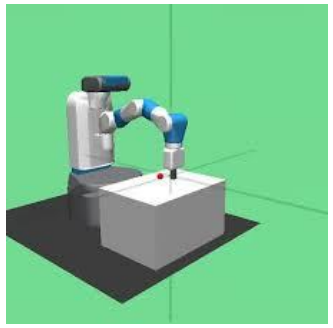
Specification-Guided Reinforcement Learning

Suguman Bansal

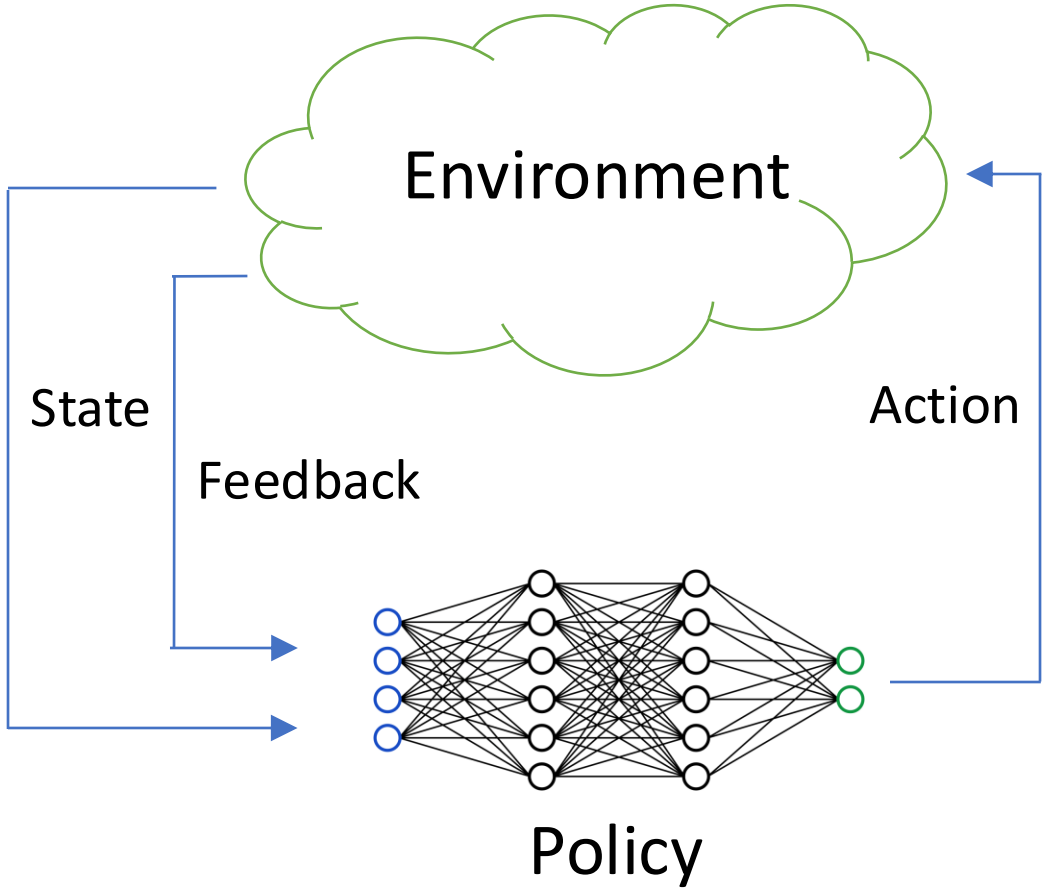
Georgia Institute of Technology

suguman@gatech.edu

Reinforcement Learning (RL)



Given a **task**,
generate a **policy** that
accomplishes it



A Simple Example

State is the position of the robot
$\boldsymbol{s = (x, y) \in \mathbb{R}^2}$

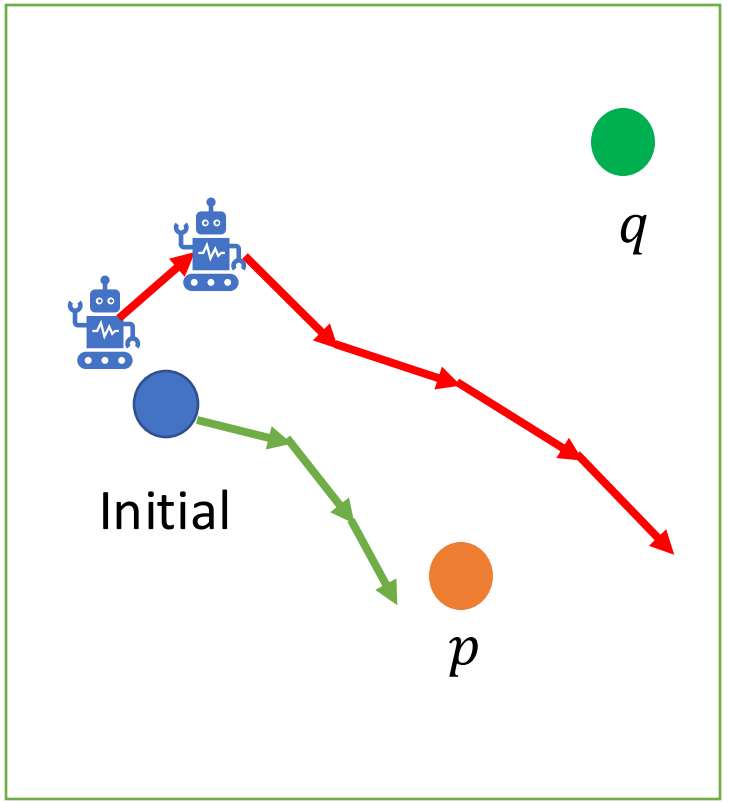
Action is the velocity vector
$\boldsymbol{a = (v_x, v_y) \in \mathbb{R}^2}$

Probability with which action a in state s changes state to $t \sim P(\cdot|s, a)$

$$\boldsymbol{t = (x', y') \sim \mathcal{N}\left((x + v_x, y + v_y), \Sigma\right)}$$

Unknown to the RL algorithm

Task: Reach p

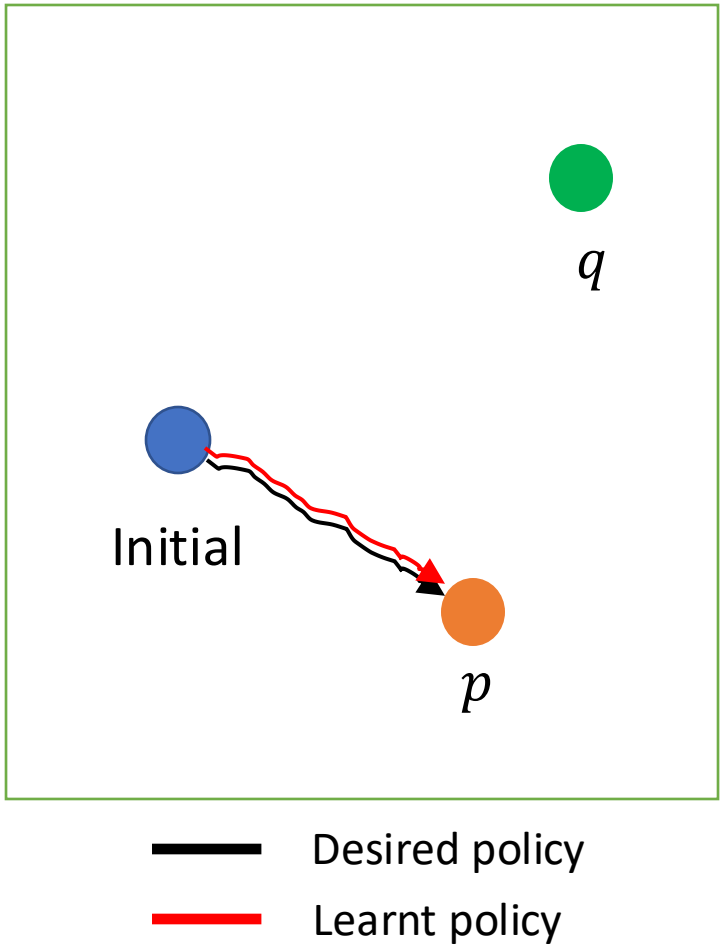


Task Specification as **Rewards**

```
# reach p

def reward_function(s):
    if  state.at(p) :
        return 1
    else :
        return 0
```

Optimal policy is one that eventually reaches p



Task Specification as **Rewards**

Challenging to express complex tasks

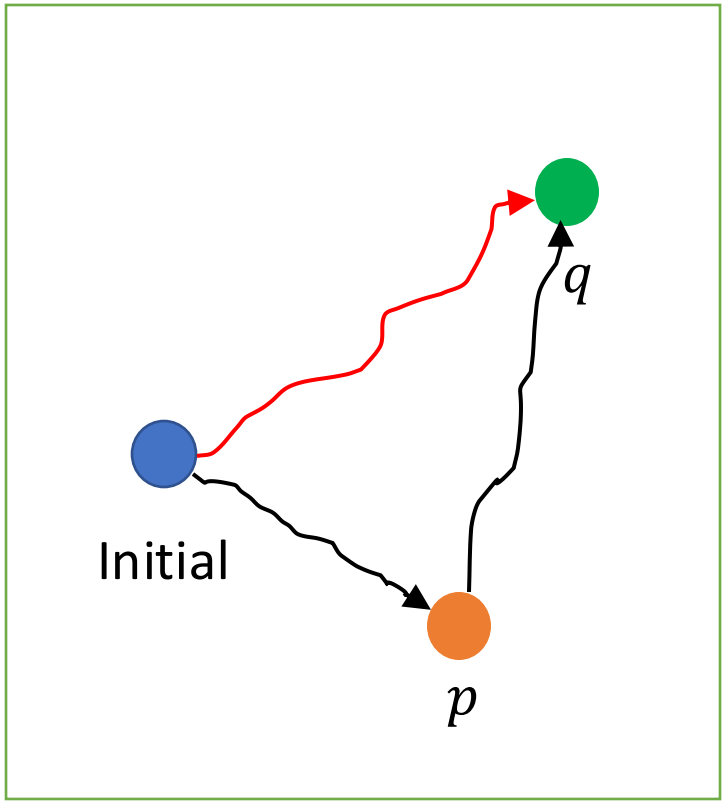
```
# reach p then reach q

def reward_function(s):
    if  state.at(p) :
        return 1
    else :
        return 0

    if  state.at(q) :
        return 1
    else :
        return 0
```

reach p then reach q

Use an **auxiliary variable** to keep track of whether p has been reached



Cumbersome to write reward functions

Reward Hacking

Reward misspecification is common and can lead erroneous behavior

Eg. **negative rewards** for crashing compensated by **positive rewards** for hitting targets!

Reward engineering is hard!

Why is writing reward functions hard in reinforcement learning?

Writing reward functions in reinforcement learning is challenging because the appropriate reward function is often not immediately obvious and requires a deep understanding of the problem and the desired behavior of the learning agent. Additionally, the reward function may need to be adjusted and fine-tuned over the course of training to effectively guide the learning process. Furthermore, the reward function must be designed to properly balance short-term rewards with long-term goals, which can be difficult to accurately predict. Overall, crafting an effective reward function in reinforcement learning requires a combination of domain knowledge,

The need to manually write rewards hinders **usability of RL!**

Reward Sparsity

- Rewards for long-horizon tasks are sparse
- Requires a lot of samples to "stumble into" a good policy

```
count = 0  # global variable

def get_rewards(s):
        if state.at(O) :
                return -10
        if count == 0 and state.at(S_1) :
                count = 1
        if count == 0 and state.at(S_2) :
                count = 1
        if count == 1 and state.at(S_3) :
```

"Correct rewards" not sufficient
Non-sparse rewards necessary

Task Specification as **Temporal Logics**

- Describe properties over executions
  - Logical Constraints: Predicates/propositions and Boolean operators
  - Temporal Constraints: Always, Eventually, Next (;), …
  - **Examples:** Linear Temporal Logic [Pnueli. FOCS 1977. **Turing Award 1996**], Linear Temporal Logic over Finite Traces [DeGiacomo and Vardi. IJCAI 2013], Signal Temporal Logic [Donzé. RV 2013], and many more
- Easy natural-language like syntax

"reach p"

Eventually (p)

"reach p then reach q"

Eventually (p) ; Eventually (q)

Specification-Guided RL

[Camacho et. al. SOCS 17] [De Giacomo et. al. ICAPS 19] [Ng, Harada, Russel. ICML 99] [Camacho et. al. IJCAI 19] [Bozkurt et. al. 2019] [Hahn et. al. TACAS 19, ATVA 20] [Balakrishnan and Deshmukh. IROS 19] [Li, Vasile, Belta IROS 17] [Jothimurugan, Bastani, Alur. NeurIPS 2019] [Kalagarla, Jain, and Nuzzo. CDC 2021] [Jothimurugan,Bansal, Bastani, Alur. NeurIPS 2021] [Jothimurugan,Bansal, Bastani, Alur. CAV 2022]

- Reinforcement Learning (from Rewards)
- Reinforcement Learning from Temporal Specifications
- Practical Algorithm I: Automata-based approach
- Practical Algorithm II: Compositional approach

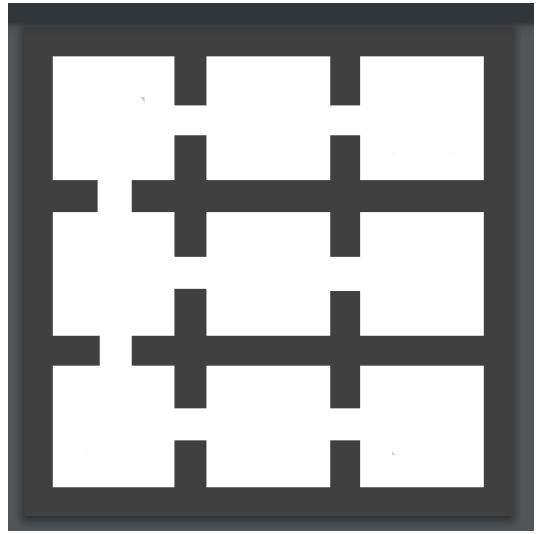
RL from Rewards

Markov Decision Process (MDP)

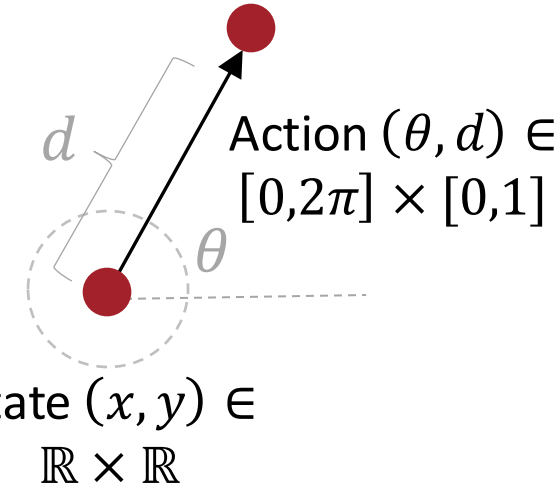
Environment is an MDP $M = (S, A, P, \eta)$

- S is the set of states
- A is the set of actions
- $P : S \times A \times S \rightarrow [0,1]$ is transition probability
  $P(s, a, s')$ is probability of transitioning from s to s' on a
- $\eta : S \rightarrow [0,1]$ is the initial state distribution

Policy $\Pi : (S \times A)^* S \rightarrow D(A)$



9-rooms Environment



RL from Rewards

Given, an MDP

a Reward function $R : S \times A \rightarrow \mathbb{R}$, and

a Discount factor $0 < \gamma < 1$

Assuming transition probabilities P are unknown

Generate, policy $\Pi : (S \times A)^* S \rightarrow D(A)$

s.t. it optimizes the

(expected) discounted-sum reward

Expected discounted-sum reward

Discounted-sum on a single trajectory

- Rewards along trajectory τ with rewards $r_0, r_1, \ldots$
- Discounted-sum is given by $\mathrm{DS}(\tau, \gamma) = \Sigma_{\{i=0\}}^{\infty} r_i \cdot \gamma^i$

Expected discounted-sum of a policy Π

- $\tau \sim \Pi$ means trajectory τ sampled from the MDP using policy Π
  - Start in the initial state s_0
  - Choose action a_0 using policy Π
  - Choose next state s_1 using MDP transition probability $P(\cdot\,|s_0, a_0)$

  } Repeat

- Expected discounted-sum is $\mathbb{E}_{\boldsymbol{\tau \sim \Pi}}\,[\mathrm{DS}(\tau, \gamma)]$

If transition probabilities were known ….

Given, an MDP
a Reward function $R : S \times A \rightarrow \mathbb{R}$, and
a Discount factor $0 < \gamma < 1$

} Optimization Problem

Generate, policy $\Pi : (S \times A)^* S \rightarrow D(A)$
s.t. it optimizes the (expected) discounted-sum reward

- Memoryless deterministic optimal policy $\Pi : S \rightarrow A$ [Shapley 1953]
- Solve via Bellman equations [Bellman 1957] [Howard 1960]

$$V(s) = \max_{\{a \in A\}} R(s,a) + \gamma \cdot \Sigma_{t \in S} \boldsymbol{P}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{t}) \cdot V(t)$$

- Then, expected discounted-sum reward is $V(s_0)$

Solving Bellman Equations

All methods require the transition probabilities

Linear Programming

- Exact method but slow and memory-heavy

Value iteration

- Approximate method but fast convergence

Policy iteration

- Typically, faster convergence than value iteration

Value iteration, Q-values

$V(s) = \max_{\{a \in A\}} Q(s,a)$ where $Q(s,a) = R(s,a) + \gamma \cdot \Sigma_{t \in S} P(s,a,t) \cdot V(t)$

Initialize $V(s)$ arbitrary values

Repeat until $V(s)$ converge:

For all states s :

For all actions a :

$Q(s,a) \leftarrow R(s,a) + \gamma \cdot \Sigma_{t \in S} P(s,a,t) \cdot V(t)$

$V(s) \leftarrow \max_{\{a \in A\}} Q(s,a)$

Policy extraction: $\Pi(s) = argmax_{\{a \in A\}} Q(s,a)$

In the absence of transition probabilities

Simulator of MDP is available (current state is known)

Reset to an initial state

```
reset():
  self.state ~ η
  return self.state
```

Sample actions from the current state

```
step(a):
  self.state ~ P(·|self.state, a)
  return self.state
```

Sufficient to sample multiple trajectories in the MDP

Q-learning

[Watkins and Dayan. ML 1992]

Insight: Apply value iteration without

Initialize the Q values arbitrarily

Repeat until convergence:

Choose some action a in current state s

Sample next state s' and obse

Update the Q value for state s and a

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha$$

Best action based on current Q:

$a = \text{argmax}_{a'} Q(s,a')$

Also need to **explore**:

Choose a with **probability ε** and a **uniformly random action** with **probability $1-\varepsilon$**

Algorithms and Guarantees

Algorithms: Sampling-based

- Model-free: Do not learn transition probabilities explicitly
- Model-based: Explicitly learn the transition probabilities, solve Bellman equations on the estimated MDP

Theoretical guarantees (MDP has finitely many states)

Asymptotic guarantees [Watkins and Dayan. ML 1992]	Converges to optimal policy if sampled infinitely-many times
PAC guarantees [Kearns and Singh. ML 2002]	Converges to near-optimal policy within finite number of samples times (with high confidence)

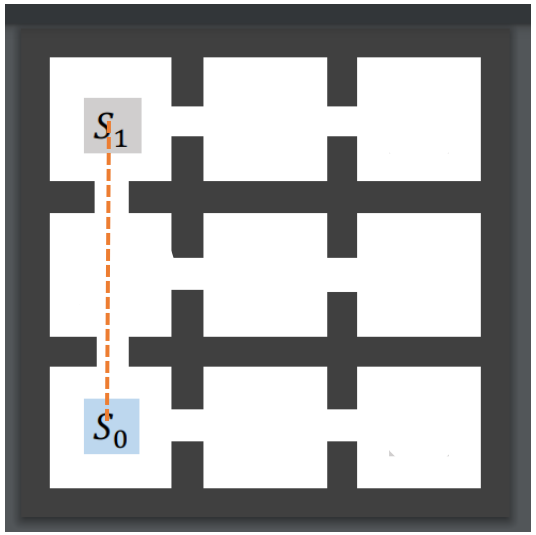
RL from Temporal Specifications

RL from Temporal Specifications

Given, a MDP and
a Temporal Specification
a Labelling function $S \rightarrow Prop$

Assuming transition probabilities P are unknown

Generate,
policy $\Pi : (S \times A)^* S \rightarrow D(A)$
s.t. it optimizes
(expected) probability of satisfaction



9-rooms Environment

Spec: Eventually S_1

Optimal Policy
As shown in illustration

Expected probability of satisfaction

- *Value* of a trajectory τ w.r.t. temporal logic formula φ
  - 1 if $\tau \vDash \varphi$, else 0

Expected probability of satisfaction of a policy Π

- $\tau \sim \Pi$ means trajectory τ sampled from the MDP using policy Π
  - Start in the initial state s_0
  - Choose action a_0 using policy Π
  - Choose next state s_1 using MDP transition probability $P(\cdot \,|s_0, a_0)$

  } Repeat

- Expected probability of satisfaction is $\mathbb{E}_{\boldsymbol{\tau \sim \Pi}}[\tau \vDash \varphi]$

If transition probabilities were known…

Given, a MDP and

a Temporal Specification

a Labelling function $S \rightarrow Prop$

} Probabilistic Model Checking

Generate, policy $\Pi : (S \times A)^* S \rightarrow D(A)$

s.t. it optimizes (expected) probability of satisfaction

- Obtain equivalent automata of LTL formula
- Product of MDP with automata
- Solve automata objective on product

In the absence of transition probabilities…

Simulator of MDP is available (current state is known)

Reset to an initial state

```
reset():
  self.state ~ η
  return self.state
```

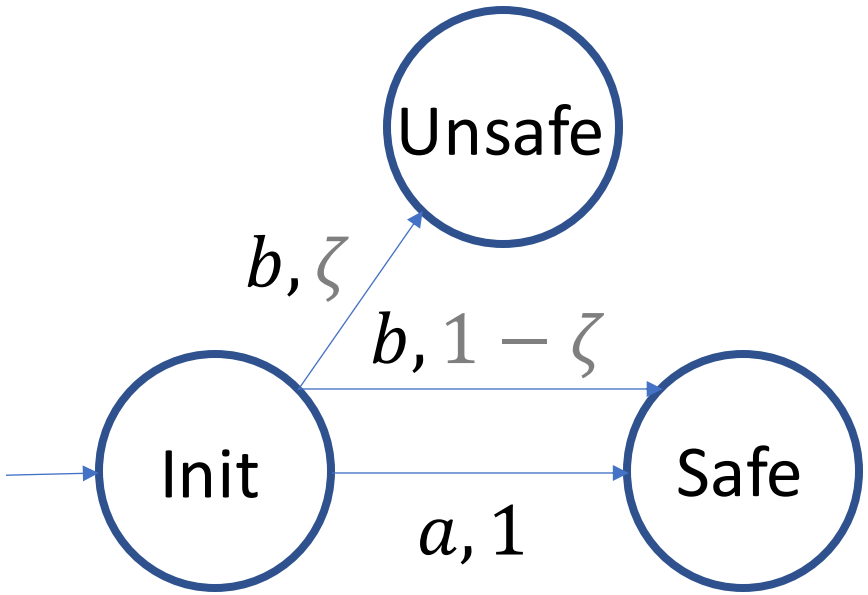
Sample actions from the current state

```
step(a):
  self.state ~ P(·|self.state, a)
  return self.state
```

Sufficient to sample multiple trajectories in the MDP

Sampling-Based Approach

Spec: Always (Not Unsafe)



Optimal policy

- Take action a from Init

Can we learn optimal policy in finitely-many samples?

- From every state, sample every action K times
- Suppose, K times sample b from Init
- If $K \ll \frac{1}{\zeta}$, may never discover edge to Unsafe
- If $K \ll \frac{1}{\zeta}$, may learn to take b from Init

Probably Approximately Correct (PAC) Learning

An **RL algorithm** is an iterative process where for each iteration $n = 1, 2, \ldots$

- Calls `step(a)` or calls `reset()`
- Outputs current best policy π_n

PAC if **near-optimal policies** with **high confidence** after finitely-many samples

Given ε, δ, an RL algorithm is **PAC** if with probability $> 1 - \delta$, **all policies after N steps are ε-optimal**

No PAC Theorem

Theorem [Alur, **Bansal**, Bastani, Jothimurugan. Principles of System Design, 2022]
There **can not exist** a PAC algorithm for RL from temporal specifications

Theorem [Yang, Carbin, Littman. IJCAI 2023]
PAC algorithm for RL from temporal specifications exists iff the specification is **finitary**.

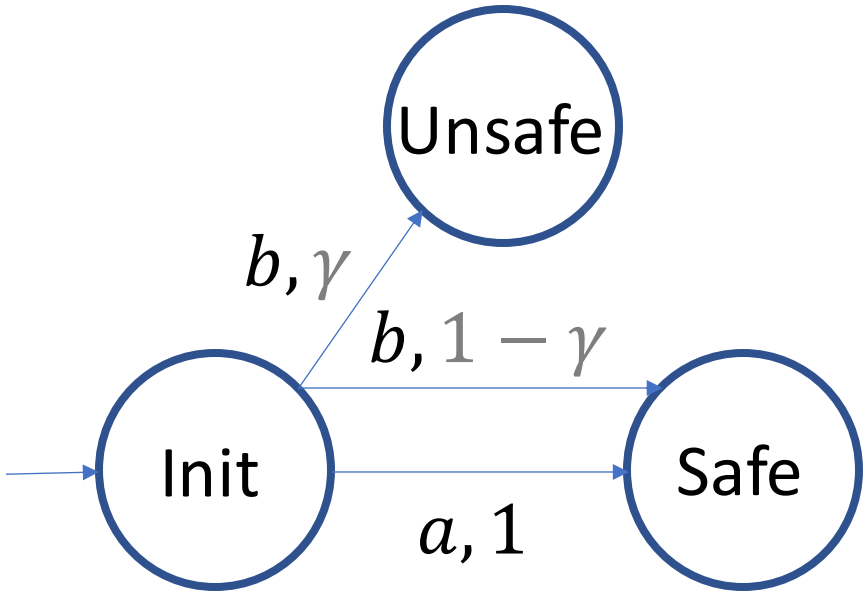
Countering PAC impossibility

PAC under assumptions

- Assumptions on the MDP
  - Minimum transition probability [Ashok et. al. CAV 2019]
  - Mixing time [Perez et. al. AAAI 2023]
  - Topology of the MDP [Fu and Topcu. RSS 2014] [Daca et. al. TACAS 2016]

PAC Possibility: If $p_{\{min\}}$ were known

Spec: Always (Not Unsafe)



Optimal policy

- Take action a from Init

Can we learn optimal policy in finitely-many samples?

- Least non-zero transition probability $p_{\{min\}}$
- From every state,
  sample every action $O(\frac{1}{p_{\{min\}}})$ times
- With high-probability,
  good estimate of transition probabilities

Countering PAC impossibility

PAC under assumptions

- Assumptions on the MDP
  - Minimum transition probability [Ashok et. al. CAV 2019]
  - Mixing time [Perez et. al. AAAI 2023]
  - Topology of the MDP [Fu and Topcu. RSS 2014] [Daca et. al. TACAS 2016]

- Assumptions on length of executions
  - Fixed trajectory length [Carbin, Littman, Yang. IJCAI 2023]
  - Expected trajectory length [Svoboda, **Bansal**, Chatterjee. ICML 2024]

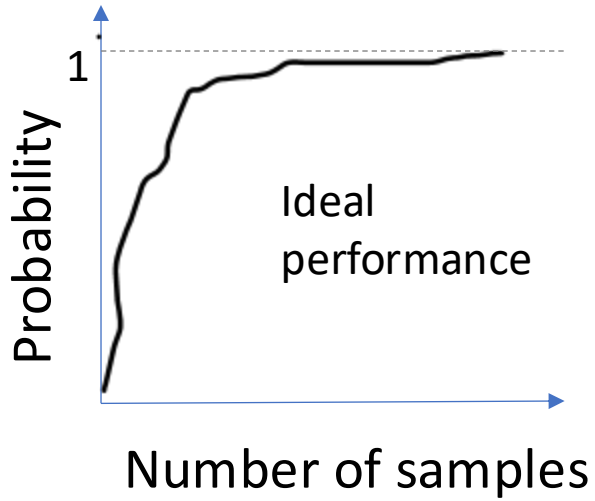
Open Problems

- Quantitative semantics for logics for PAC possibility
  - Semantics shall retain intuitiveness of temporal logic yet obtain guarantees
- When possible, tighter theoretical analysis
  - Large gap between theoretical upper bounds and practical performance

Practical Algorithms

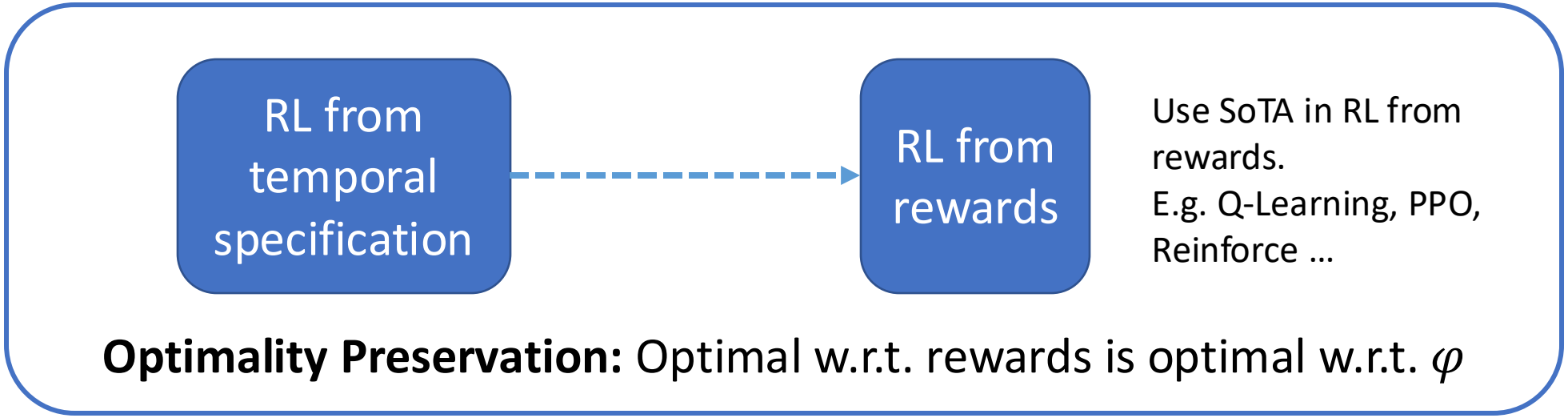
Practical concerns

- Model-free algorithm
- Fast convergence to optimal policy



- Scalability to complex and long-horizon tasks

Algorithmic Approaches



- Automata-based approaches
- Compositional approaches

Practical Algorithms I: Automata-Based

Linear-Temporal Logic (LTL)

[Pnueli. FOCS 1977. Turing Award 1996],

Specification language

- Temporal logic over discrete time

Syntax

- Boolean variables and operators
- Temporal operators: Always, Eventually, Next, …

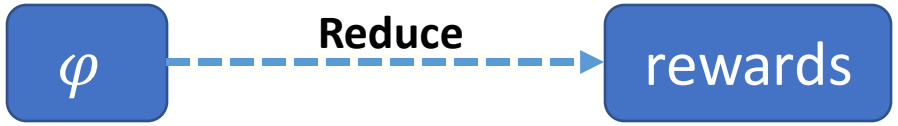
Example: Always (Request → (Grant ∨ Next Grant))

- "Every request is granted within the two steps"

Request	T	F	T	F	T	T
Grant	T	F	F	T	F	T

.....

Reduction I: Specification Translation



Optimality Preservation: For all MDP M, optimal w.r.t. rewards is optimal w.r.t. φ

No **optimality preserving** specification translation exists for LTL

[1] D. Sadigh, et al. "A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications learning." CDC 2014

[2]. Icarte et. al. "Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning". JAIR 2022

LTL vs Rewards

Suppose, specification is Eventually P

- LTL captures long-horizon

LTL vs Rewards

Suppose, specification is Eventually P

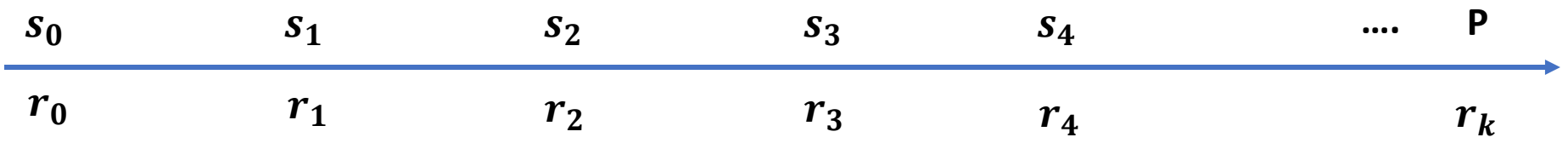
- LTL captures long-horizon

LTL vs Rewards

Suppose, specification is Eventually P

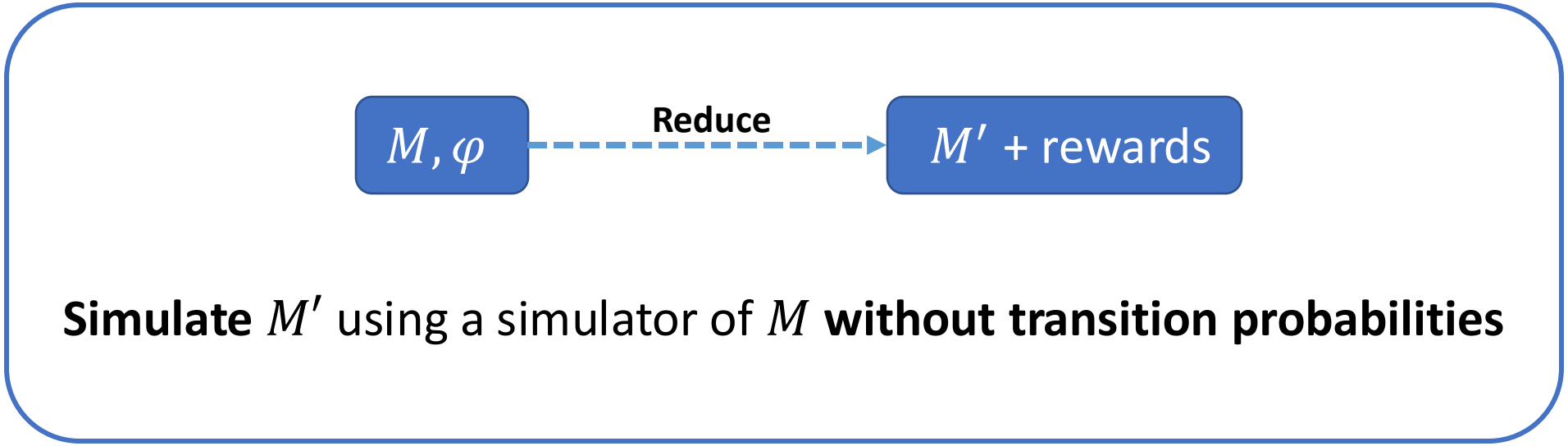
- LTL captures long-horizon

- Rewards capture short-horizon



$$\text{Reward} = \Sigma_{\{i=0\}}^{\infty} r_i \cdot \gamma^i$$

Reduction II: Sampling-Based Reduction

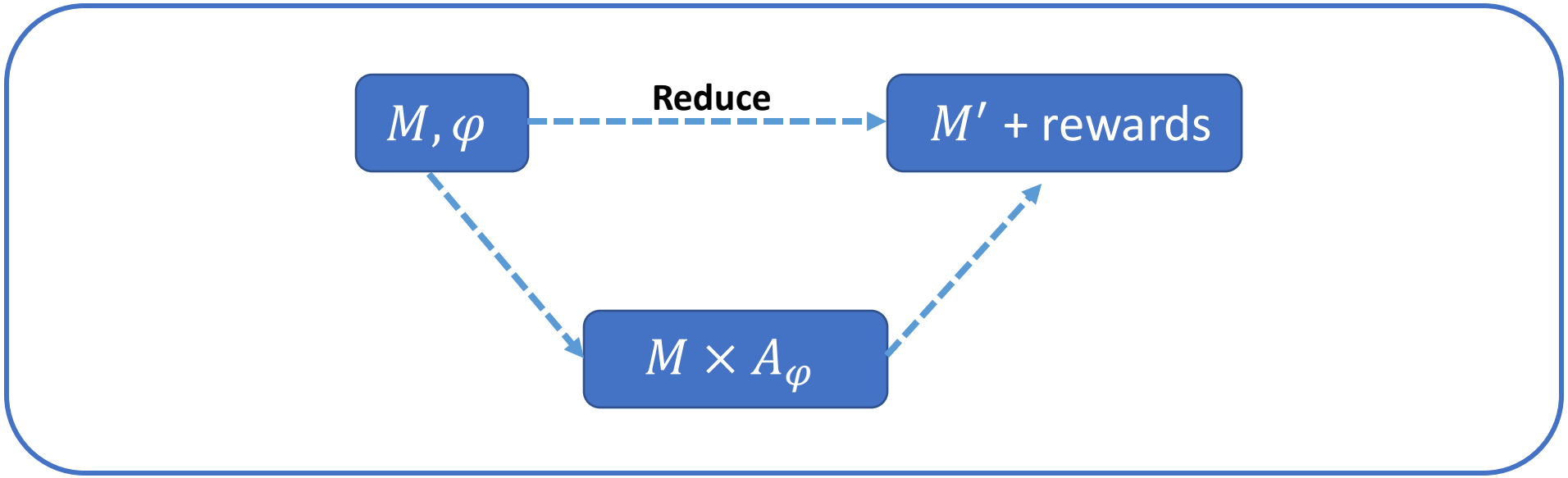


Simulate M' using a simulator of M **without transition probabilities**

Optimality preserving sampling-based reduction exist but non-constructive!

[1] Hahn, Ernst Moritz, et al. "Good-for-MDPs automata for probabilistic analysis and reinforcement learning."
[2]. Hahn, Ernst Moritz, el al "Faithful and Effective Reward Schemes for Model-Free Reinforcement Learning of Omega-Regular Objectives"

Reduction II: Sampling-Based Reduction



RL Learnable

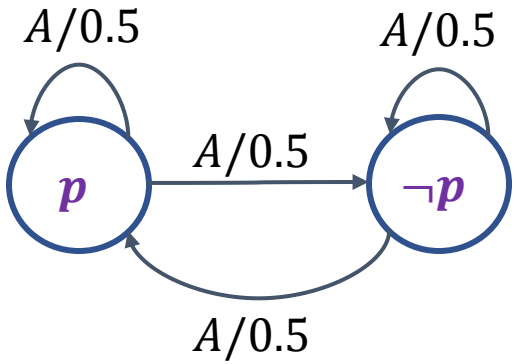
- Simulator of $M \times A_\varphi$ using a simulator of M without transition probabilities

Optimality preservation

- For optimal policies π and $\pi^\times$, $\Pr[\pi \vDash M, \varphi] = \Pr[\pi^\times \vDash M \times A_\varphi]$

Non-Deterministic Buchi (NBA) Automata

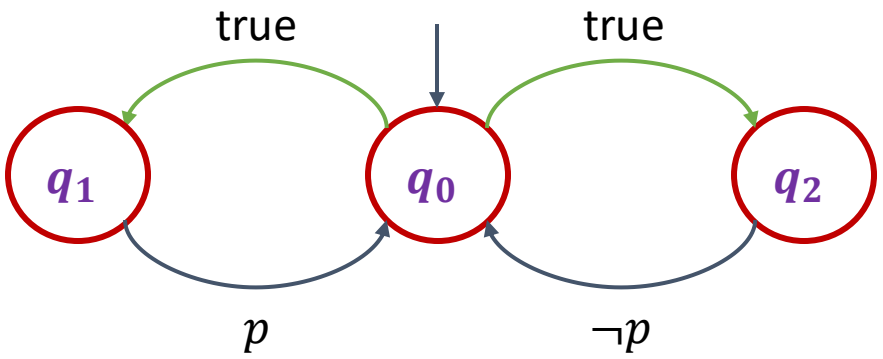
$$\Pr[\pi \vDash M, \varphi] \geq \Pr[\pi^{\times} \vDash M \times A_{\varphi}]$$



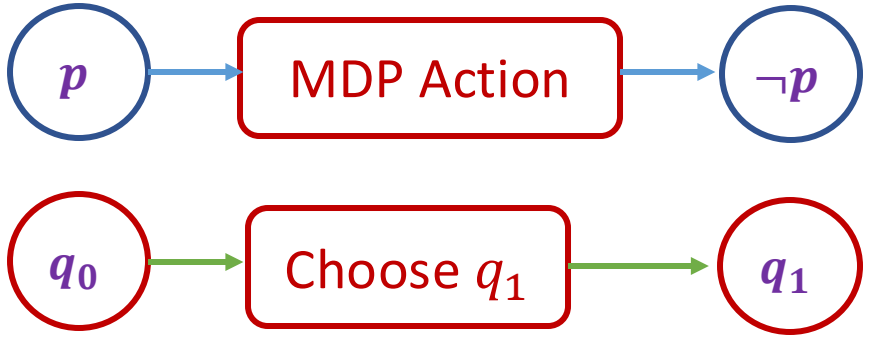
MDP in which p holds with probability **half** at **every step**

$\varphi = (\text{true})^{\omega}$

$\Pr[\pi \vDash M, \varphi] = 1$



NBA accepting $\varphi = (\text{true})^{\omega}$



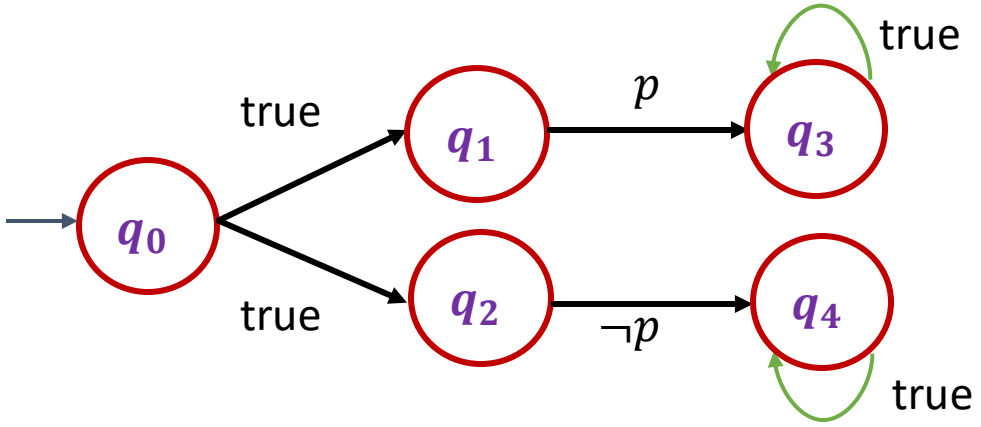
Occurs with **probability 0.5**

Can no longer reach accepting transitions. Buchi condition violated.

Limit-Deterministic Buchi (LDBA) Automata

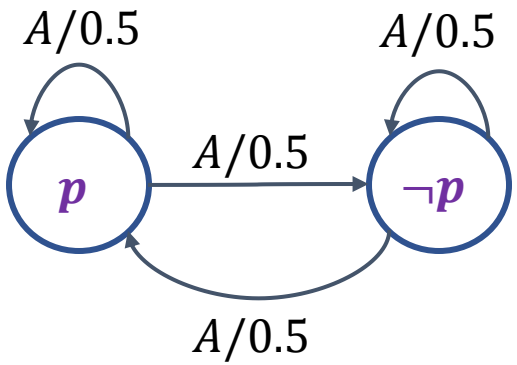
LDBA is an NBA $(Q_i \cup Q_f, \Sigma, q_0, \delta, F)$ s.t.

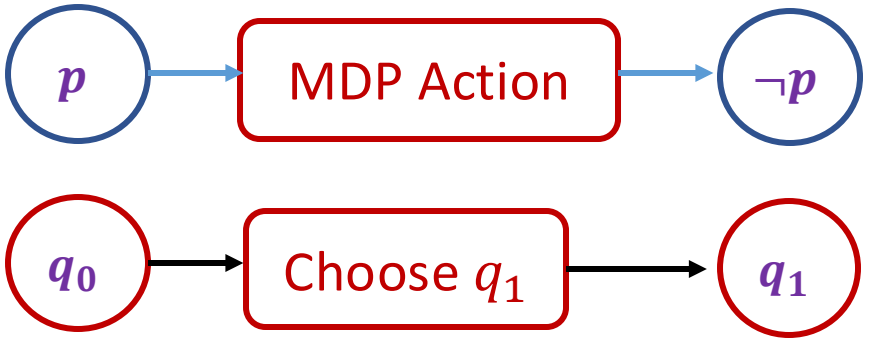
- $Q_i \cap Q_f = \emptyset$ and Q_f is deterministic
- $F \subseteq Q_f \times \Sigma \times Q_f$
- $\delta(q, \sigma) \subseteq Q_f$ for all $q \in Q_f$



LDBA accepting $\varphi = (\text{true})^{\omega}$

$\Pr[\pi \vDash M, \varphi] \geq \Pr[\pi^{\times} \vDash M \times A_{\varphi}]$





Occurs with **probability 0.5**

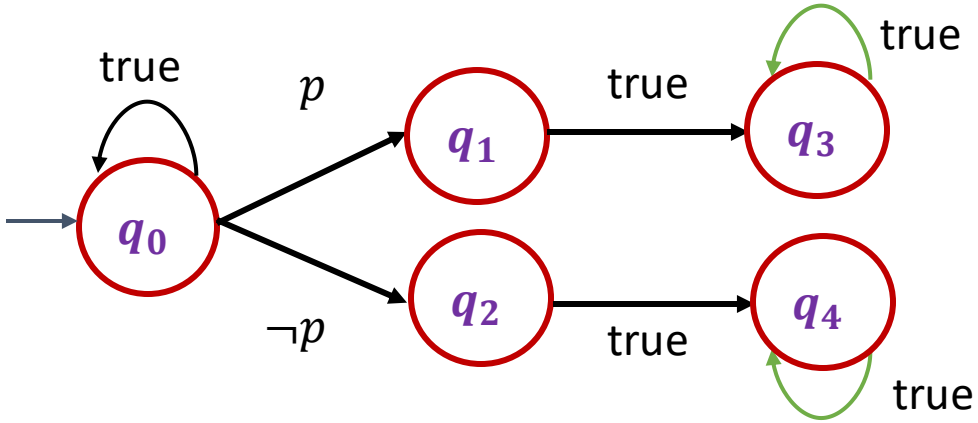
Can no longer reach accepting transitions. Buchi condition violated.

Good-for-MDP LDBA

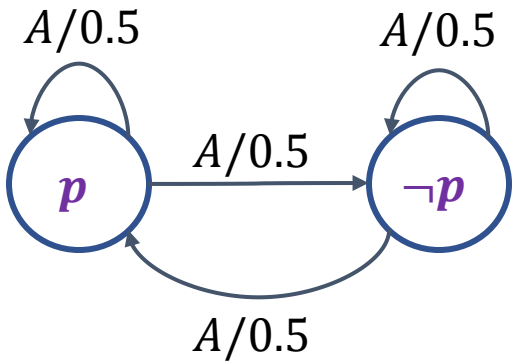
LDBA is Good-for-MDP if

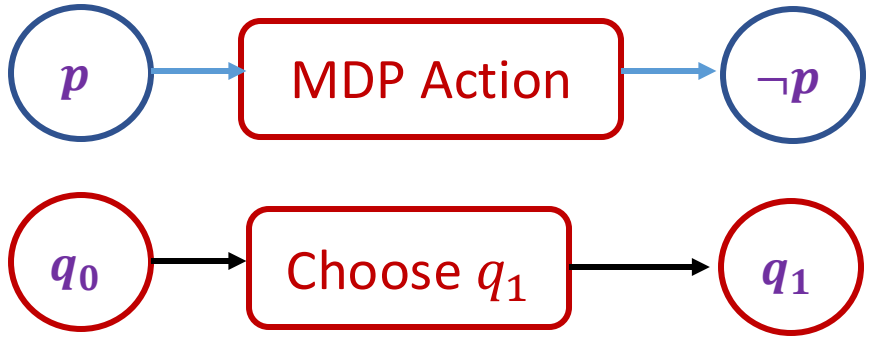
$$\Pr[\pi \vDash M, \varphi] = \Pr[\pi^{\times} \vDash M \times A_{\varphi}]$$

LTL to Good-for-MDP LDBA translation exists



LDBA accepting **all sequences**





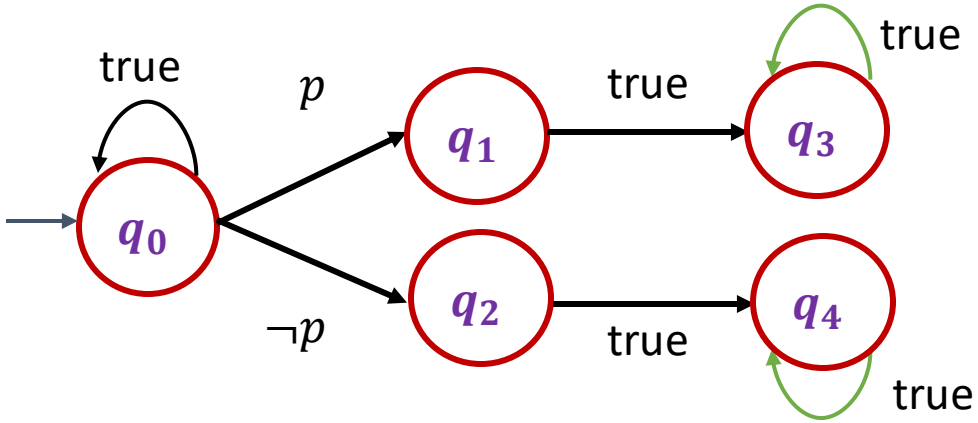
Will always visit accepting transition since $\neg p$ can advance product from q_1

Good-for-MDP LDBA

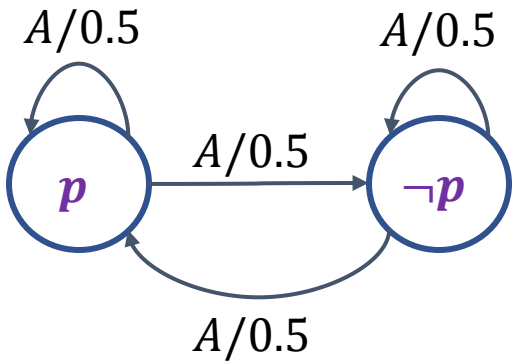
LDBA is Good-for-MDP if

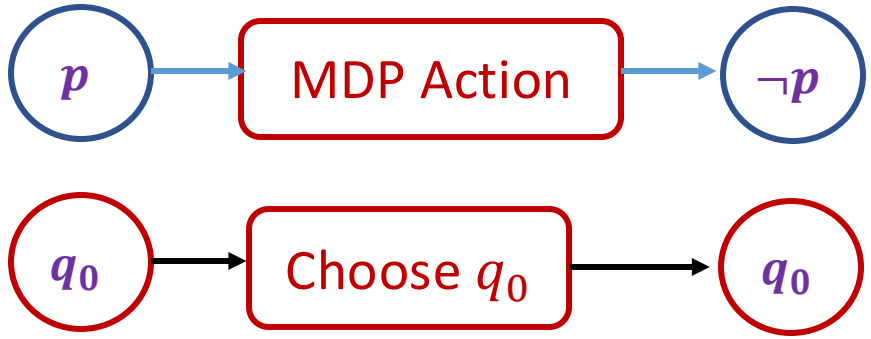
$$\Pr[\pi \vDash M, \varphi] = \Pr[\pi^{\times} \vDash M \times A_{\varphi}]$$

LTL to Good-for-MDP LDBA translation exists



LDBA accepting **all sequences**



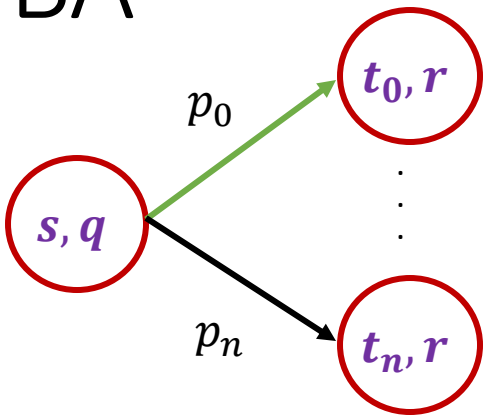


Can visit accepting transition since $\neg p$ can advance product from q_0

RL Learnability of Good-for-MDP LDBA

Reachability is more learnable

- Buchi w.p. 1 iff Reachability w.p. 1
- Exists ζ^* s.t. for all $\zeta \in [\zeta^*, 1)$, reachability optimal is Buchi optimal

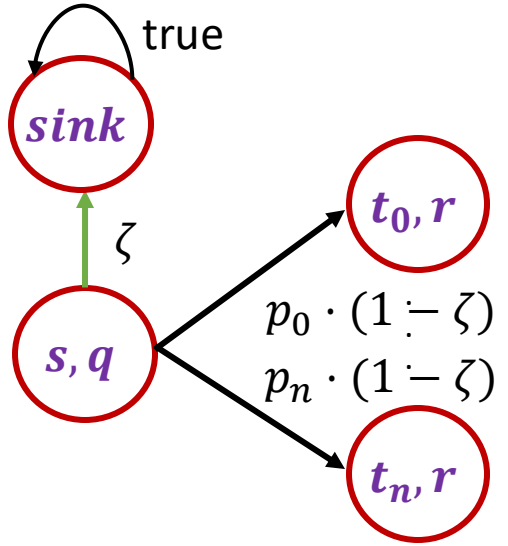


Buchi in product

Simulate-able product

```
step(a):
  // self.current = (s,q)
  // if (q,L(s),δ(q,L(s))) is accepting:
     // toss a biased coin with Pr(H) = ζ
     // if H, return self.state = sink
     // else, return self.state = (t,δ(q,L(s)))
                                  where t ~ P(·|s,a)
```

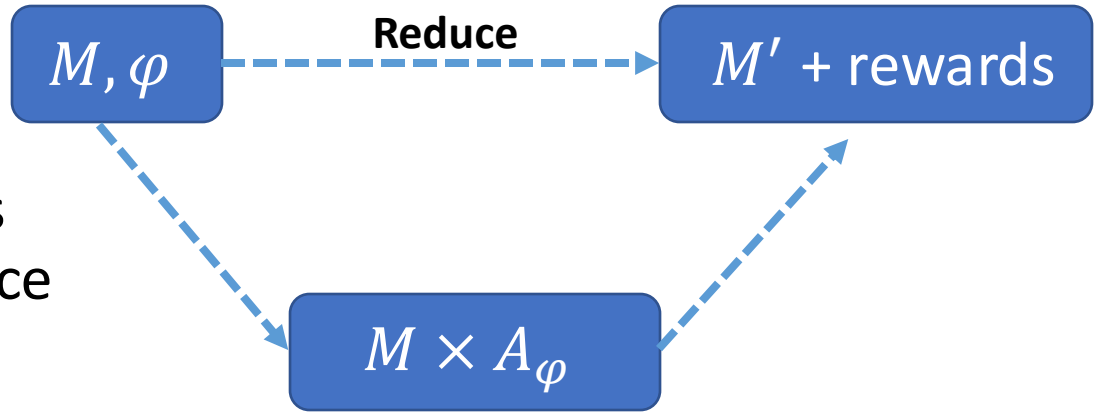
For $\zeta > 0$:



Reachability in product

Drawbacks

- Non-constructive
  - Eg. ζ parameter
  - Further reduction to M'+ rewards requires non-constructive parameters in the absence of transition probabilities

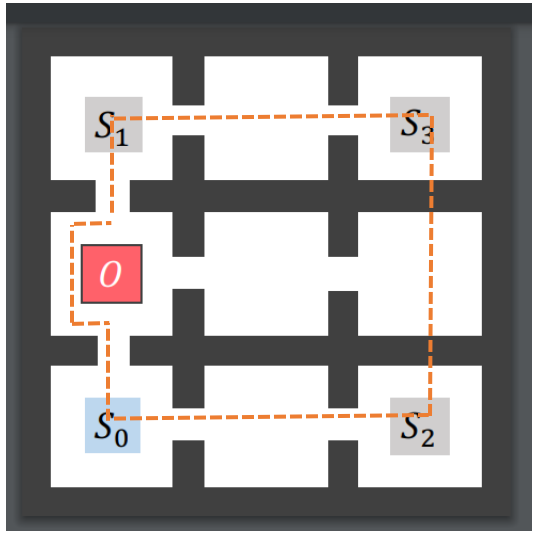


- Rewards are sparse
  - Only when visiting an accepting transition
  - Poor algorithmic performance
  - Poor scalability to long-horizon specifications

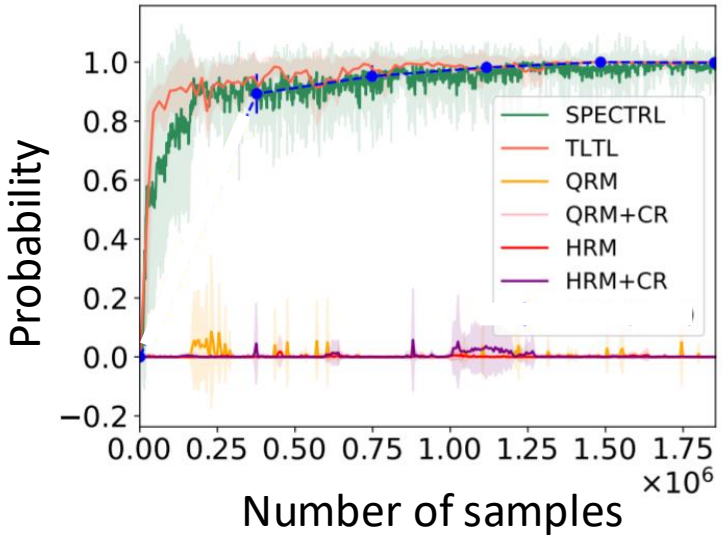
Practical Algorithms II: Compositional Approach

Myopia in RL

RL is good at short-horizon task but poor at long-horizon tasks

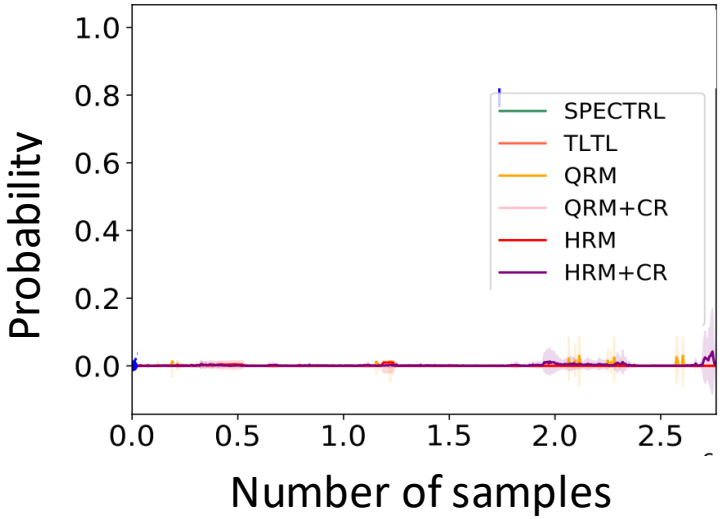


Visit S_1 or S_2
while avoiding O



Learns to visit S_2 via obstacle-free path

Visit S_1 or S_2 then visit S_3
while avoiding O



Futile to learn to visit S_2
Better to learn to visit S_1

SpectRL temporal logic [Jothimurugan, Bastani, Alur. NeurIPS 19]

$$\varphi := \text{eventually } b \mid \varphi \text{ always } b \mid \varphi \ ; \varphi \mid \varphi \text{ or } \varphi$$

SpectRL temporal logic [Jothimurugan, Bastani, Alur. NeurIPS 19]

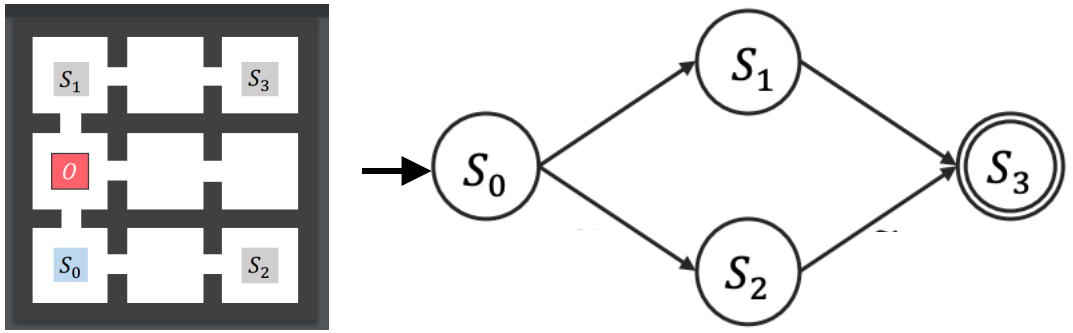
$$\varphi := \text{eventually } b \mid \varphi \text{ always } b \mid \varphi \,;\, \varphi \mid \varphi \text{ or } \varphi$$

Theorem

SpectRL specifications generate DAG abstractions of environment s.t.

a) Every edge is a subtask
b) Every path from initial to final satisfies the specification

"Visit S_1 or S_2 then S_3 while avoiding O"



DiRL: Compositional RL from Temporal Logic

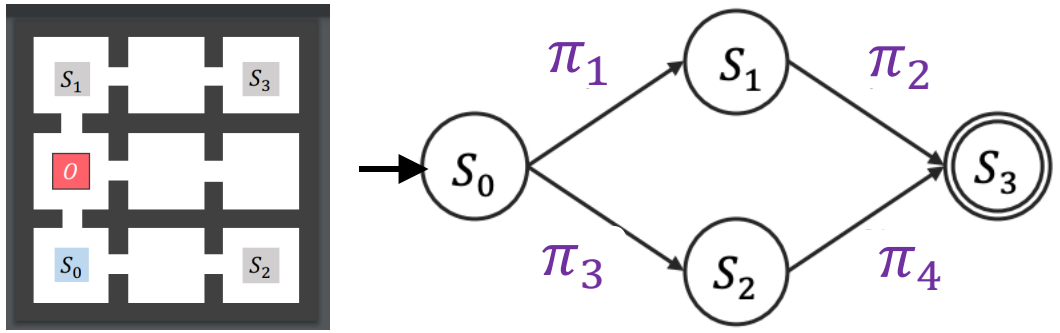
[Jothimurugan, **Bansal**, Bastani, Alur. NeurIPS 2021]

Decomposition phase

"Every edge is a subtask"

Learn a policy for every edge

"Visit S_1 or S_2 then S_3 while avoiding O"



Composition phase

"Every path to final satisfies the specification"

Compute **best policy** to final state

- Best policy = Path with maximum probability
- Edge probability = Estimated success of sub-task policy

Policy $S_0 \rightarrow S_1 \rightarrow S_3$

π_1 until S_1 reached;
π_2 until S_3 reached

DiRL: Compositional RL from Temporal Logic

[Jothimurugan, **Bansal**, Bastani, Alur. NeurIPS 2021]

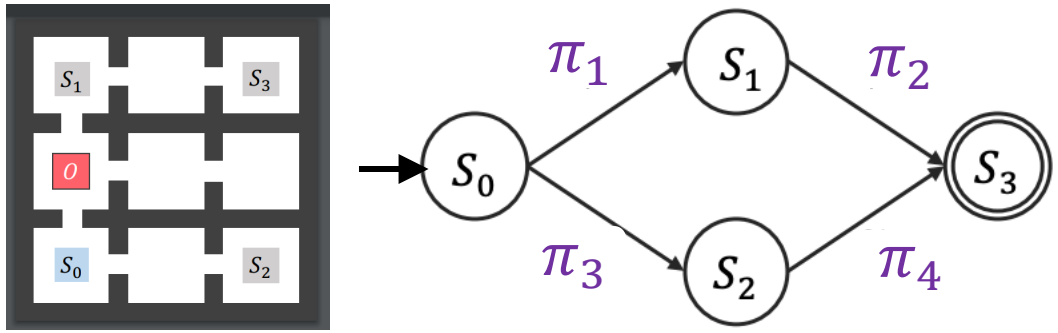
Decomposition phase

"Every edge is a subtask"

Learn a policy for every edge

- State distribution on DAG state?

"Visit S_1 or S_2 then S_3 while avoiding O"



Composition phase

"Every path to final satisfies the specification"

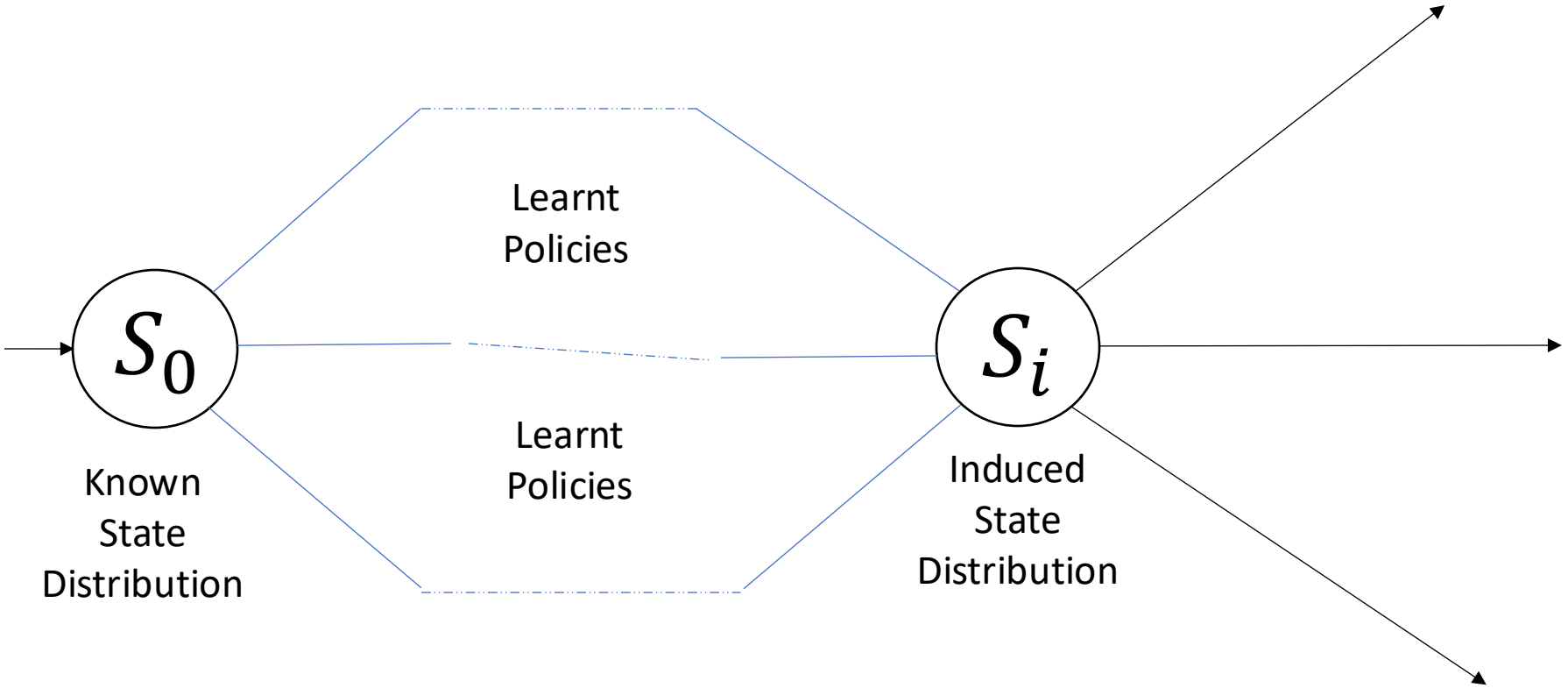
Compute **best policy** to final state

- Best policy = Path with maximum probability
- Edge probability = Estimated success of sub-task policy

Policy $S_0 \rightarrow S_1 \rightarrow S_3$

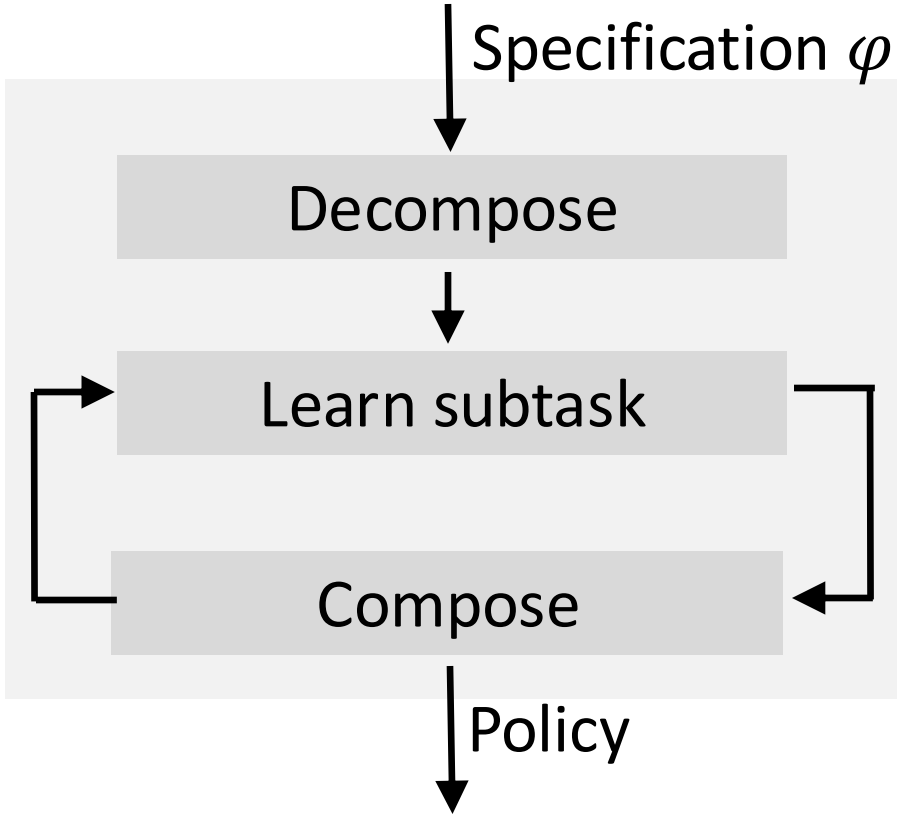
π_1 until S_1 reached;
π_2 until S_3 reached

Learning State Distributions



Choose order of learning states and edges

DiRL: Compositional RL from Temporal Logic



Theorem

DiRL is $O(|E| \cdot \log(|V|))$

+

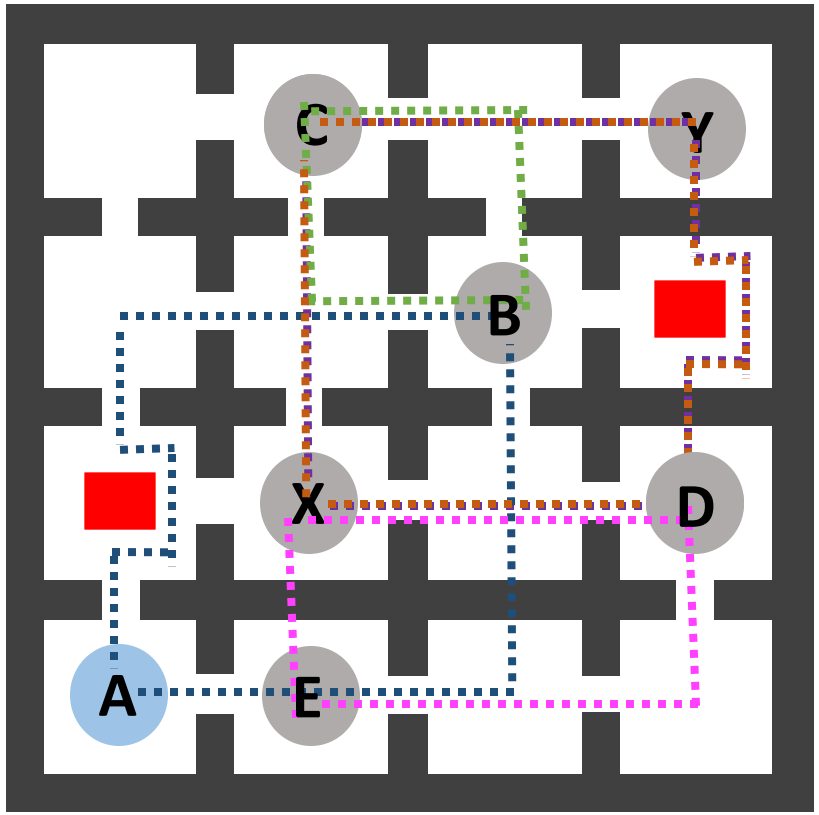
Number of RL calls is $O(|E|)$

Theorem

Size of DAG

$|V| = O(|\varphi|)$ and $|E| = O(|\varphi^2|)$

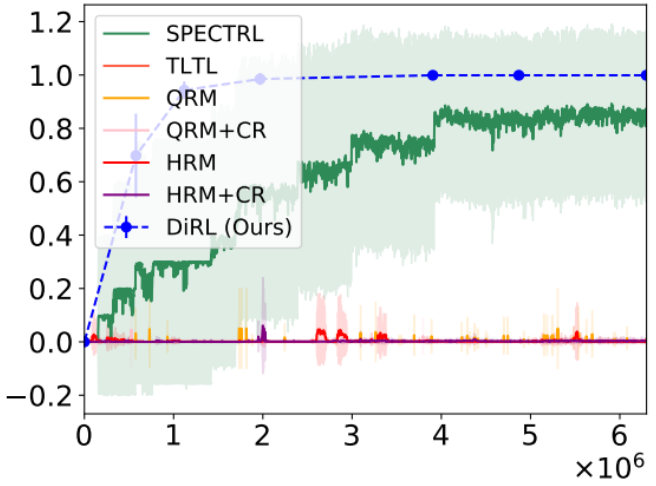
Scalability
In Specification



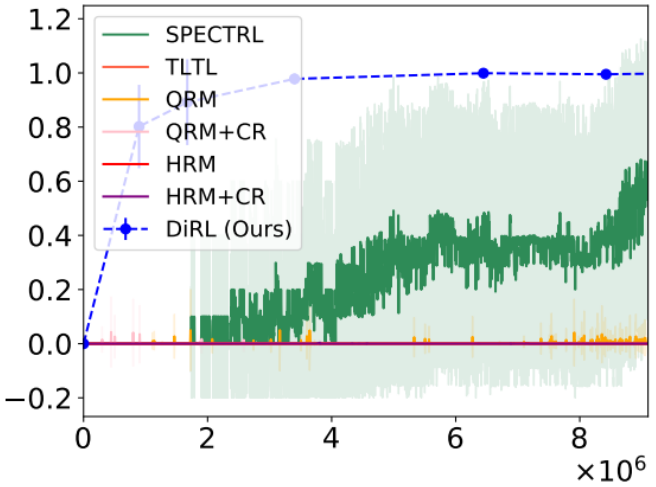
X to Y

From X,
Visit Corner-1 or Corner-2,
then Visit Y,
while avoiding Obstacles

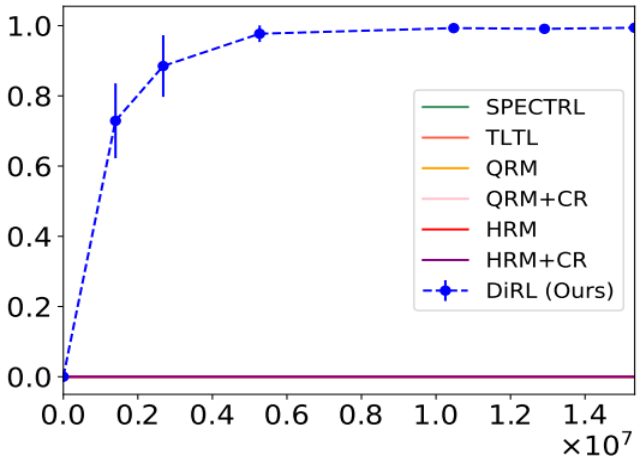
A to B



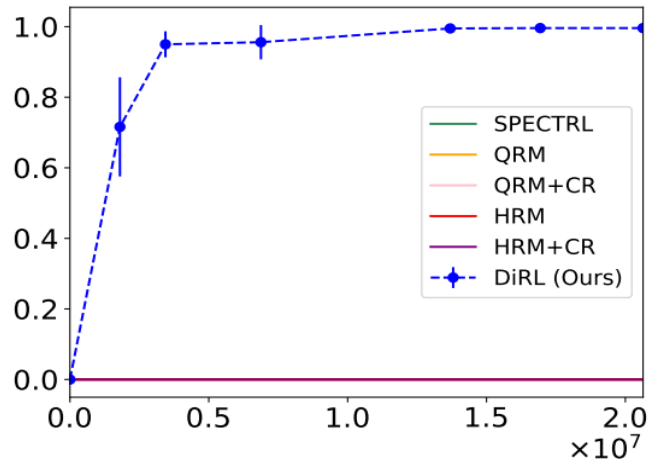
A to B **B to C**



A to B **B to C** **C to D**

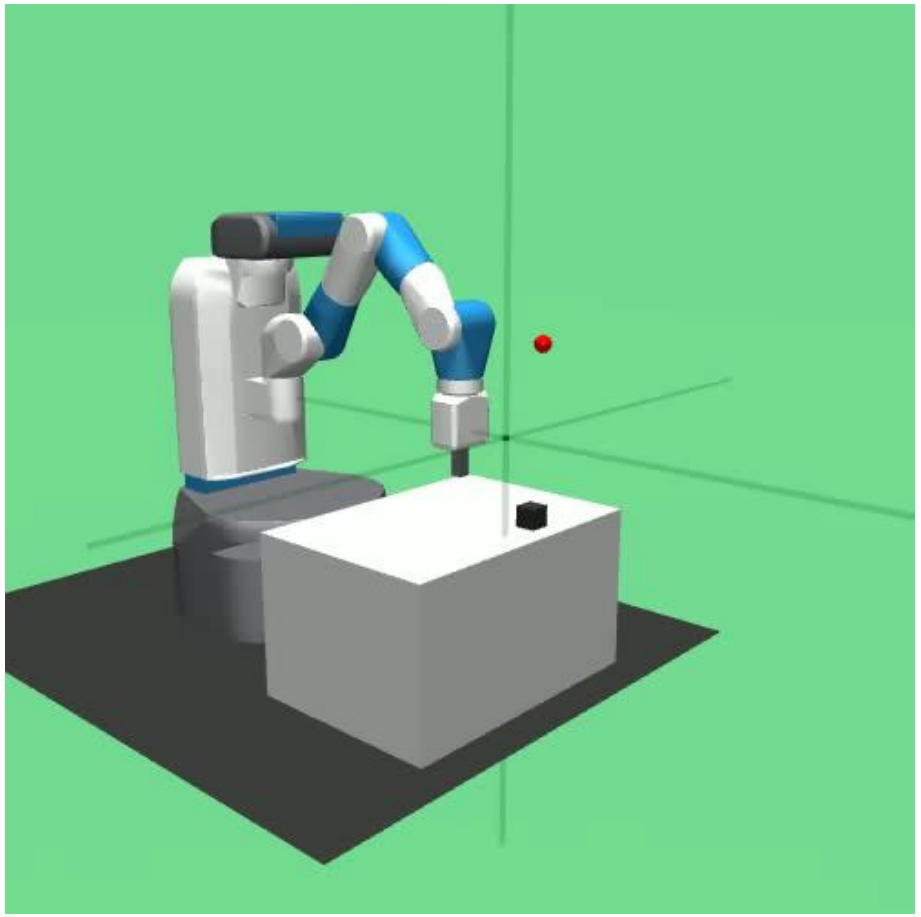


A to B **B to C** **C to D** **D to E**



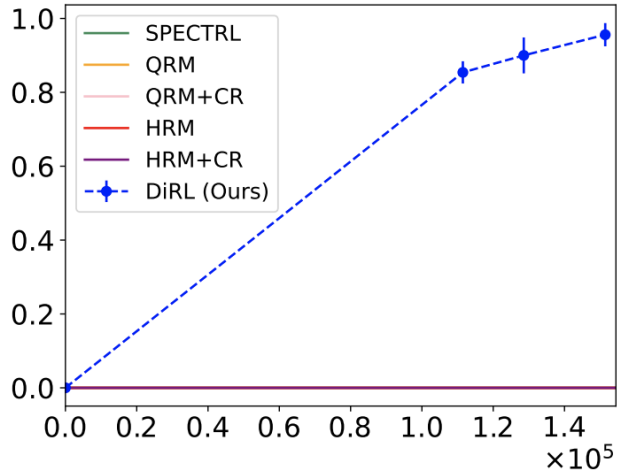
x-axis: Number of samples y-axis: Probability of satisfaction of spec.

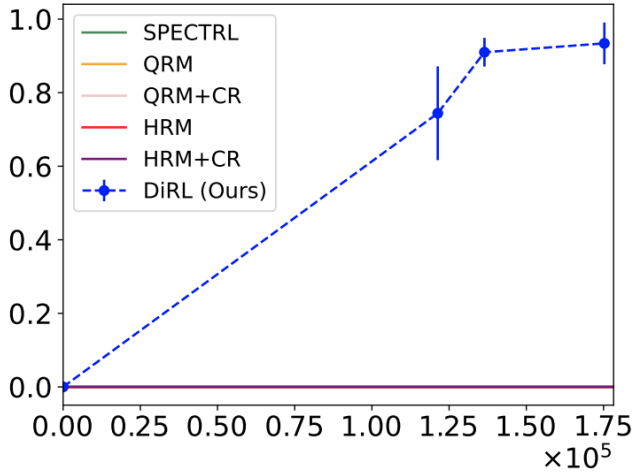
Scalability
In (dimensionality of) Environment



Simulation of Baxter Robotic Arm

OpenAI Gym



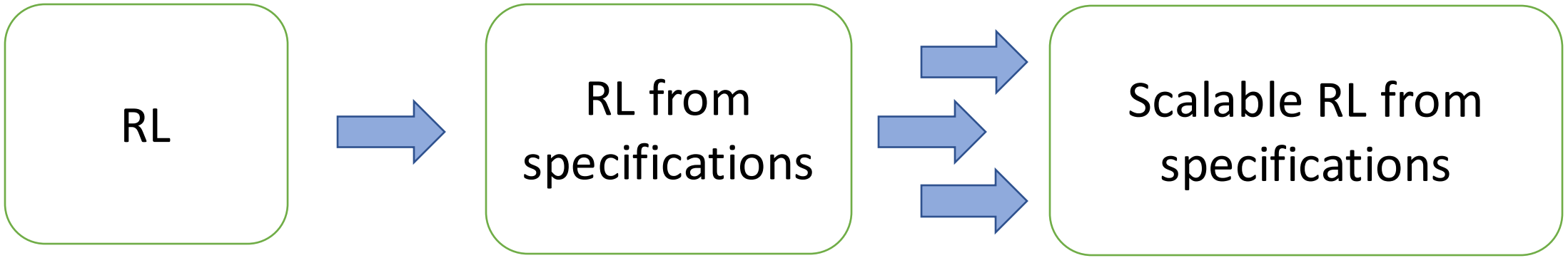


x-axis: Number of samples y-axis: Probability of satisfaction of spec.

Open Problems

- Lack of theoretical guarantees
  - Asymptotic guarantees in RL from specifications? [Ongoing work: Svoboda, Chatterjee, **Bansal**]
  - Q-learning style algorithm
- Unsatisfiability evaluation
- Specification-refinement to enable learning

SUMMARY



Using specifications for verifiability, generalizability, multi-agent, etc