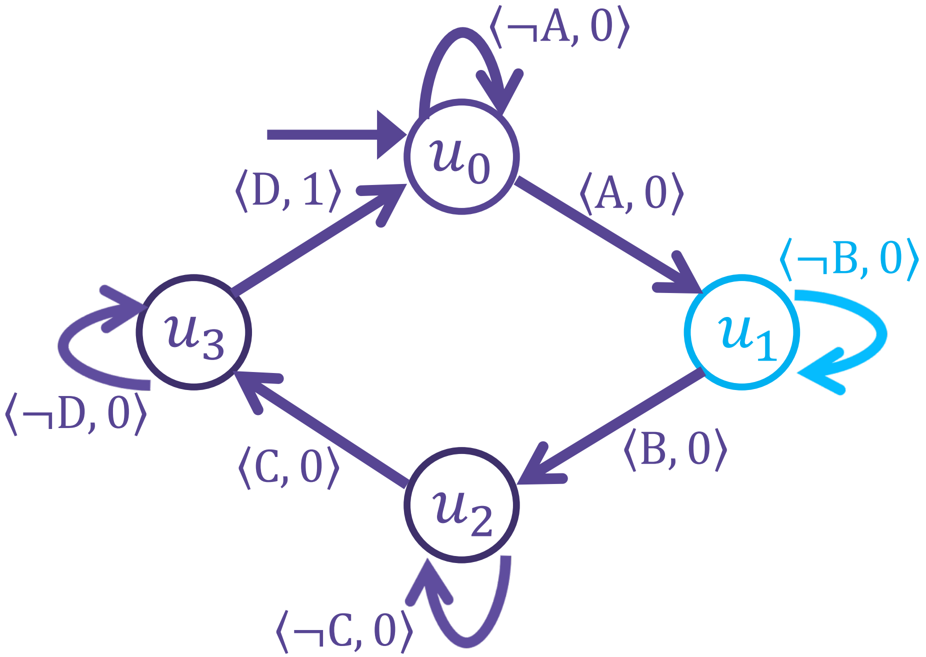
Reward Machines in Action



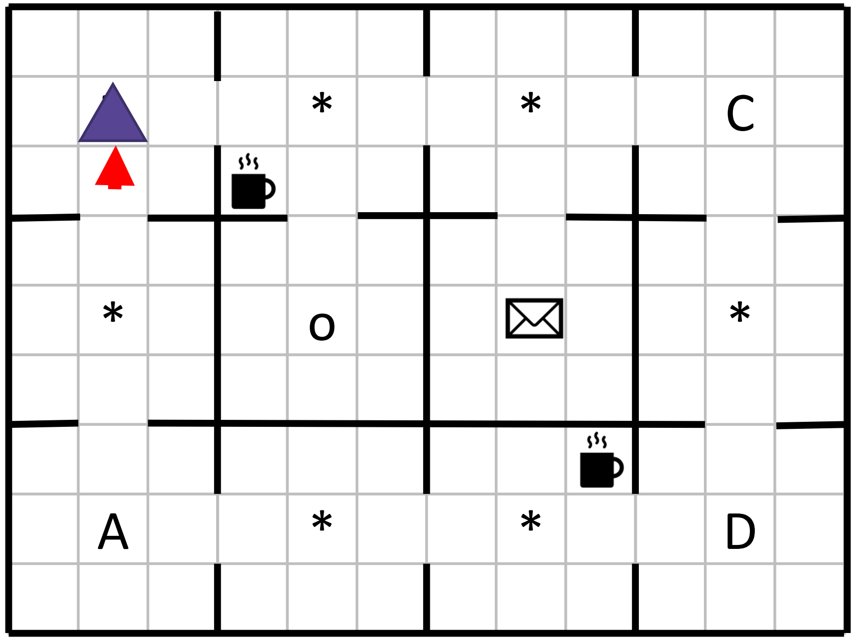


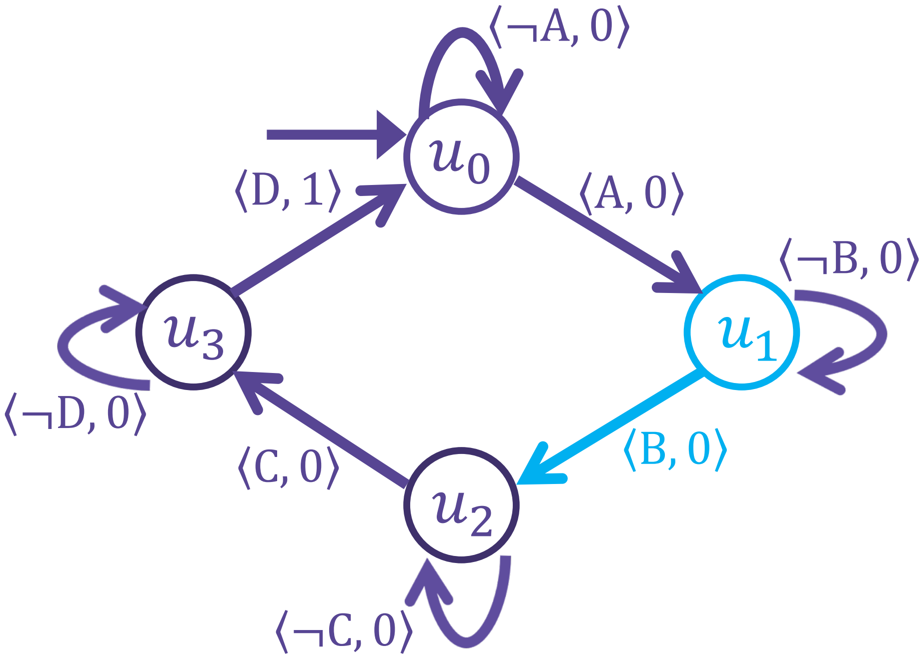
Reward Machines in Action



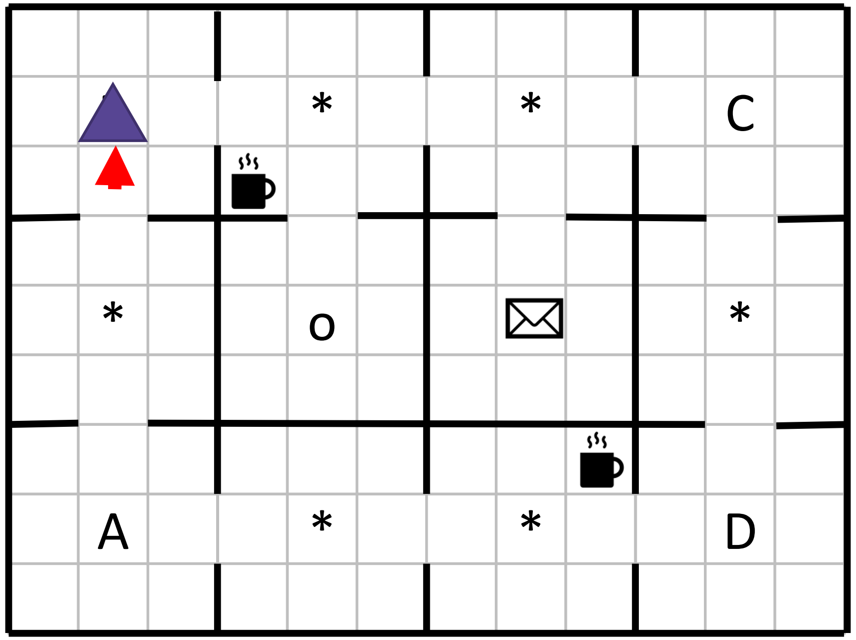


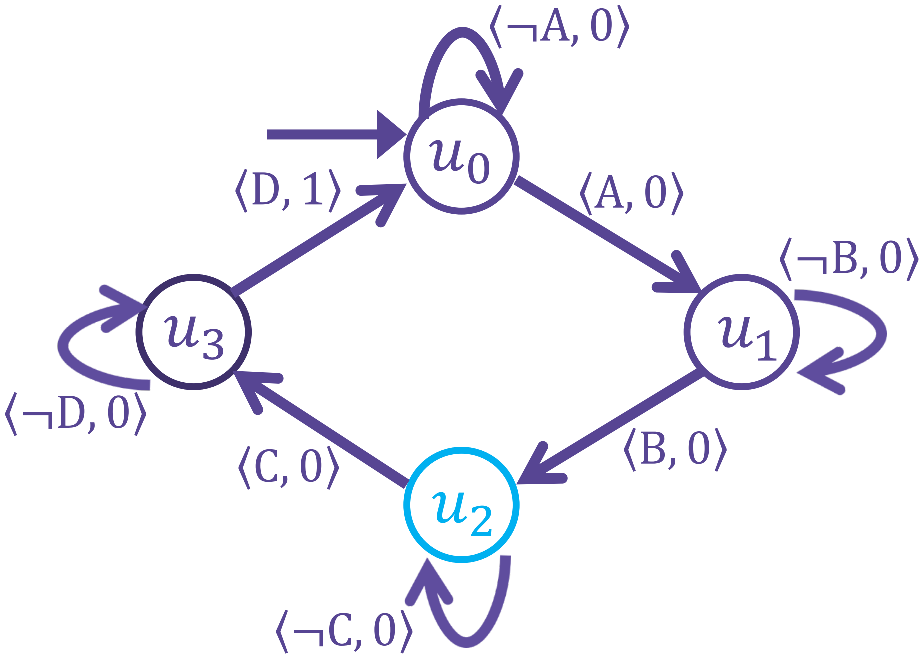
Reward Machines in Action



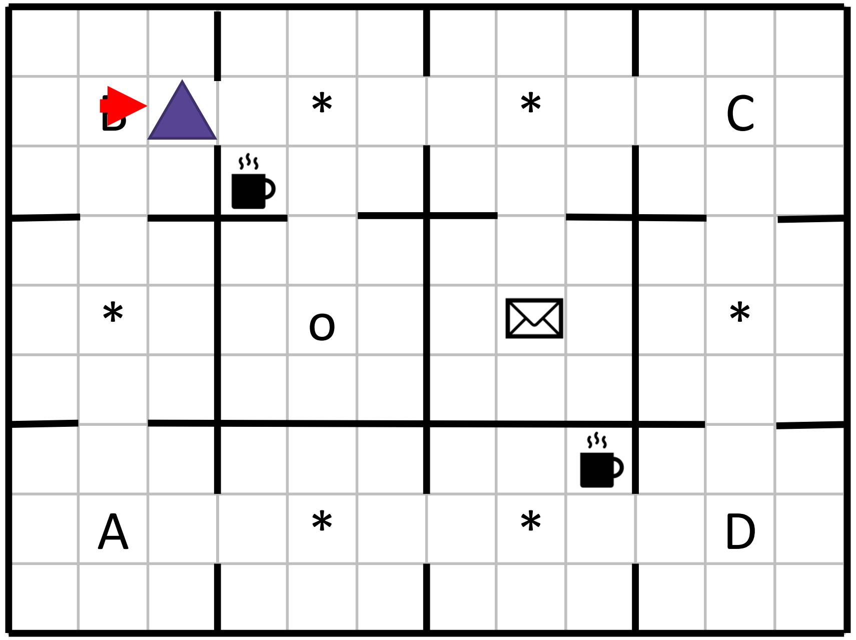


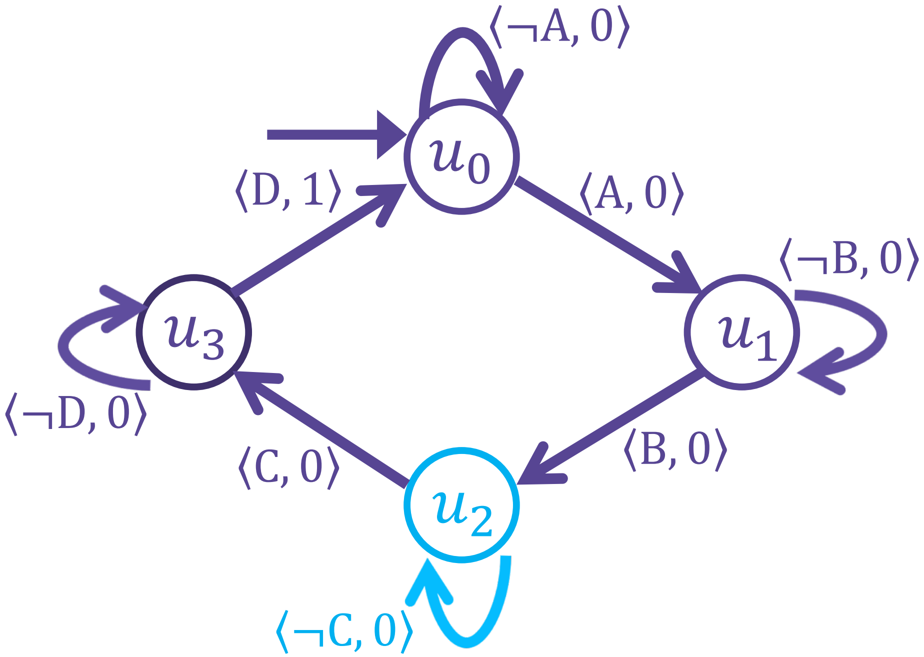
Reward Machines in Action





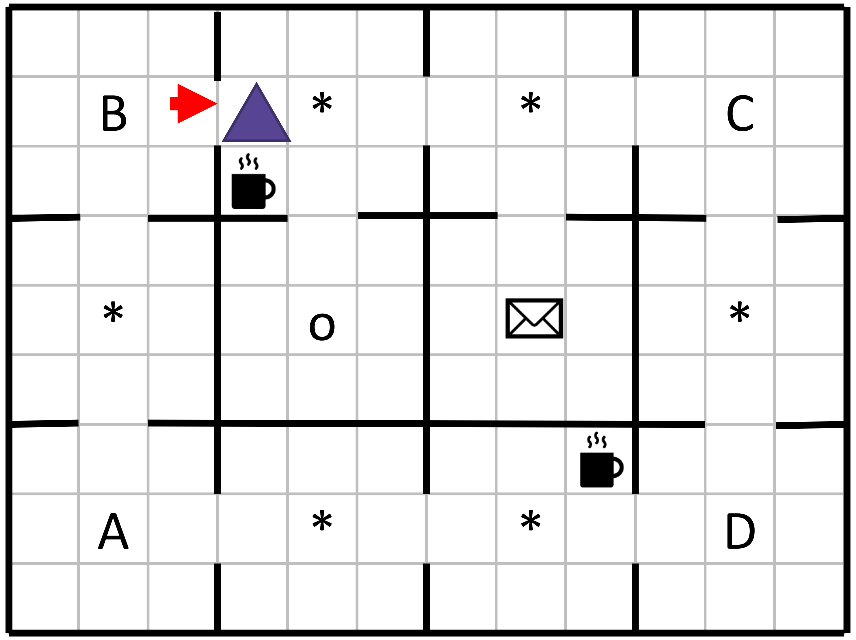
Reward Machines in Action

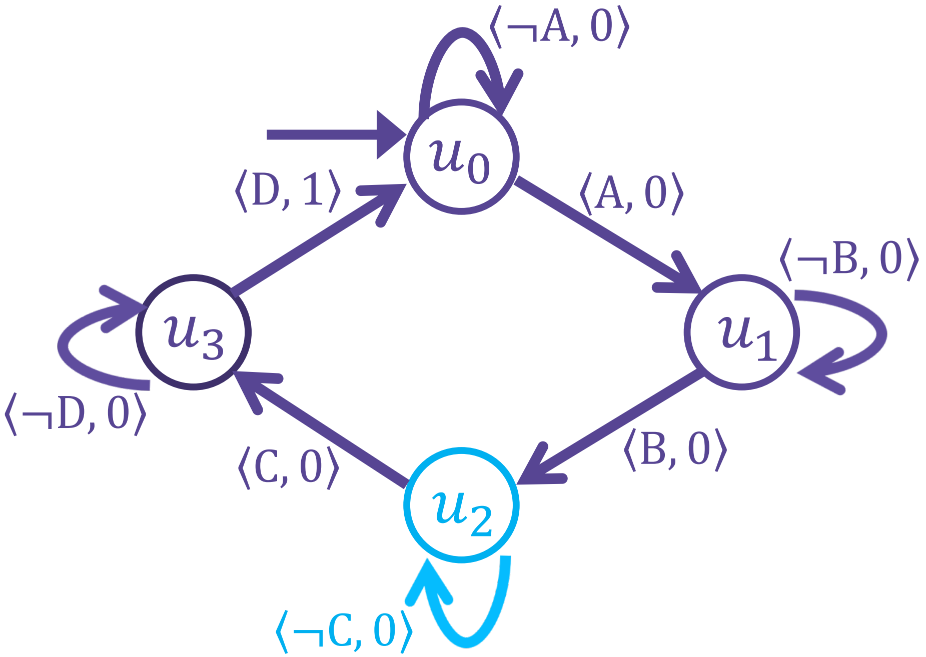




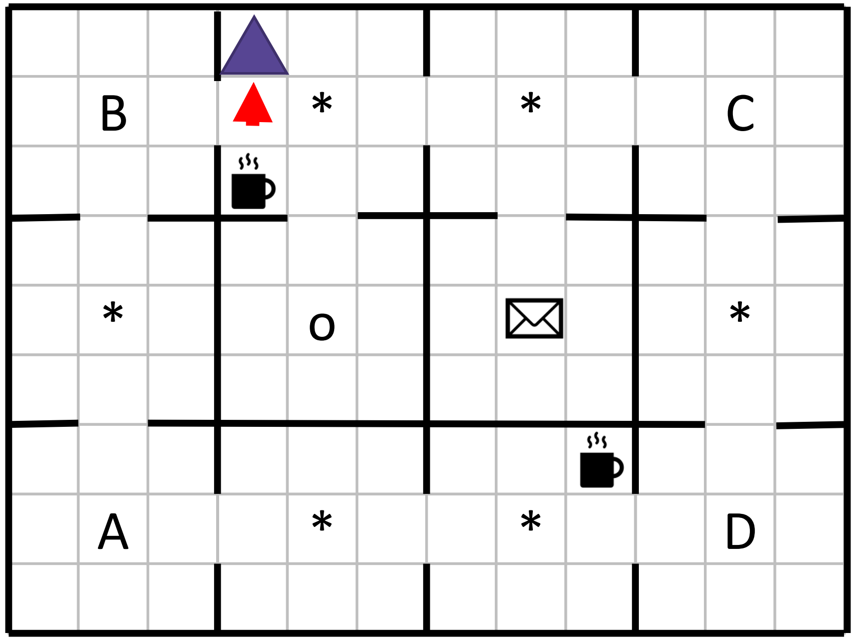
$\langle \neg A, 0\rangle$
u_0
$\langle D, 1\rangle$
$\langle A, 0\rangle$
$\langle \neg B, 0\rangle$
u_3
u_1
$\langle \neg D, 0\rangle$
$\langle C, 0\rangle$
$\langle B, 0\rangle$
u_2
$\langle \neg C, 0\rangle$

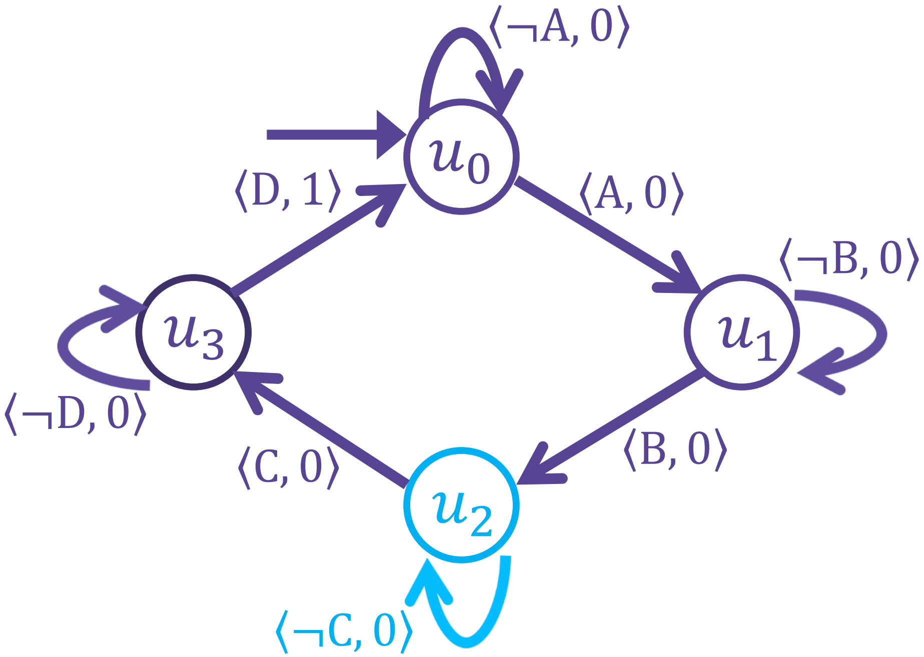
Reward Machines in Action



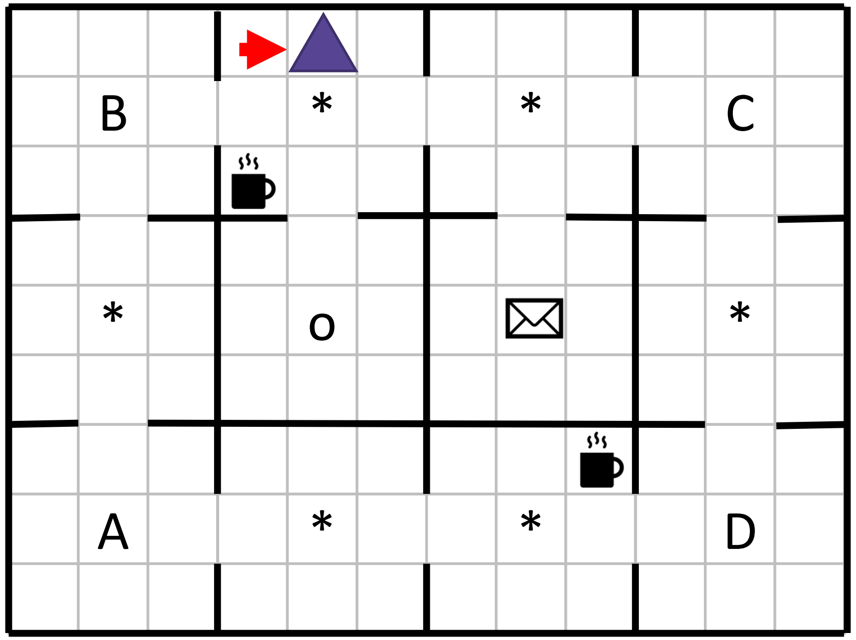

$\langle \neg A, 0 \rangle$
u_0
$\langle D, 1 \rangle$
$\langle A, 0 \rangle$
$\langle \neg B, 0 \rangle$
u_3
u_1
$\langle \neg D, 0 \rangle$
$\langle C, 0 \rangle$
$\langle B, 0 \rangle$
u_2
$\langle \neg C, 0 \rangle$

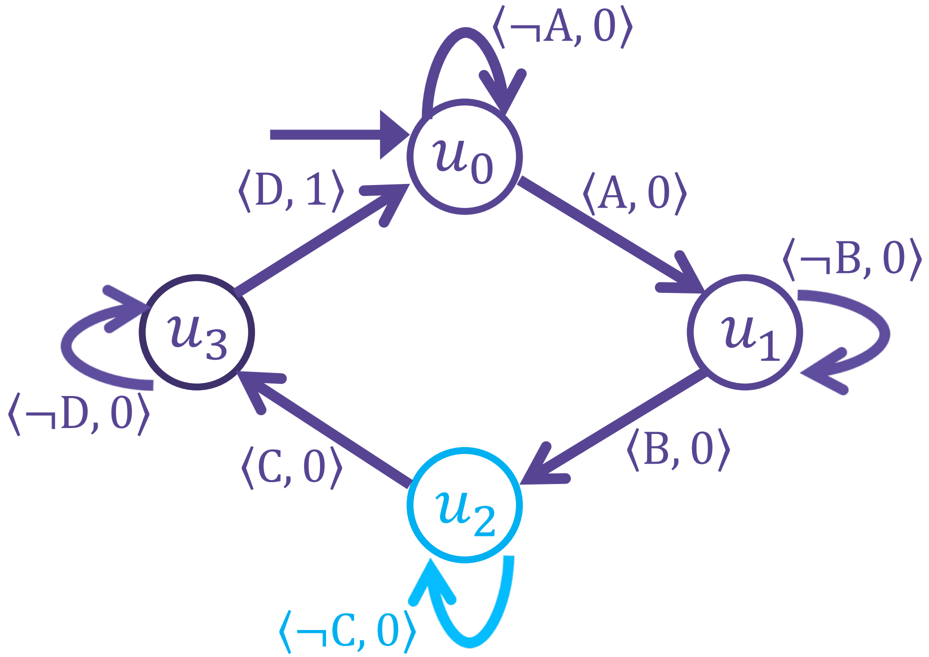
Reward Machines in Action



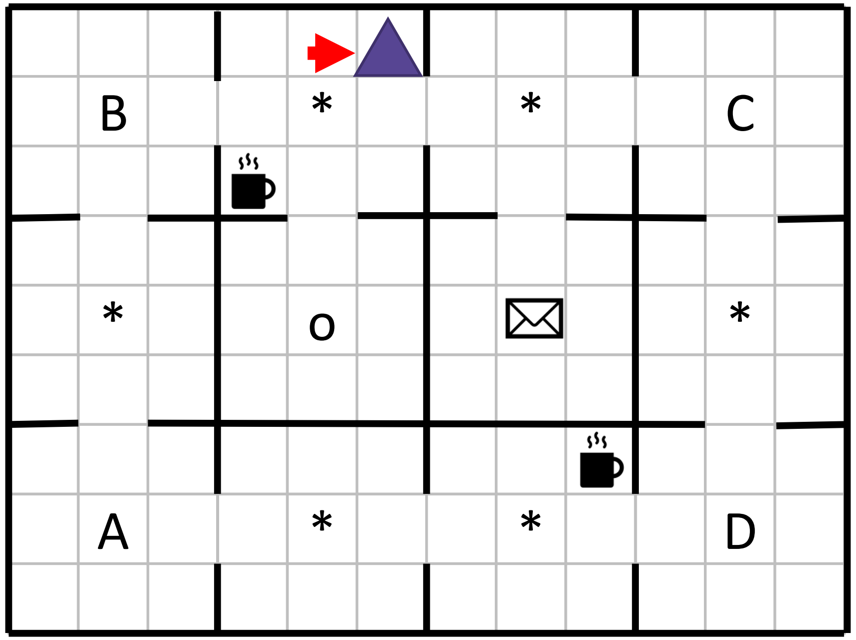


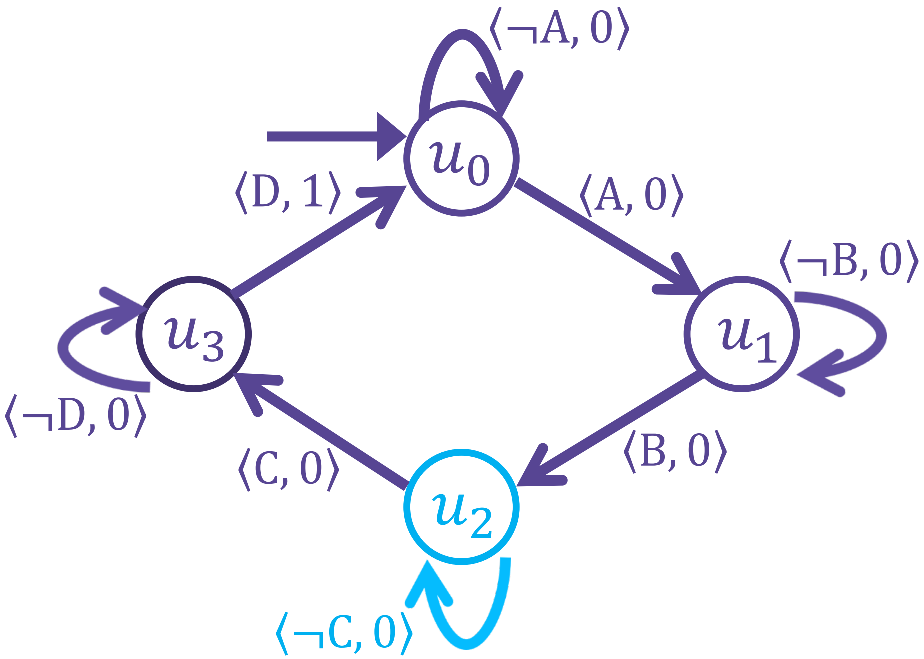
Reward Machines in Action



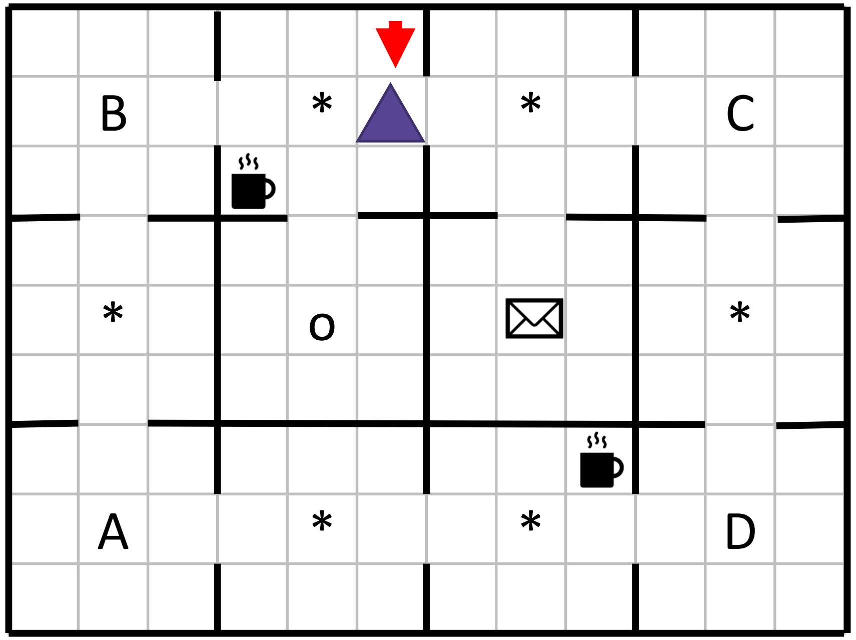


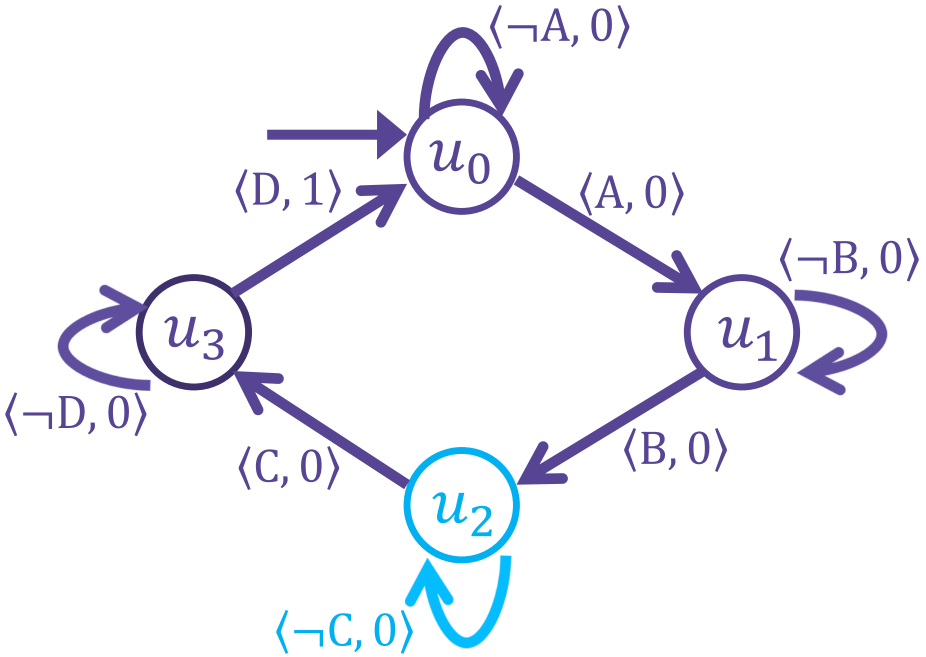
Reward Machines in Action



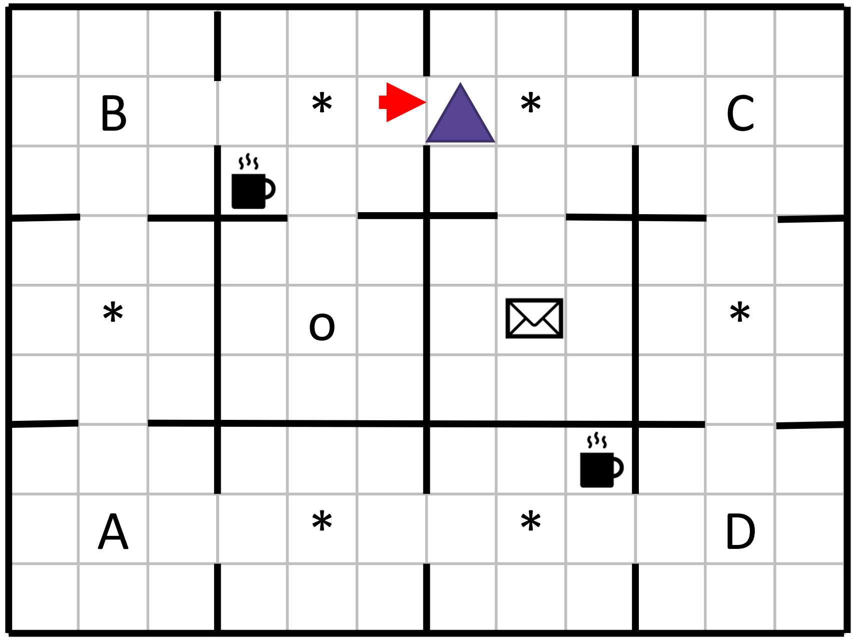


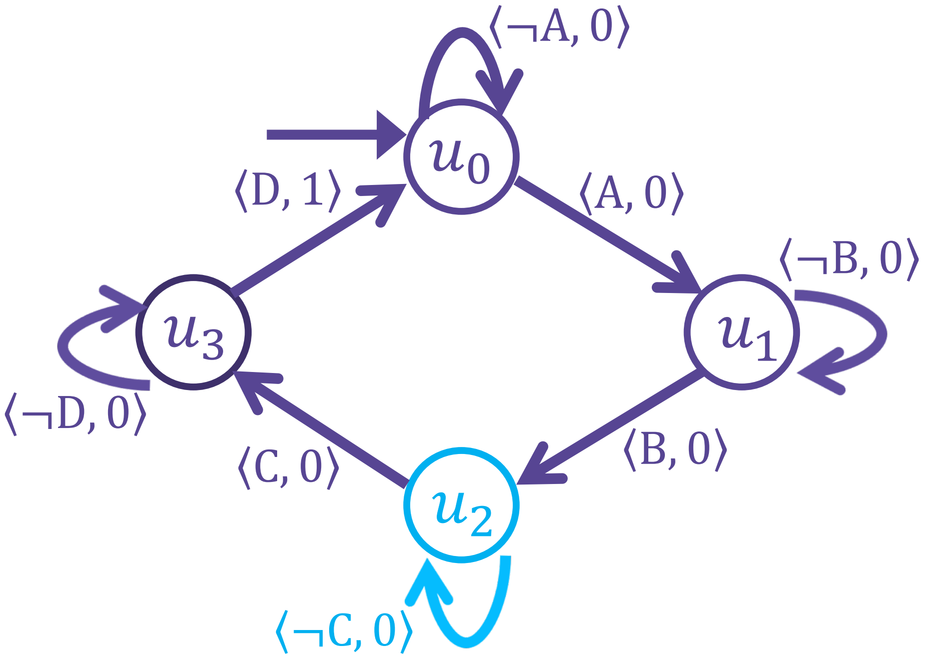
Reward Machines in Action



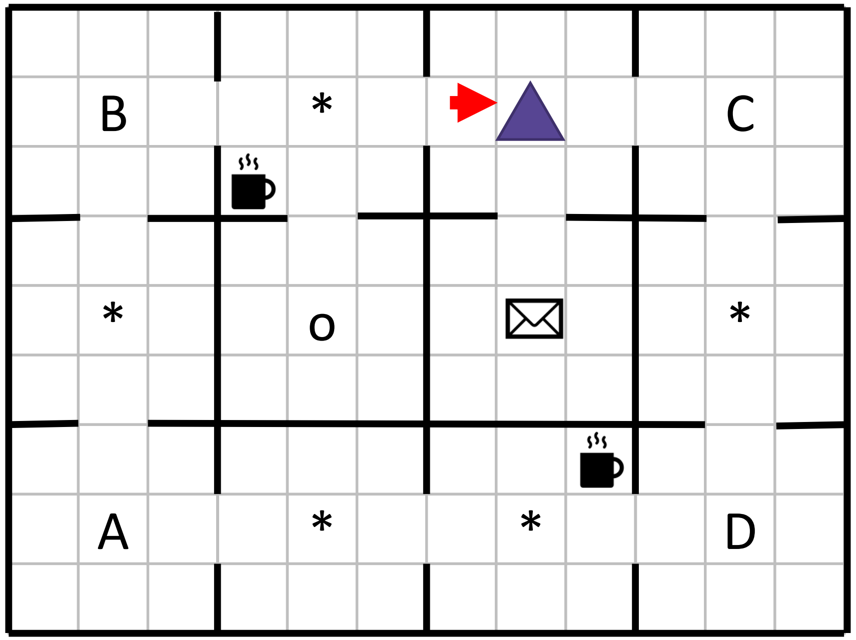


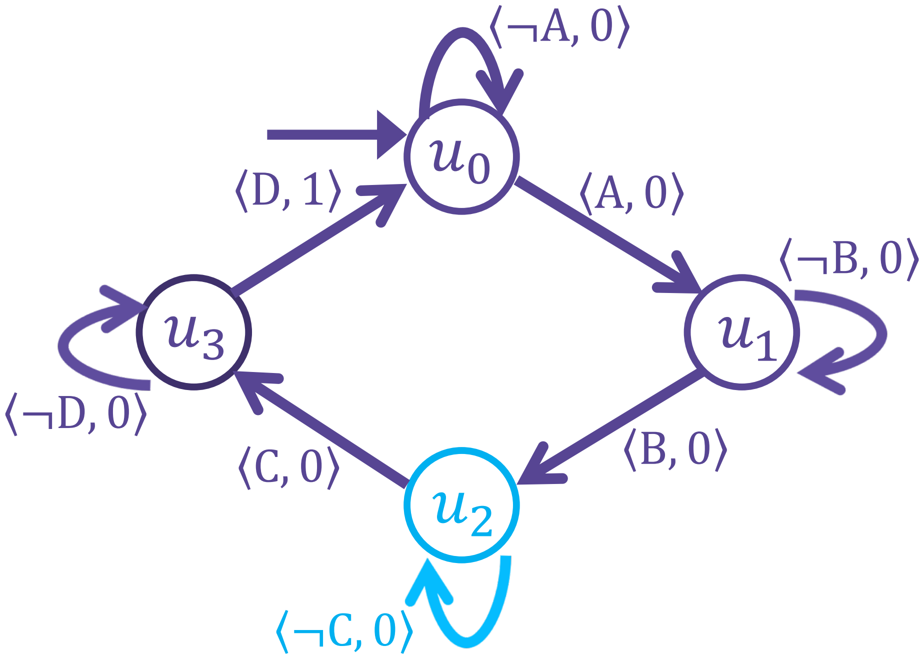
Reward Machines in Action



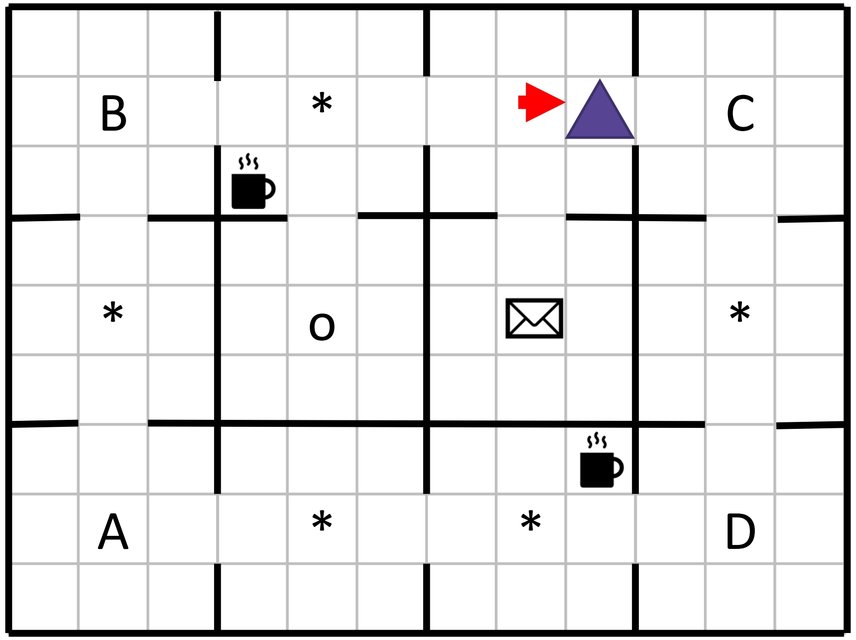

$\langle \neg A, 0 \rangle$
u_0
$\langle D, 1 \rangle$
$\langle A, 0 \rangle$
$\langle \neg B, 0 \rangle$
u_3
u_1
$\langle \neg D, 0 \rangle$
$\langle C, 0 \rangle$
$\langle B, 0 \rangle$
u_2
$\langle \neg C, 0 \rangle$

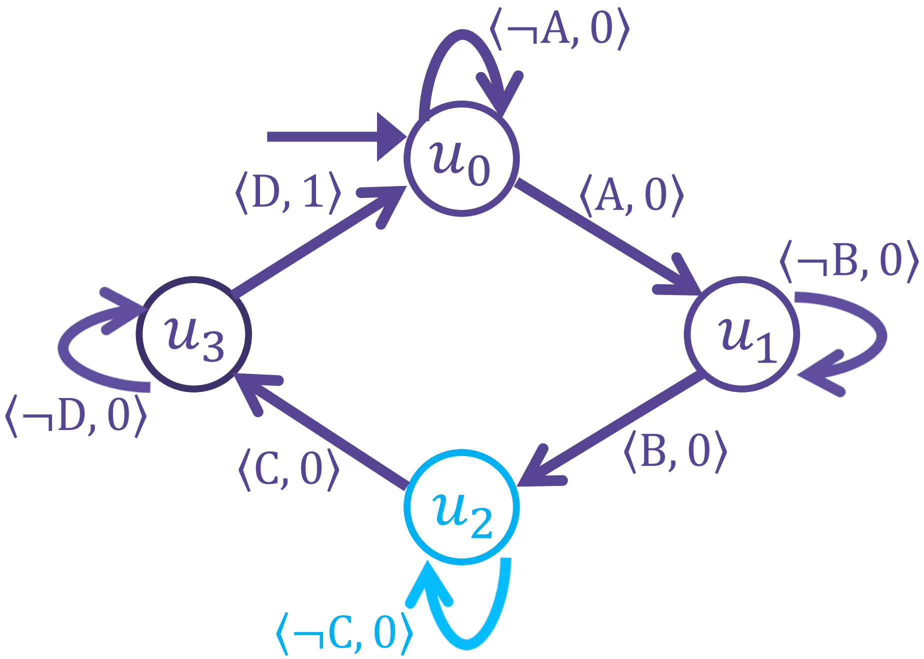
Reward Machines in Action



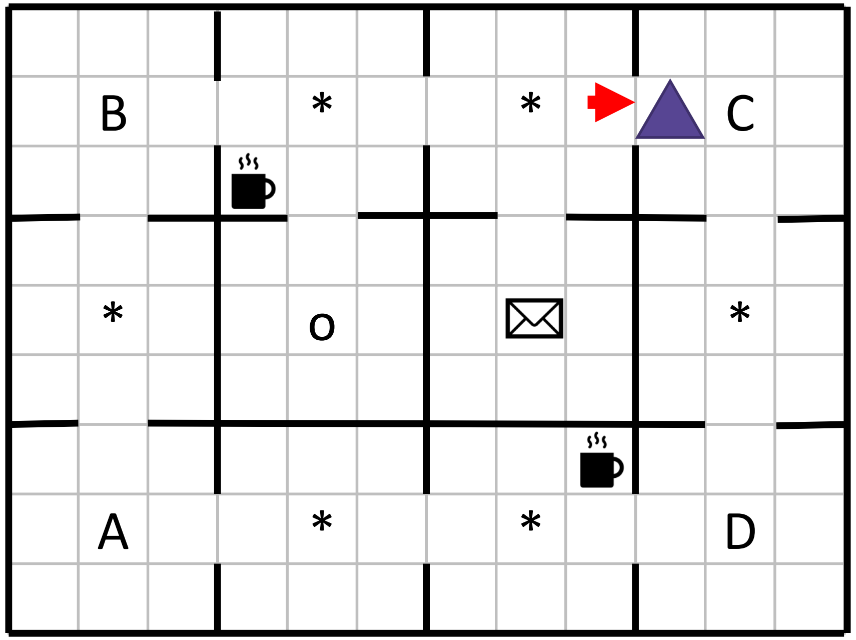


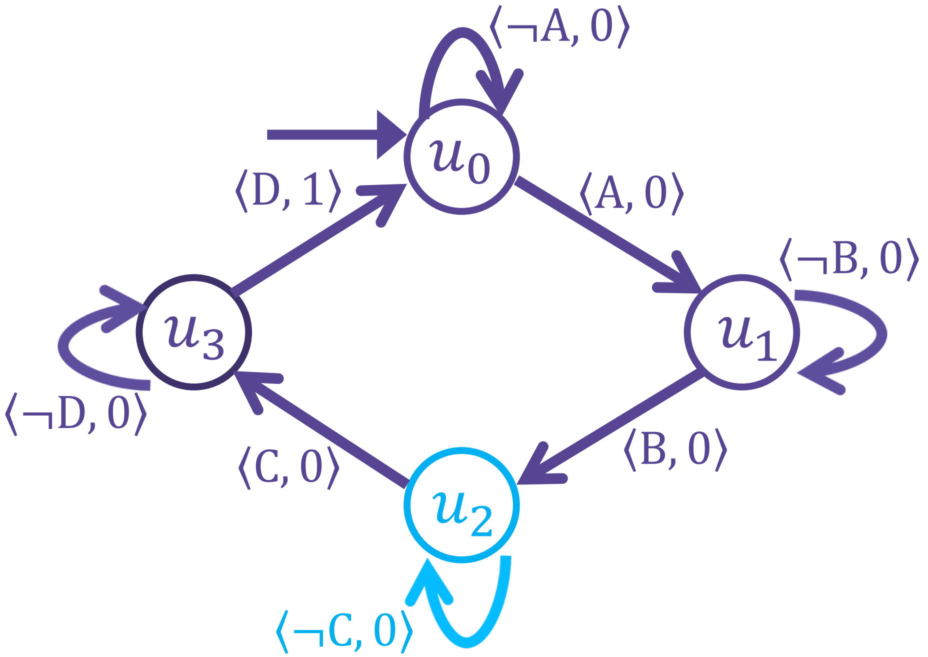
Reward Machines in Action



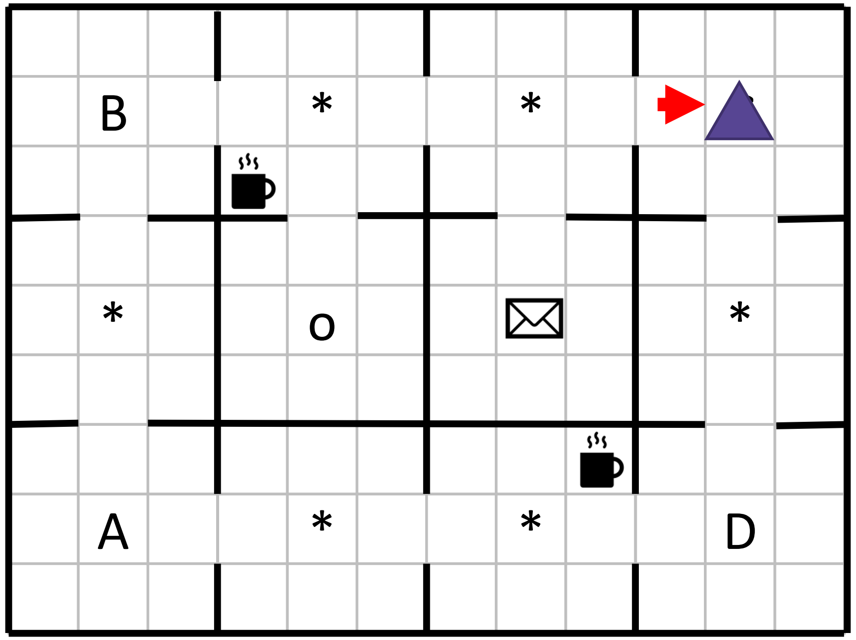


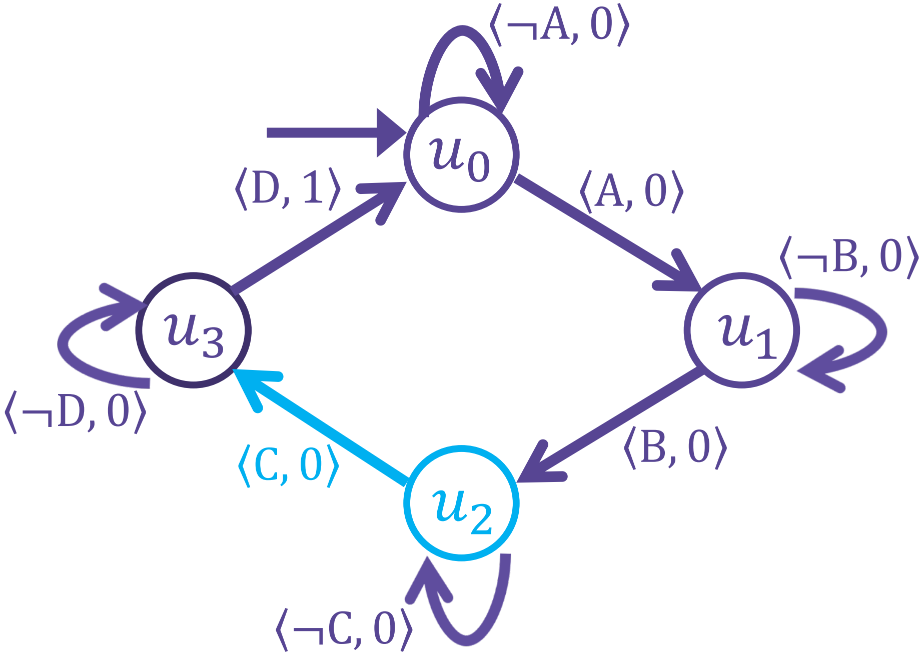
Reward Machines in Action



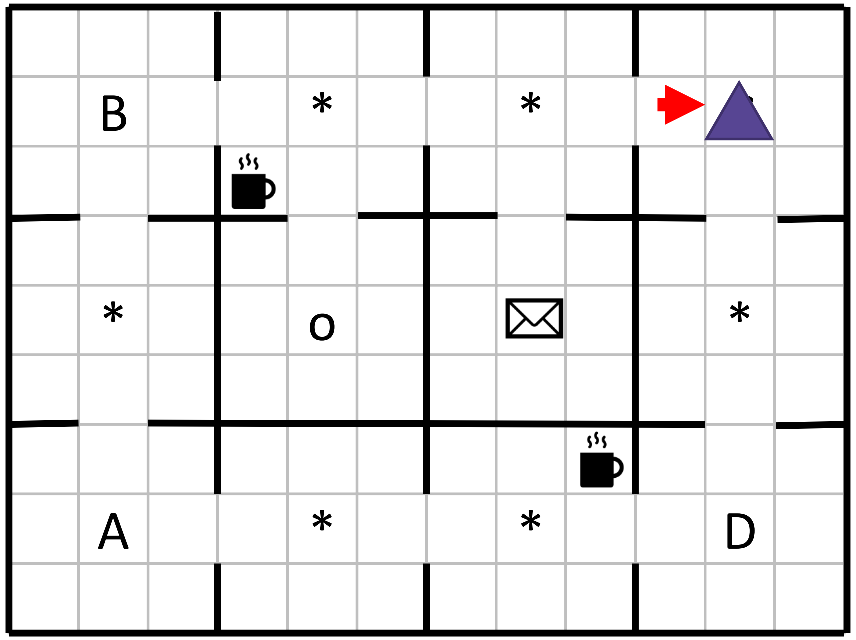


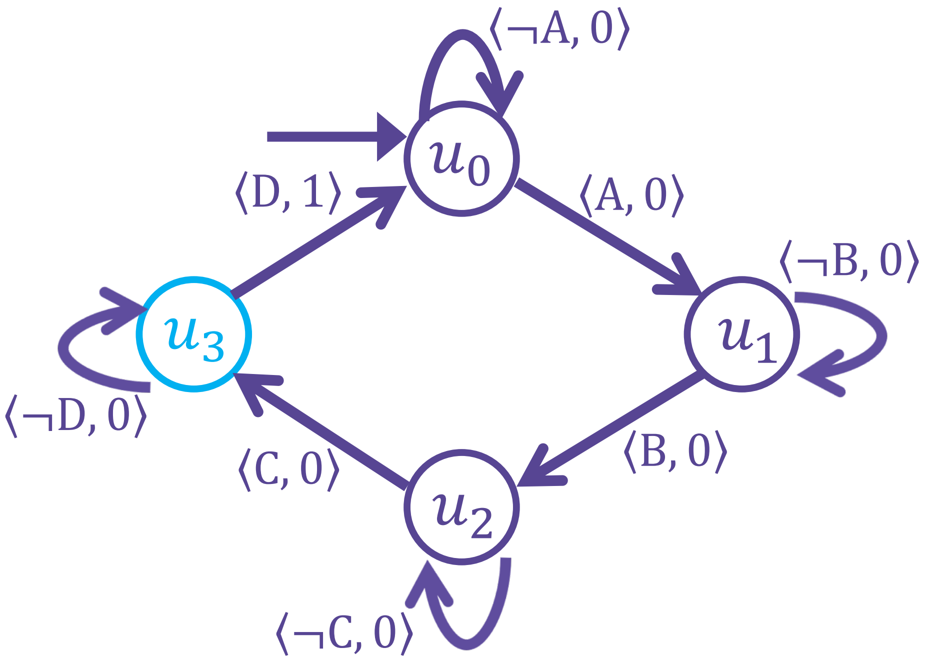
Reward Machines in Action



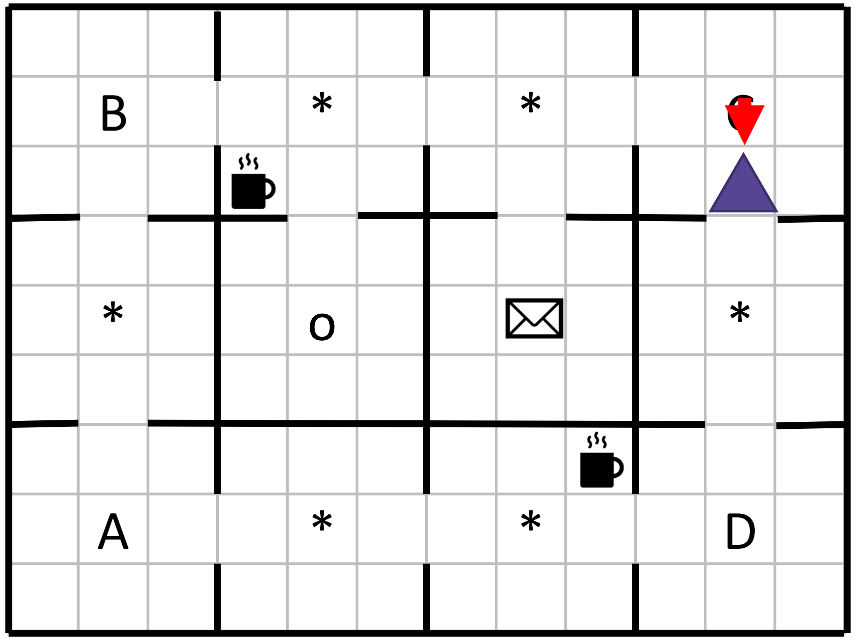


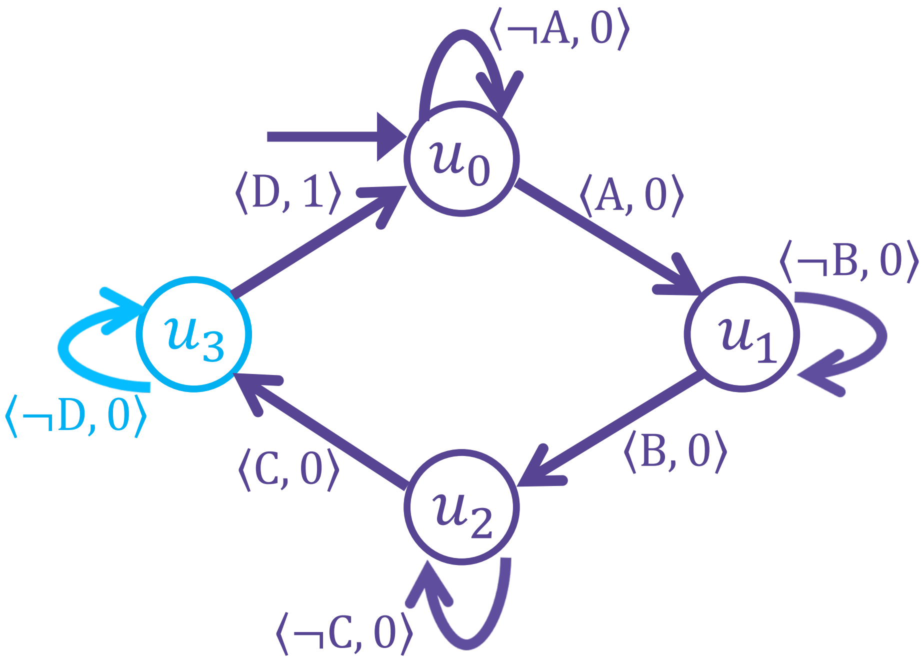
Reward Machines in Action



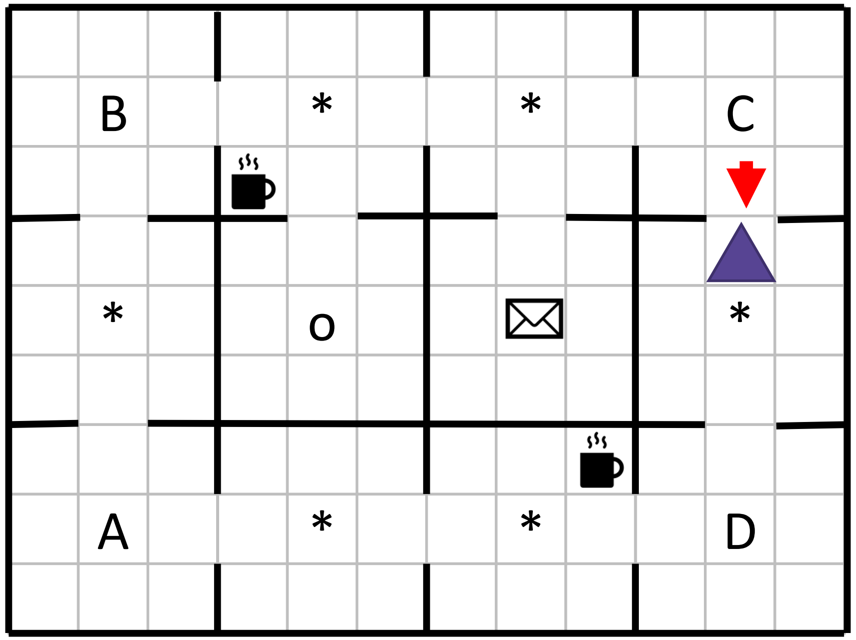


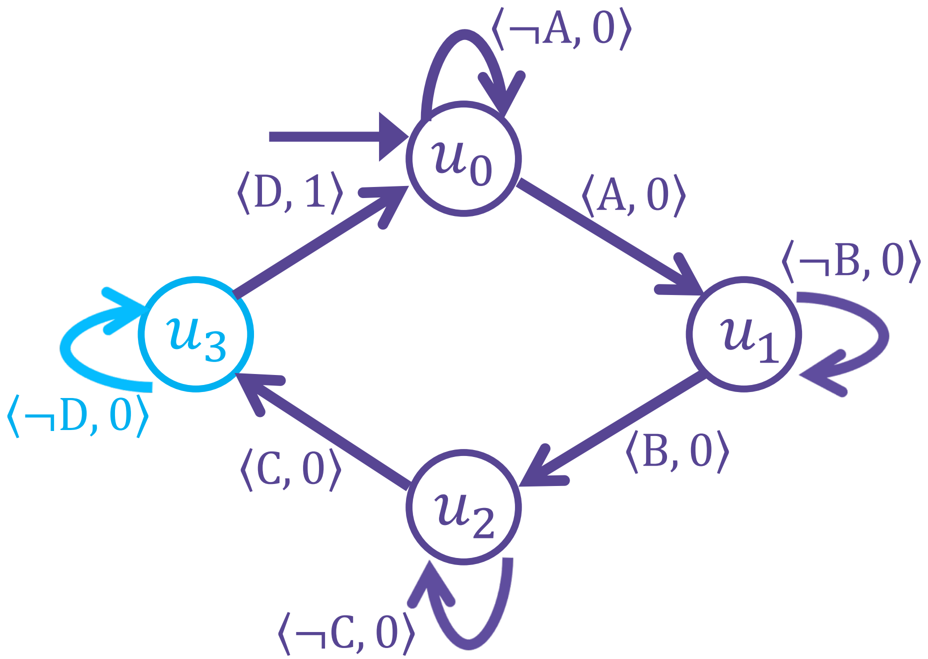
Reward Machines in Action



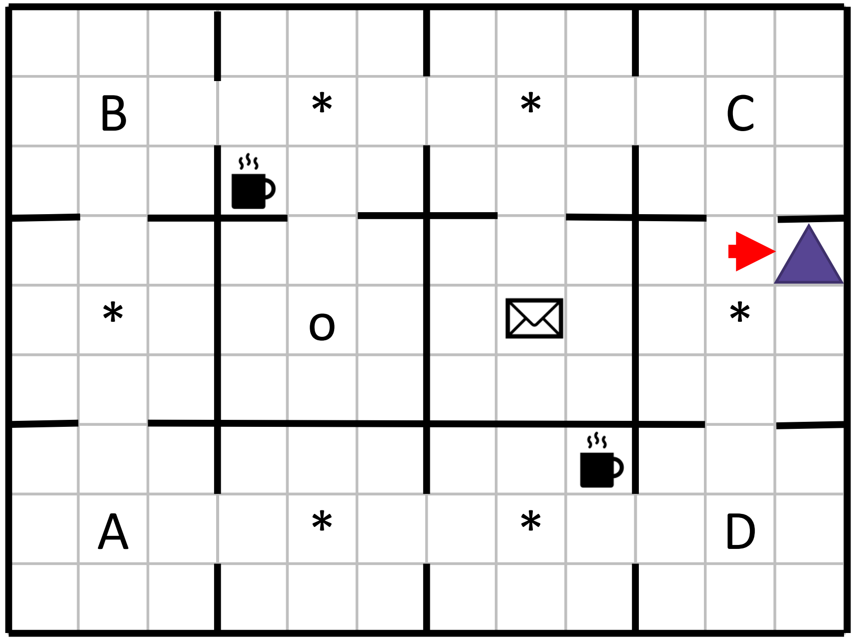


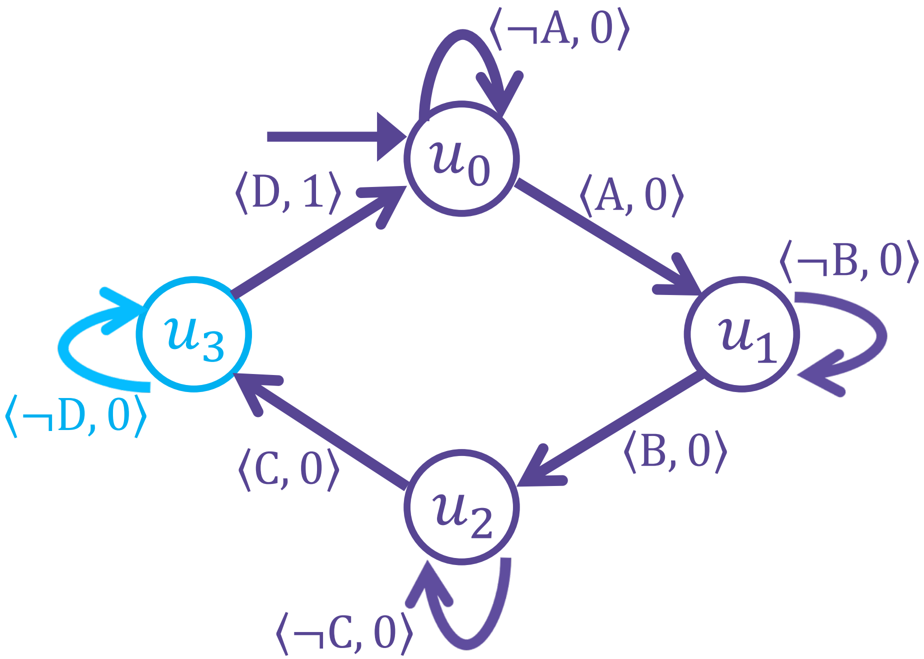
Reward Machines in Action



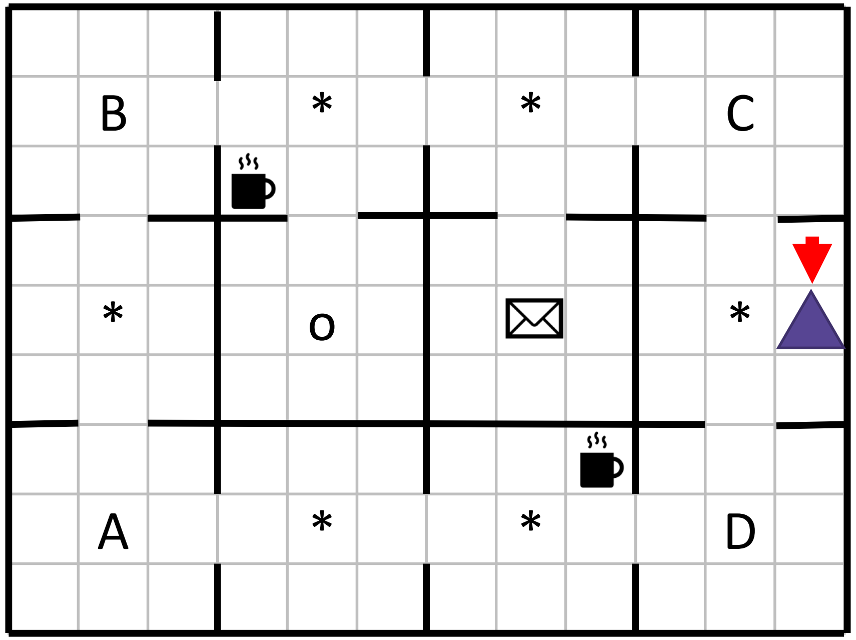


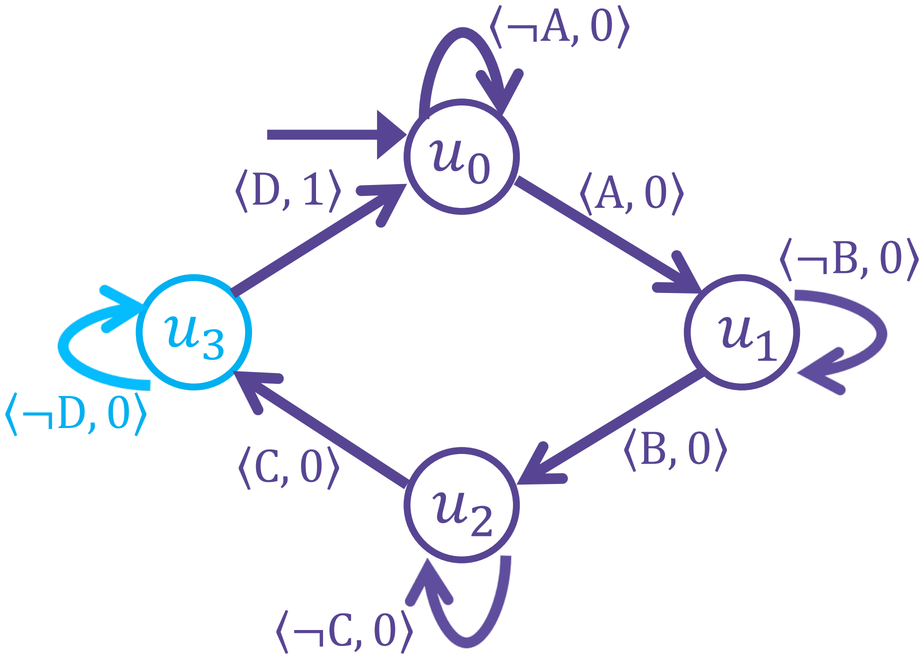
Reward Machines in Action



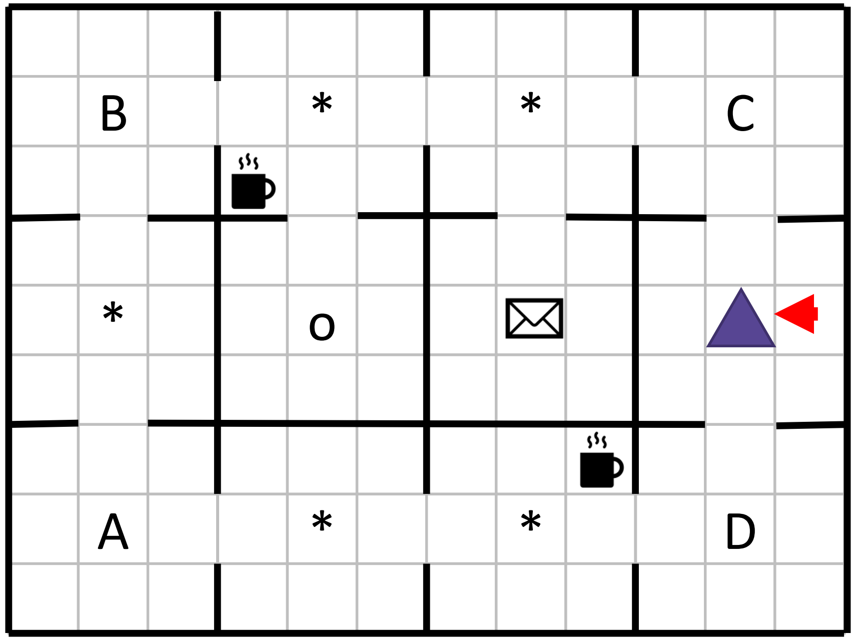


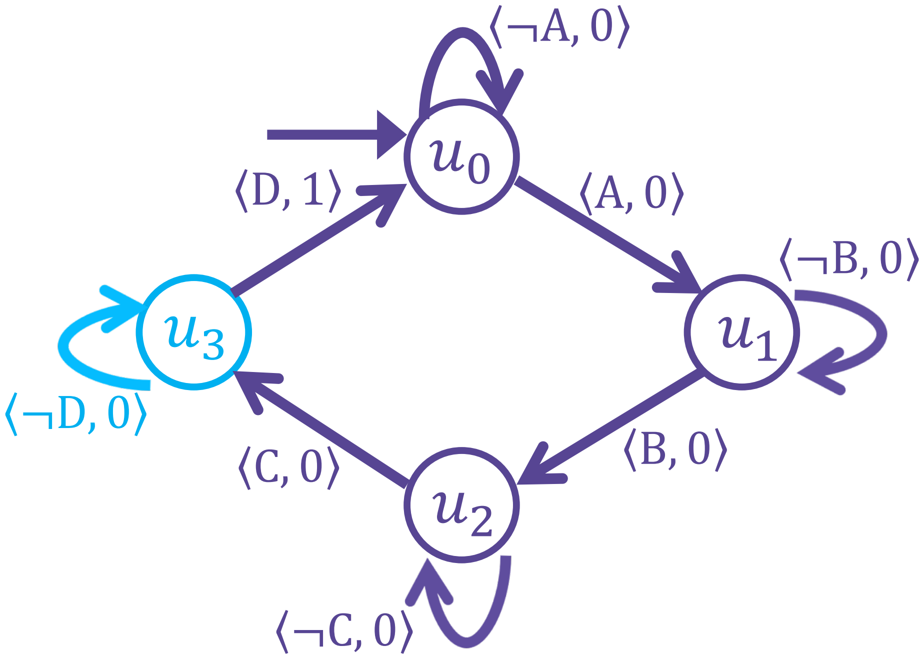
Reward Machines in Action



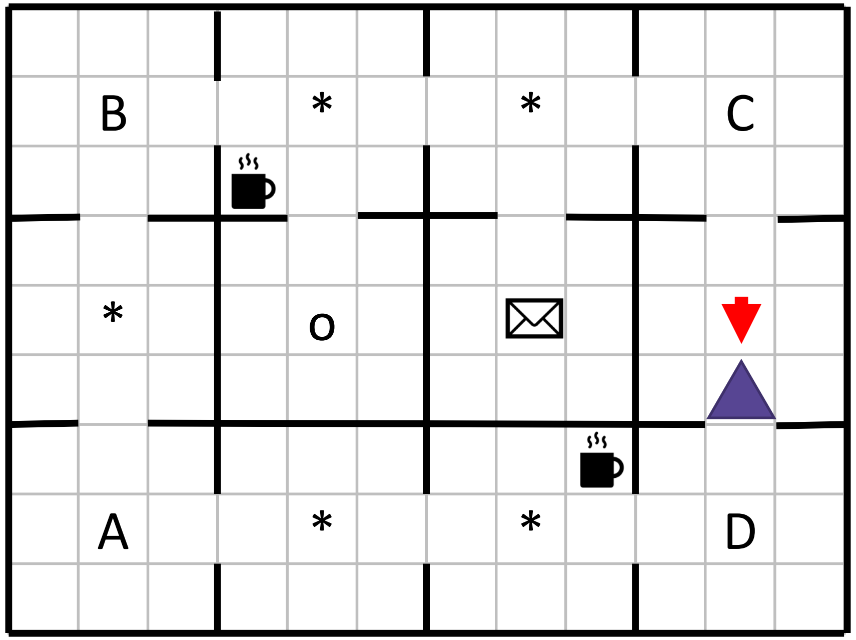

$\langle \neg A, 0\rangle$
u_0
$\langle D, 1\rangle$
$\langle A, 0\rangle$
$\langle \neg B, 0\rangle$
u_3
u_1
$\langle \neg D, 0\rangle$
$\langle C, 0\rangle$
$\langle B, 0\rangle$
u_2
$\langle \neg C, 0\rangle$

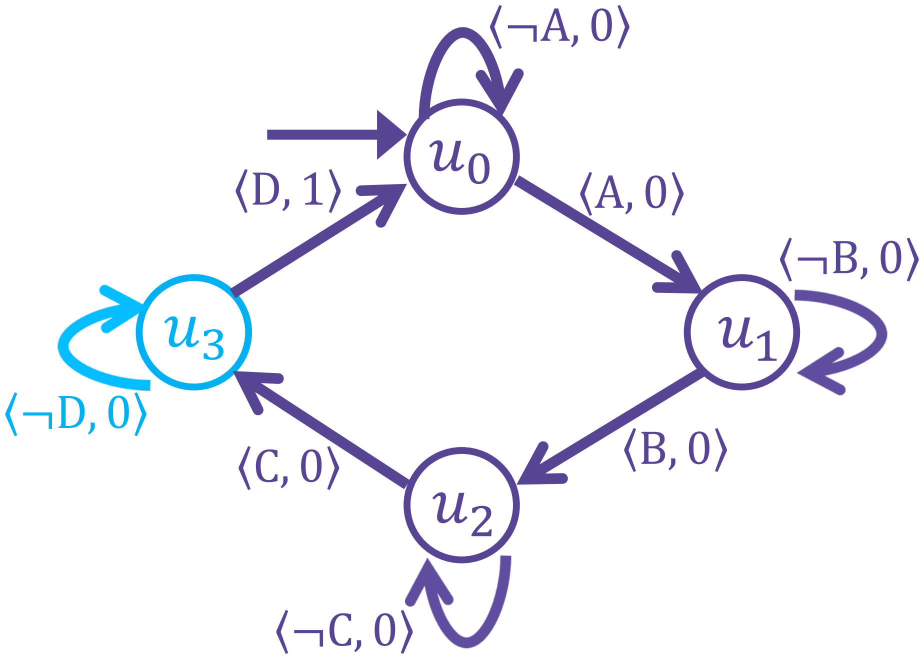
Reward Machines in Action



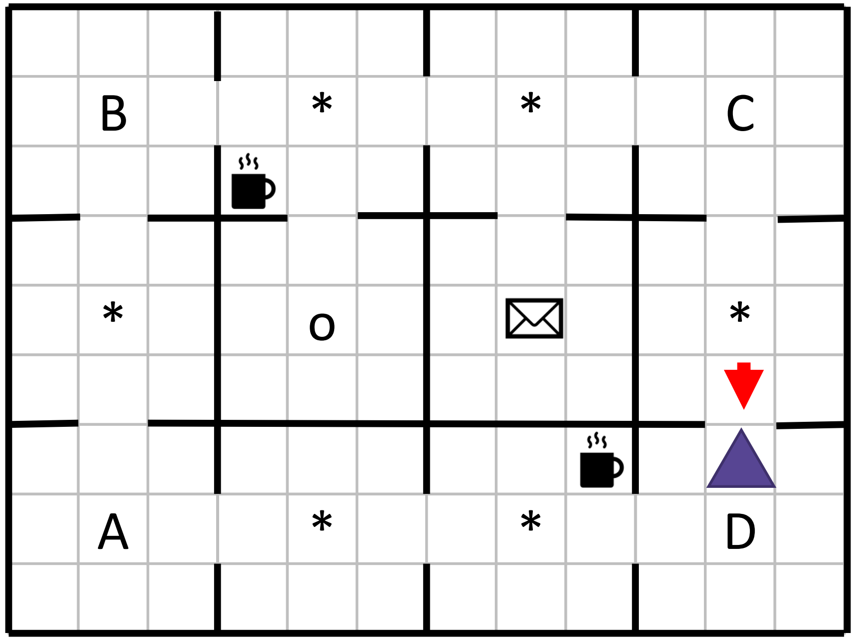

$\langle \neg A, 0 \rangle$
u_0
$\langle D, 1 \rangle$
$\langle A, 0 \rangle$
$\langle \neg B, 0 \rangle$
u_3
u_1
$\langle \neg D, 0 \rangle$
$\langle B, 0 \rangle$
$\langle C, 0 \rangle$
u_2
$\langle \neg C, 0 \rangle$

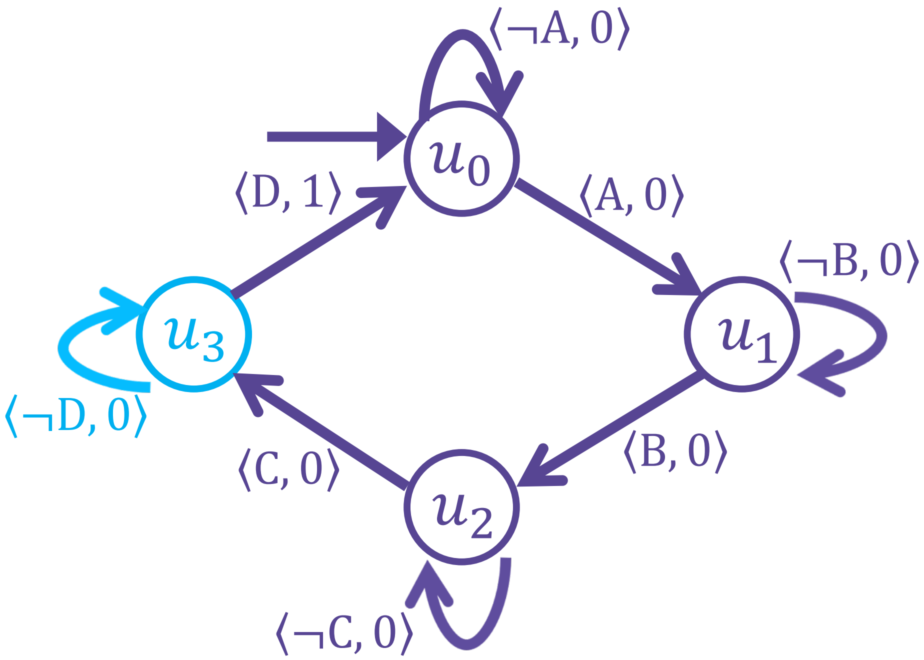
Reward Machines in Action



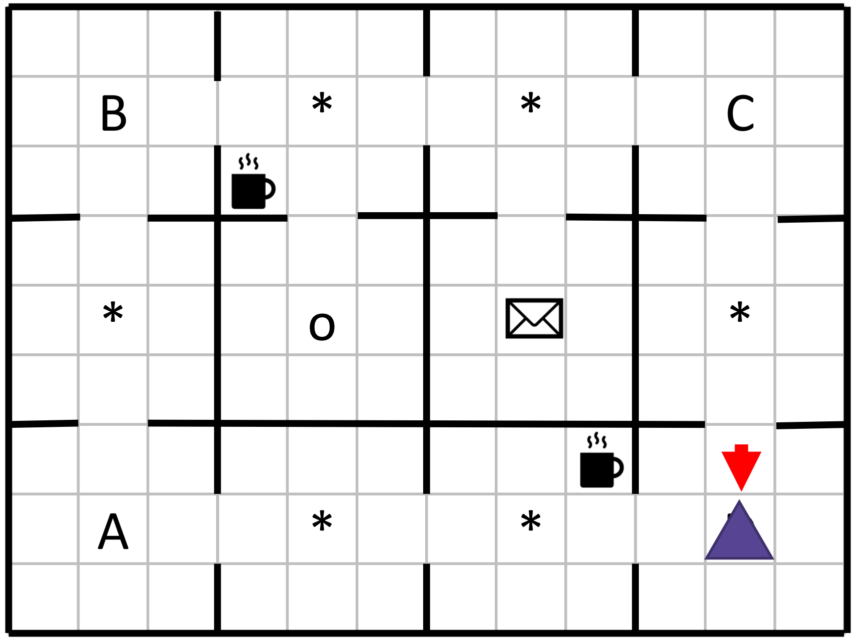


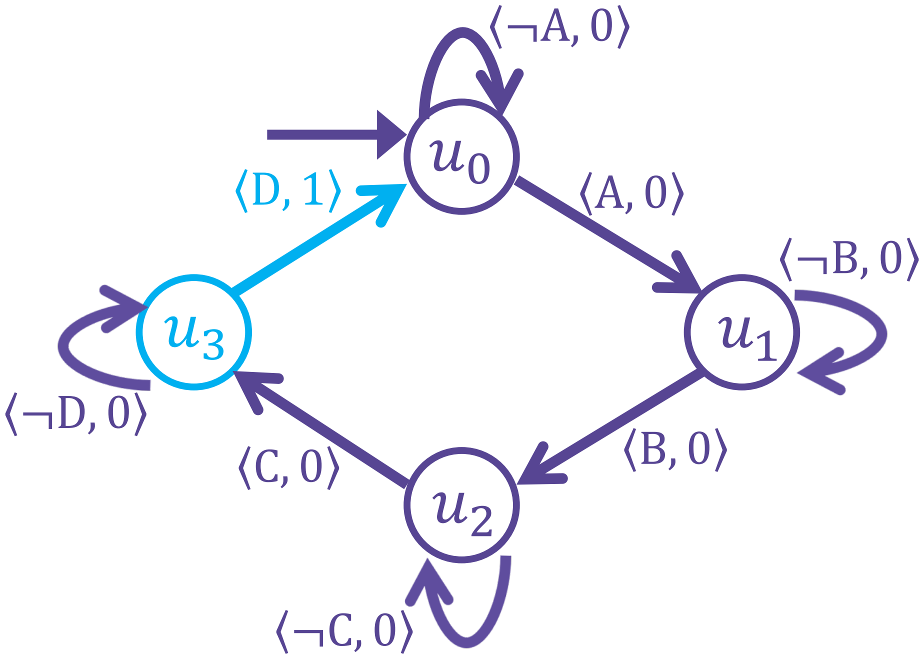
Reward Machines in Action



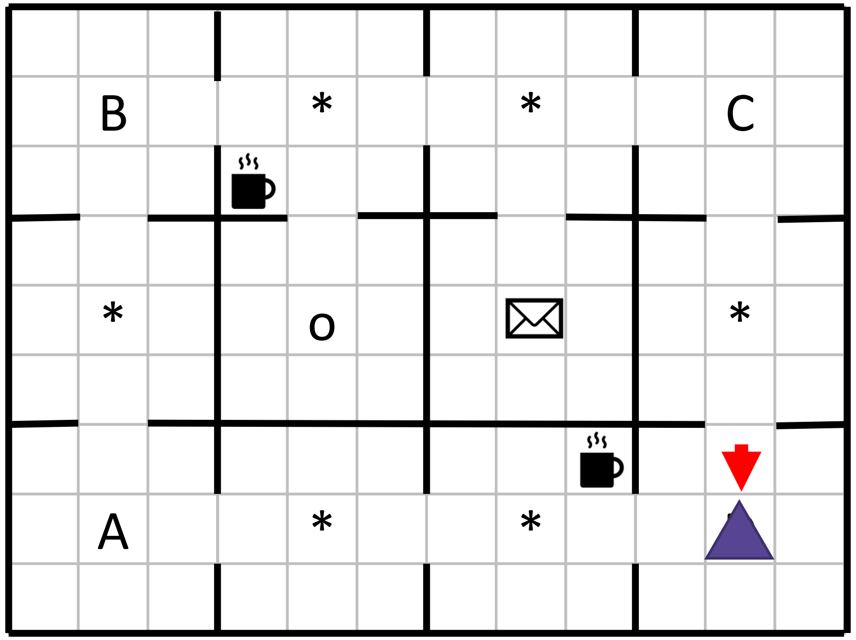


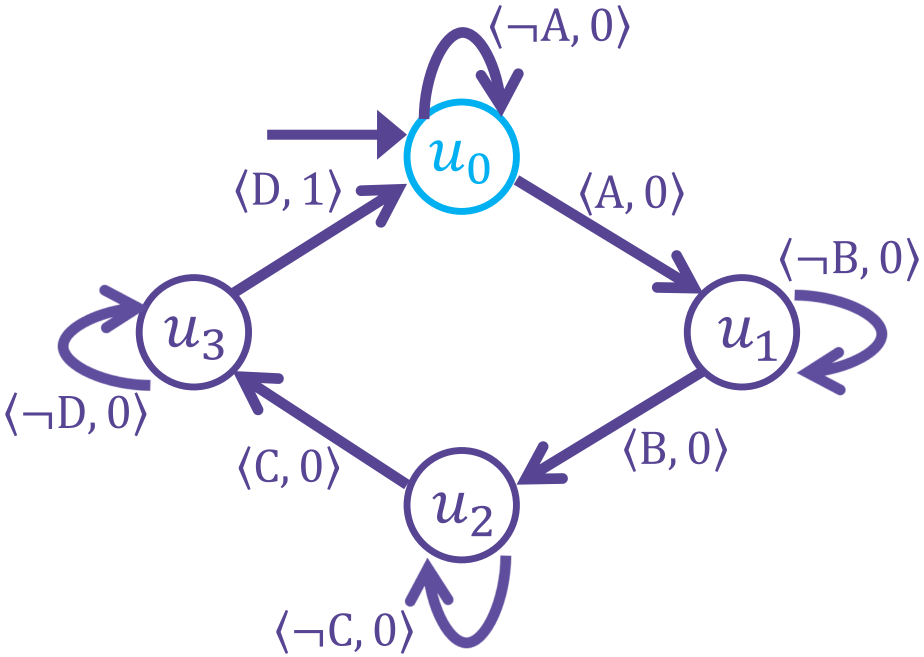
Reward Machines in Action





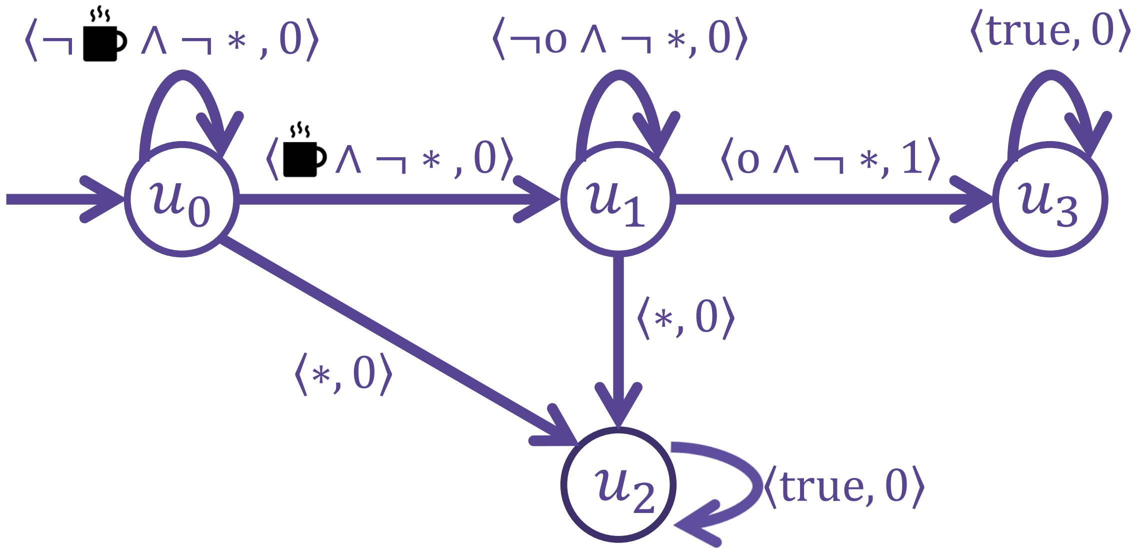
Reward Machines in Action





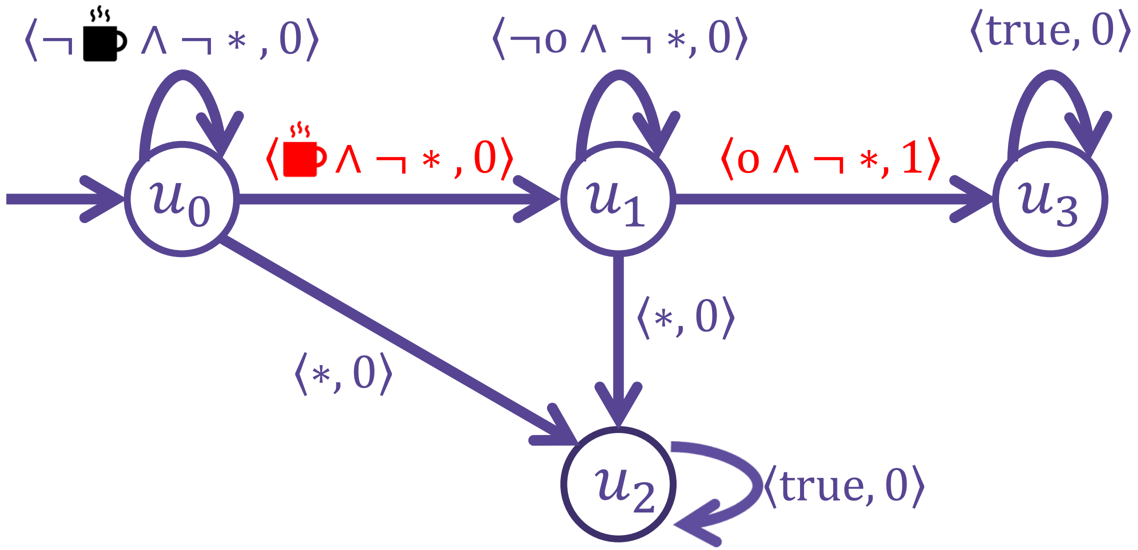
Other Reward Machines

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



Other Reward Machines

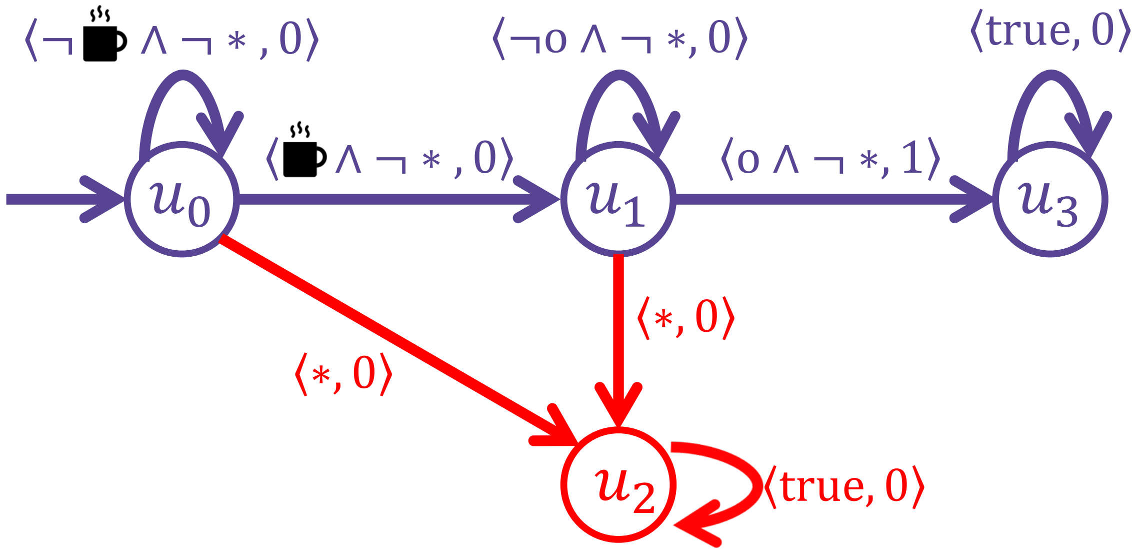
Task: Deliver coffee to the office, while avoiding furniture.



$\langle \neg ☕ \wedge \neg *, 0 \rangle$
$\langle ☕ \wedge \neg *, 0 \rangle$
$\langle \neg o \wedge \neg *, 0 \rangle$
$\langle o \wedge \neg *, 1 \rangle$
$\langle \text{true}, 0 \rangle$
u_0
u_1
u_3
$\langle *, 0 \rangle$
$\langle *, 0 \rangle$
u_2
$\langle \text{true}, 0 \rangle$

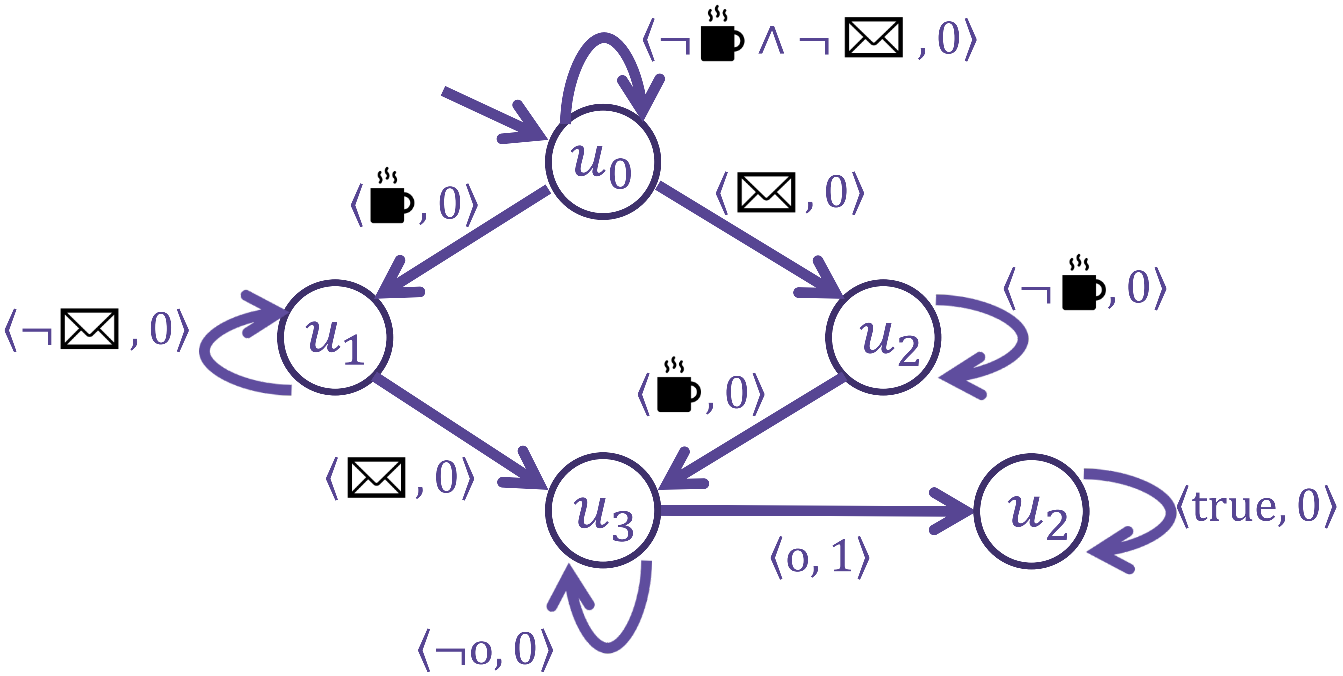
Other Reward Machines

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



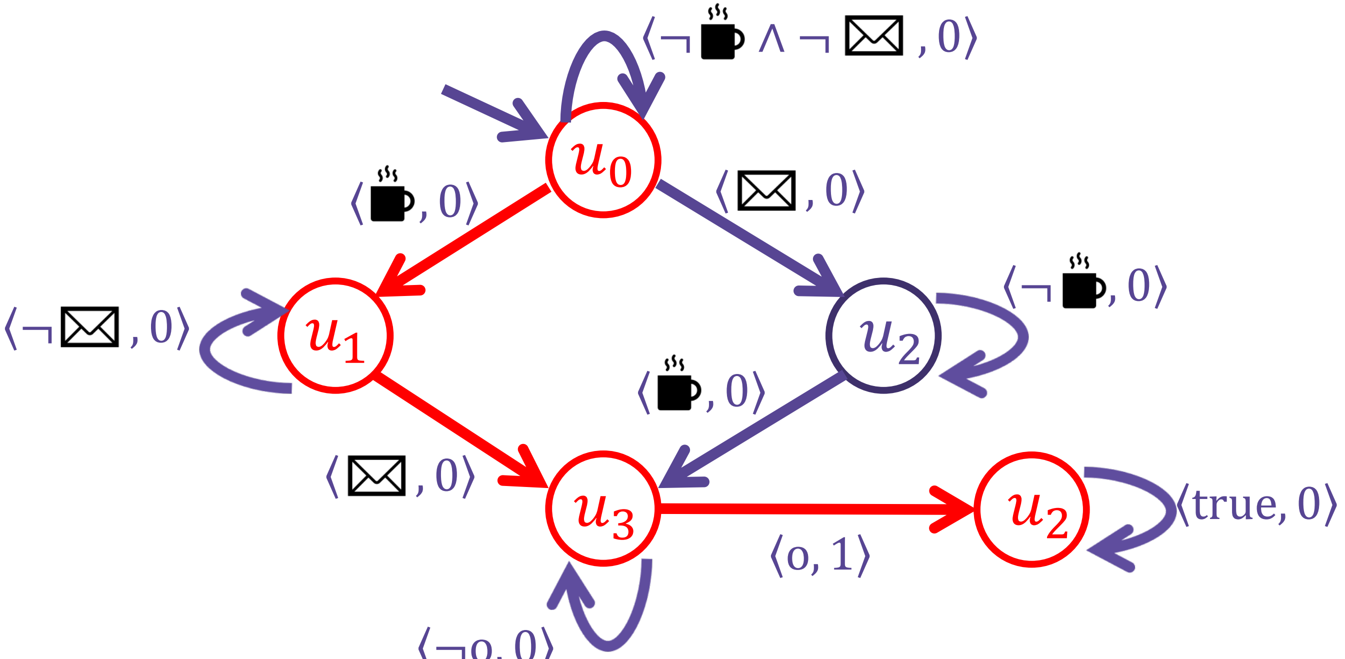
Other Reward Machines

Task: Deliver coffee and mail to the office.



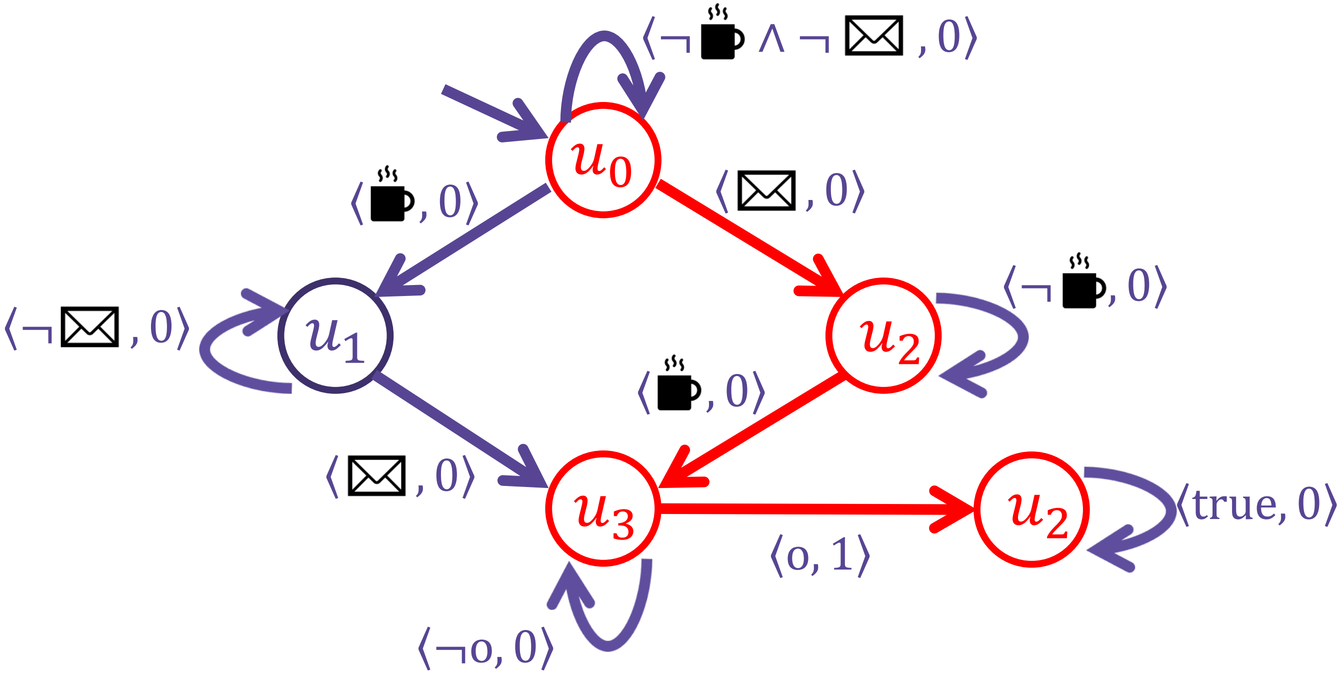
Other Reward Machines

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

Methods for Exploiting RM Structure

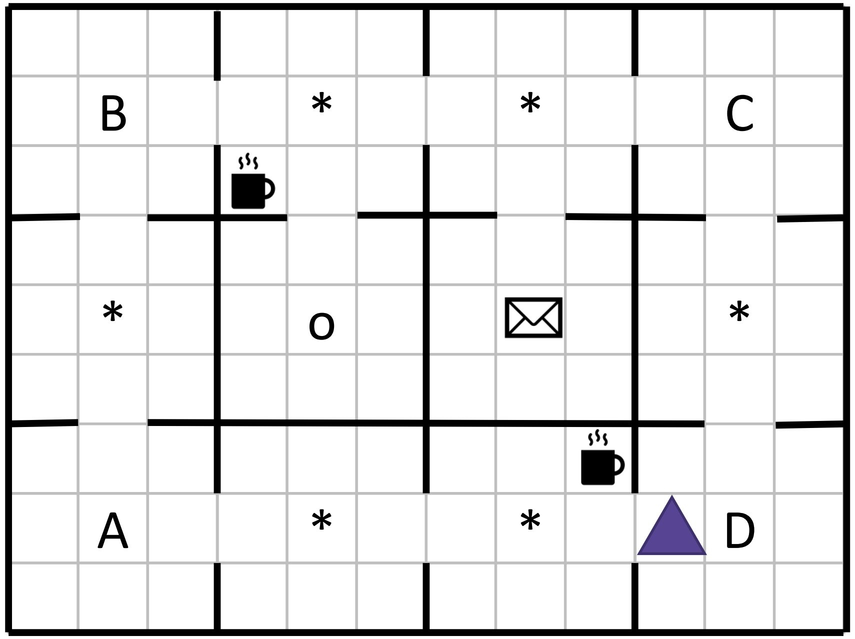
Baselines based on existing methods:

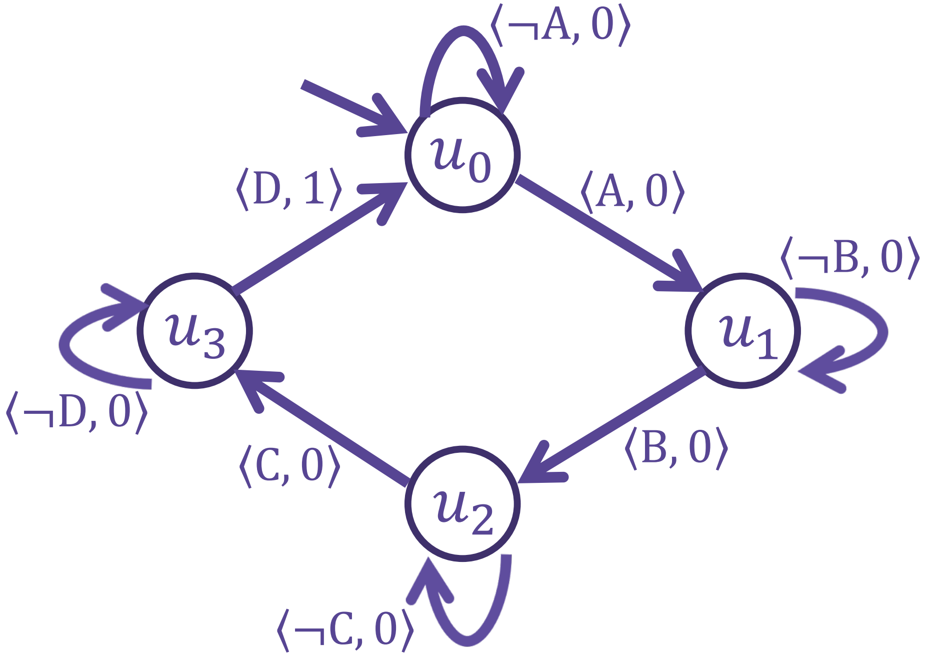
1. Q-learning over an equivalent MDP (Q-learning)
2. Hierarchical RL based on options (HRL)
3. HRL with RM-based pruning (HRL-RM)

Our approaches:

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

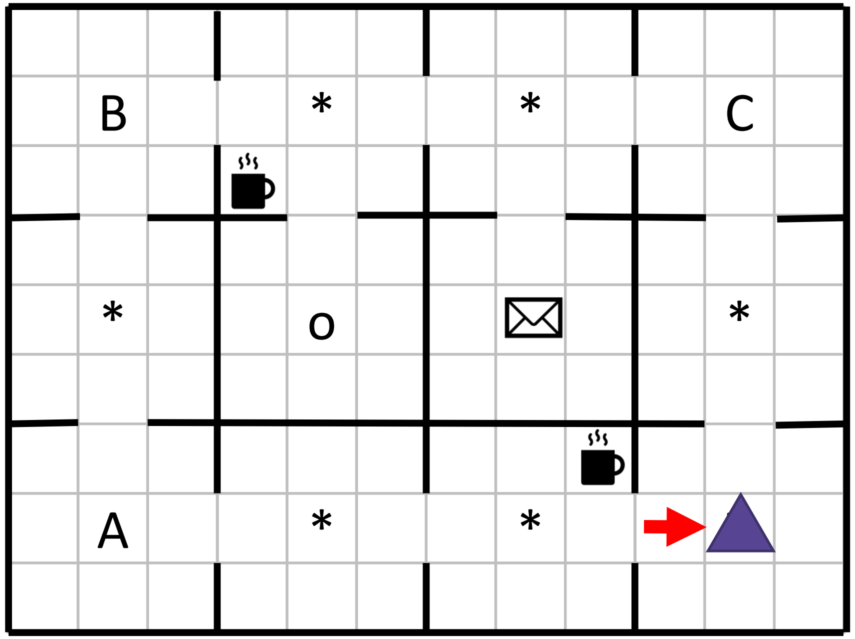
1. Q-Learning Baseline

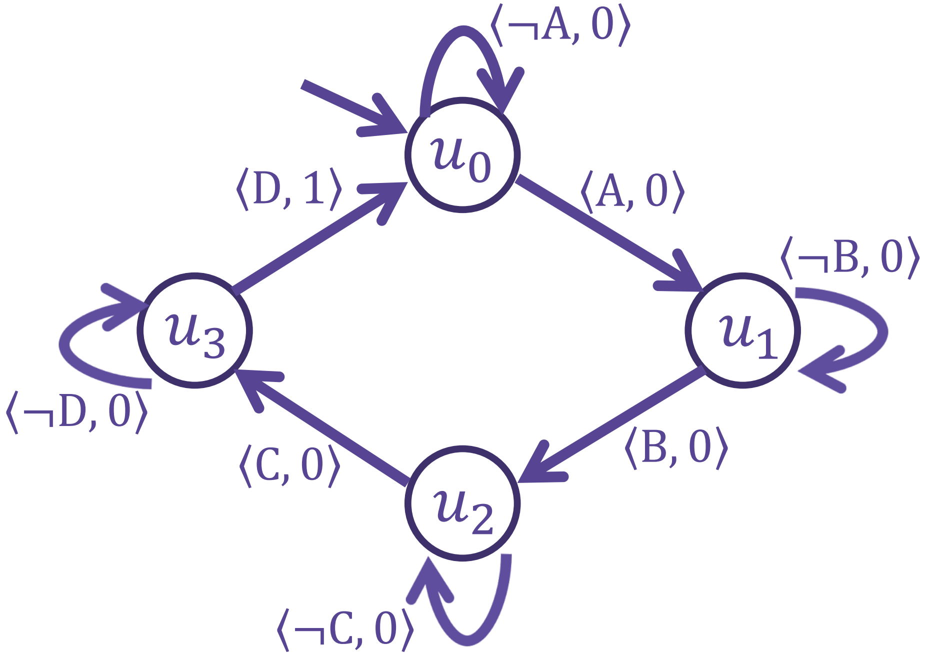




A Reward Machine may define a non-Markovian reward function.

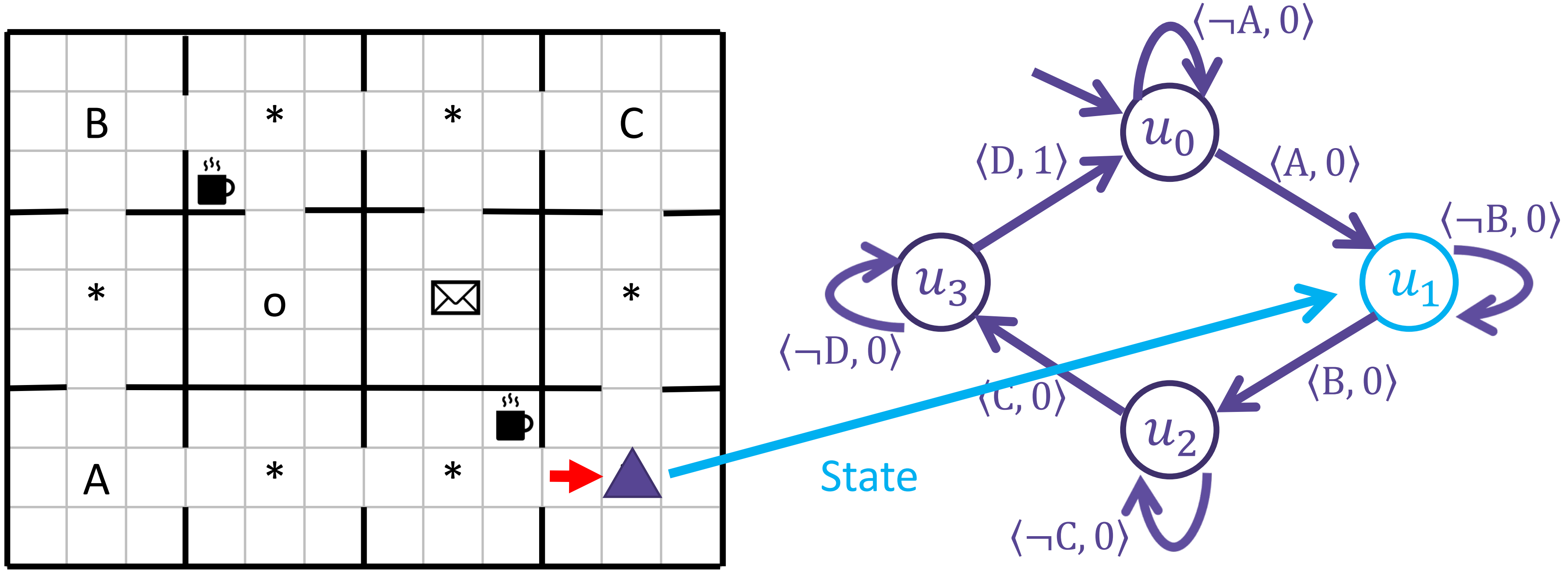
1. Q-Learning Baseline





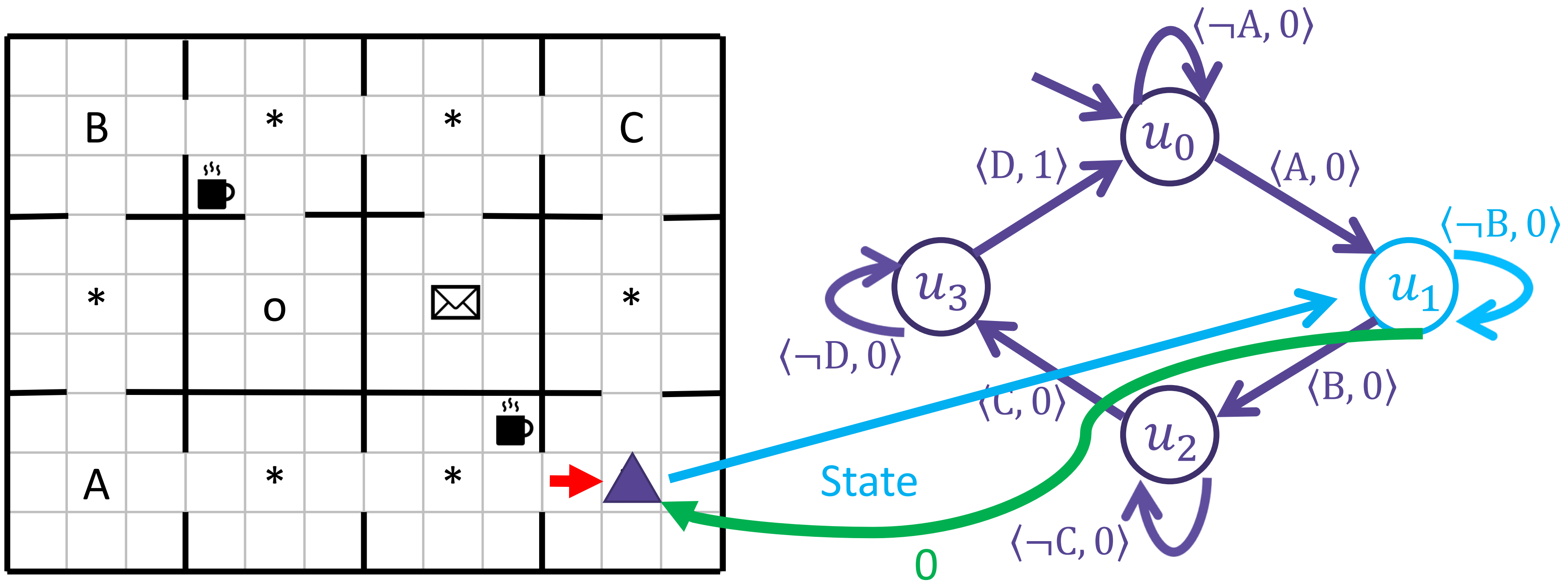
A Reward Machine may define a non-Markovian reward function.

1. Q-Learning Baseline



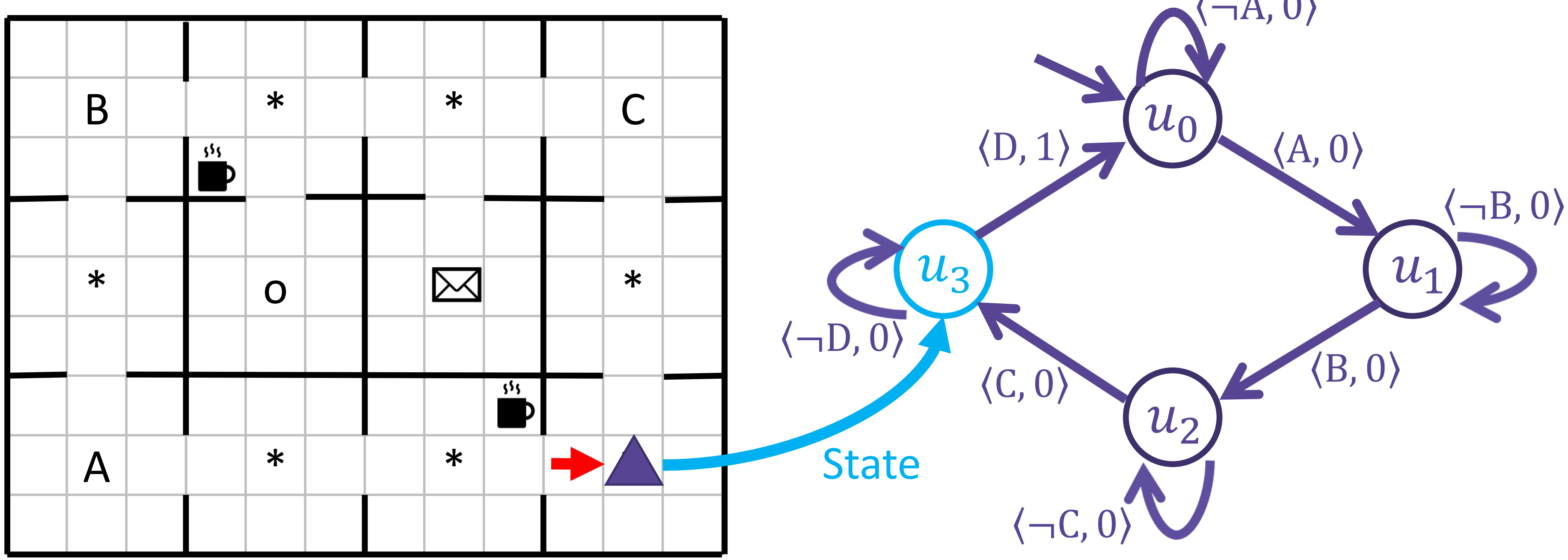
A Reward Machine may define a non-Markovian reward function.

1. Q-Learning Baseline



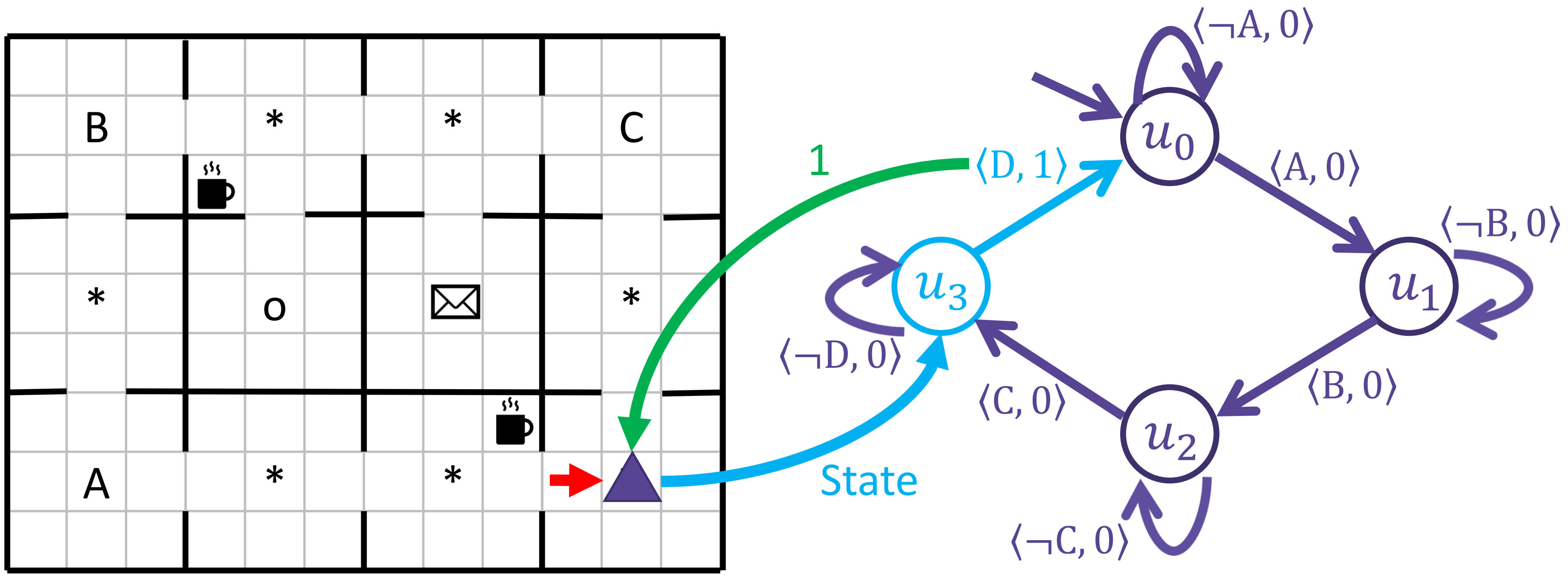
A Reward Machine may define a non-Markovian reward function.

1. Q-Learning Baseline



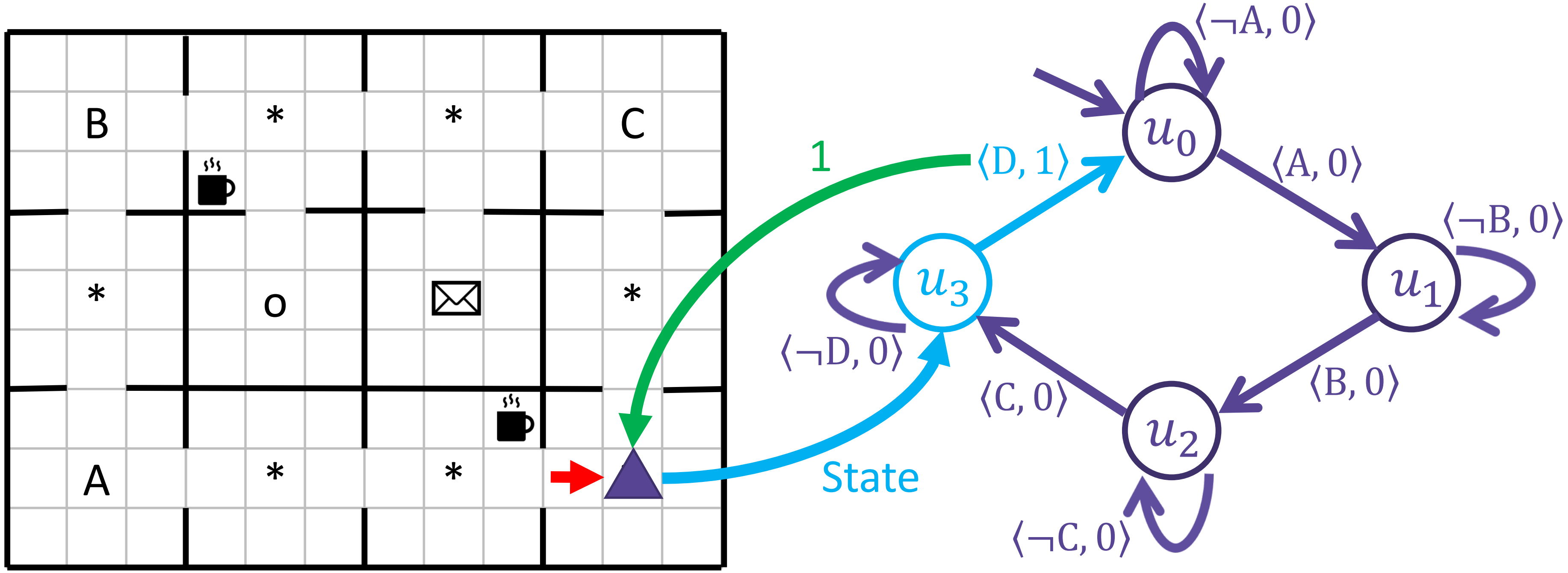
A Reward Machine may define a non-Markovian reward function.

1. Q-Learning Baseline



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

1. Q-Learning Baseline

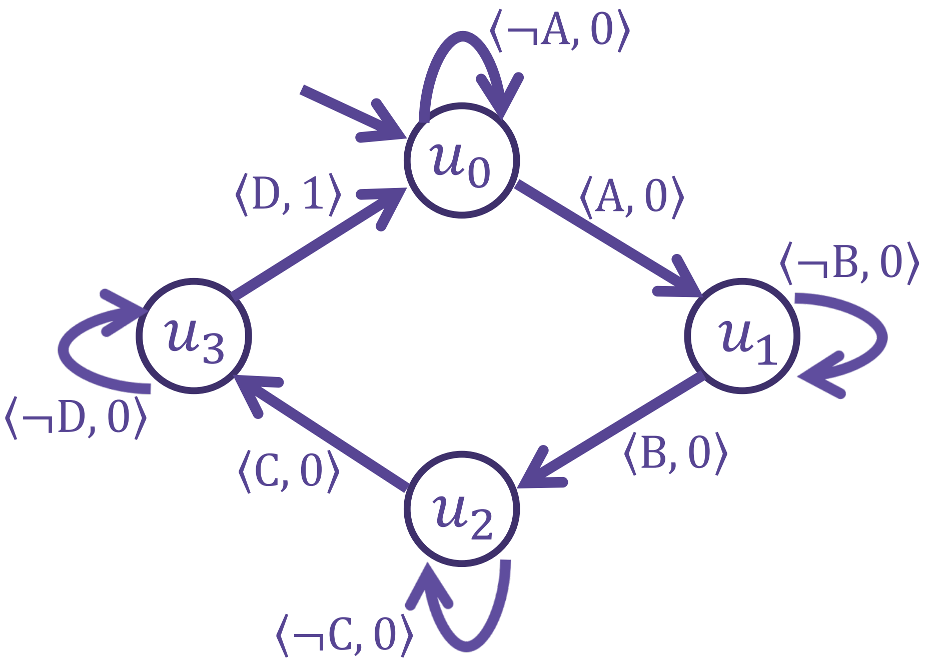


Solution: Include RM state as part of agent’s state representation.

Use standard Q-learning on resulting MDP.

2. Option-Based Hierarchical RL (HRL)

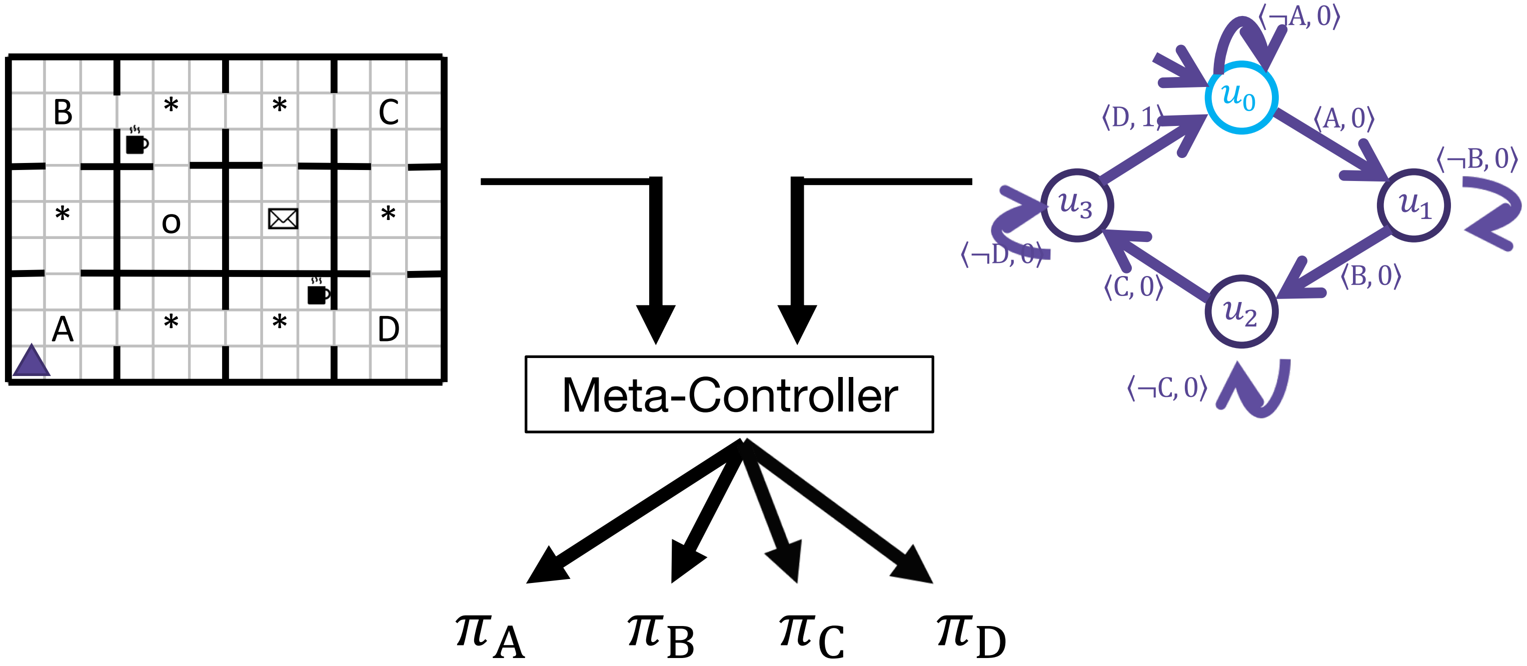
Learn one **option policy** for each proposition mentioned in the RM


$\langle \neg A, 0\rangle$
u_0
$\langle D, 1\rangle$
$\langle A, 0\rangle$
$\langle \neg B, 0\rangle$
u_3
u_1
$\langle \neg D, 0\rangle$
$\langle B, 0\rangle$
$\langle C, 0\rangle$
u_2
$\langle \neg C, 0\rangle$

- 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

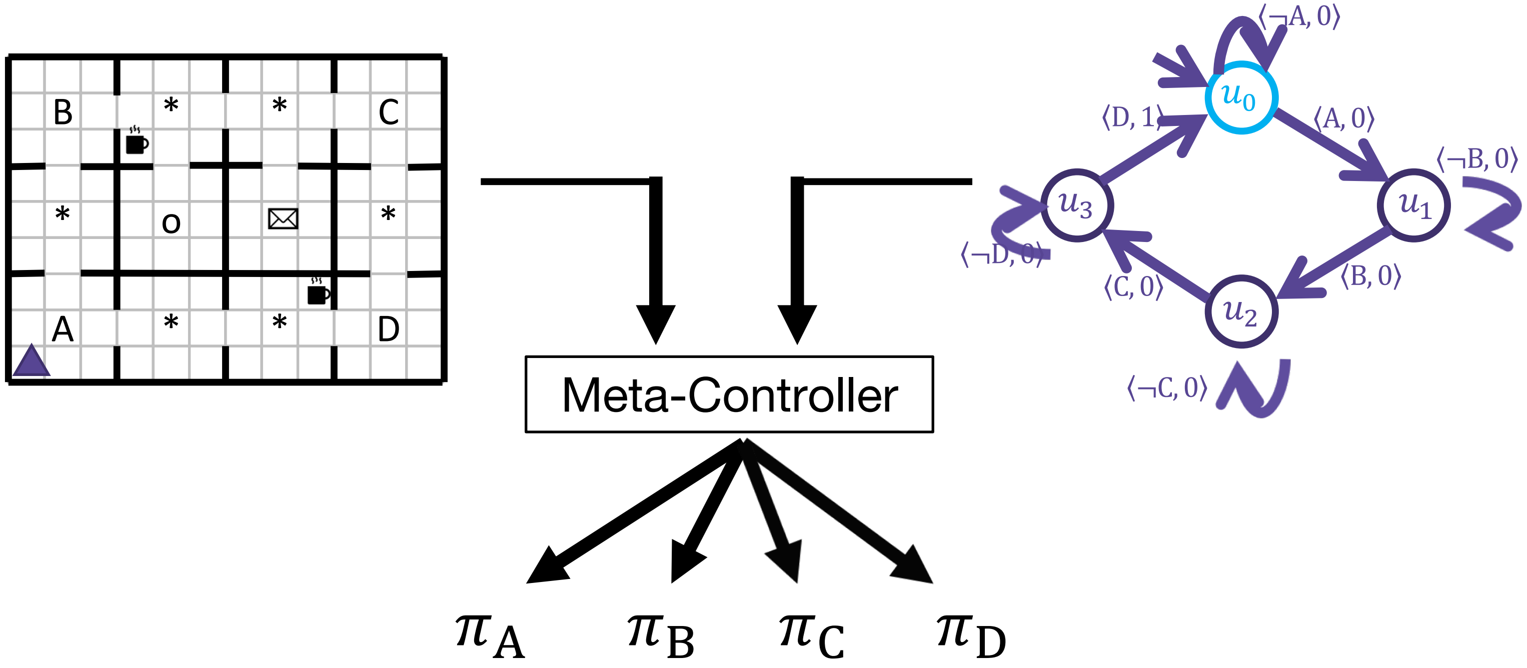
2. Option-Based Hierarchical RL (HRL)

Simultaneously learn when to use each option policy


B
*
*
C
*
o
*
A
*
*
D
Meta-Controller
π_{A}
π_{B}
π_{C}
π_{D}
$\langle \neg\mathrm{A}, 0\rangle$
u_0
$\langle \mathrm{D}, 1\rangle$
$\langle \mathrm{A}, 0\rangle$
$\langle \neg\mathrm{B}, 0\rangle$
u_3
u_1
$\langle \neg\mathrm{D}, 0\rangle$
$\langle \mathrm{C}, 0\rangle$
$\langle \mathrm{B}, 0\rangle$
u_2
$\langle \neg\mathrm{C}, 0\rangle$

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

Prune irrelevant options using current RM state

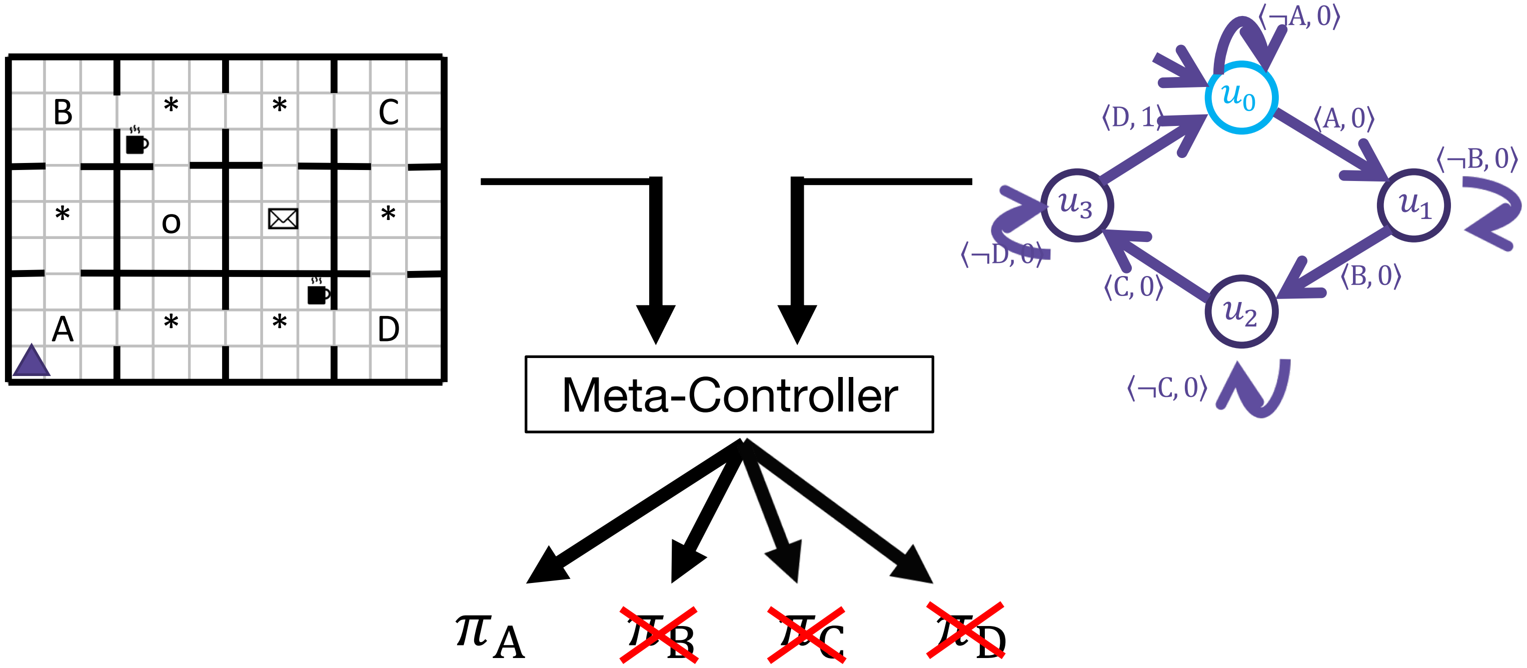


Meta-Controller

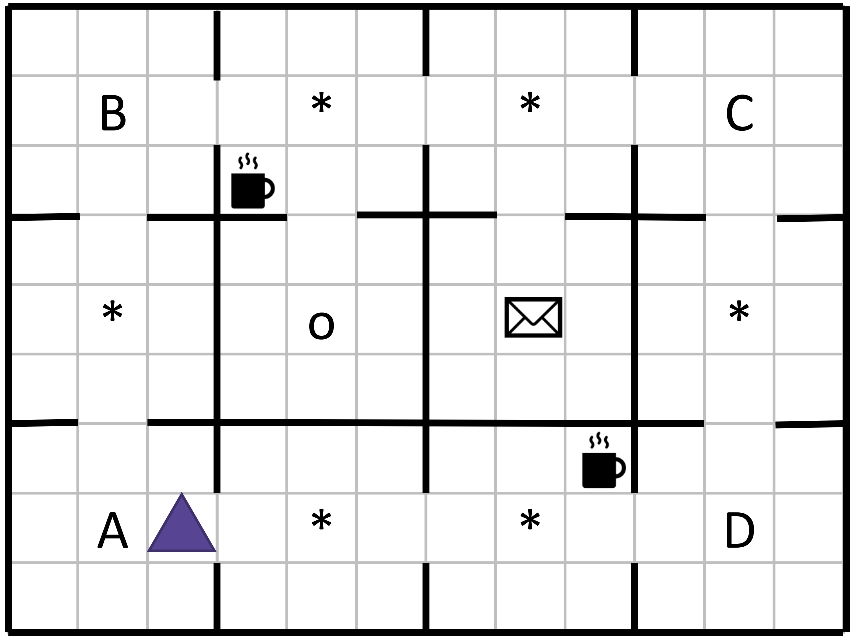
π_A π_B π_C π_D

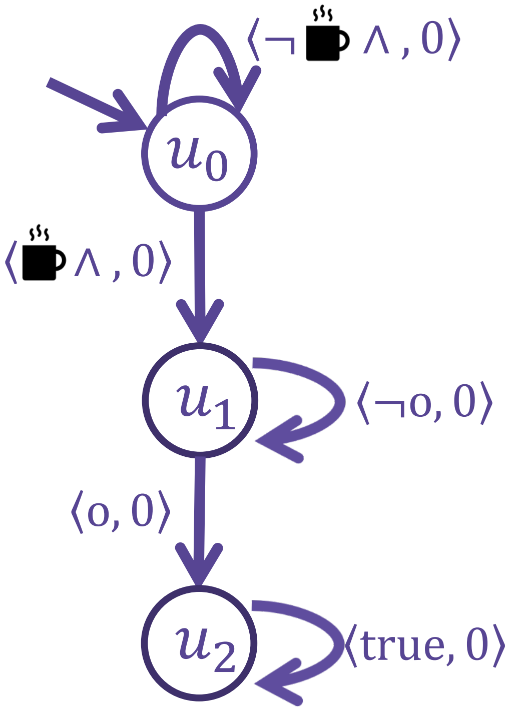
3. HRL with RM-Based Pruning (HRL-RM)

Prune irrelevant options using current RM state



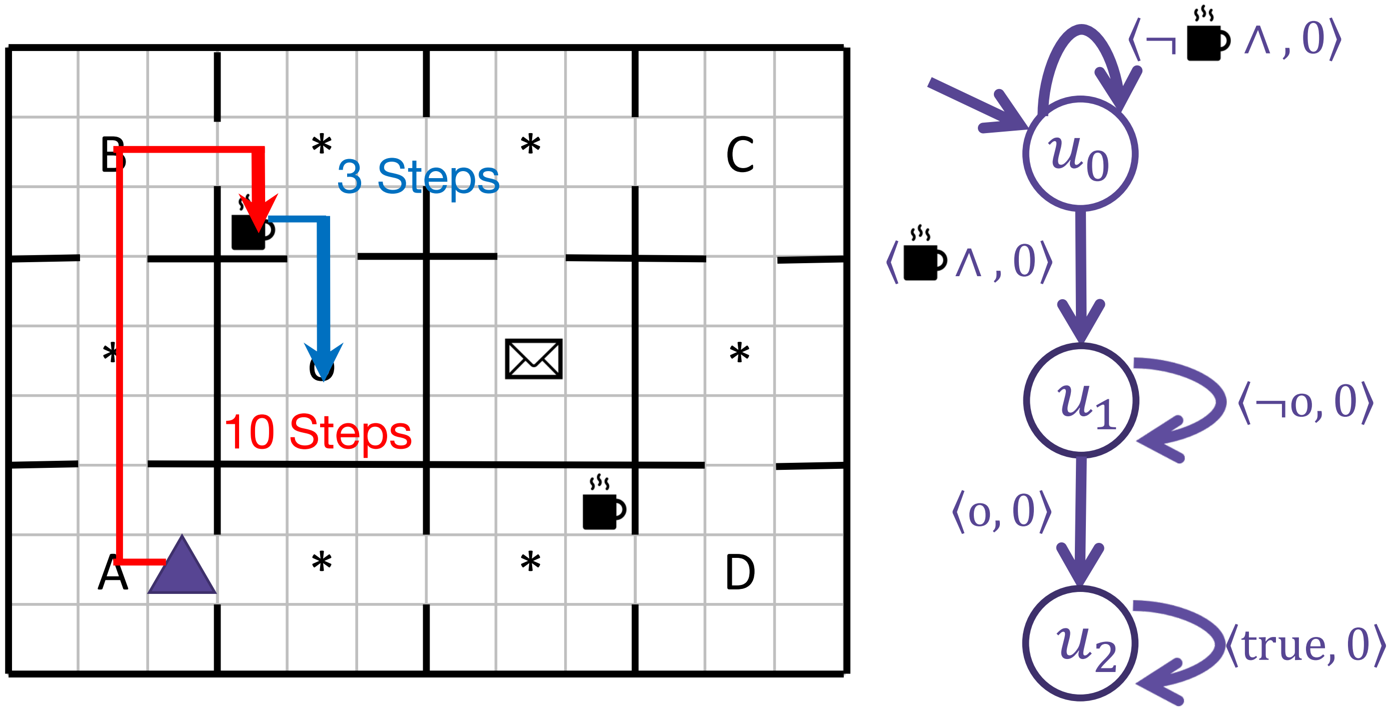
HRL Methods Can Find Suboptimal Policies





HRL approaches find “locally” optimal solutions.

HRL Methods Can Find Suboptimal Policies

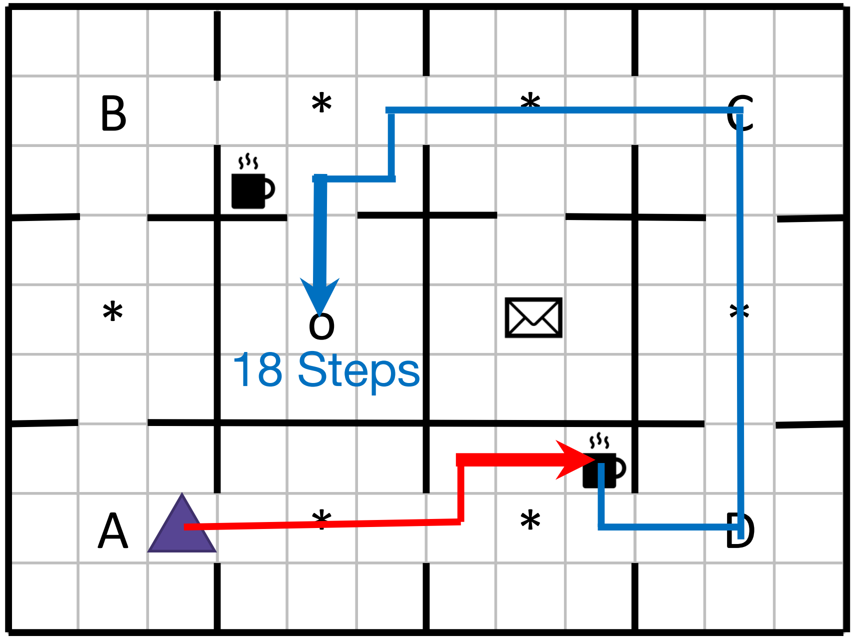


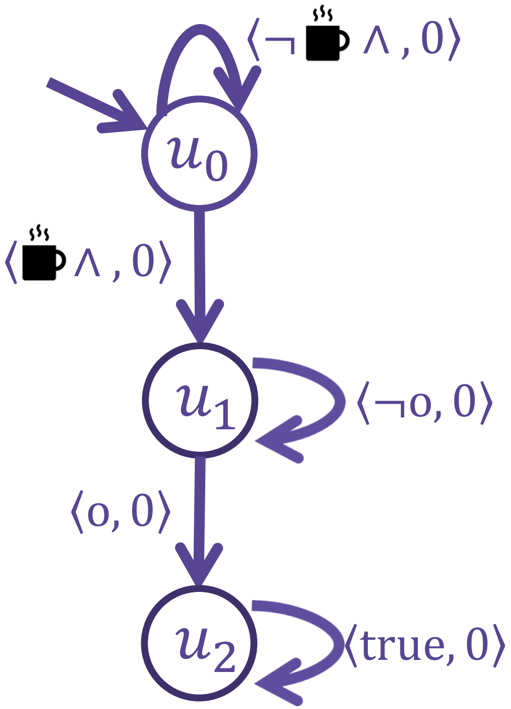
Optimal solution ($\gamma < 1$)

- 13 total steps

HRL approaches find “locally” optimal solutions.

HRL Methods Can Find Suboptimal Policies



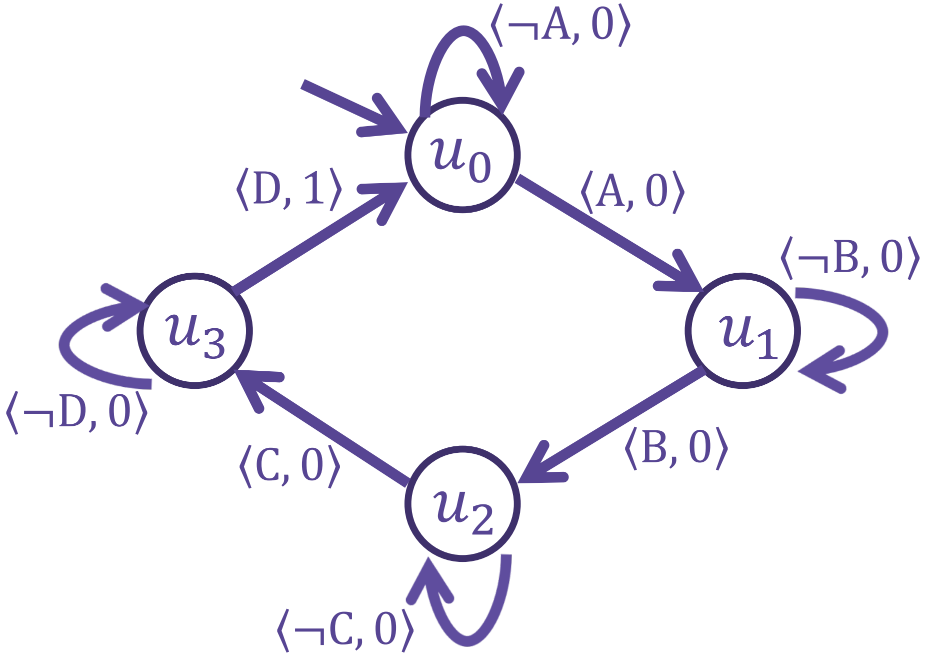

$\langle \neg ☕ \wedge, 0 \rangle$
u_0
$\langle ☕ \wedge, 0 \rangle$
u_1
$\langle \neg o, 0 \rangle$
$\langle o, 0 \rangle$
u_2
$\langle \text{true}, 0 \rangle$

Learns two options:

1. Getting ☕
2. Getting to “o”

HRL approaches find “locally” optimal solutions.

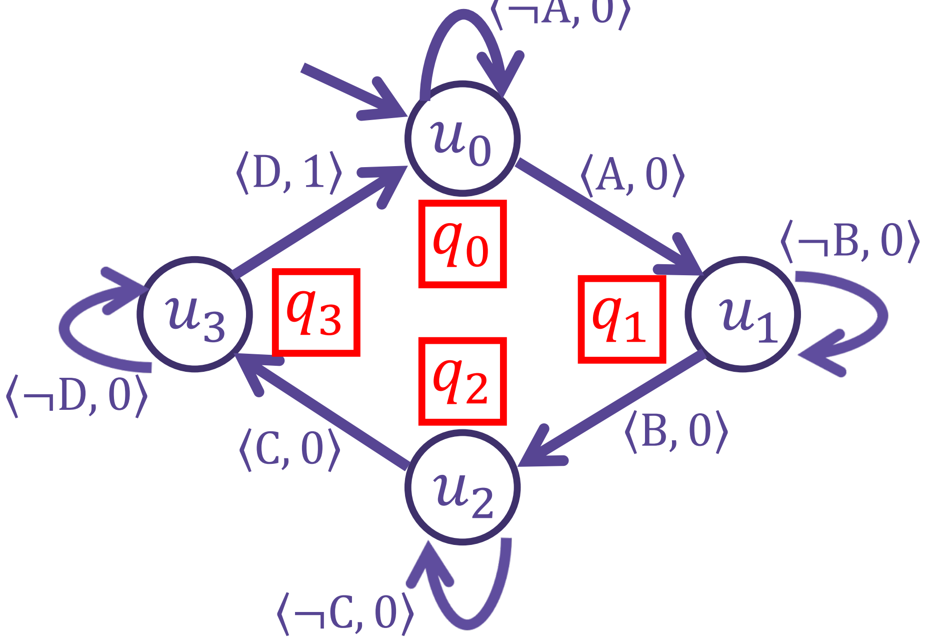
4. Q-Learning for Reward Machines (QRM)


$\langle \neg A, 0 \rangle$
u_0
$\langle D, 1 \rangle$
$\langle A, 0 \rangle$
$\langle \neg B, 0 \rangle$
u_3
u_1
$\langle \neg D, 0 \rangle$
$\langle B, 0 \rangle$
$\langle C, 0 \rangle$
u_2
$\langle \neg C, 0 \rangle$

4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

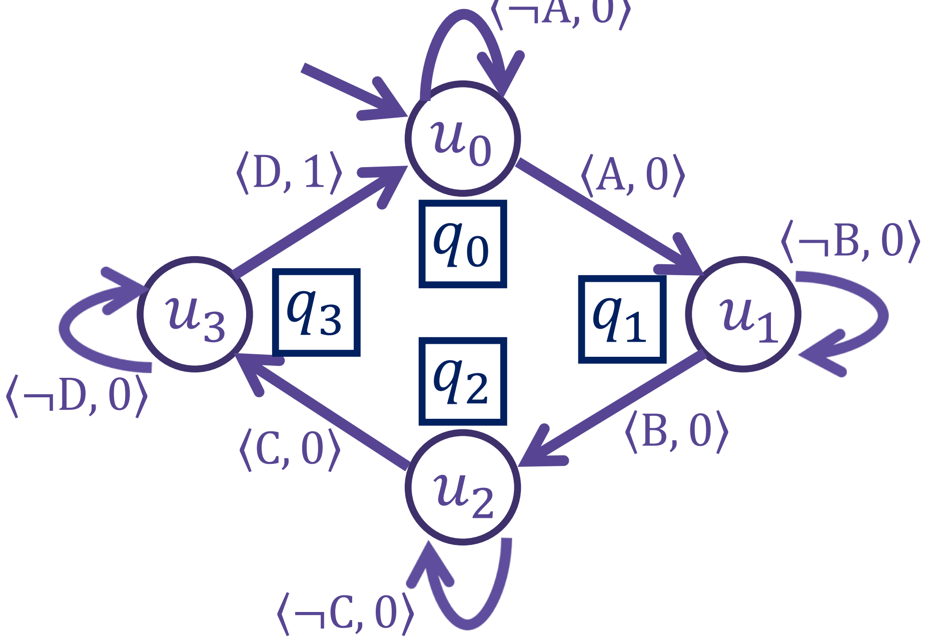
1. Learn one policy (q-value function) per state in the Reward Machine.



4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

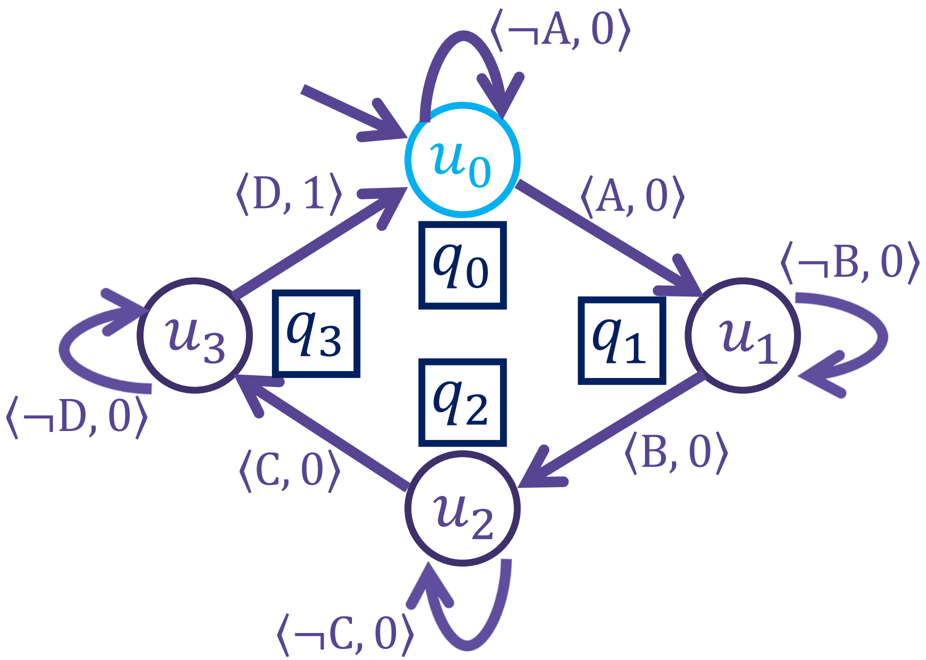
1. Learn one policy (q-value function) per state in the Reward Machine.
2. Select actions using the policy of the current RM state.



4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

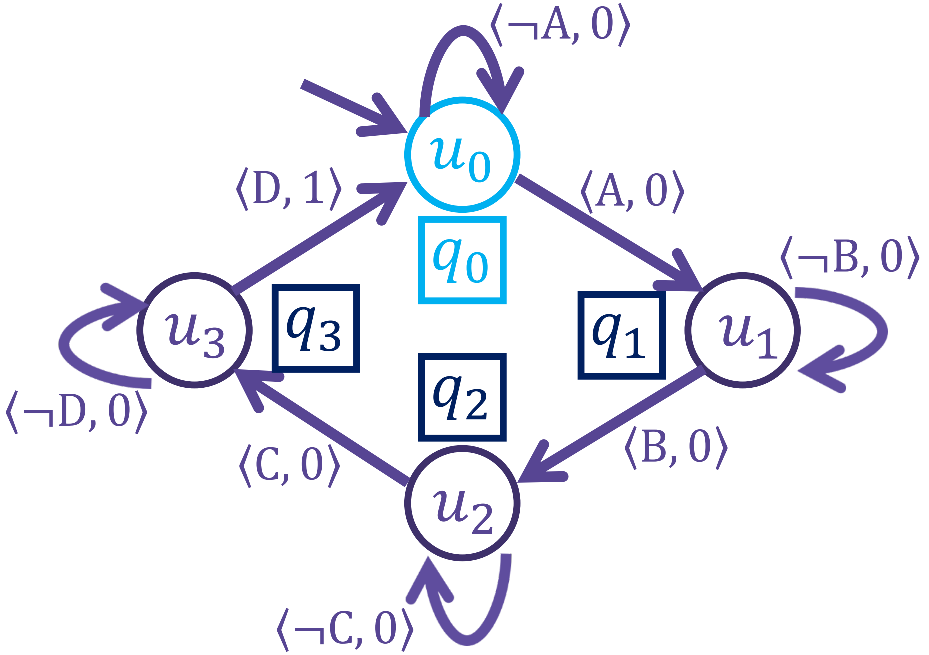
1. Learn one policy (q-value function) per state in the Reward Machine.
2. Select actions using the policy of the current RM state.



4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

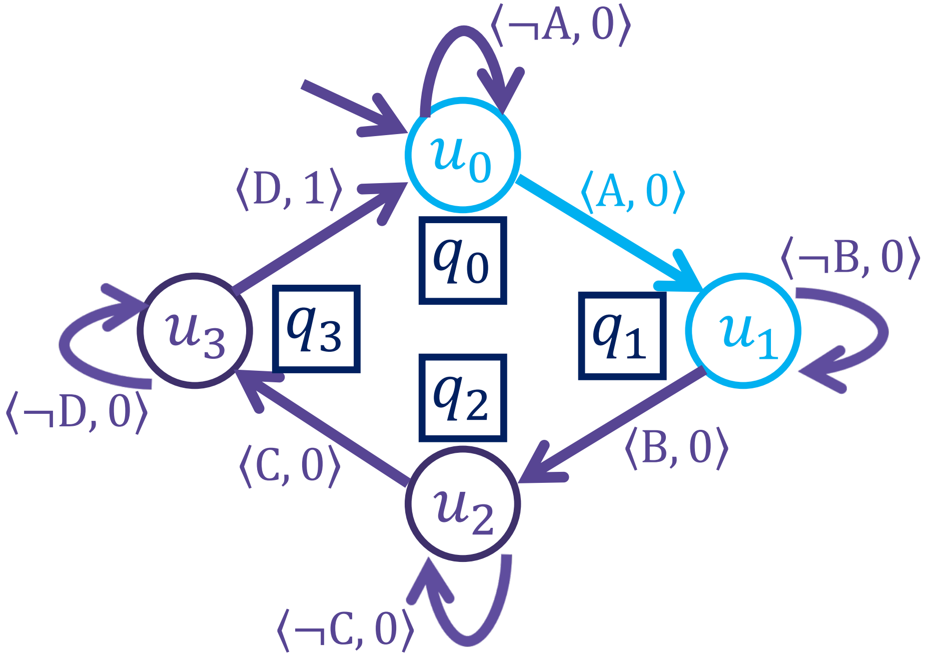
1. Learn one policy (q-value function) per state in the Reward Machine.
2. Select actions using the policy of the current RM state.



4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

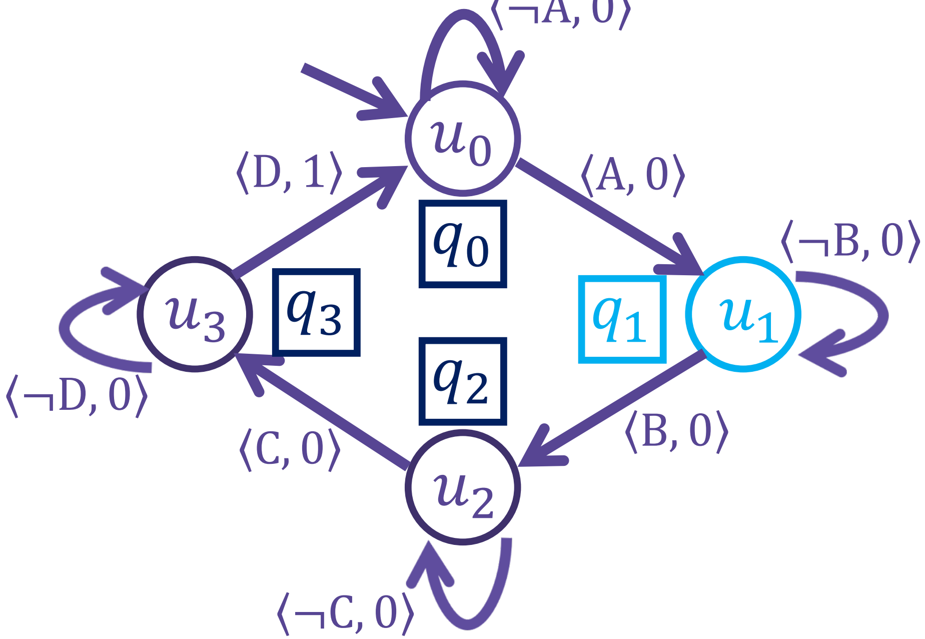
1. Learn one policy (q-value function) per state in the Reward Machine.
2. Select actions using the policy of the current RM state.



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.

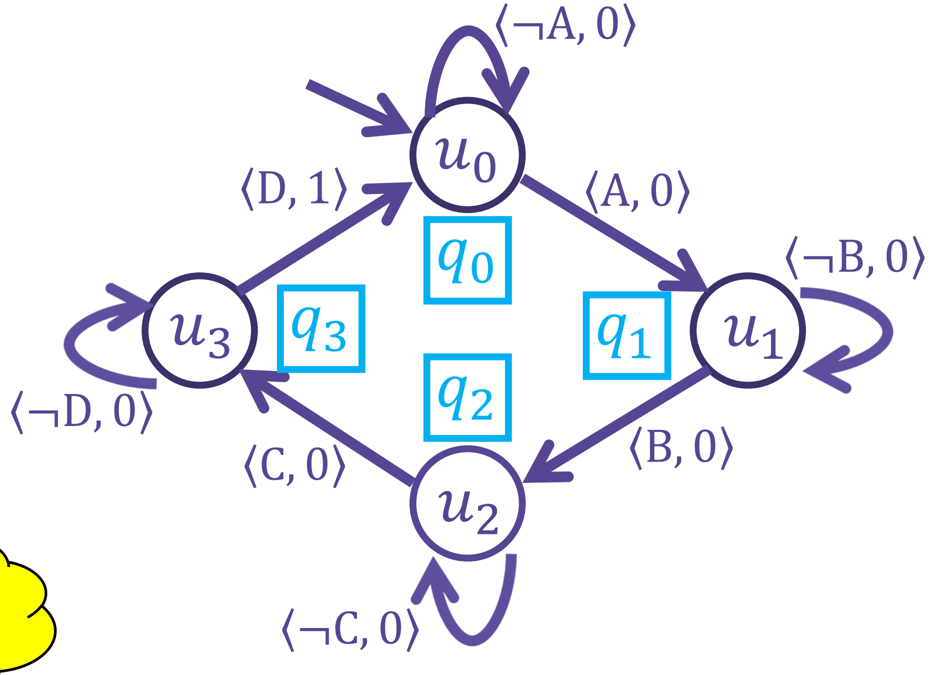


4. Q-Learning for Reward Machines (QRM)

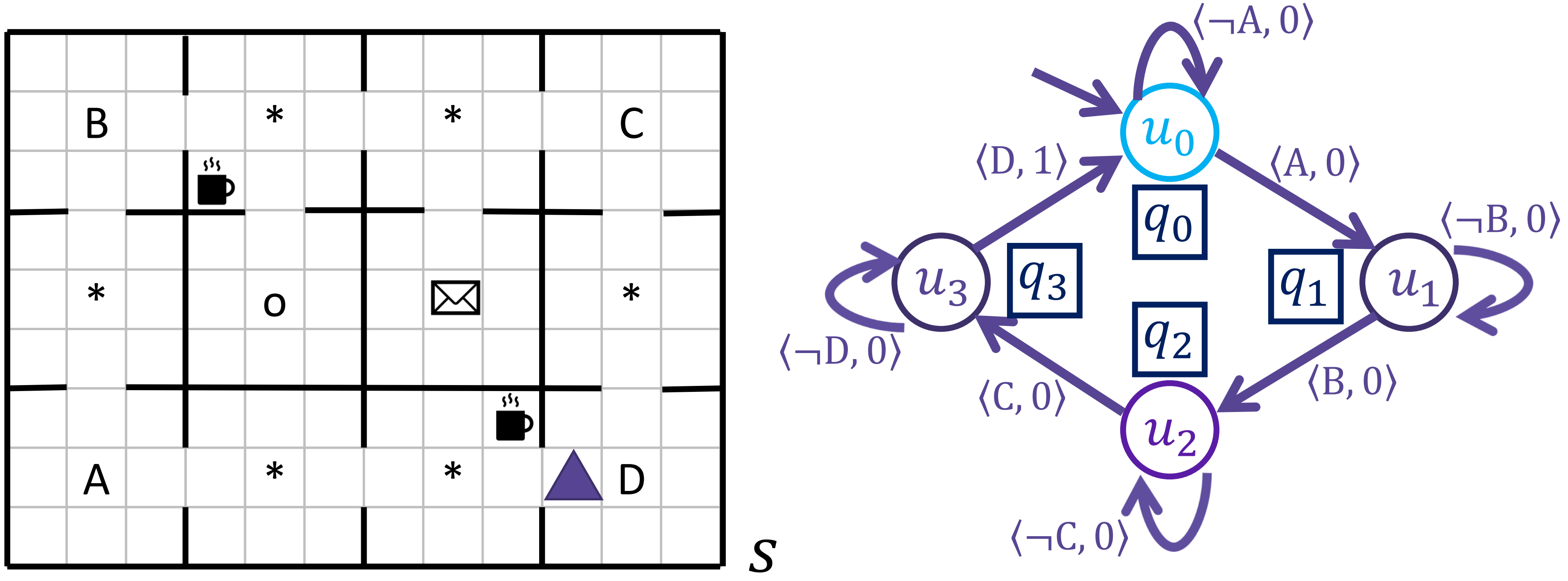
QRM (our approach)

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.

Remember this!

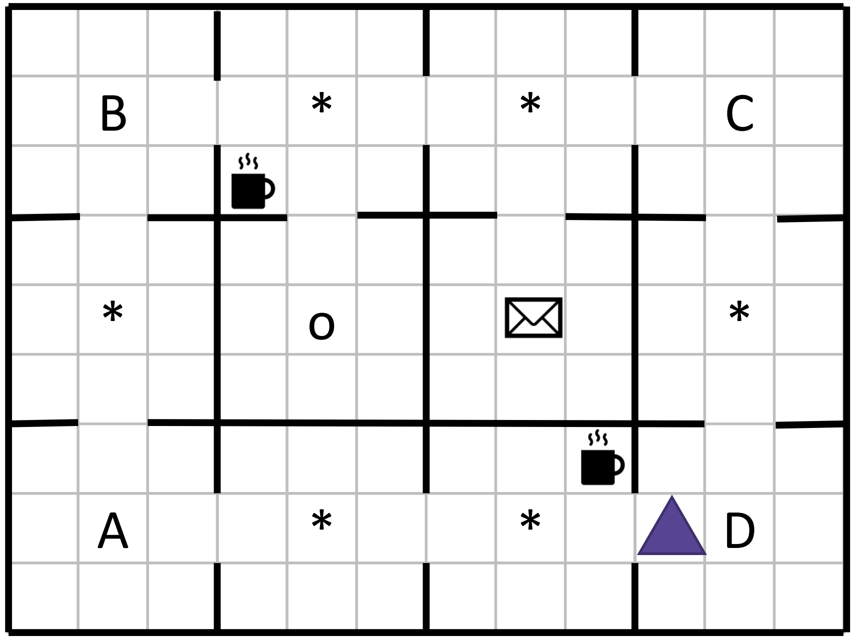


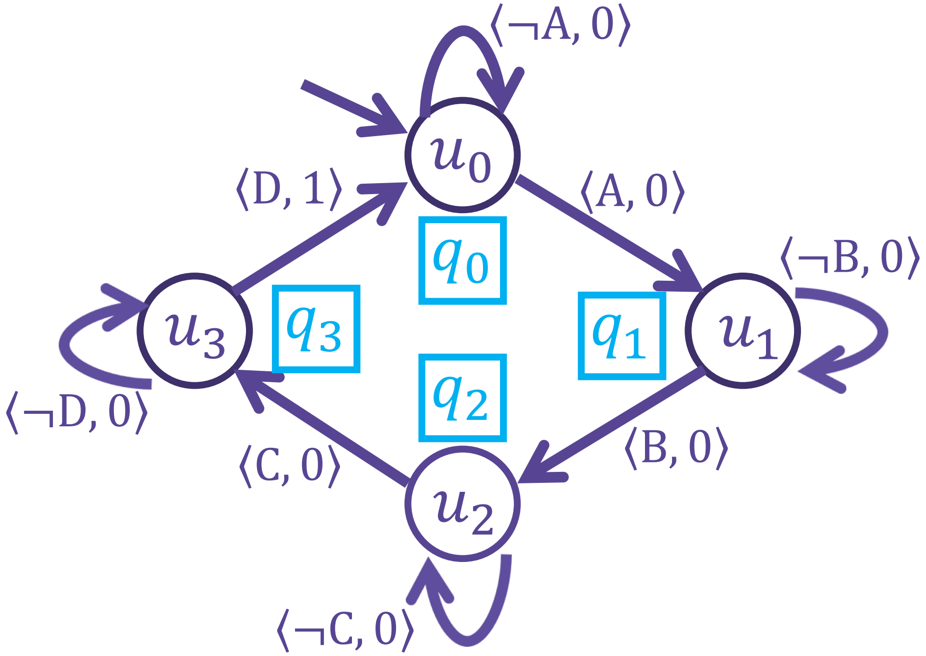
QRM In Action



Select an action according to the current RM state.

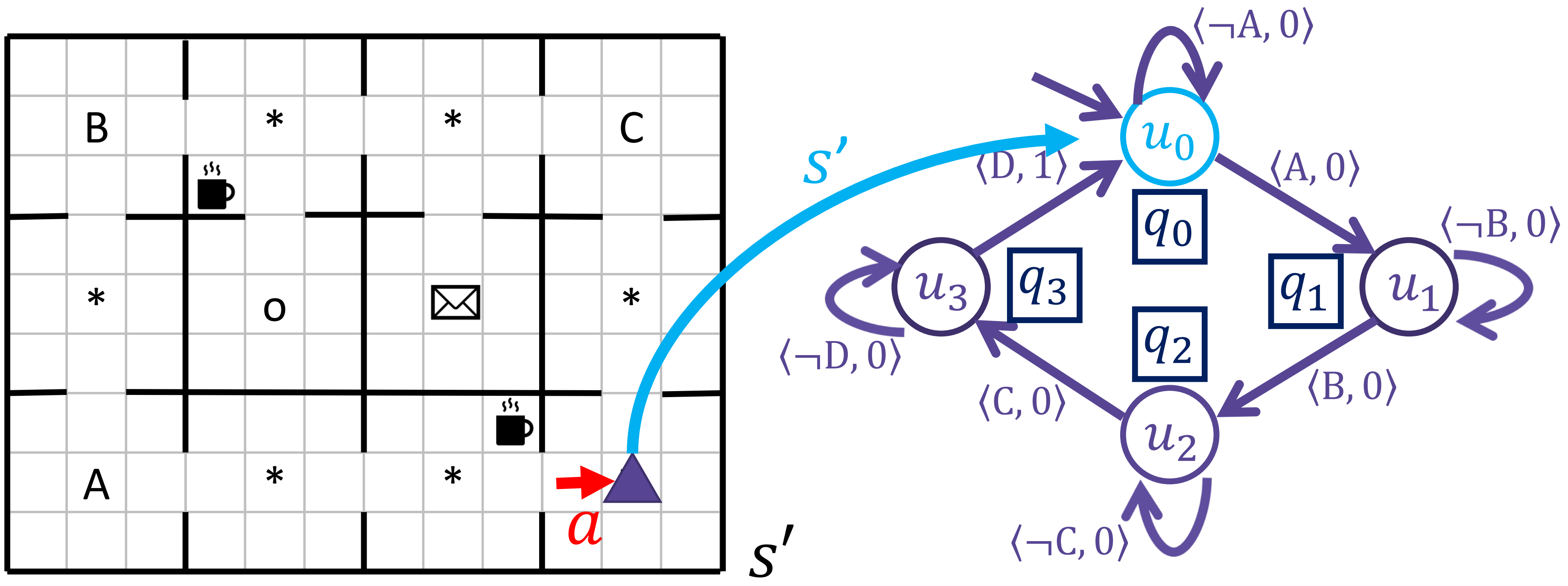
QRM In Action



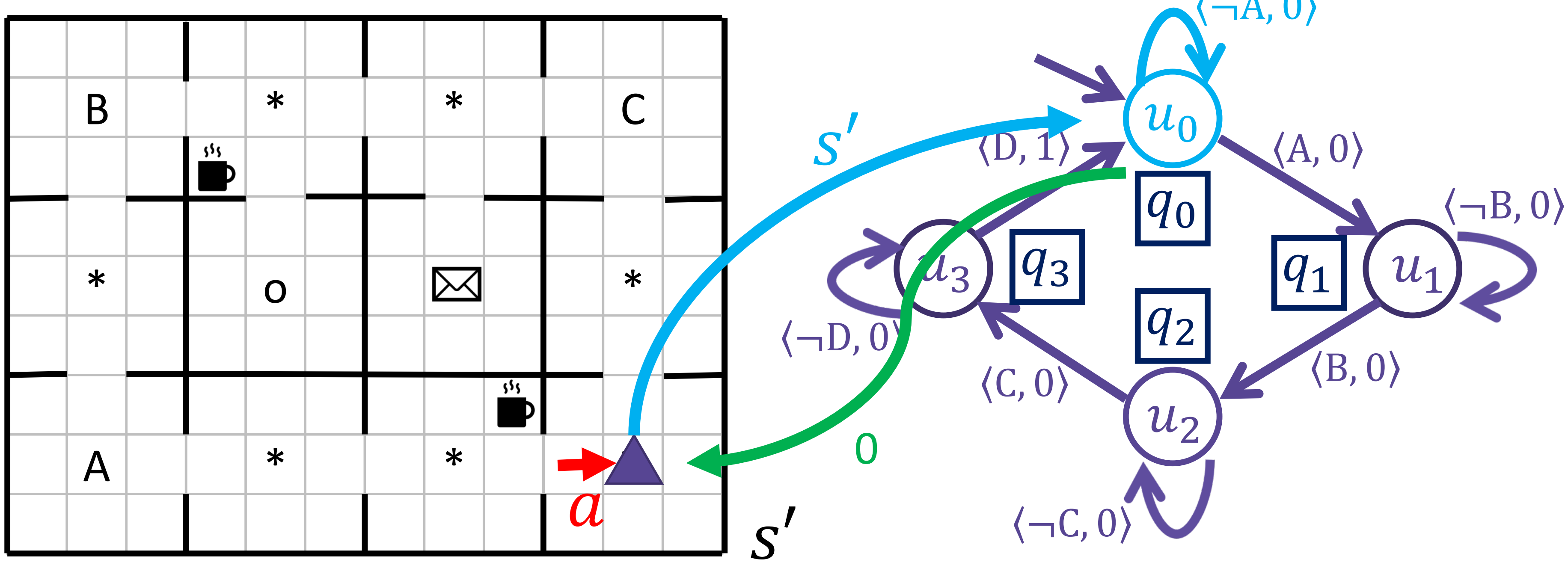


Update each q-value function as if RM were in corresponding state.

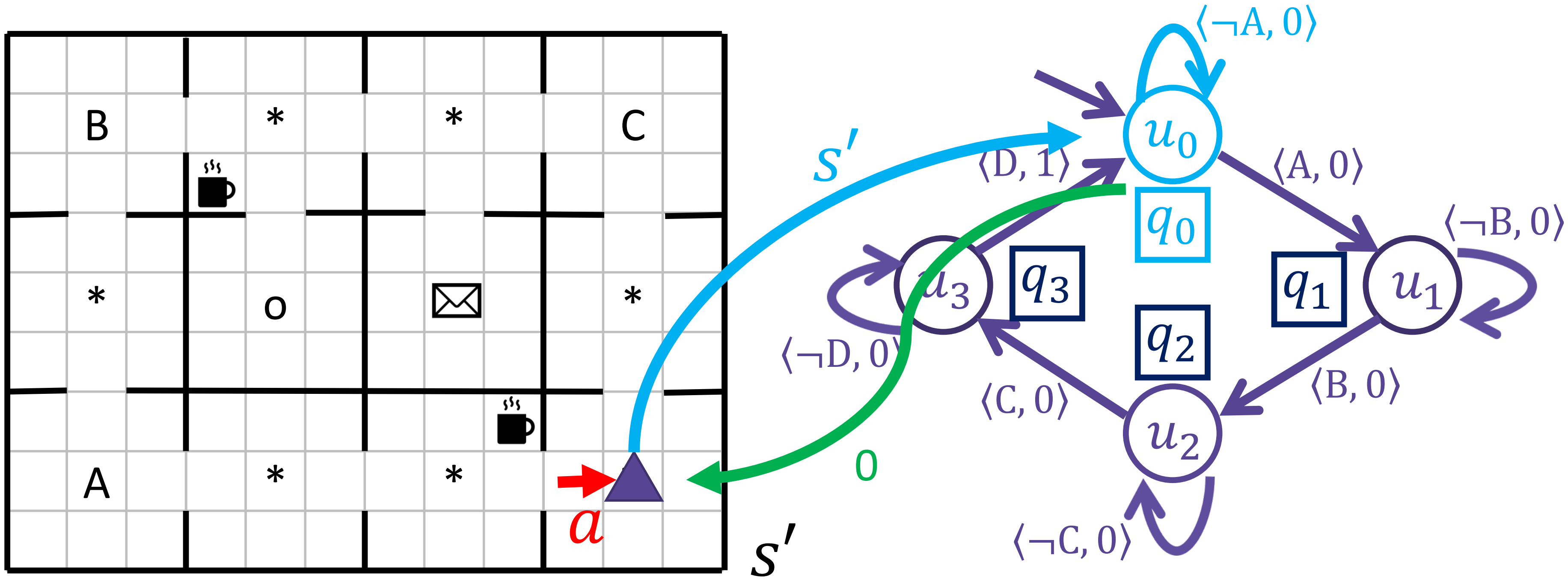
QRM In Action



QRM In Action

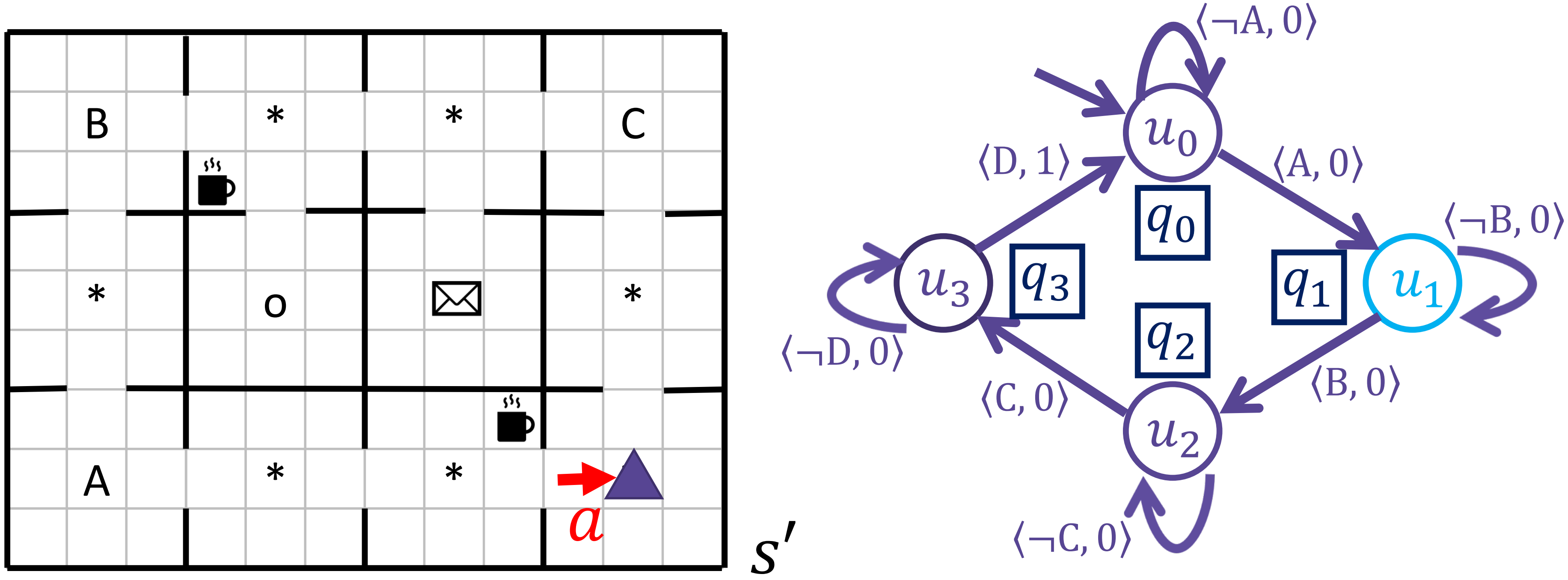


QRM In Action

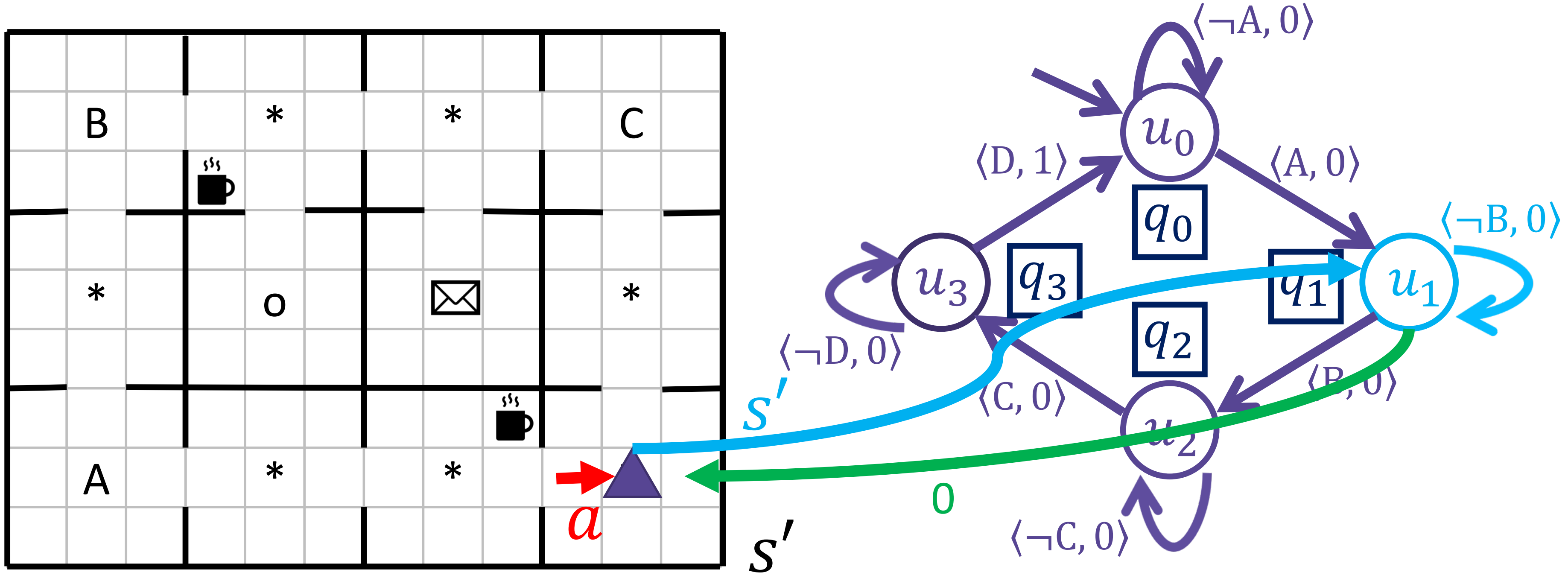


$$q_0(s, a) \leftarrow 0 + \gamma \cdot \max_{a'} q_0(s', a')$$

QRM In Action

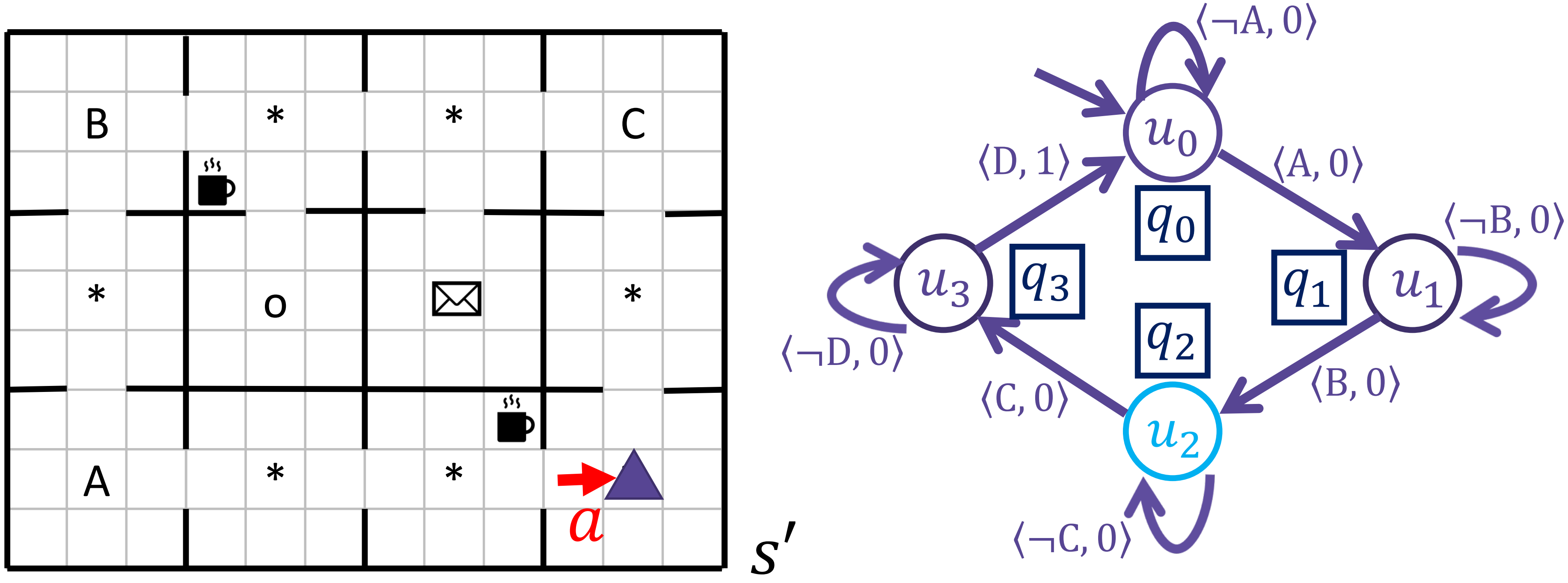


QRM In Action

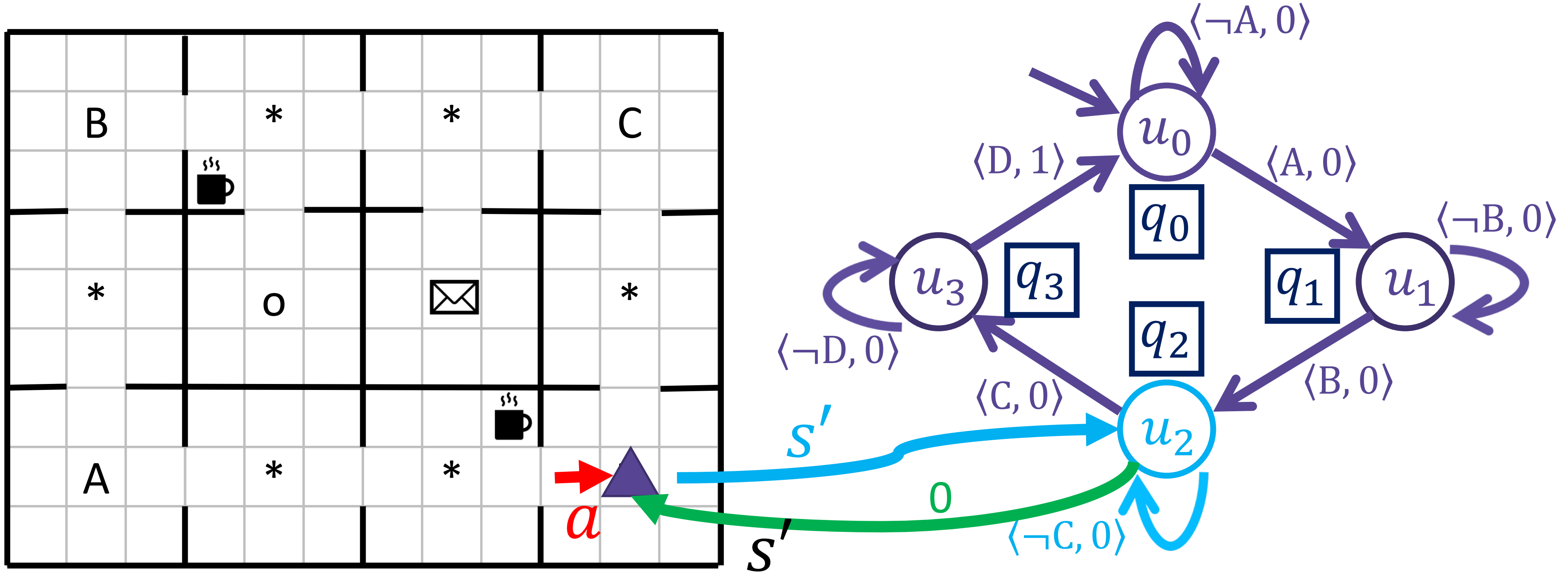


$$q_1(s, a) \leftarrow 0 + \gamma \cdot \max_{a'} q_1(s', a')$$

QRM In Action

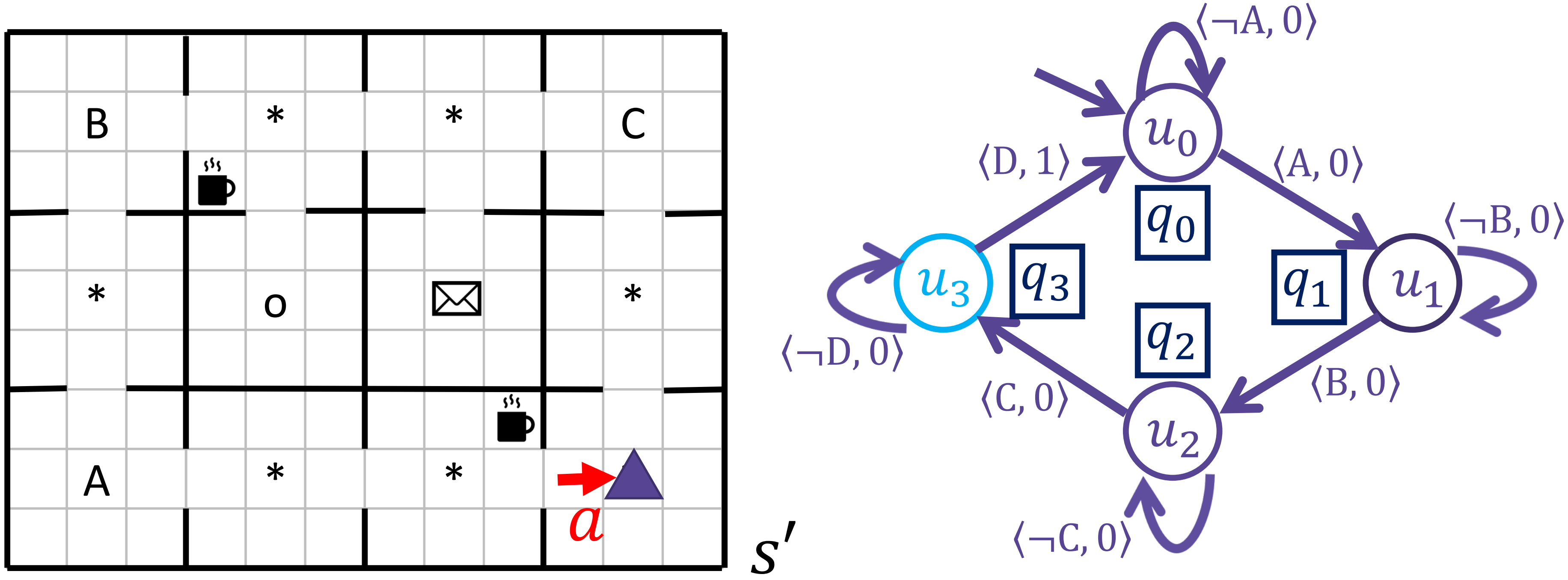


QRM In Action

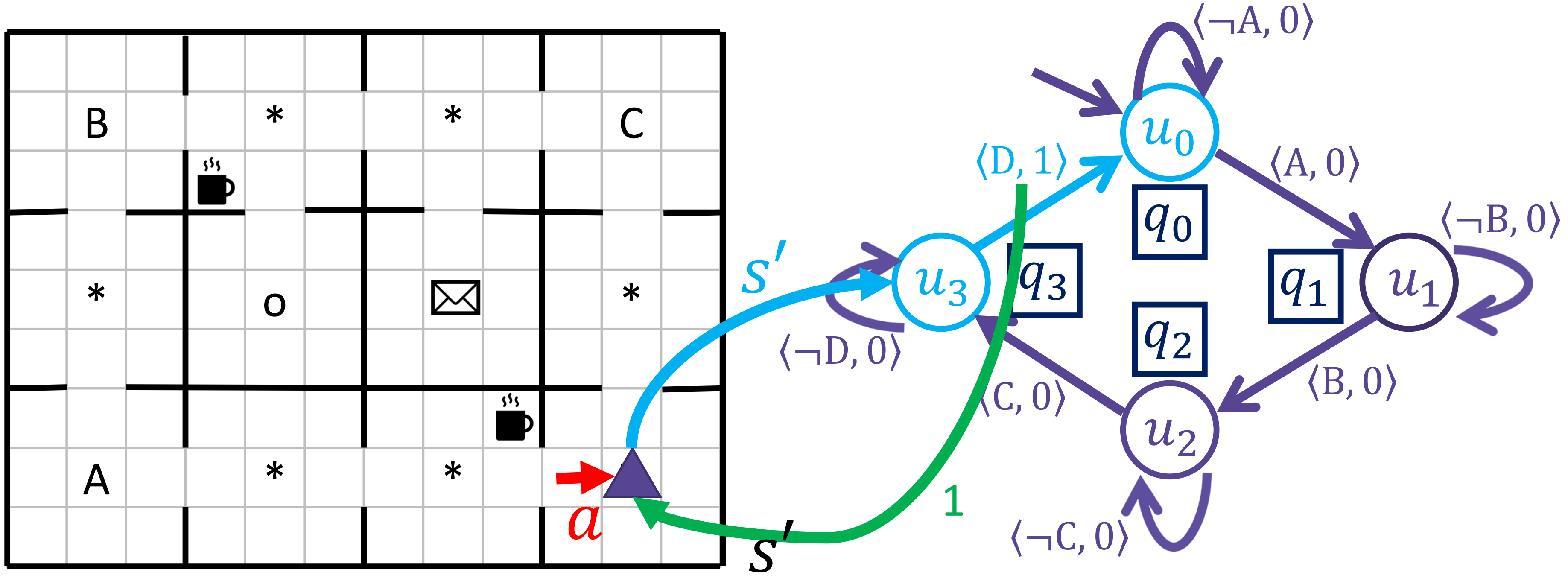


$$q_2(s, a) \leftarrow 0 + \gamma \cdot \max_{a'} q_2(s', a')$$

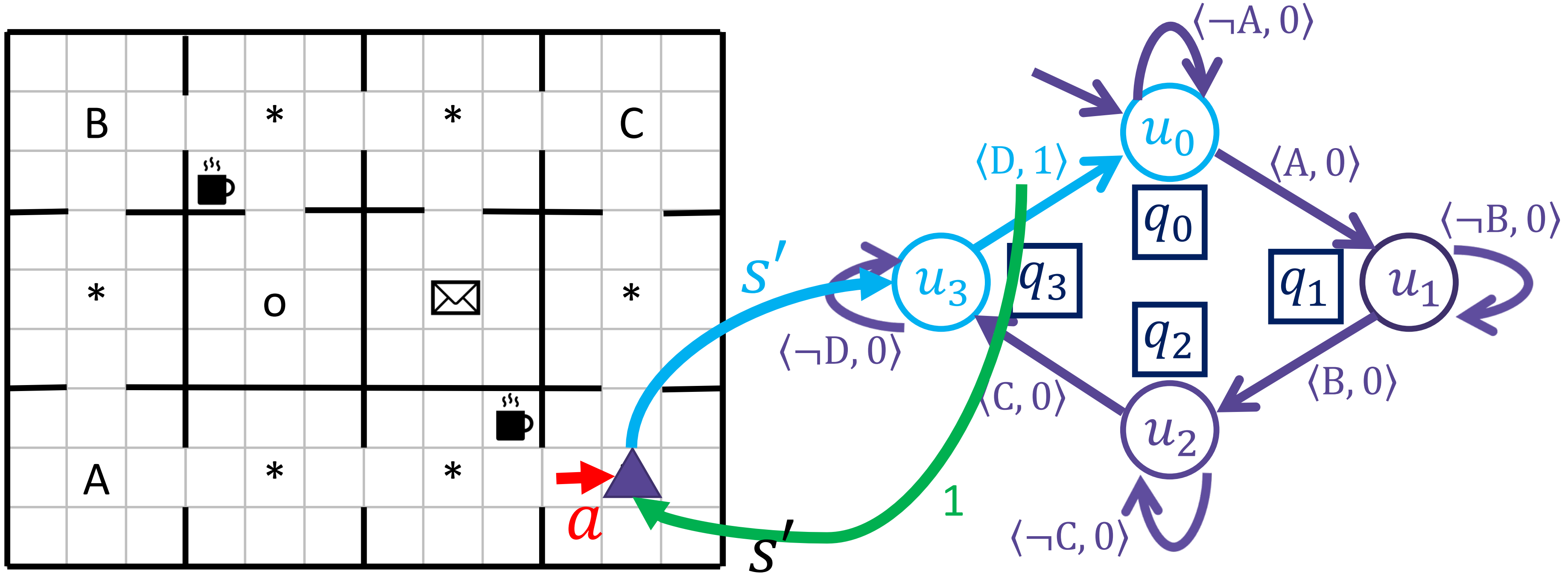
QRM In Action



QRM In Action

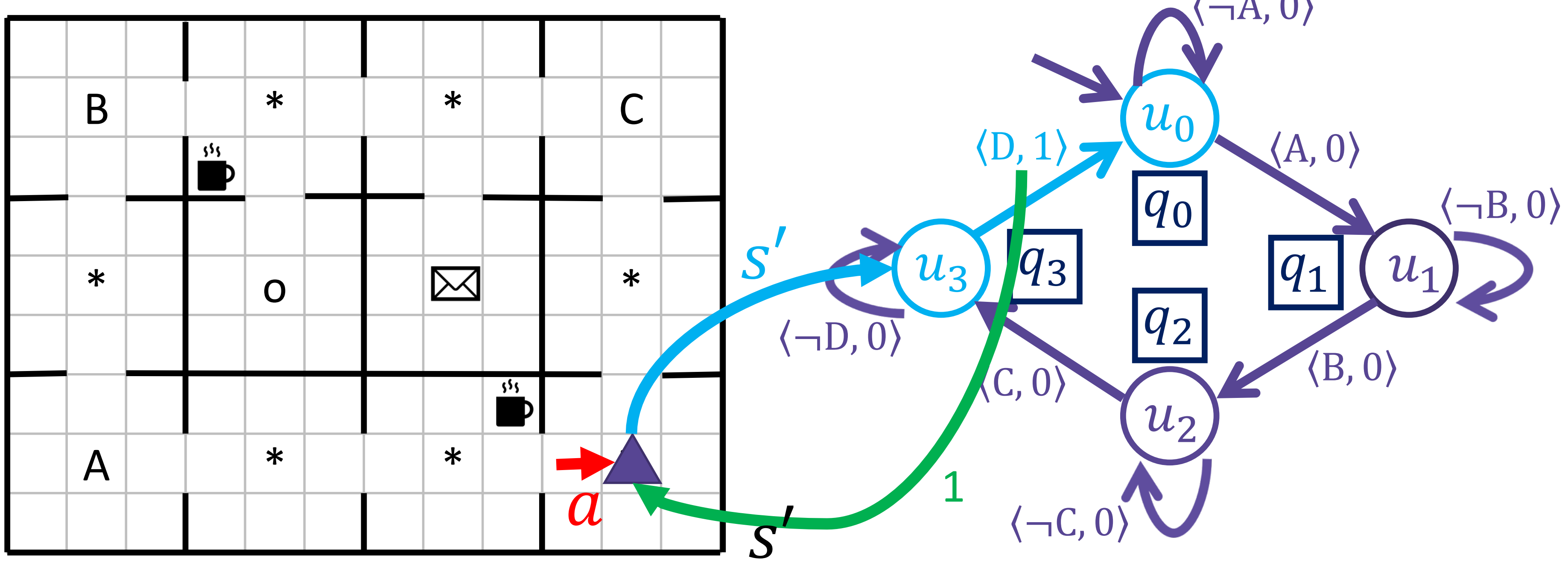


QRM In Action



$$q_3(s, a) \leftarrow 1 + \gamma \cdot \max_{a'} q_0(s', a')$$

QRM In Action



$$q_3(s, a) \leftarrow 1 + \gamma \cdot \max_{a'} q_0(s', a')$$

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)

Reward Shaping Intuition: Some reward functions are easier to learn policies for than others, even if those functions that have the same optimal policy.

Given any MDP and **potential function** $\Phi : S \rightarrow \mathbb{R}$, changing the reward function of the MDP to:

$$r'(s,a,s') = r(s,a,s') + \gamma\Phi(s') - \Phi(s)$$

will not change the set of optimal policies.

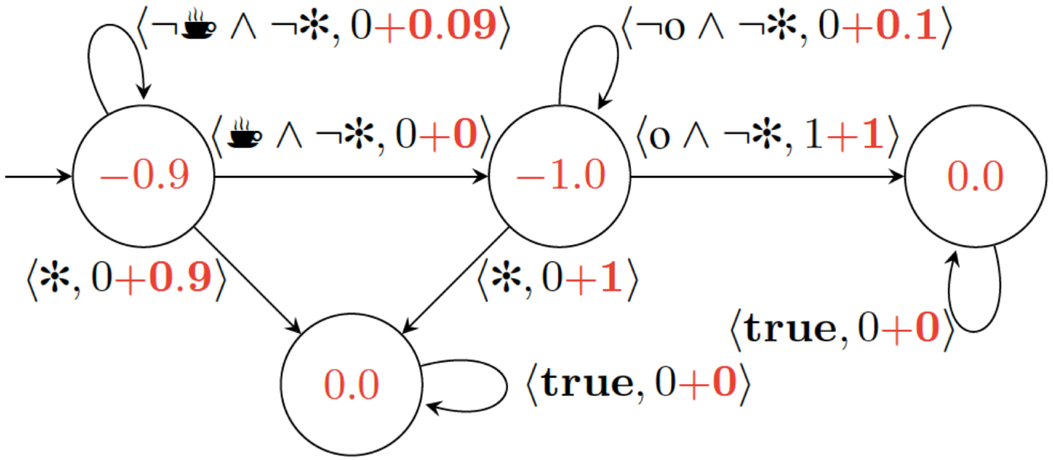
Thus, if we find a function that also allows us to learn optimal policies more quickly, we are guaranteed that the found policies are still optimal with respect to the original reward function.

[Ng, Harada, Russell, 1999]

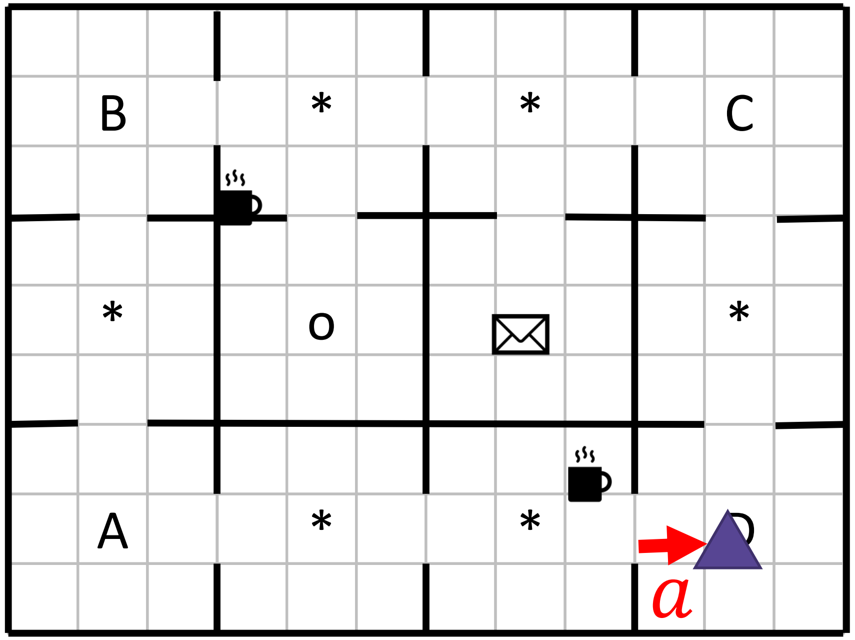
5. QRM + Reward Shaping (QRM + RS)

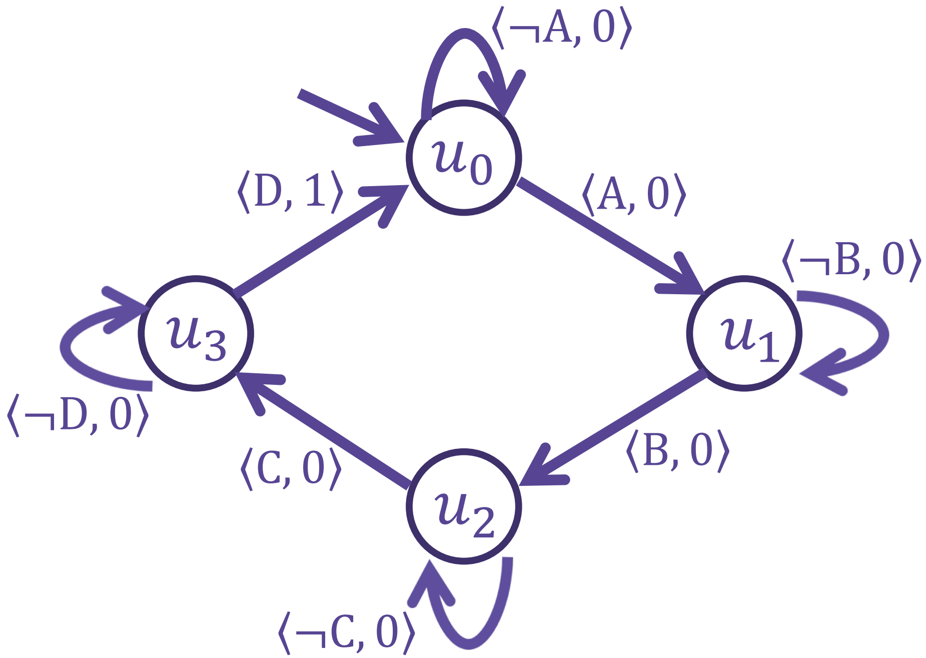
QRM + RS (our approach)

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 converges to the optimal policy in the limit, as does QRM + RS.

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)

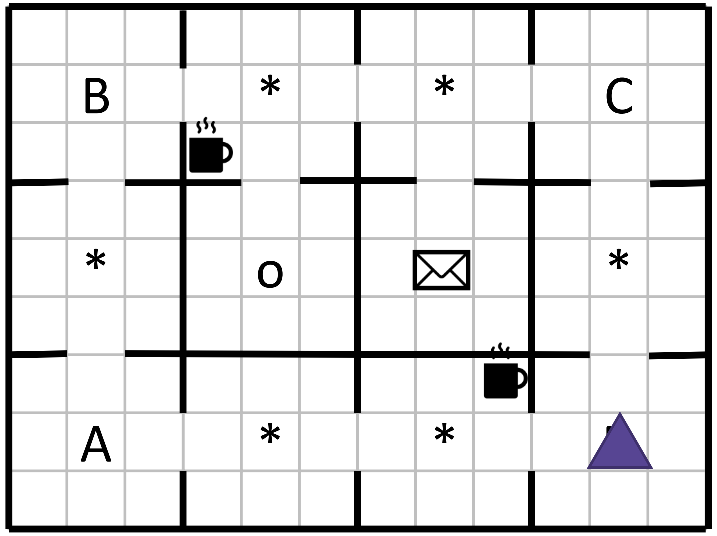
Test in Discrete Domains

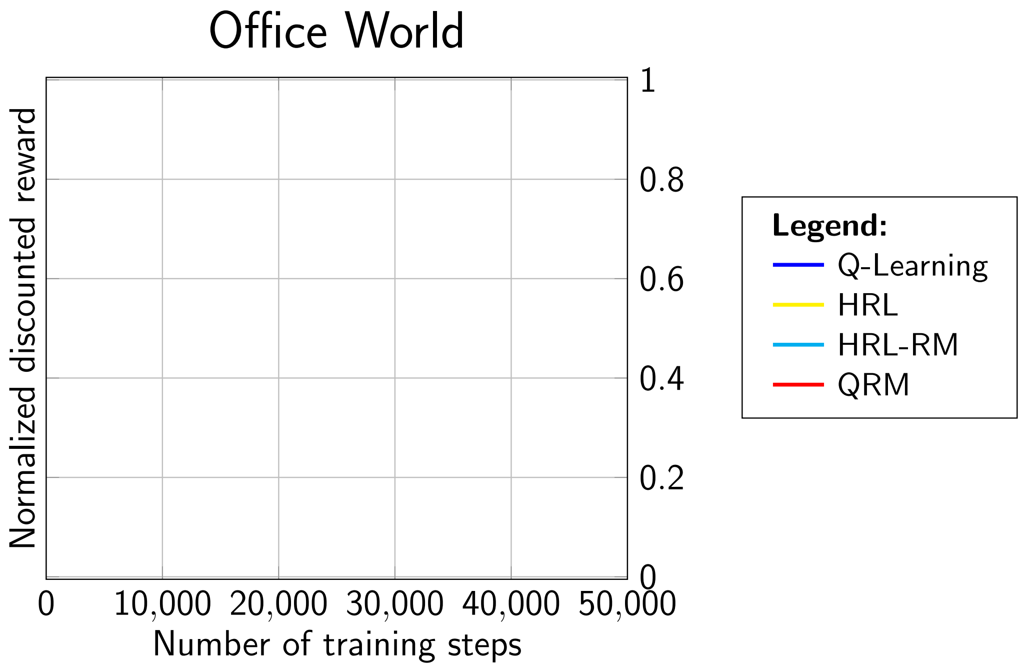
Tested all five approaches

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	✓	✓

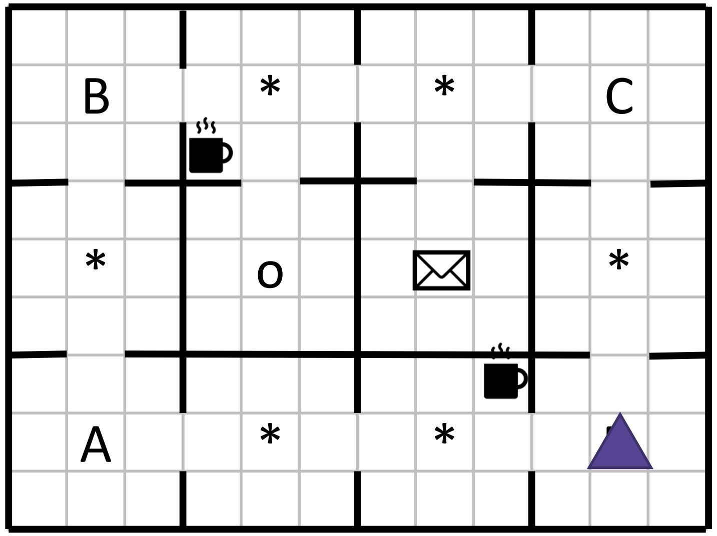
Office World Experiments

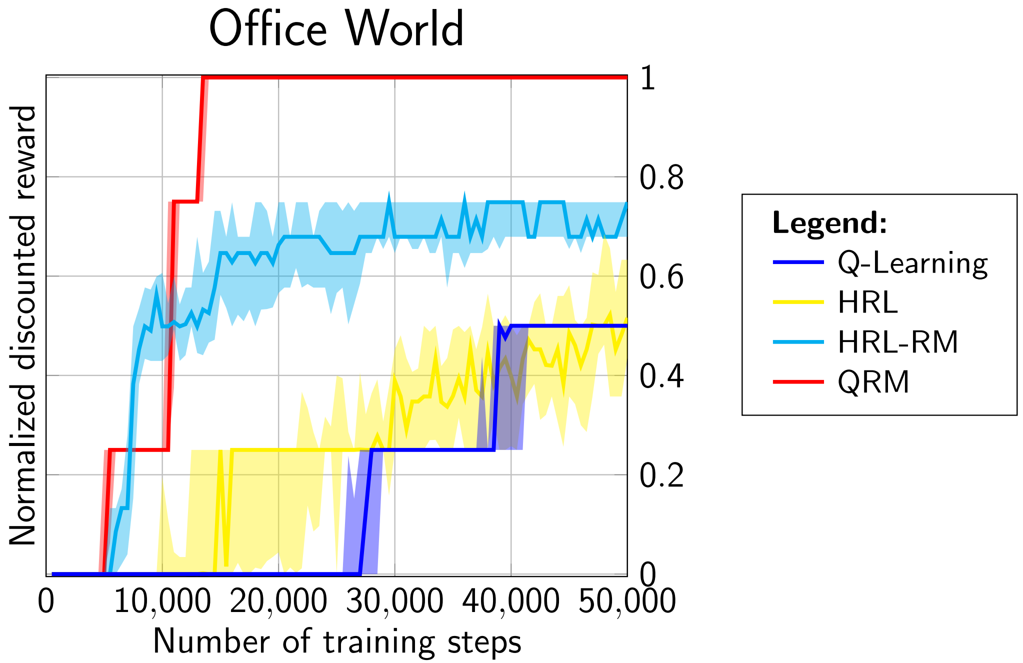




4 tasks, 30 independent trials per task

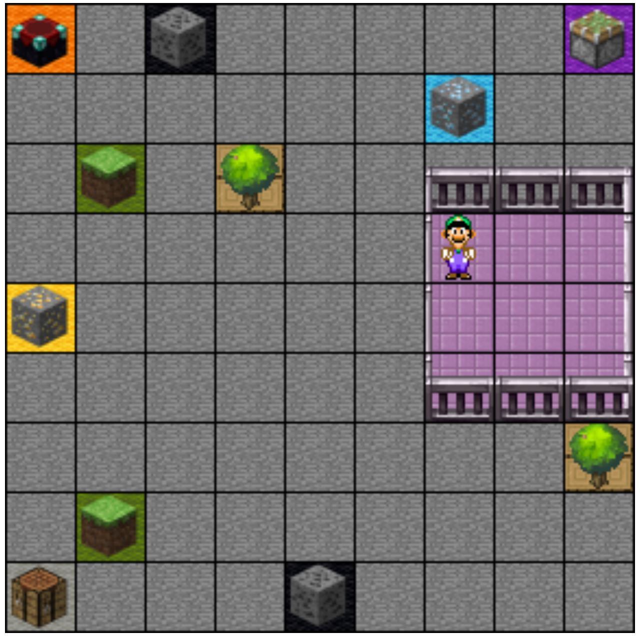
Office World Experiments

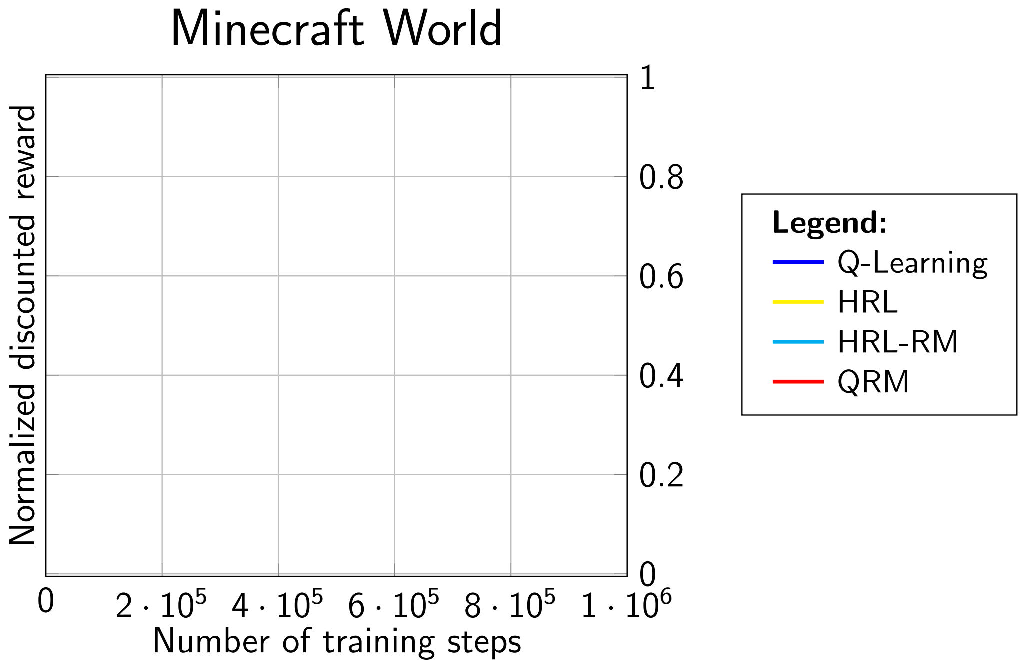




4 tasks, 30 independent trials per task

Minecraft World Experiments

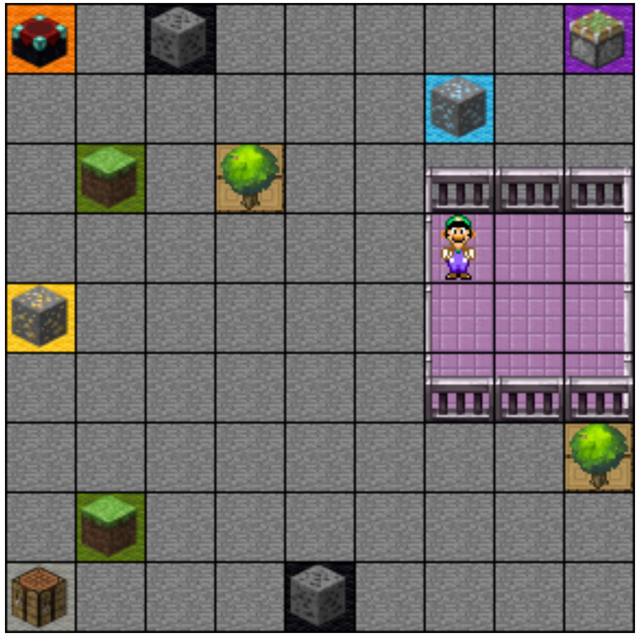


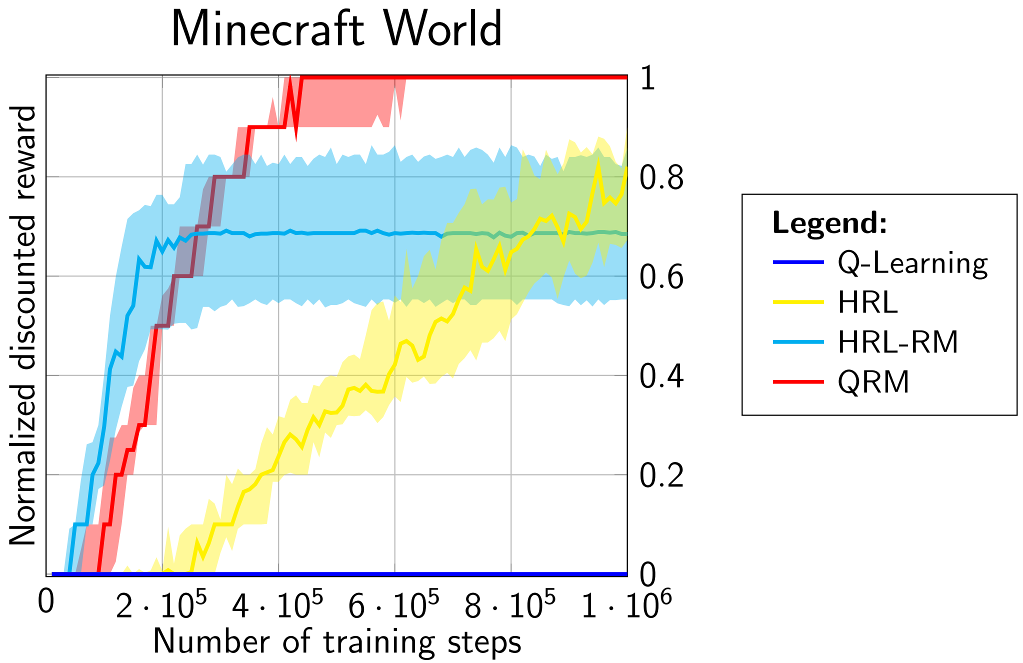


10 tasks over 10 random maps, 3 independent trials per combination

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)

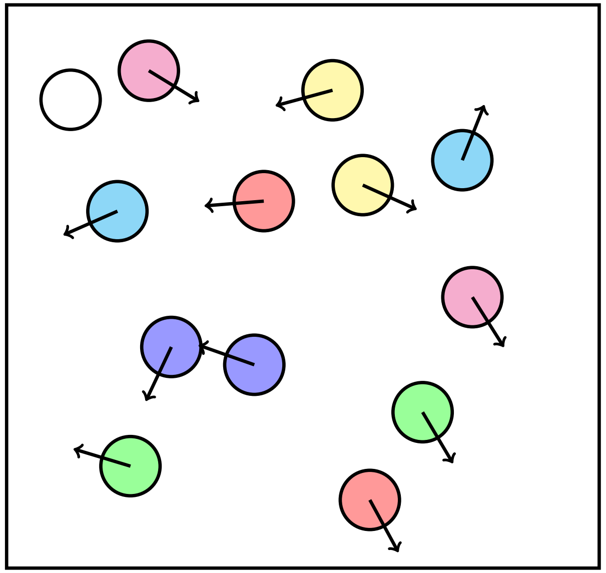
Function Approximation with QRM

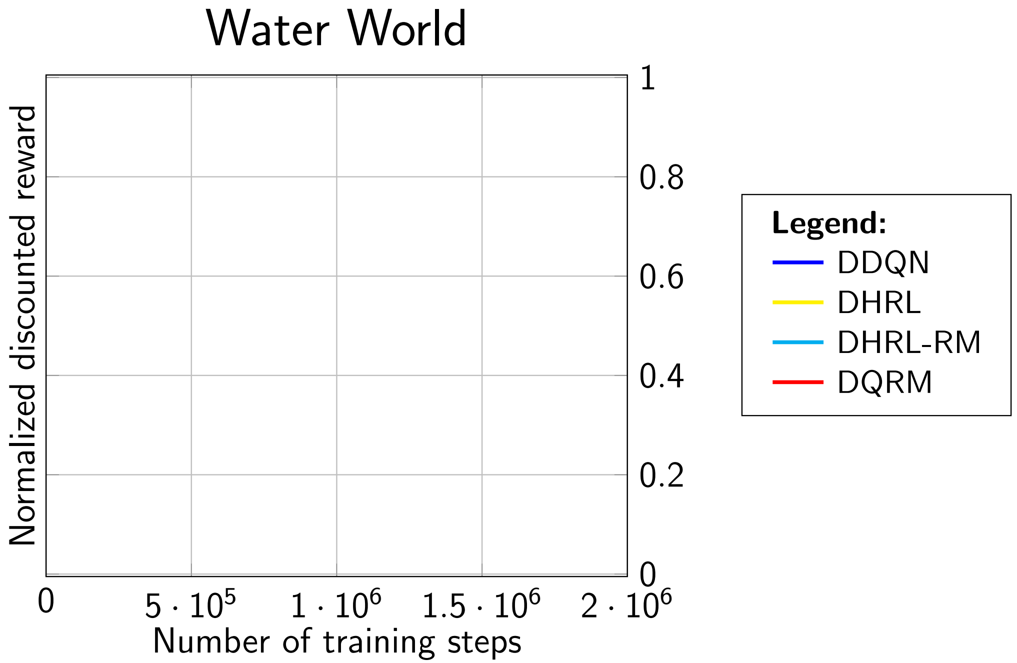
From tabular QRM to Deep QRM

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

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

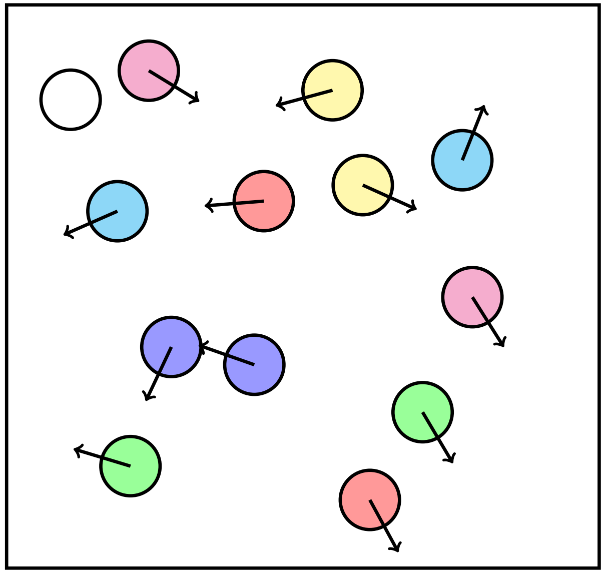
Water World Experiments

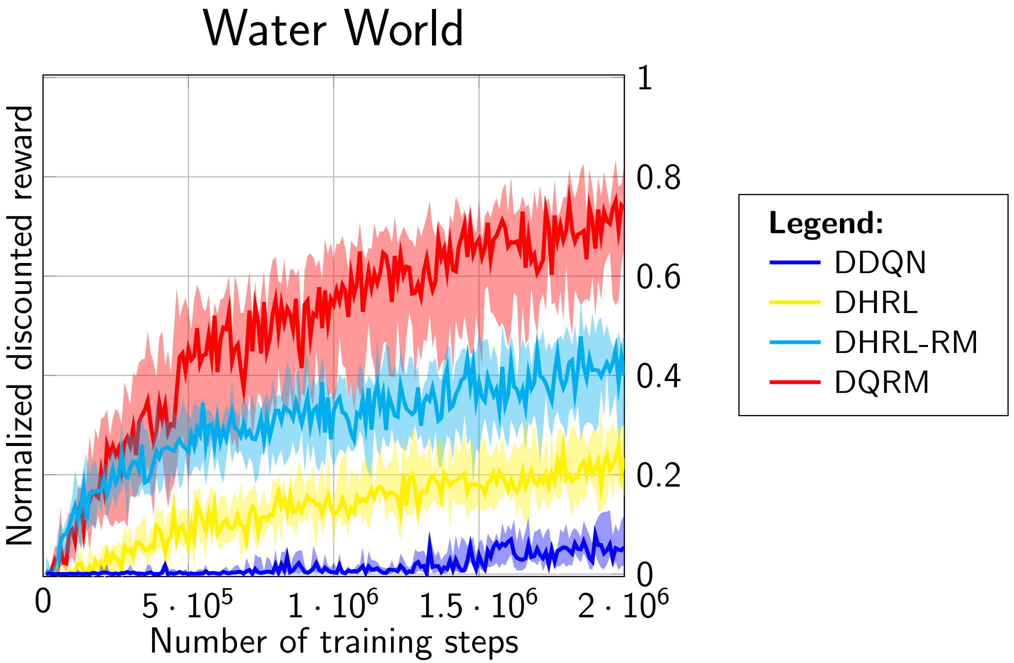




10 tasks over 10 random maps, 3 independent trials per combination

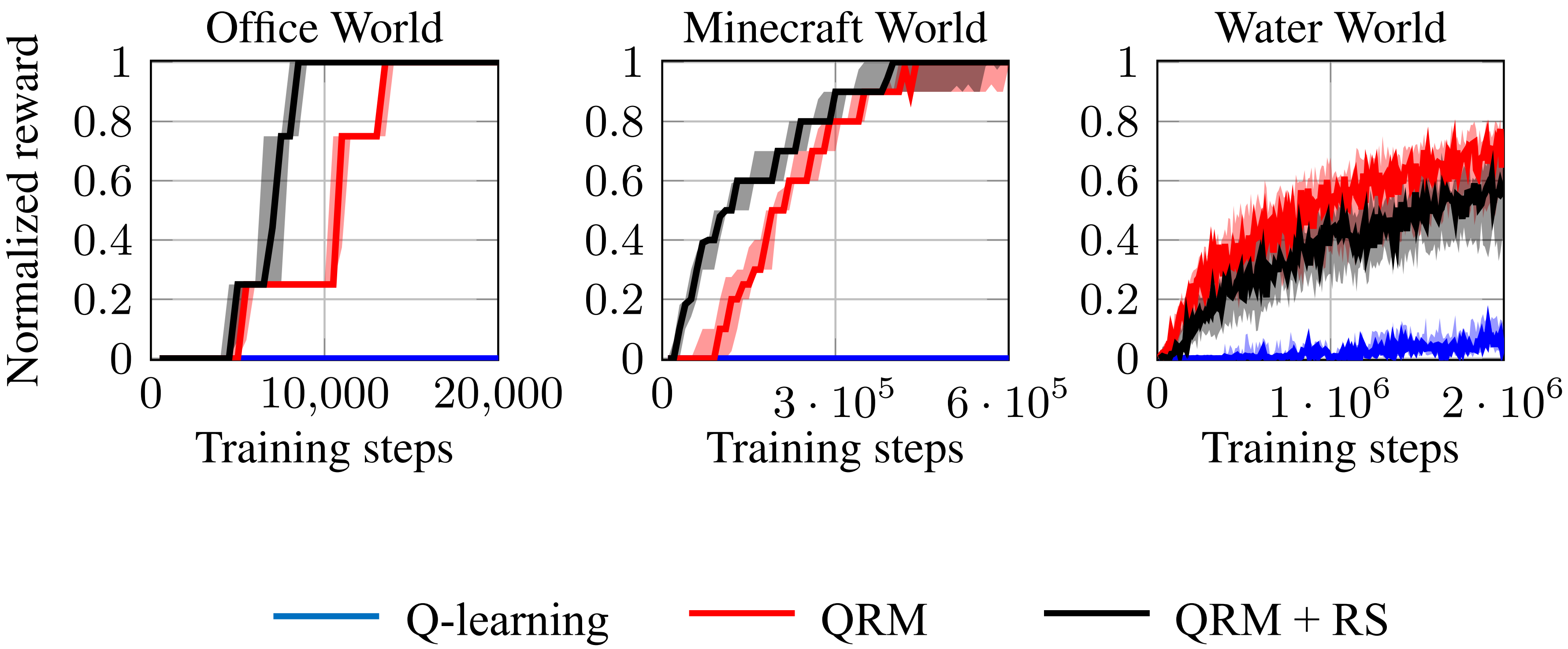
Water World Experiments





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

QRM + Reward Shaping (QRM + RS)



Discount factor γ of 0.9 and exploration constant ϵ of 0.1

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?

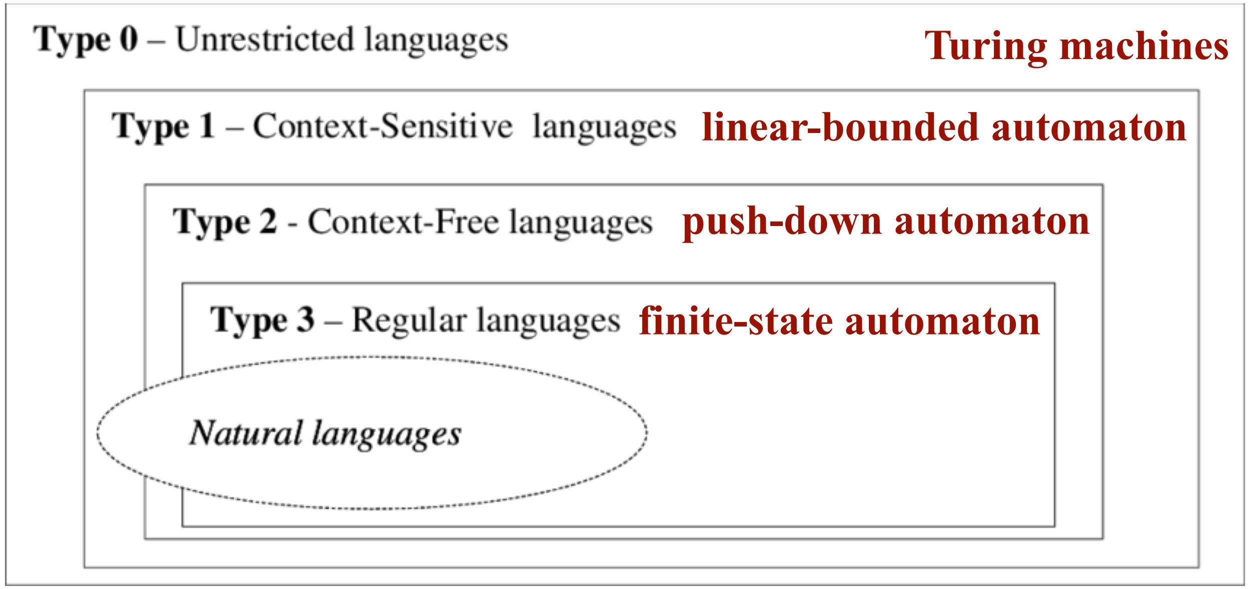
1. Specify RM
   - Directly
   - Via automatic translation from specifications in various languages
2. Generate RM from high-level goal specifications
3. Learn RM

1. **Reward Specification:** one size does *not* fit all

Do not need to specify Reward Machines directly.

Reward Machines are a form of Mealy Machine.

Specify reward-worthy behavior in **any formal language that is translatable to finite-state automata**.



Type 0 – Unrestricted languages **Turing machines**

Type 1 – Context-Sensitive languages **linear-bounded automaton**

Type 2 - Context-Free languages **push-down automaton**

Type 3 – Regular languages **finite-state automaton**

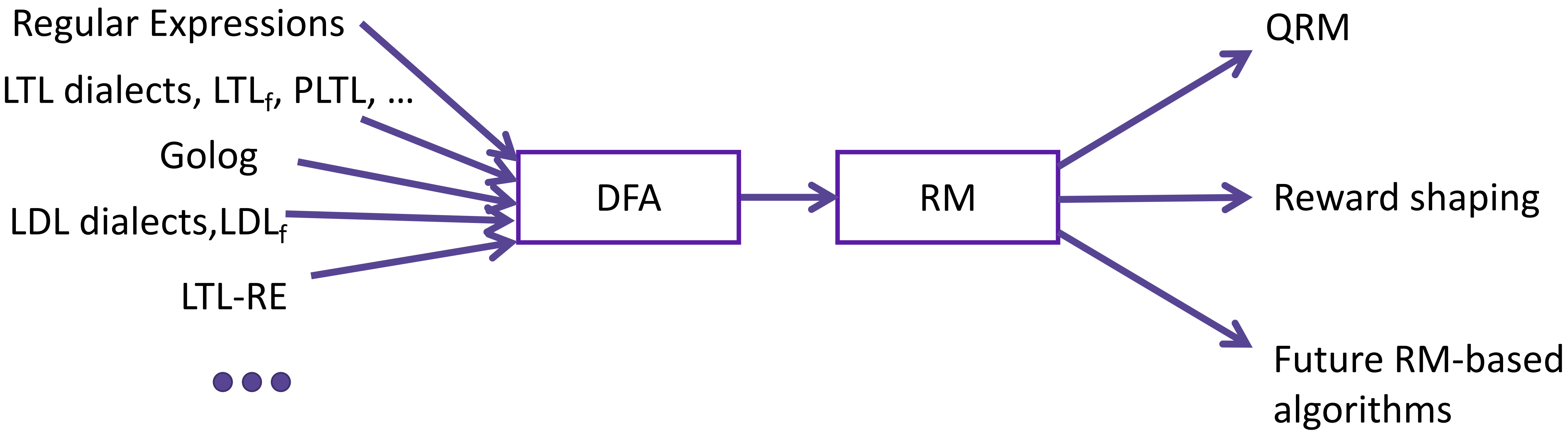
Natural languages

The Chomsky Hierarchy

Noam Chomsky

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**.

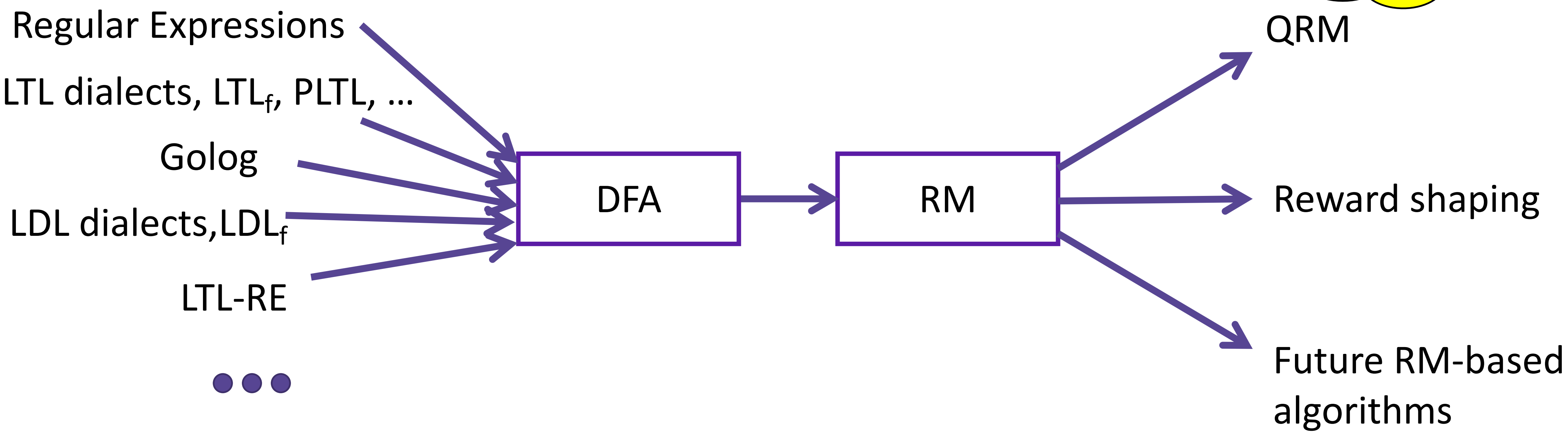


[Camacho, Toro Icarte, Klassen, Valenzano, M., IJCAI19]

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**.

Remember this!



Regular Expressions

LTL dialects, LTL_f, PLTL, …

Golog

LDL dialects,LDL_f

LTL-RE

●●●

DFA

RM

QRM

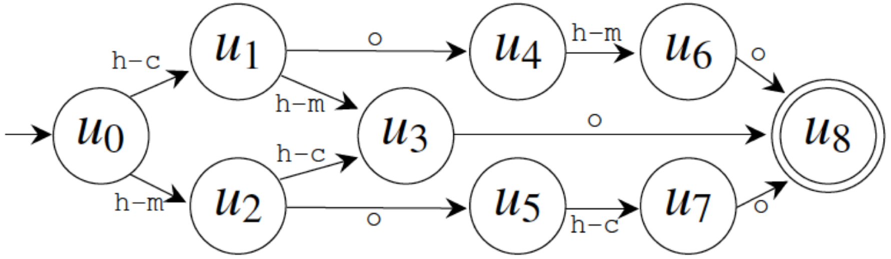
Reward shaping

Future RM-based algorithms

[Camacho, Toro Icarte, Klassen, Valenzano, M., IJCAI19]

2. Generate RM using a Symbolic Planner

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

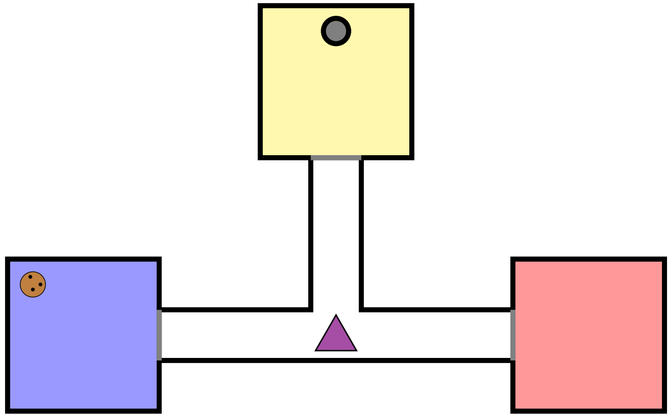


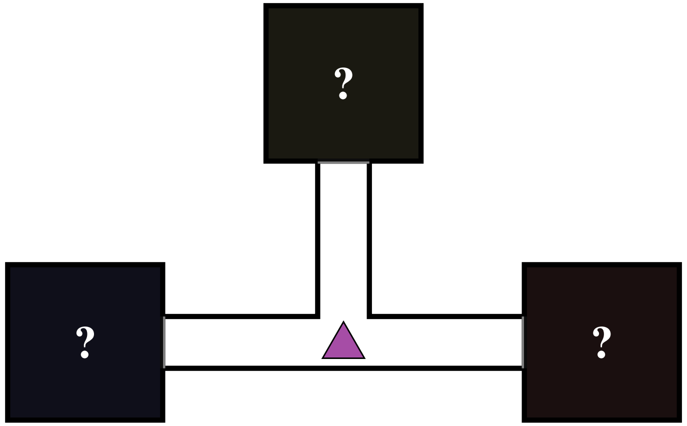
u_0: ∅
u_1: {get-coffee}
u_2: {get-mail}
u_3: {get-coffee, get-mail}
u_4: {get-coffee, deliver-coffee}

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]

3. Learn RMs for Partially-Observable RL





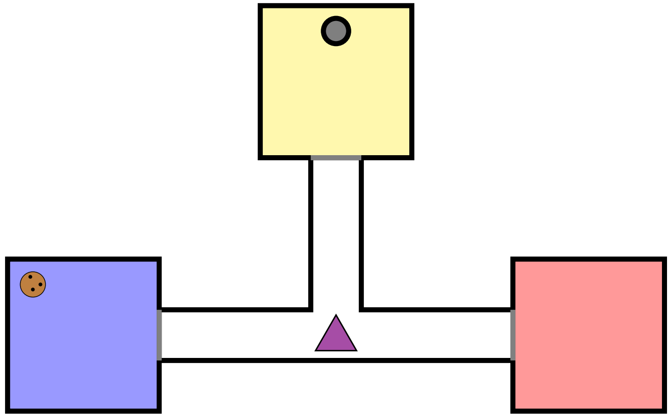
Problem: Find a policy that maximizes the external reward given by a partially observable environment

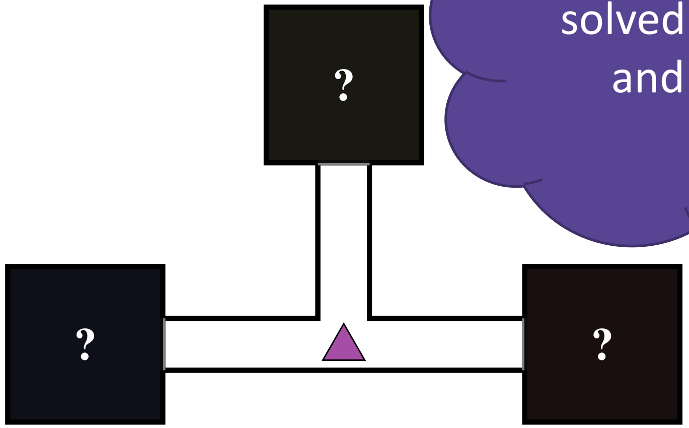
Assumptions: Agent has a set of high-level binary classifiers/event detectors (e.g., button-pushed, cookies, etc.)

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

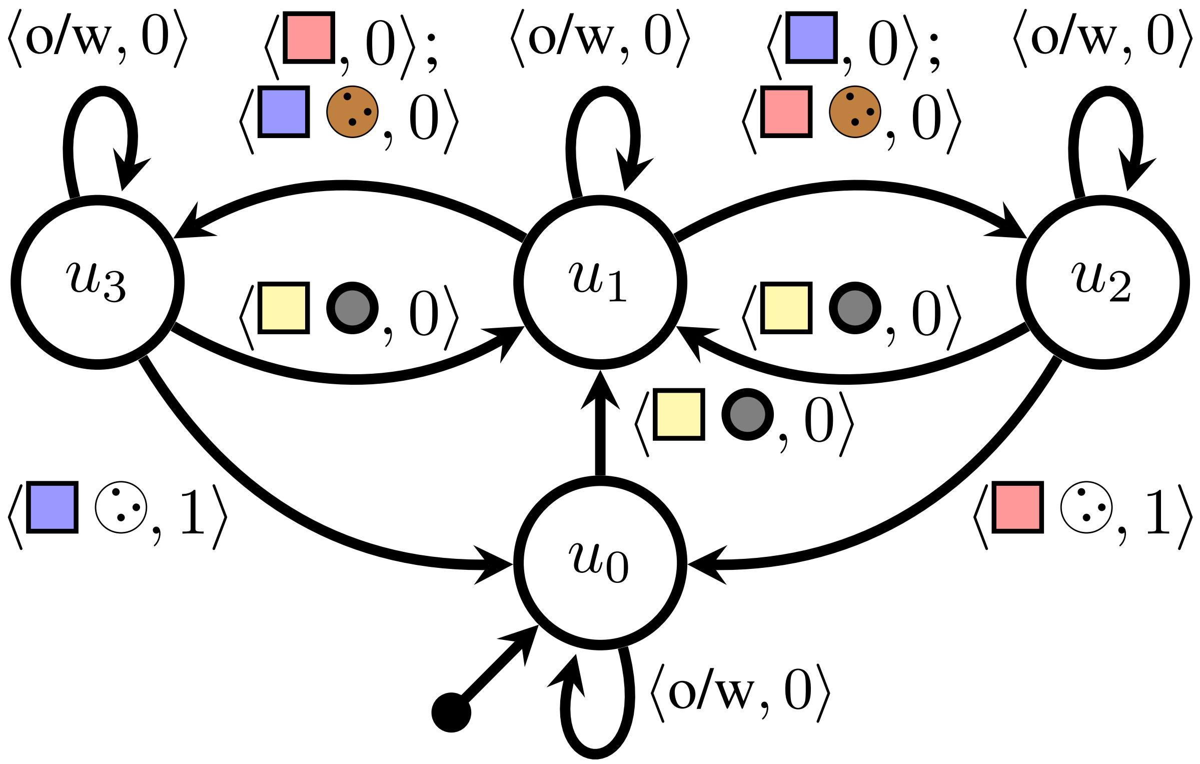
Problem: Find a policy that maximizes the external reward given by a partially observable environment

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

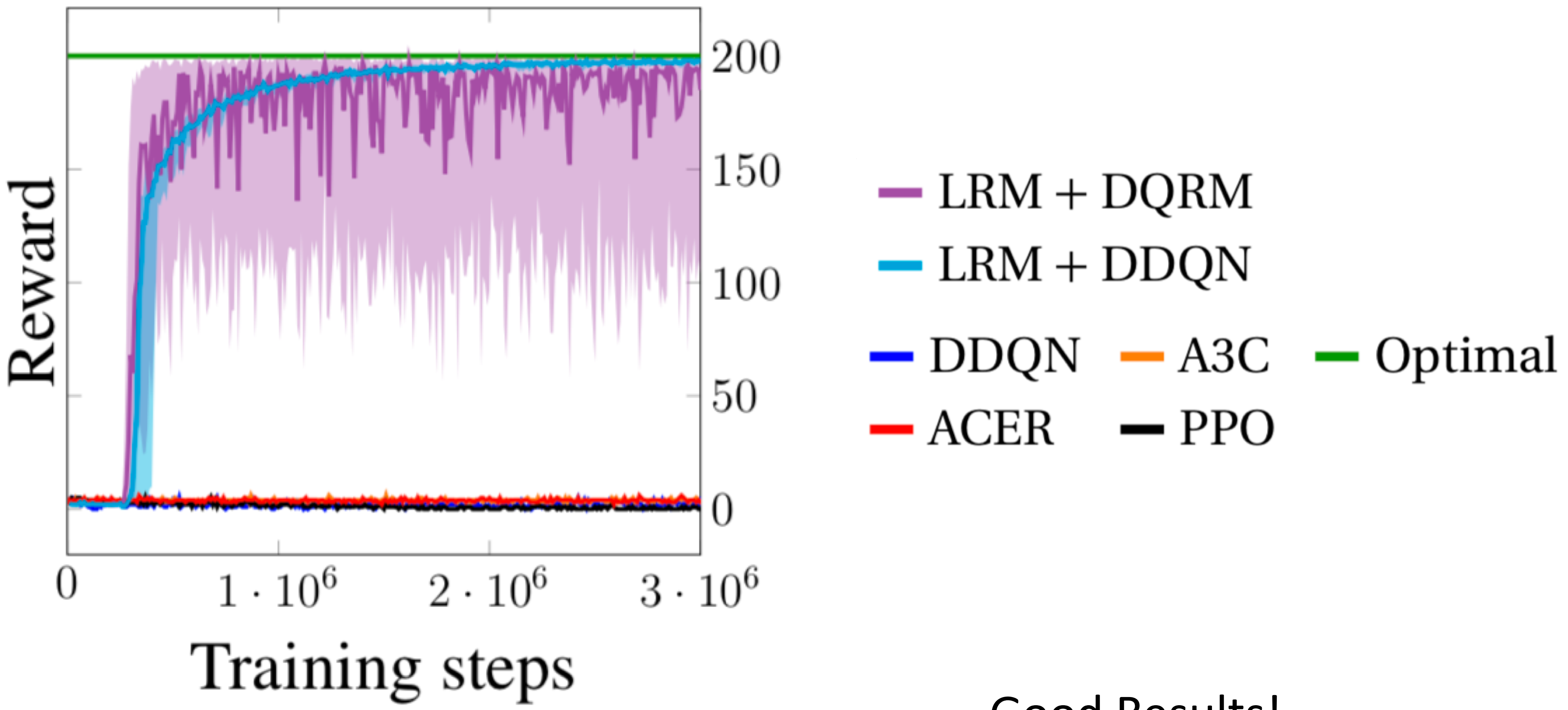
3. Learn Reward Machines (LRM)



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)



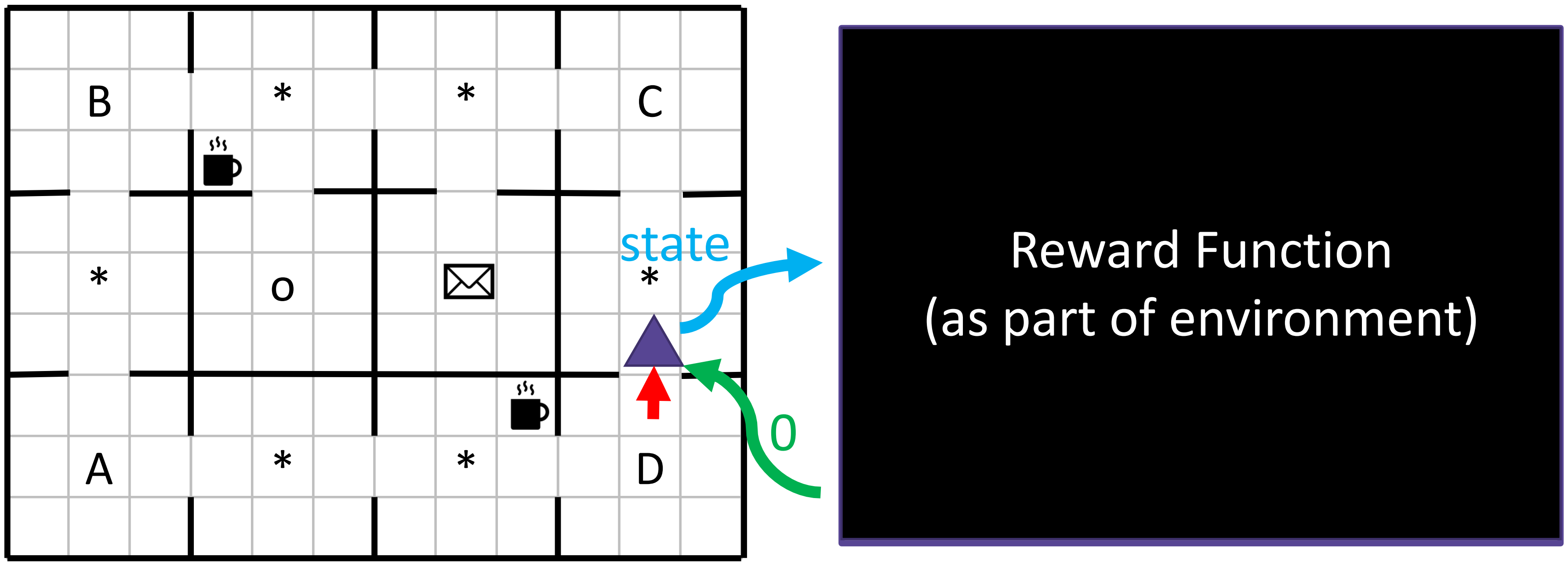
Good Results!

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

RECAP

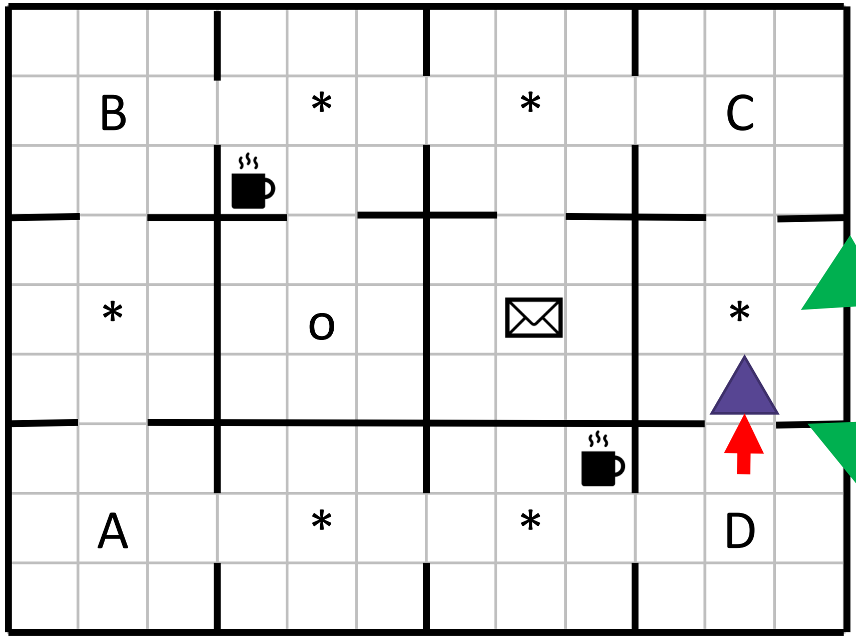
Can exploiting the alphabet and structure of language help RL agents learn and think?

Key Insight: Reveal Reward Function to the Agent



Reward Function
(as part of environment)

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
```

Contributions

- **Reward Machines (RMs):** An automata-based structure that can be used to define reward functions.
- **QRM:** An RL algorithm that exploits an RM's structure

[Camacho, Toro Icarte, Klassen, Valenzano, McIlraith, *ICML* 2018]

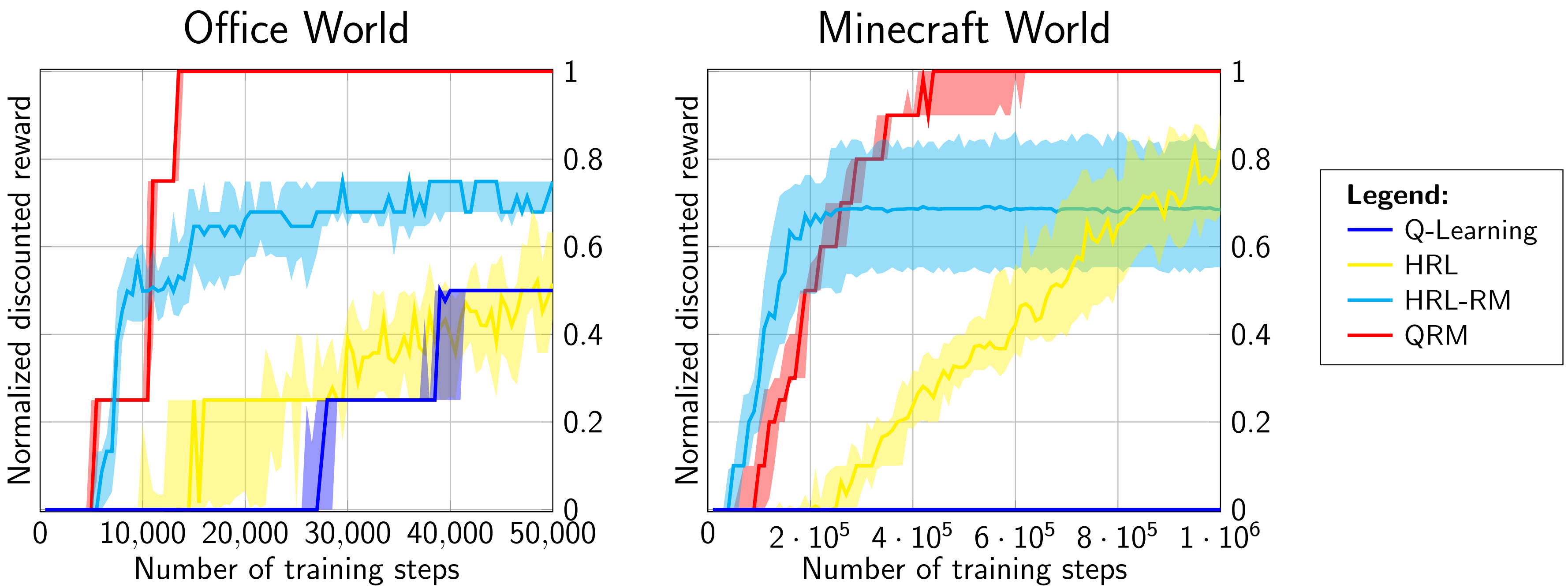
- **QRM+RS**: Automated RM-based reward shaping
- **Translation to RM from other languages:** RMs as a normal form representation for reward functions

[Camacho, Toro Icarte, Klassen, Valenzano, McIlraith, *IJCAI* 2019]

- **LRM:** learning RMs from experience in partially observable environments

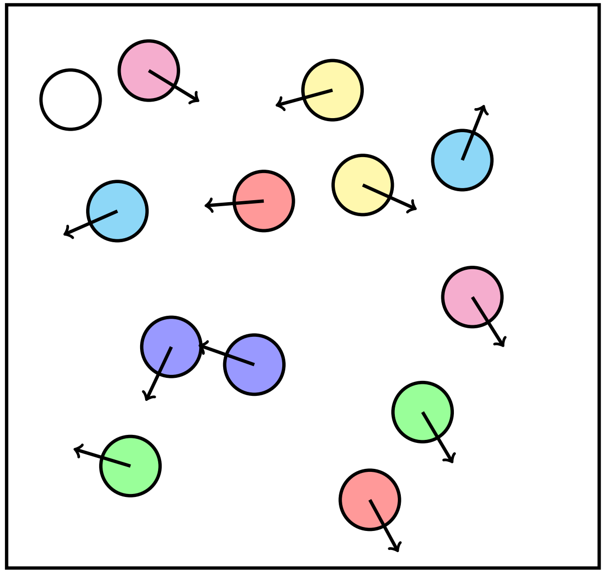
[Toro Icarte, Waldie, Klassen, Valenzano, Castro, McIlraith, *NeurIPS* 2019]

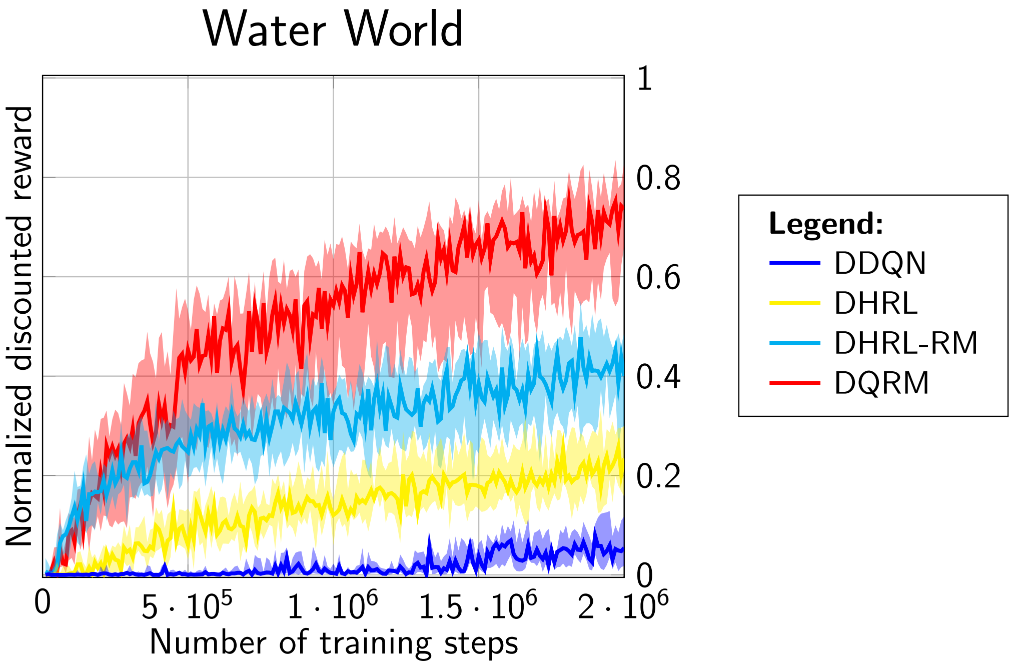
Great Results in Discrete Domains



QRM outperforms HRL and standard Q-learning in two domains

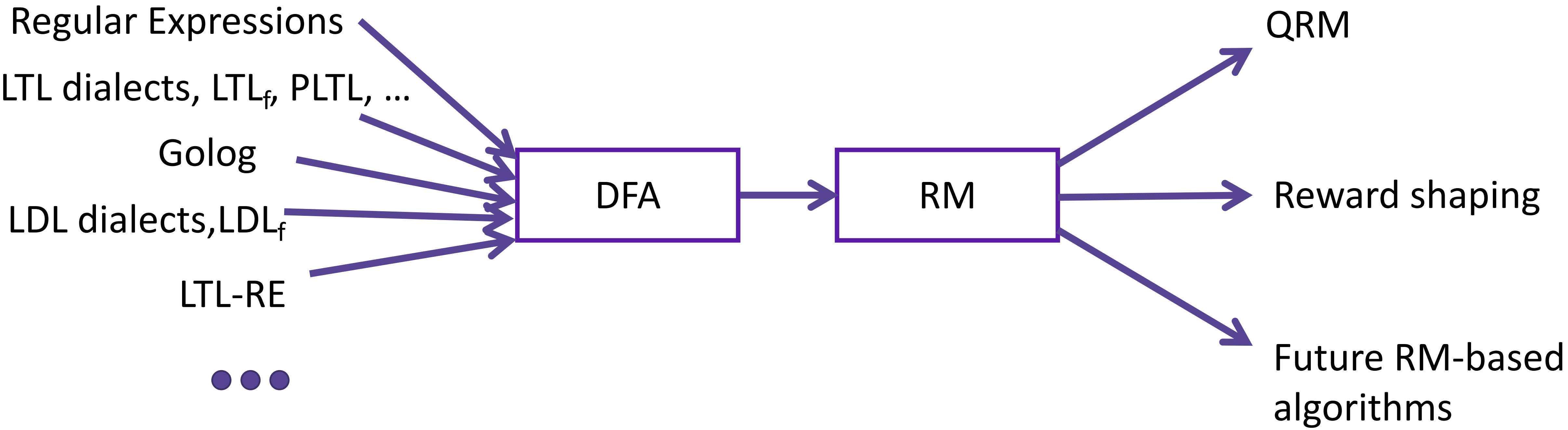
…and in Continuous Domains



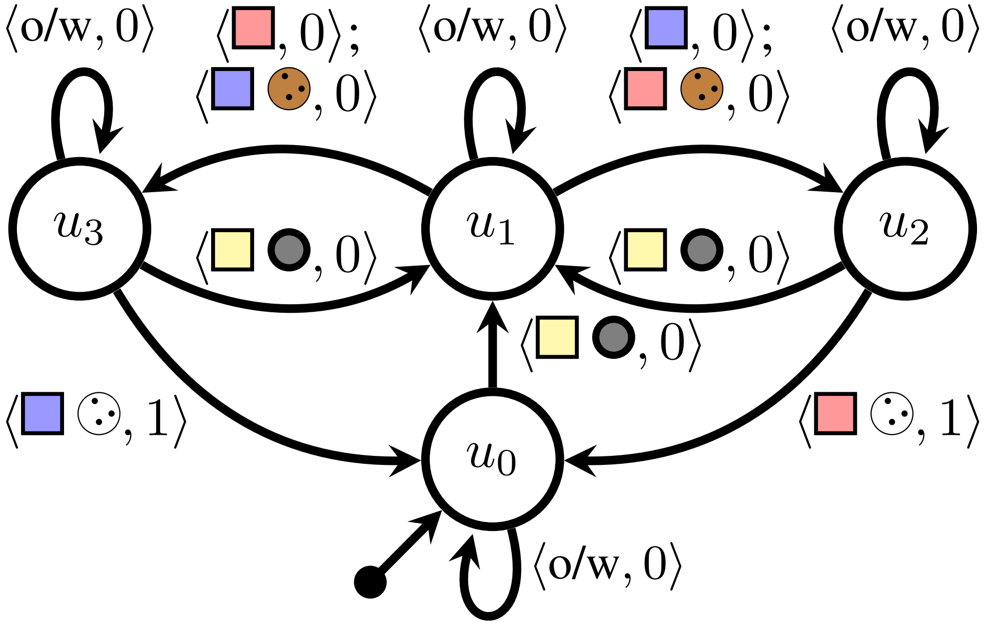


… and is also effective when combined with deep learning

We can construct RMs from a diversity of formal languages …



…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

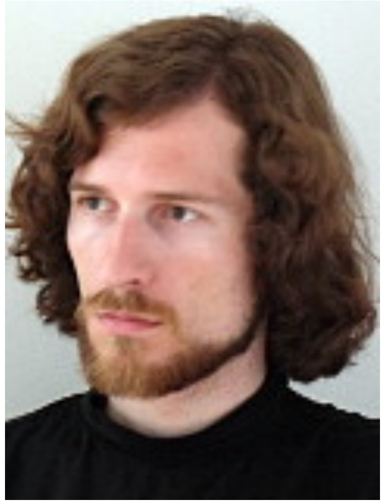
Other related work

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.

Acknowledgements

Rodrigo Toro Icarte



Toryn Klassen

Richard Valenzano

ELEMENT AI

Alberto Camacho

Ethan Waldie

Margarita Castro