Leveraging Formal Task Structure in Reinforcement Learning

Ameesh Shah, UC Berkeley

Simons Institute, 5.01

Reinforcement learning for autonomous behavior

Agent

Actions

Environment

Consequences:
Observations
Rewards

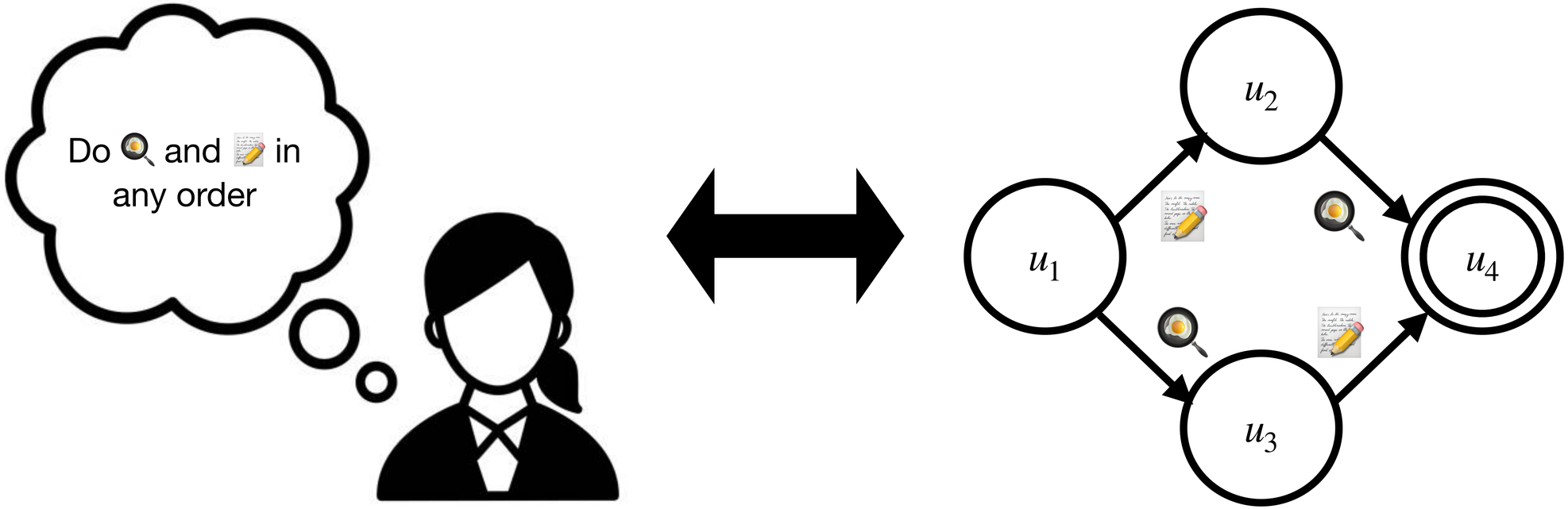
Reward design for reinforcement learning is hard

- The success of Reinforcement Learning (**RL**) relies on designing good ***reward functions***
  - Rewards should incentivize desirable behavior from our agents and discourage undesirable behavior
  - However, “reward engineering” is…
    - Tedious at best
    - Difficult to impossible at worst
- The key question: how do we represent desirable behavior?

A classic example of a reward function leading to undesirable behavior

How do we represent desirable behavior?

- **Formal methods** has been asking (and answering) this question in a number of contexts
- To verify computational systems, FM has developed representations for desirable properties in the form of *formal task specifications* (formal specs or φ for short)



Can we use formal specifications in RL?

- A natural question emerges:
  - Can we use formal specifications as objectives in RL?
    - Yes! (Lots of prior work)
  - The more salient question: *When* should we use formal specifications as objectives in RL?

Pros and Cons of using φ in RL

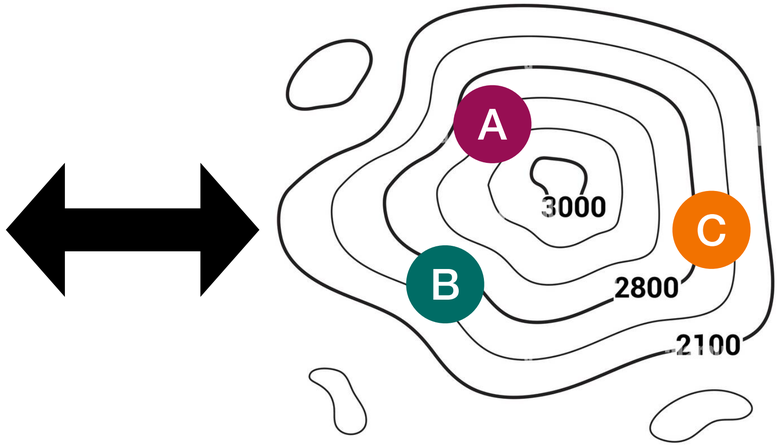
Pros	Cons
★ Precise notion of satisfaction	- Difficult to optimize and provides sparse feedback
★ Provides compositionality (can break down and build up separate specs)	- Solution space for specs is non-continuous and difficult to search
★ Easy to capture non-Markovian semantics e.g. temporally ordered tasks	- Difficult to construct by hand

Pros and Cons of using φ in RL

Pros	Cons
★ **Precise notion of satisfaction**	- **Difficult to optimize and provides sparse feedback**
★ Provides compositionality (can break down and build up separate specs)	- Solution space for specs is non-continuous and difficult to search
★ Easy to capture non-Markovian semantics e.g. temporally ordered tasks	- Difficult to construct by hand

Using φ in RL: Differentiating “correct” and “optimal”

- In correctness-critical settings, we want to learn policies that **certainly** achieve the task at hand
- However, we may have **optimality** conditions alongside our task
- Example: A patrolling quadruped



Correctness: Patrol the three regions consistently!

Optimality: Navigate the terrain safely (avoid collisions!)

LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

Defining “Correct and Optimal” using φ and r

- It’s difficult to express both the correctness condition *and* the optimality condition in a single expression.
- However, the **precision** and Boolean satisfaction of formal specifications allows us to define “correct” behavior
- And the **expressivity** of Markovian reward functions allow us to define “optimality” within correct behaviors.
- Combining the two allows us to ask: which “correct” policy is the most “optimal”?

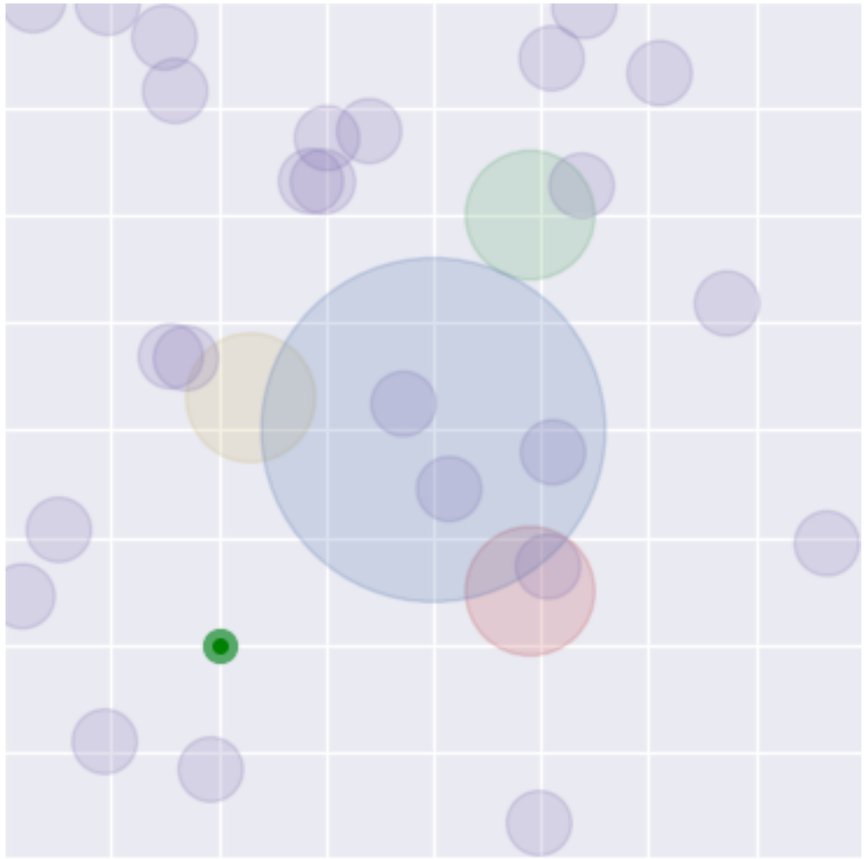
LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

Demonstrating Example: Correctness and Optimality

Task φ: Patrol red, green, and yellow, all while avoiding blue.

Optimality condition r**: +1** for each visit to a purple circle.

I *must* accomplish my task. Amongst all ways of accomplishing my task, choose the optimal one.



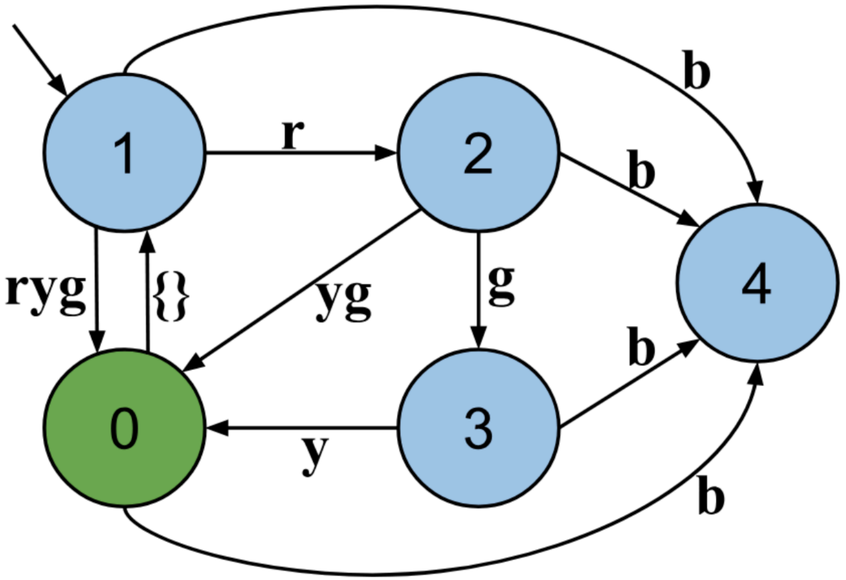
“FlatWorld”

LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

How do we specify φ and r in our setting?

Task specification φ

Buchi Automata (Linear Temporal Logic)



Satisfaction condition: visit accepting states (green) infinitely often.

Optimality Condition r

Use the standard definition of MDP reward:

$$r(s, a, s') = \begin{cases} +1 & \text{if s' in purple} \\ 0 & \text{otherwise} \end{cases}$$

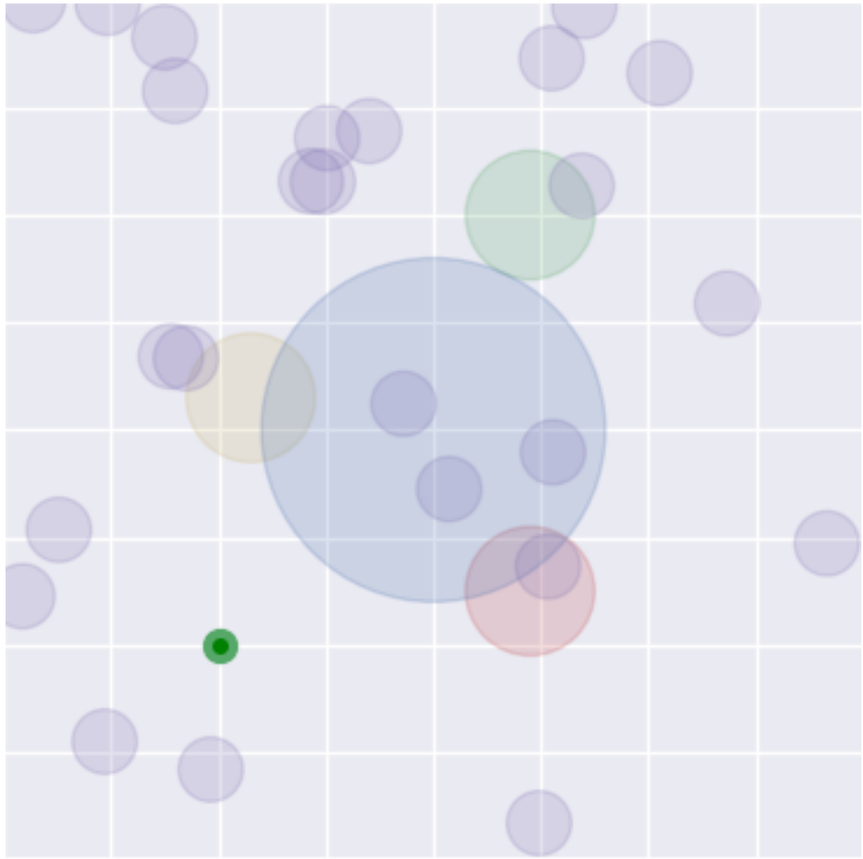
At each transition, receive a scalar reward.

LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

Approach Outline

1. **How do I combine φ with *r* in a form that is readily optimized by (deep) RL?**

2. Does the objective from (1) work in practice? If not, what can we do?



LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

LTL-constrained Policy Optimization: Step-by-Step

1. **Compile φ down to a *proxy reward function* that, when optimized, closely approximates satisfaction of the original property.**

LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

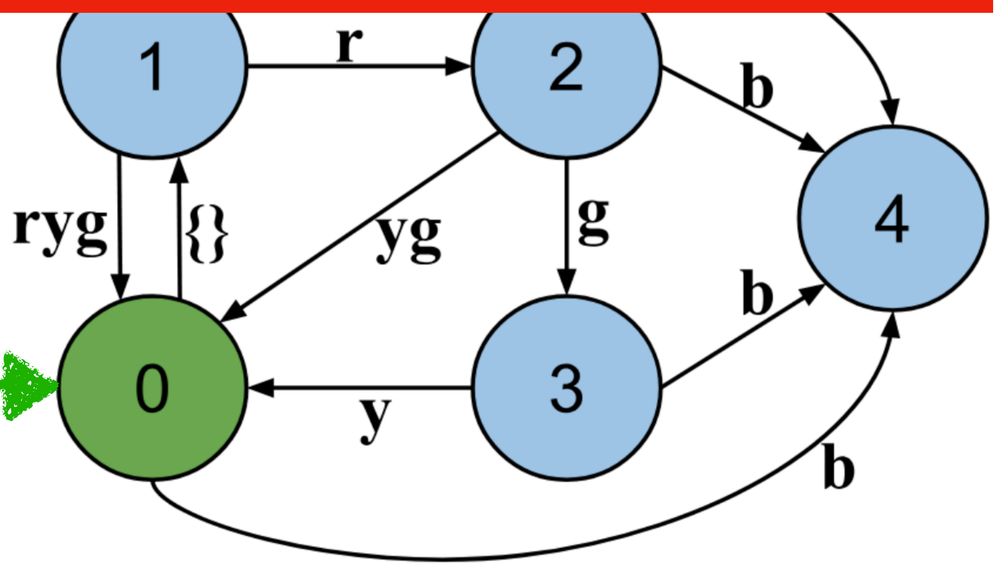
Step 1: Create a proxy reward for φ

- There is no direct way to enforce satisfaction of LTL in RL due to the infinite acceptance condition.
- Instead, we use a ***proxy reward:***

A policy that maximizes r_φ will be *approximately* optimal w.r.t the original LTL satisfaction condition.

$$r_{\text{LTL}}(b_t) = \begin{cases} 1 & \text{if } (b_t \in \mathcal{B}^*) \\ 0 & \text{otherwise} \end{cases}$$

get a reward every time I visit the accepting state!



LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

LTL-constrained Policy Optimization: Step-by-Step

1. Compile φ down to a *proxy reward function* that, when optimized, closely approximates satisfaction of the original property. ✅

2. **Add the φ-proxy reward and the optimality reward (MDP reward) together, and multiply the φ-proxy reward by a Lagrange multiplier λ:**

$$r_{\text{DUAL}}(s, b, a, s', b') = r_{\text{MDP}}(s, a, s') + \lambda r_{\text{LTL}}(b')$$

For large enough λ, the policy that optimizes this reward will satisfy φ with maximum probability AND achieve optimal reward.

LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

LTL-constrained Policy Optimization: Step-by-Step

1. Compile φ down to a *proxy reward function* that, when optimized, closely approximates satisfaction of the original property. ✅

2. Add the φ-proxy reward and the optimality reward (MDP reward) together, and multiply the φ-proxy reward by a Lagrange multiplier λ: ✅

$$r_{\text{DUAL}}(s, b, a, s', b') = r_{\text{MDP}}(s, a, s') + \lambda r_{\text{LTL}}(b')$$

3. **Learn a policy that optimizes the combined reward above.**

We can just use our favorite RL algorithm for this!

LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

LTL-constrained Policy Optimization: Step-by-Step

1. Compile φ down to a *proxy reward function* that, when optimized, closely approximates satisfaction of the original property. ✅

2. Add the φ-proxy reward and the optimality reward (MDP reward) together, and multiply the φ-proxy reward by a Lagrange multiplier λ: ✅

$$r_{\text{DUAL}}(s, b, a, s', b') = r_{\text{MDP}}(s, a, s') + \lambda r_{\text{LTL}}(b')$$

3. Learn a policy that optimizes the combined reward above. ✅

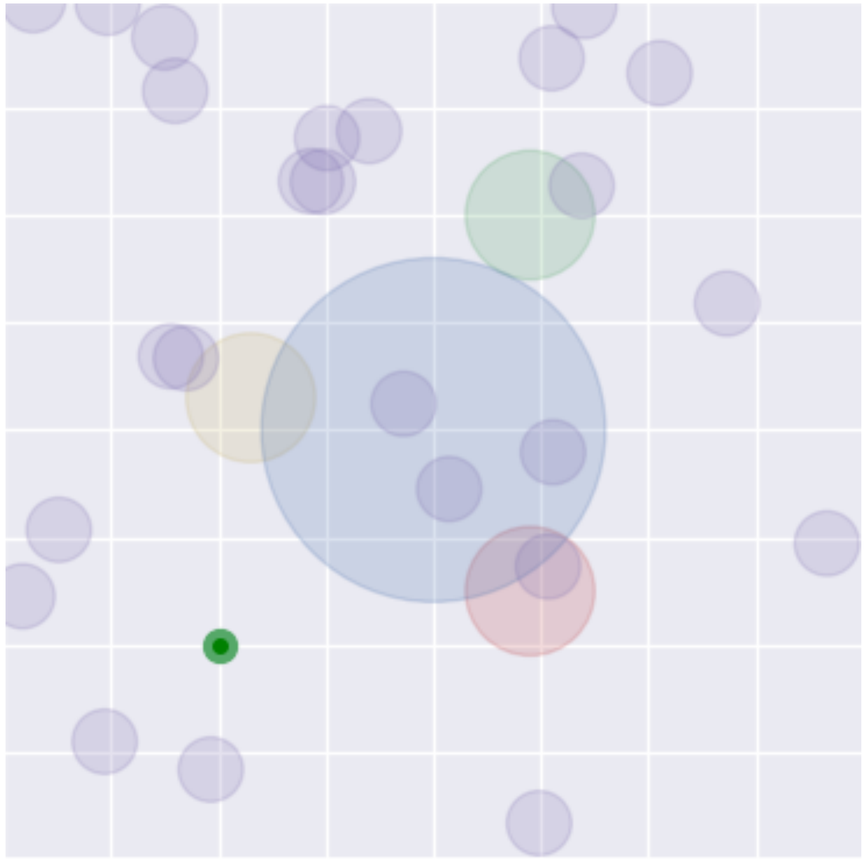
4. **Profit?**

LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

Approach Outline

1. How do I combine φ with r in a form that is readily optimized by (deep) RL?

2. **Does the objective from (1) work in practice? If not, what can we do?**



LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

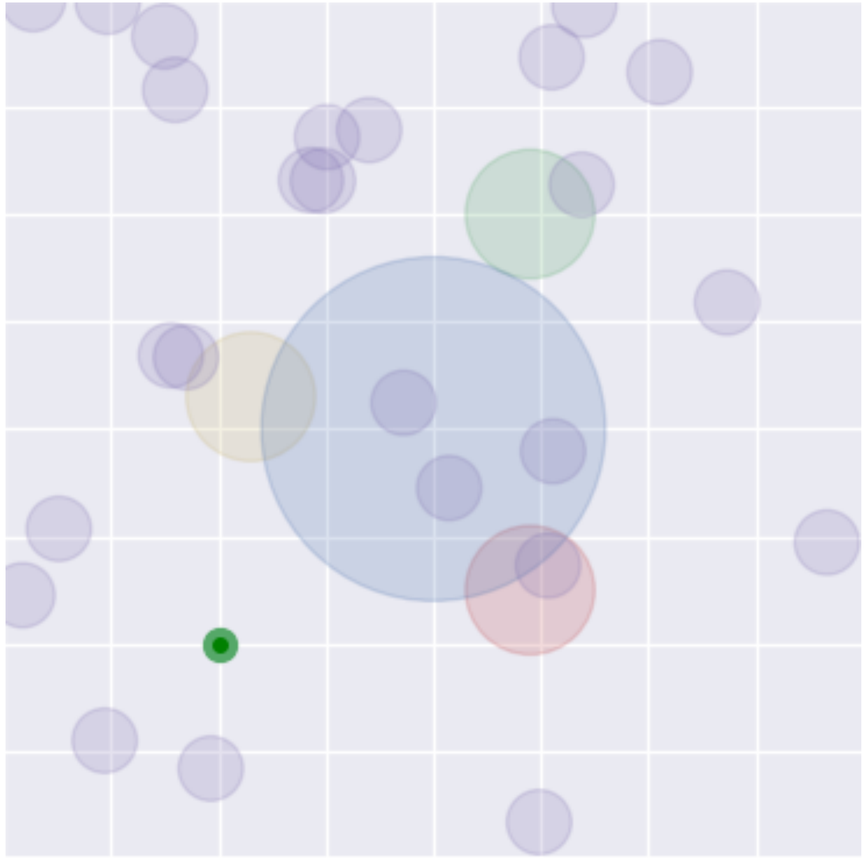
Getting LTL-Constrained Deep RL to work in practice

Task φ: Patrol red, green, and yellow, all while avoiding blue.

Optimality condition r**: +1** for each visit to a purple circle.

$$r_{\text{DUAL}}(s, b, a, s', b') = r_{\text{MDP}}(s, a, s') + \lambda r_{\text{LTL}}(b')$$

Let's train a policy on this reward and see what happens in this example.



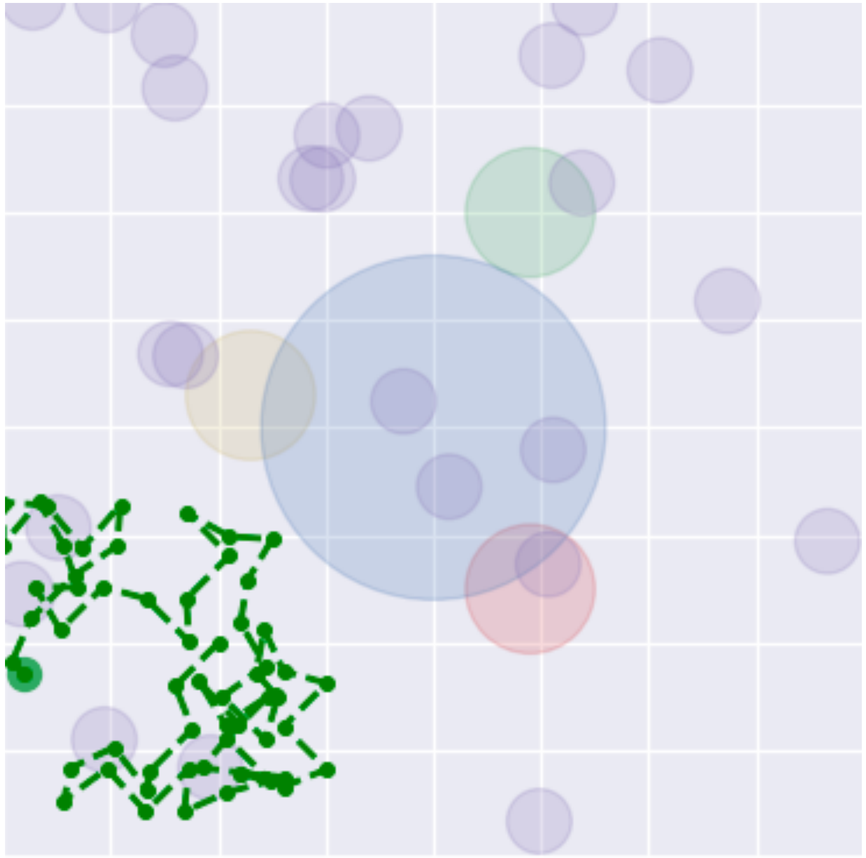
LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

Getting LTL-Constrained Deep RL to work in practice

Why does this happen?

r_{LTL} is **sparse.**

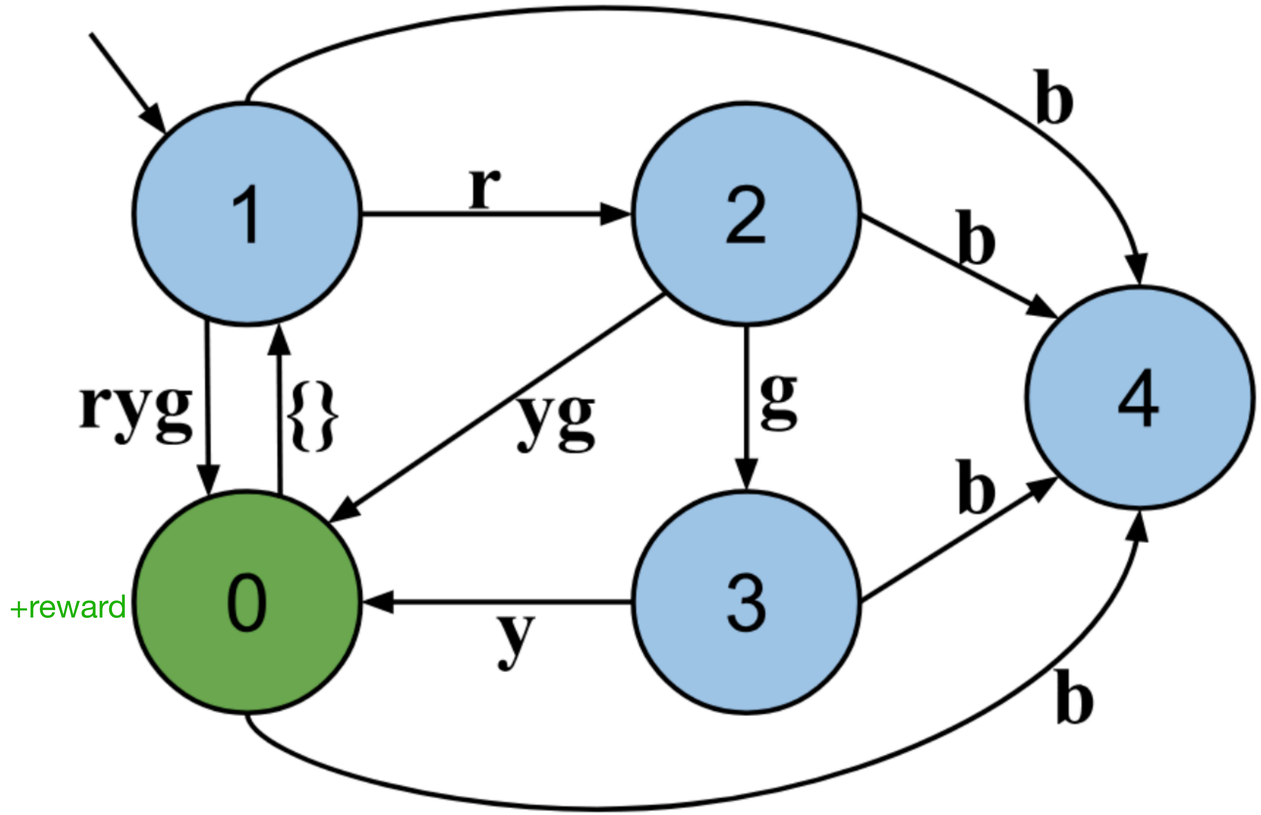
if r_{MDP} is **dense**, a policy can get ‘distracted’ during learning, and ignore r_{LTL}.



LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

Addressing the sparsity issue of reward proxies

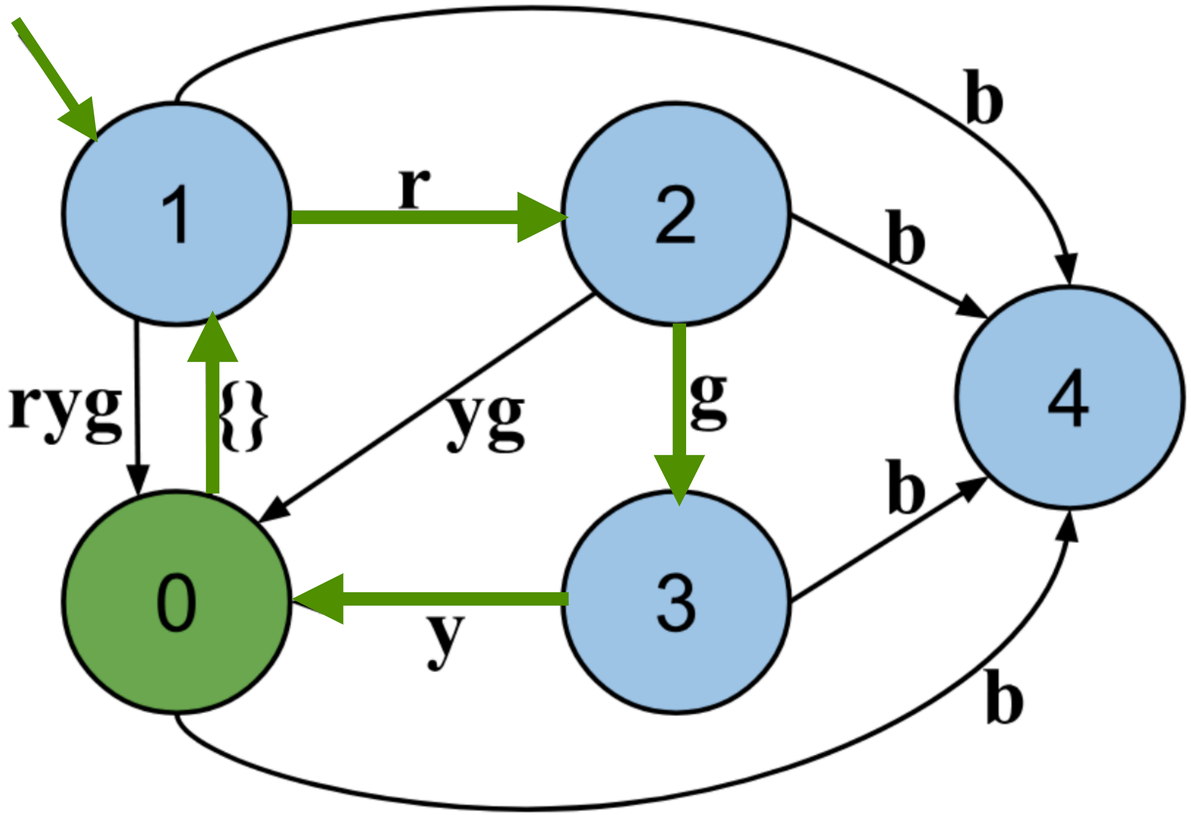
Can we **shape** r_{LTL} to make it more dense?

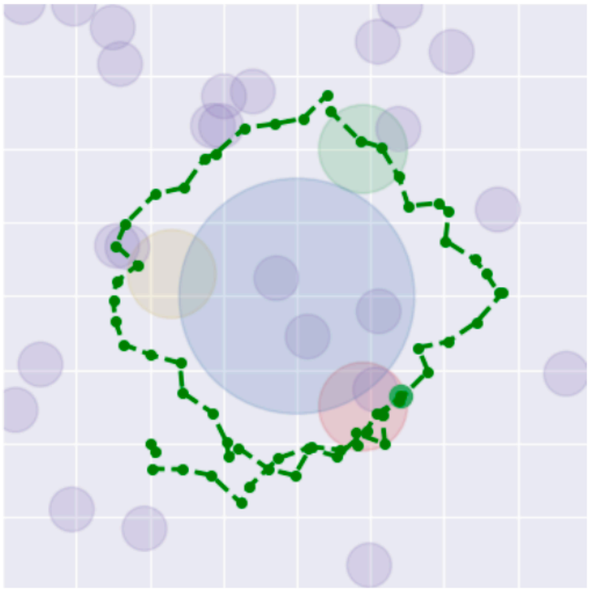


LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

Addressing the sparsity issue of reward proxies

Key idea: every visit to an accepting state must traverse an "accepting cycle" within our BA.



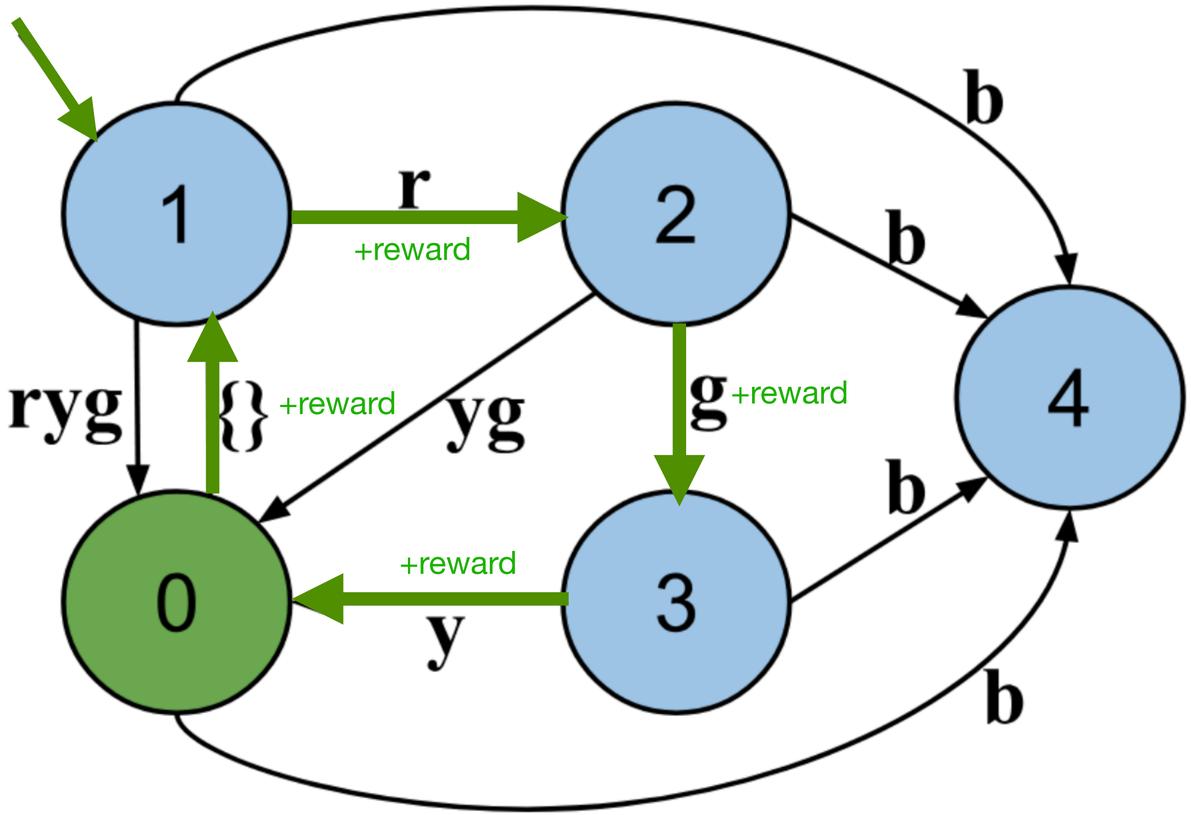


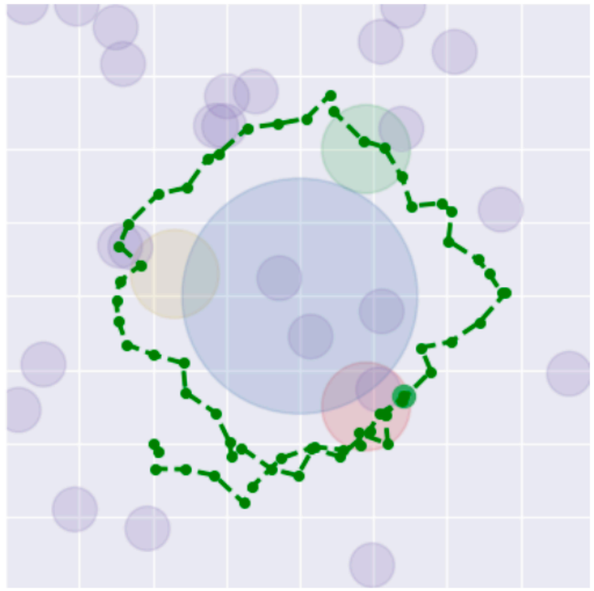
An accepting trajectory for our example.

LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

Addressing the sparsity issue of reward proxies

So, why not give a reward every time we *progress* along an accepting cycle?





Reward becomes denser!

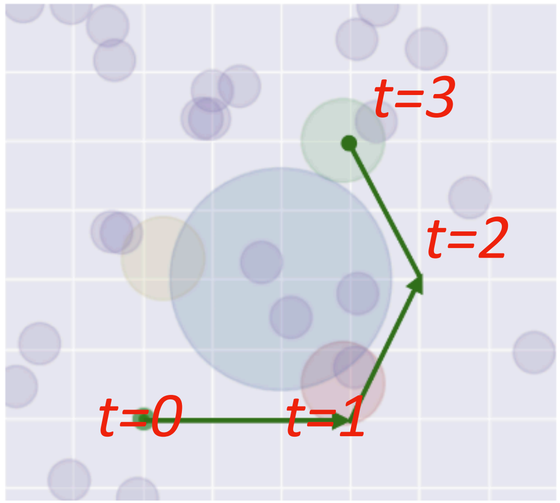
LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

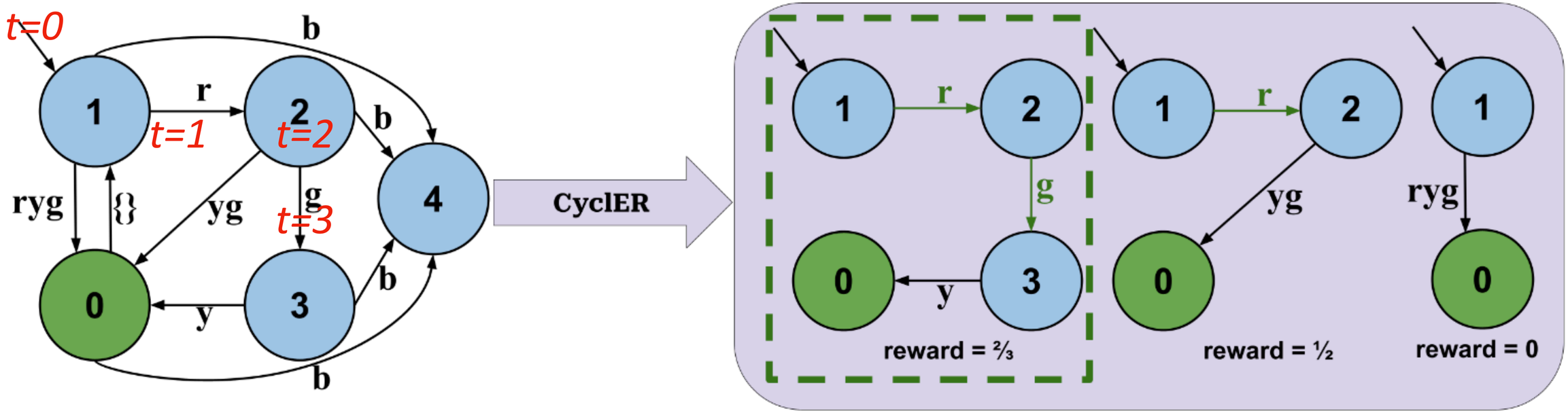
Reward Shaping with Cycle Experience Replay (CyclER)

- Counterfactually reason over all possible accepting cycles at the end of each trajectory to find the accepting cycle that provides the most dense reward
- Retroactively update the rewards from this trajectory based on the most dense accepting cycle
- Notably: we can incorporate Quantitative Semantics (QS) into CyclER and densify reward even further!

LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

CyclER: Example

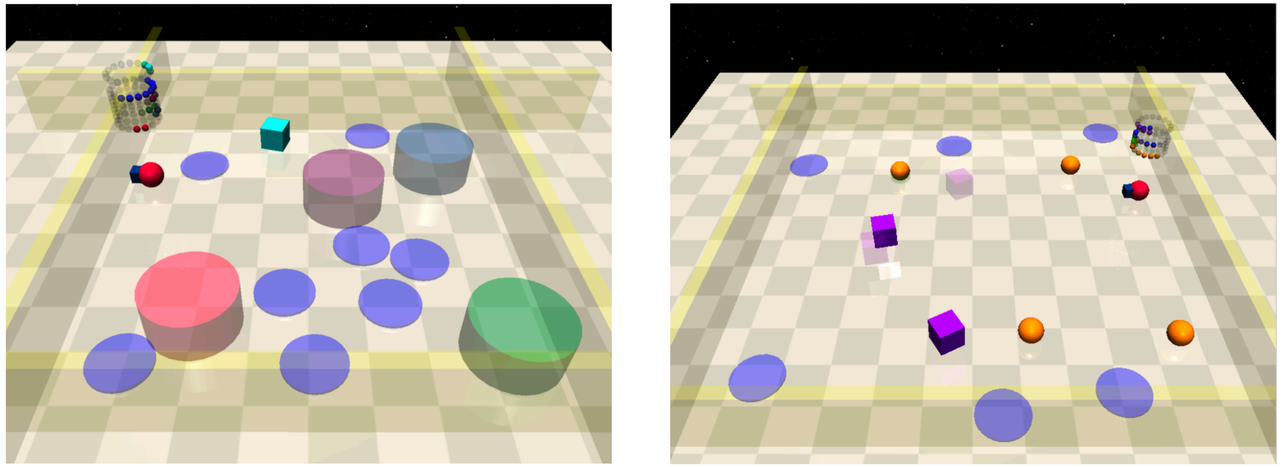




LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

CyclER: Experimental Evaluation

- Evaluated in environments with both discrete and continuous state/action spaces
- Tasks: Navigation tasks of varying complexity
- Baselines: Existing reward shaping methods for LTL-guided RL



Example MuJoCo Safety-Gymnasium Environments

LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

CyclER: Experimental Evaluation

	FlatWorld		ZonesEnv		ButtonsEnv	
	r_{LTL}	r_{MDP}	r_{LTL}	r_{MDP}	r_{LTL}	r_{MDP}
CyclER	$\mathbf{2.0 \pm 0.5}$	45.3 ± 8.5	$\mathbf{1.8 \pm 0.4}$	-27.8 ± 4.55	$\mathbf{2.6 \pm 0.3}$	30.4 ± 5.6
LCER	0.0 ± 0.0	103.4 ± 76.6	0.0 ± 0.0	-3.8 ± 1.9	0.0 ± 0.0	118.8 ± 143.2
LCER, no r_{MDP}	0.8 ± 0.4	30.8 ± 10.1	0.0 ± 0.0	-2.7 ± 0.9	0.6 ± 0.4	13.2 ± 8.53
TLTL	-	-	0.0 ± 0.0	-4.0 ± 2.2	0.0 ± 0.0	9.6 ± 6.7
BHNR	-	-	0.0 ± 0.0	-0.8 ± 0.6	0.0 ± 0.0	35.8 ± 7.9

CyclER succeeds where other baselines cannot accomplish the LTL task!

Even a baseline (blue) that completely ignores MDP reward cannot accomplish the LTL task, which means that sparsity is an issue even in the absence of r_{MDP}.

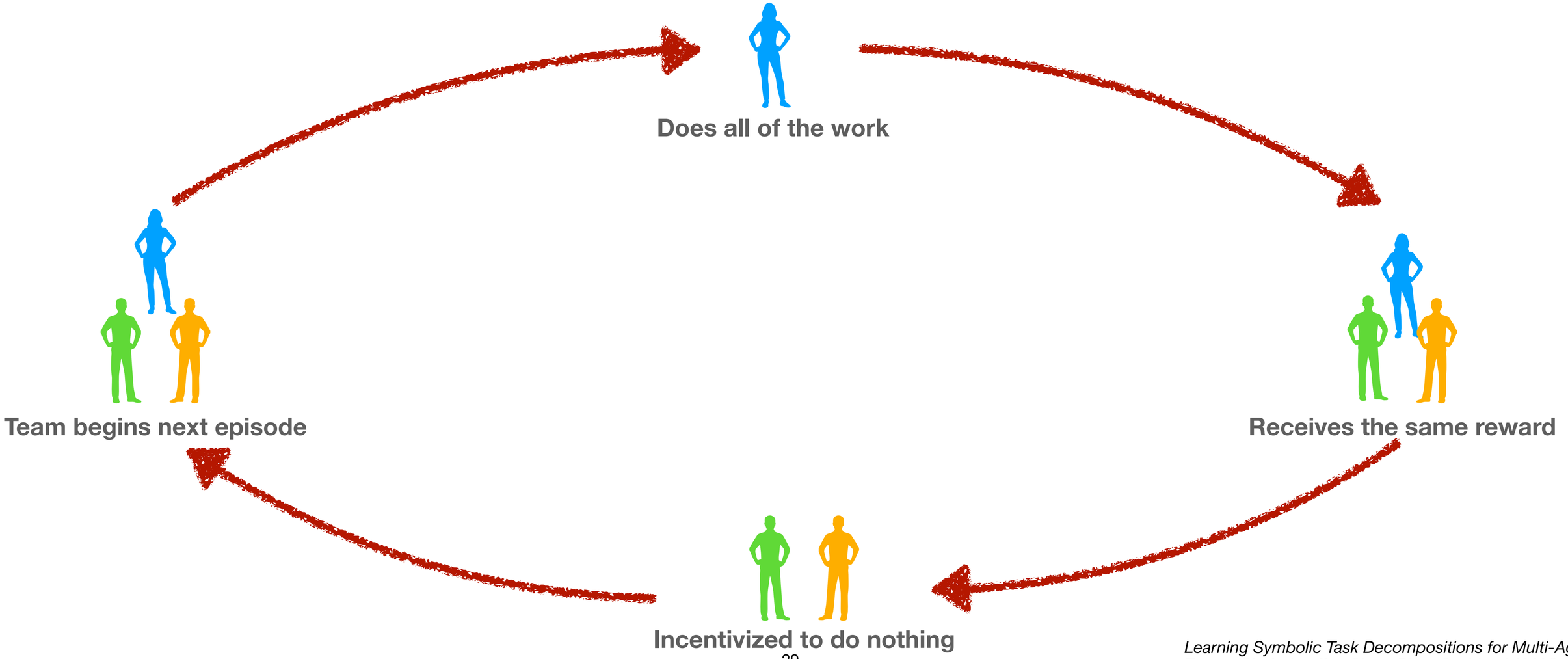
LTL-Constrained Policy Optimization With Cycle Experience Replay. **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*

Pros and Cons of using φ in RL

Pros	Cons
★Precise notion of satisfaction ✅	- ~~Difficult to optimize and provides sparse feedback~~ ✅
★ **Provides compositionality (can break down and build up separate specs)**	- **Solution space for specs is non-continuous and difficult to search**
★ Easy to capture non-Markovian semantics e.g. temporally ordered tasks	- Difficult to construct by hand

The “Group Project” Problem in Multi-Agent RL

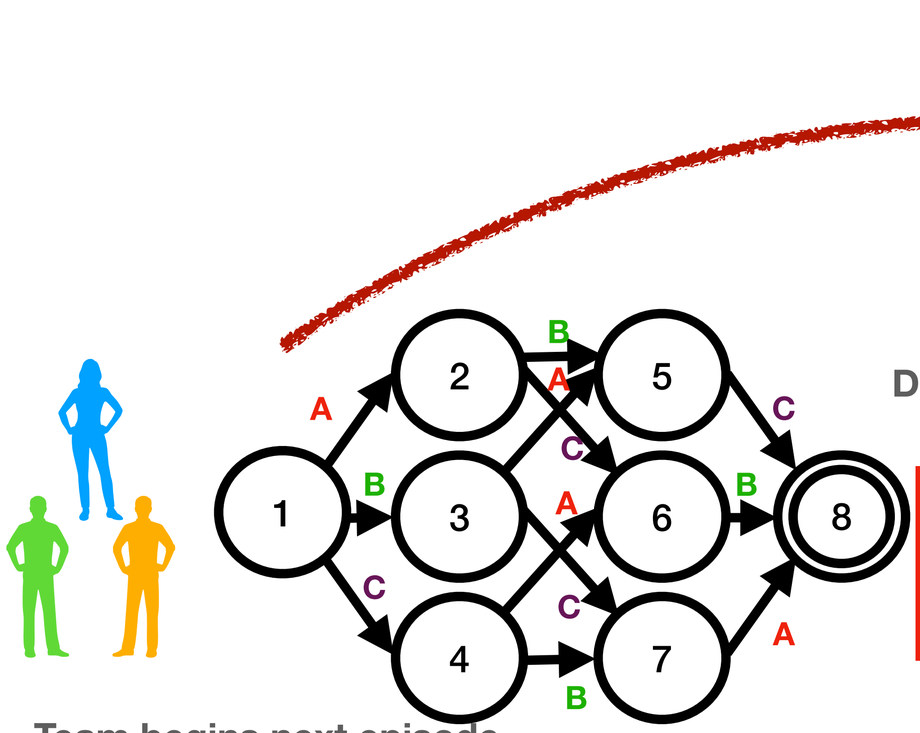
When a team of agents **share reward** in a cooperative setting, issues may arise:

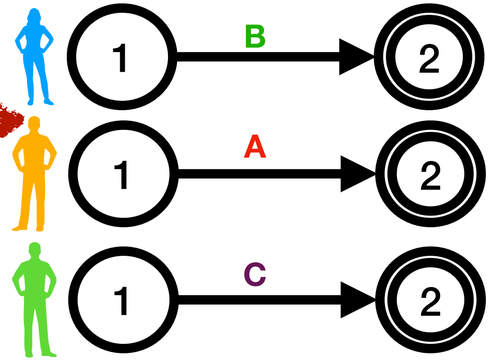


Learning Symbolic Task Decompositions for Multi-Agent Teams. **Shah,** Lauffer, Chen, Pitta, Seshia. *AAMAS, 2025.*

Using φ in MARL can resolve credit assignment issues

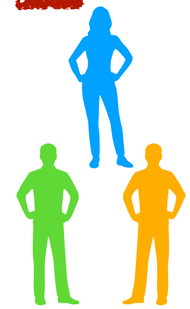
By representing the task symbolically, we can **decompose** it into individual sub-tasks.



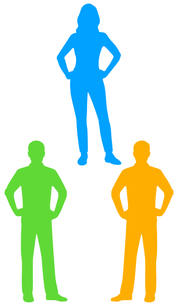


Decompose the task and assign each agent their own sub-task

We can verify that a decomposition is valid: completing each sub-task will complete the overall task.



eceive *separate* rewards for accomplishing their own tasks

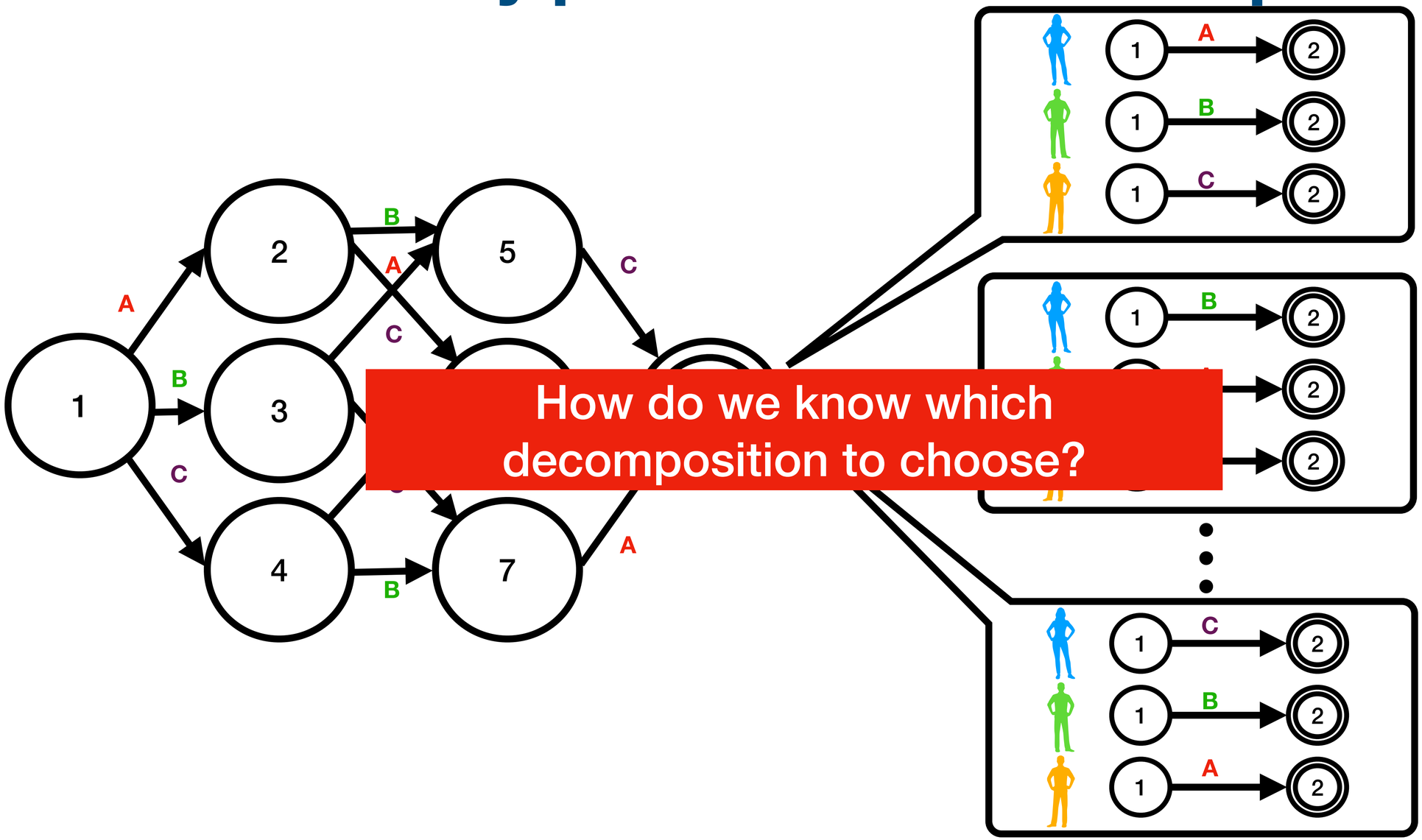


Incentivized to do something useful!

Team begins next episode

Learning Symbolic Task Decompositions for Multi-Agent Teams. **Shah,** Lauffer, Chen, Pitta, Seshia. *AAMAS, 2025.*

Problem: many possible valid decompositions exist



Learning Symbolic Task Decompositions for Multi-Agent Teams. **Shah,** Lauffer, Chen, Pitta, Seshia. *AAMAS, 2025.*

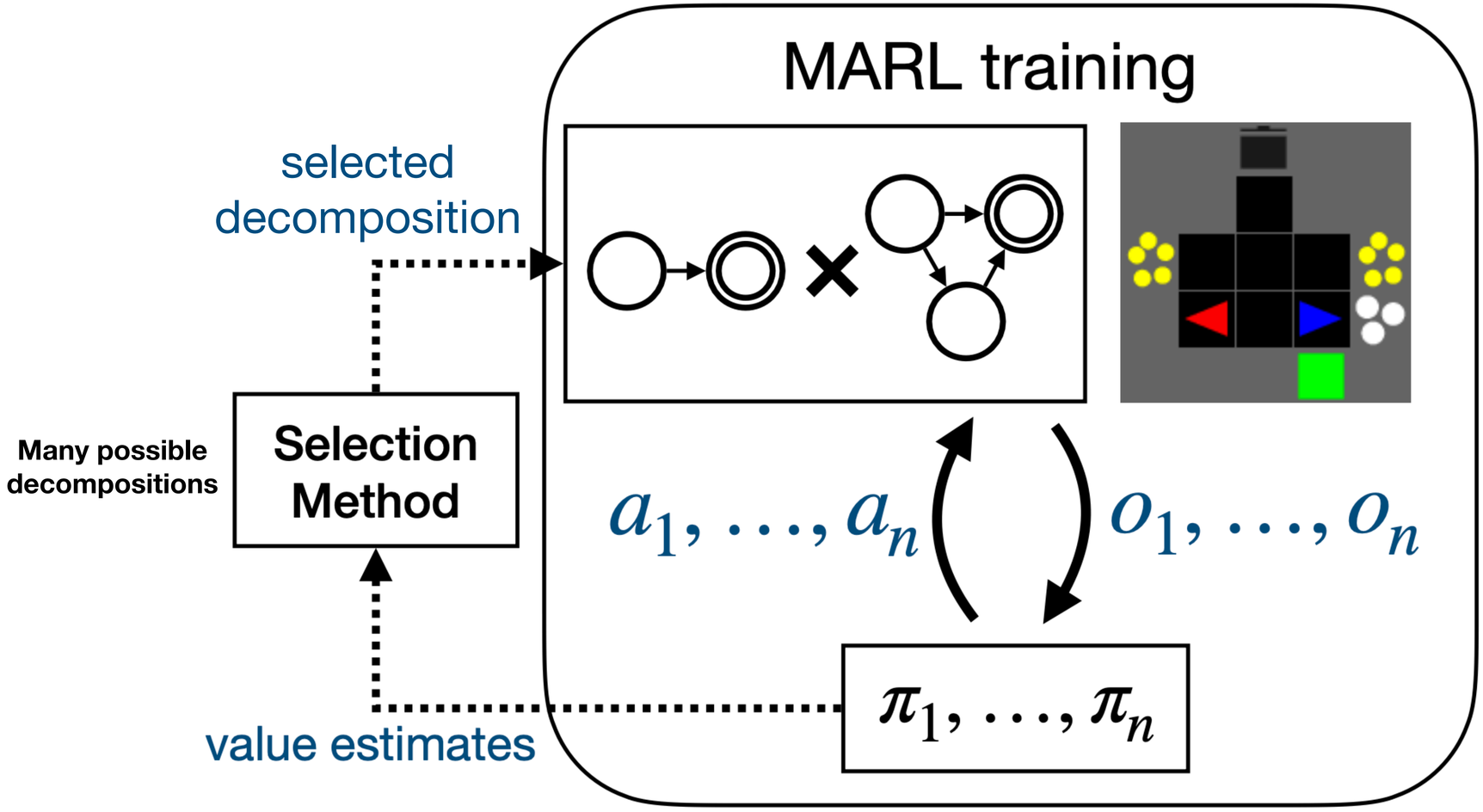
Summary of our problem

- Decomposing symbolic task representations enables assigning sub-tasks to individual agents with the guarantee that completing all sub-tasks will complete the overall task
  - This allows for learning more efficient strategies for completing cooperative tasks.
- **The problem:** many valid decompositions may exist. We don't know which one solves the task most efficiently (optimally)
  - because we don't have **a-priori knowledge** of the environment.

Learning Symbolic Task Decompositions for Multi-Agent Teams. **Shah,** Lauffer, Chen, Pitta, Seshia. *AAMAS, 2025.*

Our work: Learning Task Decompositions On-The-Fly

As we collect experience, use it to inform which decomposition is the best!



Learning Symbolic Task Decompositions for Multi-Agent Teams. **Shah,** Lauffer, Chen, Pitta, Seshia. *AAMAS, 2025.*

Overview of our procedure

Learning Optimal Task Decompositions (LOTaD)

1. Generate a set of *candidate* decompositions.

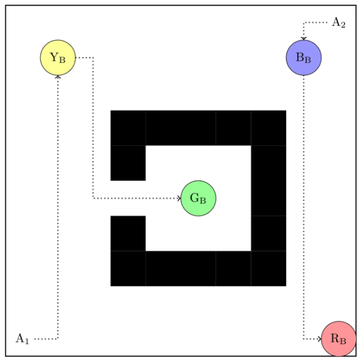
2. Use a task-conditioned policy to accelerate learning across different candidates.

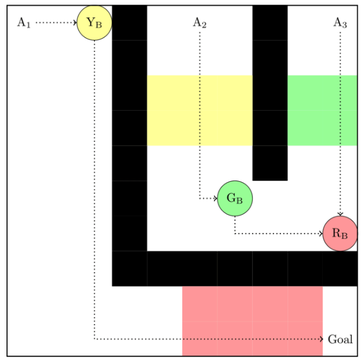
3. Use the Upper Confidence Bound (UCB) algorithm to balance exploring and exploiting different candidates.

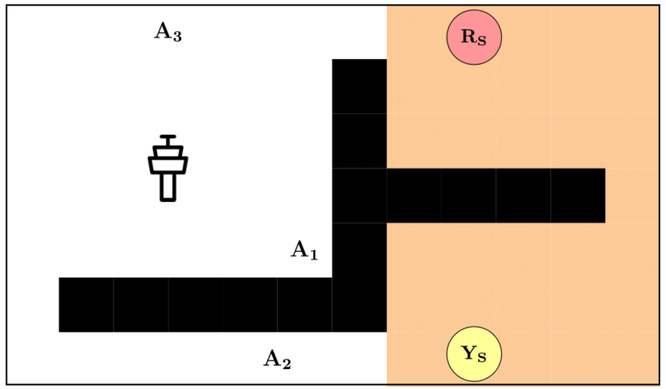
Learning Symbolic Task Decompositions for Multi-Agent Teams. **Shah,** Lauffer, Chen, Pitta, Seshia. *AAMAS, 2025.*

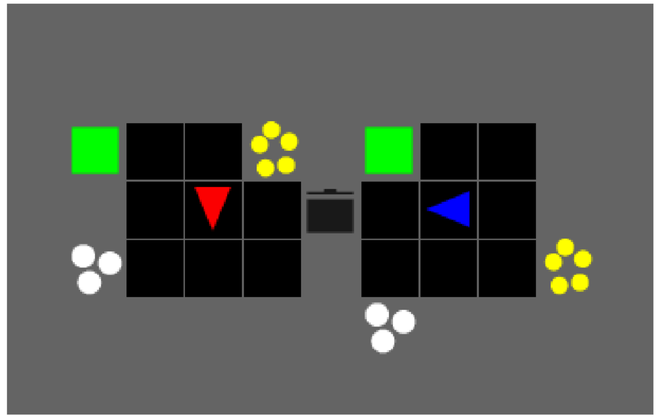
Experiments: Evaluating LOTaD

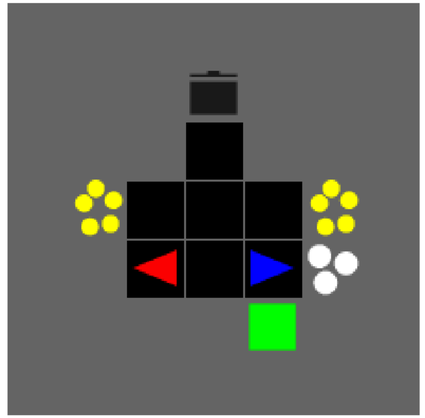
- **Baselines:**
  - ATAD: Prior work on generating task decompositions that uses heuristics to select and fix a single decomposition for use during MARL
  - Monolithic: All agents are given the original symbolic task as their task
- **Example environments:**





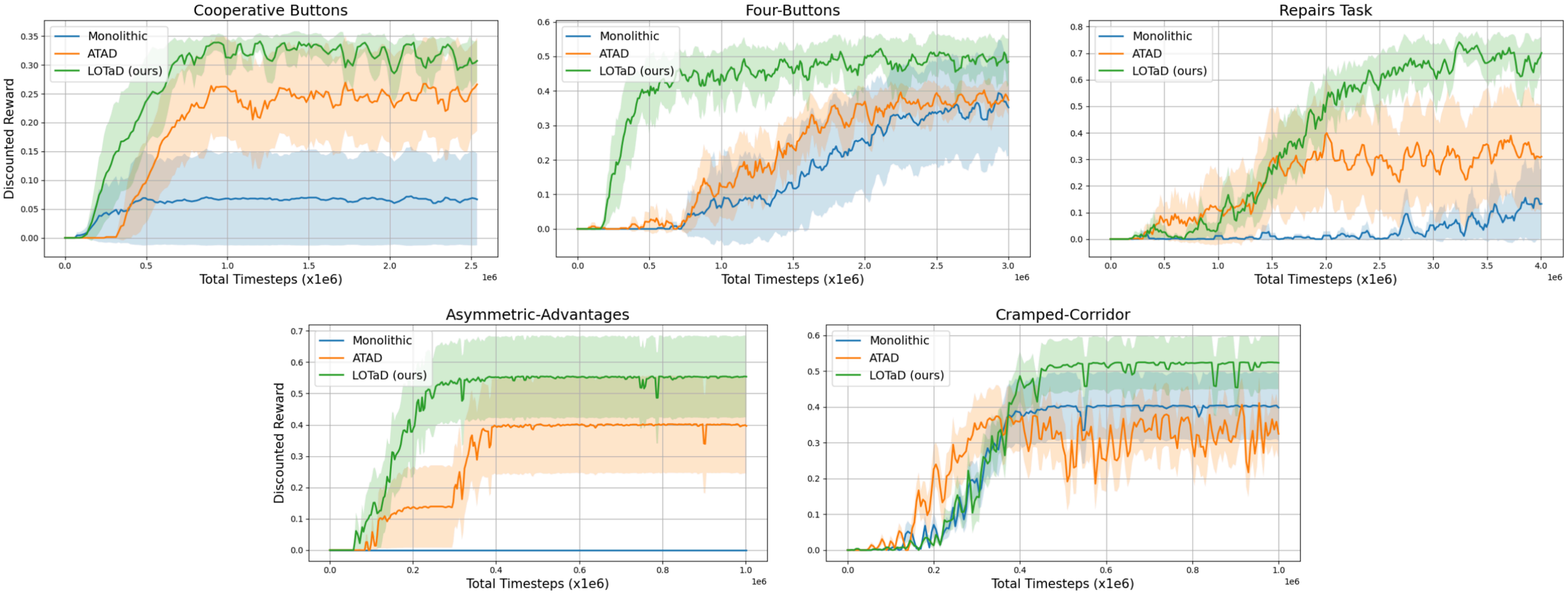






Learning Symbolic Task Decompositions for Multi-Agent Teams. **Shah,** Lauffer, Chen, Pitta, Seshia. *AAMAS, 2025.*

Experiments: Evaluating LOTaD



LOTaD strongly outperforms existing baselines!

Learning Symbolic Task Decompositions for Multi-Agent Teams. **Shah,** Lauffer, Chen, Pitta, Seshia. *AAMAS, 2025.*

Conclusions and Next Steps

- Formal task specifications, such as automata, can act as a precise and composable means of identifying objectives in RL
- When should you use formal specs?
  - When you have a precise satisfaction condition in correctness-critical settings
  - When your task is a *composition* of separable sub-tasks
- Next steps: exploring how formal structure can augment foundation model capabilities

Thanks!

- Papers discussed today:
  - *LTL-Constrained Policy Optimization With Cycle Experience Replay.* **Shah,** Voloshin, Yang, Verma, Chaudhuri, Seshia. *TMLR, 2025.*
  - *Learning Symbolic Task Decompositions for Multi-Agent Teams*. **Shah**, Lauffer, Chen, Pitta, Seshia. *AAMAS, 2025.*
- reach out to me: ameesh@berkeley.edu
- Thanks to my collaborators!
  - Berkeley: Niklas Lauffer, Thomas Chen, Nikhil Pitta, Sanjit A. Seshia
  - Non-Berkeley: Cameron Voloshin, Chenxi Yang, Abhinav Verma, Swarat Chaudhuri