Reward Machines: Structuring reward function specifications and reducing sample complexity in reinforcement learning

Sheila A. McIlraith
Department of Computer Science
University of Toronto

CLP 2019
International Conference on Logic Programming

September 25, 2019
Acknowledgements

Rodrigo Toro Icarte
Acknowledgements

Rodrigo Toro Icarte  Toryn Klassen  Richard Valenzano

[Toro Icarte, Klassen, Valenzano, M., ICML 2018; AAMAS 2018; CAI 2018]
... and most recently with

Alberto Camacho

[Camacho, Toro Icarte, Klassen, Valenzano, M., IJCAI 2019]
[Camacho, Chen, Sanner, M., SoCS2017]
... and with

Léon Illanes  Xi Yan  Ethan Waldie  Margarita Castro

[Illanes, Yan, Toro Icarte, M., RLDM 2019]
[Toro Icarte, Waldie, Klassen, Valenzano, Castro, M., NeurIPS 2019]
Deep Reinforcement Learning

It might look goofy ...

https://www.youtube.com/watch?v=gn4nRCC9TwQ
LANGUAGE
Humans have evolved languages over thousands of years to provide useful abstractions for understanding and interacting with each other and with the physical world. There are roughly 6500 spoken languages in the world today – though at least 2000 have fewer that 1000 speakers.

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

While psychologists continue to debate how (and whether) language shapes the way we think, there is some agreement that the alphabet and structure of a language can have a significant impact on learning and reasoning.
We use language to communicate high-level goals, intentions and objectives, and to support coordination with others.

We also use language to teach – to transfer knowledge.

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

Can exploiting the alphabet and structure of language help RL agents learn and think?
How do we advise, instruct, task, ... and impart knowledge to our RL agents?

Photo: Javier Pierin (Getty Images)
Goals and Preferences

• Run the dishwasher when it’s full or when dishes are needed for the next meal.

• Make sure the bath temperature is between 38 – 43 celcius immediately before letting someone enter the bathtub.

• Do not vacuum while someone in the house is sleeping.

• While there are dirty dishes on the counter, load them into the dishwasher.

• ...
Goals and Preferences

• When getting ice cream, please always open the freezer, take out the ice cream, serve yourself, put the ice cream back in the freezer, and close the freezer door.
Linear Temporal Logic (LTL)

A compelling logic to express temporally extended properties of state traces

\[ \varphi = T \mid F \mid p \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \varphi^0 \mid \varphi U \varphi \]

**Syntax of LTL**

- **Atomic Propositions:** \( AP \)
- **Logic connectives:** \( \land, \lor, \neg \)
- **next:** \( \circ \varphi \)
- **until:** \( \psi U \chi \)
- **eventually:** \( \diamond \varphi = \text{true} U \varphi \)
- **always:** \( \square \varphi = \neg \diamond \neg \varphi \)
- **release:** \( \psi R \chi = \neg (\neg \psi U \neg \chi) \)

**Properties**

- LTL can be interpreted over **finite** or **infinite** words
- LTL can be transformed into automata.

**Figure:** NBA for LTL formula \( \square (p \rightarrow \diamond \diamond q) \)
Goals and Preferences

• Do not vacuum while someone is sleeping

always[¬ (vacuum ∧ sleeping)]
Goals and Preferences

• Do not vacuum while someone is sleeping

  $$\text{always}[\neg (\text{vacuum} \land \text{sleeping})]$$

• When getting an ice cream for someone ...

  $$\text{always}[\text{get(ice-cream)} \rightarrow \text{eventually } [\text{open(freezer)} \land \text{next[remove(ice-cream,freezer)} \land \text{next[serve(ice-cream)} \land \text{next[replace(ice-cream,freezer)} \land \text{next[close(freezer)]}]})]$$
How do we communicate this to our RL agent?
MOTIVATION
Challenges to RL

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

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

Agent

Action

Environment
Transition Function
Reward Function

Reward

State
Running Example

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>▲</td>
<td>Agent</td>
</tr>
<tr>
<td>*</td>
<td>Furniture</td>
</tr>
<tr>
<td>☕</td>
<td>Coffee Machine</td>
</tr>
<tr>
<td>💌</td>
<td>Mail Room</td>
</tr>
<tr>
<td>☻</td>
<td>Office</td>
</tr>
<tr>
<td>A, B, C, D</td>
<td>Marked Locations</td>
</tr>
</tbody>
</table>

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Toy Problem Disclaimer

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

These “toy problems” challenge state-of-the-art RL techniques.
Task: Visit A, B, C, and D, in order.

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

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

Reward Function (as part of environment)

[Diagram showing a grid with positions A, B, C, D and reward conditions]

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

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Task: Visit A, B, C, and D, in order.
Task: Visit A, B, C, and D, in order.

Reward Function (as part of environment)
Task: Visit A, B, C, and D, in order.
count = 0  # global variable

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

**Task:** Visit A, B, C, and D, in order.
Task: Visit A, B, C, and D, in order.
count = 0  # global variable

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

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

**Reward Function** (as part of environment)

---

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Task: Visit A, B, C, and D, in order.
Task: Visit A, B, C, and D, in order.

Reward Function (as part of environment)
Simple Idea:

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

The agent can exploit structure in the reward function.

count = 0  # global variable

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

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Decoupling Transition and Reward Functions

Agent

Action

Environment
Transition Function
Reward Function

Reward

State
Decoupling Transition and Reward Functions

Agent

Action

Reward

Environment

Transition Function

State
The Rest of the Talk

- Reward Machines (RM)
  - Exploiting RM Structure in Learning
  - Experiments
  - Creating Reward Machines
  - Recap
REWARD MACHINES
count = 0  # global variable

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

Encode reward function in an automata-like structure
Reward Machine
Reward Machine

- finite set of states $U$
Reward Machine

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

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

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

Conditions are over properties of the current state:

$$P = \{\text{\ding{100}}, \text{\ding{102}}, o, *, A, B, C, D\}$$
Reward Machine

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

Conditions are over properties of the current state:

$$P = \{\square, \bigcirc, o, *, A, B, C, D\}$$

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

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action

State

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
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

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action

State

Mclraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action
Reward Machines in Action
Reward Machines in Action
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action
Reward Machines in Action
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action
Reward Machines in Action
Reward Machines in Action
Reward Machines in Action
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action
Reward Machines in Action
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Reward Machines in Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
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 to the office.
Other Reward Machines

**Task:** Deliver coffee to the office, while avoiding furniture.
Other Reward Machines

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

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

Task: Deliver coffee and mail to the office.

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Other Reward Machines

**Task:** Deliver coffee and mail to the office.
Other Reward Machines

Task: Deliver coffee and mail to the office.
The Rest of the Talk

- Reward Machines (RM)

► Exploiting RM Structure in Learning

- Experiments

- Creating Reward Machines

- Recap
EXPLOITING RM STRUCTURE IN LEARNING
Methods for Exploiting RM Structure

We have explored 5 ideas:

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

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

\[ NewQ(s, a) = Q(s, a) + \alpha [R(s, a) + \gamma \max Q'(s', a') - Q(s, a)] \]

- New Q-value for that state and that action
- Current Q-value
- Reward for taking that action at that state
- Learning Rate
- Discount rate
- Maximum expected future reward given the new s' and all possible actions at that new state
1. Q-Learning Baseline

A Reward Machine may define a non-Markovian reward function.
1. Q-Learning Baseline

A Reward Machine may define a non-Markovian reward function.
1. Q-Learning Baseline

A Reward Machine may define a non-Markovian reward function.
1. Q-Learning Baseline

A Reward Machine may define a non-Markovian reward function.
1. Q-Learning Baseline

A Reward Machine may define a non-Markovian reward function.
1. Q-Learning Baseline

A Reward Machine may define a non-Markovian reward function.
1. Q-Learning Baseline

**Solution:** Include RM state as part of agent’s state representation. Use standard Q-learning on resulting MDP.
2. Option-Based Hierarchical RL (HRL)

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

- RM refers to A, B, C, and D
- Learn policies $\pi_A$, $\pi_B$, $\pi_C$, and $\pi_D$
- Optimize $\pi_i$, to satisfy $i$ optimally
2. Option-Based Hierarchical RL (HRL)

Simultaneously learning when to use each option policy.
3. HRL with RM-Based Pruning (HRL-RM)

Prune irrelevant options using current RM state
3. HRL with RM-Based Pruning (HRL-RM)

Prune irrelevant options using current RM state

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
HRL Methods Can Find Suboptimal Policies

HRL approaches find “locally” optimal solutions.
HRL Methods Can Find Suboptimal Policies

HRL approaches find “locally” optimal solutions.
HRL Methods Can Find Suboptimal Policies

HRL approaches find “locally” optimal solutions.
HRL Methods Can Find Suboptimal Policies

Learns two options:
1. Getting
2. Getting to “o”

HRL approaches find “locally” optimal solutions.

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
4. Q-Learning for Reward Machines (QRM)
4. Q-Learning for Reward Machines (QRM)

**QRM (our approach)**

1. Learn one policy (q-value function) per state in the RM.
4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

1. Learn one policy (q-value function) per state in the RM.
2. Select actions using the policy of the current RM state.
4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

1. Learn one policy (q-value function) per state in the RM.
2. Select actions using the policy of the current RM state.
4. Q-Learning for Reward Machines (QRM)

**QRM (our approach)**

1. Learn one policy (q-value function) per state in the RM.
2. Select actions using the policy of the current RM state.
4. Q-Learning for Reward Machines (QRM)

**QRM (our approach)**

1. Learn one policy (q-value function) per state in the RM.
2. Select actions using the policy of the current RM state.
4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

1. Learn one policy (q-value function) per state in the RM.
2. Select actions using the policy of the current RM state.
4. Q-Learning for Reward Machines (QRM)

QRM (our approach)

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

2. Select actions using the policy of the current RM state.

3. Reuse experience to update all q-value functions on every transition via off-policy reinforcement learning.
QRM In Action

\[
\begin{array}{|c|c|c|c|}
\hline
B & * & * & C \\
\hline
* & O & * & * \\
\hline
A & * & * & D \\
\hline
\end{array}
\]
QRM In Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
QRM In Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
QRM In Action

\[ q_0(s, a) \leftarrow 0 + \gamma \cdot \max_{a'} q_0(s', a') \]
QRM In Action

\[ S' \]

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
QRM In Action

\[ q_1(s, a) \leftarrow 0 + \gamma \cdot \max_{a'} q_1(s', a') \]
QRM In Action
QRM In Action

\[ q_2(s, a) \leftarrow 0 + \gamma \cdot \max_{a'} q_2(s', a') \]
QRM In Action

\[ S' \]
QRM In Action

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
QRM In Action

\[ q_3(s, a) \leftarrow 1 + \gamma \cdot \max_{a'} q_0(s', a') \]
\(q_3(s, a) \leftarrow 1 + \gamma \cdot \max_{a'} q_0(s', a')\)
Recall: Methods for Exploiting RM Structure

We have explored 5 ideas:

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

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

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

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

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

will not change the set of optimal policies.

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

[Ng, Harada, Russell, 1999]
5. QRM + Reward Shaping (QRM + RS)

QRM + RS (our approach)
1. Treat the RM itself as an MDP and perform value iteration over the RM.
2. Apply QRM to the shaped RM

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
**Theorem:** QRM converges to the optimal policy in the limit, as does QRM + RS.
The Rest of the Talk

- Reward Machines (RM)
- Exploiting RM Structure in Learning

▶ Experiments

- Creating Reward Machines
- Concluding Remarks
EXPERIMENTS
Test Domains

• Two domains with a discrete action and state-space
  ▪ Office domain (4 tasks)
  ▪ Craft domain (10 tasks)

• One domain with a continuous state-space
  ▪ Water World domain (10 tasks)
Test in Discrete Domains

Tested all five approaches

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Optimality?</th>
<th>Decomposition?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-Learning</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>HRL</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>HRL-RM</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>QRM</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>QRM + RS</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Office World Experiments

4 tasks, 30 independent trials per task

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Office World Experiments

4 tasks, 30 independent trials per task
Minecraft World Experiments

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

Tasks from Andreas et al. (ICML 2017)
Minecraft World Experiments

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

Tasks from Andreas et al. (ICML 2017)
Function Approximation with QRM

From tabular QRM to Deep QRM

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Optimality?</th>
<th>Decomposition?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-Learning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HRL</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>HRL-RM</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>QRM</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>QRM + RS</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Water World Experiments

10 tasks over 10 random maps, 3 independent trials per combination
Water World Experiments

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

Legend:
- DDQN
- DHRL
- DHRL-RM
- DQRM

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
QRM + Reward Shaping (QRM + RS)

Discount factor $\gamma$ of 0.9 and exploration constant $\epsilon$ of 0.1

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
The Rest of the Talk

• Reward Machines (RM)

• Exploiting RM Structure in Learning

• Experiments

▶ Creating Reward Machines

• Recap
CREATING REWARD MACHINES
Creating Reward Machines

Where do Reward Machines come from?

1. Specify RM
   - Directly
   - Via automatic translation from specifications in various languages

2. Generate RM from high-level goal specifications

3. Learn RM
1. Reward Specification: one size does not fit all

Do not need to specify Reward Machines directly.

Reward Machines are a form of Mealy Machine.

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

The Chomsky Hierarchy

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
1. **Reward Specification:** one size does *not* fit all

Many of these languages are natively declarative and composable. E.g.,

- Dialects of Linear Temporal Logic (LTL) including LTL$_f$, PLTL, ...
- Linear Dynamic Logic (LTD), LDL$_f$
- LTL-RE, Golog-variants
- Regular languages

**KEY IDEA:**

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

[Camacho, Toro Icarte, Klassen, Valenzano, M., IJCAI 2019]
2. Generate RM using a Symbolic Planner

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

[Leon Illanes; Xi Yan; Rodrigo A Toro Icarte; M., RLDM19]
Problem: Find a policy that maximizes the collected external reward given by a partially observable environment

Assumptions: The agent has access to a set of high-level binary classifiers (e.g., rooms, cookies, keys, etc.)

Key Insight: Learn an RM such that its internal state can be effectively used as external memory by the agent to solve the task.
3. Learn RMs for Partially-Observable RL

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

**Assumptions:** The agent has access to a set of high-level binary classifiers (e.g., rooms, cookies, keys, etc.)

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

These “toy problems” cannot be solved by A3C, PPO, and ACER with LSTMs
**Task**: Eat as many cookies as possible before the timeout.

- The button 🍪 causes a cookie 🍪 to randomly appear in the red 🟥 or blue 🌈 room.
- The agent receives a +1 reward for eating a cookie.
- Pressing 🍪 before reaching 🍪 will randomly move it.
- There is no cookie at the beginning of the episode.
Results Cookie Domain

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Approach

Perfect Reward Machines
Perfect RMs make the environment Markovian w.r.t. $O \times U$, i.e.:

$$
\Pr(o_{t+1}, r_{t+1}|o_0, a_0, \ldots, o_t, a_t) = \Pr(o_{t+1}, r_{t+1}|o_t, u_t, a_t)
$$

for every possible trace $o_0, a_0, \ldots, o_t, a_t$ generated by any policy.

- Learning RM posed as discrete optimization problem.
- Solved using Tabu search.

[Toro Icarte; Waldie; Klassen; Valenzano; Castro; M, NeurIPS 2019]
RECAP
Can exploiting the alphabet and structure of language help RL agents learn and think?
Key Insight: Reveal Reward Function to the Agent

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

Reward Function
(as part of environment)

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Key Insight: Reveal Reward Function to the Agent

count = 0  # global variable

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

Reward Machines serves as a normal form representation for reward functions. They

• supports **reward-function-tailored q-learning** via QRM and DQRM (and **reward shaping**) while preserving convergence guarantees.

• **Reduce sample complexity**, allowing problems to be solved that could not otherwise be reasonably solves by state-of-the-art RL methods

• serve as a **lingua franca** for specification of reward-worthy behavior in a diversity of other languages (LTL variants, regular languages, etc.) allowing **composition of multiple specifications** and all benefiting from QRM/DQRM/reward shaping

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
Great Results in Discrete Domains

QRM outperforms HRL and standard Q-learning in two domains

McIlraith, Toro Icarte, Valenzano, Klassen, 2019
...and in Continuous Domains

... and is also effective when combined with deep learning
...and they can be learned in partially observable environments to solve hard problems
Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning
Toro Icarte, Klassen, Valenzano, McIlraith
ICML 2018
Code: https://bitbucket.org/RToroIcarte/qrm

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

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

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

Play with the code, read the papers, ...
Other related work

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

Non-Markovian Rewards Expressed in LTL: Guiding Search Via Reward Shaping (Extended Version)
Camacho, Chen, Sanner, McIlraith
Extended Abstract: SoCS 2017, RLDM 2017
*Linear temporal logic (LTL) formulas are used to express non-Markovian reward in fully specified MDPs. LTL is translated to automata and reward shaping is used over the automata to help solve the MDP.*
Thanks to my group (current and alum)!

McIlraith, Toro Icarte, Valenzano, Klassen, 2019