Lecture 27

CS 6260 Automated Planning and Learning

Agent Interrogation Algorithm

Department of Computer Science and Engineering

Indian Institute of Technology Madras

Logistics

Logistics

- 3 Exams:
  1. [Quiz 1] Feb 19: 08:00 AM – 08:50 AM
  2. [Quiz 2] Apr 10: 05:00 PM – 05:50 PM
  3. [End Sem] May 12

- Office Hours: 1 hour after class every week Tu, W, Th
  - No office hours today

- ROS based Assignment
  - https://wiki.ros.org/ROS/Tutorials

Recap: Agent Interrogation Algorithm

Deterministic and Stationary Setting

Input

- Predicates (User vocabulary)
  - With their evaluation functions
- List of capabilities.

Output

- PDDL-like description of each capability.

Siddharth Srivastava

Shashank R Marpally

Asking the Right Questions: Learning Interpretable Action Models Through Query Answering

Pulkit Verma, Shashank Rao Marpally, Siddharth Srivastava

In Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence, 2021 (CORE Ranking: A*)

Assumptions

- User's vocabulary matches simulator's vocabulary.
- Black-Box AI provides a list of capabilities.
- Stationary agent model.
- Deterministic environment.
- Fully observable setting.

Exponential Search for Learning Correct Description

- Consider the following 4 predicates/concepts:
  - (has_key)
  - (door_open)
  - (door_adjacent ?x)
  - (player_at ?x)
- Consider just one capability: (open-door ?x)
- $9^{|C|\times|P|} = 9^{1\times4}$=6561 possible models (Assuming deterministic models/ descriptions, i.e., no probabilities).

```
(:action open-door
  :parameters (?l1)
  :precondition (and
    (+/-/∅)(has_key)
    (+/-/∅)(door_open)
    (+/-/∅)(door_adjacent ?l1)
    (+/-/∅)(player_at ?l1))
  :effect (and
    (+/-/∅)(has_key)
    (+/-/∅)(door_open)
    (+/-/∅)(door_adjacent ?l1)
    (+/-/∅)(player_at ?l1))
```

Simple Queries

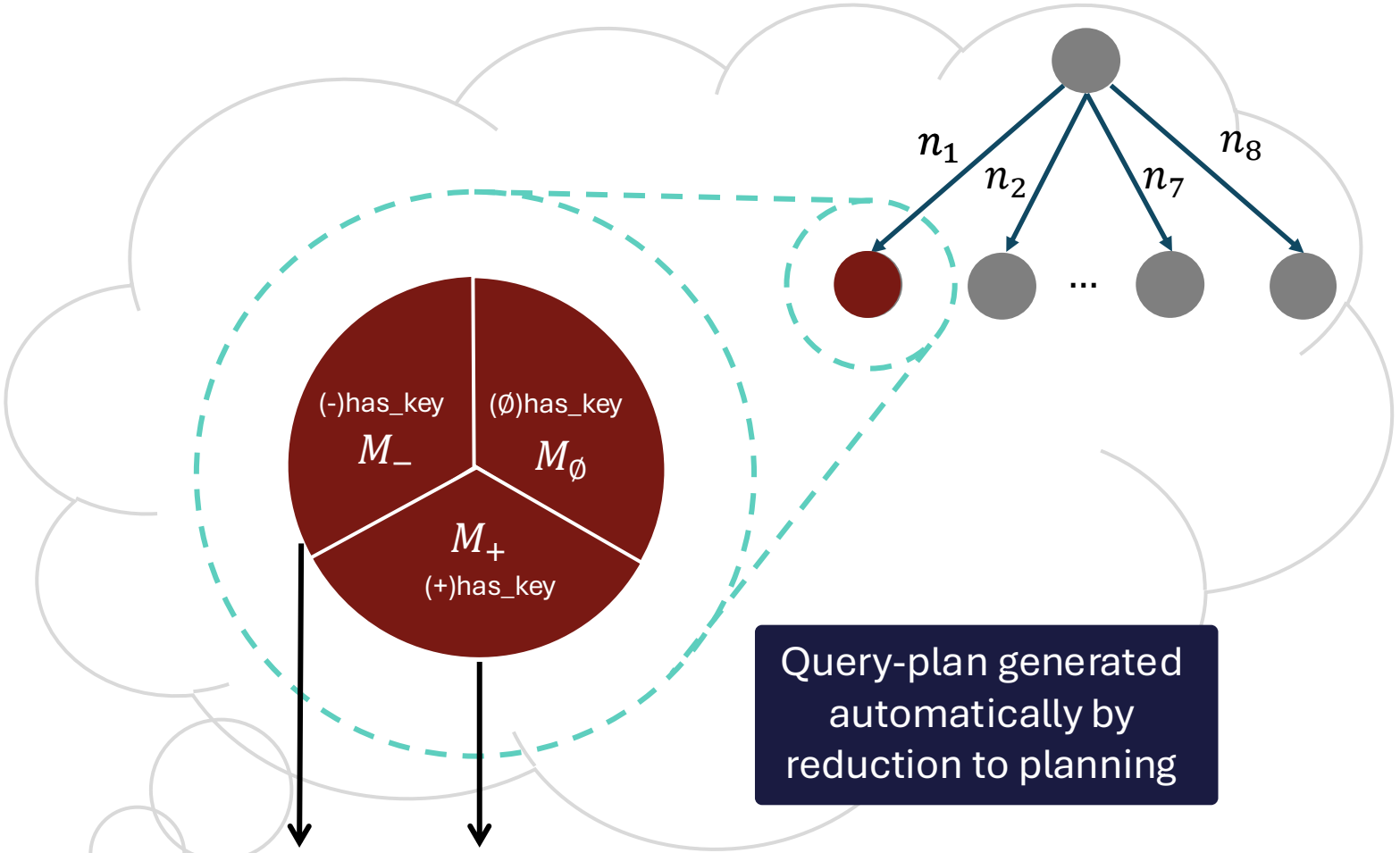
Query	In state s_I, what will happen if you execute the plan $\pi = \langle c_1, \ldots, c_n \rangle$?
Response	I can execute first ℓ steps of the plan, ending up in state s_F.

Plan Outcome Queries

- How to generate the queries?
- How to use the responses to generate models?

We have a reduction that converts this to a planning problem, so it automatically generates queries.

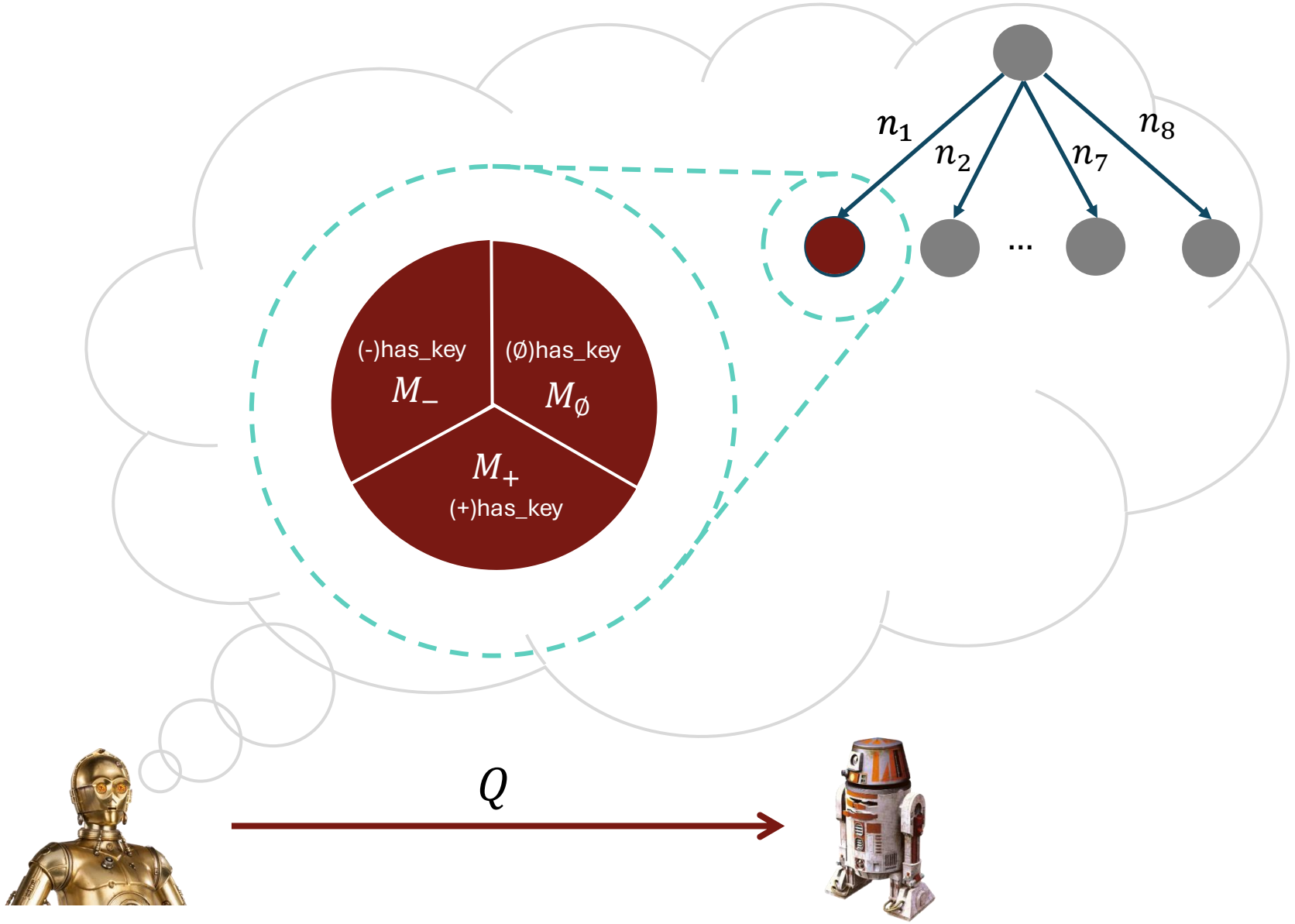
Hierarchical Query Synthesis



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

[**Verma**, Marpally, Srivastava; AAAI '21]

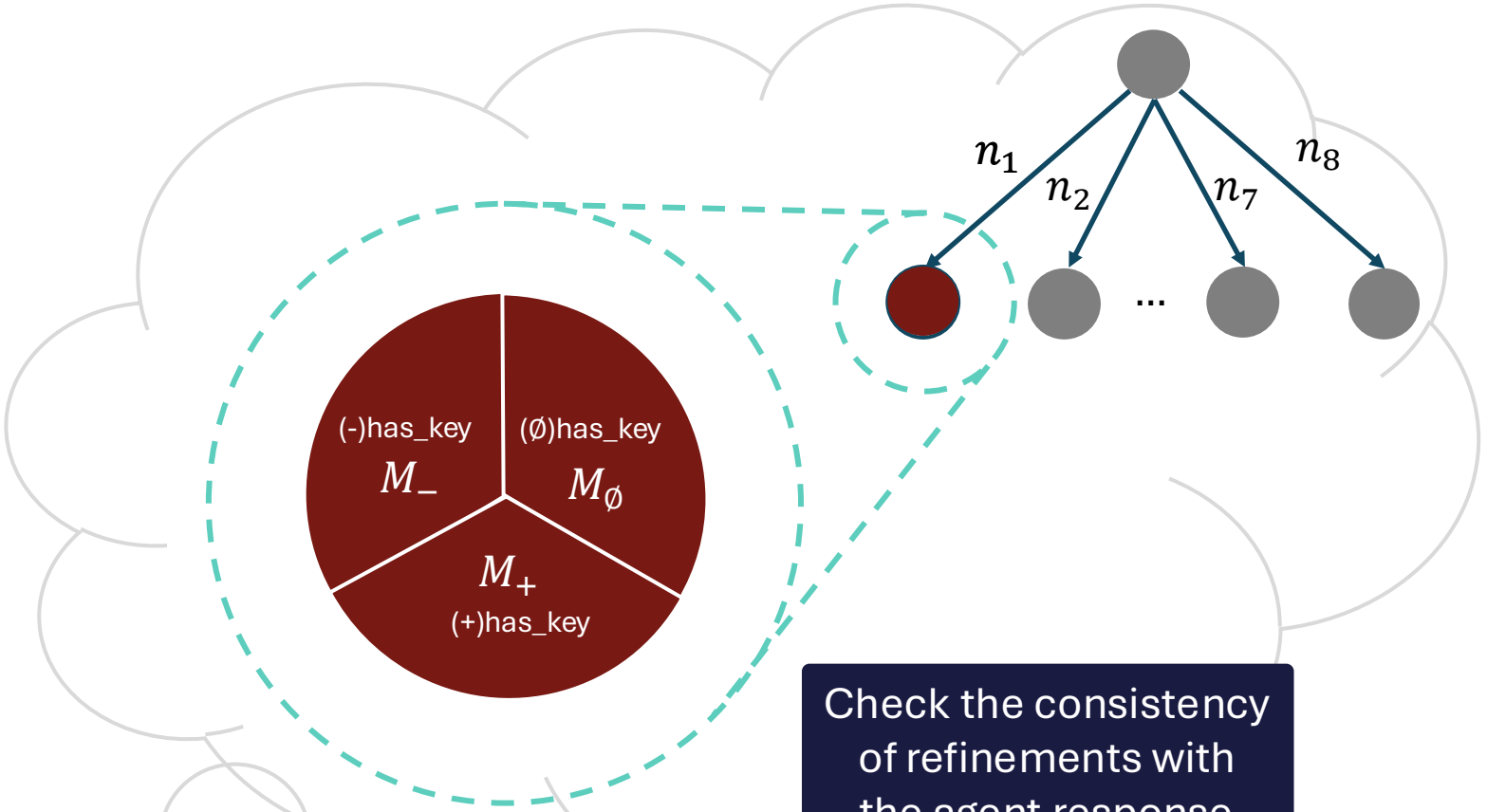
Hierarchical Query Synthesis




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

[**Verma**, Marpally, Srivastava; AAAI '21]

Hierarchical Query Synthesis



Check the consistency of refinements with the agent response

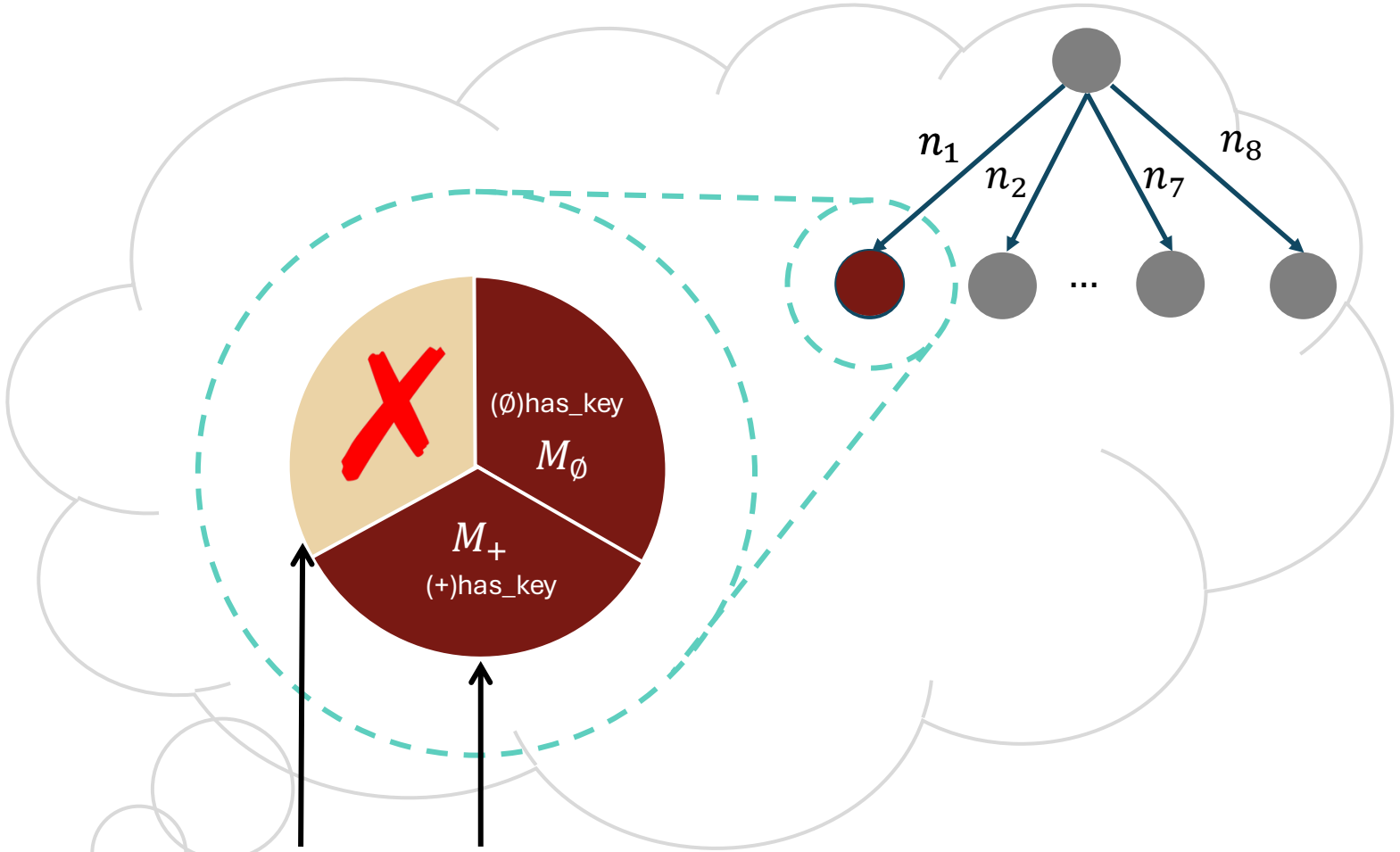
$$\theta = Q(Agent)$$

$$Q(M_-) \neq Q(M_+)$$

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

[**Verma**, Marpally, Srivastava; AAAI '21]

Hierarchical Query Synthesis

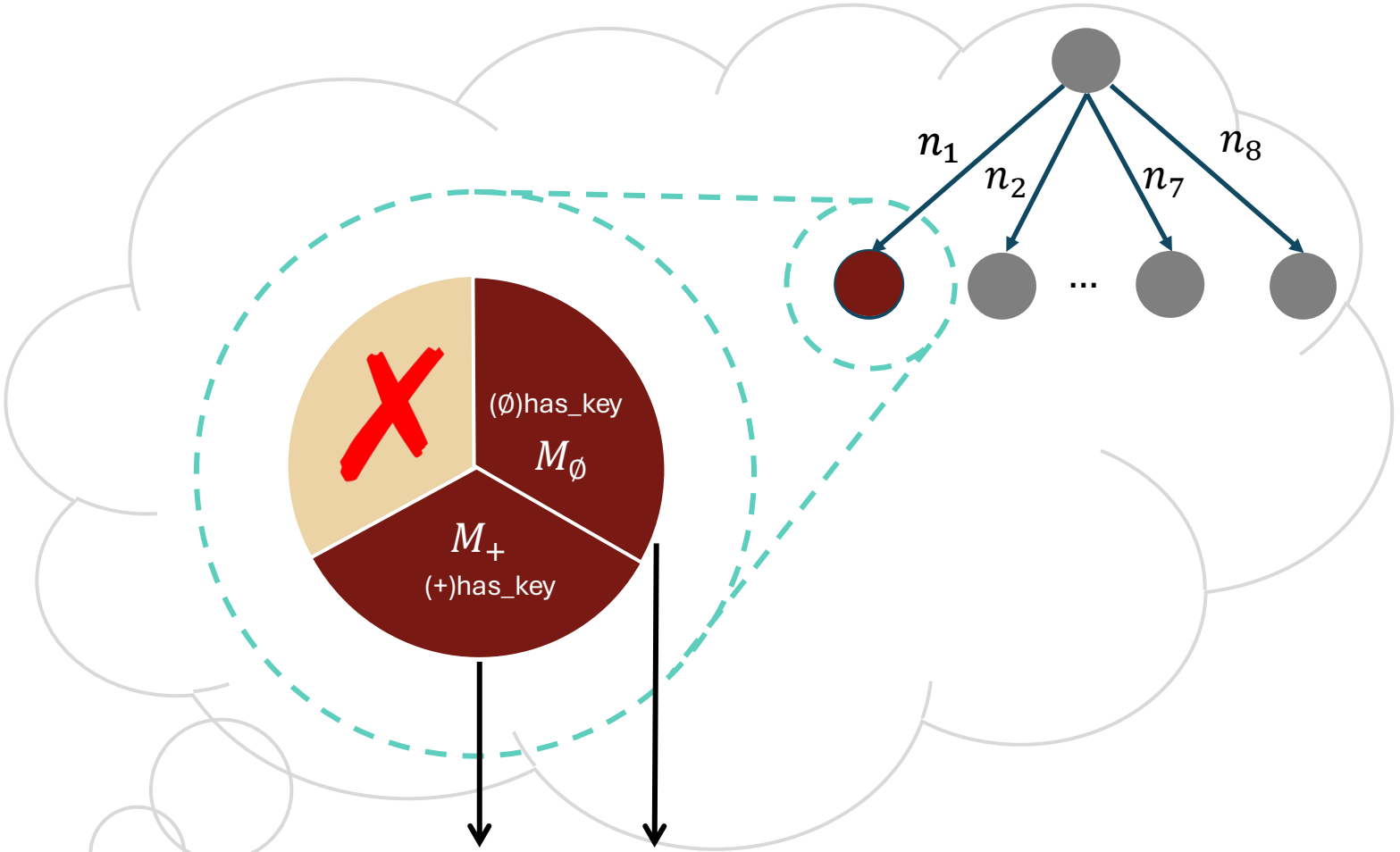



```
(:action open-door
  :parameters (?l1)
  :precondition (and
n1 (+/-/∅)(has_key)
n2 (+/-/∅)(door_open)
n3 (+/-/∅)(door_adjacent ?l1)
n4 (+/-/∅)(player_at ?l1))
  :effect (and
n5 (+/-/∅)(has_key)
n6 (+/-/∅)(door_open)
n7 (+/-/∅)(door_adjacent ?l1)
n8 (+/-/∅)(player_at ?l1))
```

Reject abstract model(s) that are not consistent with the agent

[**Verma**, Marpally, Srivastava; AAAI '21]

Hierarchical Query Synthesis

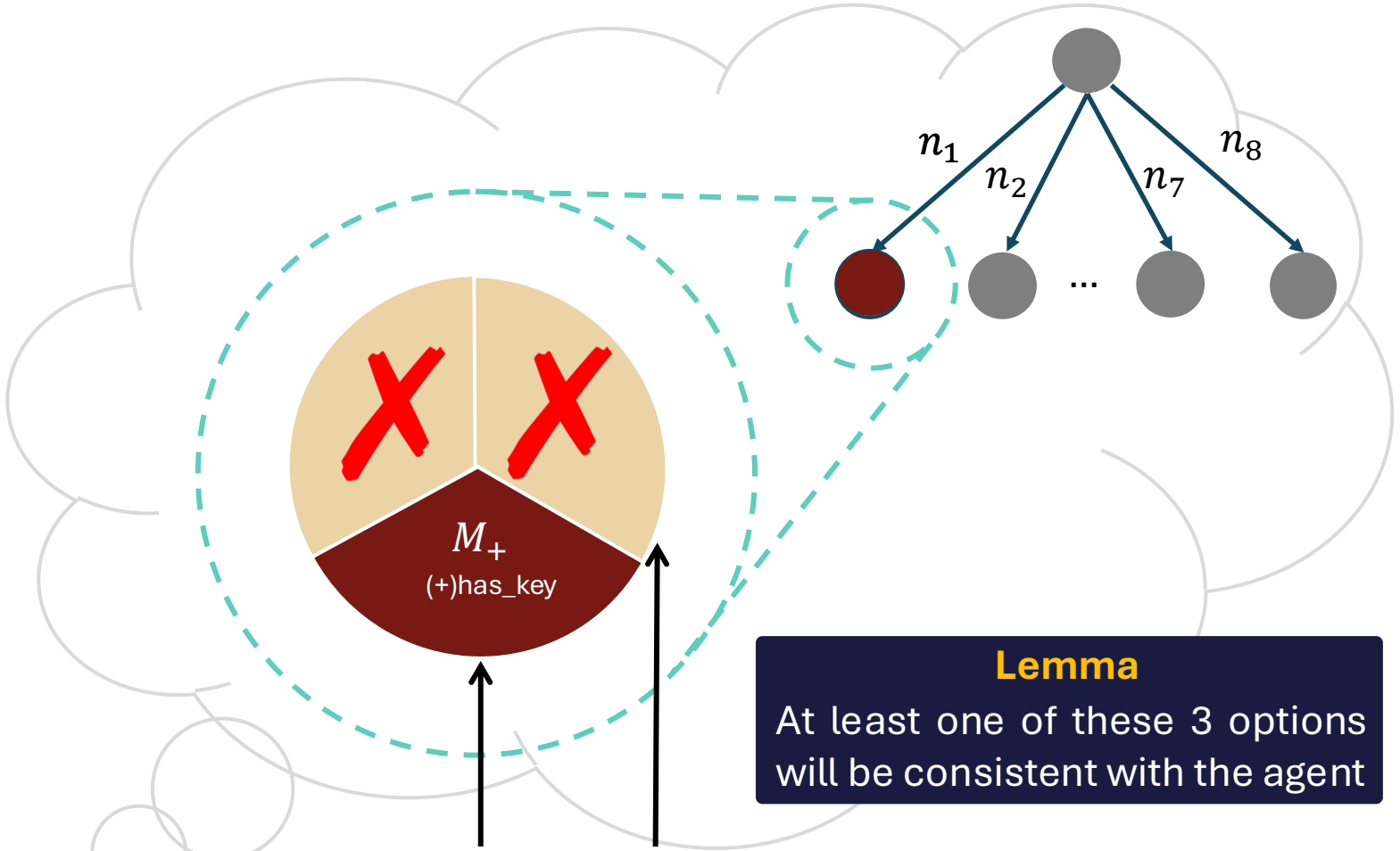



```
(:action open-door
  :parameters (?l1)
  :precondition (and
n1 (+/-/∅)(has_key)
n2 (+/-/∅)(door_open)
n3 (+/-/∅)(door_adjacent ?l1)
n4 (+/-/∅)(player_at ?l1))
  :effect (and
n5 (+/-/∅)(has_key)
n6 (+/-/∅)(door_open)
n7 (+/-/∅)(door_adjacent ?l1)
n8 (+/-/∅)(player_at ?l1))
```

Generate a distinguishing query for these two abstract models

[**Verma**, Marpally, Srivastava; AAAI '21]

Hierarchical Query Synthesis



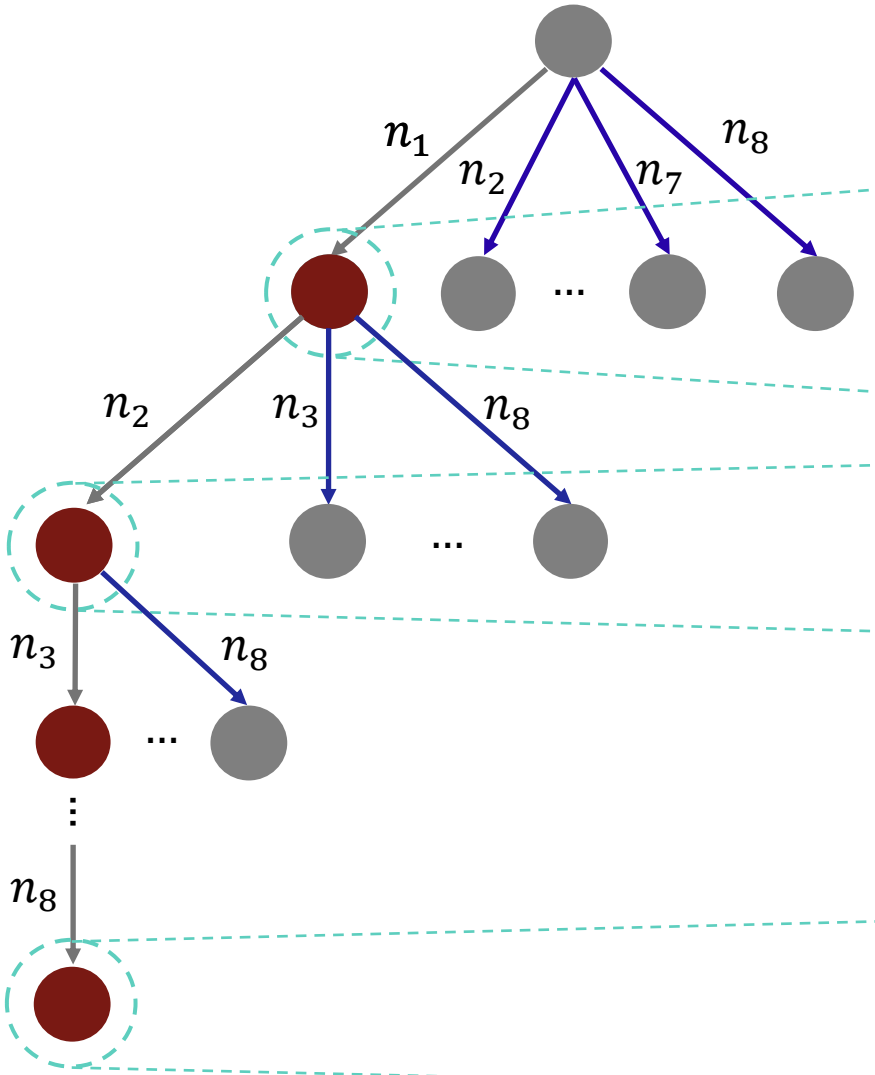
Lemma
At least one of these 3 options will be consistent with the agent

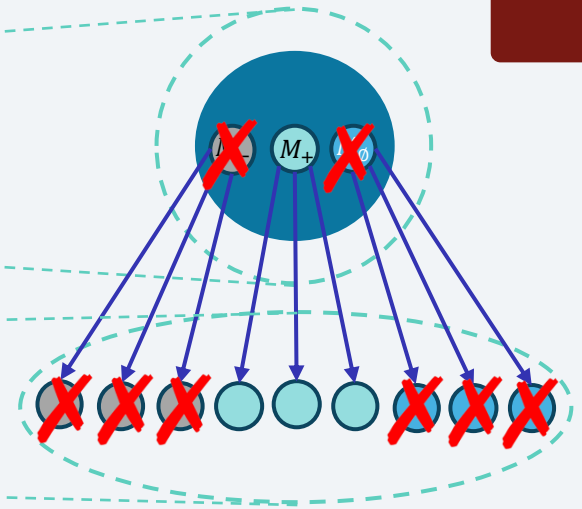
```
(:action open-door
  :parameters (?l1)
  :precondition (and
n1      (+)(has_key)
n2   (+/-/∅)(door_open)
n3   (+/-/∅)(door_adjacent ?l1)
n4   (+/-/∅)(player_at ?l1))
  :effect (and
n5   (+/-/∅)(has_key)
n6   (+/-/∅)(door_open)
n7   (+/-/∅)(door_adjacent ?l1)
n8   (+/-/∅)(player_at ?l1))
```

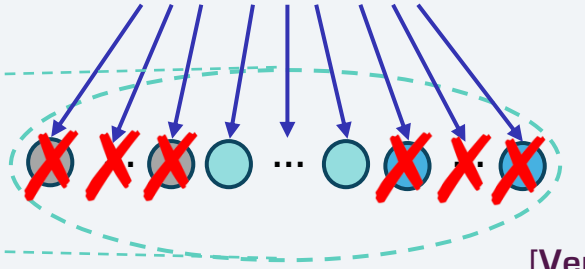
Reject the abstract model
that is not consistent with the agent

[**Verma**, Marpally, Srivastava; AAAI '21]

Hierarchical Query Synthesis







Key feature of the algorithm

Whenever we prune an abstract model, we prune a large number of concrete models.

Active Learning

[**Verma**, Marpally, Srivastava; AAAI '21]

Recap: Agent Interrogation Algorithm

Logistics

- 3 Exams:
  1. [Quiz 1] Feb 19: 08:00 AM – 08:50 AM
  2. [Quiz 2] Apr 10: 05:00 PM – 05:50 PM
  3. [End Sem] May 12

- Office Hours: 1 hour after class every week Tu, W, Th
  - No office hours today

- ROS based Assignment
  - https://wiki.ros.org/ROS/Tutorials

Exponential Search for Learning Correct Description

- Consider the following 3 predicates:
  - `(on ?x ?y)`
  - `(clear ?x)`
  - `(ontable ?x)`

- Consider just one action:
  `(stack ?x ?y)`

```
(:action stack
  :parameters (?x ?y)
  :precondition (and
     (clear ?x)
     (clear ?y))
  :effect (and
     (on ?x ?y)
     (not (clear ?y))))
```

Agent Interrogation Algorithm

```
(:action stack
   :parameters (?x ?y)
   :precondition (and )
   :effect (and )
)
```

Algorithm 1 Agent Interrogation Algorithm (AIA)

Input: $\mathcal{A}, \mathbb{A}_H, \mathbb{P}^*, \mathbb{S}$
Output: poss_models
Initialize poss_models $= \{\phi\}$
for γ in some input pal ordering Γ **do**
  new_models $\leftarrow$ poss_models
  pruned_models$= \{\}$
  for each $\mathcal{M}'$ in new_models **do**
    for each pair $\{i, j\}$ in $\{+, -, \emptyset\}$ **do**
      $\mathcal{Q}, \mathcal{M}_i, \mathcal{M}_j \leftarrow$ generate_query$(\mathcal{M}', i, j, \gamma, \mathbb{S})$
      $\mathcal{M}_{prune} \leftarrow$filter_models$(\mathcal{Q}, \mathcal{M}^{\mathcal{A}}, \mathcal{M}_i, \mathcal{M}_j)$
      pruned_models$\leftarrow$ pruned_models $\cup\, \mathcal{M}_{prune}$
    end for
  end for
  if pruned_models is $\emptyset$ **then**
    update_pal_ordering$(\Gamma, \mathbb{S})$
    continue
  end if
  poss_models $\leftarrow$ new_models $\times \{\gamma^+, \gamma^-, \gamma^{\emptyset}\} \setminus$ pruned_models
end for

Agent Interrogation Algorithm

```
(:action stack
   :parameters (?x ?y)
   :precondition (and )
   :effect (and )
)
```

$\mathbb{S}$ = {{¬clear(A),clear(B), ontable(A), ontable(B)},}

$poss\_models$ = $\{\phi\}$

Algorithm 1 Agent Interrogation Algorithm (AIA)

Input: $\mathcal{A}, \mathbb{A}_H, \mathbb{P}^*, \mathbb{S}$
Output: poss_models
Initialize poss_models $= \{\phi\}$
for γ in some input pal ordering Γ **do**
    new_models $\leftarrow$ poss_models
    pruned_models$= \{\}$
    for each $\mathcal{M}'$ in new_models **do**
        for each pair $\{i, j\}$ in $\{+, -, \emptyset\}$ **do**
            $\mathcal{Q}, \mathcal{M}_i, \mathcal{M}_j \leftarrow$ generate_query$(\mathcal{M}', i, j, \gamma, \mathbb{S})$
            $\mathcal{M}_{prune} \leftarrow$filter_models$(\mathcal{Q}, \mathcal{M}^{\mathcal{A}}, \mathcal{M}_i, \mathcal{M}_j)$
            pruned_models$\leftarrow$ pruned_models $\cup\, \mathcal{M}_{prune}$
        end for
    end for
    if pruned_models is $\emptyset$ **then**
        update_pal_ordering$(\Gamma, \mathbb{S})$
        continue
    end if
    poss_models $\leftarrow$ new_models $\times \{\gamma^+, \gamma^-, \gamma^\emptyset\} \setminus$ pruned_models
end for

PAL : $\langle p, a, \ell \rangle$

```
a = {stack (?x ?y)}
p = {(clear ?x), (clear ?y),
(ontable ?x), (ontable ?y),
(on ?x ?y), (on ?y ?x)}
ℓ = {pre, eff}

Γ = {⟨(clear ?x), stack (?x ?y), pre⟩,
⟨(clear ?x), stack (?x ?y), eff⟩,
⟨(clear ?y), stack (?x ?y), pre⟩,
⟨(clear ?y), stack (?x ?y), eff⟩,
⟨(ontable ?x), stack (?x ?y), pre⟩,
⟨(ontable ?x), stack (?x ?y), eff⟩,
⟨(ontable ?y), stack (?x ?y), pre⟩,
⟨(ontable ?y), stack (?x ?y), eff⟩,
⟨(on ?x ?y), stack (?x ?y), pre⟩,
⟨(on ?x ?y), stack (?x ?y), eff⟩,
⟨(on ?y ?x), stack (?x ?y), pre⟩,
⟨(on ?y ?x), stack (?x ?y), eff⟩}
```

Algorithm 1 Agent Interrogation Algorithm (AIA)

Input: $\mathcal{A}, \mathbb{A}_H, \mathbb{P}^*, \mathbb{S}$
Output: poss_models
Initialize poss_models $= \{\phi\}$
for γ in some input pal ordering Γ **do**
  new_models $\leftarrow$ poss_models
  pruned_models$= \{\}$
  for each $\mathcal{M}'$ in new_models **do**
    for each pair $\{i, j\}$ in $\{+, -, \emptyset\}$ **do**
      $\mathcal{Q}, \mathcal{M}_i, \mathcal{M}_j \leftarrow$ generate_query$(\mathcal{M}', i, j, \gamma, \mathbb{S})$
      $\mathcal{M}_{prune} \leftarrow$filter_models$(\mathcal{Q}, \mathcal{M}^{\mathcal{A}}, \mathcal{M}_i, \mathcal{M}_j)$
      pruned_models$\leftarrow$ pruned_models $\cup\, \mathcal{M}_{prune}$
    end for
  end for
  if pruned_models is $\emptyset$ **then**
    update_pal_ordering$(\Gamma, \mathbb{S})$
    continue
  end if
  poss_models $\leftarrow$ new_models $\times \{\gamma^+, \gamma^-, \gamma^\emptyset\} \setminus$ pruned_models
end for

PAL : $\langle p, a, \ell \rangle$

```
a = {stack (?x ?y)}
p = {(clear ?x), (clear ?y),
(ontable ?x), (ontable ?y),
(on ?x ?y), (on ?y ?x)}
ℓ = {pre, eff}

γ = ⟨(clear ?x), stack (?x ?y), pre⟩
{i,j} ∈ {{+,-}, {+, ∅}, {-, ∅}}
```

Algorithm 1 Agent Interrogation Algorithm (AIA)

Input: $\mathcal{A}, \mathbb{A}_H, \mathbb{P}^*, \mathbb{S}$
Output: poss_models
Initialize poss_models $= \{\phi\}$
for γ in some input pal ordering Γ **do**
    new_models $\leftarrow$ poss_models
    pruned_models$= \{\}$
    for each $\mathcal{M}'$ in new_models **do**
        for each pair $\{i, j\}$ in $\{+, -, \emptyset\}$ **do**
            $\mathcal{Q}, \mathcal{M}_i, \mathcal{M}_j \leftarrow$ generate_query$(\mathcal{M}', i, j, \gamma, \mathbb{S})$
            $\mathcal{M}_{prune} \leftarrow$filter_models$(\mathcal{Q}, \mathcal{M}^{\mathcal{A}}, \mathcal{M}_i, \mathcal{M}_j)$
            pruned_models$\leftarrow$ pruned_models $\cup\, \mathcal{M}_{prune}$
        end for
    end for
    if pruned_models is $\emptyset$ **then**
        update_pal_ordering$(\Gamma, \mathbb{S})$
        continue
    end if
    poss_models $\leftarrow$ new_models $\times \{\gamma^+, \gamma^-, \gamma^\emptyset\} \setminus$ pruned_models
end for

Generate Query

γ = ⟨(clear ?x), stack (?x ?y), pre⟩
{i,j} = {+,-}

a = {stack (?x ?y)}
p = {(clear ?x)}
ℓ = {pre}

s_0 ={¬clear(A),clear(B),
ontable(A), ontable(B)}
s_0' = {¬clear(A),clear(B)}
π = ⟨stack (A, B)⟩

Algorithm 1 Agent Interrogation Algorithm (AIA)

Input: $\mathcal{A}, \mathbb{A}_H, \mathbb{P}^*, \mathbb{S}$
Output: poss_models
Initialize poss_models $= \{\phi\}$
for γ in some input pal ordering Γ **do**
  new_models $\leftarrow$ poss_models
  pruned_models$= \{\}$
  for each $\mathcal{M}'$ in new_models **do**
    for each pair $\{i, j\}$ in $\{+, -, \emptyset\}$ **do**
      $\mathcal{Q}, \mathcal{M}_i, \mathcal{M}_j \leftarrow$ generate_query$(\mathcal{M}', i, j, \gamma, \mathbb{S})$
      $\mathcal{M}_{prune} \leftarrow$filter_models$(\mathcal{Q}, \mathcal{M}^{\mathcal{A}}, \mathcal{M}_i, \mathcal{M}_j)$
      pruned_models$\leftarrow$ pruned_models $\cup\, \mathcal{M}_{prune}$
    end for
  end for
  if pruned_models is $\emptyset$ **then**
    update_pal_ordering$(\Gamma, \mathbb{S})$
    continue
  end if
  poss_models $\leftarrow$ new_models $\times \{\gamma^+, \gamma^-, \gamma^\emptyset\} \setminus$ pruned_models
end for

Predicate Abstraction

- If the agent can't execute the query, we cannot prune the models.
- A consistent model might say it can execute the whole plan in the query whereas the ground truth model might not be able to execute.
- Why?
  - Predicate Abstraction can make more plans feasible in abstract model.

Predicate Abstraction