Induction and Exploitation of Subgoal Automata for Reinforcement Learning

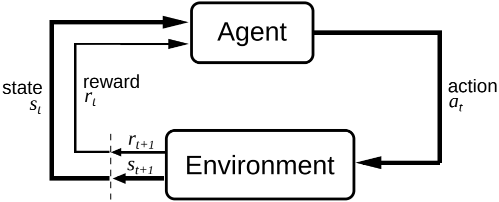
Daniel Furelos-Blanco Mark Law Anders Jonsson Krysia Broda Alessandra Russo

Imperial College London

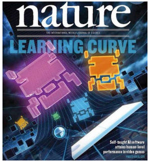
Universitat Pompeu Fabra Barcelona

Motivation I

Reinforcement learning (RL) is a family of algorithms for controlling an agent that acts in an environment. The *goal* is to maximise some measure of cumulative reward that the agent receives.



[Sutton and Barto, 1998]



[Mnih et al., 2015]

[Silver et al., 2018]

Motivation II

Despite of the recent advancements, there still are many challenging tasks.

Montezuma's Revenge
[Bellemare et al., 2013]

Animal-AI Olympics
[Beyret et al., 2019]

NetHack
[Küttler et al., 2020]

We aim to solve this kind of tasks by exploiting their *structure* in the form of (temporal) *abstractions*.

Motivation III

Finite-state automata have been used as a means for *abstraction* across different areas of Artificial Intelligence (AI):

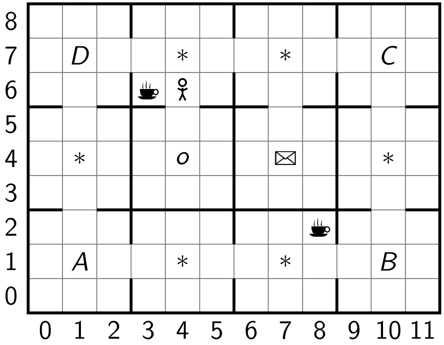
- Control of agents in robotics [Brooks, 1989] and games [Buckland, 2004].
- Planning [Bonet et al., 2009, Hu and De Giacomo, 2011, Segovia Aguas et al., 2018].
- Reinforcement learning:
    - Abstract decision hierarchies [Parr and Russell, 1997, Leonetti et al., 2012].
    - Memory in partially observable environments [Meuleau et al., 1999, Toro Icarte et al., 2019].
    - Represent reward functions (reward machines) [Toro Icarte et al., 2018].
    - Ease the interpretation of the policies encoded by neural networks [Koul et al., 2019].

Problem Formulation I

We consider *episodic POMDPs* $\mathcal{M}^{\Sigma} = \langle S, S_T, S_G, \Sigma, A, p, r, \gamma, \nu \rangle$ where:

- S is a finite set of *latent* states,
- $S_T \subseteq S$ is a finite set of *terminal* latent states,
- $S_G \subseteq S_T$ is a finite set of *goal* latent states,
- Σ is a finite set of *visible* states,
- $\nu : S \to \Delta(\Sigma)$ is a mapping from latent states to probability distributions over visible states, and
- A, p, r and γ are defined as for MDPs.

The OfficeWorld environment [Toro Icarte et al., 2018]



Example – Deliver coffee to the office while avoiding $*$ (Coffee).

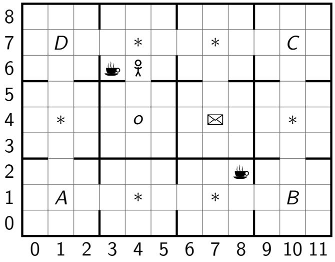
- Latent state: (x, y, has_coffee)
- Visible state: (x, y)
- Goal state: $(4, 4, \top)$
- Terminal states: $(4, 7, \top)$, $(4, 7, \bot)$, ...

Problem Formulation II

- At each step t, the agent observes a tuple $\boldsymbol{\sigma}_t = \langle \sigma_t^{\Sigma}, \sigma_t^{T}, \sigma_t^{G} \rangle$, where:
  - $\sigma_t^{\Sigma} \in \Sigma$ is a visible state,
  - $\sigma_t^{T} = \mathbb{I}[s_t \in S_T]$ indicates if the latent state is terminal, and
  - $\sigma_t^{G} = \mathbb{I}[s_t \in S_G]$ indicates if the latent state is a goal state.

- The tasks are enhanced by a set of propositional variables $\mathcal{O}$ (also called *observables*).

- A *labeling function* $L : \Sigma \to 2^{\mathcal{O}}$ maps visible states into subsets of observables called *observations*.

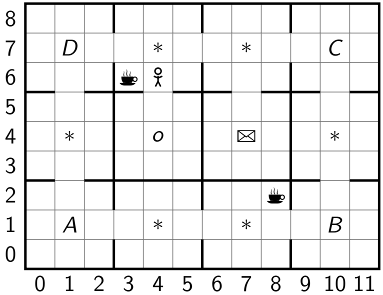


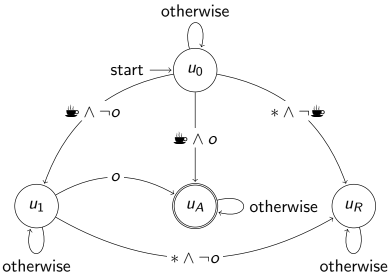
Examples

- At $t = 0$, $\boldsymbol{\sigma}_t = \langle (4,6), \perp, \perp \rangle$.
- Observables $\mathcal{O} = \{$☕, ✉, $o, A, B, C, D, *\}$.
- $L((4,6)) = \emptyset$.
- $L((3,6)) = \{$☕$\}$.

Subgoal Automata

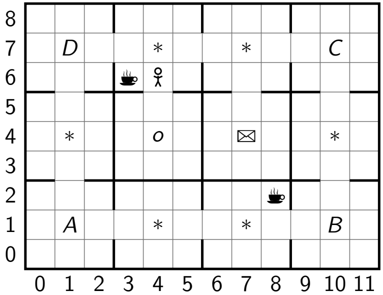
A type of *deterministic* finite automaton that encodes the subgoals of an *episodic goal-oriented* task as *propositional logic formulas*.

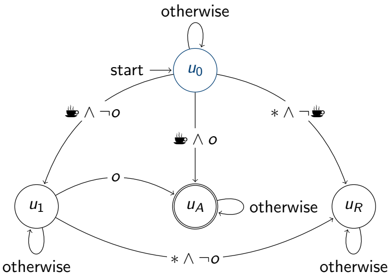




Subgoal Automata

A type of *deterministic* finite automaton that encodes the subgoals of an *episodic goal-oriented* task as *propositional logic formulas*.



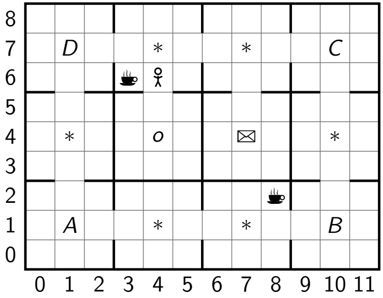


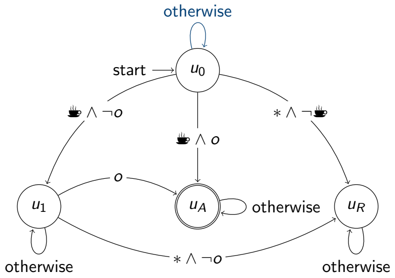
Traces

- Execution trace: λ
- Observation trace: $\lambda_{L,\mathcal{O}}$

Subgoal Automata

A type of *deterministic* finite automaton that encodes the subgoals of an *episodic goal-oriented* task as *propositional logic formulas*.



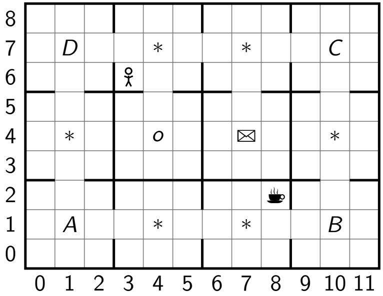


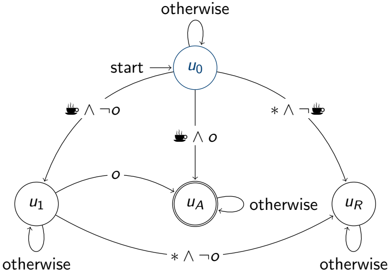
Traces

- Execution trace: $\lambda = \langle\langle(4,6), \perp, \perp\rangle,$
- Observation trace: $\lambda_{L,\mathcal{O}} = \langle\{\},$

Subgoal Automata

A type of *deterministic* finite automaton that encodes the subgoals of an *episodic goal-oriented* task as *propositional logic formulas*.



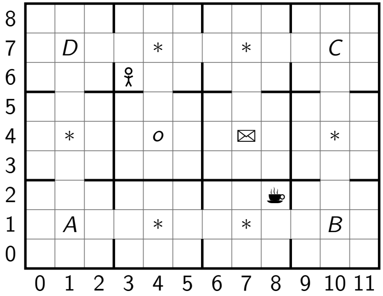


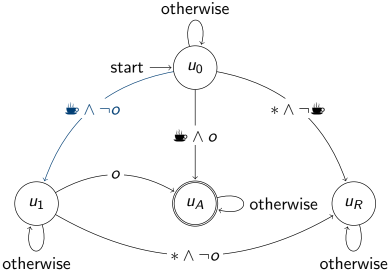
Traces

- Execution trace: $\lambda = \langle \langle (4, 6), \perp, \perp \rangle, \leftarrow,$
- Observation trace: $\lambda_{L,\mathcal{O}} = \langle \{\},$

Subgoal Automata

A type of *deterministic* finite automaton that encodes the subgoals of an *episodic goal-oriented* task as *propositional logic formulas*.



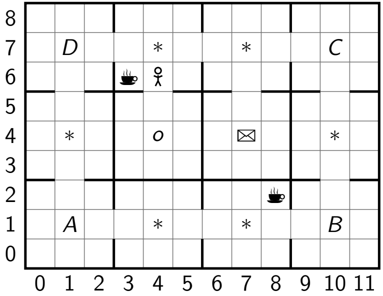


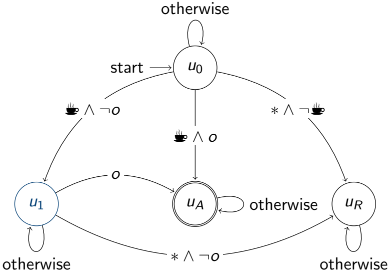
Traces

- Execution trace: $\lambda = \langle\langle(4,6), \bot, \bot\rangle, \leftarrow, 0, \langle(3,6), \bot, \bot\rangle,$
- Observation trace: $\lambda_{L,\mathcal{O}} = \langle\{\}, \{$☕$\},$

Subgoal Automata

A type of *deterministic* finite automaton that encodes the subgoals of an *episodic goal-oriented* task as *propositional logic formulas*.



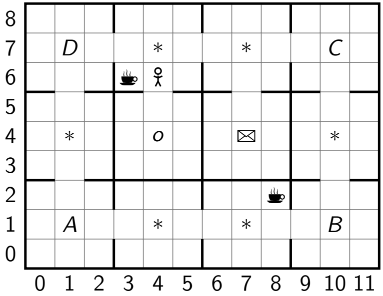


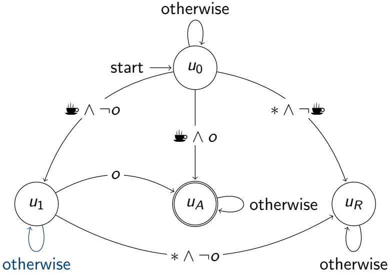
Traces

- Execution trace: $\lambda = \langle\langle(4,6),\bot,\bot\rangle, \leftarrow, 0, \langle(3,6),\bot,\bot\rangle, \rightarrow,$
- Observation trace: $\lambda_{L,\mathcal{O}} = \langle\{\}, \{☕\},$

Subgoal Automata

A type of *deterministic* finite automaton that encodes the subgoals of an *episodic goal-oriented* task as *propositional logic formulas*.



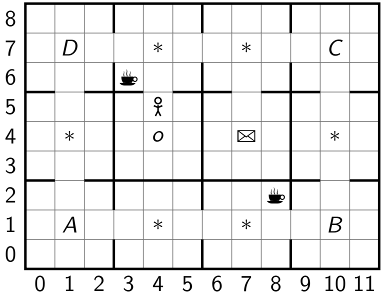


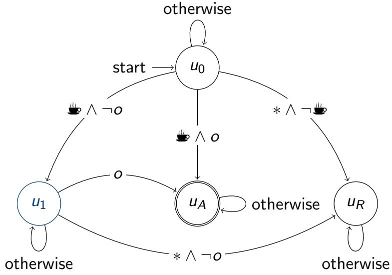
Traces

- Execution trace: $\lambda = \langle\langle(4,6),\perp,\perp\rangle, \leftarrow, 0, \langle(3,6),\perp,\perp\rangle, \rightarrow, 0, \langle(4,6),\perp,\perp\rangle,$
- Observation trace: $\lambda_{L,\mathcal{O}} = \langle\{\}, \{\text{☕}\}, \{\},$

Subgoal Automata

A type of *deterministic* finite automaton that encodes the subgoals of an *episodic goal-oriented* task as *propositional logic formulas*.



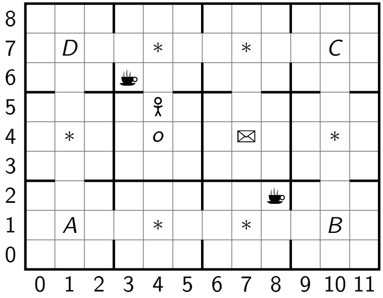


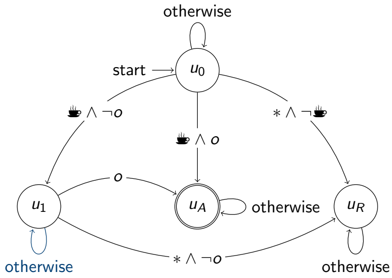
Traces

- Execution trace: $\lambda = \langle\langle(4,6),\perp,\perp\rangle, \leftarrow, 0, \langle(3,6),\perp,\perp\rangle, \rightarrow, 0, \langle(4,6),\perp,\perp\rangle, \downarrow,$
- Observation trace: $\lambda_{L,\mathcal{O}} = \langle\{\}, \{☕\}, \{\},$

Subgoal Automata

A type of *deterministic* finite automaton that encodes the subgoals of an *episodic goal-oriented* task as *propositional logic formulas*.



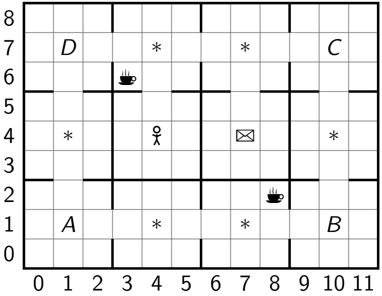


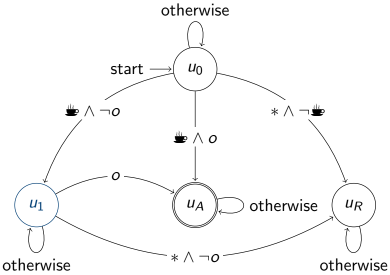
Traces

- Execution trace: $\lambda = \langle\langle(4,6),\perp,\perp\rangle, \leftarrow, 0, \langle(3,6),\perp,\perp\rangle, \rightarrow, 0, \langle(4,6),\perp,\perp\rangle, \downarrow, 0, \langle(4,5),\perp,\perp\rangle,$
- Observation trace: $\lambda_{L,\mathcal{O}} = \langle\{\}, \{☕\}, \{\}, \{\},$

Subgoal Automata

A type of *deterministic* finite automaton that encodes the subgoals of an *episodic goal-oriented* task as *propositional logic formulas*.



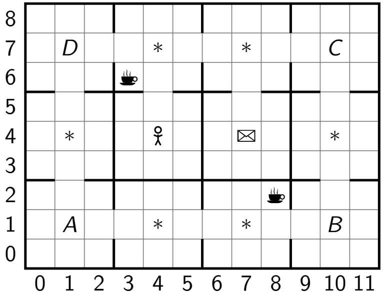


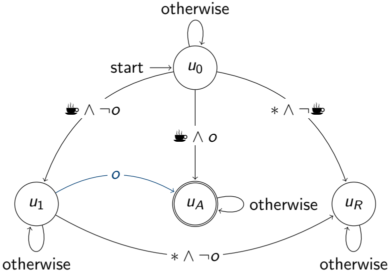
Traces

- Execution trace: $\lambda = \langle\langle(4,6),\bot,\bot\rangle, \leftarrow, 0, \langle(3,6),\bot,\bot\rangle, \rightarrow, 0, \langle(4,6),\bot,\bot\rangle,$
  $\downarrow, 0, \langle(4,5),\bot,\bot\rangle, \downarrow,$
- Observation trace: $\lambda_{L,\mathcal{O}} = \langle\{\}, \{$☕$\}, \{\}, \{\},$

Subgoal Automata

A type of *deterministic* finite automaton that encodes the subgoals of an *episodic goal-oriented* task as *propositional logic formulas*.



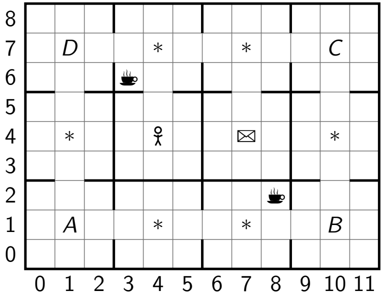


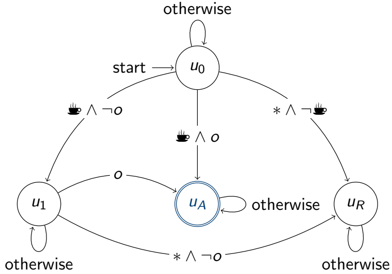
Traces

- Execution trace: $\lambda = \langle \langle (4,6), \bot, \bot \rangle, \leftarrow, 0, \langle (3,6), \bot, \bot \rangle, \rightarrow, 0, \langle (4,6), \bot, \bot \rangle, \downarrow, 0, \langle (4,5), \bot, \bot \rangle, \downarrow, 1, \langle (4,4), \top, \top \rangle \rangle.$
- Observation trace: $\lambda_{L,\mathcal{O}} = \langle \{\}, \{\text{☕}\}, \{\}, \{\}, \{o\} \rangle.$

Subgoal Automata

A type of *deterministic* finite automaton that encodes the subgoals of an *episodic goal-oriented* task as *propositional logic formulas*.





Traces

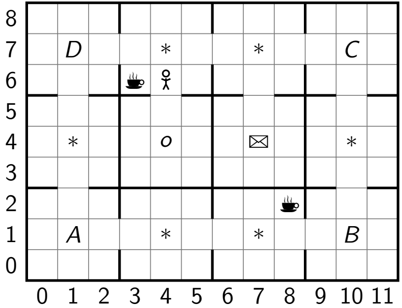
- Execution trace: $\lambda = \langle \langle (4,6), \bot, \bot \rangle, \leftarrow, 0, \langle (3,6), \bot, \bot \rangle, \rightarrow, 0, \langle (4,6), \bot, \bot \rangle, \downarrow, 0, \langle (4,5), \bot, \bot \rangle, \downarrow, 1, \langle (4,4), \top, \top \rangle \rangle.$
- Observation trace: $\lambda_{L,\mathcal{O}} = \langle \{\}, \{\text{☕}\}, \{\}, \{\}, \{o\} \rangle.$

Learning Subgoal Automata from Traces I

Traces

An execution trace $\lambda = \langle \boldsymbol{\sigma}_0, a_0, r_1, \ldots, r_n, \boldsymbol{\sigma}_n \rangle$ can be of three types:

- goal if $\sigma_n^G = \top$ (i.e., the latent state s_n is a goal state).
- dead-end if $\sigma_n^T = \top \wedge \sigma_n^G = \bot$ (i.e., the latent state s_n is a dead-end state).
- incomplete if $\sigma_n^T = \bot$ (i.e., the latent state s_n is not terminal).



Examples

Goal

- $\lambda^G = \langle \langle (4,6), \bot, \bot \rangle, \leftarrow, 0, \langle (3,6), \bot, \bot \rangle, \rightarrow, 0, \langle (4,6), \bot, \bot \rangle, \downarrow, 0, \langle (4,5), \bot, \bot \rangle, \downarrow, 1, \langle (4,4), \top, \top \rangle \rangle$.
- $\lambda^G_{L,\mathcal{O}} = \langle \{\}, \{☕\}, \{\}, \{\}, \{o\} \rangle$.

Dead-end

- $\lambda^D = \langle \langle (4,6), \bot, \bot \rangle, \uparrow, 0, \langle (4,7), \top, \bot \rangle \rangle$.
- $\lambda^D_{L,\mathcal{O}} = \langle \{\}, \{*\} \rangle$.

Learning Subgoal Automata from Traces II

Learning Task

Input

- A set of states $U \supseteq \{u_0, u_A, u_R\}$.
- A set of observables $\mathcal{O}$.
- A set of traces $\Lambda^G_{L,\mathcal{O}} \cup \Lambda^D_{L,\mathcal{O}} \cup \Lambda^I_{L,\mathcal{O}}$.
- A max. number of edges κ between states.

Output

The automaton's transition function s.t. it:

1. accepts all goal traces $\Lambda^G_{L,\mathcal{O}}$;
2. rejects all dead-end traces $\Lambda^D_{L,\mathcal{O}}$; and
3. neither accepts nor rejects incomplete traces $\Lambda^I_{L,\mathcal{O}}$.

- Subgoal automata are represented using *Answer Set Programming (ASP)*.
- The automaton learning task is described as an *ILASP task* [Law et al., 2015].
- Example:

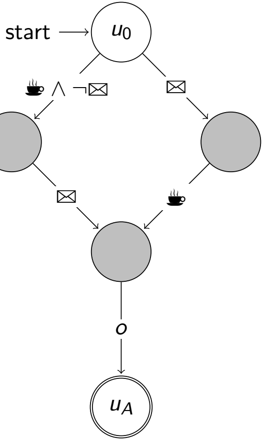
$$u_0 \xrightarrow{\text{☕} \wedge \neg o} u_1 \Rightarrow \delta(\underbrace{u_0}_{\textit{from}}, \underbrace{u_1}_{\textit{to}}, \underbrace{1}_{\textit{edge id}}, \mathtt{T})\text{:-}\underbrace{\mathtt{obs}(\text{☕}, \mathtt{T}), \mathtt{not\ obs}(o, \mathtt{T})}_{\textit{condition}}, \mathtt{step}(\mathtt{T}).$$

$$\langle \{\}, \{\text{☕}\}, \{\}, \{\}, \{o\} \rangle \Rightarrow \{\mathtt{obs}(\text{☕}, 1).\ \mathtt{obs}(o, 4).\}$$

Learning Subgoal Automata from Traces III

Symmetry Breaking

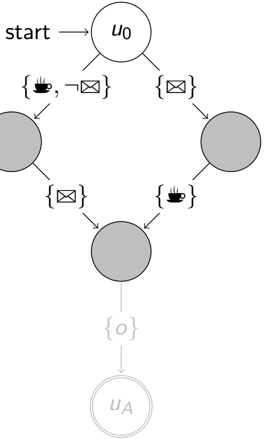
- Encode rules that impose a unique *BFS traversal*.
- Example for CoffeeMail:



Learning Subgoal Automata from Traces III

Symmetry Breaking

- Encode rules that impose a unique *BFS traversal*.
- Example for CoffeeMail:

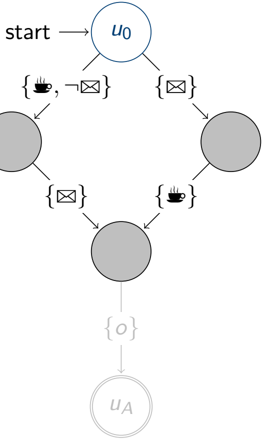


- The formulas are interpreted as comparable sets of labels.
- The accepting state is excluded: it already has a fixed name.

Learning Subgoal Automata from Traces III

Symmetry Breaking

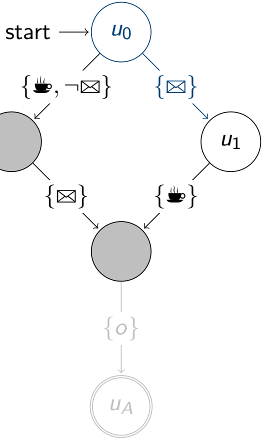
- Encode rules that impose a unique *BFS traversal*.
- Example for CoffeeMail:



Learning Subgoal Automata from Traces III

Symmetry Breaking

- Encode rules that impose a unique *BFS traversal*.
- Example for CoffeeMail:

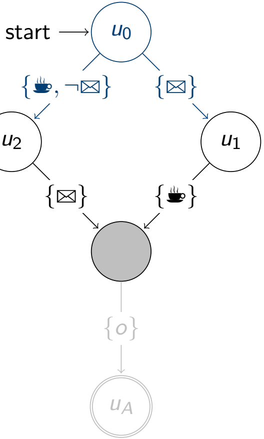


- Assume {✉} is lower than {☕, ¬✉}. So we choose the edge labeled with the former first.
- The next available id is given to the shaded state.

Learning Subgoal Automata from Traces III

Symmetry Breaking

- Encode rules that impose a unique *BFS traversal*.
- Example for CoffeeMail:

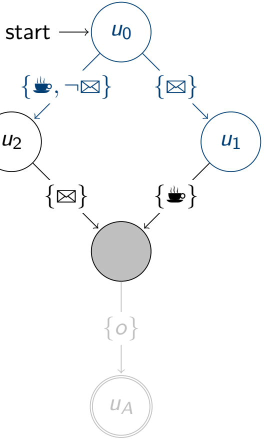


- We continue selecting the lowest edges from the state with the lowest id.

Learning Subgoal Automata from Traces III

Symmetry Breaking

- Encode rules that impose a unique *BFS traversal*.
- Example for CoffeeMail:

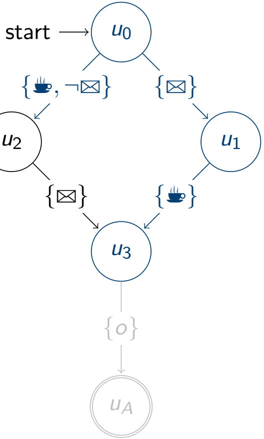


- We continue selecting the lowest edges from the state with the lowest id.

Learning Subgoal Automata from Traces III

Symmetry Breaking

- Encode rules that impose a unique *BFS traversal*.
- Example for CoffeeMail:

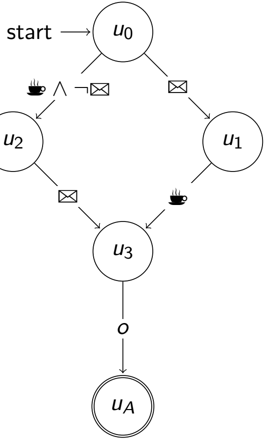


- We continue selecting the lowest edges from the state with the lowest id.

Reinforcement Learning Algorithms

HRL (Hierarchical RL) using the options framework [Sutton et al., 1999]

- There are two *decision levels*:
  1. From a given automaton state, choose a subgoal to pursue.
     Example: From u_0, choose between ☕ $\wedge \neg$✉ and ✉.
  2. Given a chosen subgoal, choose an action.
     Example: Choose an action (up, down, left or right) to reach the subgoal ☕ $\wedge \neg$✉.
- The policy can be *suboptimal*.



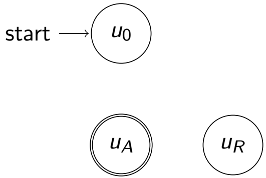
Interleaved Automaton Learning Algorithm I

The RL and automata learning processes are *interleaved*:

- The *initial automaton* does not accept nor reject anything.
- The automaton learner runs when a *counterexample* is found.
  Example: the current state is a goal state and the current automaton state is not the accepting state (u_A).
- The number of states of the automaton is increased when the automaton learning task becomes unsatisfiable.
  $\implies$ The *minimal* automaton is found for a particular value of κ.

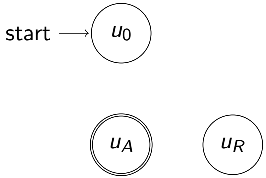
Interleaved Automaton Learning Algorithm II

Example – Coffee (loops are omitted)



Interleaved Automaton Learning Algorithm II

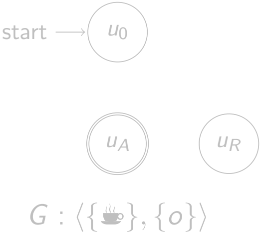
Example – COFFEE (loops are omitted)

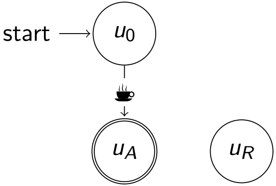


$G : \langle \{ ☕ \}, \{ o \} \rangle$

Interleaved Automaton Learning Algorithm II

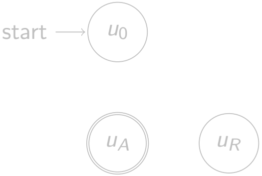
Example – COFFEE (loops are omitted)



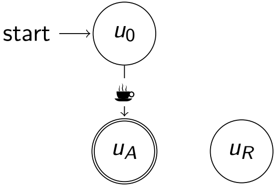


Interleaved Automaton Learning Algorithm II

Example – COFFEE (loops are omitted)



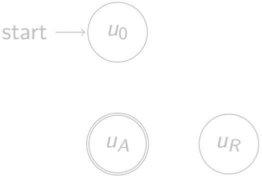
$G : \langle\{☕\}, \{o\}\rangle$



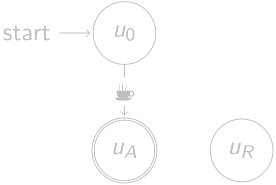
$D : \langle\{*\}\rangle$

Interleaved Automaton Learning Algorithm II

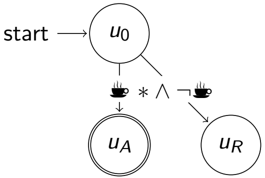
Example – Coffee (loops are omitted)



$G : \langle \{☕\}, \{o\} \rangle$

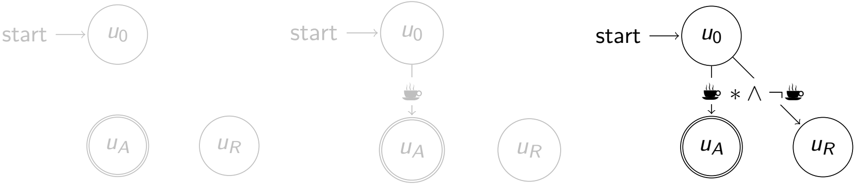


$D : \langle \{*\} \rangle$



Interleaved Automaton Learning Algorithm II

Example – Coffee (loops are omitted)



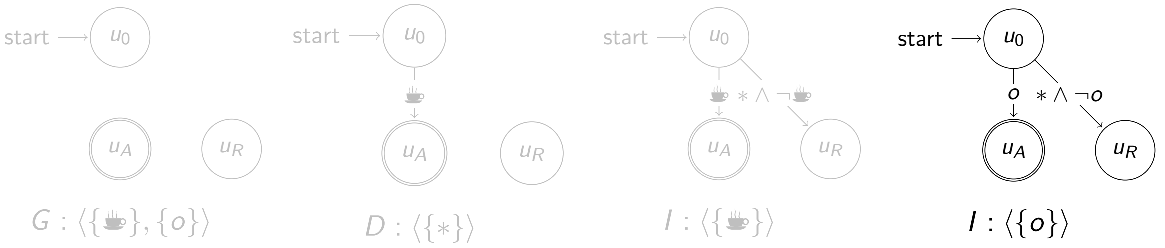
$G : \langle\{$☕$\}, \{o\}\rangle$ $D : \langle\{*\}\rangle$ $I : \langle\{$☕$\}\rangle$

Interleaved Automaton Learning Algorithm II

Example – Coffee (loops are omitted)

Interleaved Automaton Learning Algorithm II

Example – Coffee (loops are omitted)



Interleaved Automaton Learning Algorithm II

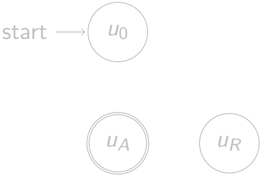
Example – Coffee (loops are omitted)

UNSATISFIABLE,

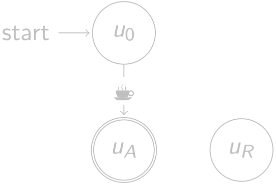
$U = \{u_0, u_1, u_A, u_R\}$

Interleaved Automaton Learning Algorithm II

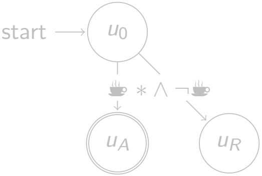
Example – Coffee (loops are omitted)



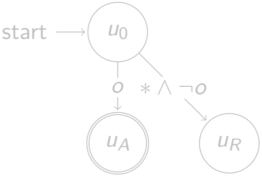
$G : \langle\{☕\}, \{o\}\rangle$



$D : \langle\{*\}\rangle$

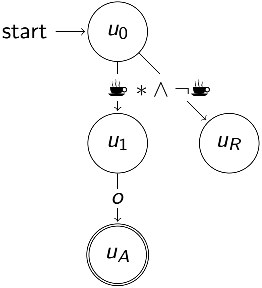


$I : \langle\{☕\}\rangle$



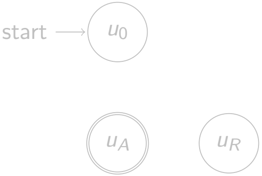
$I : \langle\{o\}\rangle$

UNSATISFIABLE,
$U = \{u_0, u_1, u_A, u_R\}$

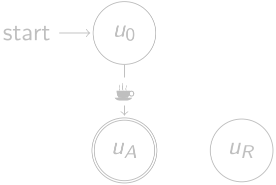


Interleaved Automaton Learning Algorithm II

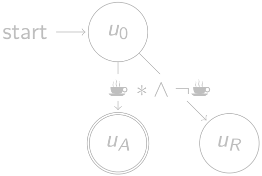
Example – Coffee (loops are omitted)



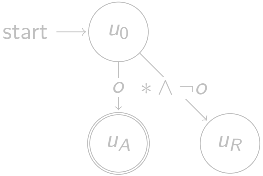
$G : \langle \{☕\}, \{o\} \rangle$



$D : \langle \{*\} \rangle$

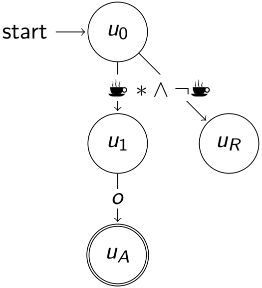


$I : \langle \{☕\} \rangle$



$I : \langle \{o\} \rangle$

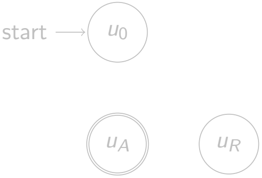
UNSATISFIABLE,
$U = \{u_0, u_1, u_A, u_R\}$



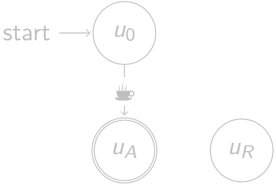
$D : \langle \{☕\}, \{*\} \rangle$

Interleaved Automaton Learning Algorithm II

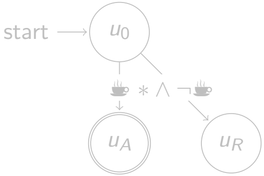
Example – COFFEE (loops are omitted)



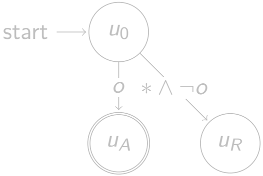
$G : \langle\{☕\}, \{o\}\rangle$



$D : \langle\{*\}\rangle$

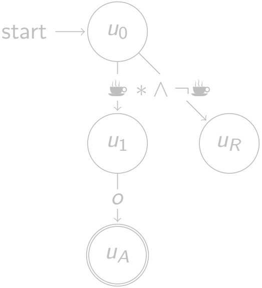


$I : \langle\{☕\}\rangle$

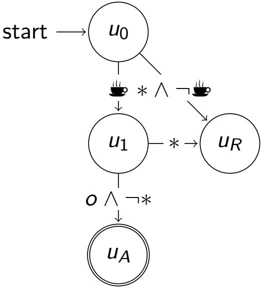


$I : \langle\{o\}\rangle$

UNSATISFIABLE,
$U = \{u_0, u_1, u_A, u_R\}$

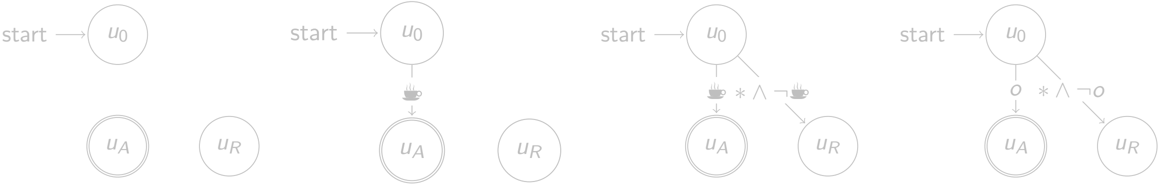


$D : \langle\{☕\}, \{*\}\rangle$



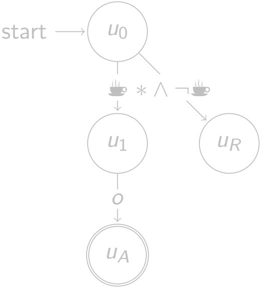
Interleaved Automaton Learning Algorithm II

Example – COFFEE (loops are omitted)

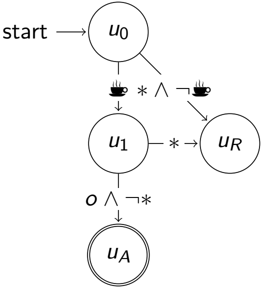


$G : \langle\{$☕$\}, \{o\}\rangle$ $D : \langle\{*\}\rangle$ $I : \langle\{$☕$\}\rangle$ $I : \langle\{o\}\rangle$

UNSATISFIABLE,
$U = \{u_0, u_1, u_A, u_R\}$



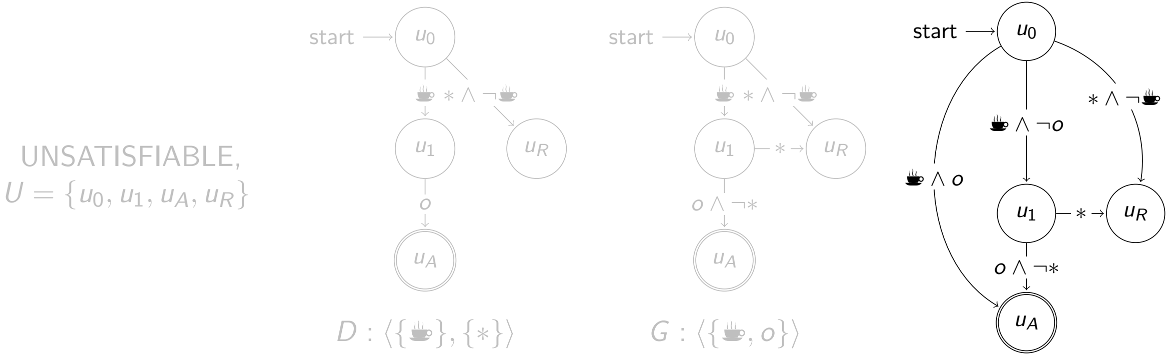
$D : \langle\{$☕$\}, \{*\}\rangle$



$G : \langle\{$☕$, o\}\rangle$

Interleaved Automaton Learning Algorithm II

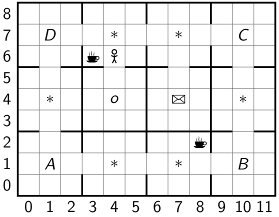
Example – COFFEE (loops are omitted)



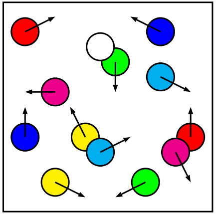
Experimental Results for HRL I

Setting

- Domains:
  - OfficeWorld - Discrete state space - Tasks: Coffee, CoffeeMail, VisitABCD.
  - WaterWorld - Continuous state space - Tasks: RGB, RG-B, RGBC.
- Algorithms:
  - HRL: standard version giving a reward of $+1$ for achieving subgoals.
  - HRL_G: HRL but with penalties for reaching dead-ends and after each step.
- ILASP2 is used to learn the automata.



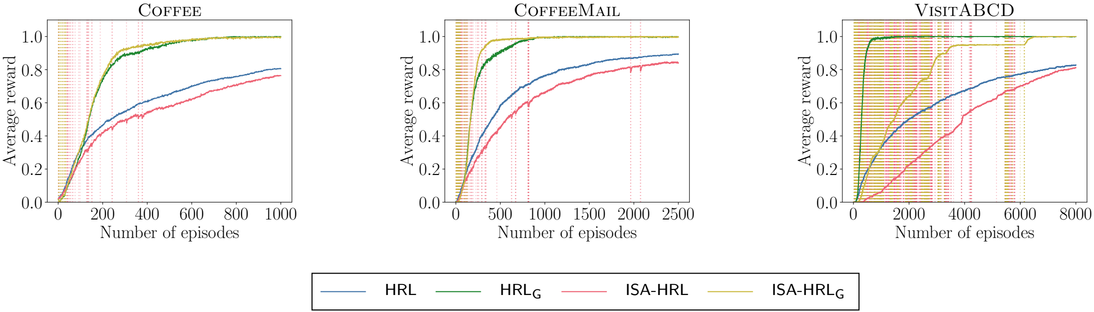
The OfficeWorld environment [Toro Icarte et al., 2018]



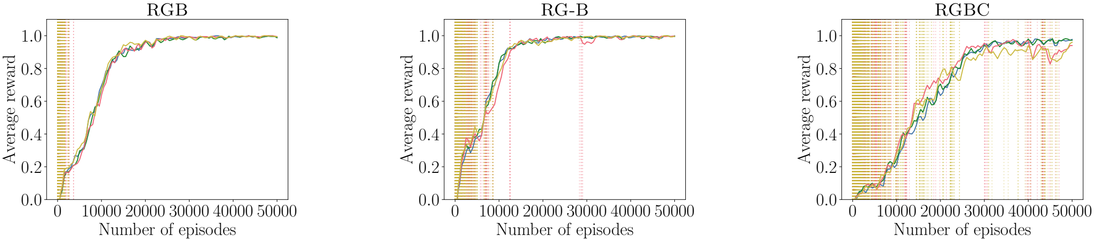
The WaterWorld environment [Toro Icarte et al., 2018]

Experimental Results for HRL II

OfficeWorld



WaterWorld



Experimental Results for HRL III

Automaton Learning Running Time

- By default: symmetry breaking enabled, acyclicity enforced, $\kappa = 1$, and trace compression.
- More subgoals, more states $\implies$ More running time, more examples and longer examples.
- The number of goal examples $\simeq$ the number of paths from the initial state to the accepting state.

	Time (s.)	# Examples				Example Length
		All	G	D	I	
Coffee	0.4 (0.0)	8.7 (0.4)	2.4 (0.1)	3.0 (0.1)	3.2 (0.3)	2.8 (2.1)
CoffeeMail	18.9 (3.3)	29.0 (1.5)	3.9 (0.3)	9.3 (0.6)	15.8 (1.0)	4.0 (2.6)
VisitABCD	163.2 (44.3)	54.9 (3.8)	1.6 (0.1)	15.2 (0.9)	38.1 (3.1)	5.5 (3.1)

Table 1: Automaton learning statistics for the OfficeWorld tasks using HRL_G.

Experimental Results for HRL IV

Impact of the Automaton Learning Parameters

- Relaxing the constraints (e.g., no symmetry breaking, allow cycles) increases the automaton learning running time.

	Acyclic		Cyclic	
	No SB	SB	No SB	SB
Coffee	0.5 (0.0)	0.4 (0.0)	0.5 (0.0)	0.5 (0.0)
CoffeeMail	277.4 (70.2)	18.9 (3.3)	4204.3 (1334.4)*	774.7 (434.4)
VisitABCD	1070.0 (725.6)	163.2 (44.3)	3293.5 (1199.2)*	1961.7 (1123.8)

Table 2: Total automaton learning time when symmetry breaking is disabled (No SB) and enabled (SB). $*$ = timed out between 1 and 10 runs out of 20 runs.

Conclusions

- Method that interleaves the learning and exploitation of an automaton whose edges encode the subgoals of an episodic goal-oriented task.
- The automata are learnt using a state-of-the-art inductive logic programming system from traces observed by the agent.
- The automaton structure is exploited with existing RL algorithms.

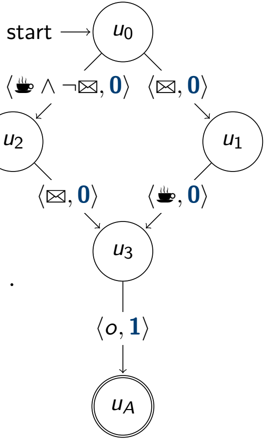
Supplementary Slides I

Another RL Algorithm - QRM (Q-Learning for Reward Machines) [Toro Icarte et al., 2018]

- Requires "transforming" the subgoal automaton into an RM by assuming we know the POMDP's *reward function*.
- Equivalent to *learning a policy over $\Sigma \times U$*:
  - Each automaton state $u_i \in U$ has its own Q-function.
  - These Q-functions are coupled through Q-learning updates:

$$Q_{u}(\sigma_t^{\Sigma}, a) = Q_u(\sigma_t^{\Sigma}, a) + \alpha \left(r(u, u') + \gamma \max_{a'} Q_{u'}(\sigma_{t+1}^{\Sigma}, a') - Q_u(\sigma_t^{\Sigma}, a) \right).$$

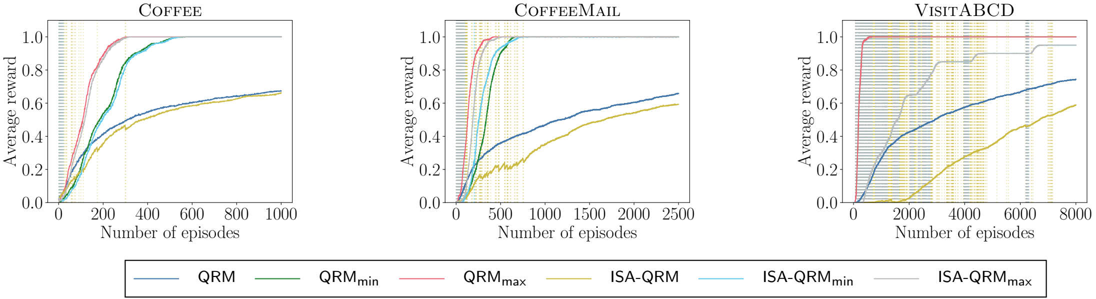
- The policies choose actions in order to reach a goal state.
- Guarantees *optimality* in the tabular case.



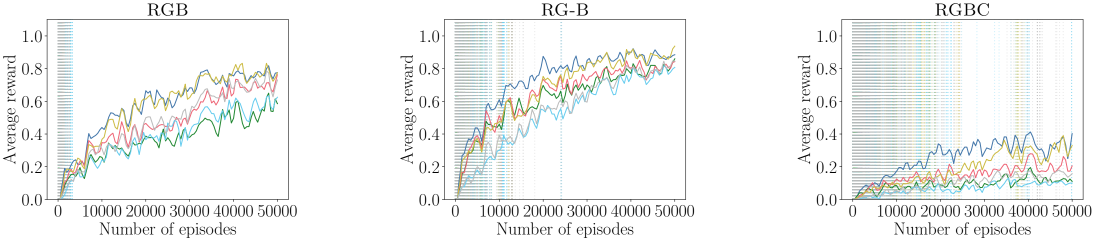
Supplementary Slides II

Experimental Results for QRM

OfficeWorld



WaterWorld



Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013).
The Arcade Learning Environment: An Evaluation Platform for General Agents.
Journal of Artificial Intelligence Research, 47:253–279.

Beyret, B., Hernández-Orallo, J., Cheke, L., Halina, M., Shanahan, M., and Crosby, M. (2019).
The Animal-AI Environment: Training and Testing Animal-Like Artificial Cognition.

Bonet, B., Palacios, H., and Geffner, H. (2009).
Automatic Derivation of Memoryless Policies and Finite-State Controllers Using Classical Planners.
In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Brooks, R. A. (1989).
A Robot that Walks; Emergent Behaviors from a Carefully Evolved Network.
Neural Computation, 1(2):253–262.

Buckland, M. (2004).
AI Game Programming by Example.
Wordware Publishing Inc.

Hu, Y. and De Giacomo, G. (2011).
Generalized Planning: Synthesizing Plans that Work for Multiple Environments.
In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 918–923.

Koul, A., Fern, A., and Greydanus, S. (2019).
Learning Finite State Representations of Recurrent Policy Networks.
In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Küttler, H., Nardelli, N., Miller, A. H., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. (2020).
The NetHack Learning Environment.
In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*.

Law, M., Russo, A., and Broda, K. (2015).
The ILASP System for Learning Answer Set Programs.

Leonetti, M., Iocchi, L., and Patrizi, F. (2012).
Automatic Generation and Learning of Finite-State Controllers.
In *Proceedings of the International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA)*, pages 135–144.

Meuleau, N., Peshkin, L., Kim, K., and Kaelbling, L. P. (1999).
Learning Finite-State Controllers for Partially Observable Environments.
In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 427–436.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015).
Human-level control through deep reinforcement learning.
Nature, 518(7540):529–533.

Parr, R. and Russell, S. J. (1997).
Reinforcement Learning with Hierarchies of Machines.
In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS) Conference*, pages 1043–1049.

Segovia Aguas, J., Jiménez, S., and Jonsson, A. (2018).
Computing Hierarchical Finite State Controllers with Classical Planning.
J. Artif. Intell. Res., 62:755–797.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018).
A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play.
Science, 362(6419):1140–1144.

Sutton, R. S. and Barto, A. G. (1998).
Reinforcement Learning: An Introduction.
MIT Press.

Sutton, R. S., Precup, D., and Singh, S. P. (1999).
Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning.
Artif. Intell., 112(1-2):181–211.

Toro Icarte, R., Klassen, T. Q., Valenzano, R. A., and McIlraith, S. A. (2018).
Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning.
In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2112–2121.

Toro Icarte, R., Waldie, E., Klassen, T. Q., Valenzano, R. A., Castro, M. P., and McIlraith, S. A. (2019).
Learning Reward Machines for Partially Observable Reinforcement Learning.
In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS) Conference*, pages 15497–15508.