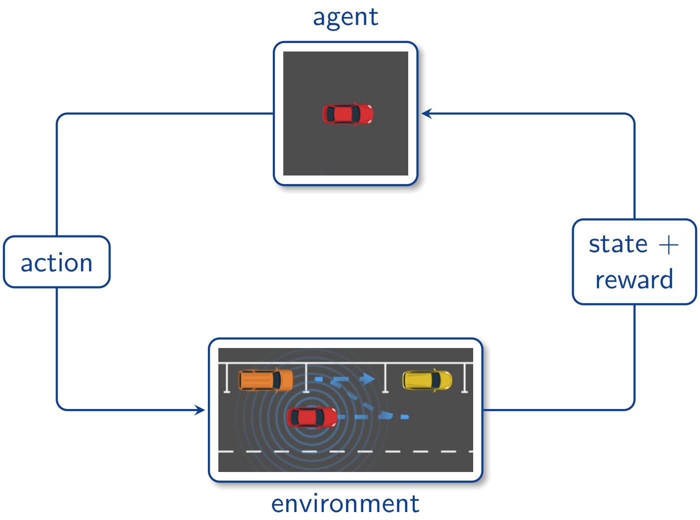
Reinforcement Learning with Reward Machines

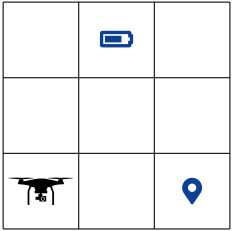
Daniel Neider

tu dortmund university

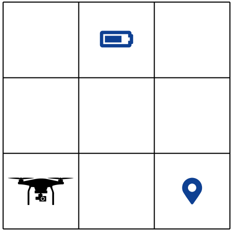
CENTER FOR TRUSTWORTHY DATA SCIENCE AND SECURITY
UA RUHR | RESEARCH ALLIANCE

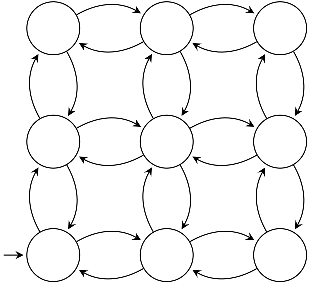
Theorietag “Automaten und Formale Sprachen”
RPTU/MPI-SWS, Kaiserslautern, Germany
4 October 2023


agent
action
state + reward
environment



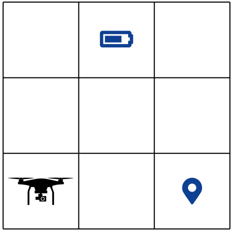
- Actions $A = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
- Labels $\mathcal{P} = \{$📍, 🔋$\}$

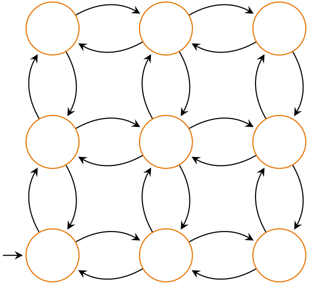




- Actions $A = \{$↑, ↓, ←, →$\}$
- Labels $\mathcal{P} = \{$📍, 🔋$\}$

$$\mathcal{M} = (S, s_I, A, \mathcal{P}, p, L, R, \gamma)$$

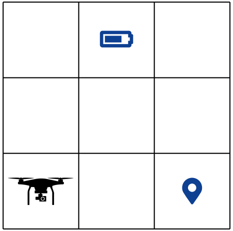


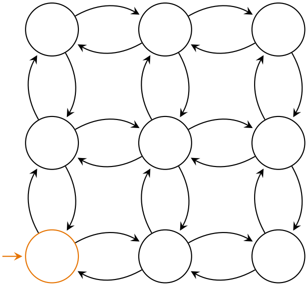


- Actions $A = \{$↑, ↓, ←, →$\}$
- Labels $\mathcal{P} = \{$📍, 🔋$\}$

$$\mathcal{M} = (S, s_I, A, \mathcal{P}, p, L, R, \gamma)$$

set S of states

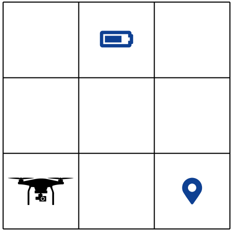


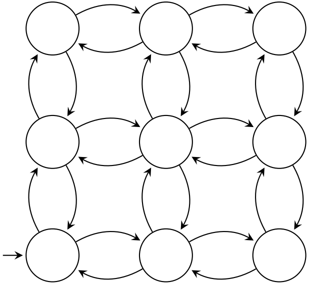


- Actions $A = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
- Labels $\mathcal{P} = \{$📍, 🔋$\}$

$$\mathcal{M} = (S, s_I, A, \mathcal{P}, p, L, R, \gamma)$$

initial state $s_I \in S$



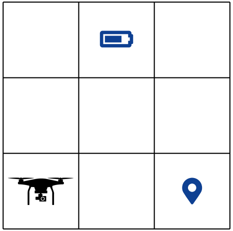


- Actions $A = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
- Labels $\mathcal{P} = \{$📍, 🔋$\}$

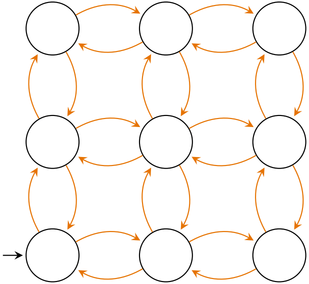
$$\mathcal{M} = (S, s_I, A, \mathcal{P}, p, L, R, \gamma)$$

actions A and labels $\mathcal{P}$

Labeled Markov Decision Processes



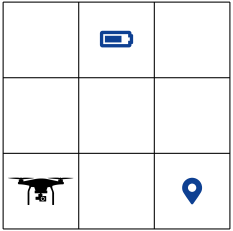
- Actions $A = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
- Labels $\mathcal{P} = \{$📍, 🔋$\}$

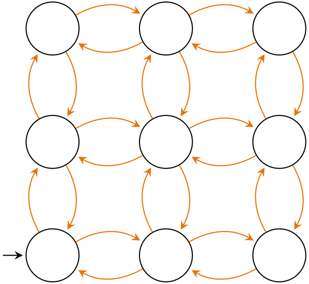


$$\mathcal{M} = (S, s_I, A, \mathcal{P}, p, L, R, \gamma)$$

transition function
$p\colon S \times A \times S \to [0, 1]$

Labeled Markov Decision Processes

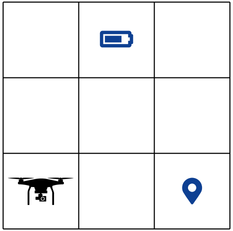


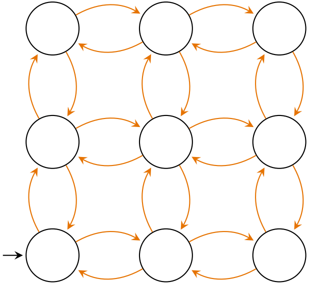


- Actions $A = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
- Labels $\mathcal{P} = \{$📍, 🔋$\}$

$$\mathcal{M} = (S, s_I, A, \mathcal{P}, p, L, R, \gamma)$$

labeling function
$L \colon S \times A \times S \to \mathcal{P}$

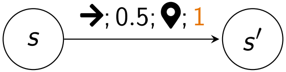




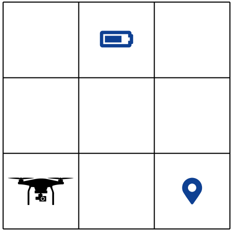
- Actions $A = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
- Labels $\mathcal{P} = \{$📍, 🔋$\}$

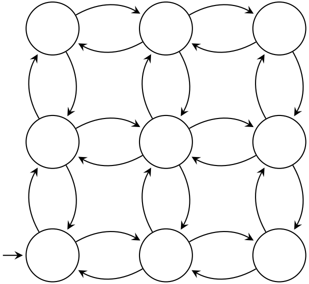
$$\mathcal{M} = (S, s_I, A, \mathcal{P}, p, L, R, \gamma)$$

reward function
$R\colon S \times A \times S \to \mathbb{R}$



Labeled Markov Decision Processes





- Actions $A = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
- Labels $\mathcal{P} = \{$📍, 🔋$\}$

$$\mathcal{M} = (S, s_I, A, \mathcal{P}, p, L, R, \gamma)$$

discount factor $\gamma \in (0, 1)$

Find a (probabilistic) policy $\pi \colon S \times A \to [0, 1]$ maximizing the expected discounted reward

$$\mathbb{E}_\pi \left[\sum_{i=0}^{k} \gamma^i \cdot R(s_i, a_{i+1}, s_{i+1}) \right]$$

of every trajectory $s_0 a_0 s_1 \ldots s_{k+1}$, $k \in \mathbb{N}$, through the MDP

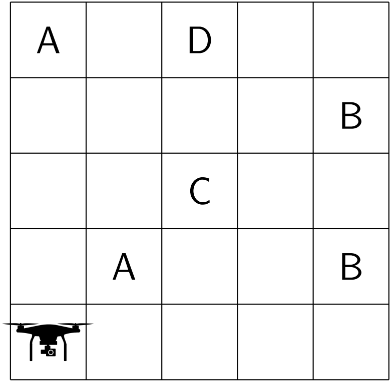
Find a (probabilistic) policy $\pi\colon S \times A \to [0,1]$ maximizing the expected discounted reward

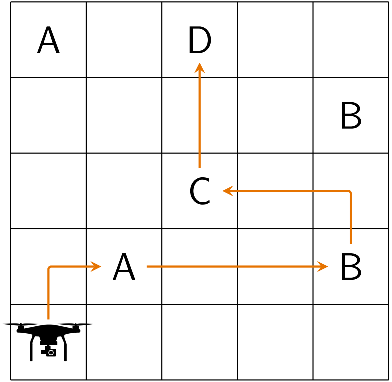
$$\mathbb{E}_\pi\left[\sum_{i=0}^{k} \gamma^i \cdot R(s_i, a_{i+1}, s_{i+1})\right]$$

of every trajectory $s_0 a_0 s_1 \dots s_{k+1}$, $k \in \mathbb{N}$, through the MDP

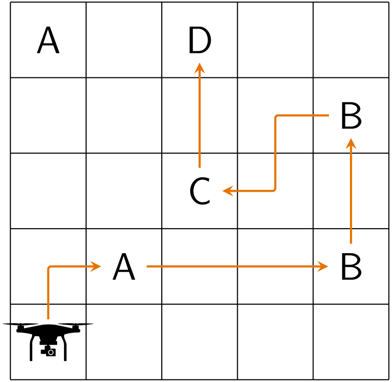
Q-Learning

1. Maintain a table $Q\colon S \times A \to \mathbb{R}$ (initialized to, e.g., 0)
2. Explore the environment according to π, resulting in a trajectory $s_0 a_1 s_1 a_2 s_2 \dots$
3. In step t, update Q by
   $Q(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha\left[R(s_t, a_{t+1}, s_{t+1}) + \gamma \max_a Q(s_{t+1}, a)\right]$
4. After each episode, update π by $\pi(s, a) \leftarrow \arg\max_{a \in A} Q(s, a)$
5. Repeat until this process converges; π is then the optimal policy


A
D
B
C
A
B

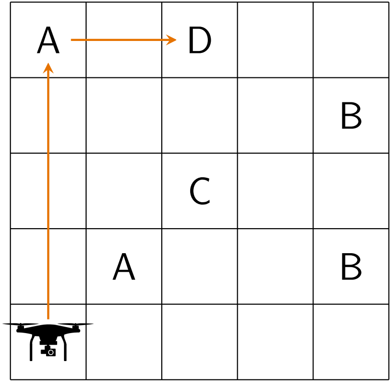


A; B; C; D: 1



A; B; C; D: 1

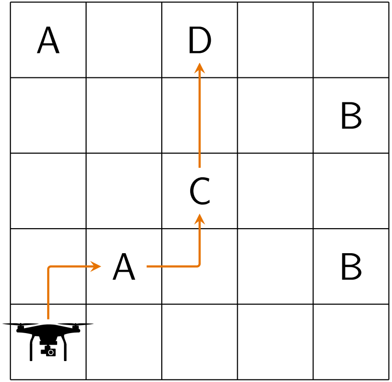
A; B; B; C; D: 1



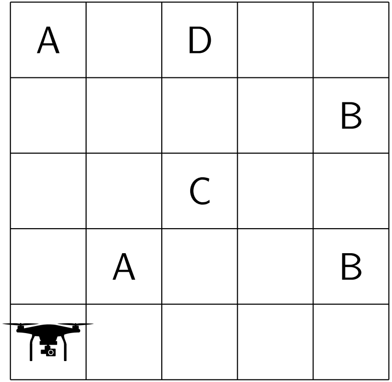
A; B; C; D: 1

A; B; B; C; D: 1

A; D: -1

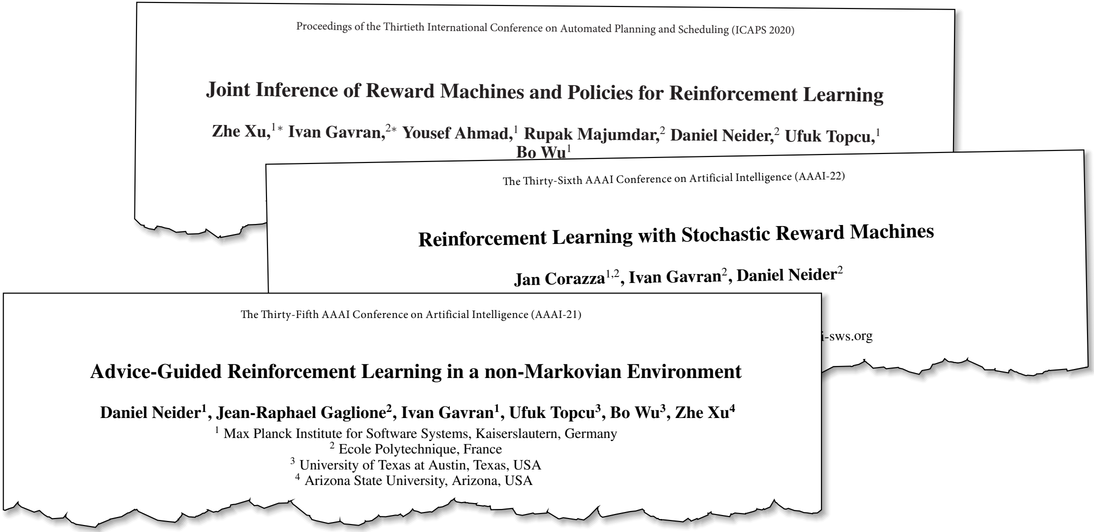


A; B; C; D:	1
A; B; B; C; D:	1
A; D:	-1
A; C; D:	-1



A; B; C; D:	1
A; B; B; C; D:	1
A; D:	-1
A; C; D:	-1

How to handle such situations?



Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)

Joint Inference of Reward Machines and Policies for Reinforcement Learning

Zhe Xu,[1*] Ivan Gavran,[2*] Yousef Ahmad,[1] Rupak Majumdar,[2] Daniel Neider,[2] Ufuk Topcu,[1] Bo Wu[1]

The Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-22)

Reinforcement Learning with Stochastic Reward Machines

Jan Corazza[1,2], Ivan Gavran[2], Daniel Neider[2]

i-sws.org

The Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21)

Advice-Guided Reinforcement Learning in a non-Markovian Environment

Daniel Neider[1], Jean-Raphael Gaglione[2], Ivan Gavran[1], Ufuk Topcu[3], Bo Wu[3], Zhe Xu[4]

[1] Max Planck Institute for Software Systems, Kaiserslautern, Germany
[2] Ecole Polytechnique, France
[3] University of Texas at Austin, Texas, USA
[4] Arizona State University, Arizona, USA

1. Joint Inference of Policies and Reward Machines

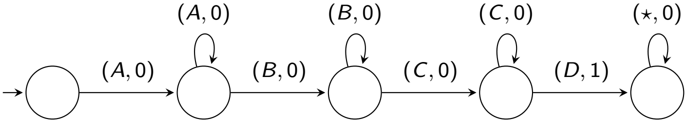
(joint work with Yousef Ahmad, Ivan Gavran, Rupak Majumdar, Ufuk Topcu, Bo Wu, and Zhe Xu)

A; B; C; D:	1
A; B; B; C; D:	1
A; D:	-1
A; C; D:	-1

Bacchus et al. (1996)
Jothimurgan et al. (2019)
Icarte et al. (2018)
Brafman et al. (2018)

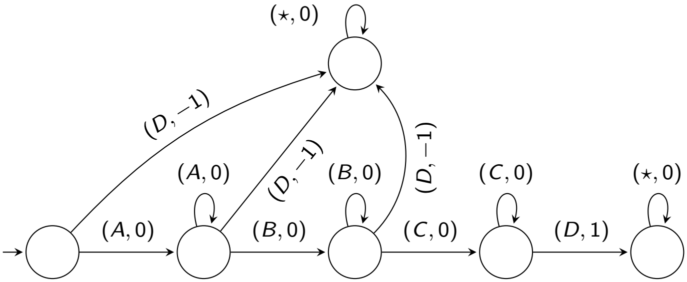
"Use automata/temporal logic to capture non-Markovian rewards"

A; B; C; D:	1
A; B; B; C; D:	1
A; D:	-1
A; C; D:	-1



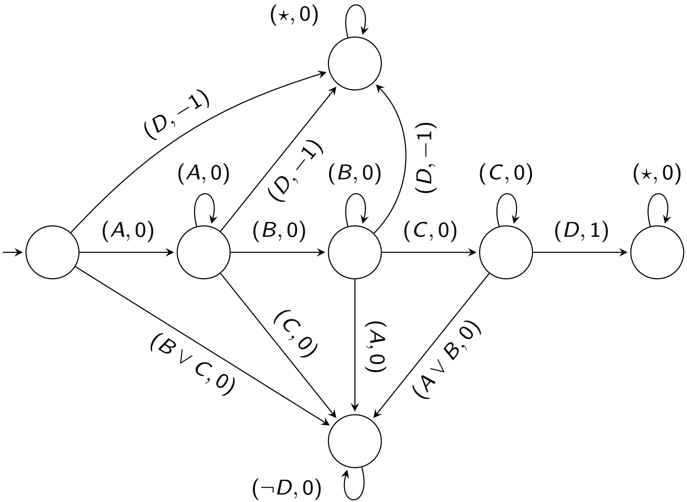
Bacchus et al. (1996)
Jothimurgan et al. (2019)
Icarte et al. (2018)
Brafman et al. (2018)

A; B; C; D:	1
A; B; B; C; D:	1
A; D:	-1
A; C; D:	-1



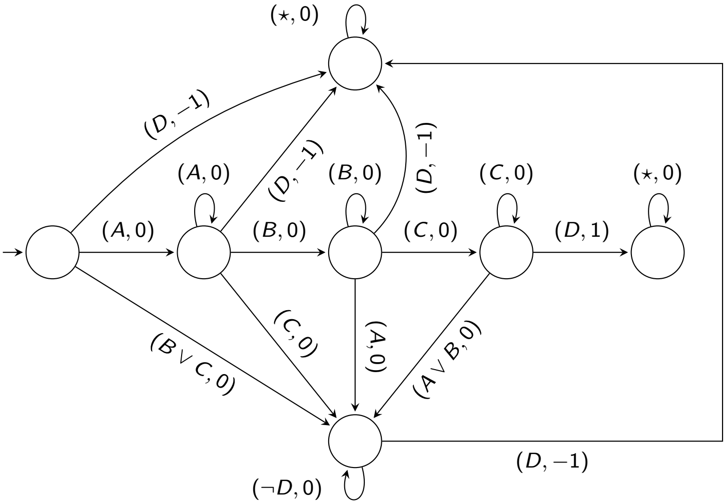
Bacchus et al. (1996)
Jothimurgan et al. (2019)
Icarte et al. (2018)
Brafman et al. (2018)

A; B; C; D:	1
A; B; B; C; D:	1
A; D:	-1
A; C; D:	-1



Bacchus et al. (1996)
Jothimurgan et al. (2019)
Icarte et al. (2018)
Brafman et al. (2018)

A; B; C; D:	1
A; B; B; C; D:	1
A; D:	-1
A; C; D:	-1



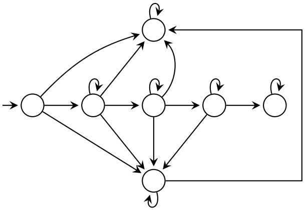
Bacchus et al. (1996)
Jothimurgan et al. (2019)
Icarte et al. (2018)
Brafman et al. (2018)

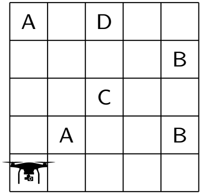
A; B; C; D:	1

A; B; B; C; D:	1

A; D:	-1

A; C; D:	-1





Bacchus et al. (1996)
Jothimurgan et al. (2019)
Icarte et al. (2018)
Brafman et al. (2018)

Icarte et al. (2018) have proposed an extension of the Q-learning algorithm, named QRM, that can handle reward machines

- avoids building the cross-product explicitly
- exploits the structure of the reward machine during exploration

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

Rodrigo Toro Icarte [1 2] Toryn Q. Klassen [1] Richard Valenzano [3] Sheila A. McIlraith [1 2]

Abstract

In this paper we propose Reward Machines – a type of finite state machine that supports the specification of reward functions while exposing reward function structure to the learner and supporting decomposition. We then present Q-Learning for Reward Machines (QRM), an algorithm which appropriately decomposes the reward machine and uses off-policy q-learning to simultaneously learn subpolicies for the different components. QRM is guaranteed to converge to an optimal policy in the tabular case, in contrast to Hierarchical Reinforcement Learning methods which might converge to suboptimal policies. We demonstrate this behavior experimentally in two discrete domains. We also show how function approximation methods like neural networks can be incorporated into QRM, and that doing so can find better policies more quickly than hierarchical methods in a domain with a continuous state space.

1. Introduction

A standard assumption in reinforcement learning (RL) is that the agent does not have access to the environment model (Sutton & Barto, 1998). This means that it does not know, a priori, the transition probabilities or reward function manifest in the environment. To learn optimal behavior, an RL agent must therefore interact with the environment and learn from its experience. While assuming that the transition probabilities are unknown seems reasonable, there is less reason to hide the reward function from the agent. Artificial agents cannot inherently perceive reward from the environment; someone must program those rewards functions (even if the agent is interacting with the real world). Typically, though, a programmed reward function is given as a black box to the agent. The agent can query the function for the reward in the current situation, but does not have access to whatever structures or high-level ideas the programmer may have used in defining it. However, an agent that had access to the specification of the reward function might be able to use it to decompose the problem and speed up learning. We consider a way to do so in this paper.

Previous work on giving an agent knowledge about the reward function focus on defining a task specification language, usually based on sub-goal sequences (Singh, 1992a;b) or linear temporal logic (Li et al., 2017; Camacho et al., 2017; Littman et al., 2017; Toro Icarte et al., 2018; Hasanbeig et al., 2018), and then generate a reward function towards fulfilling that specification. In this work, we instead directly tackle the problem of defining reward functions that expose structure to the agent. As such, our approach is able to reward behaviors to varying degrees in manners that cannot be expressed by previous approaches.

There are two main contributions of this work. First, we introduce a type of finite state machine, called the *Reward Machine*, which we use in defining rewards. A reward machine allows for composing different reward functions in flexible ways, including concatenations, loops, and conditional rules. As an agent acts in the environment, moving from state to state, it also moves from state to state within a reward machine (as determined by high-level events detected within the environment). After every transition, the reward machine outputs the reward function the agent should use at that time. For example, we might construct a reward machine for "*delivering coffee to an office*" using two states. In the first state, the agent does not receive any rewards, but it moves to the second state whenever it gets the coffee. In the second state, the agent gets rewards after delivering the coffee. The advantage of defining rewards this way is that the agent knows that the problem consists of two stages and might use this information for decomposing it.

Our second contribution is to introduce an algorithm, called *Q-Learning for Reward Machines (QRM)*, that can exploit a reward machine's internal structure to decompose the problem and thereby improve sample efficiency. QRM's task decomposition does not prune optimal policies and uses *q-*

[1]Department of Computer Science, University of Toronto, Toronto, Ontario, Canada [2]Vector Institute, Toronto, Ontario, Canada [3]Element AI, Toronto, Ontario, Canada. Correspondence to: Rodrigo Toro Icarte <rntoro@cs.toronto.edu>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

ICML 2018

Icarte et al. (2018) have proposed an extension of the Q-learning algorithm, named QRM, that can handle reward machines

- avoids building the cross-product explicitly
- exploits the structure of the reward machine during exploration

Problem

How does one construct reward machines?

- direct construction, from temporal logics, learning, ...

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

Rodrigo Toro Icarte[1,2] Toryn Q. Klassen[1] Richard Valenzano[3] Sheila A. McIlraith[1,2]

Abstract

In this paper we propose Reward Machines – a type of finite state machine that supports the specification of reward functions while exposing reward function structure to the learner and supporting decomposition. We then present Q-Learning for Reward Machines (QRM), an algorithm which appropriately decomposes the reward machine and uses off-policy q-learning to simultaneously learn subpolicies for the different components. QRM is guaranteed to converge to an optimal policy in the tabular case, in contrast to Hierarchical Reinforcement Learning methods which might converge to suboptimal policies. We demonstrate this behavior experimentally in two discrete domains. We also show how function approximation methods like neural networks can be incorporated into QRM, and that doing so can find better policies more quickly than hierarchical methods in a domain with a continuous state space.

1. Introduction

A standard assumption in reinforcement learning (RL) is that the agent does not have access to the environment model (Sutton & Barto, 1998). This means that it does not know, a priori, the transition probabilities or reward function manifest in the environment. To learn optimal behavior, an RL agent must therefore interact with the environment and learn from its experience. While assuming that the transition probabilities are unknown seems reasonable, there is less reason to hide the reward function from the agent. Artificial agents cannot inherently perceive reward from the environment; someone must program those rewards functions (even if the agent is interacting with the real world). Typically, though, a programmed reward function is given as a black box to the agent. The agent can query the function for the reward in the current situation, but does not have access to whatever structures or high-level ideas the programmer may have used in defining it. However, an agent that had access to the specification of the reward function might be able to use it to decompose the problem and speed up learning. We consider a way to do so in this paper.

Previous work on giving an agent knowledge about the reward function focus on defining a task specification language, usually based on sub-goal sequences (Singh, 1992a;b) or linear temporal logic (Li et al., 2017; Camacho et al., 2017; Littman et al., 2017; Toro Icarte et al., 2018; Hasanbeig et al., 2018), and then generate a reward function towards fulfilling that specification. In this work, we instead directly tackle the problem of defining reward functions that expose structure to the agent. As such, our approach is able to reward behaviors to varying degrees in manners that cannot be expressed by previous approaches.

There are two main contributions of this work. First, we introduce a type of finite state machine, called the *Reward Machine*, which we use in defining rewards. A reward machine allows for composing different reward functions in flexible ways, including concatenations, loops, and conditional rules. As an agent acts in the environment, moving from state to state, it also moves from state to state within a reward machine (as determined by high-level events detected within the environment). After every transition, the reward machine outputs the reward function the agent should use at that time. For example, we might construct a reward machine for "*delivering coffee to an office*" using two states. In the first state, the agent does not receive any rewards, but it moves to the second state whenever it gets the coffee. In the second state, the agent gets rewards after delivering the coffee. The advantage of defining rewards this way is that the agent knows that the problem consists of two stages and might use this information for decomposing it.

Our second contribution is to introduce an algorithm, called *Q-Learning for Reward Machines (QRM)*, that can exploit a reward machine's internal structure to decompose the problem and thereby improve sample efficiency. QRM's task decomposition does not prune optimal policies and uses *q*-

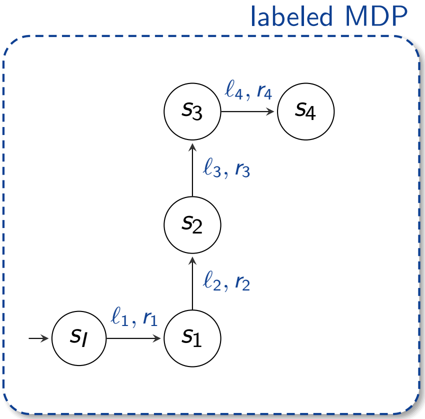
[1]Department of Computer Science, University of Toronto, Toronto, Ontario, Canada [2]Vector Institute, Toronto, Ontario, Canada [3]Element AI, Toronto, Ontario, Canada. Correspondence to: Rodrigo Toro Icarte <rntoro@cs.toronto.edu>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

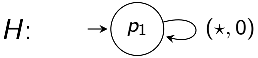
ICML 2018

Key idea

- Given the current hypothesis reward machine H, perform QRM and record the resulting label sequence $\lambda = \ell_1 \ldots \ell_n$ and reward sequence $\rho = r_1 \ldots r_n$
- If the pair (λ, ρ) contradicts H, learn a new reward machine H'
- Repeat until this process converges to the "true" reward machine and an optimal policy



Initialize reward machine H;

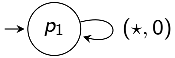


The JIRP Algorithm

Initialize reward machine H;

Initialize a set Q of q-functions;

H:



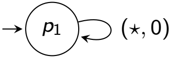
Q: $\{q^{p_1}\}$

Initialize reward machine H;

Initialize a set Q of q-functions;

Initialize a sample X of traces;

H:



Q: $\{q^{p_1}\}$

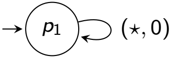
X: $\emptyset$

Initialize H, Q, X;

repeat

$(\lambda, \rho, Q) \leftarrow \mathrm{QRM}(H, Q)$;

H:



Q: $\{q^{p_1}\}$

X: $\emptyset$

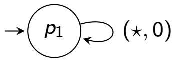
Initialize H, Q, X;

repeat

$A; B; C; D$

$(\lambda, \rho, Q) \leftarrow \mathrm{QRM}(H, Q)$;

H:



Q: $\{q^{p_1}\}$

X: $\emptyset$

Initialize H, Q, X;

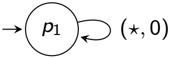
repeat

$(\lambda, \rho, Q) \leftarrow \text{QRM}(H, Q)$;

$A; B; C; D$

$0; 0; 0; 1$

H:



Q: $\{q^{p_1}\}$

X: $\emptyset$

Initialize H, Q, X;

repeat

$(\lambda, \rho, Q) \leftarrow \mathrm{QRM}(H, Q)$;

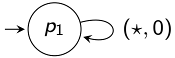
$A; B; C; D$

$0; 0; 0; 1$

if $H(\lambda) \neq \rho$ **then**

add (λ, ρ) to X;

H:



Q: $\{q^{p_1}\}$

X: $\emptyset$

Initialize H, Q, X;

repeat

$(\lambda, \rho, Q) \leftarrow \text{QRM}(H, Q)$;

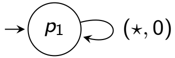
$A; B; C; D$

$0; 0; 0; 1$

if $H(\lambda) \neq \rho$ **then**

add (λ, ρ) to X;

H:



Q: $\{q^{p_1}\}$

X: $\{(A; B; C; D / 0; 0; 0; 1)\}$

Initialize H, Q, X;

repeat

$(\lambda, \rho, Q) \leftarrow \text{QRM}(H, Q)$;

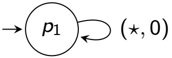
if $H(\lambda) \neq \rho$ **then**

add (λ, ρ) to X;

if X was modified **then**

$H \leftarrow \text{infer}(X)$;

H:



Q: $\{q^{p_1}\}$

X: $\{(A; B; C; D/0; 0; 0; 1)\}$

Initialize H, Q, X;

repeat

$(\lambda, \rho, Q) \leftarrow \mathrm{QRM}(H, Q)$;

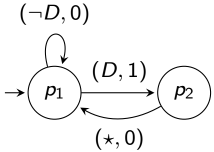
if $H(\lambda) \neq \rho$ **then**

add (λ, ρ) to X;

if X was modified **then**

$H \leftarrow \mathrm{infer}(X)$;

H:



Q: $\{q^{p_1}\}$

X: $\{(A; B; C; D / 0; 0; 0; 1)\}$

Initialize H, Q, X;

repeat

  $(\lambda, \rho, Q) \leftarrow \mathrm{QRM}(H, Q)$;

  if $H(\lambda) \neq \rho$ **then**

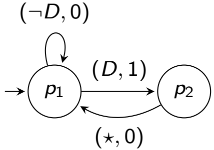
    add (λ, ρ) to X;

  if X was modified **then**

    $H \leftarrow \mathrm{infer}(X)$;

    re-initialize Q if necessary;

H:



Q: $\{q^{p_1}\}$

X: $\{(A; B; C; D / 0; 0; 0; 1)\}$

Initialize H, Q, X;

repeat

$(\lambda, \rho, Q) \leftarrow \mathrm{QRM}(H, Q)$;

if $H(\lambda) \neq \rho$ **then**

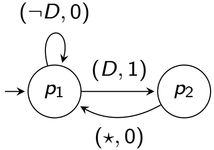
add (λ, ρ) to X;

if X was modified **then**

$H \leftarrow \mathrm{infer}(X)$;

re-initialize Q if necessary;

H:



Q: $\{q^{p_1}, q^{p_2}\}$

X: $\{(A; B; C; D / 0; 0; 0; 1)\}$

Initialize H, Q, X;

repeat

$(\lambda, \rho, Q) \leftarrow \text{QRM}(H, Q)$;

if $H(\lambda) \neq \rho$ **then**

add (λ, ρ) to X;

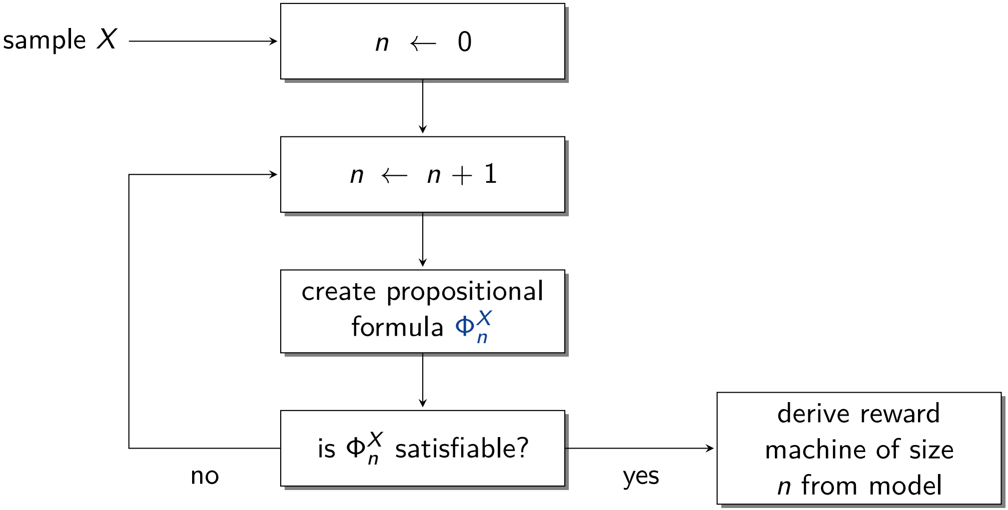
if X was modified **then**

$H \leftarrow \text{infer}(X)$;

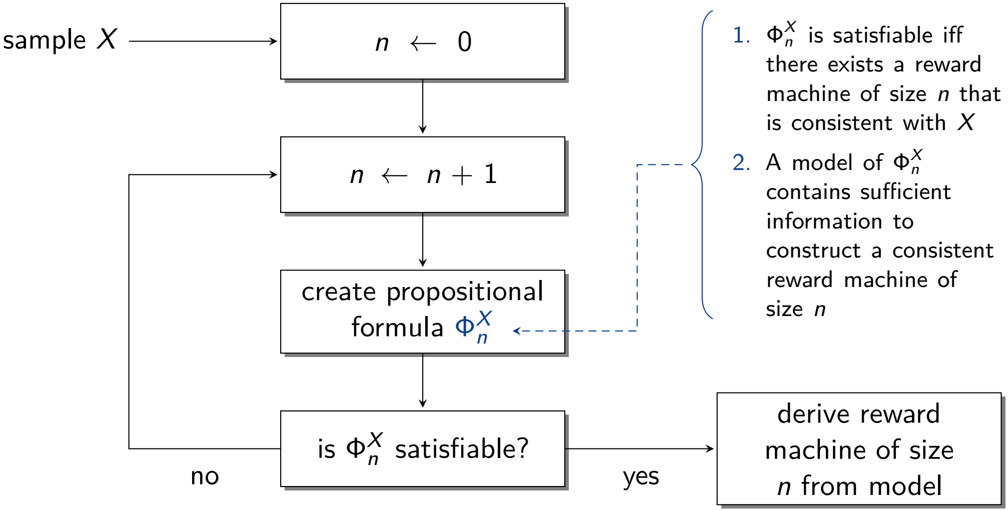
re-initialize Q if necessary;

It is crucial to infer **minimal** reward machines

A SAT-Based Inference Algorithm for Reward Machines



A SAT-Based Inference Algorithm for Reward Machines



Encoding Reward Machines in Propositional Logic

We use two sets of propositional variables to encode reward machines:

$d_{p,\ell,q}$ encodes the transition function of the reward machine
(i.e., the machine transitions from state p to state q on reading symbol ℓ)

$o_{p,\ell,r}$ encodes the output function of the reward machine
(i.e., the machine outputs reward r in state p on reading symbol ℓ)

Encoding Reward Machines in Propositional Logic

We use two sets of propositional variables to encode reward machines:

$d_{p,\ell,q}$ encodes the transition function of the reward machine (i.e., the machine transitions from state p to state q on reading symbol ℓ)

$o_{p,\ell,r}$ encodes the output function of the reward machine (i.e., the machine outputs reward r in state p on reading symbol ℓ)

Enforcing deterministic functions

We impose pseudo-Boolean constraints to enforce for each pair of state p and input a that

- exactly one variable $d_{p,\ell,q}$ is set to `true`
- exactly one variable $o_{p,\ell,r}$ is set to `true`

We introduce a set of auxiliary variables:

$x_{\lambda,p}$ encodes the run of the reward machine on all prefixes of examples (i.e., the machine reaches state p after reading the prefix λ)

We introduce a set of auxiliary variables:

$x_{\lambda,p}$ encodes the run of the reward machine on all prefixes of examples (i.e., the machine reaches state p after reading the prefix λ)

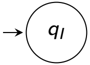
Enforcing consistency with the examples

$$\Big[\bigwedge_{u \in Pref(X)} \mathrm{one}(x_{u,q_1}, \ldots, x_{u,q_n}) \Big] \land x_{\varepsilon,q_I}$$

$$(x_{\lambda,p} \land d_{p,\ell,q}) \rightarrow x_{\lambda\ell,q}$$

$$x_{\lambda,p} \rightarrow o_{p,\ell,r}$$

reward machine:



x_{ε,q_I}

We introduce a set of auxiliary variables:

$x_{\lambda,p}$ encodes the run of the reward machine on all prefixes of examples (i.e., the machine reaches state p after reading the prefix λ)

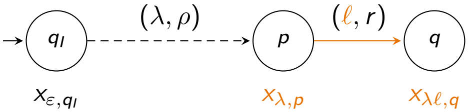
Enforcing consistency with the examples

$$\Big[\bigwedge_{u \in Pref(X)} \text{one}(x_{u,q_1}, \ldots, x_{u,q_n}) \Big] \wedge x_{\varepsilon,q_I}$$

$$(x_{\lambda,p} \wedge d_{p,\ell,q}) \rightarrow x_{\lambda\ell,q}$$

$$x_{\lambda,p} \rightarrow o_{p,\ell,r}$$

reward machine:



We introduce a set of auxiliary variables:

$x_{\lambda,p}$ encodes the run of the reward machine on all prefixes of examples (i.e., the machine reaches state p after reading the prefix λ)

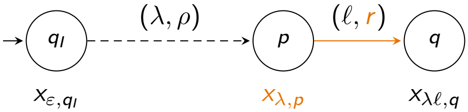
Enforcing consistency with the examples

$$\Big[\bigwedge_{u \in Pref(X)} \text{one}(x_{u,q_1}, \ldots, x_{u,q_n}) \Big] \land x_{\varepsilon,q_I}$$

$$(x_{\lambda,p} \land d_{p,\ell,q}) \rightarrow x_{\lambda\ell,q}$$

$$x_{\lambda,p} \rightarrow o_{p,\ell,r}$$

reward machine:



Theorem (Ahmad, Gavran, Majumdar, N., Topcu, Wu, and Xu)

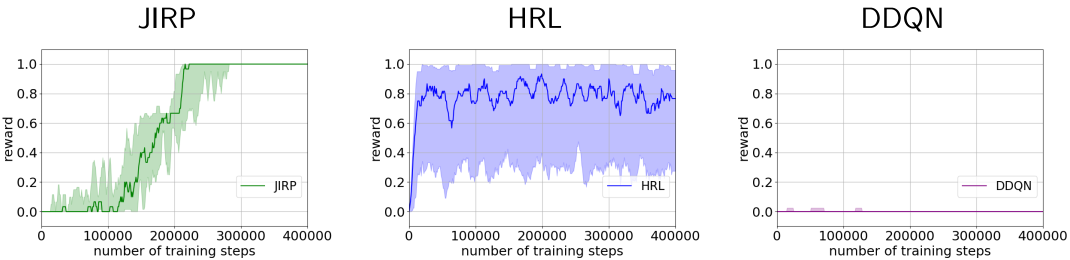
Given

- *a sufficient episode length*
- *an ε-greedy exploration strategy*

we have the following:

1. *JIRP almost surely learns the "true" reward machine*
2. *JIRP almost surely converges to an optimal policy*

Office World Scenario (Icarte et al., 2018)

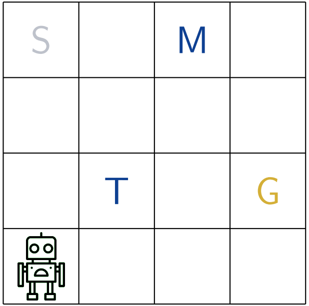


Conclusion

- JIRP is the only method that converges to an optimal policy
- JIRP converges faster than any of the competing methods

2. Reinforcement Learning with Stochastic Reward Machines

(joint work with Jan Corazza and Ivan Gavran)

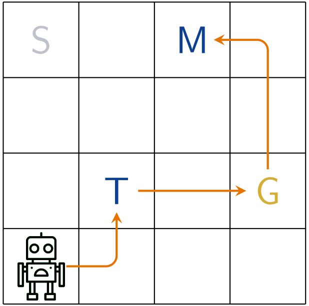


T: tools

M: market

S: silver mine

G: gold mine



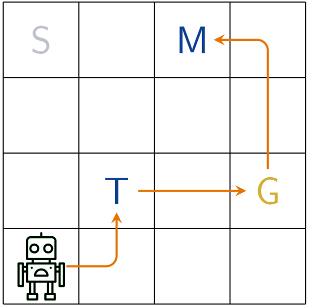
T; G; M:	1.9

T: tools

M: market

S: silver mine

G: gold mine



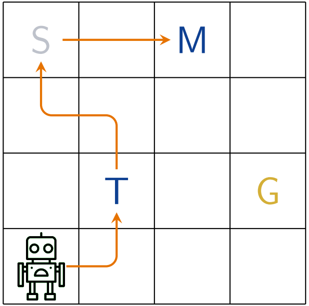
T; G; M:	1.9
T; G; M:	2.2

T: tools

M: market

S: silver mine

G: gold mine



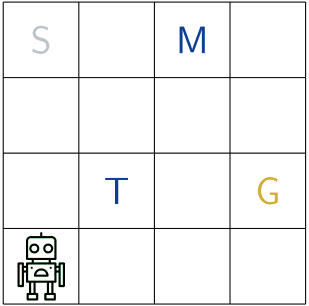
T; G; M:	1.9
T; G; M:	2.2
T; S; M:	1.2
T; S; M:	0.9

T: tools

M: market

S: silver mine

G: gold mine



T: tools

M: market

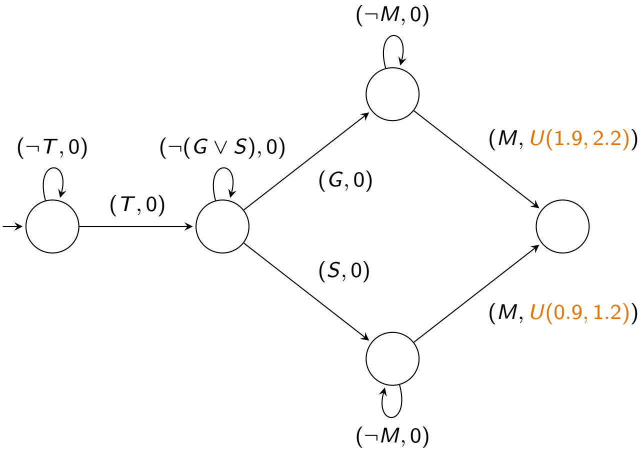
S: silver mine

G: gold mine

T; G; M:	1.9
T; G; M:	2.2
T; S; M:	1.2
T; S; M:	0.9

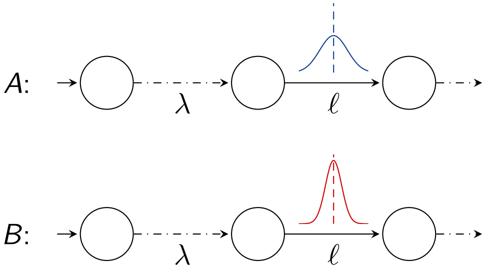
- If the label sequences are identical, no reward machine matches both traces
- If the label sequences are different, the resulting reward machine explodes in size

T; G; M:	1.9
T; G; M:	2.2
T; S; M:	1.2
T; S; M:	0.9

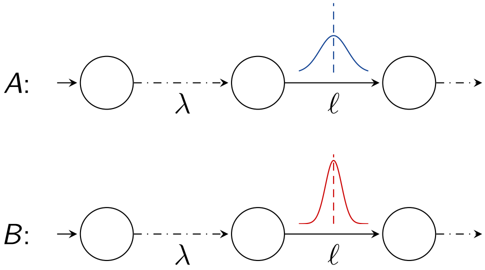


Outputs are bounded continuous distributions

Two SRMs A and B are equivalent in expectation ($A \sim_E B$) if they output sequences of distributions with equal expected values for each label sequence



Two SRMs A and B are equivalent in expectation ($A \sim_E B$) if they output sequences of distributions with equal expected values for each label sequence

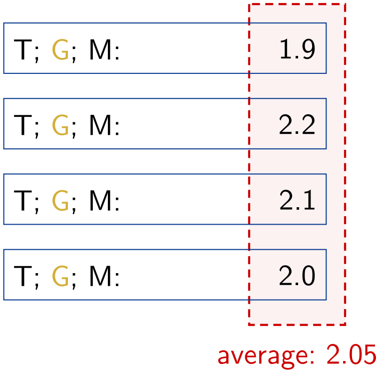


Corollary

If two SRMs are equivalent in expectation, then they induce the same optimal policy in an environment

A naive algorithm

1. Collect many samples
2. Take the average reward in every position of the same trajectory
3. Construct an ordinary reward machine based on the average rewards



A naive algorithm

1. Collect many samples
2. Take the average reward in every position of the same trajectory
3. Construct an ordinary reward machine based on the average rewards

Problem

Collecting samples is too slow!

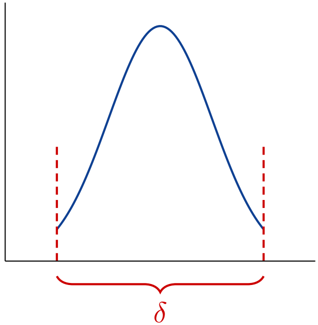
T; G; M: 1.9

T; G; M: 2.2

T; G; M: 2.1

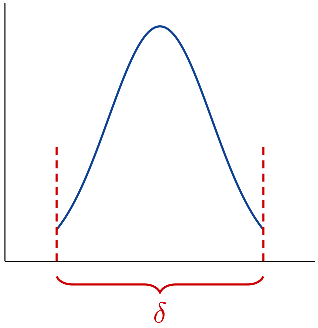
T; G; M: 2.0

average: 2.05

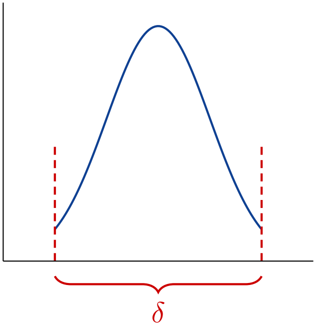


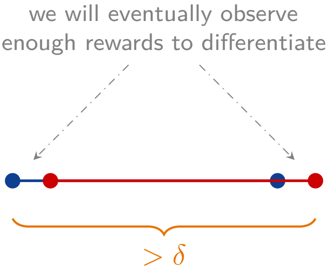
1. Probability distributions are continuous and have bounded support with "width" δ

One Can Do Better Under Two Assumptions

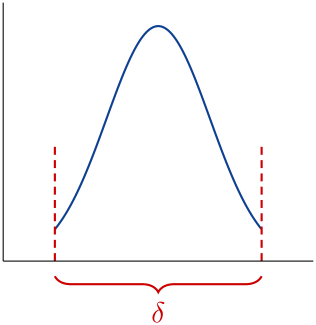


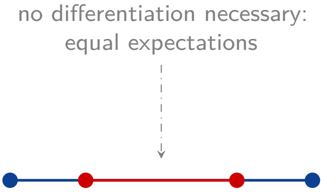
1. Probability distributions are continuous and have bounded support with "width" δ
2. The noise from one distribution does not fully conceal the signal from another one (except in symmetric circumstances)





1. Probability distributions are continuous and have bounded support with “width“ δ
2. The noise from one distribution does not fully conceal the signal from another one (except in symmetric circumstances)





1. Probability distributions are continuous and have bounded support with “width“ δ
2. The noise from one distribution does not fully conceal the signal from another one (except in symmetric circumstances)

Initialize H, Q, X, A;

repeat

$(\lambda, \rho, Q) \leftarrow \mathrm{QRM}(H, Q)$;

add (λ, ρ) to A; → one would keep a moving average in practice

Initialize H, Q, X, A;

repeat

$(\lambda, \rho, Q) \leftarrow \text{QRM}(H, Q)$;

add (λ, ρ) to A;

if H is not δ-consistent with (λ, ρ) **then**

add (λ, ρ) to X;

$H' \leftarrow \text{infer}(X)$;

infers a minimal δ-consistent "proto"-SRM

(only cares for δ-consistency, not estimating distribution)

Initialize H, Q, X, A;

repeat

$(\lambda, \rho, Q) \leftarrow \text{QRM}(H, Q)$;

add (λ, ρ) to A;

if H is not δ-consistent with (λ, ρ) **then**

add (λ, ρ) to X;

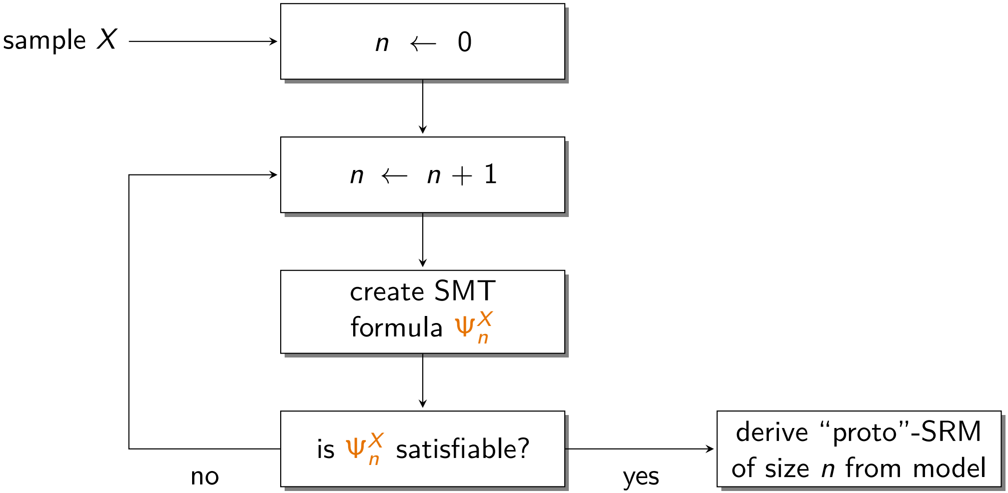
$H' \leftarrow \text{infer}(X)$;

$H \leftarrow \text{estimate}(H', A)$;

re-initialize Q if necessary;

corrects outputs of H' by estimating distribution parameters from samples in A

An SMT-Based Inference Algorithm for Stochastic Reward Machines



Encoding Stochastic Reward Machines in SMT

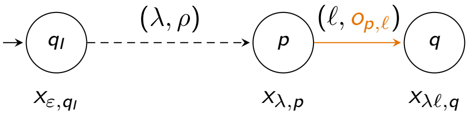
We use propositional and real-valued variables to encode a “proto”-SRM:

$d_{p,\ell,q} \in \mathbb{B}$	encodes the transition function of the reward machine
$x_{\lambda,p} \in \mathbb{B}$	encodes the run of the reward machine on prefixes from X
$o_{p,\ell} \in \mathbb{R}$	encodes a “conjectured mean” of an output distribution (i.e., the distr. returned in state p on reading symbol ℓ has mean $o_{p,\ell}$)

Enforcing consistency with the examples

$$x_{\lambda,p} \rightarrow |o_{p,\ell} - r| \leq \frac{\delta}{2}$$

SRM:



Theorem (Corazza, Gavran, N.)

Given

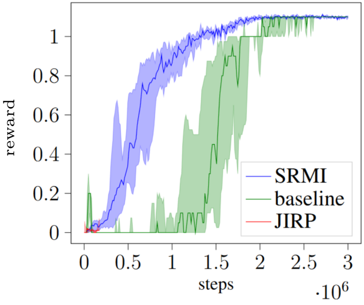
- *a sufficient episode length*
- *an ε-greedy exploration strategy*
- *Assumptions 1 and 2 hold for the "true" (environment) SRM*

we have the following:

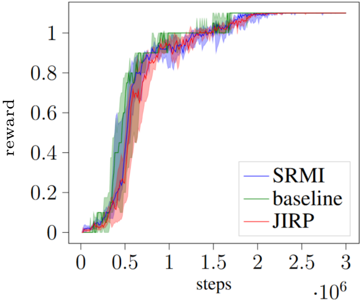
1. *SRMI almost surely learns a SRM that is equivalent in expectation to the "true" SRM*
2. *SRMI almost surely converges to an optimal policy*

Mining Environment

stochastic rewards



deterministic rewards



Conclusion

- SRMI converges faster than the baseline method
- SRMI's performance does not degrade in the case of deterministic rewards

Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

Inferring Probabilistic Reward Machines from Non-Markovian Reward Signals for Reinforcement Learning

Taylor Dohmen[1*], Noah Topper[2*], George Atia[2], Andre Beckus[3], Ashutosh Trivedi[1], Alvaro Velasquez[3]

[1] University of Colorado Boulder
[2] University of Central Florida
[3] Air Force Research Laboratory

Abstract

The success of reinforcement learning in typical settings is

2021). They also serve as a memory mechanism for reasoning over partially observable environments (Icarte et al. 2019), are useful for reward shaping to mitigate sparse re-

3. Advice-Guided Reinforcement Learning

(joint work with Jean-Raphaël Gaglione, Ivan Gavran, Ufuk Topcu, Bo Wu, and Zhe Xu) reward machine is given

Icarte et al., 2018

reward machine is inferred

Icarte et a., 2019
Furelos-Blanco et al., 2020
Gaon & Brafman, 2020
Xu et al., 2020

reward machine is given

Icarte et al., 2018

reward machine is inferred

Icarte et a., 2019
Furelos-Blanco et al., 2020
Gaon & Brafman, 2020
Xu et al., 2020

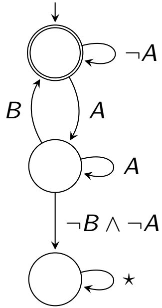
in this part

We formalize advice by means of regular languages:

- Deterministic Finite Automata (DFA)
- Regular expressions
- Linear Temporal Logic
- ...

We formalize advice by means of regular languages:

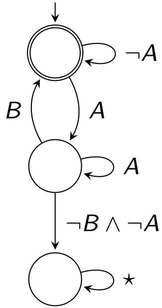
- Deterministic Finite Automata (DFA)
- Regular expressions
- Linear Temporal Logic
- ...



"every A is followed by B"

We formalize advice by means of regular languages:

- Deterministic Finite Automata (DFA)
- Regular expressions
- Linear Temporal Logic
- ...



"every A is followed by B"

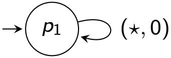
Compatibility of advice DFAs (i.e., semantics)

A reward can only be positive (negative/non-zero) if the advice DFA accepts the label sequence

- A reward machine satisfying this property is called compatible

Initialize reward machine H;

H:

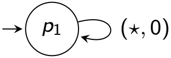


AdvisoRL

Initialize reward machine H;

Initialize a set Q of q-functions;

H:



Q: $\{q^{p_1}\}$

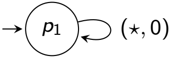
AdvisoRL

Initialize reward machine H;

Initialize a set Q of q-functions;

Initialize a sample X of traces;

H:



Q: $\{q^{p_1}\}$

X: $\emptyset$

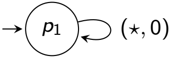
Initialize reward machine H;

Initialize a set Q of q-functions;

Initialize a sample X of traces;

Initialize a set D of advice DFAs;

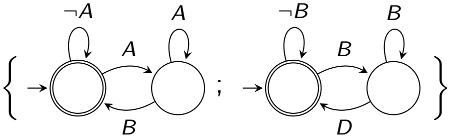
H:



Q: $\{q^{p_1}\}$

X: $\emptyset$

D:

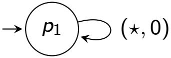


Initialize H, Q, X, D;

repeat

$(\lambda, \rho, Q) \leftarrow \mathrm{QRM}(H, Q)$;

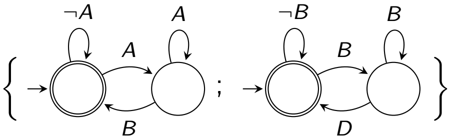
H:



Q: $\{q^{p_1}\}$

X: $\emptyset$

D:

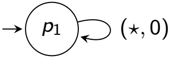


Initialize H, Q, X, D;

repeat

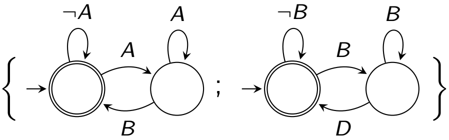
$(\lambda, \rho, Q) \leftarrow \mathrm{QRM}(H, Q)$;

$A; B; C; D$

H: 

Q: $\{q^{p_1}\}$

X: $\emptyset$

D: 

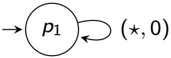
Initialize H, Q, X, D;

repeat

$(\lambda, \rho, Q) \leftarrow \text{QRM}(H, Q)$;

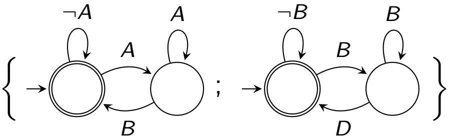
$A; B; C; D$

$0; 0; 0; 1$

H: 

Q: $\{q^{p_1}\}$

X: $\emptyset$

D: 

Initialize H, Q, X, D;

repeat

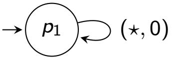
$(\lambda, \rho, Q) \leftarrow \text{QRM}(H, Q)$;

$A; B; C; D$

$0; 0; 0; 1$

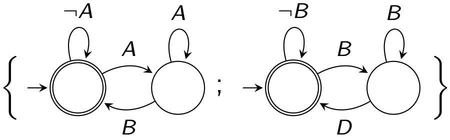
if $H(\lambda) \neq \rho$ **then**

add (λ, ρ) to X;

H: 

Q: $\{q^{p_1}\}$

X: $\emptyset$

D: 

Initialize H, Q, X, D;

repeat

$(\lambda, \rho, Q) \leftarrow \text{QRM}(H, Q)$;

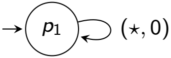
$A; B; C; D$

$0; 0; 0; 1$

if $H(\lambda) \neq \rho$ **then**

add (λ, ρ) to X;

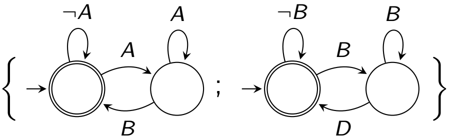
H:



Q: $\{q^{p_1}\}$

X: $\{(A; B; C; D / 0; 0; 0; 1)\}$

D:



Initialize H, Q, X, D;

repeat

$(\lambda, \rho, Q) \leftarrow \mathrm{QRM}(H, Q)$;

$A; B; C; D$

$0; 0; 0; 1$

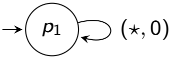
if $H(\lambda) \neq \rho$ **then**

add (λ, ρ) to X;

if (λ, ρ) is not compatible with some $\mathcal{D} \in D$ **then**

remove $\mathcal{D}$ from D;

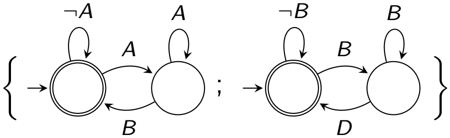
H:



Q: $\{q^{p_1}\}$

X: $\{(A; B; C; D / 0; 0; 0; 1)\}$

D:



Initialize H, Q, X, D;

repeat

$A; B; C; D$

$(\lambda, \rho, Q) \leftarrow \mathrm{QRM}(H, Q)$;

$0; 0; 0; 1$

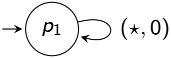
if $H(\lambda) \neq \rho$ **then**

add (λ, ρ) to X;

if (λ, ρ) is not compatible with some $\mathcal{D} \in D$ **then**

remove $\mathcal{D}$ from D;

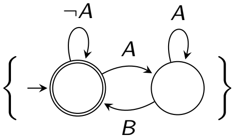
H:



Q: $\{q^{p_1}\}$

X: $\{(A; B; C; D / 0; 0; 0; 1)\}$

D:



AdvisoRL

Initialize H, Q, X, D;

repeat

$(\lambda, \rho, Q) \leftarrow \mathrm{QRM}(H, Q)$;

if $H(\lambda) \neq \rho$ **then**

add (λ, ρ) to X;

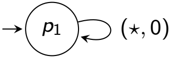
if (λ, ρ) is not compatible with some $\mathcal{D} \in D$ **then**

remove $\mathcal{D}$ from D;

if X or D were modified **then**

$H \leftarrow \mathrm{infer}(X, D)$;

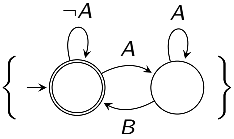
H:



Q: $\{q^{p_1}\}$

X: $\{(A; B; C; D/0; 0; 0; 1)\}$

D:



Initialize H, Q, X, D;

repeat

$(\lambda, \rho, Q) \leftarrow \text{QRM}(H, Q)$;

if $H(\lambda) \neq \rho$ **then**

add (λ, ρ) to X;

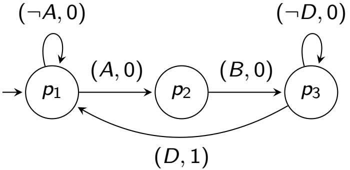
if (λ, ρ) is not compatible with some $\mathcal{D} \in D$ **then**

remove $\mathcal{D}$ from D;

if X or D were modified **then**

$H \leftarrow \text{infer}(X, D)$;

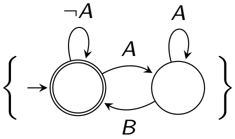
H:



Q: $\{q^{p_1}\}$

X: $\{(A; B; C; D/0; 0; 0; 1)\}$

D:



Initialize H, Q, X, D;

repeat

$(\lambda, \rho, Q) \leftarrow \text{QRM}(H, Q)$;

if $H(\lambda) \neq \rho$ **then**

add (λ, ρ) to X;

if (λ, ρ) is not compatible with some $\mathcal{D} \in D$ **then**

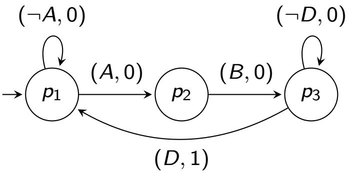
remove $\mathcal{D}$ from D;

if X or D were modified **then**

$H \leftarrow \text{infer}(X, D)$;

re-initialize Q if necessary;

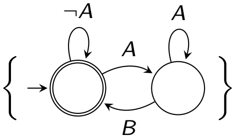
H:



Q: $\{q^{p_1}\}$

X: $\{(A; B; C; D/0; 0; 0; 1)\}$

D:



Initialize H, Q, X, D;

repeat

$(\lambda, \rho, Q) \leftarrow \mathrm{QRM}(H, Q)$;

if $H(\lambda) \neq \rho$ **then**

add (λ, ρ) to X;

if (λ, ρ) is not compatible with some $\mathcal{D} \in D$ **then**

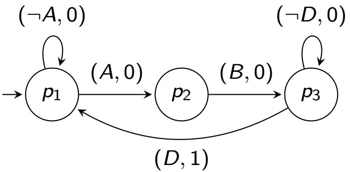
remove $\mathcal{D}$ from D;

if X or D were modified **then**

$H \leftarrow \mathrm{infer}(X, D)$;

re-initialize Q if necessary;

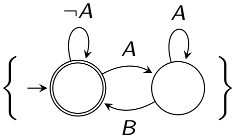
H:



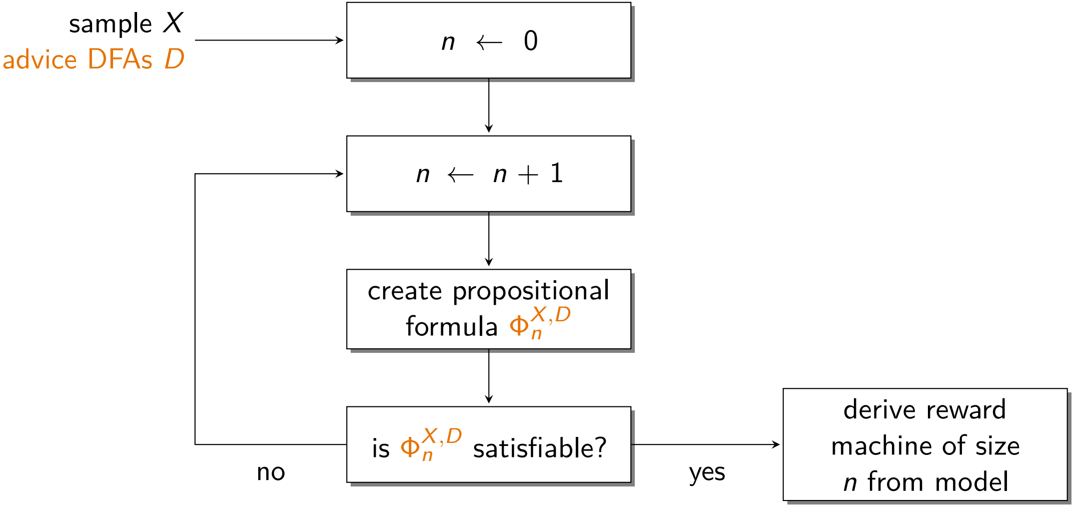
Q: $\{q^{p_1}, q^{p_2}, q^{p_3}\}$

X: $\{(A; B; C; D/0; 0; 0; 1)\}$

D:



A SAT-Based Inference Algorithm for Reward Machines



Theorem (N., Gaglione, Gavran, Topcu, Wu, Xu)

Given

- *a sufficient episode length*
- *an ε-greedy exploration strategy*

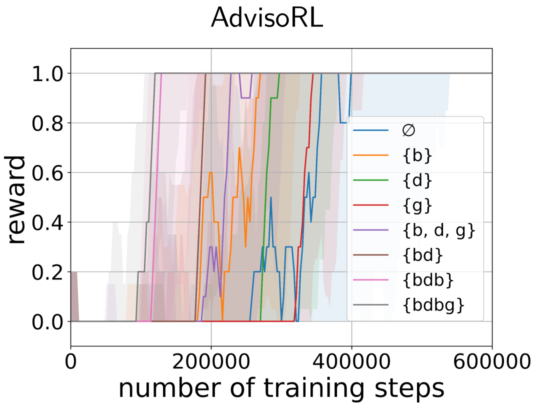
we have the following:

1. *AdvisoRL almost surely learns the "true" reward machine*
2. *AdvisoRL almost surely converges to an optimal policy*

Conclusion

- AdvisoRL's performance improves with the "quality" of the given advice
- AdvisoRL is robust to incorrect advice

Office World Scenario (Icarte et al., 2018)



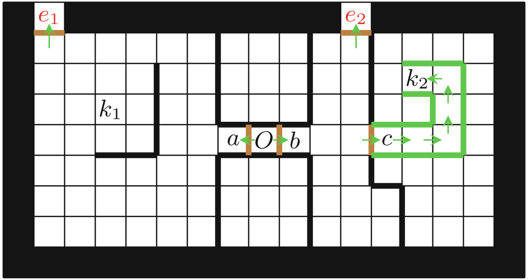
Conclusion

Reinforcement Learning with Temporal-Logic-Based Causal Diagrams

Yash Paliwal[1], Rajarshi Roy[2(✉)], Jean-Raphaël Gaglione[3], Nasim Baharisangari[1], Daniel Neider[4,5], Xiaoming Duan[6], Ufuk Topcu[3], and Zhe Xu[1]

[1] Arizona State University, Tempe, AZ, USA
xzhe1@asu.edu
[2] Max Planck Institute for Software Systems, Kaiserslautern, Germany
rajarshi008@gmail.com
[3] University of Texas at A[illegible]tin, Aust[illegible]

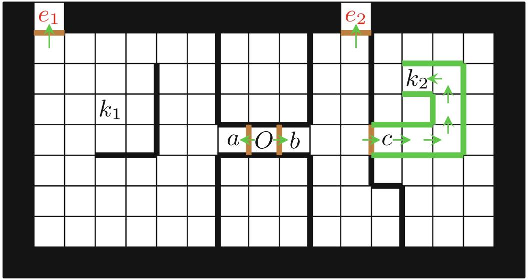
Grid World Example



b ▶ $\mathsf{G}\,\neg e_1$

c ▶ $\mathsf{X}\,\mathsf{X}\,\mathsf{X}\,\mathsf{X}\,\mathsf{X}\,k_2$

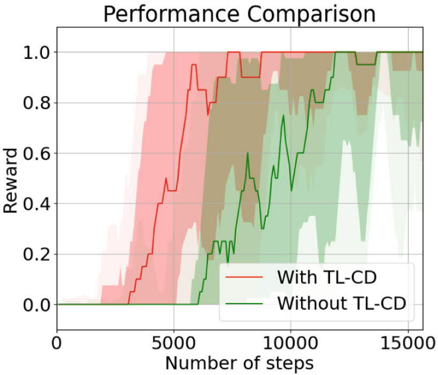
k_2 ▶ $\mathsf{G}\,\neg e_2$



$b \blacktriangleright \mathsf{G}\, \neg e_1$

$c \blacktriangleright \mathsf{X}\,\mathsf{X}\,\mathsf{X}\,\mathsf{X}\,\mathsf{X}\, k_2$

$k_2 \blacktriangleright \mathsf{G}\, \neg e_2$



Summary

- We have been on a journey through reinforcement learning with reward machines
- There are several extension (often by other research groups)
  - partial observability, active automata learning, etc.

Future work

- Incorporating (temporal) causal information
- Automatically synthesizing high level propositions
- More expressive classes of finite-state machines (e.g., counter)

Newly Established Research Center at UA Ruhr

CENTER FOR TRUSTWORTHY DATA SCIENCE AND SECURITY

UA RUHR | RESEARCH ALLIANCE

- **Three universities:** University of Duisburg-Essen, University of Bochum, TU Dortmund University
- **Four disciplines:** Computer science, IT Security, Statistics, Psychology

We offer opportunities . . .

- Collaborations with academia and industry
- Open positions for research group leaders, postdocs, Ph.D.s, students
- Internship program
- . . .