Structured Representations for Knowledge Transfer in Reinforcement Learning

Benjamin Rosman

Mobile Intelligent Autonomous Systems
Council for Scientific and Industrial Research
&
School of Computer Science and Applied Maths
University of the Witwatersrand
South Africa

RAIL LAB

our future through science

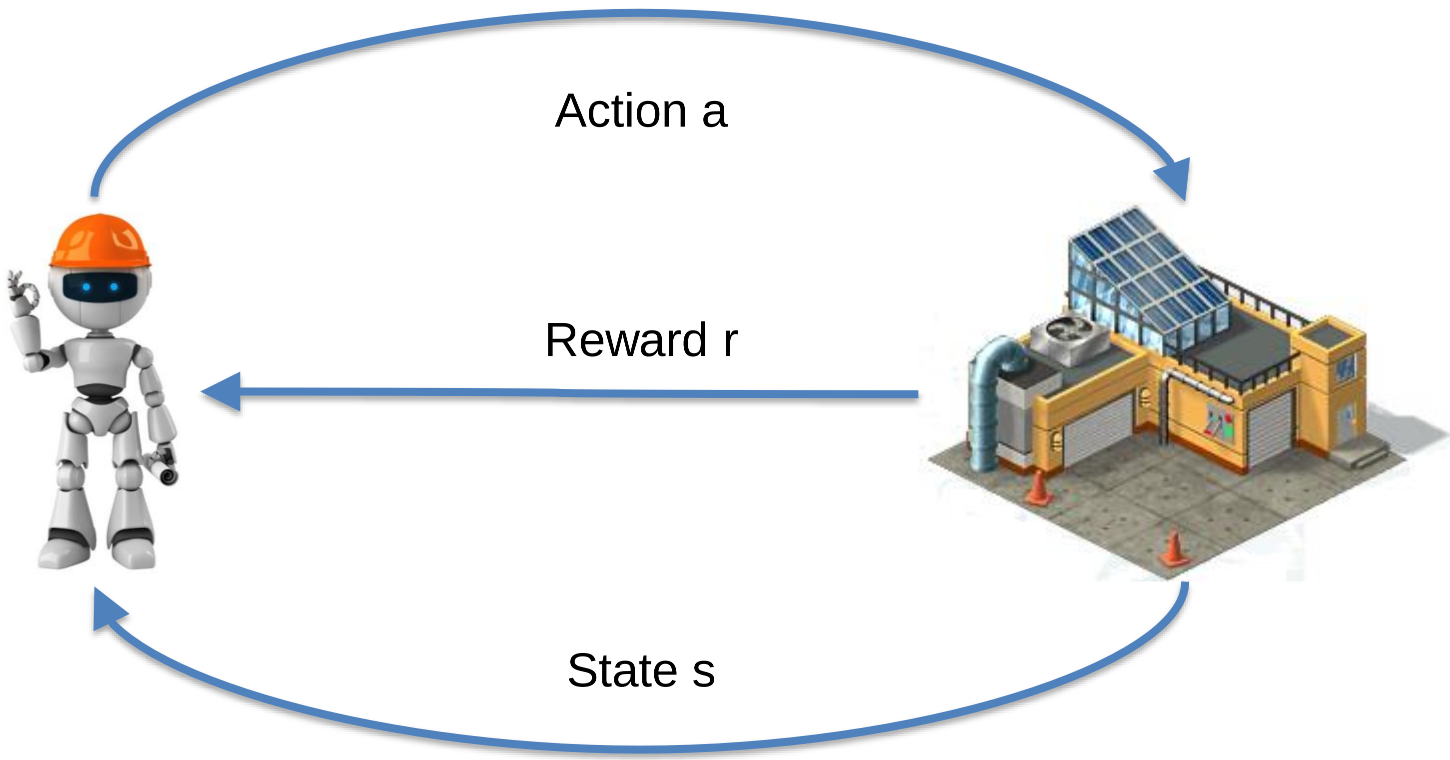
Robots solving complex tasks

Large high-dimensional action and state spaces

Many different task instances

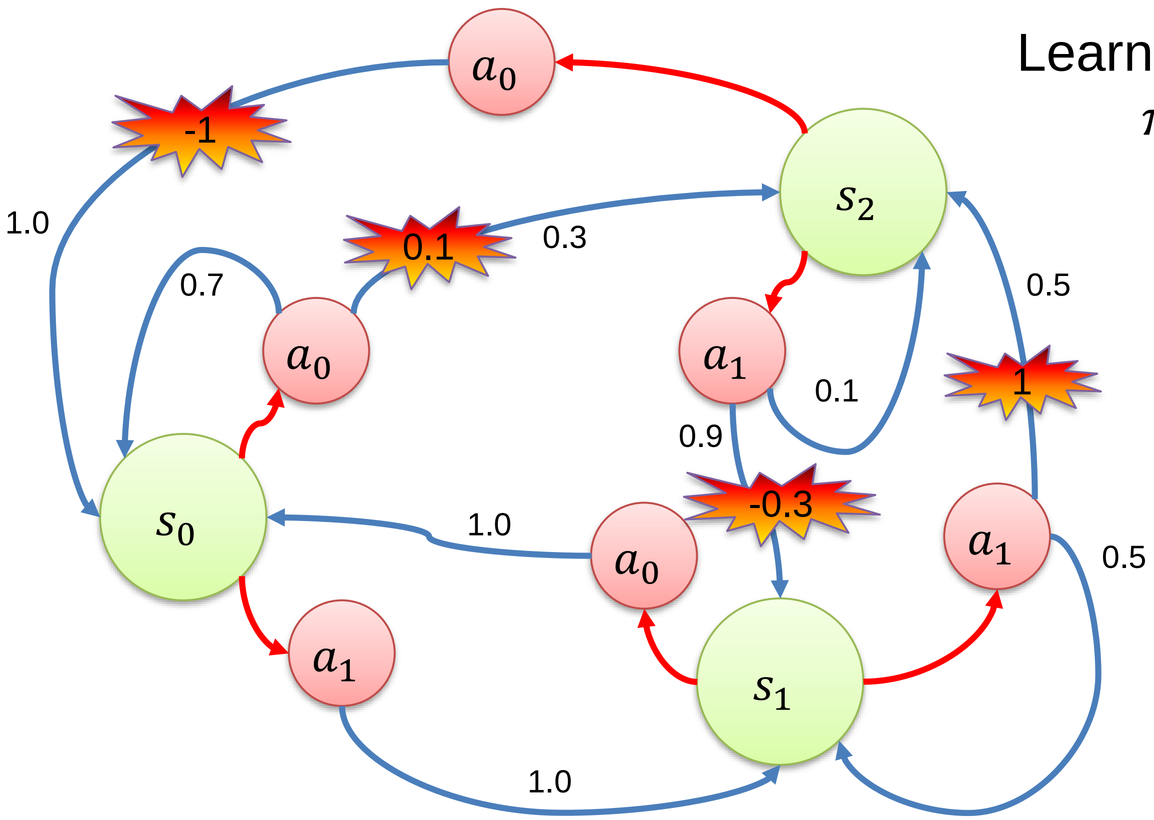
Behaviour learning

- Reinforcement learning (RL)



Markov decision process (MDP)

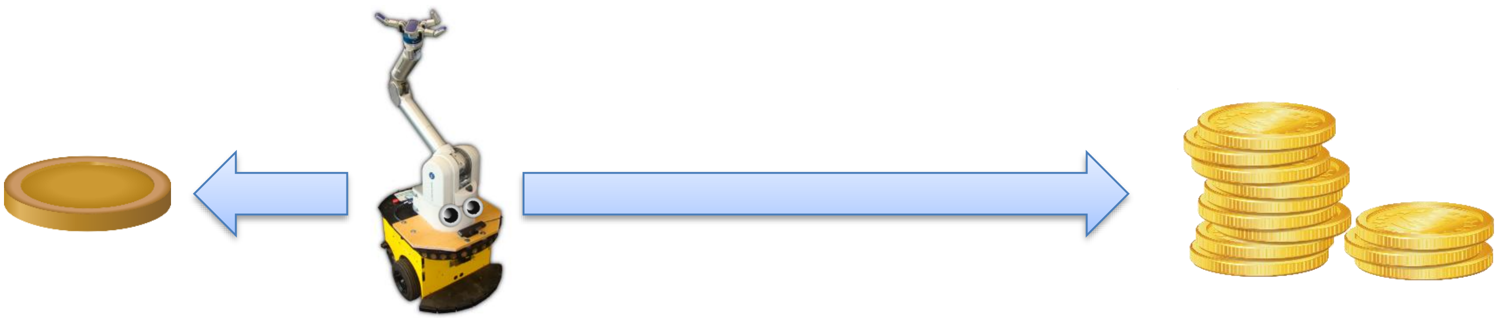
- $M = \langle S, A, T, R \rangle$



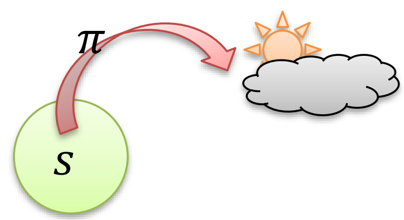
Learn optimal policy:
$\pi^* : S \rightarrow A$

Looking into the future

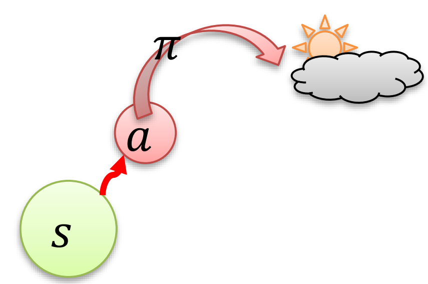
- Can't just rely on immediate rewards



- Define **value functions**:
    - $V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\}$



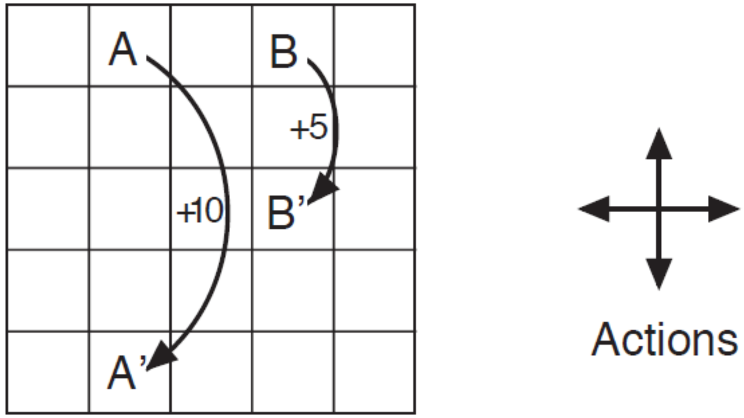
    - $Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\}$



    - V* (Q*) is a proxy for π*

Value functions example

- Random policy:

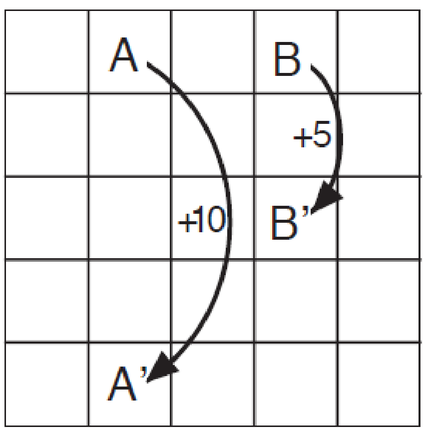


(a)

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

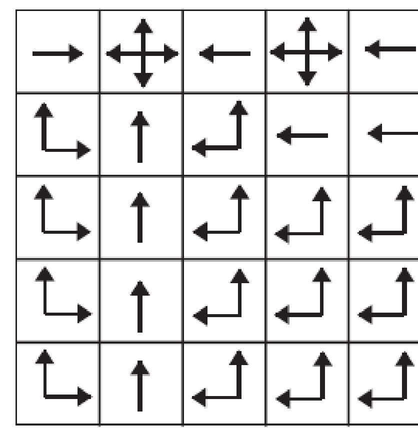
- Optimal:



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*



c) π_*

RL algorithms

- So: solve a large system of nonlinear value function equations (Bellman equations)

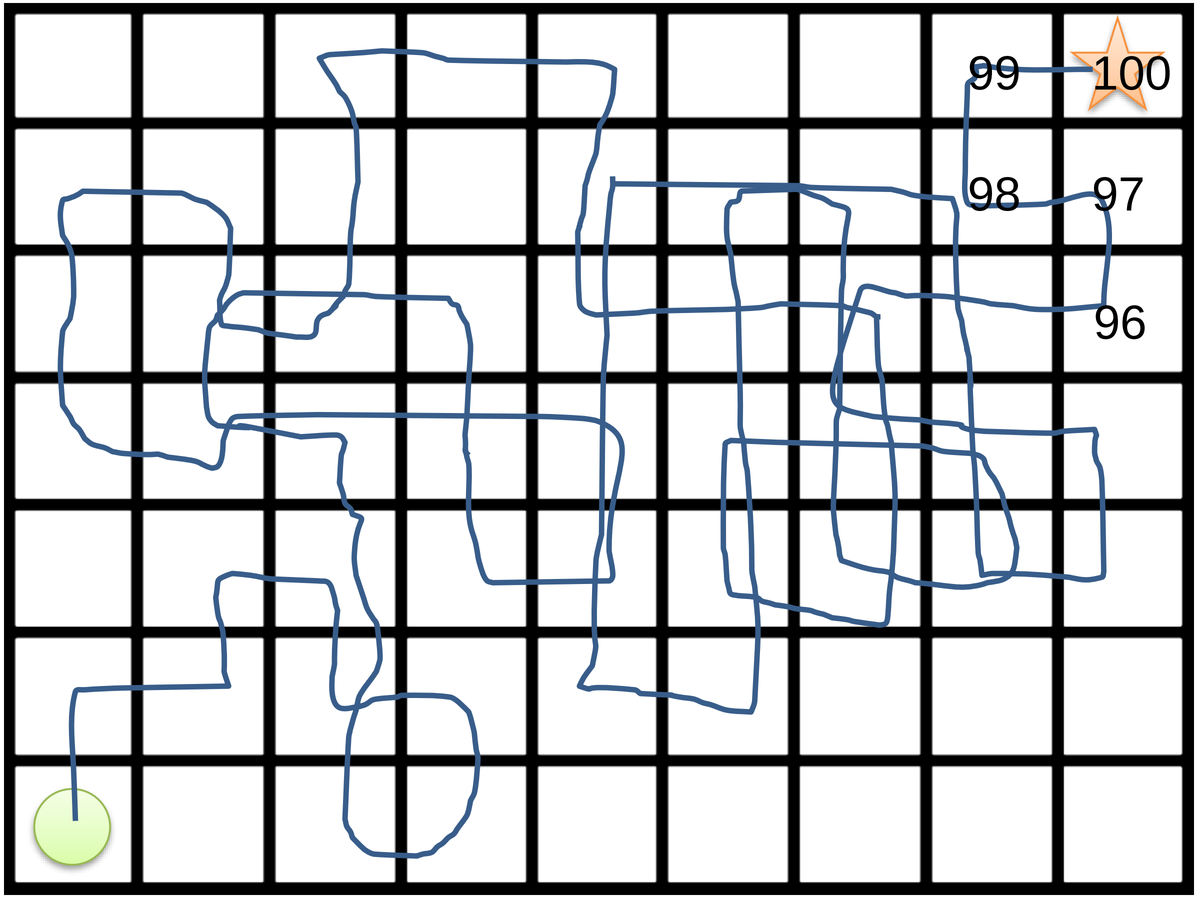
$$
\begin{aligned}
V^{\pi'}(s) &= \max_a E\left\{r_{t+1} + \gamma V^{\pi'}(s_{t+1}) \mid s_t = s, a_t = a\right\} \\
&= \max_a \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^{\pi'}(s')\right].
\end{aligned}
$$

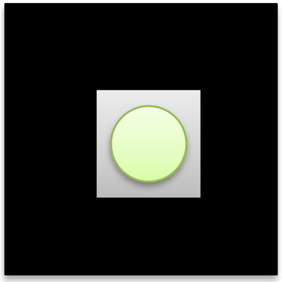
  - Optimal control problem
- But: transitions P & rewards R aren't known!

- RL learning is **trial-and-error learning** to find an **optimal policy** from experience

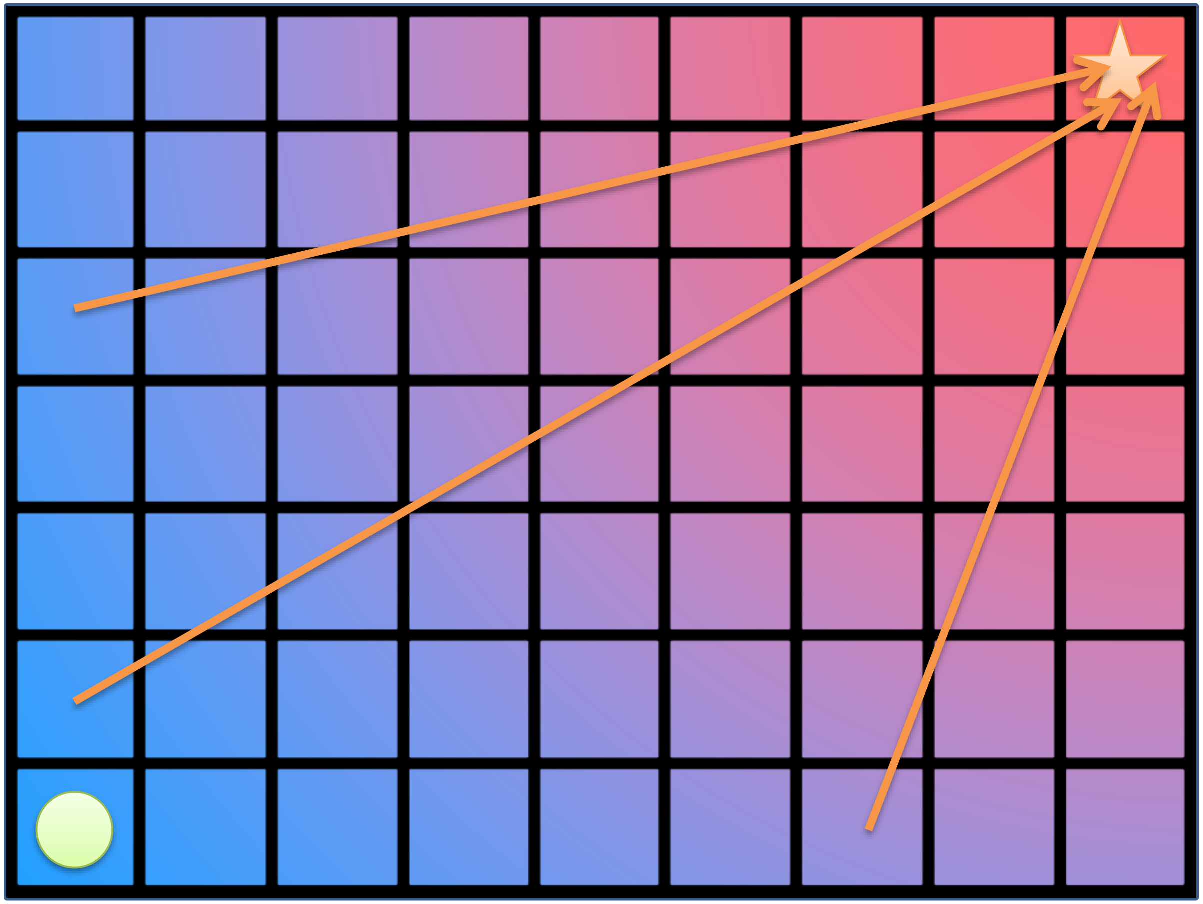
- Exploration vs exploitation

Exploring





Learned value function



An algorithm: Q-learning

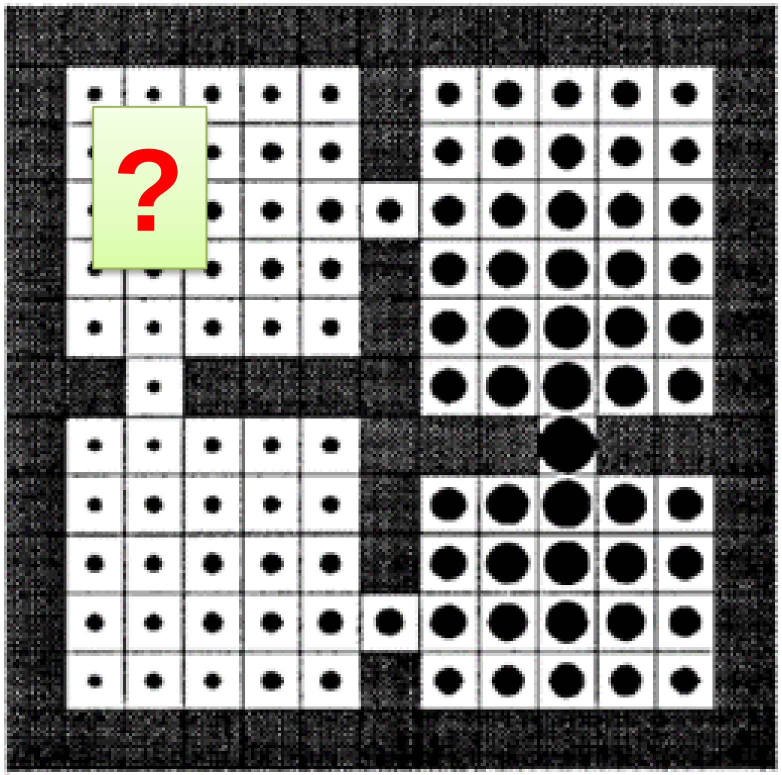
- Initialise $Q(s,a)$ arbitrarily
- Repeat (for each episode):
  - Initialise s
  - Repeat (for each step of episode):
    1. Choose a from s (ϵ-greedy policy from Q)
       - $a \leftarrow \begin{cases} \arg\max_a Q(s,a) & w.p.\ 1-\epsilon \\ random & w.p.\ \epsilon \end{cases}$ exploit / explore
    2. Take action a, observe $r,\ s'$
    3. Update estimate of Q
       - $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$ learn
    - $s \leftarrow s'$
  - Until s is terminal

immediate reward

estimated future reward

Solving tasks

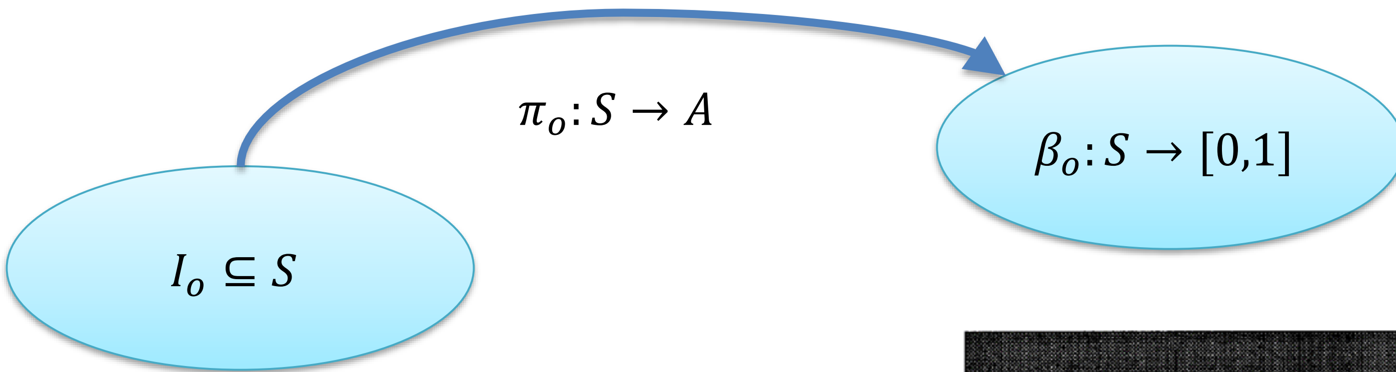
Generalising solutions?



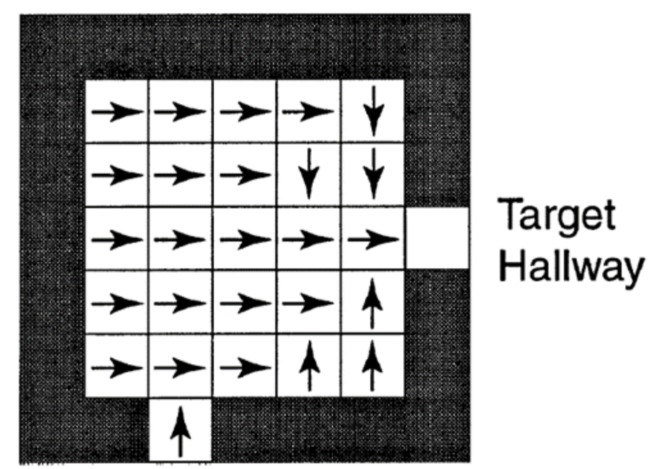
- How does this help us solve other problems?

Hierarchical RL

- Sub-behaviours: options $o = \langle I_o, \pi_o, \beta_o \rangle$
  - Policy + initiation and termination conditions



- Abstract away low level actions
- Does not affect the state space



Abstracting states

- **Aim**: learn an abstract representation of the environment
  - Use with task-level planners
  - Based on agent behaviours (skills / options)
  - General: don't need to be relearned for every new task

Steven James (in collaboration with George Konidaris)

S. James, B. Rosman, G. Konidaris. Learning to Plan with Portable Symbols. ICML/IJCAI/AAMAS 2018 Workshop on Planning and Learning, July 2018.

S. James, B. Rosman, G. Konidaris. Learning Portable Abstract Representations for High-Level Planning. *Under review.*

Requirements: planning with skills

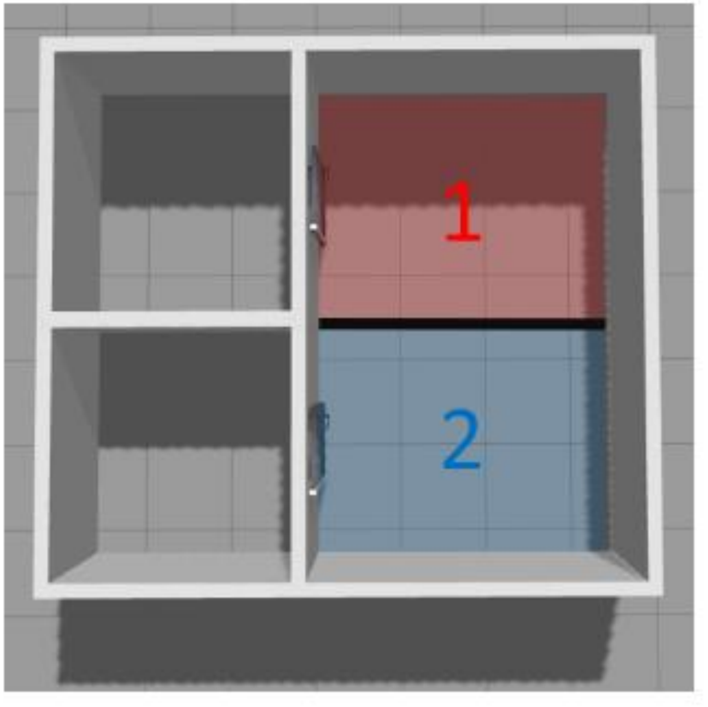
- Learn the preconditions
  - Classification problem:
    - $P(\text{can execute skill?} \mid \text{current\_state})$

"SYMBOLS"

- Learn the effects
  - Density estimation:
    - $P(\text{next\_state} \mid \text{current\_state}, \text{skill})$
  - Possible if options are subgoal i.e. $P(\text{next\_state} \mid \cancel{\text{current\_state}}, \text{skill})$

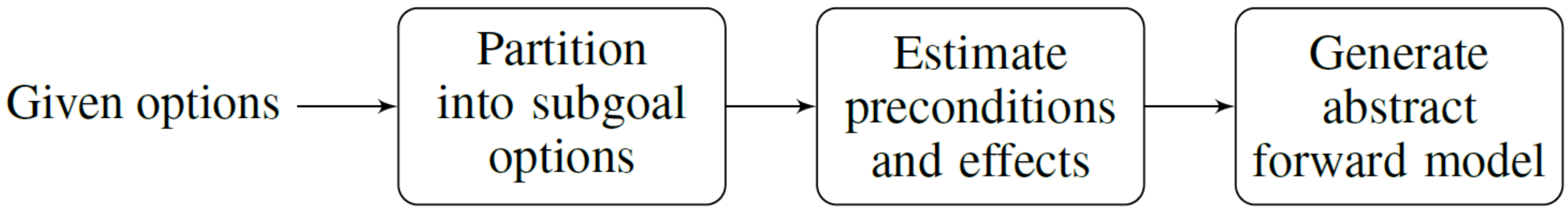
Subgoal options

- $P(\text{next\_state} \mid \cancel{\text{current\_state}}, \text{skill})$
- Partition skills to ensure property holds
  - e.g. "walk to nearest door"



Generating symbols from skills

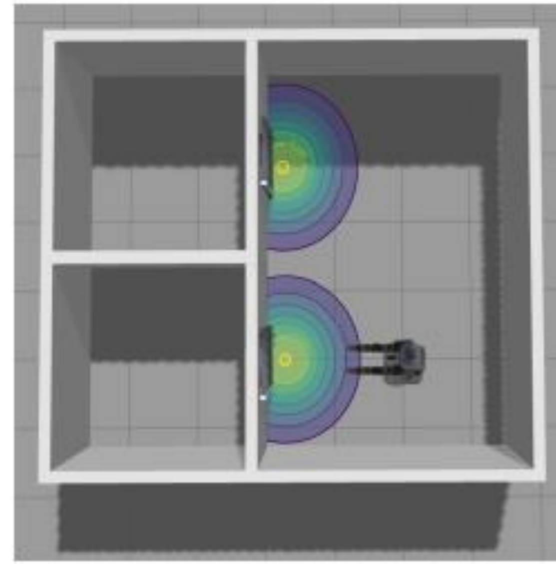
[Konidaris, 2018]



- Results in abstract MDP/propositional PPDDL
- But $P(s \in I_o)$ and $P(s' | o)$ are distributions/symbols over state space *particular to current task*
  - e.g. grounded in a specific set of xy-coordinates

Towards portability

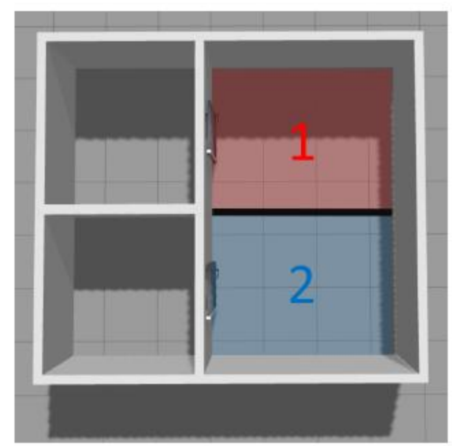
- Need a representation that facilitates transfer
- Assume agent has sensors which provide it with (lossy) observations
- Augment the state space with action-centric observations
  - **Agent space**
- e.g. robot navigating a building
  - State space: xy-coordinates
  - Agent space: video camera



Portable symbols

- Learning symbols in **agent space**
  - Portable!
  - But: non-Markov and insufficient for planning
- Add the subgoal **partition labels** to rules
  - General abstract symbols + grounding → portable rules

+



Grounding symbols

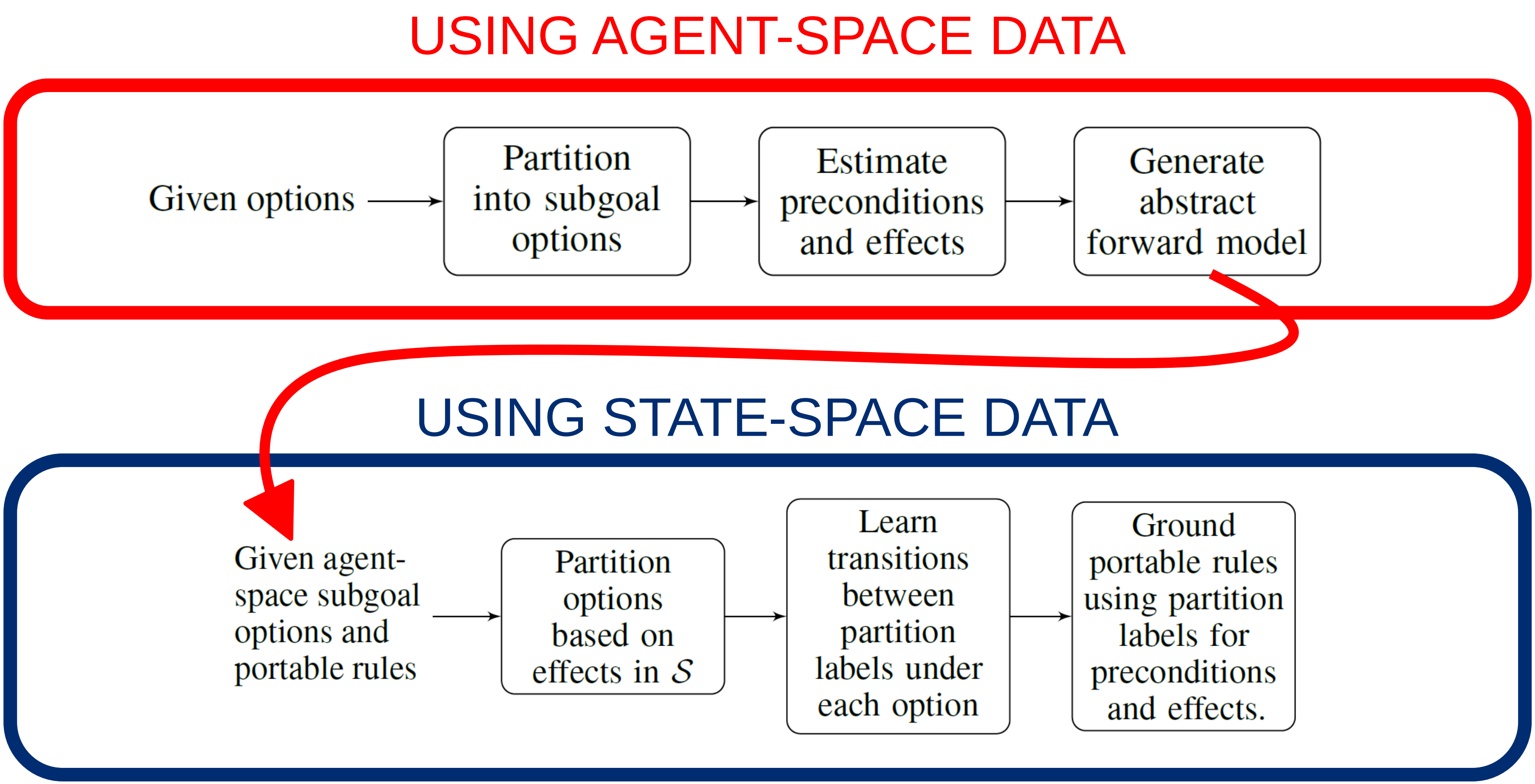
- Learn abstract symbols

- Learning linking functions:
  - Mapping partition numbers from options to their effects

- This gives us a factored MDP or a PPDDL representation

- Provably sufficient for planning

Learning grounded symbols



The treasure game

Agent and problem space

- State space: xy-position of agent, key and treasure, angle of levers and state of lock

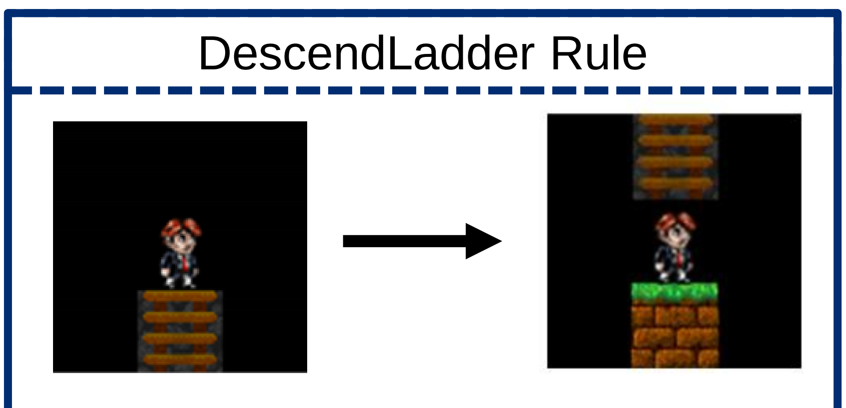
- Agent space: 9 adjacent cells about the agent

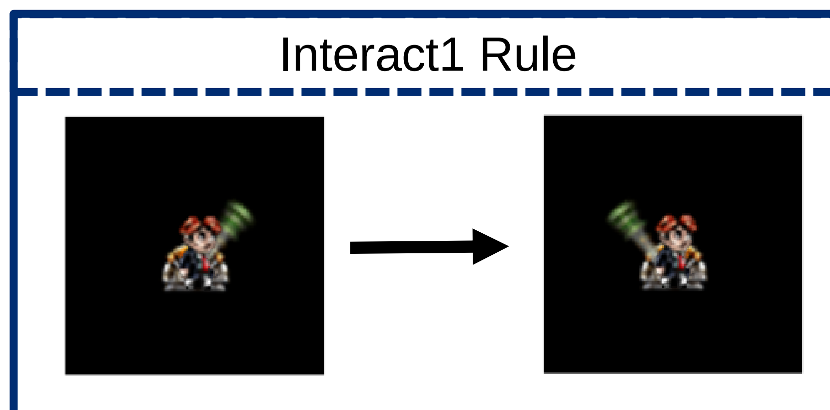
Skills

- Options:
  - `GoLeft, GoRight`
  - `JumpLeft, JumpRight`
  - `DownRight, DownLeft`
  - `Interact`
  - `ClimbLadder, DescendLadder`

Learning portable rules

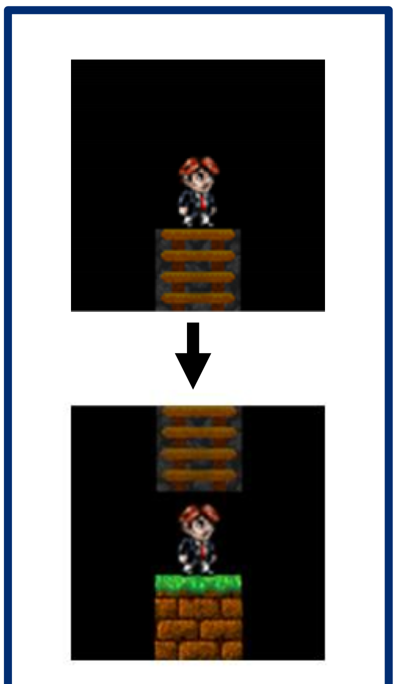
- Cluster to create subgoal agent-space options
- Use SVM and KDE to estimate preconditions and effects
- Learned rules can be transferred between tasks

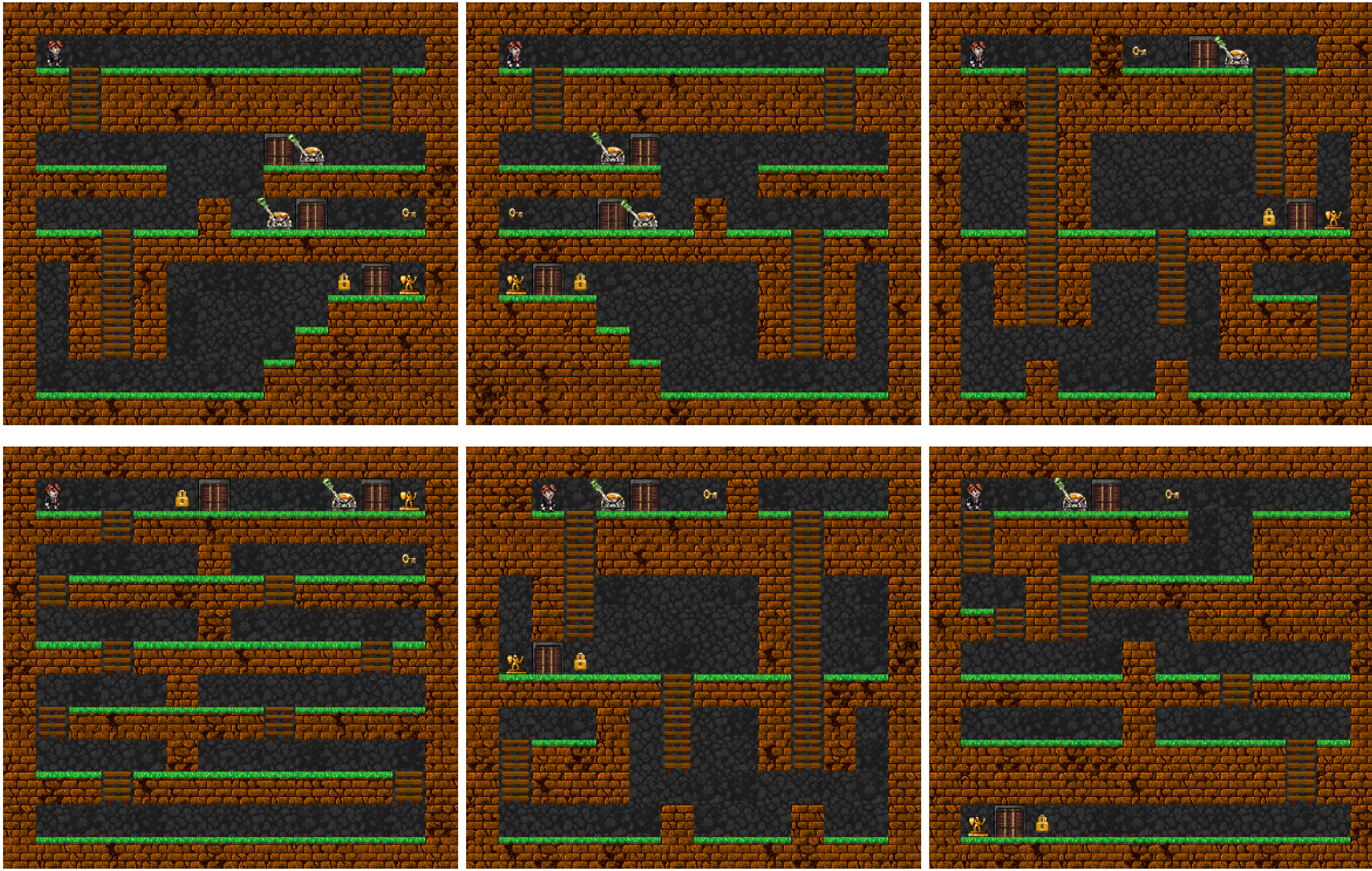




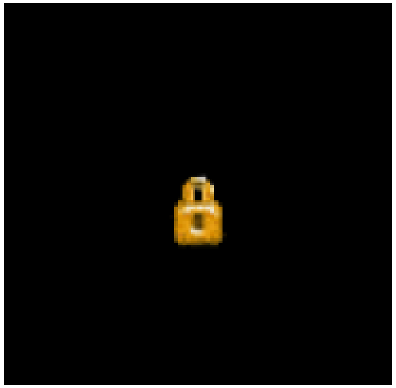
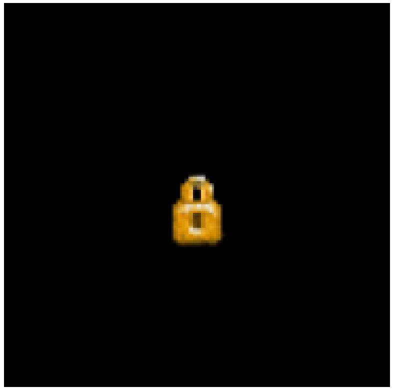
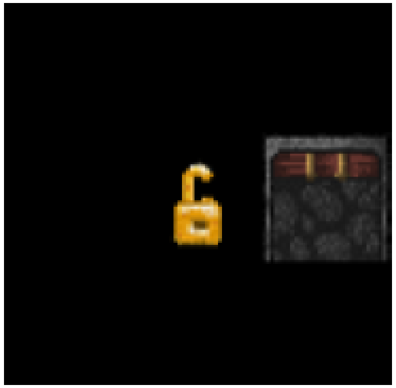
Grounding rules

- Partition options in state space to get partition numbers
  - Learn grounded rule instances: linking



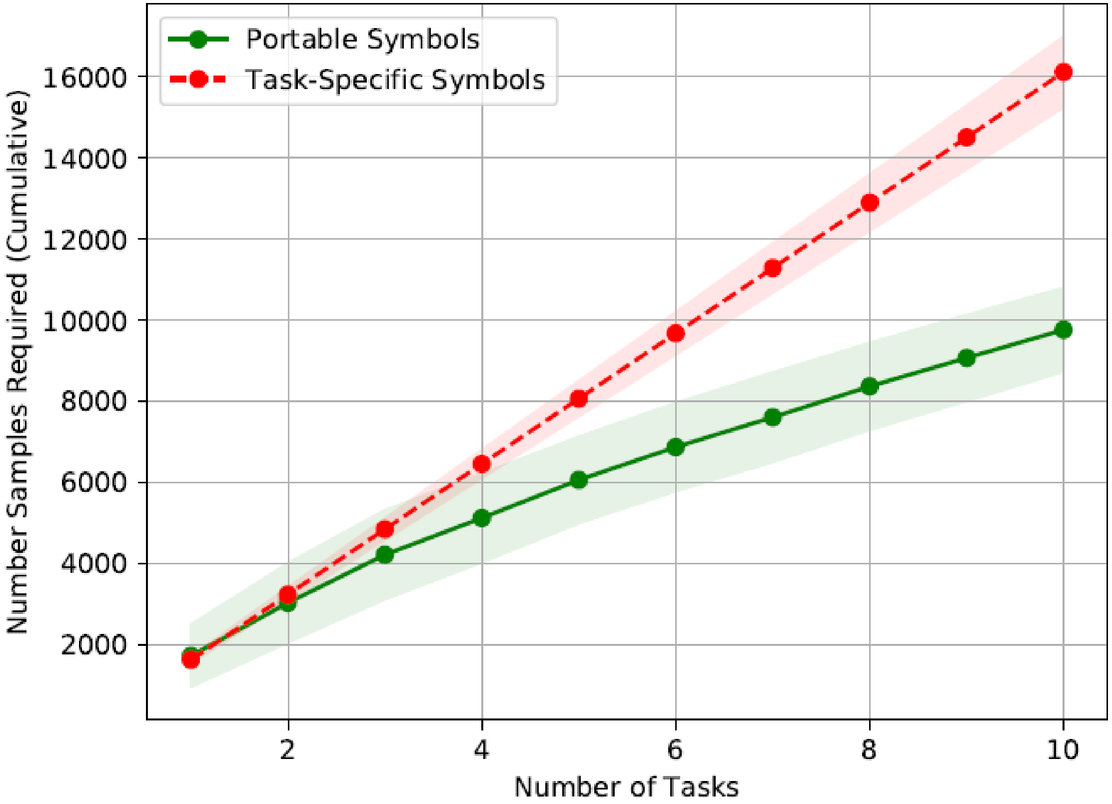


Partitioned rules

	Precondition:	Negative effect:	Positive effect:
Interact**1**:			
Interact**3**:			

Experiments

- Require fewer samples in subsequent tasks



Portable rules

- Learn abstract rules and their groundings
  - Transfer between domain instances
  - Just by learning linking functions

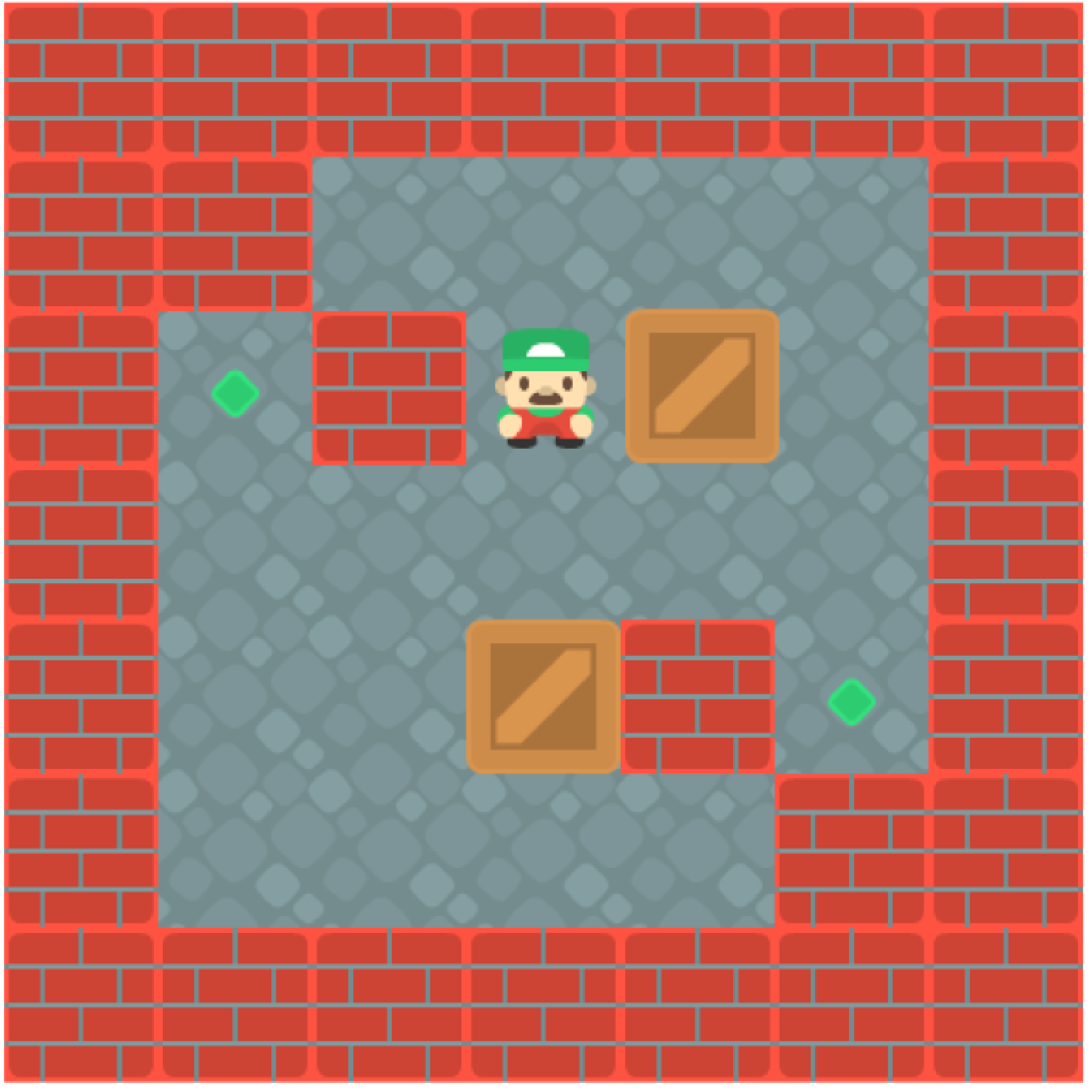
- But what if there is additional structure?

- In particular, there are many rule instances (objects of interest)?

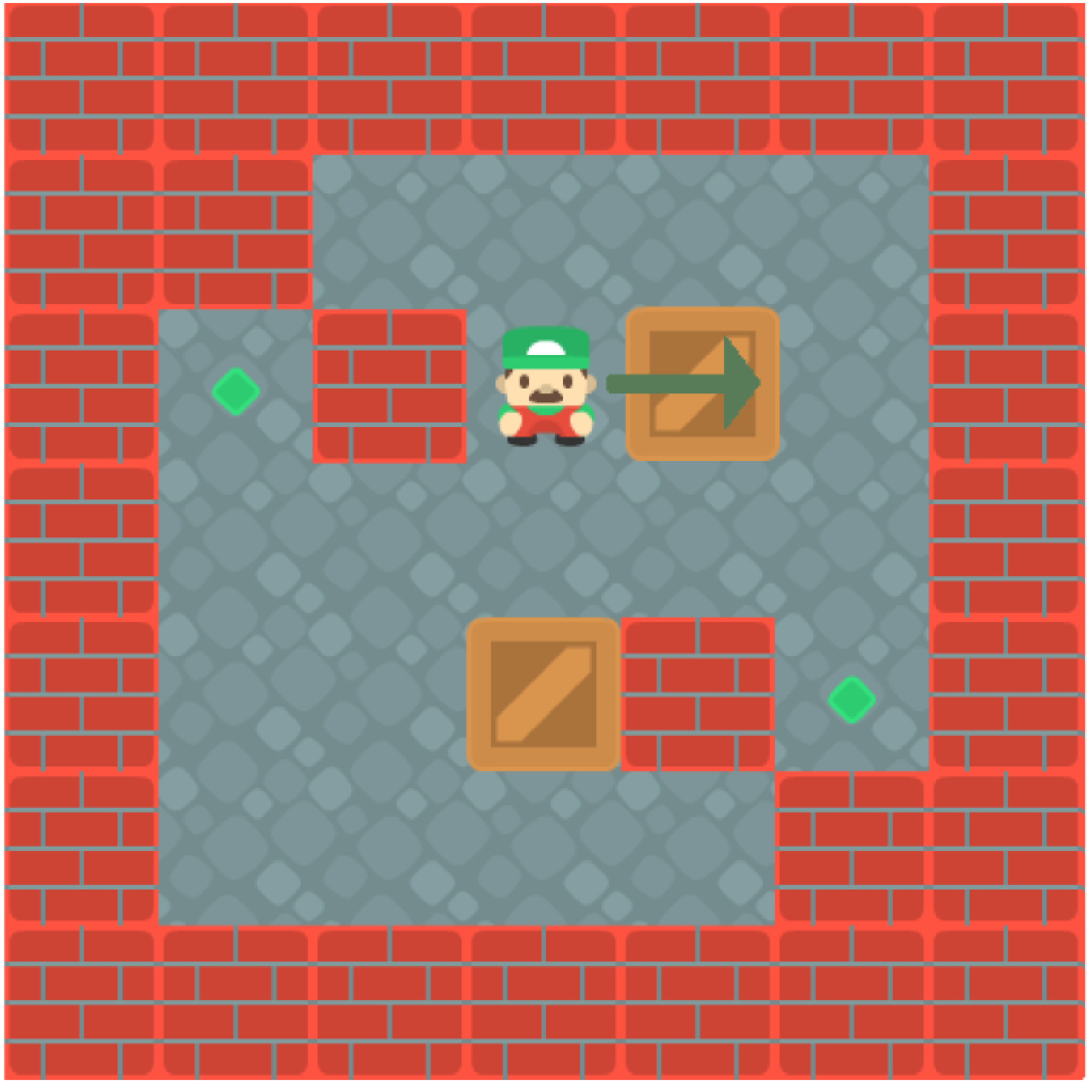
Ofir Marom

Ofir Marom and Benjamin Rosman. Zero-Shot Transfer with Deictic Object-Oriented Representation in Reinforcement Learning. NIPS, 2018.

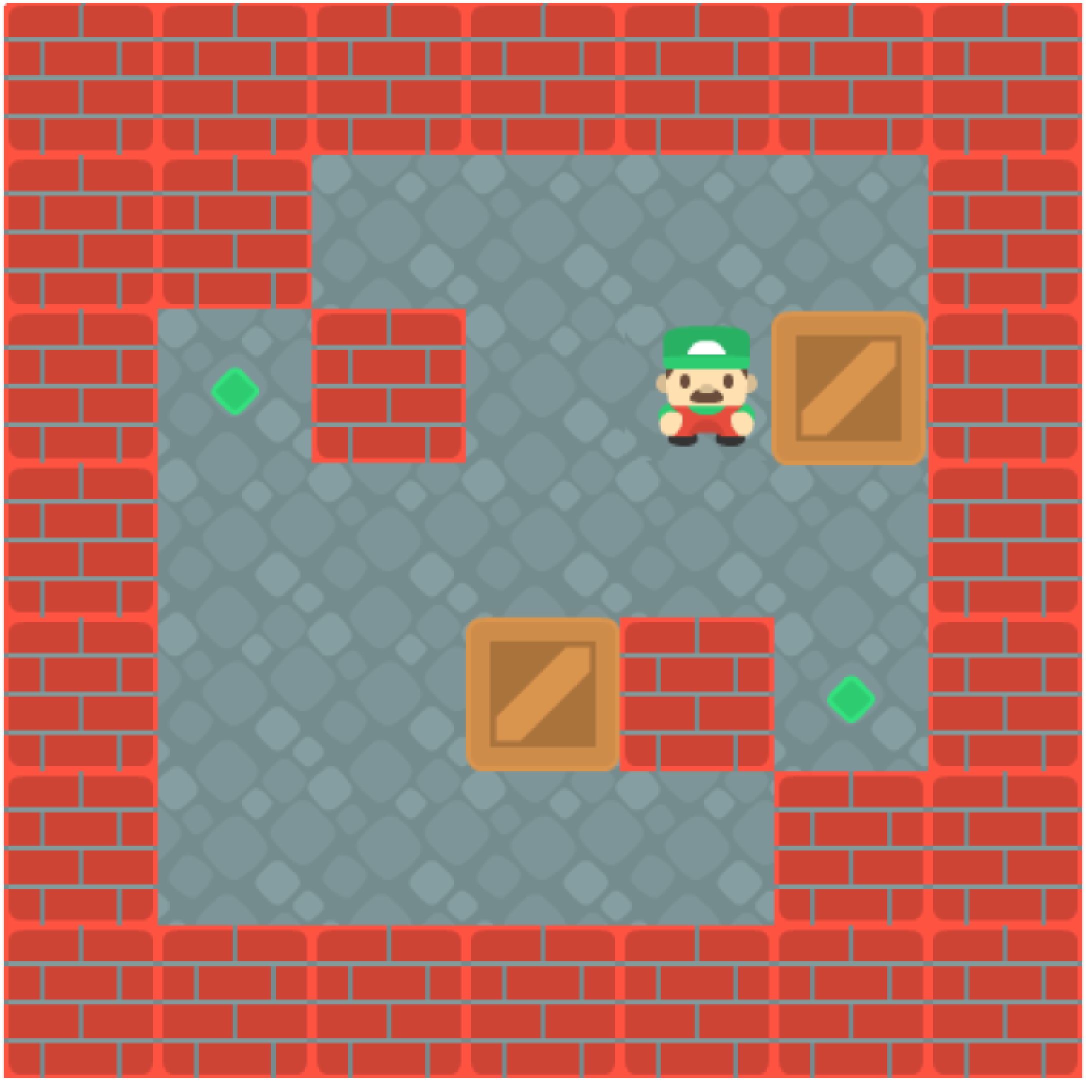
Example: Sokoban



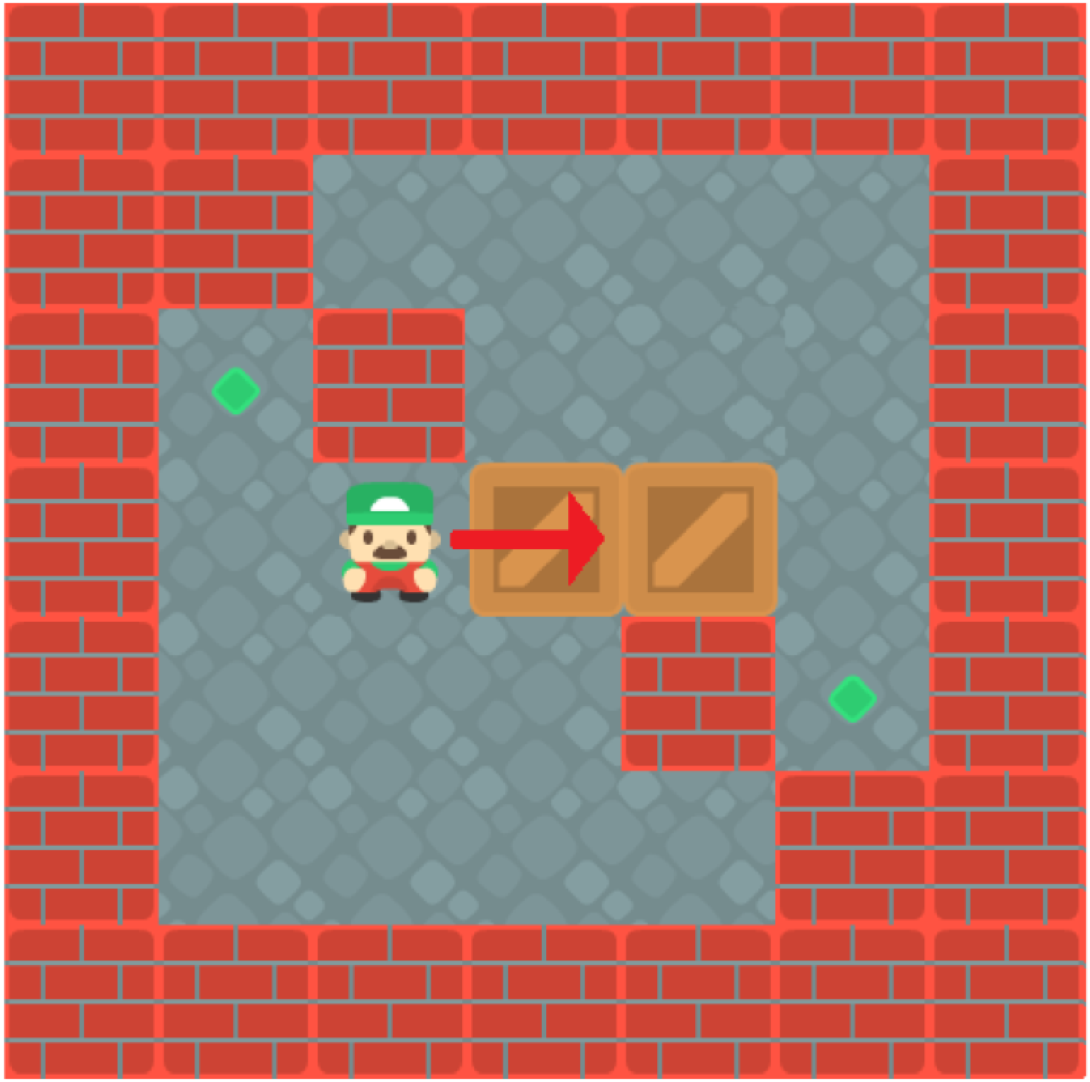
Sokoban (legal move)



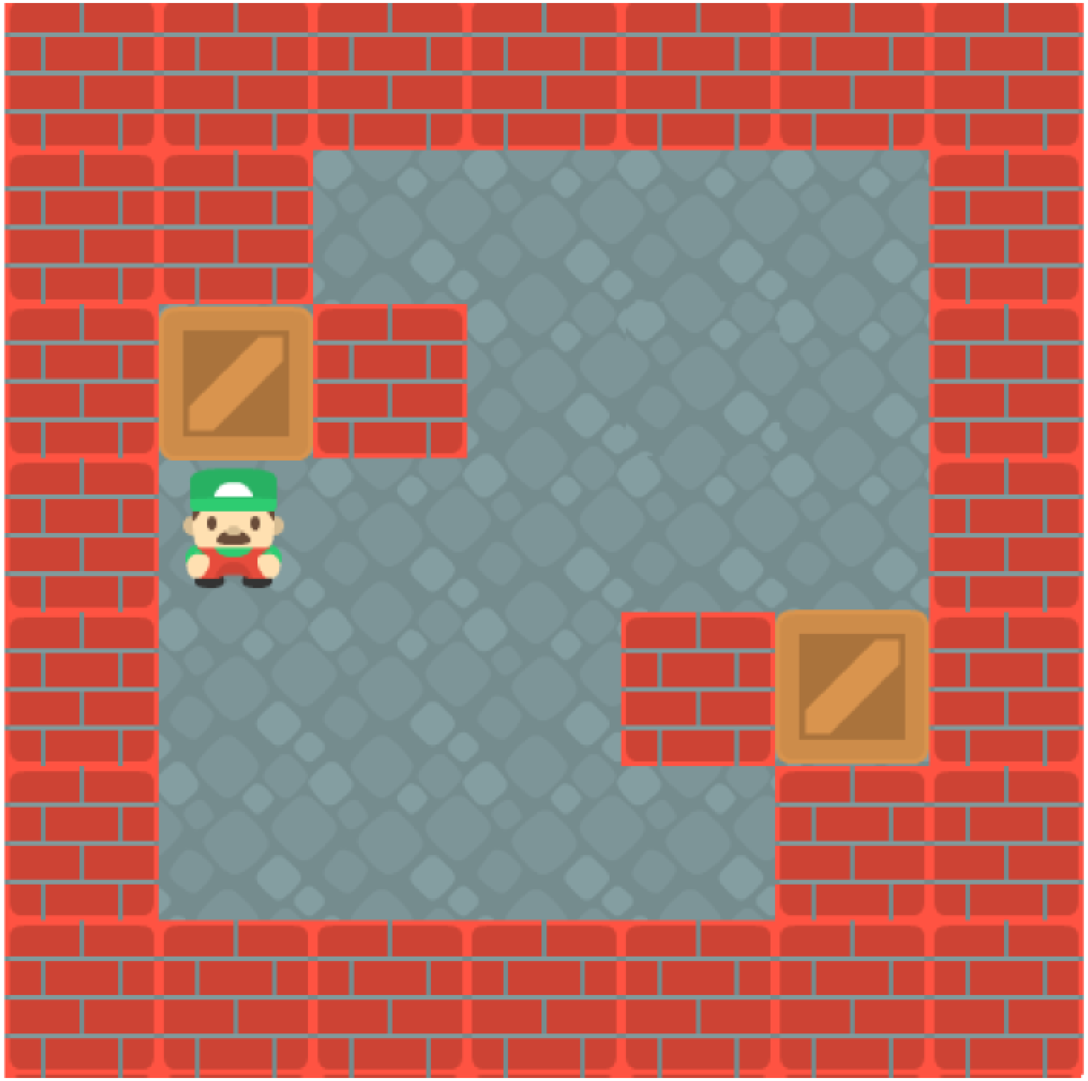
Sokoban (legal move)



Sokoban (illegal move)



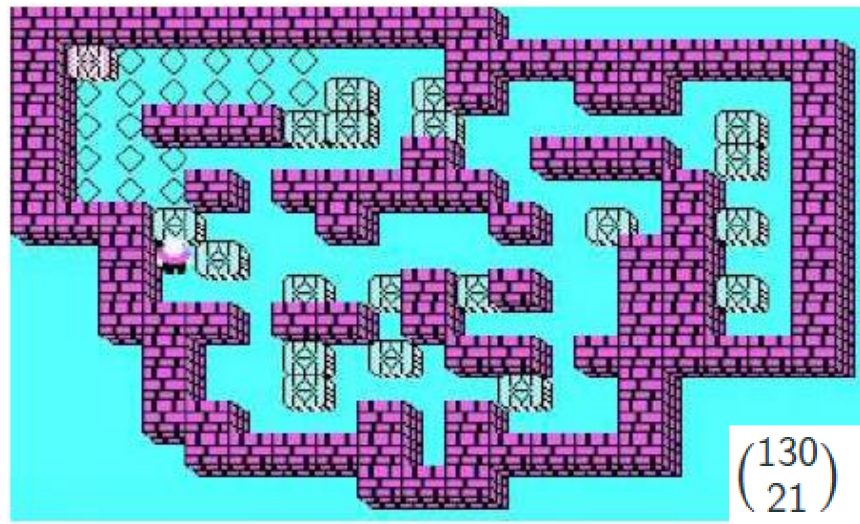
Sokoban (goal)



Representations

$$s = (agent_x = 3, agent_y = 4, box1_x = 4, box1_y = 4, box2_x = 3, box2_y = 2)$$

- Poor scalability
  - 100s of boxes?
  - Transferability?

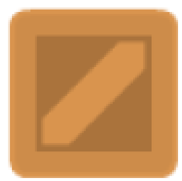
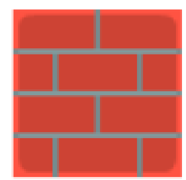


$$\binom{130}{21} \approx 10^{23}$$

- Effects of actions depend on interactions further away, complicating a mapping to agent space

Object-oriented representations

- Consider objects explicitly
  - Object classes have attributes
  - Relationships based on formal logic: $On(box_1, storage_1)$

Object Classes				
Attributes	(X,Y)	(X,Y)	(X,Y)	(X,Y)
Object Instances	box1(1,5) box2(2,4) ...	...	...	...

Propositional OO-MDPs [Duik, 2010]

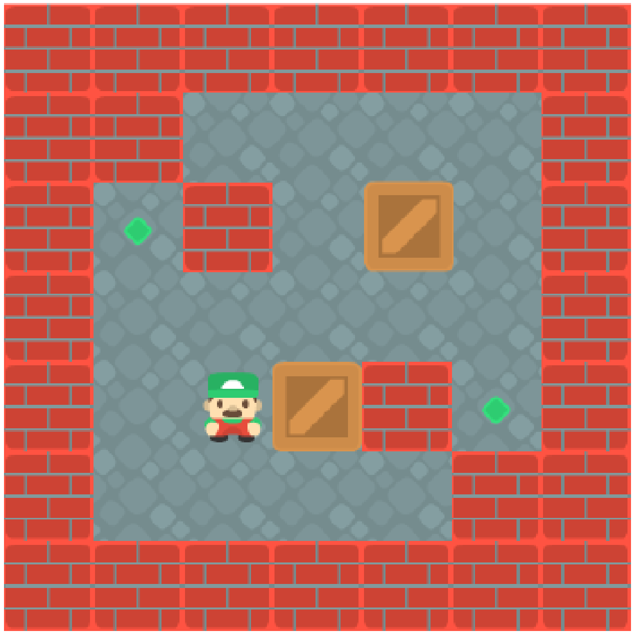
- Describe transition rules using schemas

- Propositional Object-Oriented MDPs
  - Provably efficient to learn (KWIK bounds)

$$East \wedge$$
$$Touch_{East}(Person, Wall)$$
$$\Rightarrow Person.x \leftarrow Person.x + 0$$

Benefits

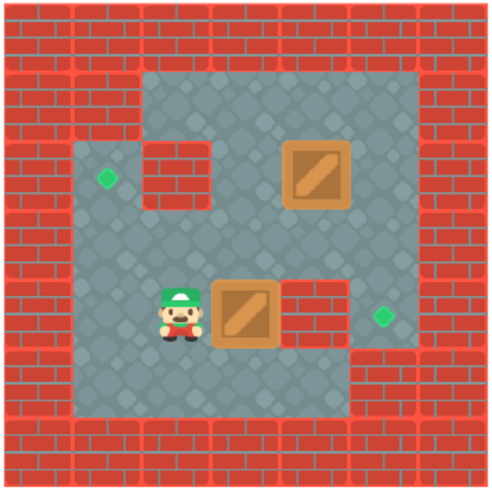
- Propositional OO-MDPs
  - Compact representation
  - Efficient learning of rules



Limitations

- Propositional OO-MDPs are efficient, but restrictive

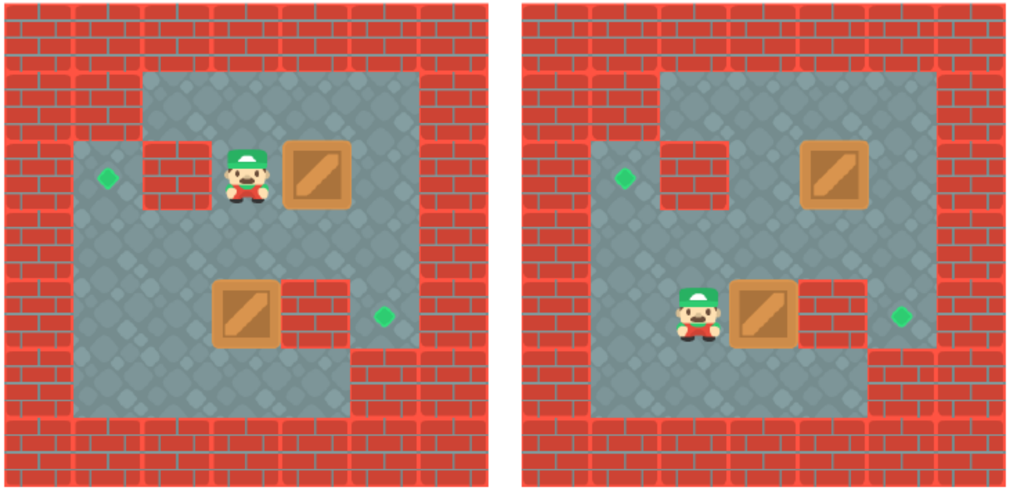
$$East \wedge$$
$$Touch_{West}(Box, Person) \wedge Touch_{East}(Box, Wall)$$
$$\Rightarrow Box.x \leftarrow ?$$



Limitations

- Propositional OO-MDPs are efficient, but restrictive
  - Restriction that **preconditions are propositional**
  - Can't refer to the same box

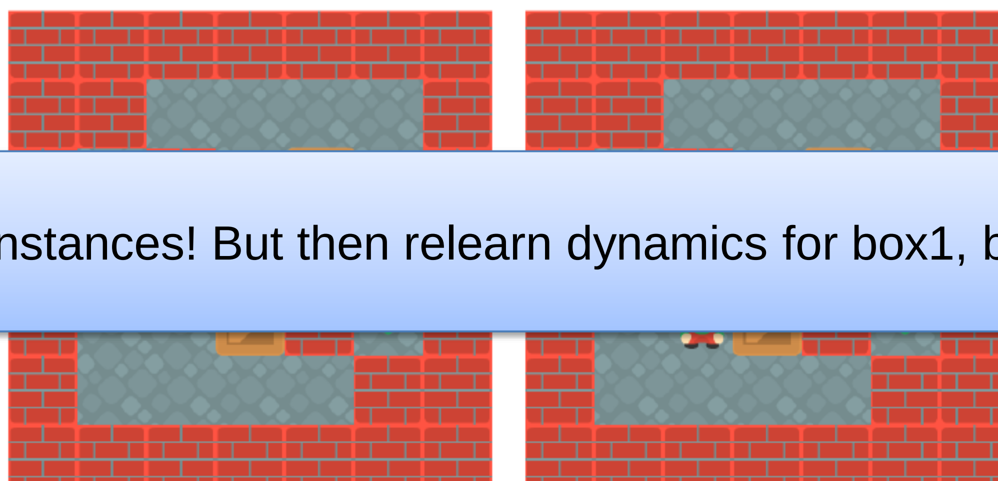
$$East \wedge$$
$$Touch_{West}(\textcolor{red}{Box}, Person) \wedge Touch_{East}(\textcolor{red}{Box}, Wall)$$
$$\Rightarrow Box.x \leftarrow ?$$



Limitations

- Propositional OO-MDPs are efficient, but restrictive
  - Restriction that **preconditions are propositional**
  - Can’t refer to the same box

$$East \wedge$$
$$Touch_{West}(Box, Person) \wedge Touch_{East}(Box, Wall)$$
$$\Rightarrow Box.x \leftarrow ?$$



Ground instances! But then relearn dynamics for box1, box2, etc.

Deictic OO-MDPs

- **Deictic predicates** instead of propositions
  - Grounded only with respect to a central deictic object (*"me" or "this"*)
  - Relates to other non-grounded objects
- Transition dynamics of $Box.x$ depends on grounded box object

$$East \wedge Touch_{West}(box, Person) \wedge Touch_{East}(box, Wall) \Rightarrow box.x \leftarrow box.x + 0$$

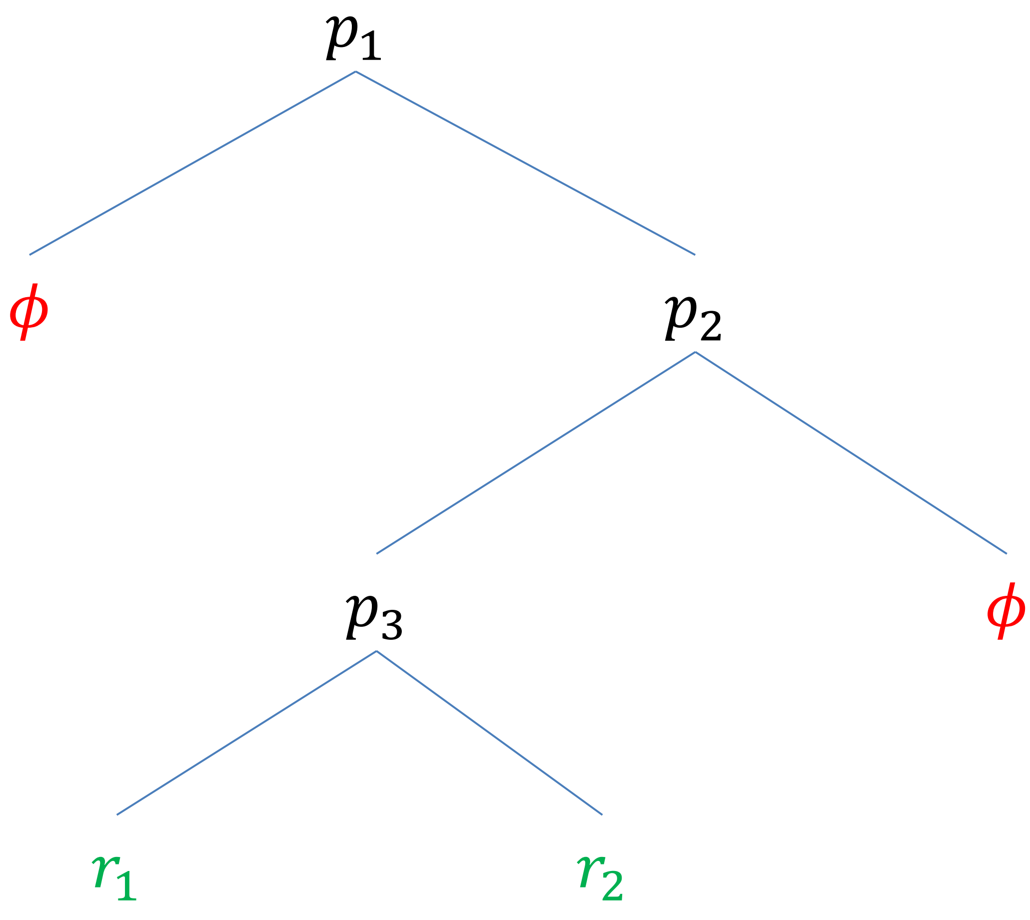
- Also provably efficient

Learning the dynamics

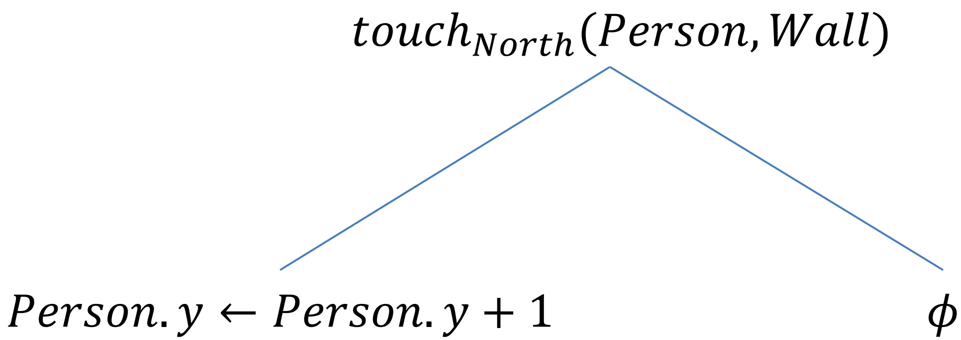
- Learning from experience:
  - For each action, how do attributes change?
- KWIK framework

- Propositional OO-MDPs: DOORMAX algorithm
  - Transition dynamics for each attribute and action must be representable as a binary tree
  - Effects at the leaf nodes
  - Each possible effect can occur at most at one leaf, except for a failure condition (globally nothing changes)

Learning the dynamics



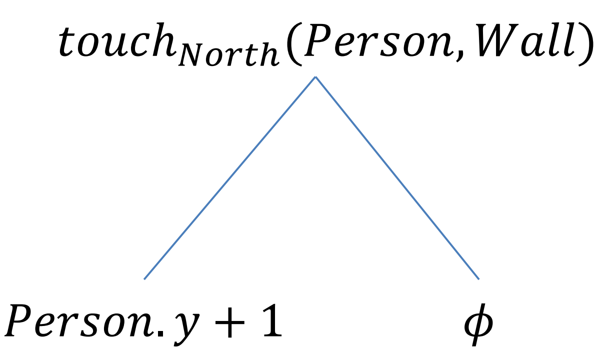
Example: action = North

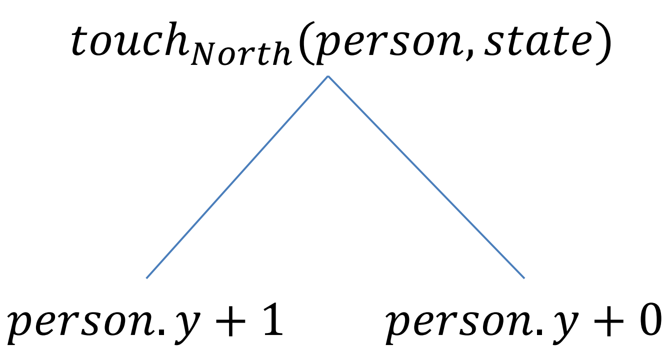


- Given:
  - Each effect can only occur once on the tree
  - Global failure condition
  - Deterministic effects
- Learn from common elements in state propositions (experience)

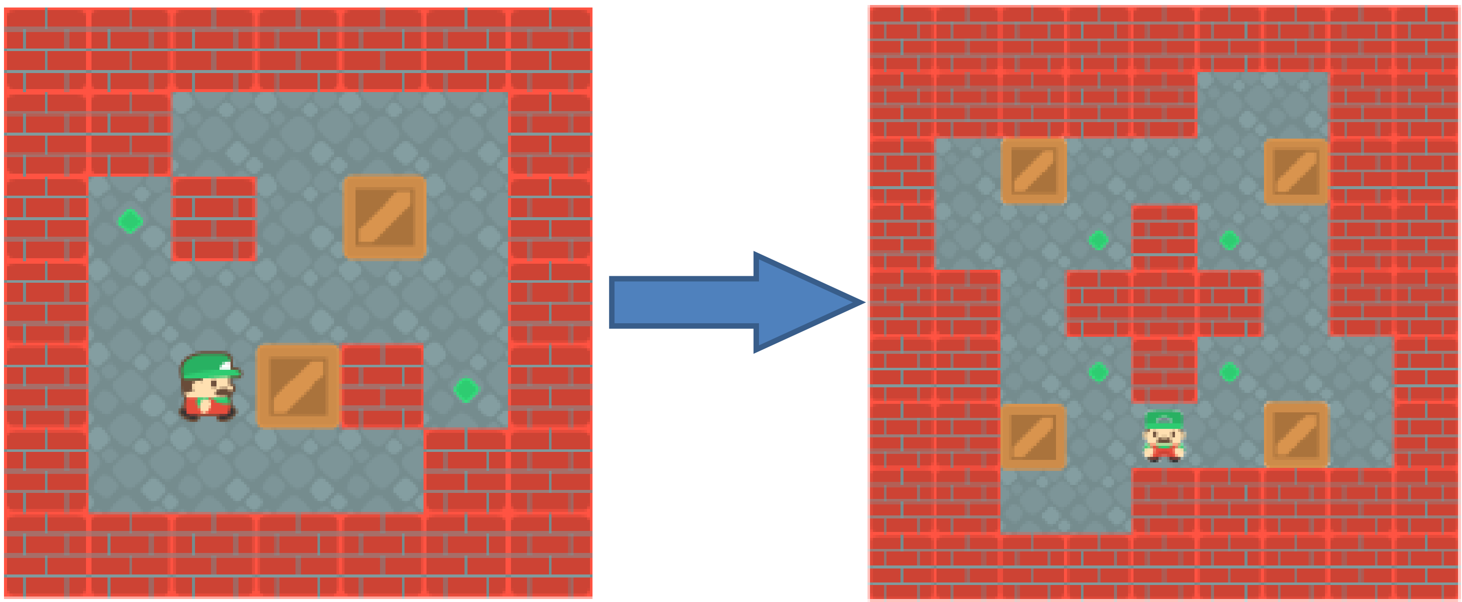
DOORMAX for deictic OO-MDPs

- We adapt the DOORMAX algorithm to deictic OO-MDPs
  - Remove global failure condition
  - Bound the number of times a condition can occur
    - Can still be learned efficiently





Experiments



$\sim 8k$ states $\quad\quad$ $\sim 1M$ states

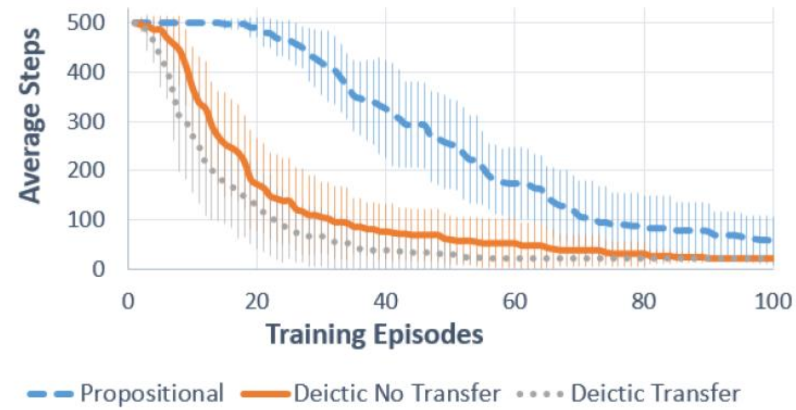
- Zero-shot transfer: one run of value iteration

Experiments

- Taxi domain
  - Multi-passengers
  - Only one in the taxi at a time
- On executing a *pickup* action
  - Change *in_taxi* attribute of **correct** passenger

Experiments

1 passenger



2 passengers

3 passengers

4 passengers

Take away thoughts

- **Reinforcement learning** gives us a powerful tool for learning behaviours, but extra work is required for generalisation

- Reasoning in an **agent-centric manner**:
  - Symbol-based view on skills
  - Enable knowledge reuse

- Reasoning in an **object-centric manner**:
  - Learn models of local object interactions
  - Efficient learning and transfer

Thank you!

www.benjaminrosman.com – www.raillab.org

Funded by

science & technology
Department:
Science and Technology
REPUBLIC OF SOUTH AFRICA

PROUDLY SOUTH AFRICAN

RAIL LAB