Lecture 32

CS 6260 Automated Planning and Learning

PPDDL and Learning PPDDL

Department of Computer Science and Engineering

Indian Institute of Technology Madras

FOND Planning*

State-transition system or *classical planning domain*: $\Sigma = (S,A,T)$ or (S,A,T)

- S - finite set of *states*
- A - finite set of *actions*
- $T: S \times A \times S$

  prediction (or *state-transition*) function
  - *partial* function: $T(s,a,s')$ is not necessarily defined for every (s,a,s')
  - a is *applicable* in s iff $T(s,a,s')$ is defined
  - Domain$(a) = \{s \in S \mid a \text{ is applicable in } s\}$
  - An action is nondeterministic if $|EXEC(s,a)| > 1$.

- *FOND planning problem*:
  - $P = (\Sigma, s_0, S_g)$
  - planning domain, initial state, set of goal states
- *Solution* for P:
  - a policy π

* Cimatti et al., Weak, strong, and strong cyclic planning via symbolic model checking, Artificial Intelligence, Vol. 147, Iss. 1–2, 2003.

Probabilistic Planning

State-transition system or *classical planning domain*:
$\Sigma = (S,A,T)$ or (S,A,T)

- S - finite set of *states*
- A - finite set of *actions*
- Transition Model: P (s' | s, a)

  prediction (or *state-transition*) function
  - *a* is *applicable* in *s* iff $P(s' \mid s,a)$ is non-zero

- *Probabilistic planning problem*:
  - $P = (\Sigma, s_0, S_g)$
  - planning domain, initial state, set of goal states

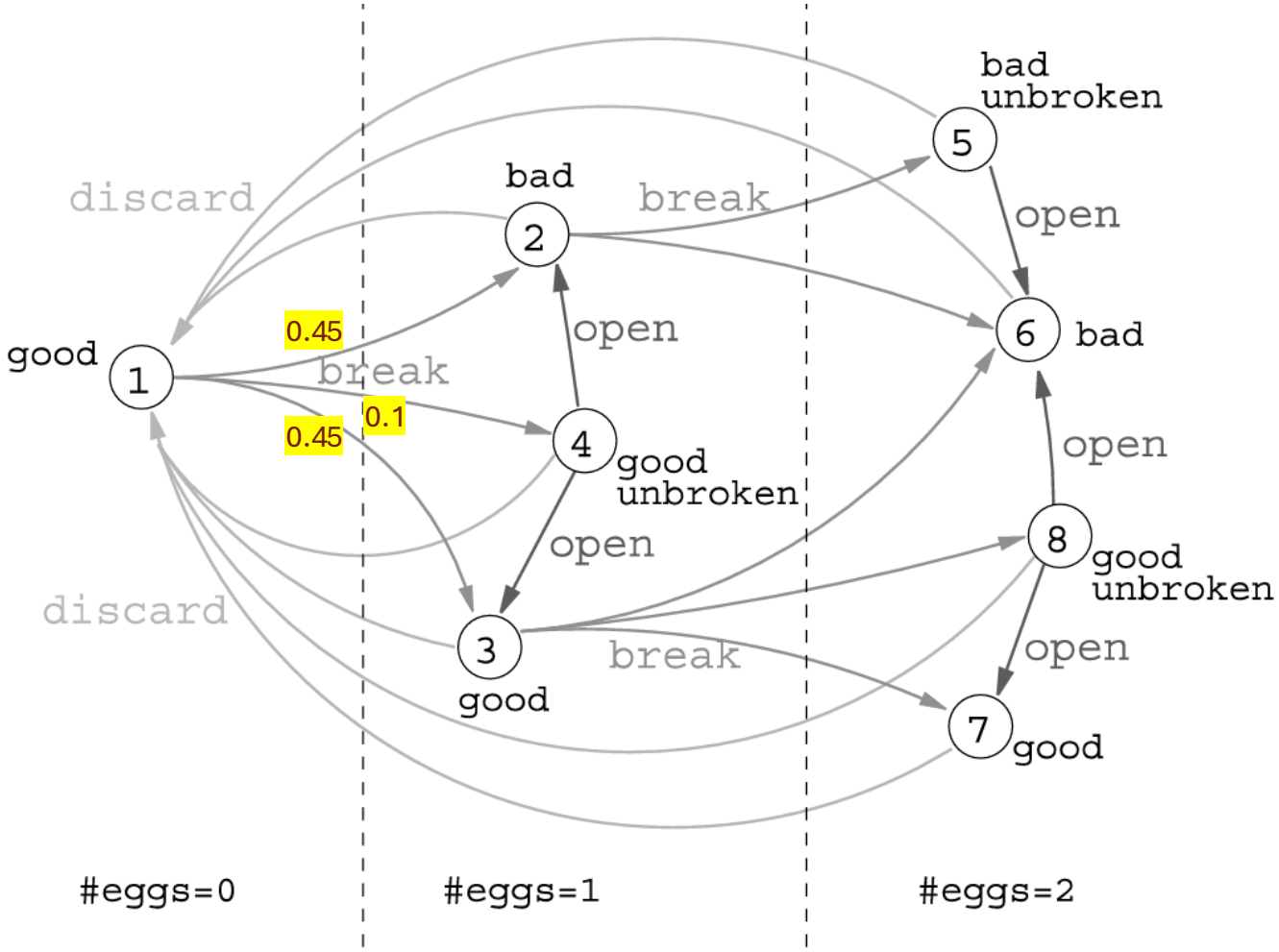
- *Solution* for *P*:
  - a policy π

Running Example: Omelette Preparation

- Propositions P:
  {#eggs=0,
  #eggs=1,
  #eggs=2,
  bad,
  good,
  unbroken}

- State S: 2^P

- Actions: {break,open,discard}



* Cimatti et al., Weak, strong, and strong cyclic planning via symbolic model checking, Artificial Intelligence, Vol. 147, Iss. 1–2, 2003.

FOND Actions: break-first-egg

```
(:action break-first-egg
        :precondition (and
                (eggs-0)
                (not (unbroken))
        )
        :effect (oneof
                ;; Outcome 1: egg breaks cleanly and is good
                (and (not (eggs-0)) (eggs-1) (good) (not (bad)))

                ;; Outcome 2: egg breaks cleanly and is bad
                (and (not (eggs-0)) (eggs-1) (not (good)) (bad))

                ;; Outcome 3: egg fails to break open (still unknown good/bad)
                (and (not (eggs-0)) (eggs-1) (good) (unbroken))
        )
)
```

Probabilistic Actions: break-first-egg

```
(:action break-first-egg
        :precondition (and
                (eggs-0)
                (not (unbroken))
        )
        :effect (and (probabilistic
                ;; Outcome 1: egg breaks cleanly and is good
                0.45 (and (not (eggs-0)) (eggs-1) (good) (not (bad)))

                ;; Outcome 2: egg breaks cleanly and is bad
                0.45 (and (not (eggs-0)) (eggs-1) (not (good)) (bad))

                ;; Outcome 3: egg fails to break open (still unknown good/bad)
                0.1 (and (not (eggs-0)) (eggs-1) (good) (unbroken))
        ))
)
```

Learning PPDDL Models

- *Same principle as what we saw earlier for deterministic models (Agent Interrogation).*
- *Active Learning more accurate than passive learning.*
- Query-based Autonomous Capability Estimation (QACE)*.

*Verma et al., Autonomous Capability Assessment of Sequential Decision-Making Systems in Stochastic Settings, NeurIPS 2023

QACE Works in 2 Phases

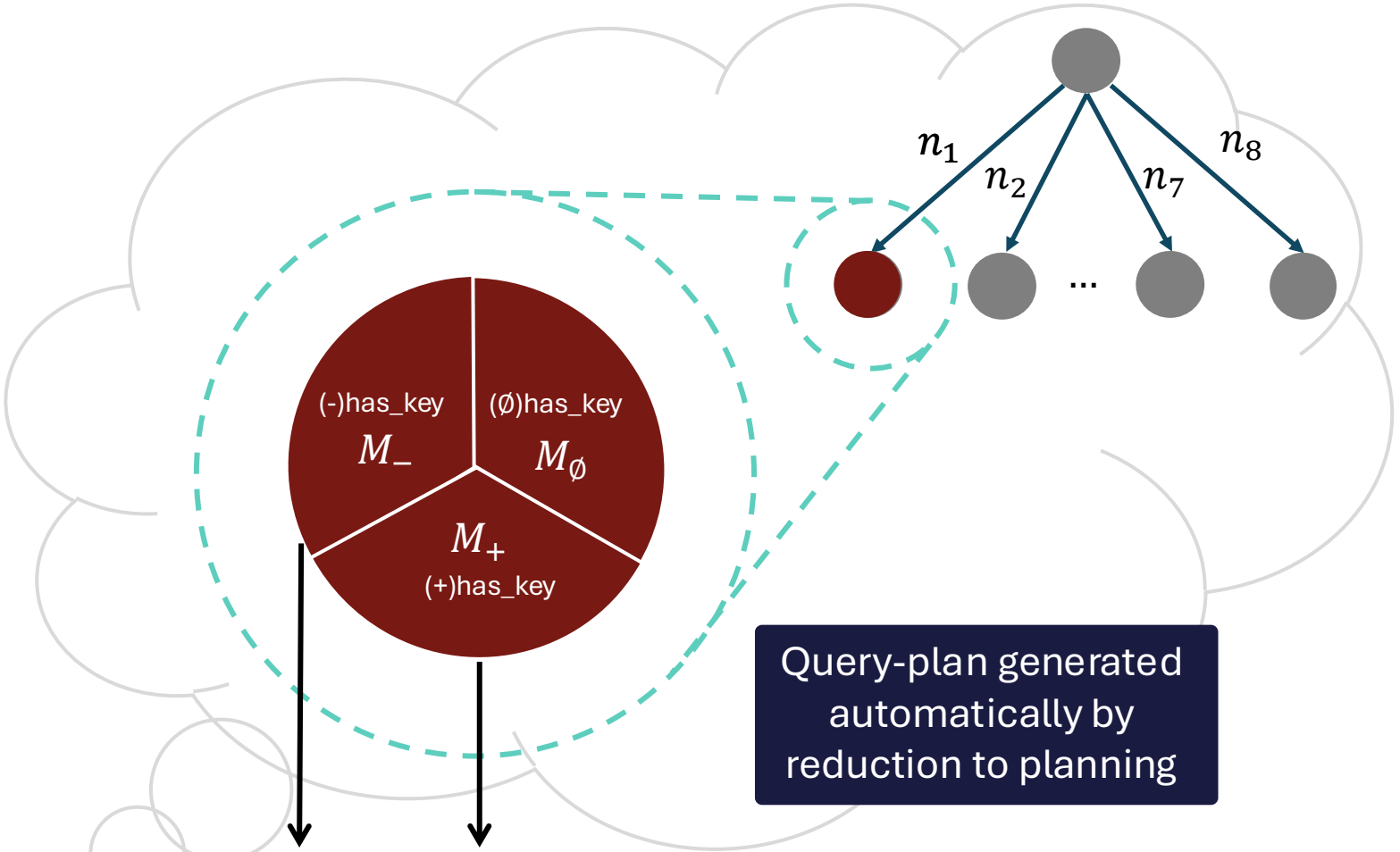
Phase 1:
Learn a Non-Deterministic Model

```
pick-item (?location ?item)
  precondition:
    (empty-arm) ∧ (robot-at ?location) ∧
    (at ?location ?item) ∧ (has-charge)
  effect:
    [!(empty-arm) ∧ (holding ?item) ∧
          !(at ?location ?item)]
    [!(has-charge)]
    []                          # no-change effect
```

Phase 2:
Convert Non-Deterministic Model to Probabilistic Model

```
pick-item (?location ?item)
  precondition:
    (empty-arm) ∧ (robot-at ?location) ∧
    (at ?location ?item) ∧ (has-charge)
  effect:
    0.8: [!(empty-arm) ∧ (holding ?item) ∧
          !(at ?location ?item)]
    0.2: [!(has-charge)]
    0.1: []                     # no-change effect
```

Hierarchical Query Synthesis for PPDDL



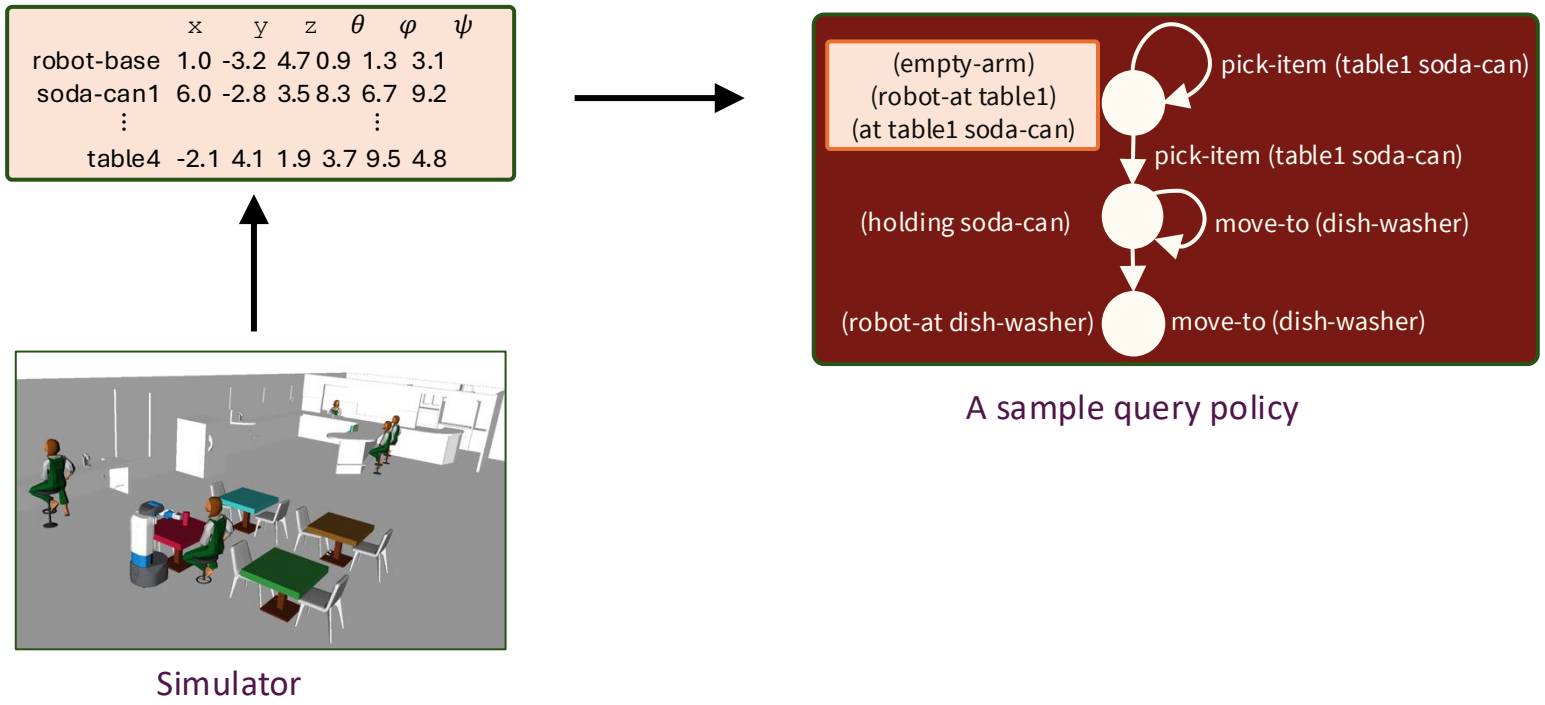
```
(:action open-door
  :parameters (?l1)
  :precondition (and
 n1  (+/-/∅)(has_key)
 n2  (+/-/∅)(door_open)
 n3  (+/-/∅)(door_adjacent ?l1)
 n4  (+/-/∅)(player_at ?l1))
  :effect (oneof (and
 n5  (+/-/∅)(has_key)
 n6  (+/-/∅)(door_open)
 n7  (+/-/∅)(door_adjacent ?l1)
 n8  (+/-/∅)(player_at ?l1)
        )(and
 n9  (+/-/∅)(has_key)
 n10 (+/-/∅)(door_open)
 n11 (+/-/∅)(door_adjacent ?l1)
 n12 (+/-/∅)(player_at ?l1))
        ))
```

[**Verma**, Karia, Srivastava; NeurIPS '23]

What Changed from Deterministic Setting

Plan replaced by Policy in the query.

Need to abstract each low level state to get high level state

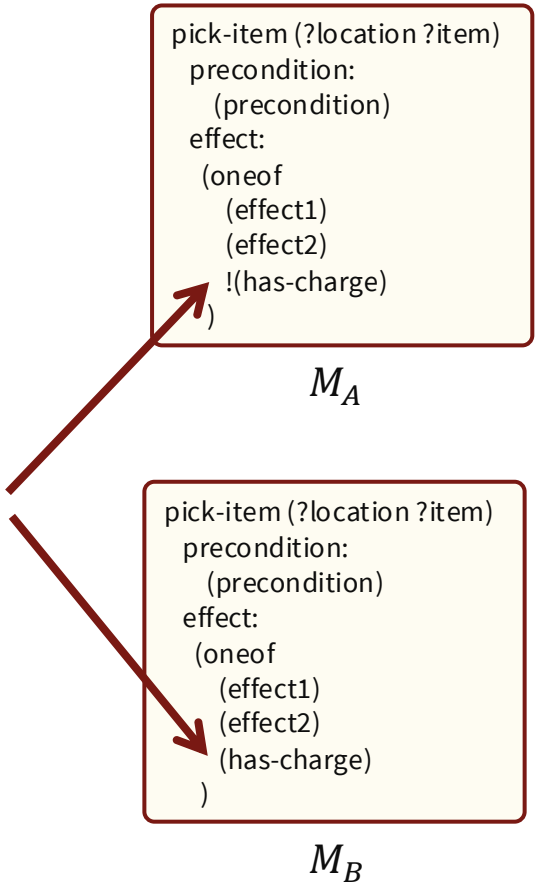


Simulator

A sample query policy

Reducing Query Synthesis to FOND Planning

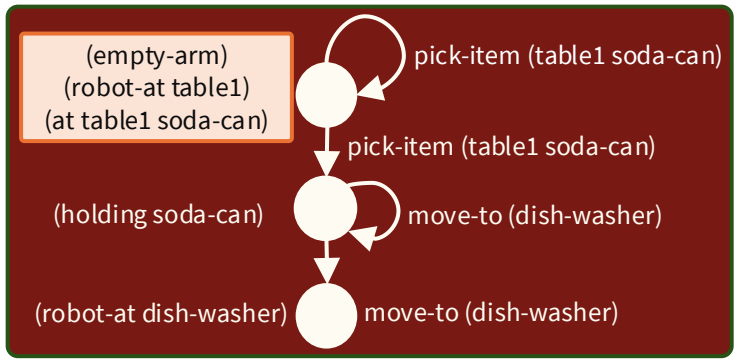
Models differ in only one predicate in precondition or effect.



pick-item (?location ?item)
precondition:
$(\text{precondition})_{M_A} \vee (\text{precondition})_{M_B}$
effect:
$(\text{precondition})_{M_A} \wedge !(\text{precondition})_{M_B} \rightarrow (\text{goal})$
$!(\text{precondition})_{M_A} \wedge (\text{precondition})_{M_B} \rightarrow (\text{goal})$
$(\text{precondition})_{M_A} \wedge (\text{precondition})_{M_B} \rightarrow$
(one of
$((\text{effect1})_{M_A} \wedge (\text{effect1})_{M_B})$
$((\text{effect2})_{M_A} \wedge (\text{effect2})_{M_B})$
$(!(\text{has-charge})_{M_A} \wedge (\text{has-charge})_{M_B})$
)

Consolidated capability used to generate the FOND Planning Domain

FOND Model to PPDDL Model: MLE




```
pick-item (?location ?item)
  precondition:
    (empty-arm) ∧ (robot-at ?location) ∧
    (at ?location ?item) ∧ (has-charge)
  effect:
    [!(empty-arm) ∧ (holding ?item) ∧
          !(at ?location ?item)]
    [!(has-charge)]
    []                          # no-change effect
```

1. Ask queries multiple times

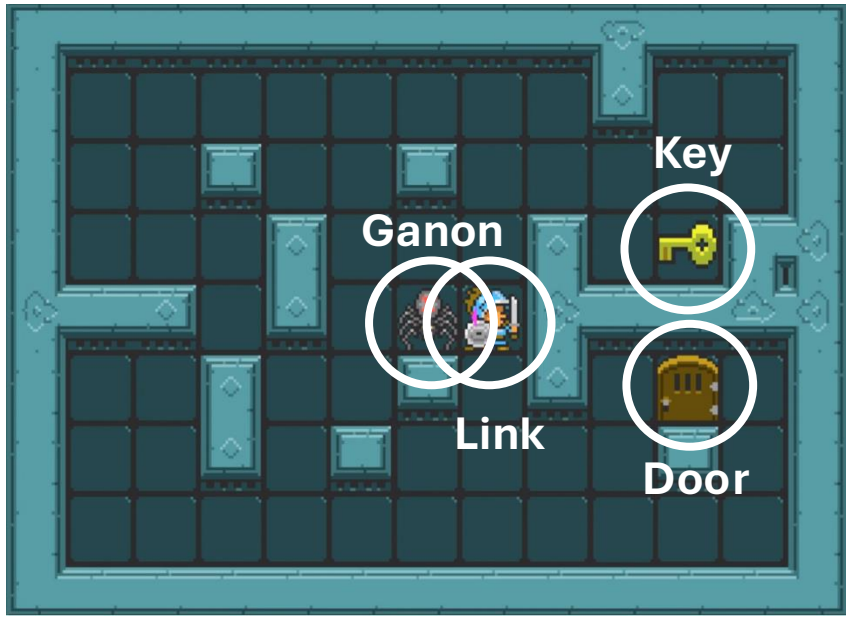
2. Perform MLE on collected effects

```
pick-item (?location ?item)
  precondition:
    (empty-arm) ∧ (robot-at ?location) ∧
    (at ?location ?item) ∧ (has-charge)
  effect:
    0.8: [!(empty-arm) ∧ (holding ?item) ∧
          !(at ?location ?item)]
    0.2: [!(has-charge)]
    0.1: []                     # no-change effect
```

What if Actions are not Part of the Input?

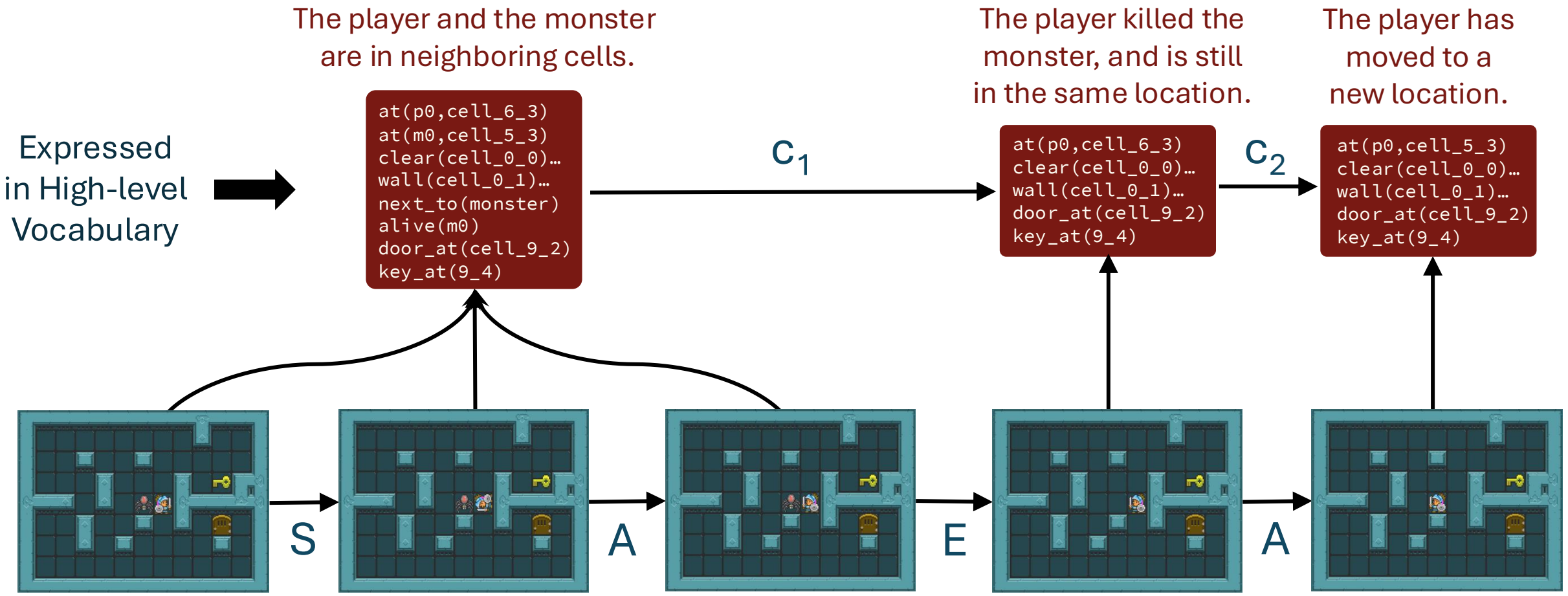
Let's see how to solve this for deterministic settings!!

High-level Vocabulary can be Less Expressive



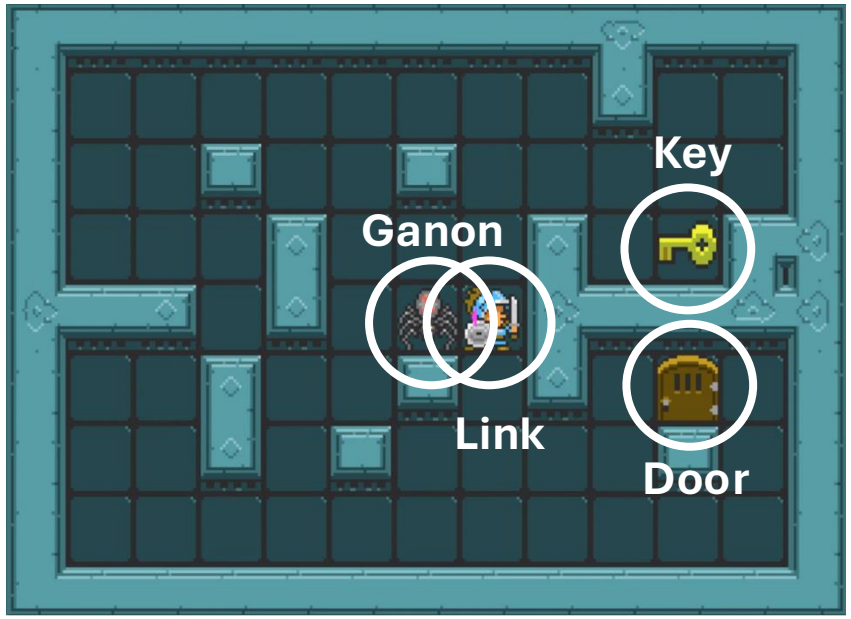
Agent's State Representation	State Representation in User's Vocabulary
pixel_1_1(#42A8B3)	(at ganon 5,3)
pixel_1_2(#42A8B3)	(at link 6,3)
.	(at key 9,4)
.	(at door 9,2)
.	
pixel_n_m(#203A3D)	

Discovering Capabilities using Predicate Abstraction



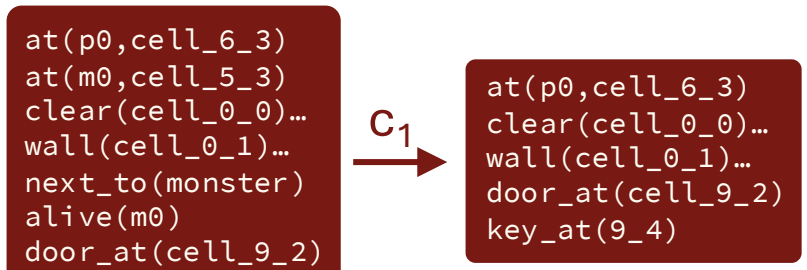
Verma, P.; Marpally S. R.; and Srivastava, S. ***Discovering User-Interpretable Capabilities of Black-Box Planning Agents***. In KR 2022.

Parameterizing Capabilities



Agent's State Representation	State Representation in User's Vocabulary
pixel_1_1(#42A8B3)	(at ganon 5,3)
pixel_1_2(#42A8B3)	(at link 6,3)
.	(at key 9,4)
.	(at door 9,2)
.	
pixel_n_m(#203A3D)	

Parameterizing Capabilities



[Sample pre and post states of a capability]

```
(:capability c4
 :parameters (?player1 ?cell1
   ?monster1 ?cell2)
 :precondition
  (and (alive ?monster1)
    (at ?player1 ?cell1)
    (at ?monster1 ?cell2)
    (next_to ?monster1))
 :effect
  (and (clear ?cell2)
   (not(alive ?monster1))
   (not(at ?monster1 ?cell2))
   (not(next_to ?monster1))))
```

How to learn these?

[Learned capability description]

For each capability:

- Extract what predicates were different in the pre and post states of the capability.
- Extract the parameters from those predicates to create a candidate parameter set.
- Learn an interpretable model of the capability using the query-response interface.

How to learn such models?

Verma, P.; Marpally S. R.; and Srivastava, S. ***Discovering User-Interpretable Capabilities of Black-Box Planning Agents***. In KR 2022.

Recap: What is a Query?

- *Every query can be viewed as a function from models to responses.*

Plan Outcome Queries:

Query: $\langle s_I, \pi \rangle$
Initial State and Plan (in terms of capabilities)

Agent's Response: $\langle \ell, s_F \rangle$
Length of plan that can be executed successfully and the final state.

How do we generate these queries?
How do we use them?

*Verma et al., Autonomous Capability Assessment of Sequential Decision-Making Systems in Stochastic Settings, NeurIPS 2023

State Reachability Query

- *Every query can be viewed as a function from models to responses.*

State Reachability Queries:

Query: $\langle s_I, s_G \rangle$
Initial State and Goal State

Agent's Response: $\langle Yes, No \rangle$
Whether it can reach from initial state to goal using its internal mechanism.

How do we generate these queries from plan outcome queries?

*Verma et al., Autonomous Capability Assessment of Sequential Decision-Making Systems in Stochastic Settings, NeurIPS 2023

Query Refinement

$s_0, \langle c_1, c_2, \ldots, c_n \rangle$

What will happen if you execute the plan $\langle c_1, c_2, \ldots, c_n \rangle$ starting in a state s_0?

Plan Outcome Query

```
(:capability c1
 :parameters (...)
 :precondition (...)
 :effect (...))
```

```
(:capability c2
 :parameters (...)
 :precondition (...)
 :effect (...))
```

. . .

$s_0, c_1, s_1, c_2, s_2, \ldots, c_n, s_n$

$\langle s_0, s_1 \rangle, \langle s_1, s_2 \rangle, \ldots, \langle s_{n-1}, s_n \rangle$

State Refinement

$\langle \bar{s}_0, \bar{s}_1 \rangle, \langle \bar{s}_1, \bar{s}_2 \rangle, \ldots, \langle \bar{s}_{n-1}, \bar{s}_n \rangle$

Can you reach state $\bar{s}_1$ from state $\bar{s}_0$?

Can you reach state $\bar{s}_2$ from state $\bar{s}_1$?

. . .

Can you reach state $\bar{s}_n$ from state $\bar{s}_{n-1}$?

State Reachability Queries

How about Non-Deterministic Settings?

Solution not trivial

Checkout this paper: https://arxiv.org/pdf/2512.16733