

Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning

Rodrigo Toro Icarte Toryn Q. Klassen
Richard Valenzano Sheila A. McIlraith



Computer Science
UNIVERSITY OF TORONTO

ELEMENT^{AI}



ICML 2018

July 13

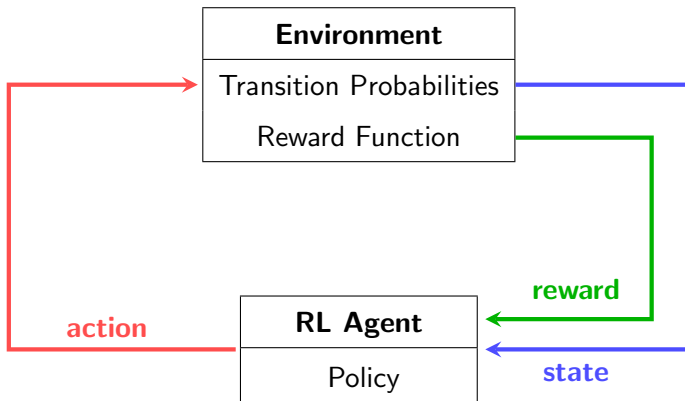
- 1 Motivation
- 2 What is a reward machine (RM)?
- 3 How to exploit a reward machine's structure
- 4 Results
- 5 Related work
- 6 Concluding remarks

- 1 Motivation**
- 2 What is a reward machine (RM)?
- 3 How to exploit a reward machine's structure
- 4 Results
- 5 Related work
- 6 Concluding remarks

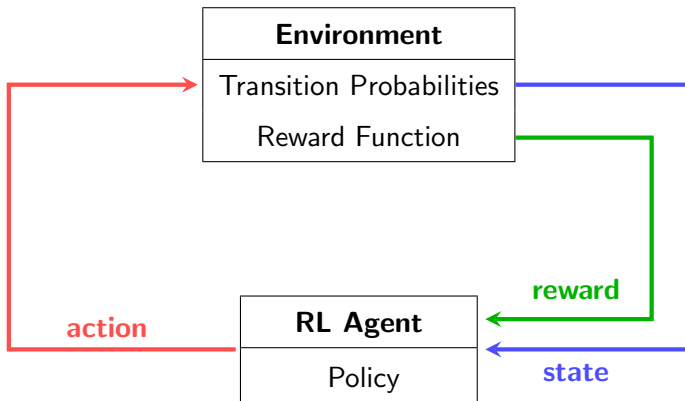
“To summarize, a nice simple idea exposing more of the structure of an RL problem and the benefits thereof.”

— Third reviewer

Reinforcement learning

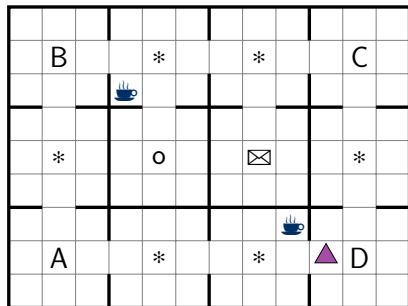


Reinforcement learning

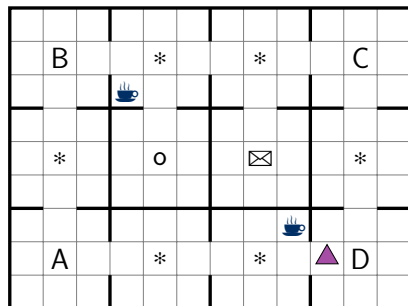


The environment might be the real world.

Running example

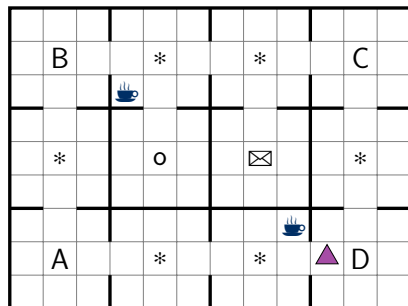


Running example



Symbol	Meaning
	Agent
	Furniture
	Coffee machine
	Mail room
	Office
A,B,C,D	Marked locations

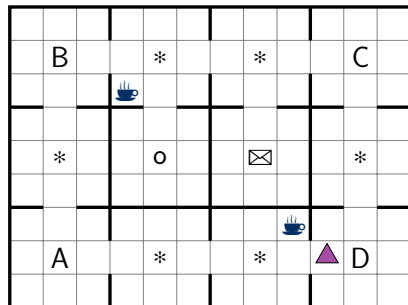
Running example



Symbol	Meaning
▲	Agent
*	Furniture
☕	Coffee machine
✉	Mail room
o	Office
A,B,C,D	Marked locations

Task: Patrol A, B, C, and D.

Running example

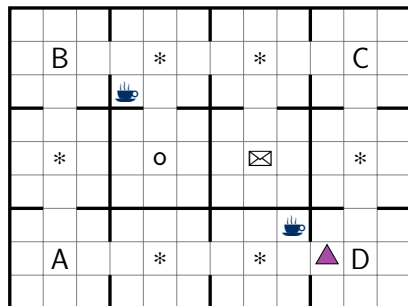


Symbol	Meaning
	Agent
	Furniture
	Coffee machine
	Mail room
	Office
A,B,C,D	Marked locations

Task: Patrol A, B, C, and D.

Someone has to program a reward function

Running example

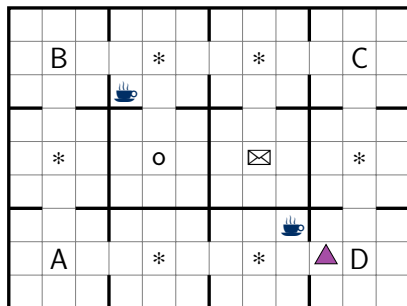


Symbol	Meaning
	Agent
	Furniture
	Coffee machine
	Mail room
	Office
A,B,C,D	Marked locations

Task: Patrol A, B, C, and D.

Someone has to program a reward function
(even if the environment is the real world).

Running example

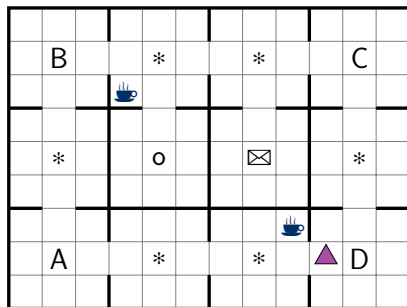


```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

Task: Patrol A, B, C, and D.

Someone has to program a reward function
(even if the environment is the real world).

Running example

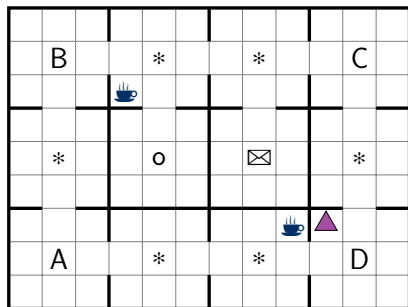


```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

Task: Patrol A, B, C, and D.

Someone has to program a reward function
(even if the environment is the real world).

Running example

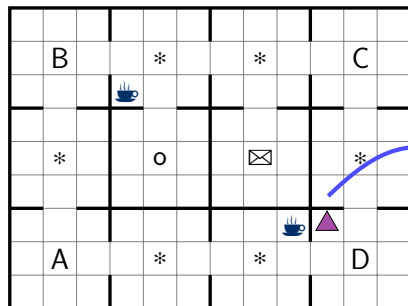


```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

Task: Patrol A, B, C, and D.

Someone has to program a reward function
(even if the environment is the real world).

Running example

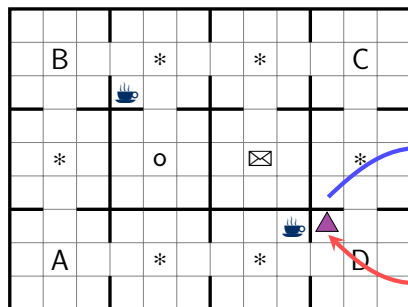


```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     Reward Function:
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

Task: Patrol A, B, C, and D.

Someone has to program a reward function
(even if the environment is the real world).

Running example

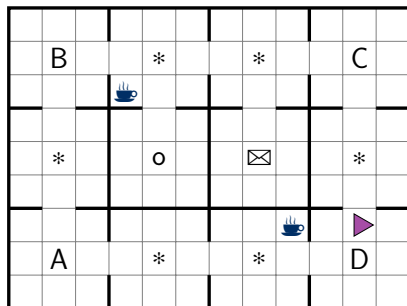


```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

Task: Patrol A, B, C, and D.

Someone has to program a reward function
(even if the environment is the real world).

Running example

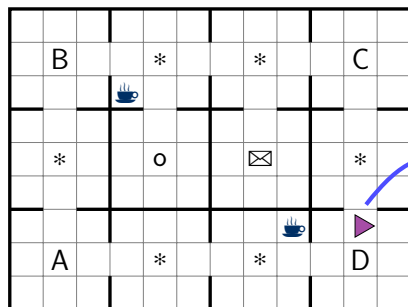


```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

Task: Patrol A, B, C, and D.

Someone has to program a reward function
(even if the environment is the real world).

Running example

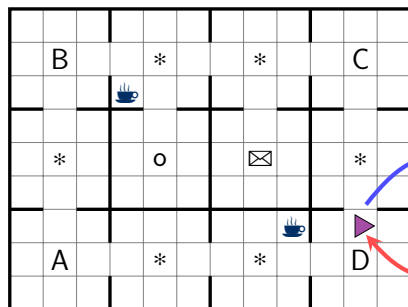


```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     Reward Function:
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

Task: Patrol A, B, C, and D.

Someone has to program a reward function
(even if the environment is the real world).

Running example



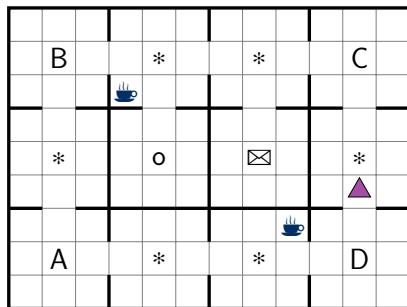
```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

Reward Function

Task: Patrol A, B, C, and D.

Someone has to program a reward function
(even if the environment is the real world).

Running example

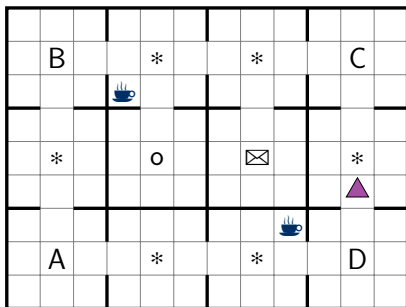


```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

Task: Patrol A, B, C, and D.

Someone has to program a reward function
(even if the environment is the real world).

Running example

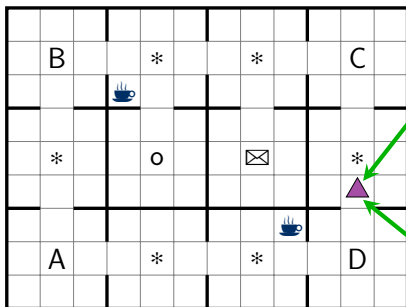


```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

Reward Function

What if we give the agent access to the reward function?

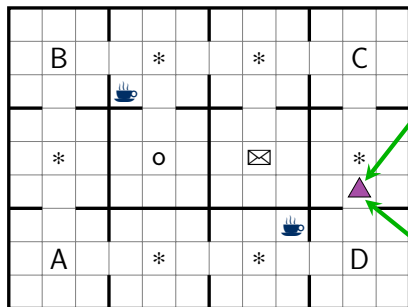
Running example



```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

What if we give the agent access to the reward function?

Running example

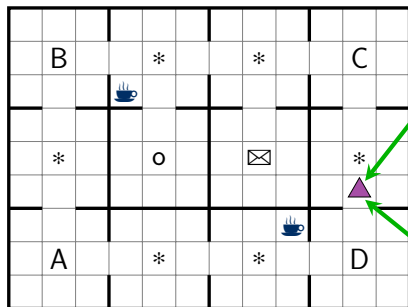


```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

What if we give the agent access to the reward function?

Is there any advantage of doing do?

Running example



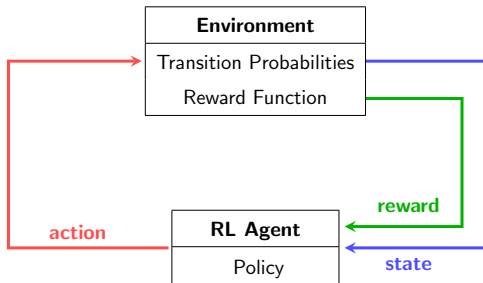
```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

What if we give the agent access to the reward function?

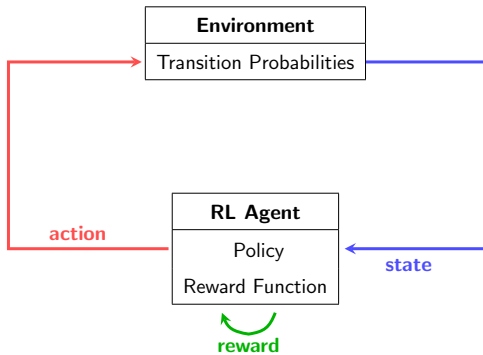
Is there any advantage of doing do?

The agent can exploit the reward structure!

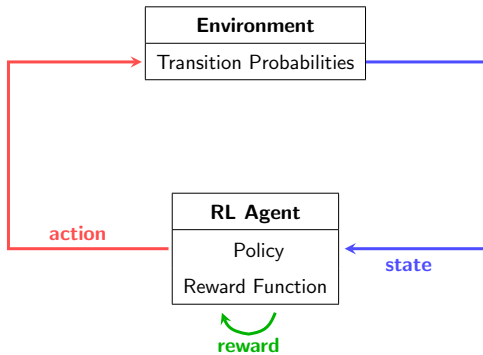
The simple idea



The simple idea

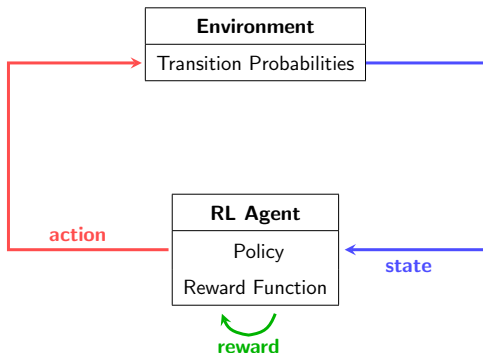


The simple idea



How to exploit the reward function definition

The simple idea



How to exploit the reward function definition

- 1 RMs:** A novel language to define reward functions.
- 2 QRM:** An RL algorithm that exploits RM's structure.

- 1 Motivation
- 2 What is a reward machine (RM)?
- 3 How to exploit a reward machine's structure
- 4 Results
- 5 Related work
- 6 Concluding remarks

- 1 Motivation
- 2 **What is a reward machine (RM)?**
- 3 How to exploit a reward machine's structure
- 4 Results
- 5 Related work
- 6 Concluding remarks

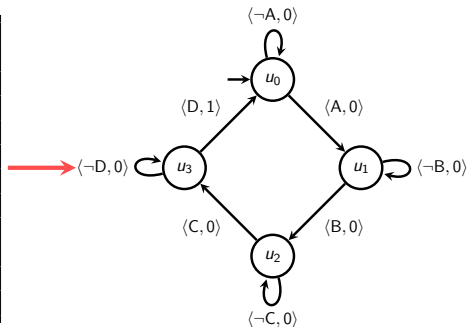
Reward machines

We encode reward functions using a finite state machine.

Reward machines

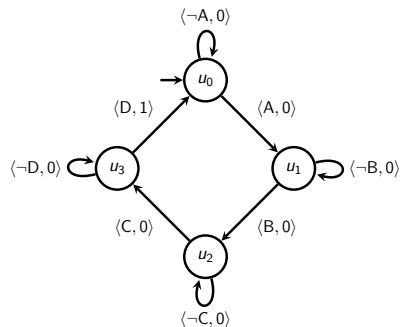
We encode reward functions using a finite state machine.

```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```



Reward machines

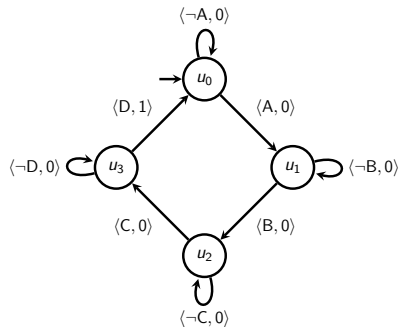
A reward machine



Reward machines

A reward machine

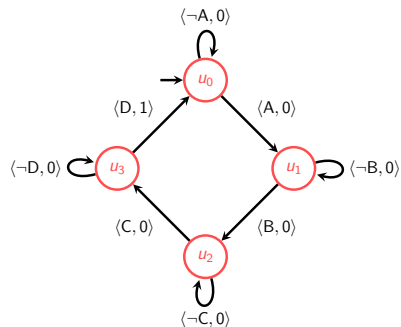
A finite set of states U



Reward machines

A reward machine

A finite set of states U

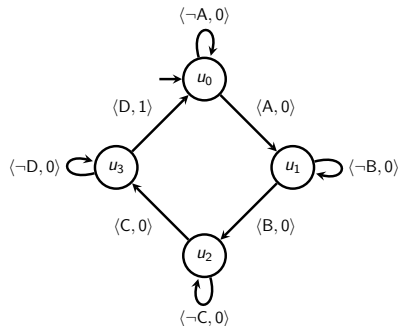


Reward machines

A reward machine

A finite set of states U

An initial state $u_0 \in U$

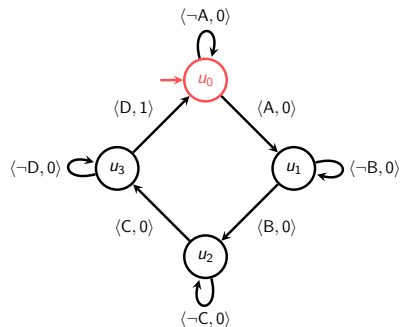


Reward machines

A reward machine

A finite set of states U

An initial state $u_0 \in U$



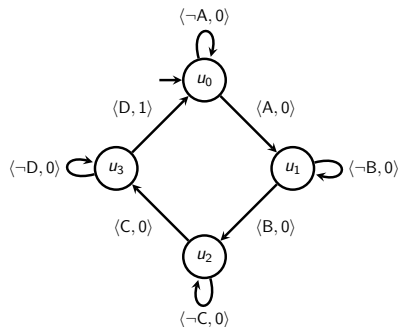
Reward machines

A reward machine

A finite set of states U

An initial state $u_0 \in U$

A set of transitions labelled by:



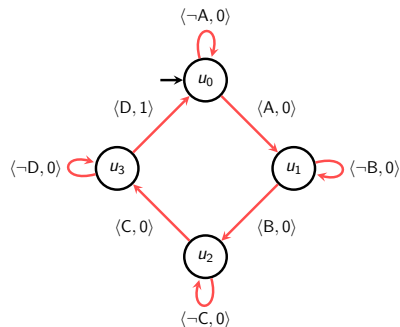
Reward machines

A reward machine

A finite set of states U

An initial state $u_0 \in U$

A set of transitions labelled by:



Reward machines

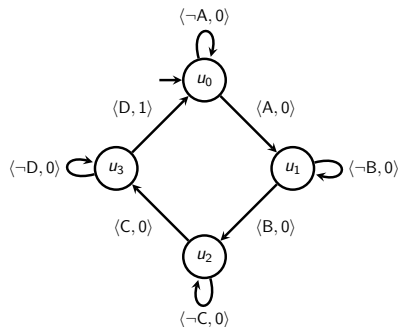
A reward machine

A finite set of states U

An initial state $u_0 \in U$

A set of transitions labelled by:

- a logical condition and



Reward machines

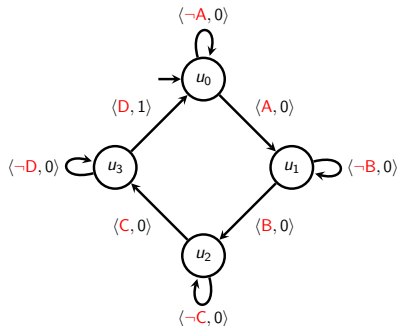
A reward machine

A finite set of states U

An initial state $u_0 \in U$

A set of transitions labelled by:

- a logical condition and



Reward machines

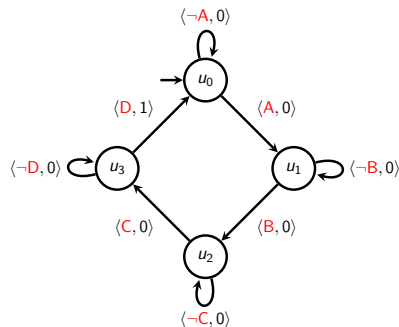
A reward machine

A finite set of states U

An initial state $u_0 \in U$

A set of transitions labelled by:

- a logical condition and



Conditions are over properties of the current state:

$$\mathcal{P} = \{\text{☕}, \text{✉}, o, *, A, B, C, D\}$$

Reward machines

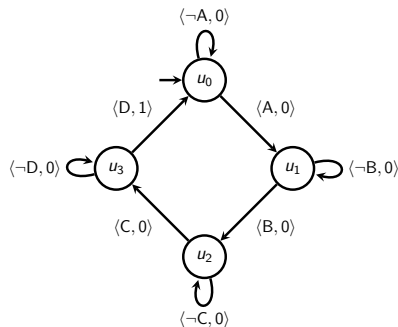
A reward machine

A finite set of states U

An initial state $u_0 \in U$

A set of transitions labelled by:

- a logical condition and
- a reward function.



Conditions are over properties of the current state:

$$\mathcal{P} = \{ \text{☕}, \text{✉}, o, *, A, B, C, D \}$$

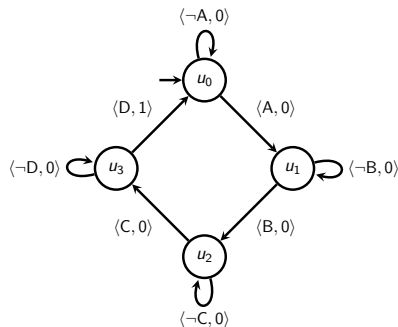
A simple reward machine

A finite set of states U

An initial state $u_0 \in U$

A set of transitions labelled by:

- a logical condition and
- a reward (constant number).



Conditions are over properties of the current state:

$$\mathcal{P} = \{ \text{☕}, \text{✉}, o, *, A, B, C, D \}$$

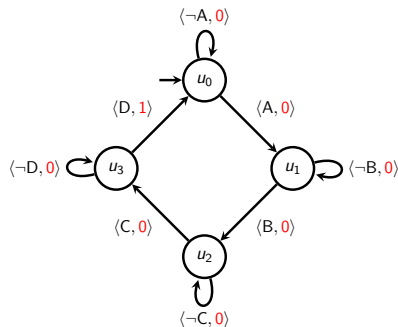
A simple reward machine

A finite set of states U

An initial state $u_0 \in U$

A set of transitions labelled by:

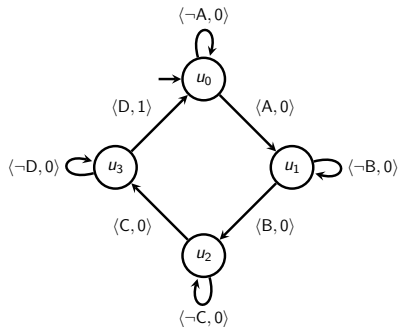
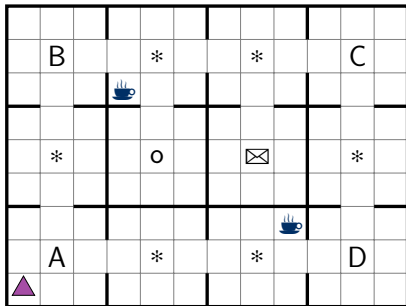
- a logical condition and
- a reward (constant number).



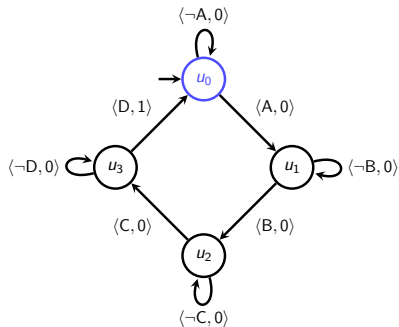
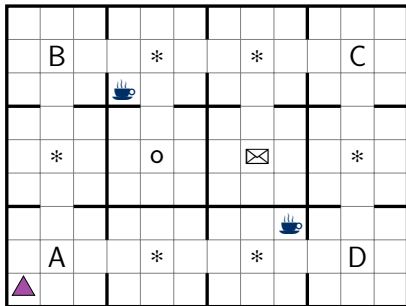
Conditions are over properties of the current state:

$$\mathcal{P} = \{ \text{☕}, \text{✉}, o, *, A, B, C, D \}$$

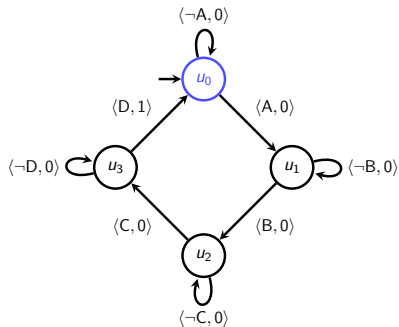
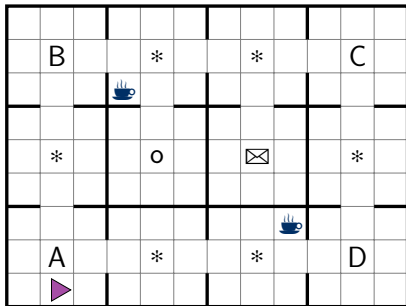
Reward machines in action



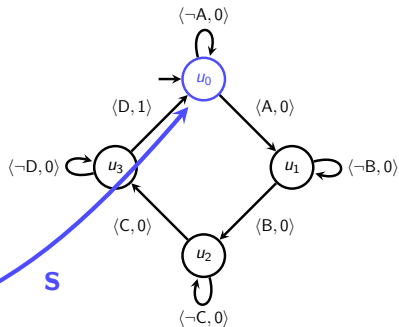
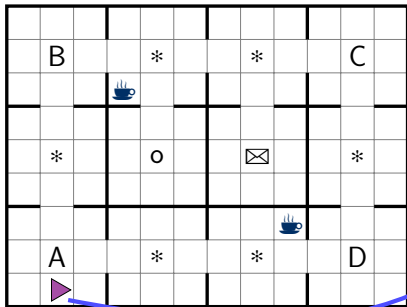
Reward machines in action



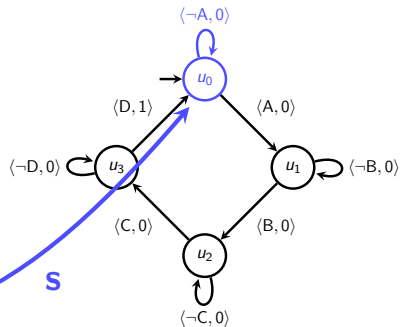
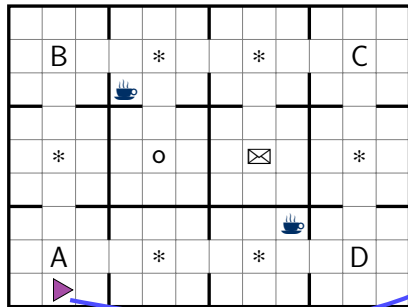
Reward machines in action



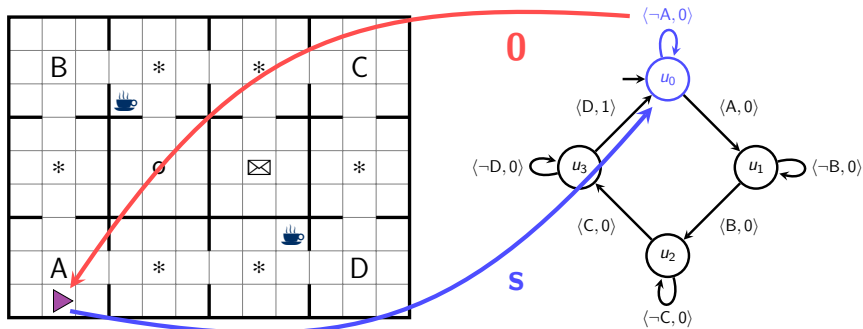
Reward machines in action



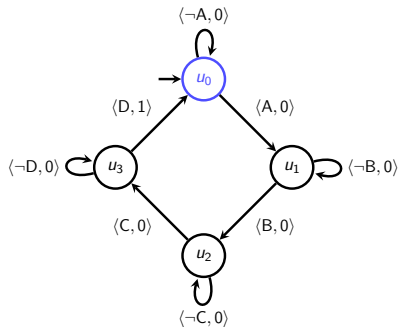
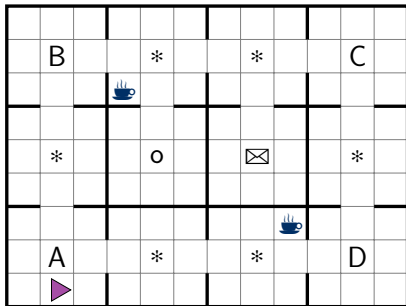
Reward machines in action



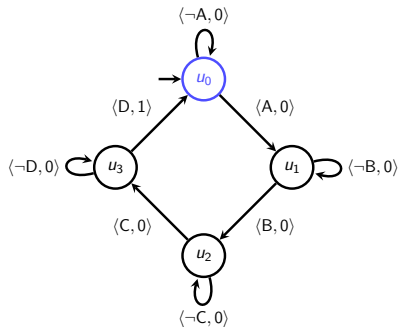
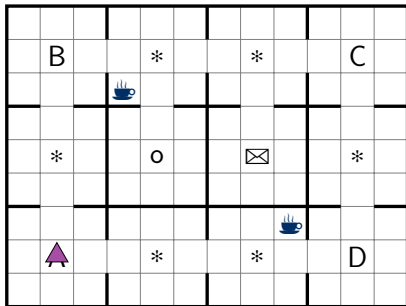
Reward machines in action



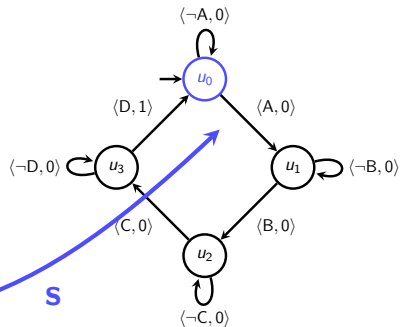
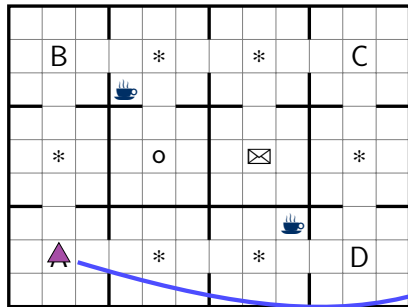
Reward machines in action



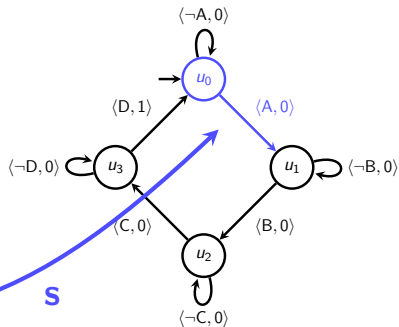
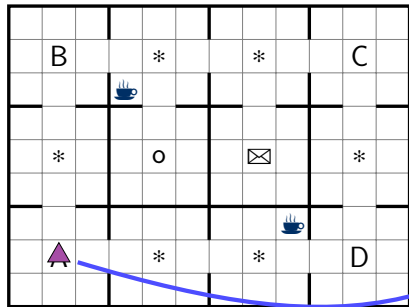
Reward machines in action



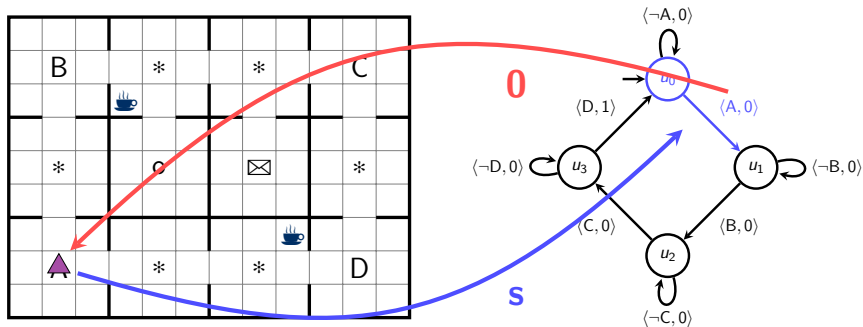
Reward machines in action



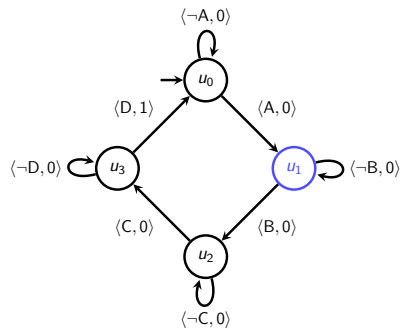
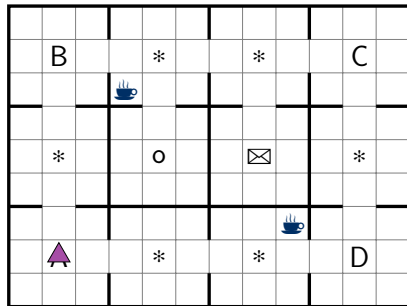
Reward machines in action



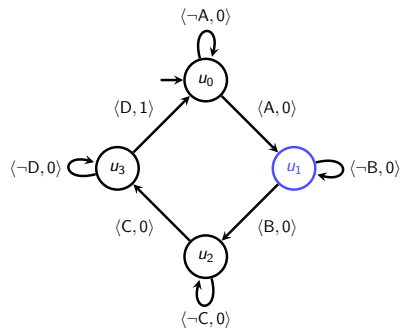
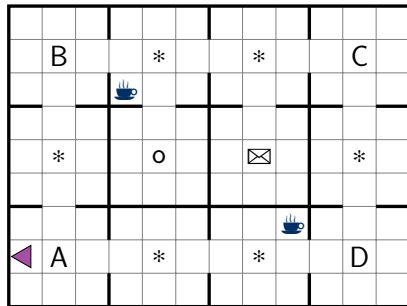
Reward machines in action



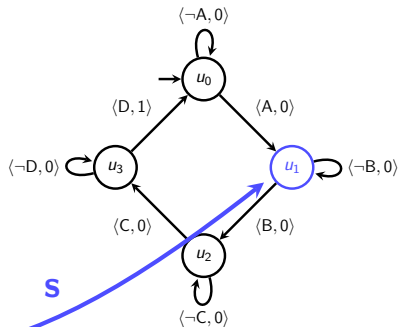
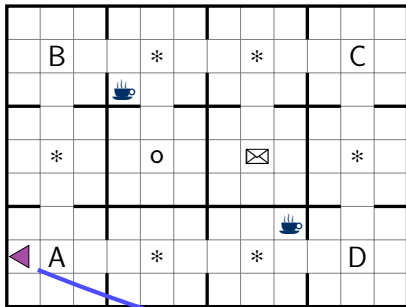
Reward machines in action



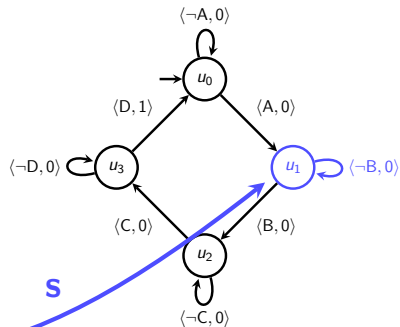
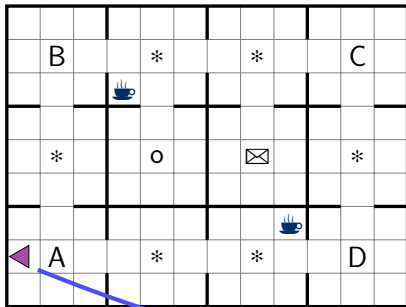
Reward machines in action



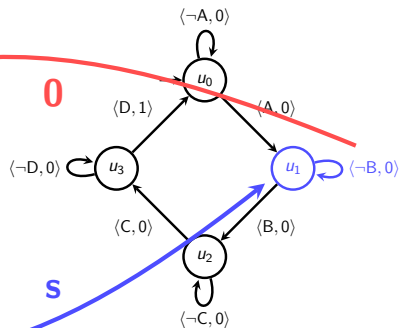
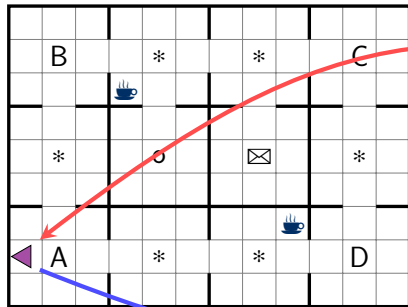
Reward machines in action



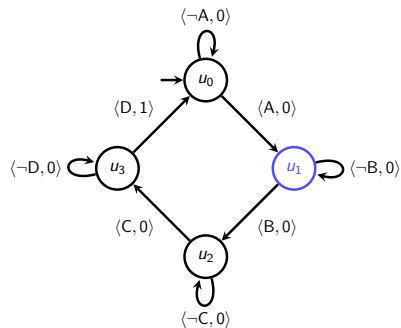
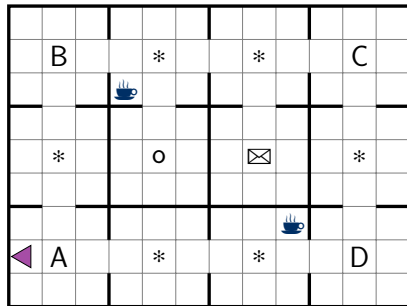
Reward machines in action



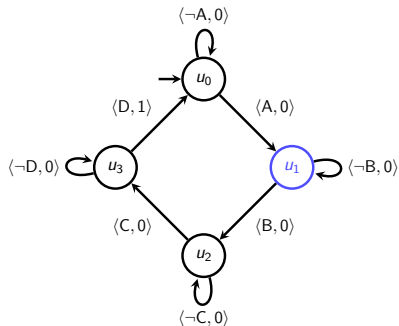
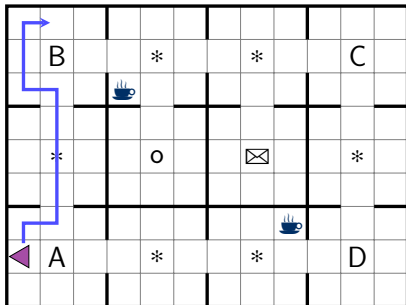
Reward machines in action



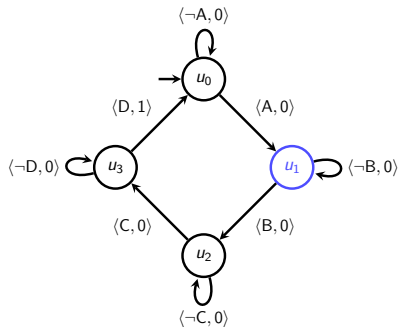
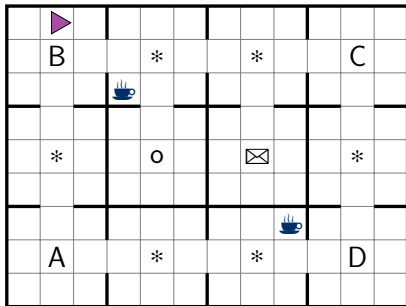
Reward machines in action



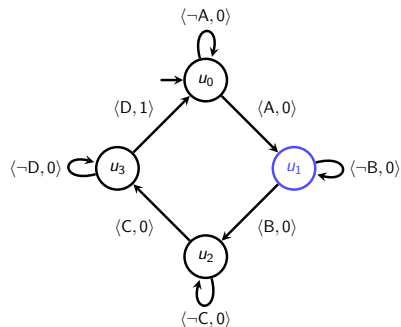
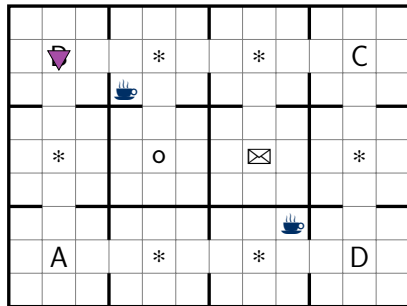
Reward machines in action



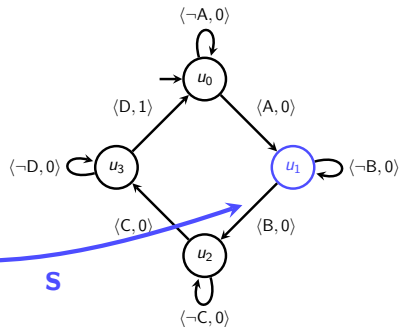
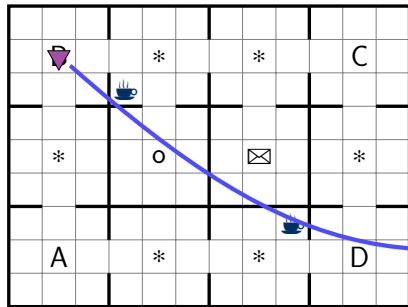
Reward machines in action



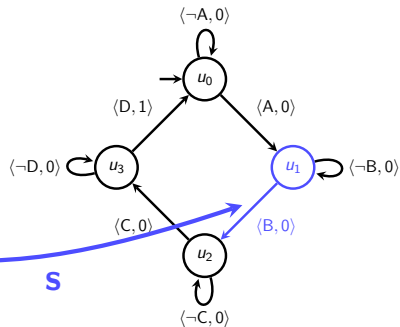
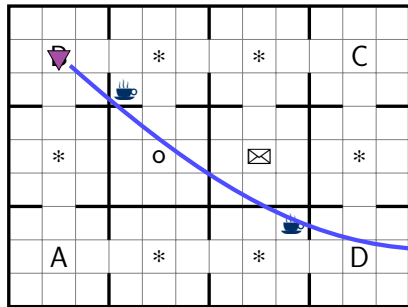
Reward machines in action



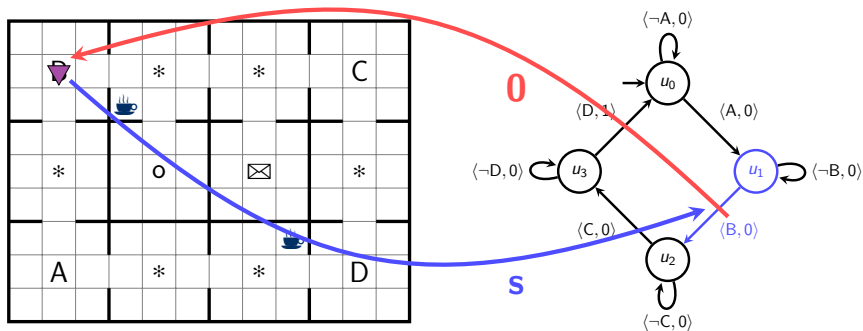
Reward machines in action



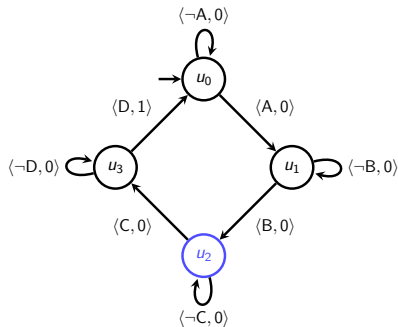
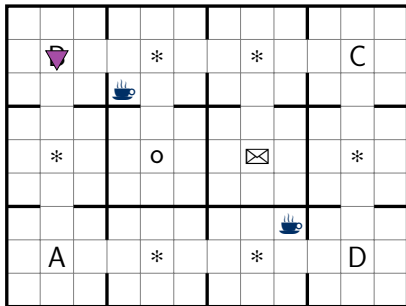
Reward machines in action



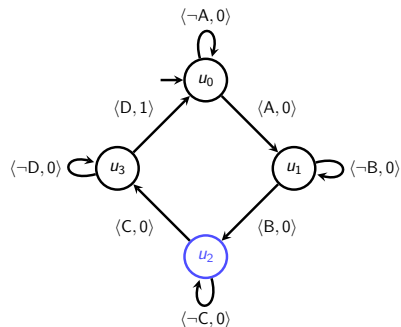
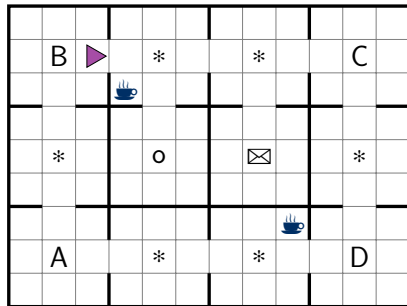
Reward machines in action



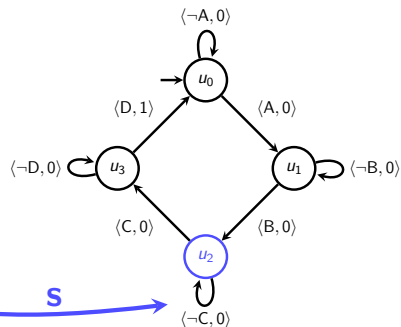
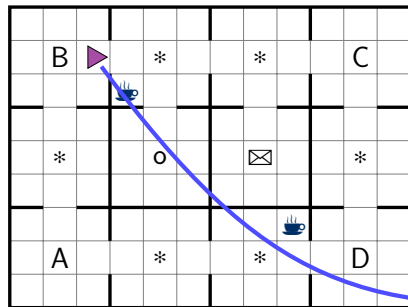
Reward machines in action



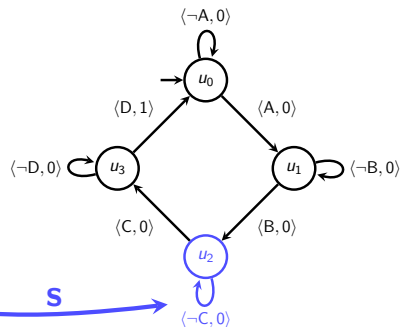
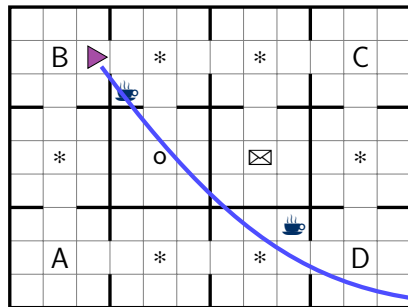
Reward machines in action



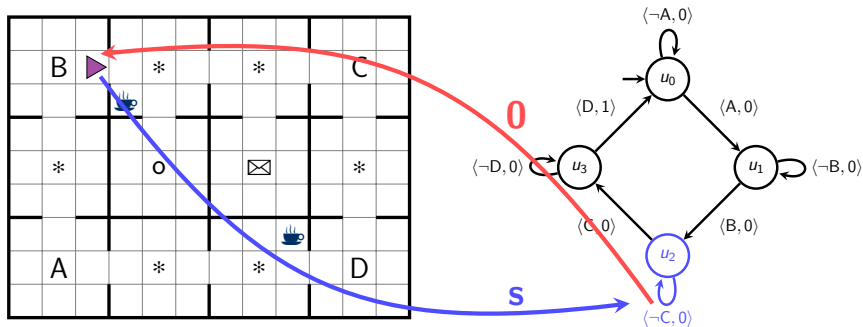
Reward machines in action



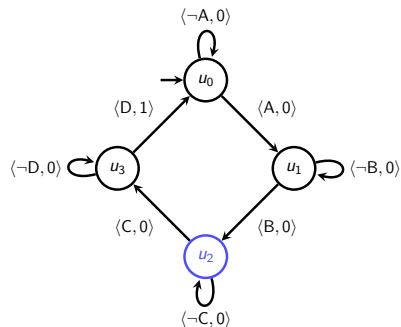
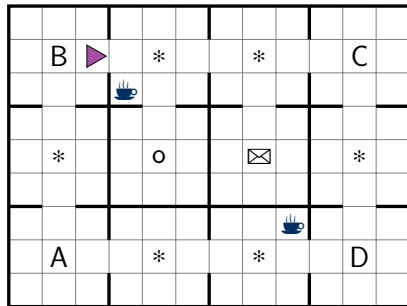
Reward machines in action



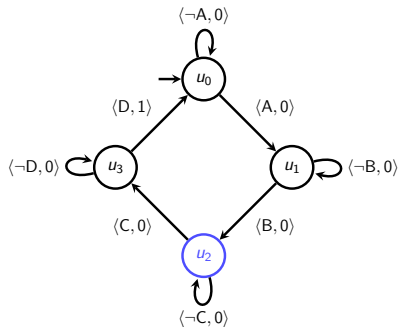
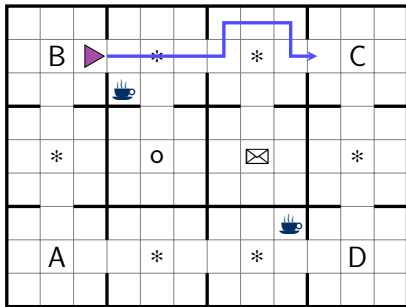
Reward machines in action



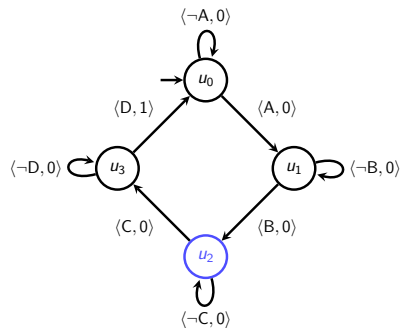
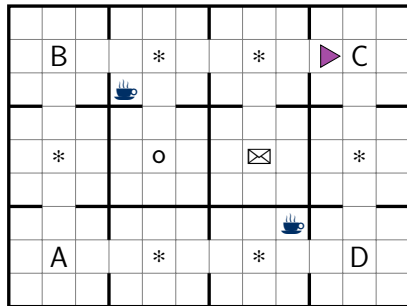
Reward machines in action



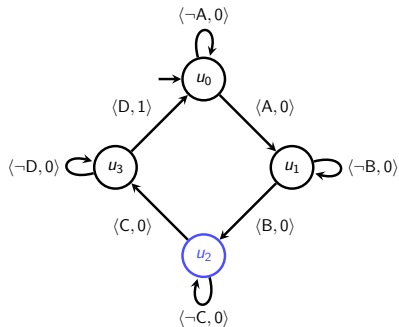
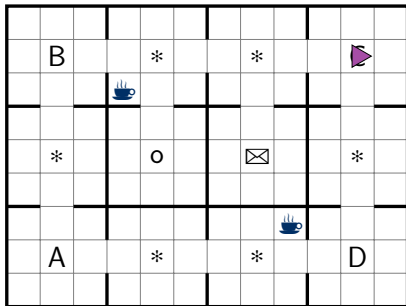
Reward machines in action



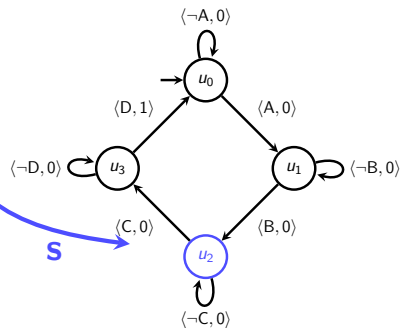
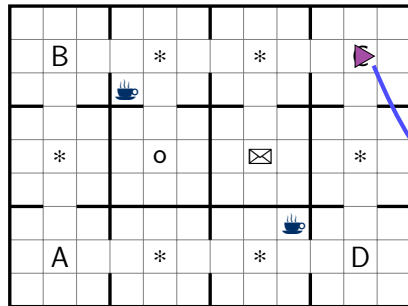
Reward machines in action



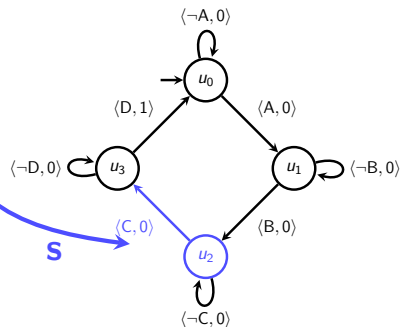
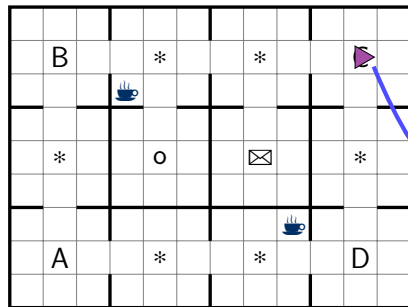
Reward machines in action



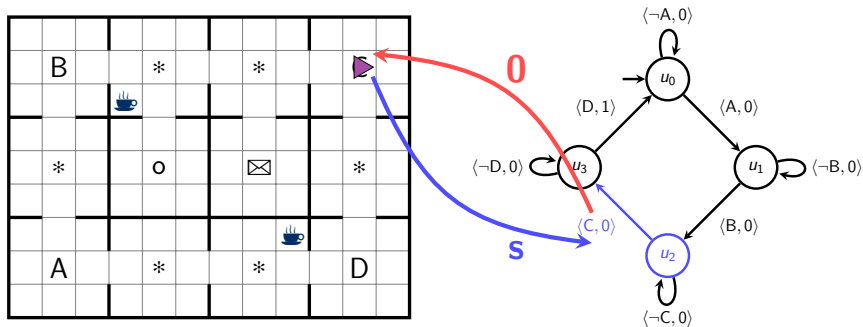
Reward machines in action



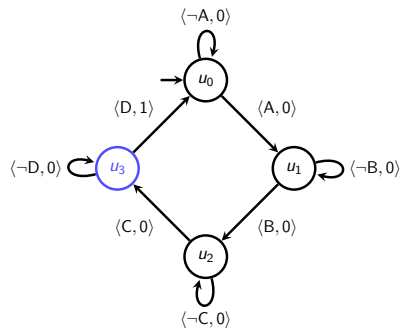
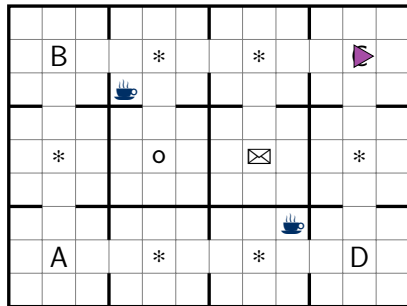
Reward machines in action



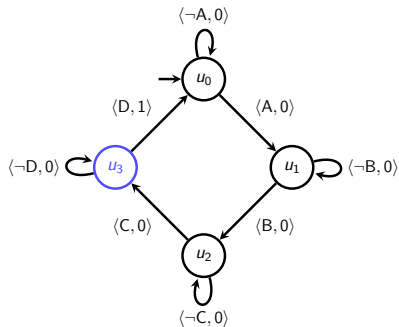
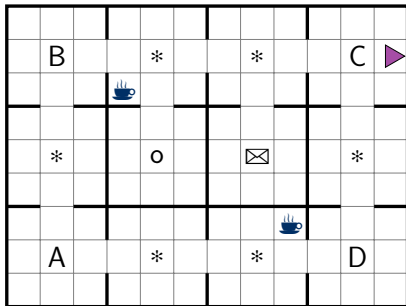
Reward machines in action



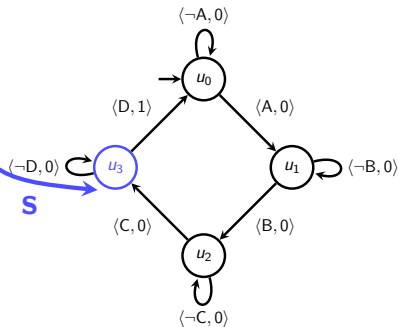
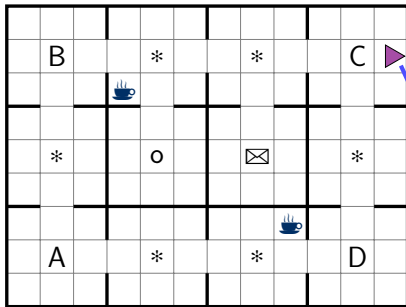
Reward machines in action



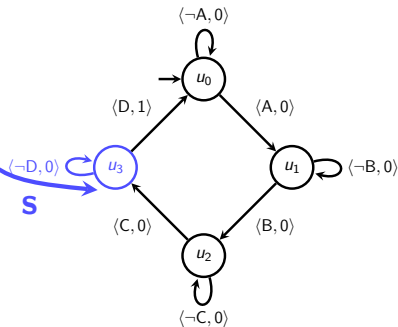
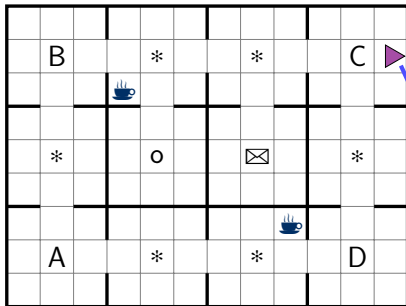
Reward machines in action



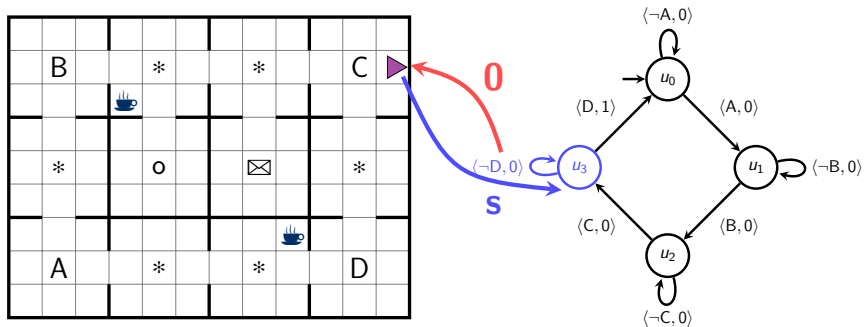
Reward machines in action



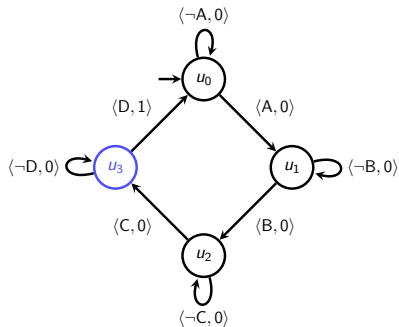
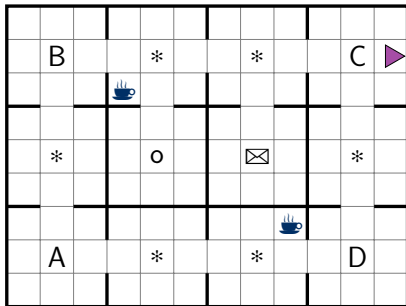
Reward machines in action



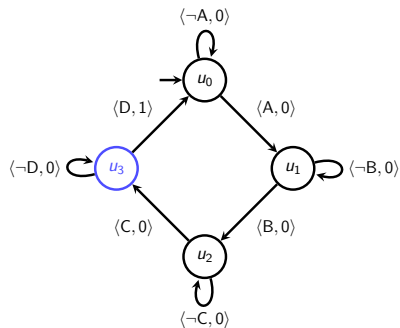
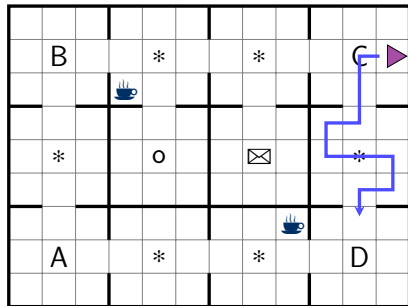
Reward machines in action



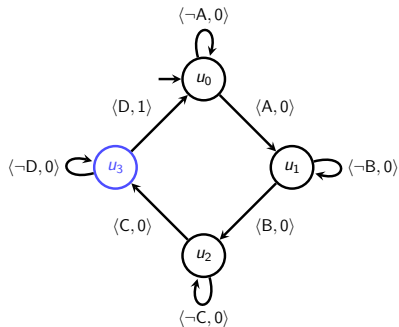
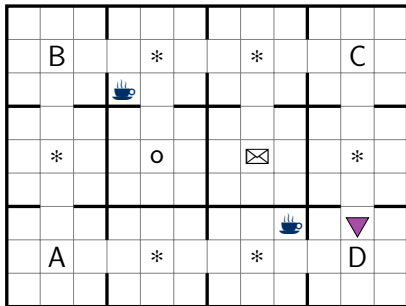
Reward machines in action



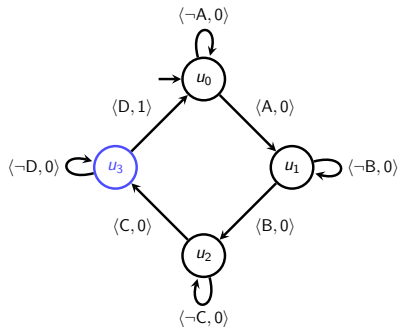
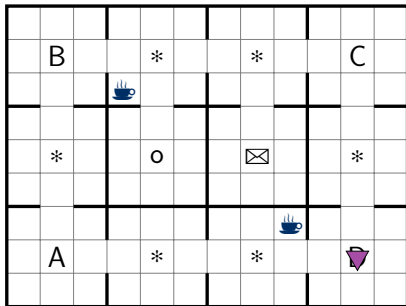
Reward machines in action



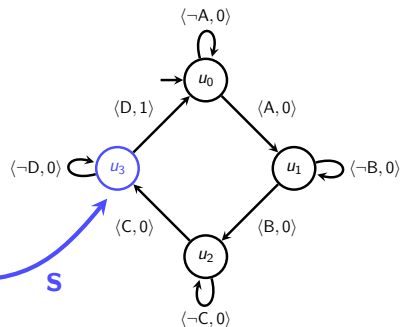
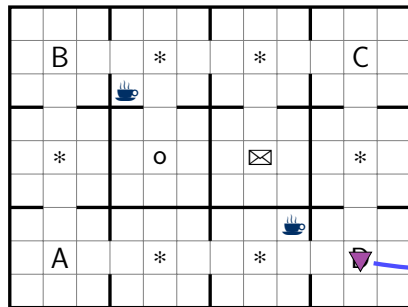
Reward machines in action



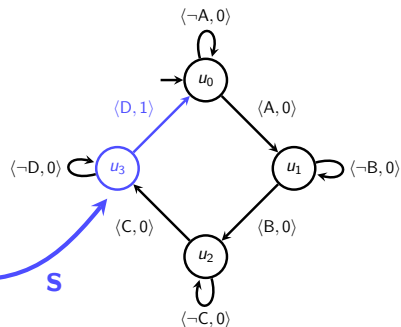
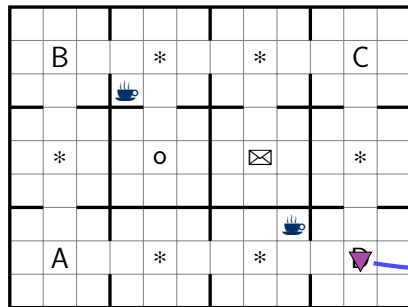
Reward machines in action



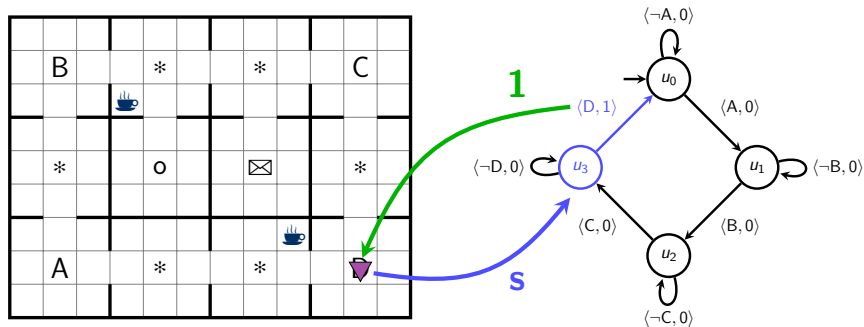
Reward machines in action



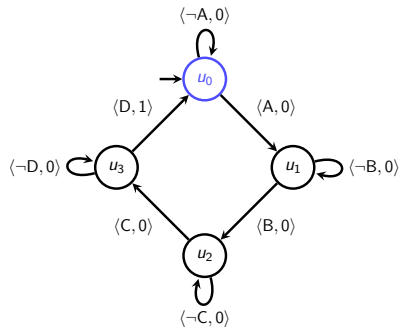
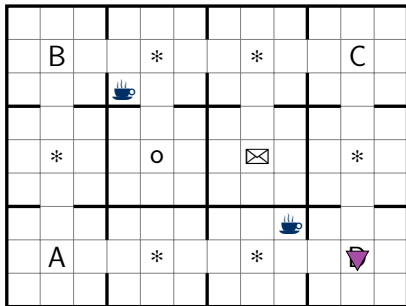
Reward machines in action



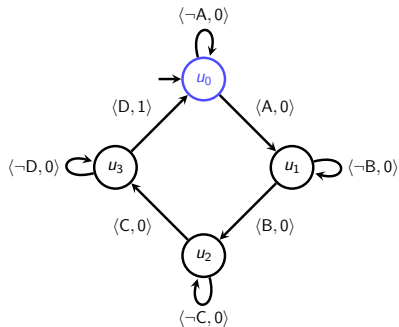
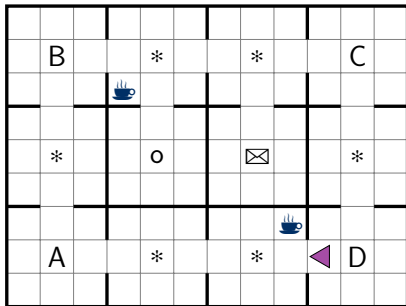
Reward machines in action



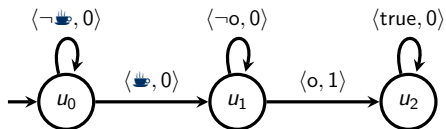
Reward machines in action



Reward machines in action

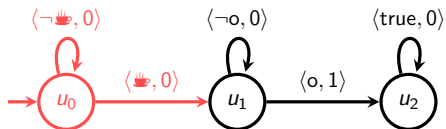


Task: Deliver coffee to the office.



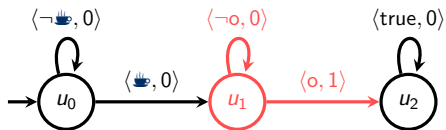
Other reward machines

Task: Deliver coffee to the office.



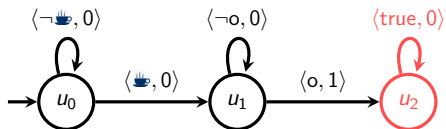
Other reward machines

Task: Deliver coffee to the office.



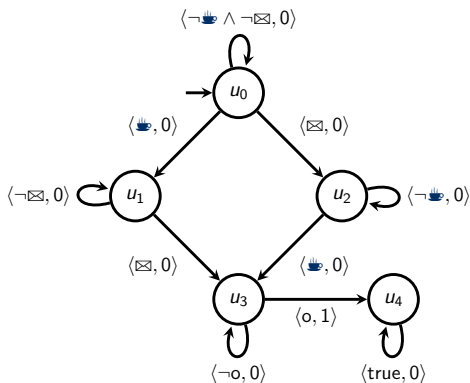
Other reward machines

Task: Deliver coffee to the office.



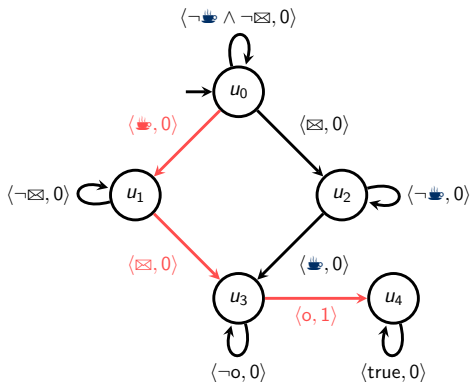
Other reward machines

Task: Deliver coffee and the mail to the office.



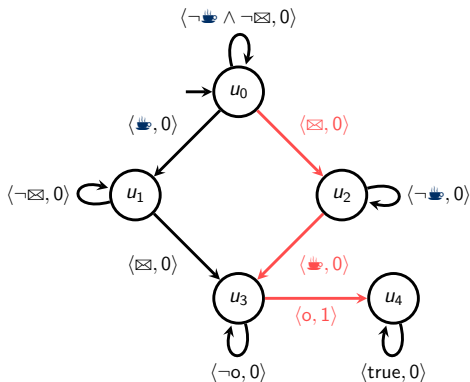
Other reward machines

Task: Deliver coffee and the mail to the office.



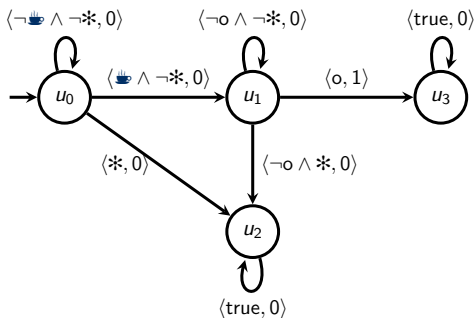
Other reward machines

Task: Deliver coffee and the mail to the office.



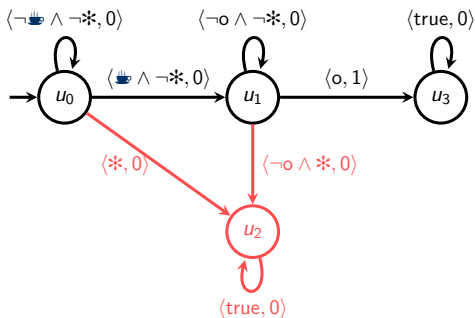
Task: Deliver coffee to the office while avoiding the furniture.

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



Other reward machines

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



- 1 Motivation
- 2 What is a reward machine (RM)?
- 3 How to exploit a reward machine's structure
- 4 Results
- 5 Related work
- 6 Concluding remarks

- 1 Motivation
- 2 What is a reward machine (RM)?
- 3 **How to exploit a reward machine's structure**
- 4 Results
- 5 Related work
- 6 Concluding remarks

Exploiting reward machines' structure

We explored 4 ideas.

We explored 4 ideas.

Baselines:

- Q-Learning over a cross-product MDP (q-learning).
- Hierarchical RL based on options (HRL).
- Hierarchical RL with option pruning (HRL-RM).

We explored 4 ideas.

Baselines:

- Q-Learning over a cross-product MDP (q-learning).
- Hierarchical RL based on options (HRL).
- Hierarchical RL with option pruning (HRL-RM).

Our approach:

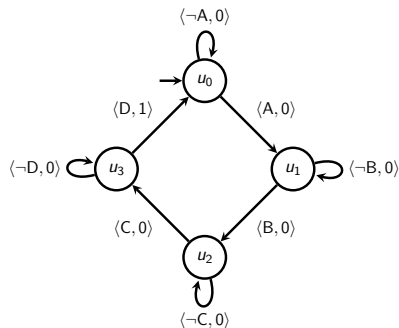
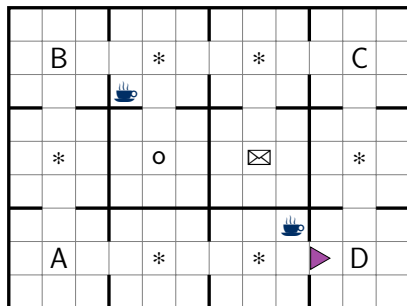
- Q-learning for Reward Machines (QRM).

Q-learning baseline

Reward machines might define non-Markovian rewards.

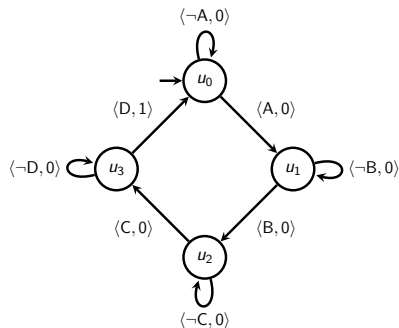
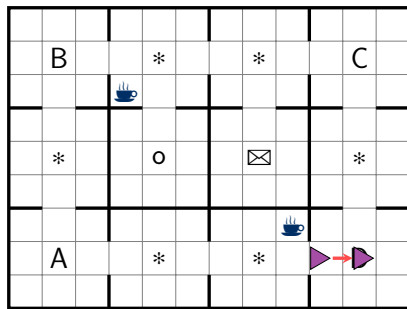
Q-learning baseline

Reward machines might define non-Markovian rewards.



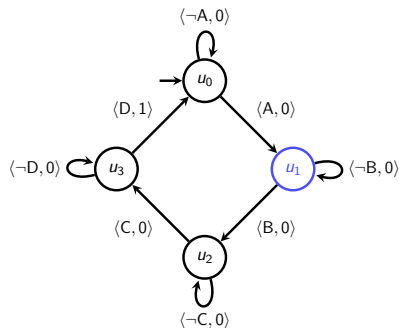
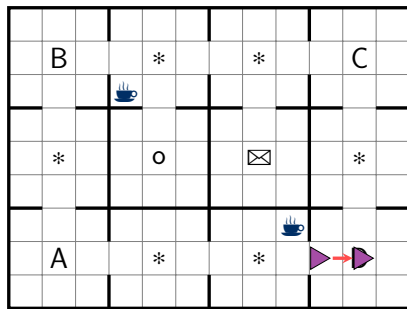
Q-learning baseline

Reward machines might define non-Markovian rewards.



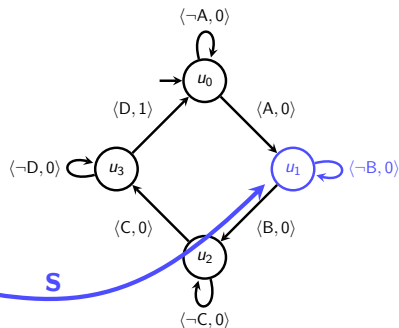
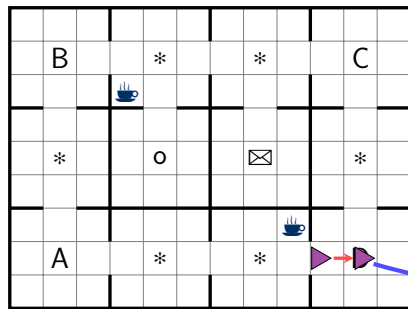
Q-learning baseline

Reward machines might define non-Markovian rewards.



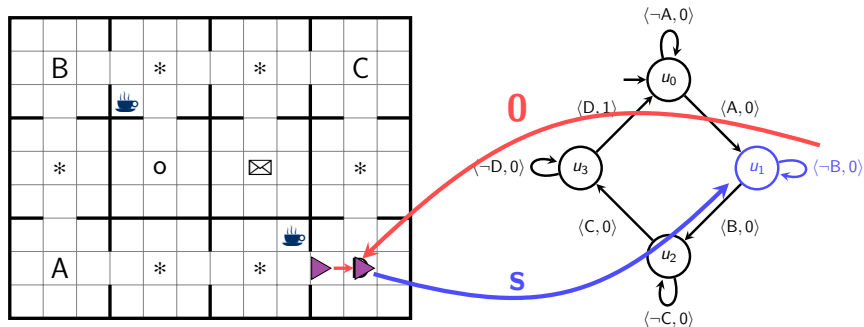
Q-learning baseline

Reward machines might define non-Markovian rewards.



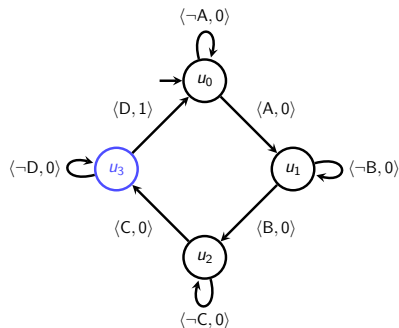
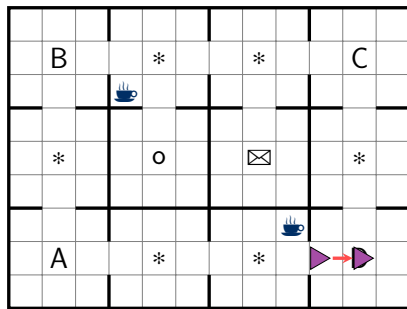
Q-learning baseline

Reward machines might define non-Markovian rewards.



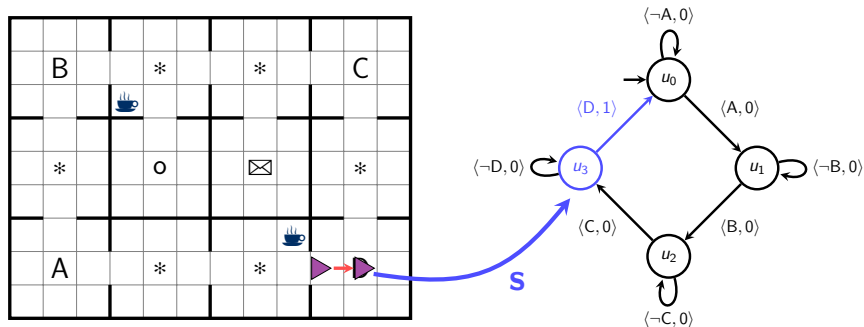
Q-learning baseline

Reward machines might define non-Markovian rewards.



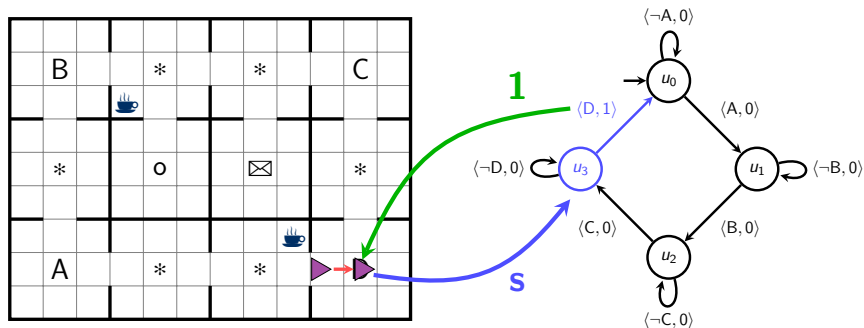
Q-learning baseline

Reward machines might define non-Markovian rewards.



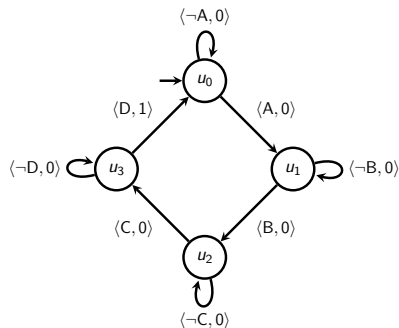
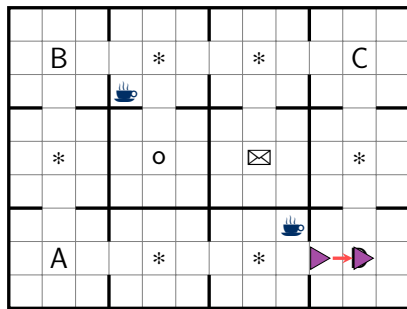
Q-learning baseline

Reward machines might define non-Markovian rewards.



Q-learning baseline

Reward machines might define non-Markovian rewards.



Solution (q-learning baseline)

- Include the RM state to the agent's state representation.
- Learn policies using standard q-learning.

Hierarchical RL baseline

Hierarchical RL baseline

HRL baseline

Learn meta-controller over a set of options (macro-actions).

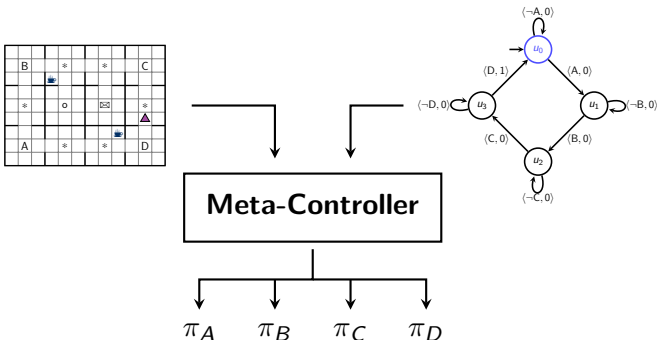
Define one option per proposition in the RM's transitions.

Optimize π_i to satisfy i optimally.

Hierarchical RL baseline

HRL baseline

Learn meta-controller over a set of options (macro-actions).
Define one option per proposition in the RM's transitions.
Optimize π_i to satisfy i optimally.



Hierarchical RL with RM pruning baseline

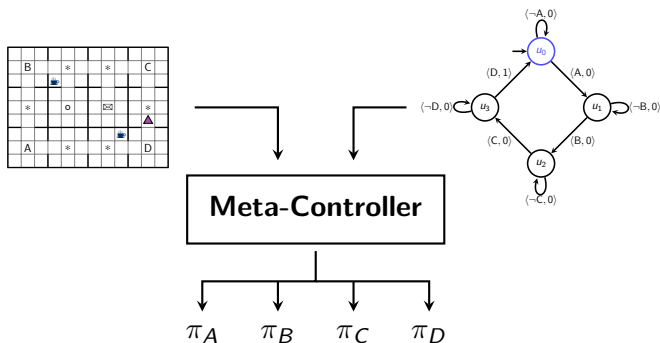
HRL-RM baseline

Prune useless options using the current reward machine state.

Hierarchical RL with RM pruning baseline

HRL-RM baseline

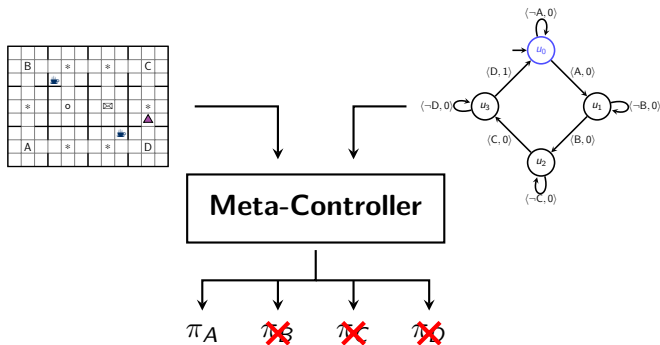
Prune useless options using the current reward machine state.



Hierarchical RL with RM pruning baseline

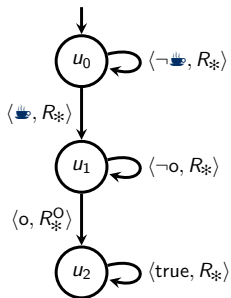
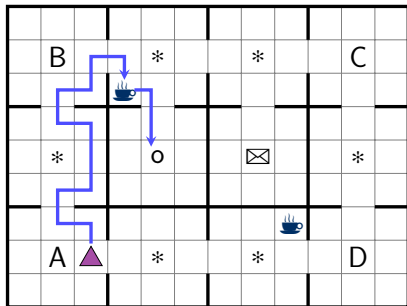
HRL-RM baseline

Prune useless options using the current reward machine state.

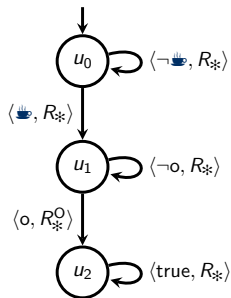
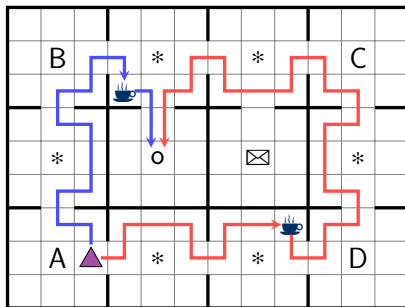


Hierarchical RL might converge to suboptimal policies

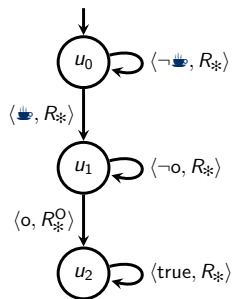
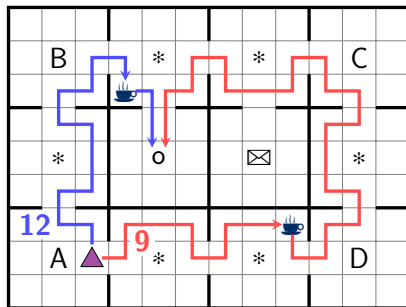
Hierarchical RL might converge to suboptimal policies



Hierarchical RL might converge to suboptimal policies



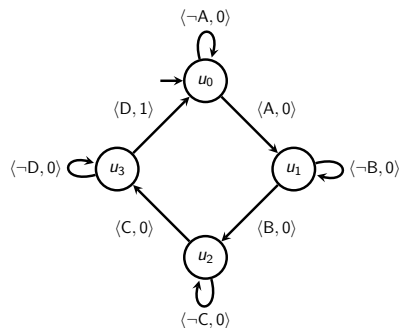
Hierarchical RL might converge to suboptimal policies



Reason: Policy π_c goes to the closest c .

Q-learning for Reward Machines (QRM)

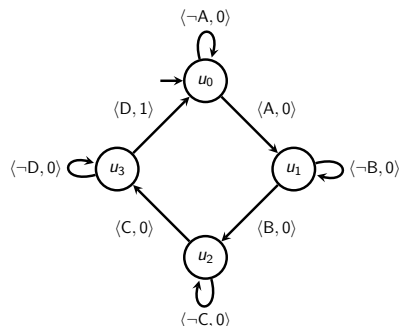
QRM (our approach)



Q-learning for Reward Machines (QRM)

QRM (our approach)

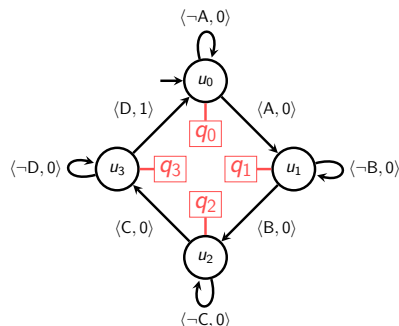
1. Learn one policy (q-function) per state in the reward machine.



Q-learning for Reward Machines (QRM)

QRM (our approach)

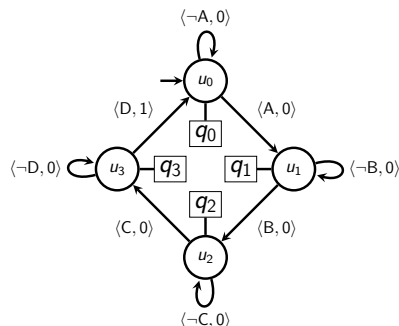
1. Learn one policy (q-function) per state in the reward machine.



Q-learning for Reward Machines (QRM)

QRM (our approach)

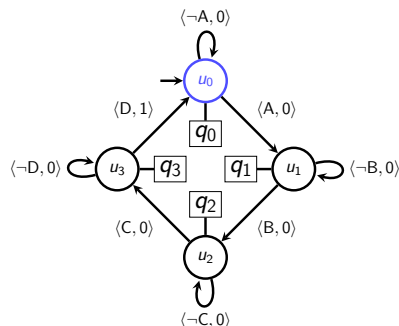
1. Learn one policy (q-function) per state in the reward machine.
2. Select actions using the policy of the current RM state.



Q-learning for Reward Machines (QRM)

QRM (our approach)

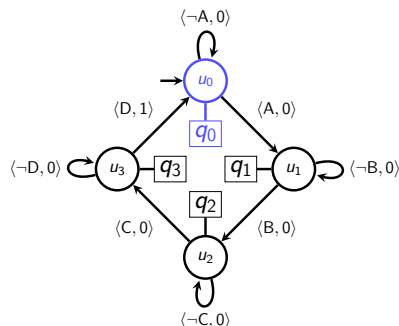
1. Learn one policy (q-function) per state in the reward machine.
2. Select actions using the policy of the current RM state.



Q-learning for Reward Machines (QRM)

QRM (our approach)

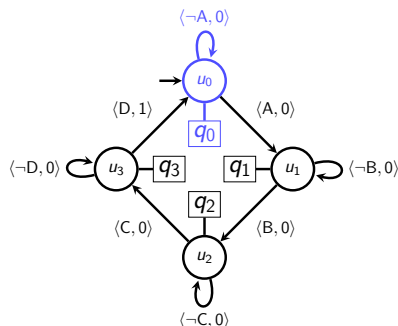
1. Learn one policy (q-function) per state in the reward machine.
2. Select actions using the policy of the current RM state.



Q-learning for Reward Machines (QRM)

QRM (our approach)

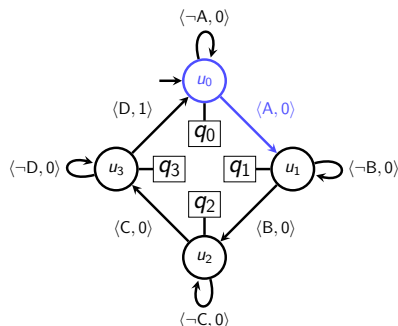
1. Learn one policy (q-function) per state in the reward machine.
2. Select actions using the policy of the current RM state.



Q-learning for Reward Machines (QRM)

QRM (our approach)

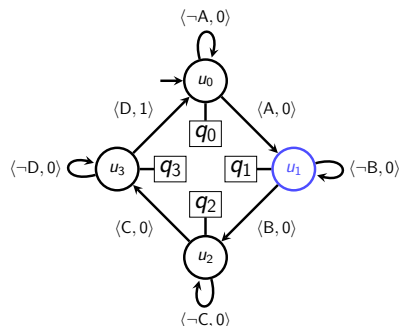
1. Learn one policy (q-function) per state in the reward machine.
2. Select actions using the policy of the current RM state.



Q-learning for Reward Machines (QRM)

QRM (our approach)

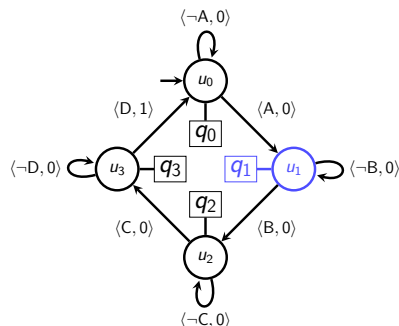
1. Learn one policy (q-function) per state in the reward machine.
2. Select actions using the policy of the current RM state.



Q-learning for Reward Machines (QRM)

QRM (our approach)

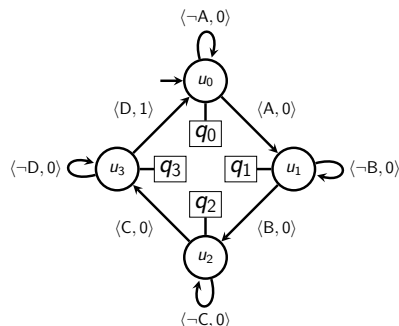
1. Learn one policy (q-function) per state in the reward machine.
2. Select actions using the policy of the current RM state.



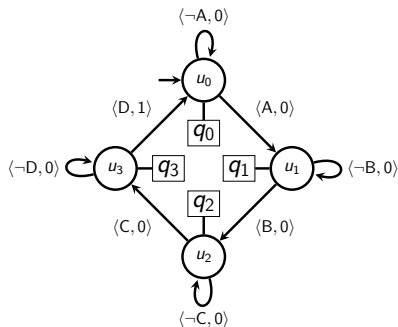
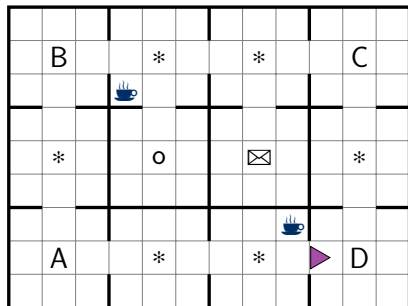
Q-learning for Reward Machines (QRM)

QRM (our approach)

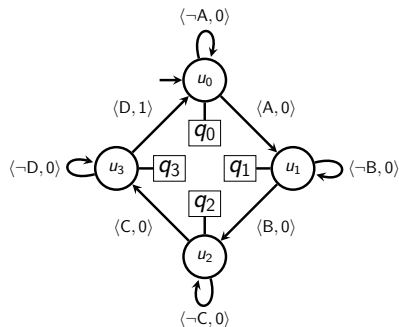
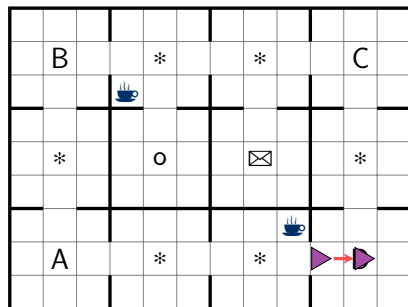
1. Learn one policy (q-function) per state in the reward machine.
2. Select actions using the policy of the current RM state.
3. Reuse experience to update all the q-values at the same time.



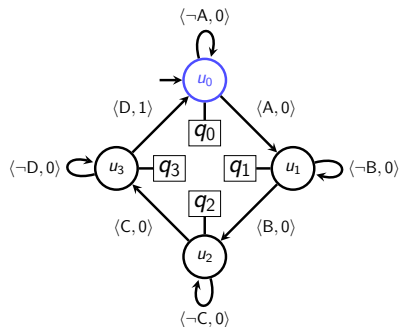
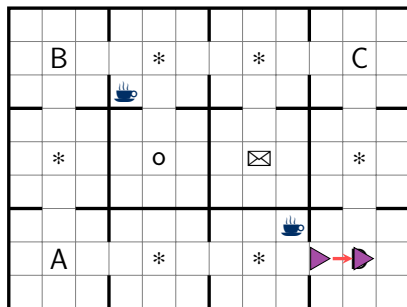
QRM learning step



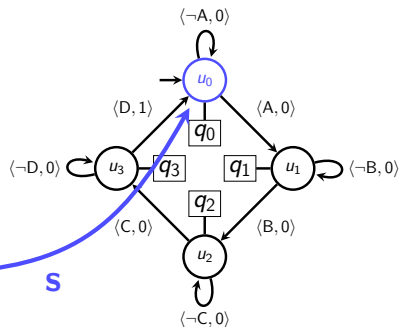
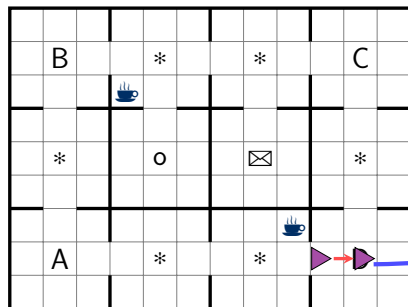
QRM learning step



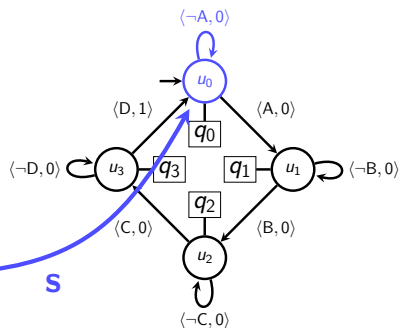
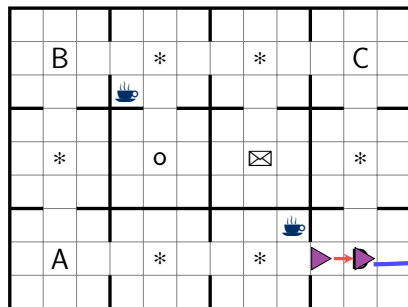
QRM learning step



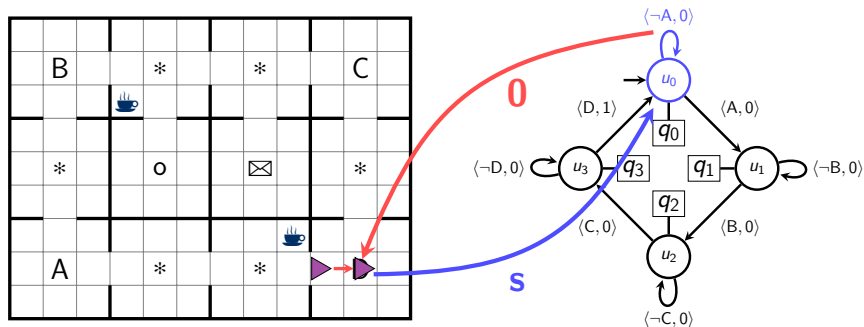
QRM learning step



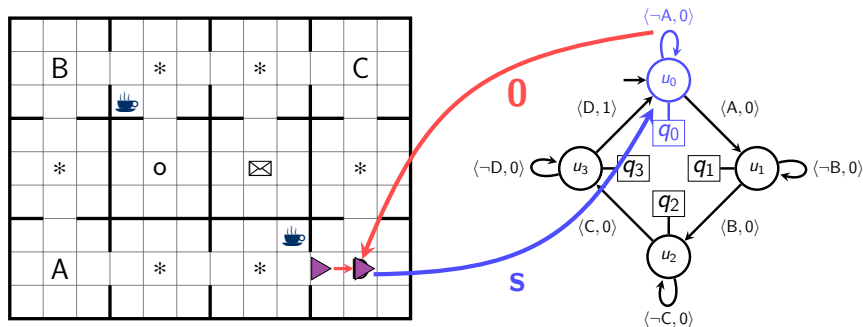
QRM learning step



QRM learning step

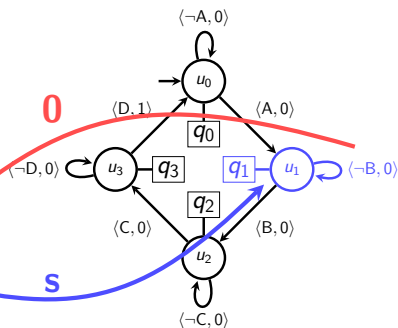
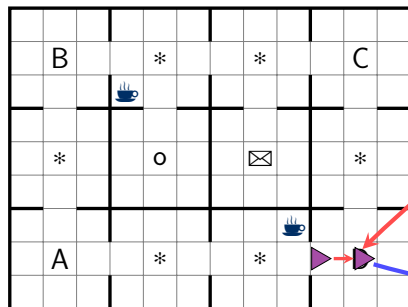


QRM learning step



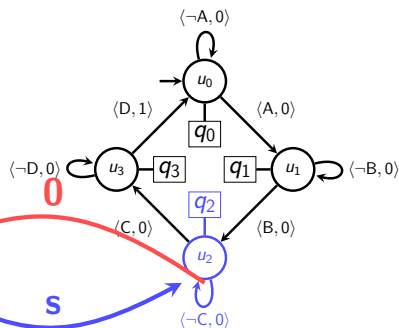
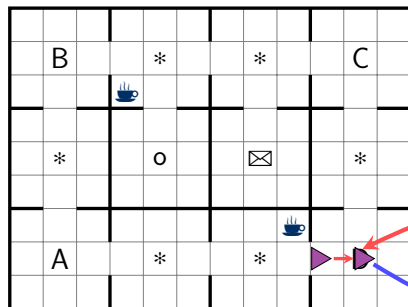
$$q_0(s, a) \leftarrow \alpha \cdot 0 + \gamma \max_{a'} q_0(s', a')$$

QRM learning step



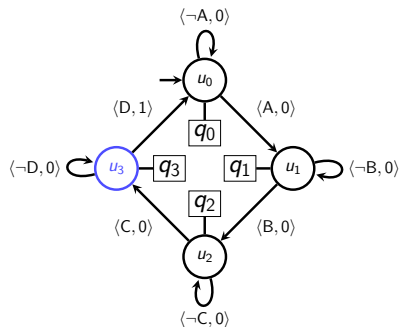
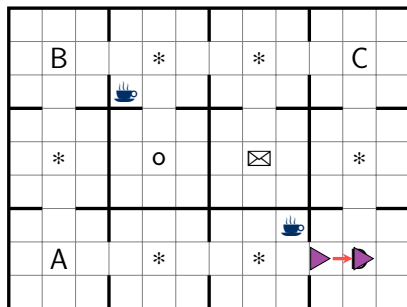
$$q_1(s, a) \leftarrow^{\alpha} 0 + \gamma \max_{a'} q_1(s', a')$$

QRM learning step

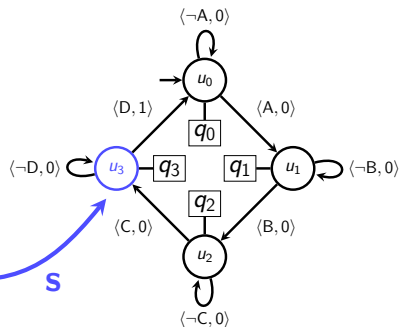
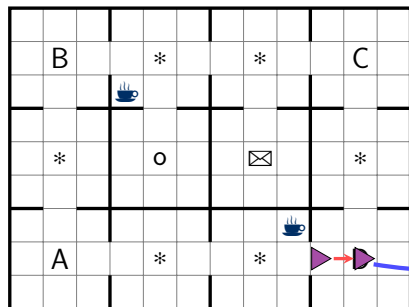


$$q_2(s, a) \leftarrow \alpha \cdot 0 + \gamma \max_{a'} q_2(s', a')$$

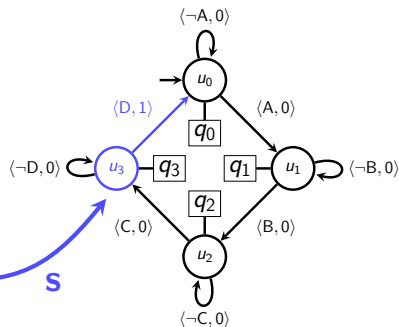
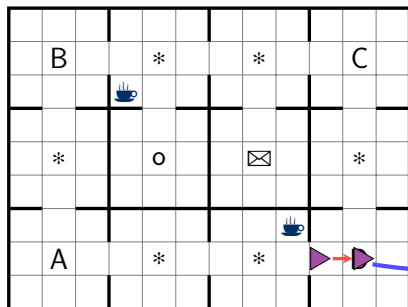
QRM learning step



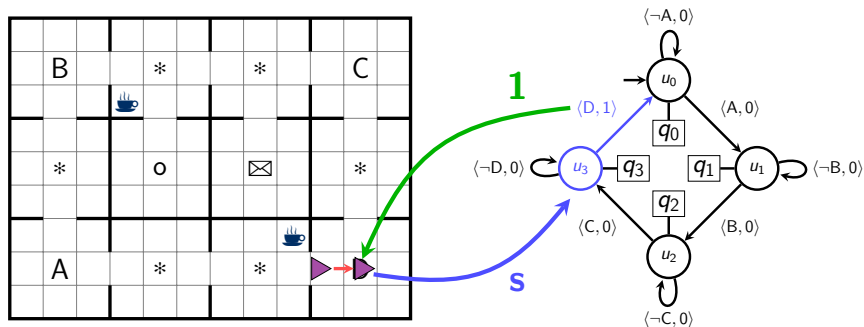
QRM learning step



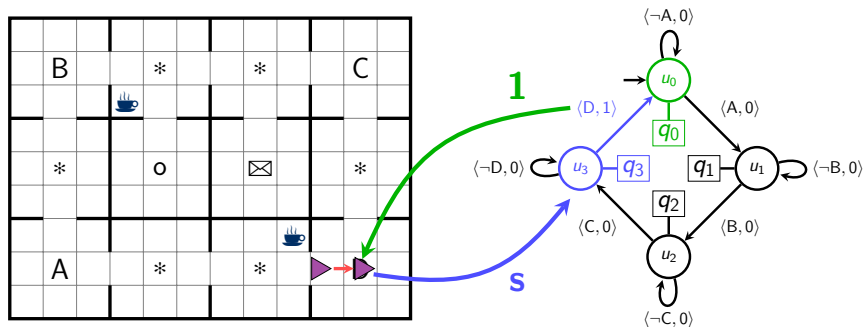
QRM learning step



QRM learning step

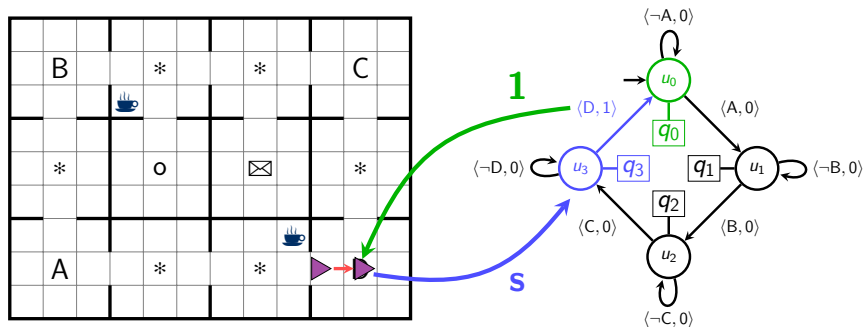


QRM learning step



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

QRM learning step



Theorem

QRM converges to an optimal policy in the limit.

- 1 Motivation
- 2 What is a reward machine (RM)?
- 3 How to exploit a reward machine's structure
- 4 Results
- 5 Related work
- 6 Concluding remarks

- 1 Motivation
- 2 What is a reward machine (RM)?
- 3 How to exploit a reward machine's structure
- 4 **Results**
- 5 Related work
- 6 Concluding remarks

Two discrete grid domains:

- Office domain (4 tasks).
- Craft domain (10 tasks).

Two discrete grid domains:

- Office domain (4 tasks).
- Craft domain (10 tasks).

One continuous state space domain:

- Water domain (10 tasks).

Algorithms

Q-Learning over a cross-product MDP (**Q-learning**)

Hierarchical RL based on options (**HRL**)

Hierarchical RL with option pruning (**HRL-RM**)

Q-learning for reward machines (**QRM**)

Algorithms

Q-Learning over a cross-product MDP (**Q-learning**)

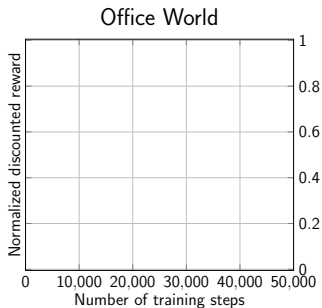
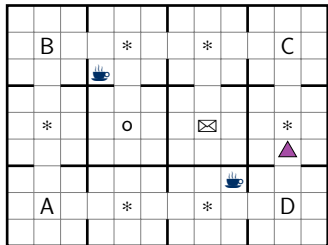
Hierarchical RL based on options (**HRL**)

Hierarchical RL with option pruning (**HRL-RM**)

Q-learning for reward machines (**QRM**)

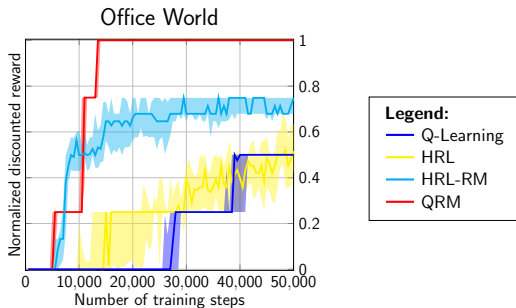
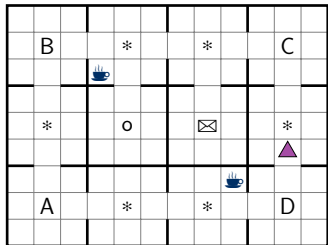
Method	Optimality?	Decomposition?
Q-learning	✓	
HRL		✓
HRL-RM		✓
QRM	✓	✓

The office domain



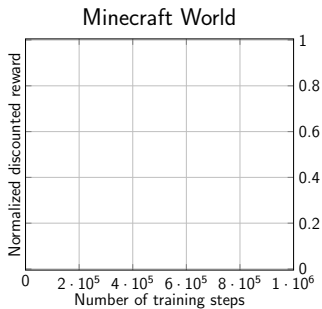
4 tasks (30 independent trials)

The office domain



4 tasks (30 independent trials)

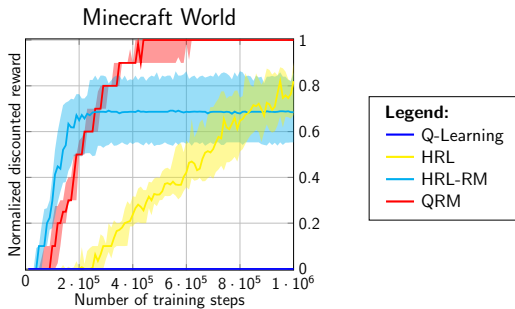
The craft domain



10 tasks defined by Andreas et al.¹ over 10 random maps (3 trials)

¹Modular Multitask Reinforcement Learning with Policy Sketches by Andreas et al. (ICML-17)

The craft domain



10 tasks defined by Andreas et al.¹ over 10 random maps (3 trials)

¹Modular Multitask Reinforcement Learning with Policy Sketches by Andreas et al. (ICML-17)

From tabular QRM to Deep QRM

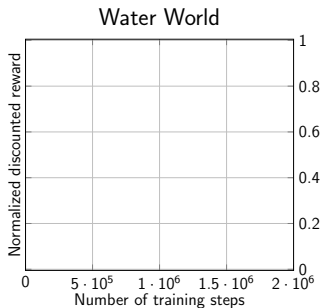
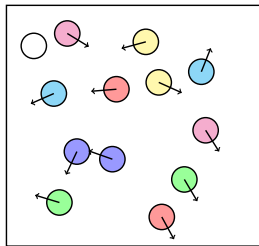
We replaced q-learning by Double DQN with prioritized experience replay in our four approaches.

From tabular QRM to Deep QRM

We replaced q-learning by Double DQN with prioritized experience replay in our four approaches.

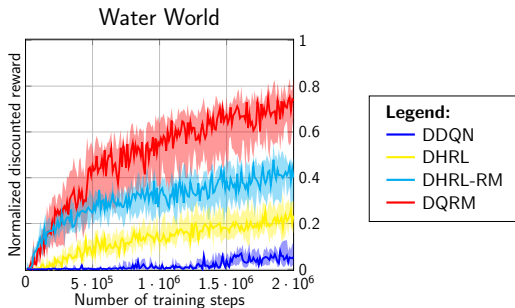
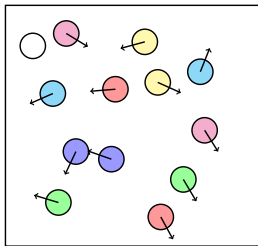
Method	Optimality?	Decomposition?
DDQN		
DHRL		✓
DHRL-RM		✓
DQRM		✓

The water domain



10 tasks over 10 random maps (3 trials per map)

The water domain



10 tasks over 10 random maps (3 trials per map)

- 1 Motivation
- 2 What is a reward machine (RM)?
- 3 How to exploit a reward machine's structure
- 4 Results
- 5 Related work
- 6 Concluding remarks

- 1 Motivation
- 2 What is a reward machine (RM)?
- 3 How to exploit a reward machine's structure
- 4 Results
- 5 **Related work**
- 6 Concluding remarks

Hierarchical RL (task decomposition):

Hierarchical RL (task decomposition):

- Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3-4), 323-339.
- Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2), 181-211.
- Parr, R., & Russell, S. J. (1998). Reinforcement learning with hierarchies of machines. In *NIPS* (pp. 1043-1049).
- Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.*, 13, 227-303.

Hierarchical RL (task decomposition):

- Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3-4), 323-339.
- Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2), 181-211.
- Parr, R., & Russell, S. J. (1998). Reinforcement learning with hierarchies of machines. In *NIPS* (pp. 1043-1049).
- Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.*, 13, 227-303.

HRL might converge to suboptimal policies.

Hierarchical RL (task decomposition):

- Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3-4), 323-339.
- Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2), 181-211.
- Parr, R., & Russell, S. J. (1998). Reinforcement learning with hierarchies of machines. In *NIPS* (pp. 1043-1049).
- Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.*, 13, 227-303.

HRL might converge to suboptimal policies.

Lot of relations between QRM and HRL (more in the paper!)

Linear Temporal Logic (task specification):

Linear Temporal Logic (task specification):

- Li, X., Vasile, C. I., & Belta, C. (2017). Reinforcement learning with temporal logic rewards. In *IROS* (pp. 3834-3839).
- Littman, M. L., Topcu, U., Fu, J., Isbell, C., Wen, M., & MacGlashan, J. (2017). Environment-independent task specifications via GLTL. *arXiv preprint arXiv:1704.04341*.
- Toro Icarte, R., Klassen, T., Valenzano, R., & McIlraith, S. (2018). Teaching multiple tasks to an RL agent using LTL. In *AAMAS* (pp. 452-461).
- Hasanbeig, M., Abate, A., & Kroening, D. (2018). Logically-Correct Reinforcement Learning. *arXiv preprint arXiv:1801.08099*.

Linear Temporal Logic (task specification):

- Li, X., Vasile, C. I., & Belta, C. (2017). Reinforcement learning with temporal logic rewards. In *IROS* (pp. 3834-3839).
- Littman, M. L., Topcu, U., Fu, J., Isbell, C., Wen, M., & MacGlashan, J. (2017). Environment-independent task specifications via GLTL. *arXiv preprint arXiv:1704.04341*.
- Toro Icarte, R., Klassen, T., Valenzano, R., & McIlraith, S. (2018). Teaching multiple tasks to an RL agent using LTL. In *AAMAS* (pp. 452-461).
- Hasanbeig, M., Abate, A., & Kroening, D. (2018). Logically-Correct Reinforcement Learning. *arXiv preprint arXiv:1801.08099*.

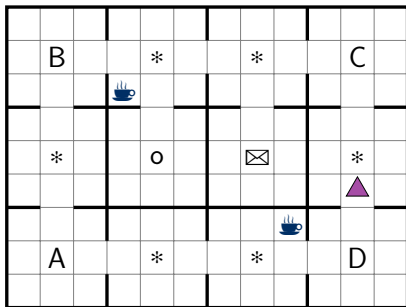
RMs can express reward functions that cannot be expressed in LTL.

- 1 Motivation
- 2 What is a reward machine (RM)?
- 3 How to exploit a reward machine's structure
- 4 Results
- 5 Related work
- 6 Concluding remarks

- 1 Motivation
- 2 What is a reward machine (RM)?
- 3 How to exploit a reward machine's structure
- 4 Results
- 5 Related work
- 6 **Concluding remarks**

Concluding remarks

We proposed to show the reward function's code to the agent

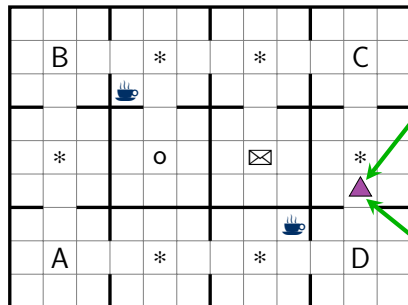


```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

Reward Function

Concluding remarks

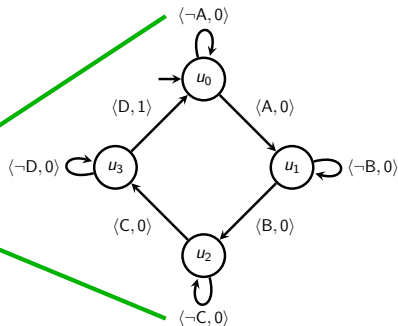
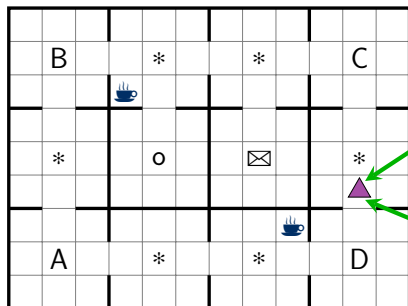
... so it can exploit the reward's structure.



```
1 m = 0 # global variable
2 def get_reward(s):
3     if m == 0 and s.at("A"):
4         m = 1
5     if m == 1 and s.at("B"):
6         m = 2
7     if m == 2 and s.at("C"):
8         m = 3
9     if m == 3 and s.at("D"):
10        m = 0
11        return 1
12    return 0
```

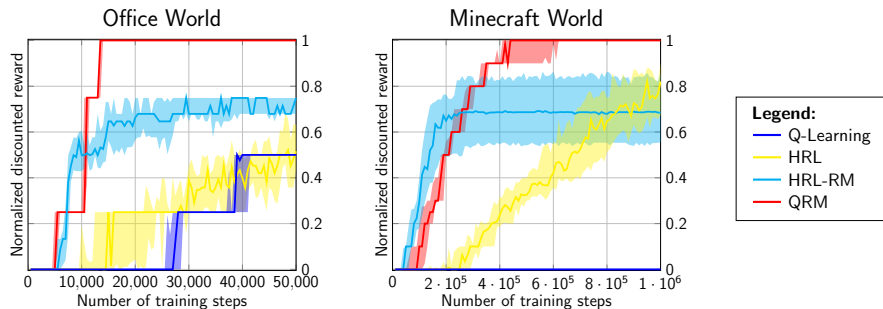
Concluding remarks

To define reward functions, we used reward machines



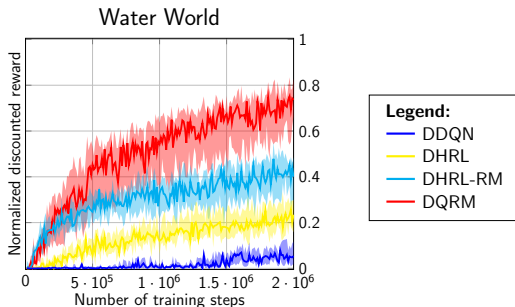
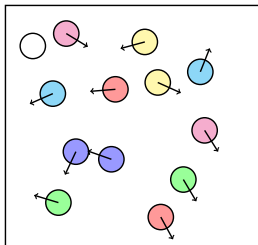
Concluding remarks

QRM outperformed plain RL and HRL in 2 discrete domains.



Concluding remarks

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



Thanks!

Title: Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning

Code: <https://bitbucket.org/RToroIcarte/qrm>

Poster: #147