# Advice-Based Exploration in Model-Based Reinforcement Learning

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

[1]University of Toronto, Toronto, Canada
{rntoro,toryn,rvalenzano,sheila}@cs.toronto.edu

[2]Vector Institute, Toronto, Canada

[3]Element AI, Toronto, Canada

May 11, 2018

# Advice-Based Exploration in Model-Based Reinforcement Learning

Rodrigo Toro Icarte    Richard Valenzano    Sheila A. McIlraith

# Motivation

Reinforcement Learning (RL) is a way of discovering how to act.

- exploration by performing random actions
- exploitation by performing actions that led to rewards

Applications include **Atari games** (Mnih et al., 2015), **board games** (Silver et al., 2017), and **data center cooling**[1].

However, very large amounts of training data are often needed.

[1] www.technologyreview.com/s/601938/the-ai-that-cut-googles-energy-bill-could-soon-help-you/

# Humans learning behavior aren't limited to pure RL.

Humans can use

- demonstrations
- feedback
- advice

What is advice?

- recommendations regarding behaviour that
  - may describe suboptimal ways of doing things,
  - may not be universally applicable,
  - or may even contain errors
- Even in these cases people often extract value and we aim to have RL agents do likewise.

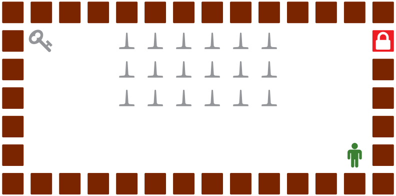
# Our contributions

- We make the first proposal to use Linear Temporal Logic (LTL) to **advise** reinforcement learners.
- We show how to use LTL advice to do model-based RL faster (as demonstrated in experiments).

# Outline

- background
  - MDPs
  - reinforcement learning
    - model-based reinforcement learning
- advice
  - the language of advice: LTL
  - using advice to guide exploration
  - experimental results

# Running example



**Actions**:

- move_left, move_right, move_up, move_down
- They fail with probability 0.2

**Rewards**:

- Door +1000; nail -10; step -1

**Goal**:

- Maximize cumulative reward

# Markov Decision Process

$\mathcal{M} = \langle S, s_0, A, \gamma, T, R \rangle$

- $S$ is a finite set of states.
- $s_0 \in S$ is the initial state.
- $A$ is a finite set of actions.
- $\gamma$ is the discount factor.
- $T(s'|s, a)$ is the transition probability function.
- $R(s, a)$ is the reward function.

**Goal**: Find the optimal **policy** $\pi_*(a|s)$

# Given the model, we can compute an optimal policy.

We can compute $\pi_*(a|s)$ by solving the Bellman equation:

$$Q_*(s,a) = R(s,a) + \gamma \sum_{s'} T(s'|s,a) \max_{a'} Q_*(s',a')$$

and then

$$\pi_*(a|s) = \max_a Q_*(s,a)$$

# What if we don't know $T(s'|s,a)$ or $R(s,a)$?

Reinforcement learning methods try to find $\pi_*(a|s)$ by sampling from $T(s'|s,a)$ and $R(s,a)$.

# Reinforcement Learning

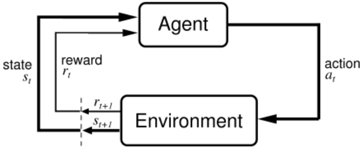
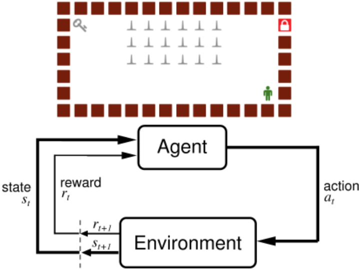
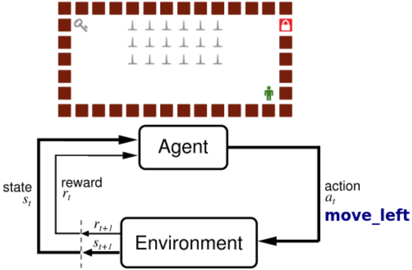



Diagram from Sutton and Barto (1998, Figure 3.1)

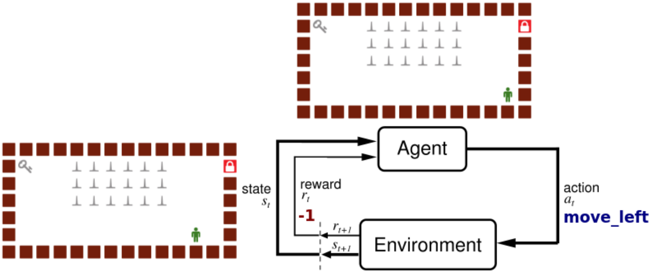
# Reinforcement Learning



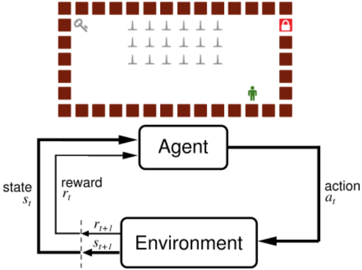
# Reinforcement Learning



# Reinforcement Learning



# Reinforcement Learning



# Two kinds of reinforcement learning

**model-free RL:** a policy is learned **without** explicitly learning $T$ and $R$

**model-based RL:** $T$ and $R$ are learned, and a policy is constructed based on them

# Model-Based Reinforcement Learning

**Idea**: Estimate $R$ and $T$ from experience (by counting):

$$\hat{R}(s,a) = \frac{1}{n(s,a)} \sum_{i=1}^{n(s,a)} r_i \qquad \hat{T}(s'|s,a) = \frac{n(s,a,s')}{n(s,a)}$$

While learning the model, how should the agent behave?

# Algorithms for Model-Based Reinforcement Learning

We'll consider **MBIE-EB** (Strehl and Littman, 2008), though in the paper we talk about R-MAX, another algorithm.

- Initialize $\hat{Q}(s,a)$ optimistically:

$$\hat{Q}(s,a) = \frac{\mathrm{R}_{\mathrm{max}}}{1-\gamma}$$

- Compute the optimal policy with an exploration bonus:

$$\underbrace{\hat{Q}_*(s,a) = \hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \max_{a'} \hat{Q}_*(s',a')}_{\text{This part is like the Bellman equation (with estimates for } R \text{ and } T\text{)}} + \underbrace{\frac{\beta}{\sqrt{n(s,a)}}}_{\text{bonus}}$$

# MBIE-EB in action

Train

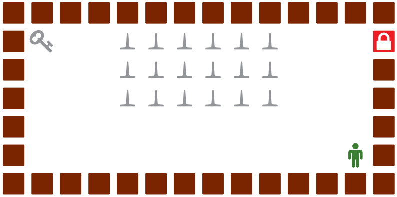
Test

How can we help this agent?

# Outline

- background
  - MDPs
  - reinforcement learning
    - model-based reinforcement learning
- advice
  - the language of advice: LTL
  - using advice to guide exploration
  - experimental results

# Advice



**Advice examples**:

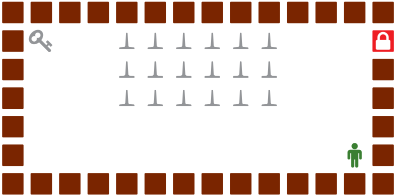
- Get the key and then go to the door
- Avoid nails

What we want to achieve with advice:

- speed up learning (if the advice is good)
- not rule out possible solutions (even if the advice is bad)

# Vocabulary

To give advice, we need to be able to describe the MDP in a symbolic way.



- Use a **labeling function** $L : S \to T(\Sigma)$
  - e.g., $\mathtt{at}(\mathtt{key}) \in L(s)$ iff the location of the agent is equal to the location of the key in state $s$.

# The language: LTL advice

Linear Temporal Logic (LTL) (Pnueli, 1977) provides temporal operators: **next** $\varphi$, $\varphi_1$ **until** $\varphi_2$, **always** $\varphi$, **eventually** $\varphi$.

**LTL advice examples**

- "Get the key and then go to the door" becomes **eventually**($\texttt{at}(\texttt{key}) \wedge$ **next eventually**($\texttt{at}(\texttt{door})$))
- "Avoid nails" becomes **always**($\forall(x \in \texttt{nails}).\neg\texttt{at}(x)$)

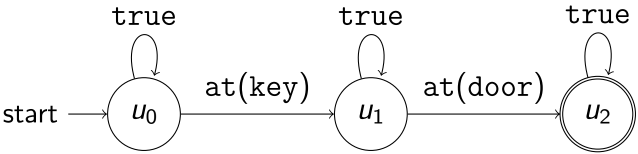
# Tracking progress in following advice

**LTL advice**

"Get the key and then go to the door"

**eventually**($\texttt{at}(\texttt{key}) \wedge$ **next eventually**($\texttt{at}(\texttt{door})$))

**Corresponding NFA:**



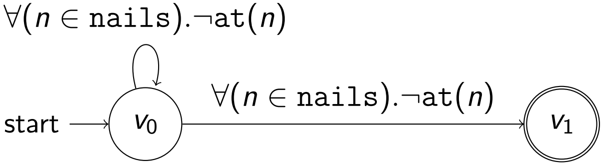
# Tracking progress in following advice

**LTL advice**

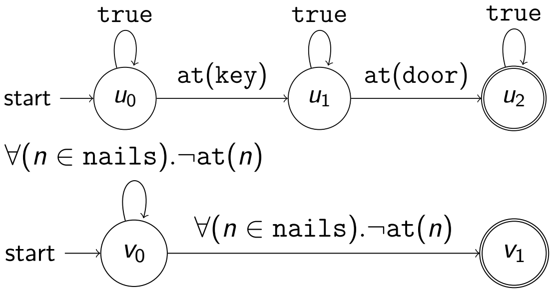
"Avoid nails"

**always**$(\forall(x \in \texttt{nails}).\neg\texttt{at}(x))$

**Corresponding NFA:**



# Guidance and avoiding dead-ends




From these, we can compute

- **guidance** formula $\hat{\varphi}_{guide}$
- **dead-ends avoidance** formula $\hat{\varphi}_{ok}$

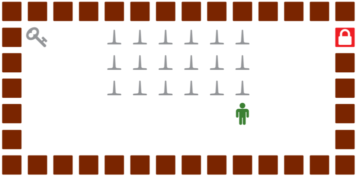
# The background knowledge function

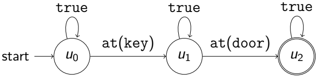
We use a function $h : S \times A \times \mathcal{L}_\Sigma \to \mathbb{N}$ to estimate the number of actions needed to make formulas true.

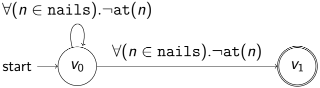
- the value of $h(s, a, \ell)$ for all **literals** $\ell$ has to be specified
  - e.g., we estimate the actions needed to make $\mathtt{at}(c)$ true using the Manhattan distance to $c$
- estimates for **conjunctions** or **disjunctions** are computed by taking maximums or minimums
  - e.g, $h(s, a, \mathtt{at}(\mathtt{key}_1) \vee \mathtt{at}(\mathtt{key}_2)) = \min\{h(s, a, \mathtt{at}(\mathtt{key}_1)),$
  $h(s, a, \mathtt{at}(\mathtt{key}_2))\}$

# Using $h$ with the guidance and avoidance formulas

$$\hat{h}(s,a) = \begin{cases} h(s,a,\hat{\varphi}_{guide}) & \text{if } h(s,a,\hat{\varphi}_{ok}) = 0 \\ h(s,a,\hat{\varphi}_{guide}) + C & \text{otherwise} \end{cases}$$






$$\hat{\varphi}_{guide} = \texttt{at}(\texttt{key}) \qquad \hat{\varphi}_{ok} = \forall(x \in \texttt{nails}).\neg\texttt{at}(x)$$

# MBIE-EB with advice

- Initialize $\hat{Q}(s,a)$ optimistically:

$$\hat{Q}(s,a) = \alpha(-\hat{h}(s,a)) + (1-\alpha)\frac{\mathrm{R_{max}}}{1-\gamma}$$

- Compute the optimal policy with an exploration bonus:

$$\hat{Q}_*(s,a) = \alpha(-1) + (1-\alpha)\hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \max_{a'} \hat{Q}_*(s',a') + \frac{\beta}{\sqrt{n(s,a)}}$$

# Advice in action

Train Test

**Advice:** get the key and then go to the door.

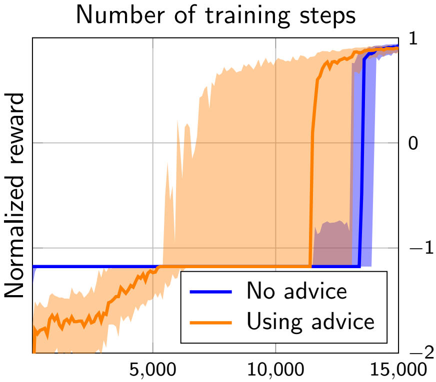
# Advice can improve performance.


**Advice:** get the key and then go the door, and avoid nails

# Less complete advice is also useful.


**Advice:** get the key and then go to the door

# As advice quality declines, so do early results.




**Advice:** get the key

# Bad advice can be recovered from.


**Advice:** go to every nail

# A larger experiment (with R-MAX-based algorithm)

**Advice:** for every key in the map, get it and then go to a door; avoid nails and holes; get all the cookies

# Conclusion

- Our approach can use LTL **advice** to reduce the training required while being robust to misleading advice.
    - The R-Max-based algorithm in the paper can be proved to converge to the optimal policy for deterministic MDPs.
- For using LTL to **define tasks**, see our AAMAS 2018 paper "Teaching Multiple Tasks to an RL Agent using LTL"
- Ideas for future work:
    - Learn the background knowledge function.
    - Use LTL advice in **model-free** RL as well.
    - Incorporate background knowledge that doesn't just give numeric estimates, but expresses propositions.
        - E.g. that halls normally lead to doors.

**Questions?**

# References

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015. doi:10.1038/nature14236.

Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977. doi:10.1109/SFCS.1977.32.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017. URL `http://arxiv.org/abs/1712.01815`.

Alexander L. Strehl and Michael L. Littman. An analysis of model-based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences*, 74(8):1309 – 1331, 2008. doi:10.1016/j.jcss.2007.08.009.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 1998.