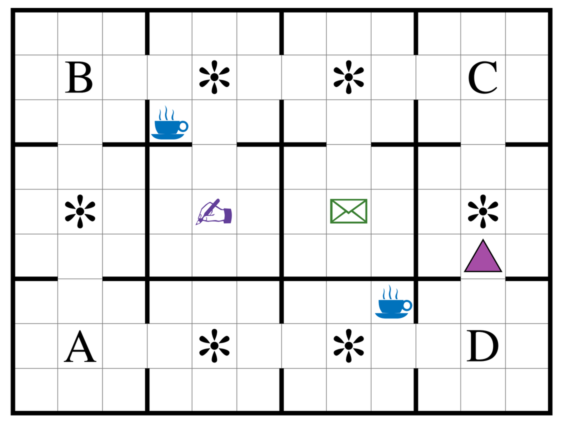
Symbolic Planning and Model-Free Reinforcement Learning: Training Taskable Agents

León Illanes[1] Xi Yan[1] Rodrigo Toro Icarte[1,2] Sheila A. McIlraith[1,2]

[1]Department of Computer Science, University of Toronto [2]Vector Institute

Running Example



Symbol	Meaning
▲	Agent
✻	Furniture
	Coffee machine
	Mail room
	Office
A, B, C, D	Marked locations

Motivation – Taskability

- Specify high-level, **goal-directed tasks** to an agent
- **Avoid reexploration** of the environment

Task examples

T1. Deliver mail to the office
T2. Deliver coffee and mail to the office
T3. Visit locations A, B, C, and D (in any order)

Possible approches

- Model-based Reinforcement Learning
- **Hierarchical Reinforcement Learning**
- Reward Shaping
- **Modular RL and Policy Sketches**
- **Structured and Decomposable Reward Functions**

In this work

- Where do the options come from?
- Where do reward functions come from?
- Where do policy sketches come from?

Answer: Typically, from a human expert.

The expert has a working model of the environment in mind and chooses options, designs reward functions, or sketches policies based on that. Given a new task, most of the expert's work will need to be repeated.

Our approach: Use an explicit high-level model.

- The model specifies abstract actions
  - These correspond to relevant options
- New tasks are very easy to specify
- We automatically find abstract solutions
- We use these solutions to guide RL agent

Symbolic Planning

"Planning is the art and practice of thinking before acting."
–Patrik Haslum

- State-space given by a set of state properties
  - e.g., propositions
- Actions given as preconditions and effects
  - Properties needed for the action to be applicable
  - Properties that change after the action is applied
- Tasks are given by an initial state and a goal condition
- Solutions or plans are sequences of actions

In the example

Propositions:
`have-mail/coffee`
`delivered-mail/coffee`
`visited-A/B/C/D`

Actions:
`get-mail/coffee`
`deliver-mail/coffee`
`go-to-A/B/C/D`

`get-coffee`:
pre: (none)
eff: `have-coffee`
obs: *coffee-machine*

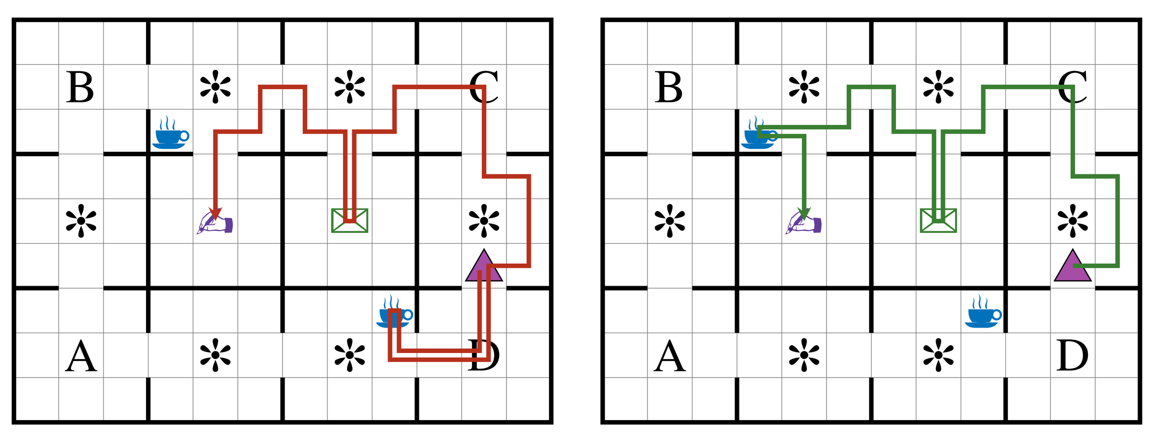
`deliver-coffee`:
pre: `have-coffee`
eff: `delivered-coffee`, **not** `have-coffee`
obs: *office*

Plans

T1. ⟨get-coffee, deliver-coffee⟩
T2. ⟨get-coffee, get-mail, deliver-coffee, deliver-mail⟩
T3. ⟨go-to-A, go-to-B, go-to-C, go-to-D⟩

Executing Abstract Plans

Even assuming we have perfect policies for the high-level actions, execution of the plans results in suboptimal behavior. Consider the plan for T2 (left) versus the optimal (right):



Can we relax the ordering constraints?

Partial-Order Plans

- A collection of actions and a partial order over them
- Every strict ordering that respects the partial order is a valid sequential plan
- Well established in the Planning literature
  - Some planners can produce partial-order plans
  - Sequential plans can be relaxed into partial-order plans

Examples

T1. Actions: `get-coffee`, `deliver-coffee`
Order: `get-coffee` ≺ `deliver-coffee`

T2. Actions: `get-coffee`, `get-mail`, `deliver-coffee`, `deliver-mail`
Order: `get-coffee` ≺ `deliver-coffee`, `get-mail` ≺ `deliver-mail`

T3. Actions: `go-to-A`, `go-to-B`, `go-to-C`, `go-to-D`
Order: (none)

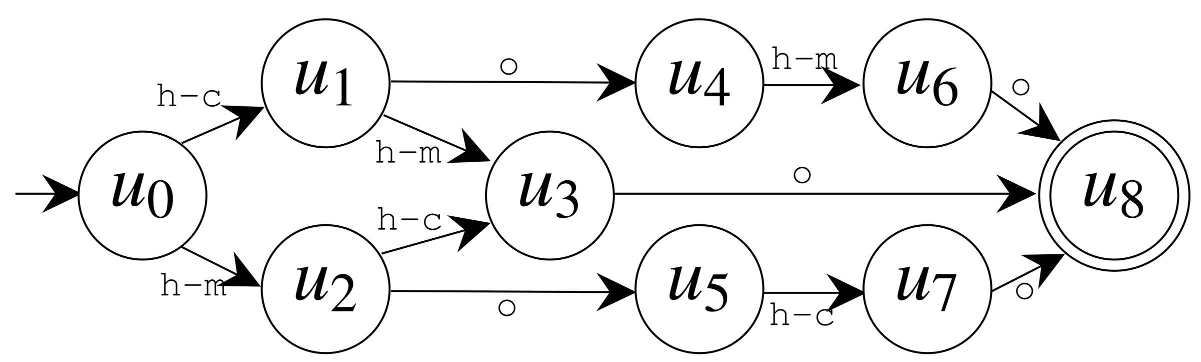
From POP to RL

- We train a metacontroller to execute a given POP
- The metacontroller is trained in a standard HRL manner
  - It is a-priori restricted to only select options that advance the execution of the POP

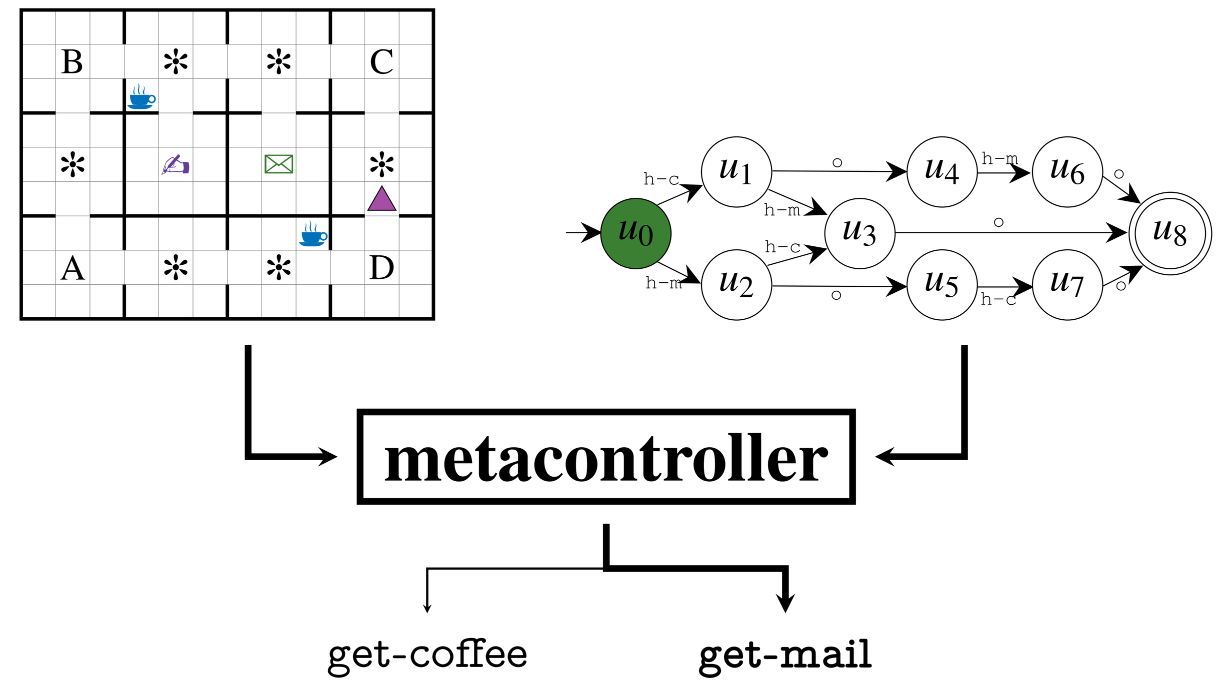
Implementation details

- POPs are represented with Reward Machines
  - Finite-state machines with transitions that match observations in the environment
- The state in the machine represents which actions in the POP have already occurred
- The transitions depend on the observed environment

Example (T2)



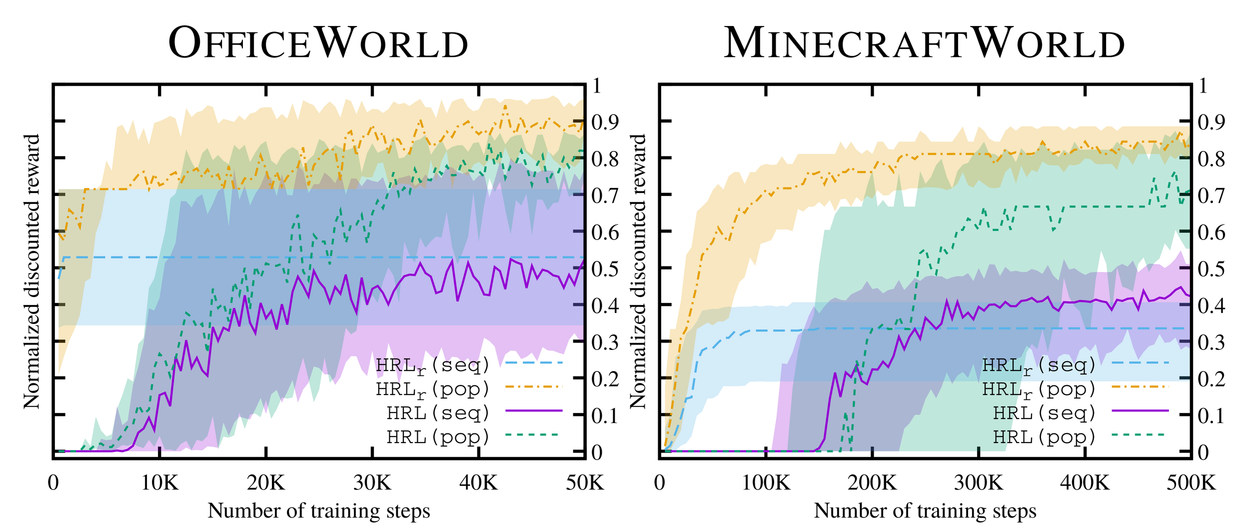
u_0: $\varnothing$
u_1: {get-coffee}
u_2: {get-mail}
u_3: {get-coffee, get-mail}
u_4: {get-coffee, deliver-coffee}
u_5: {get-mail, deliver-mail}
u_6: {get-coffee, get-mail, deliver-coffee}
u_7: {get-mail, get-coffee, deliver-coffee}
u_8: {get-coffee, get-mail, deliver-coffee, deliver-mail}



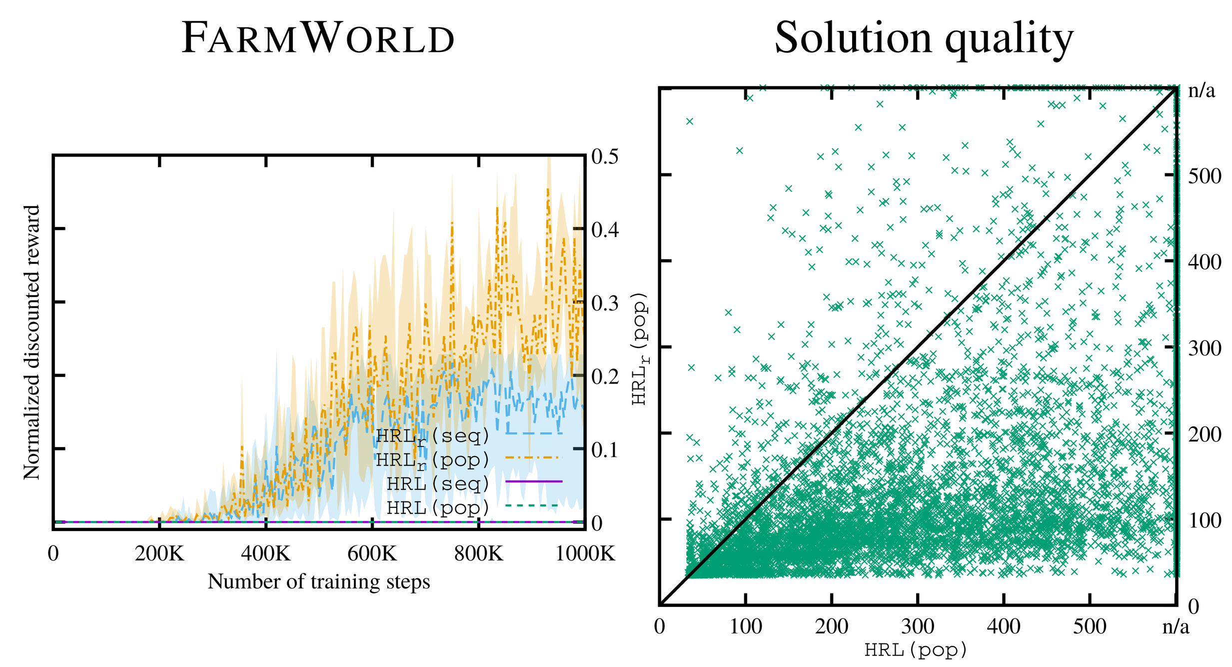
Experiments

Assume we have a well trained set of policies for the high-level actions. We compare our approach with standard HRL.

Discrete domains



Continuous domain



Summary

- Specify abstract state and action models
  - State properties, action preconditions and effects
- Use them to define tasks and solve them more efficiently
  - Find a family of abstract plans and train a metacontroller to instantiate it into a single plan