AAAI 2020 Workshop on Generalization in Planning

Learning Generalized Models by Interrogating Black-Box Autonomous Agents

Pulkit Verma and **Siddharth Srivastava**
School of Computing, Informatics, and Decision Systems Engineering,
Arizona State University,
Tempe, AZ, USA

Introduction

- How would a lay user determine whether an AI agent will be safe/reliable for a certain task?

- More challenging in settings where:
  - agent may receive updates (non-stationary)
  - agent's internal code is not available (black-box)

Introduction

- How would a lay user determine whether an AI agent will be safe/reliable for a certain task?

- More challenging in settings where:
  - agent may receive updates (non-stationary)
  - agent's internal code is not available (black-box)

- How do we assess reliability of humans in similar scenarios?

Introduction

Problem setup

- Non-stationary black-box agents

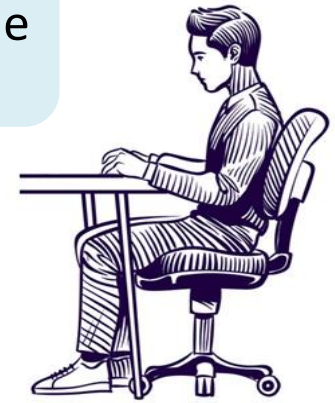
Our Approach

- Generate an interrogation policy
- Use agent's answers to representation agnostic queries to estimate an understandable relational model.
  - In this paper: STRIPS-like model.

Queries and Responses

What do you think will happen if you pickup bottle b1, then pickup bottle b2 starting in an state where your hand is empty and bottle b1 and b2 are on the floor?

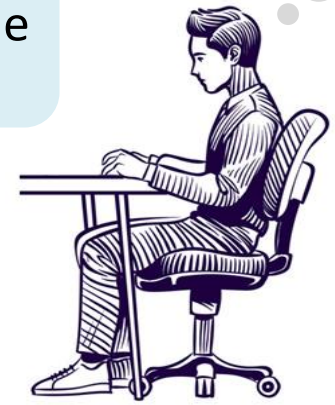
I can execute the first step of picking up bottle b1, after this my hand won't be empty and I'll be holding bottle b1.

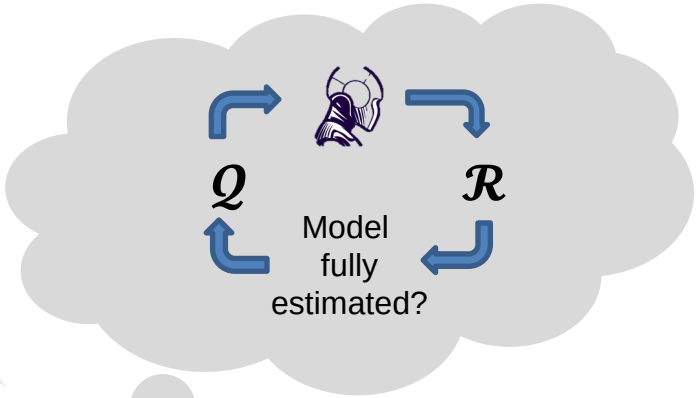


Queries and Responses

What do you think will happen if you pickup bottle b1, then pickup bottle b2 starting in an state where your hand is empty and bottle b1 and b2 are on the floor?

I can execute the first step of picking up bottle b1, after this my hand won't be empty and I'll be holding bottle b1.





Agent Interrogation Task

- Assume the set of predicates $\mathbb{P}$ and actions $\mathbb{A}$ are already known.

- Key Technical Challenges:
  - Which queries to ask?
  - In which order to ask them?

Query

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

Plan Outcome Queries:

Our Query: $\langle s_I, \pi \rangle$
Initial State and Plan

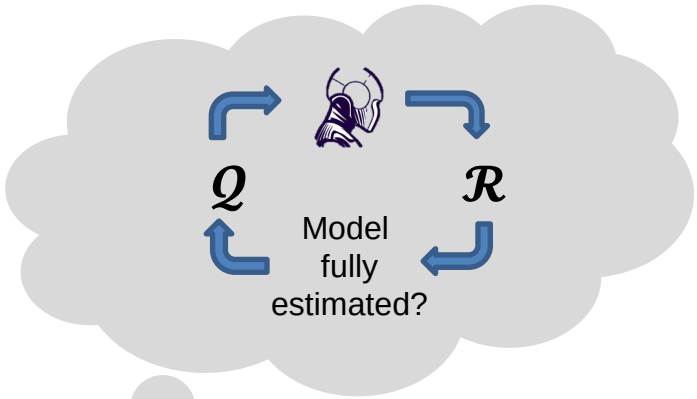
Agent's Response: $\langle \ell, s_F \rangle$
Length of plan that can be executed successfully and the final state.

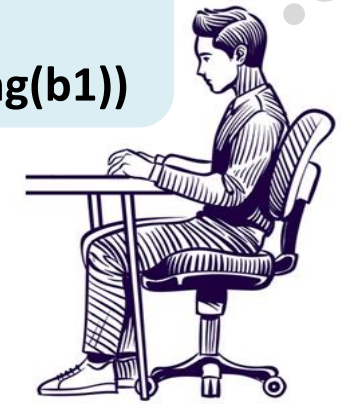
Agent may compute response using an analytical model or a simulator.

Queries and Responses

What do you think will happen if you execute the plan π: **⟨pickup(b1), pickup(b2)⟩** starting in an initial state s_I: **(ontable(b1) ^ (ontable(b2) ^ handempty)**?

I can execute only the first step ($\ell = \mathbf{1}$), and the final state after executing one step was s_F:**(¬ontable(b1) ^ (ontable(b1) ^ ¬handempty ^ holding(b1))**

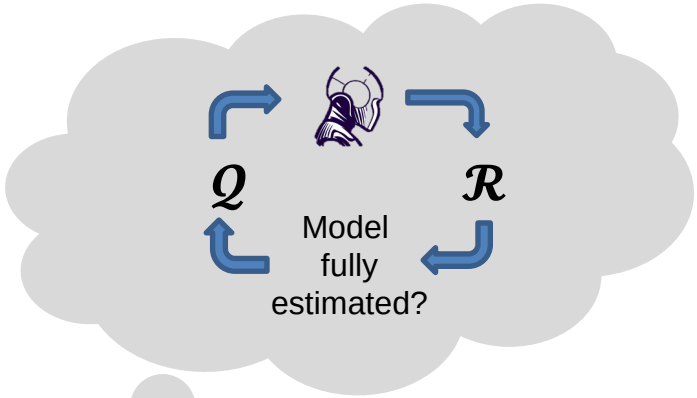




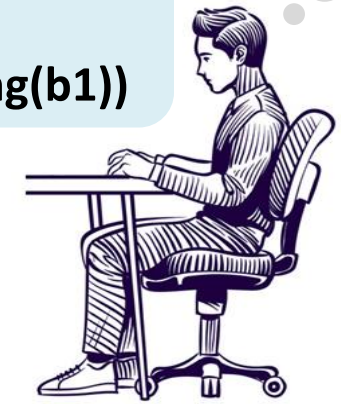
Queries and Responses

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

What do you think will happen if you execute the plan π: **⟨pickup(b1), pickup(b2)⟩** starting in an initial state s_I: **(ontable(b1) ^ (ontable(b2) ^ handempty)**?



I can execute only the first step ($\ell = \mathbf{1}$), and the final state after executing one step was
s_F:**(¬ontable(b1) ^ (ontable(b1) ^ ¬handempty ^ holding(b1))**



Abstraction

```
(define (domain blocksworld)
(:requirements :strips)
(:predicates (clear ?x)
             (ontable ?x)
             (handempty)
             (holding ?x)
(:action pickup
 :parameters (?ob)
 :precondition (and (handempty)
                    (ontable ?ob))
 :effect (and (not (handempty))
              (not (ontable ?ob)))))
```

Concrete model

abstraction

This predicate can appear in three forms:
- positive literal
- negative literal
- absent

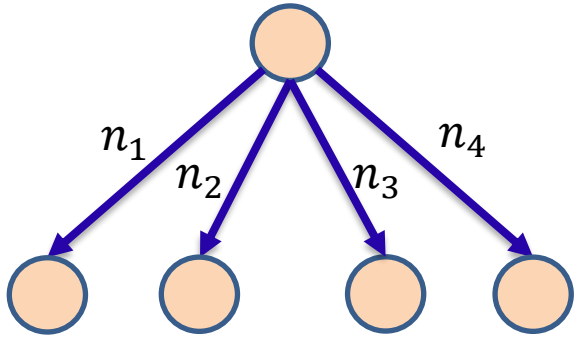
```
(define (domain blocksworld)
(:requirements :strips)
(:predicates (clear ?x)
             (ontable ?x)
             (handempty)
             (holding ?x)
(:action pickup
 :parameters (?ob)
 :precondition (and (handempty)
                    (+/-/∅)(ontable ?ob))
 :effect (and (not (handempty))
              (not (ontable ?ob)))))
```

Abstracted model

Algorithm

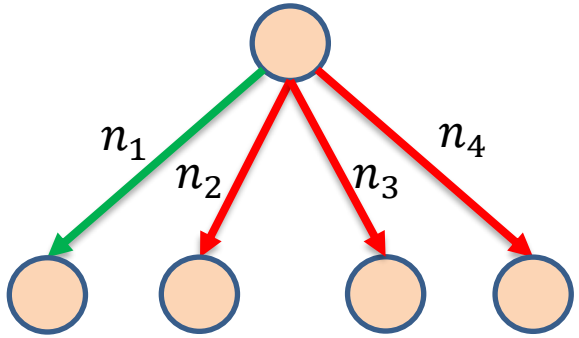
```
(:action pickup
  :parameters (?ob)
  :precondition (and (+/-/∅) (handempty)      n1
                     (+/-/∅) (ontable ?ob))   n2
  :effect (and (+/-/∅) (handempty)            n3
               (+/-/∅) (ontable ?ob)))        n4
```

Algorithm



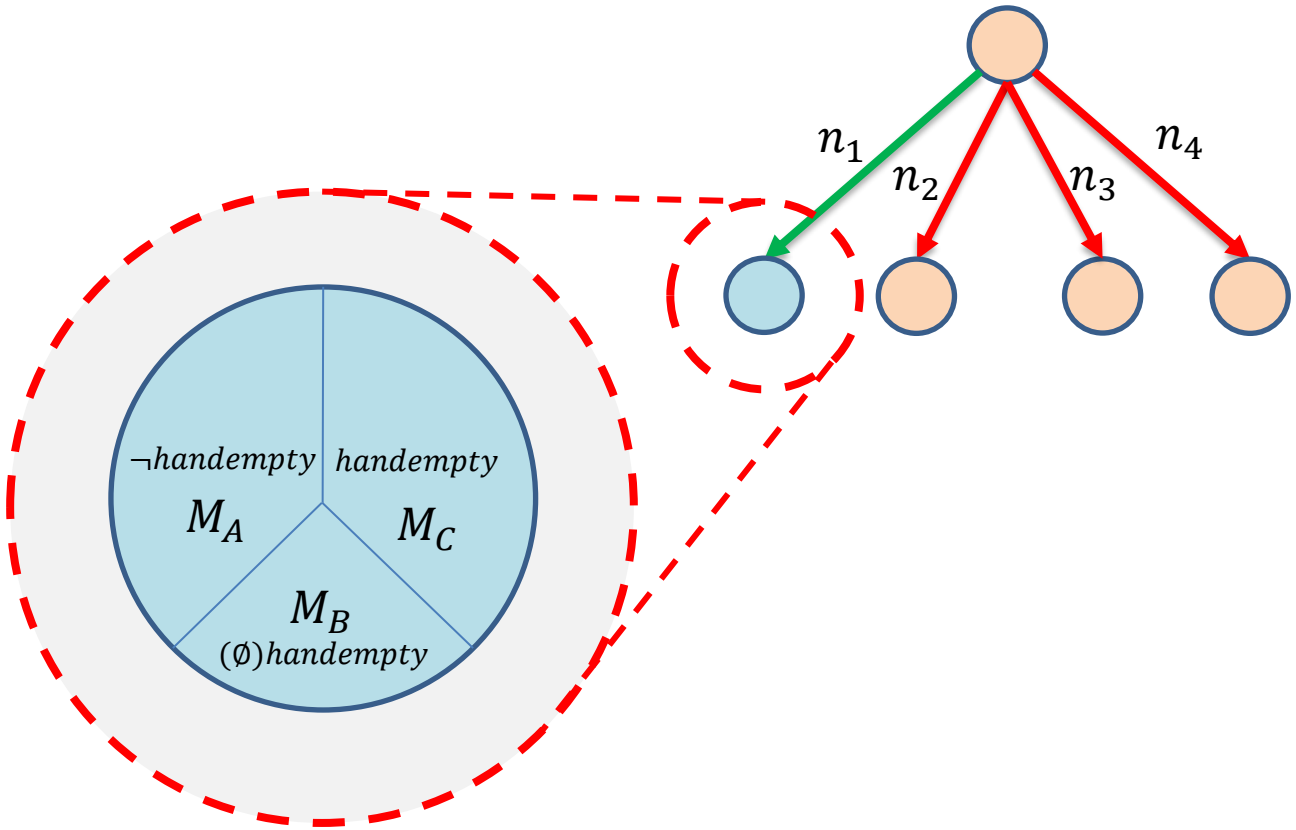

```
(:action pickup
  :parameters (?ob)
  :precondition (and (+/-/∅) (handempty)      n1
                     (+/-/∅) (ontable ?ob))   n2
  :effect (and (+/-/∅) (handempty)            n3
               (+/-/∅) (ontable ?ob)))        n4
```

Algorithm



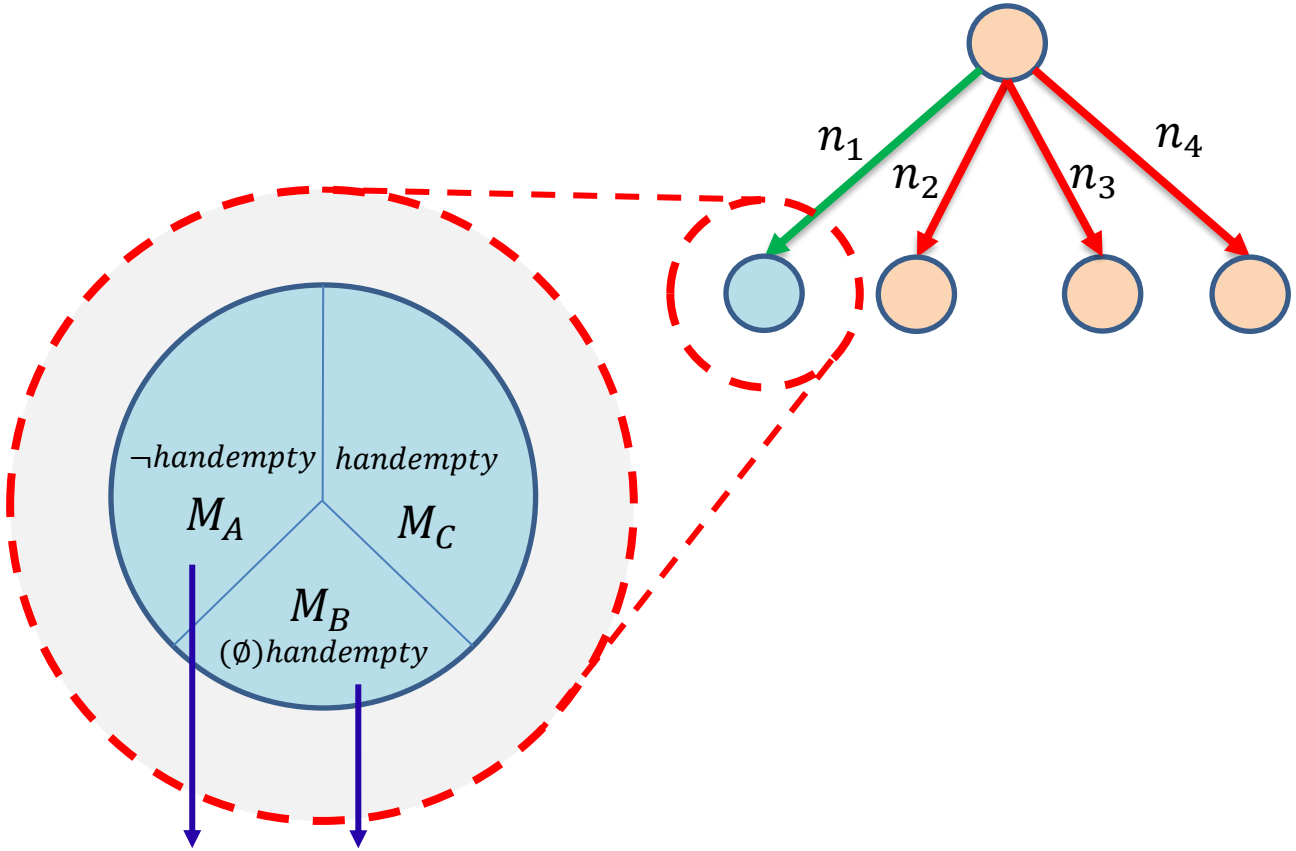

```
(:action pickup
  :parameters (?ob)
  :precondition (and (+/-/∅) (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

Algorithm




```
(:action pickup
  :parameters (?ob)
  :precondition (and (+/-/∅) (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

Algorithm

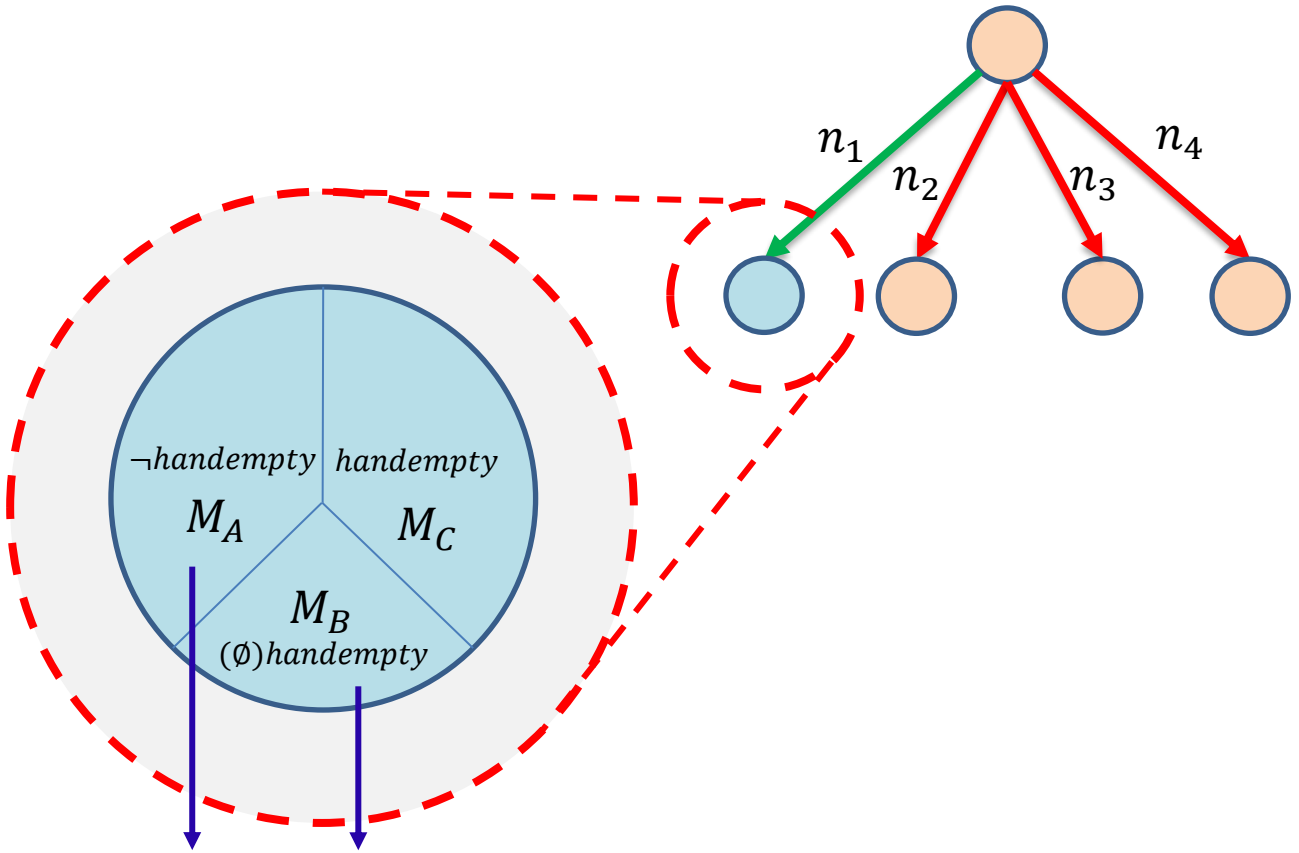



```
(:action pickup
  :parameters (?ob)
  :precondition (and (+/-/∅) (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

Generate a query that can differentiate between these two refinements,i.e., $\boldsymbol{Q}$ such that $\boldsymbol{Q}(M_A) \neq \boldsymbol{Q}(M_B)$

We reduce this query generation process to a planning problem
(see paper or visit poster for details)

Algorithm

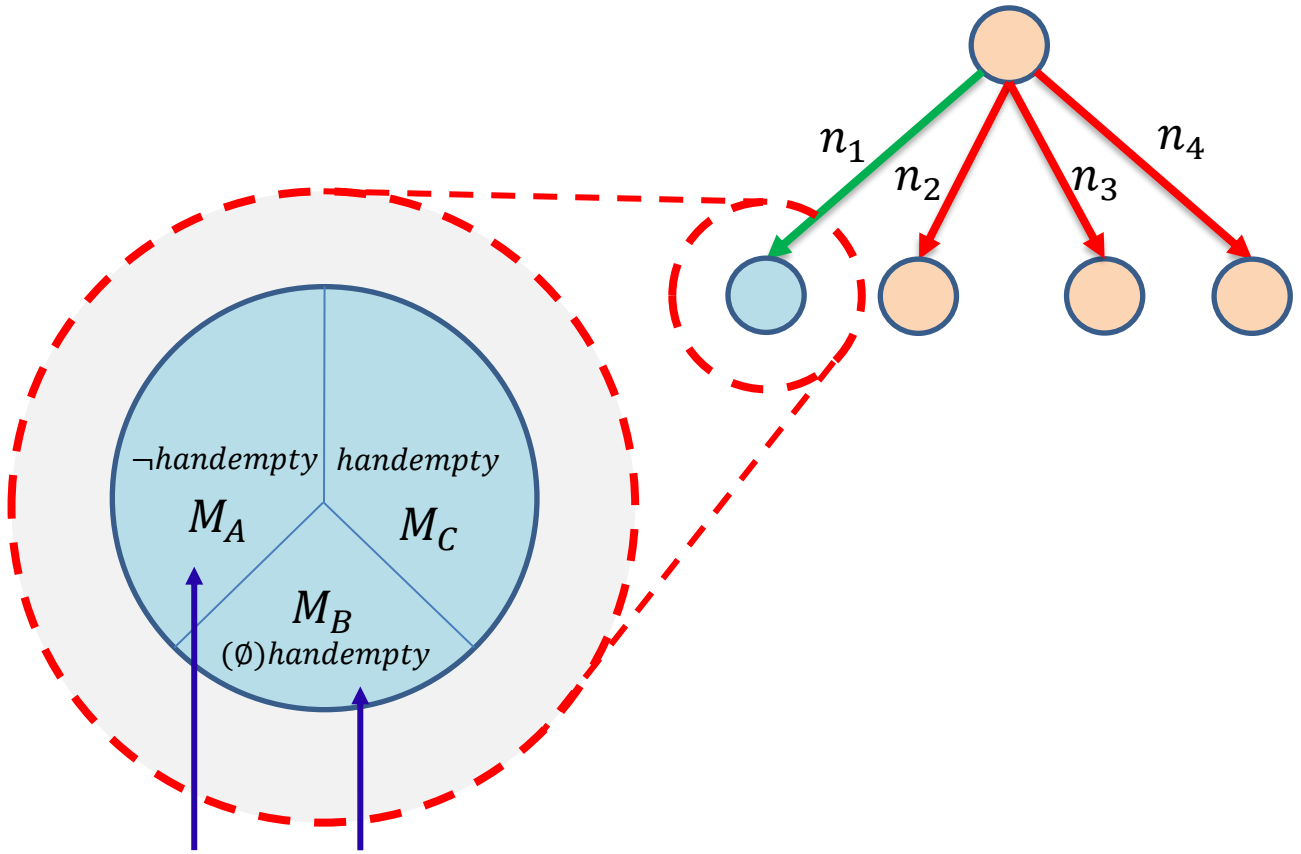



```
(:action pickup
  :parameters (?ob)
  :precondition (and (+/-/∅) (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

Q → 

Pose the query to the agent

Algorithm

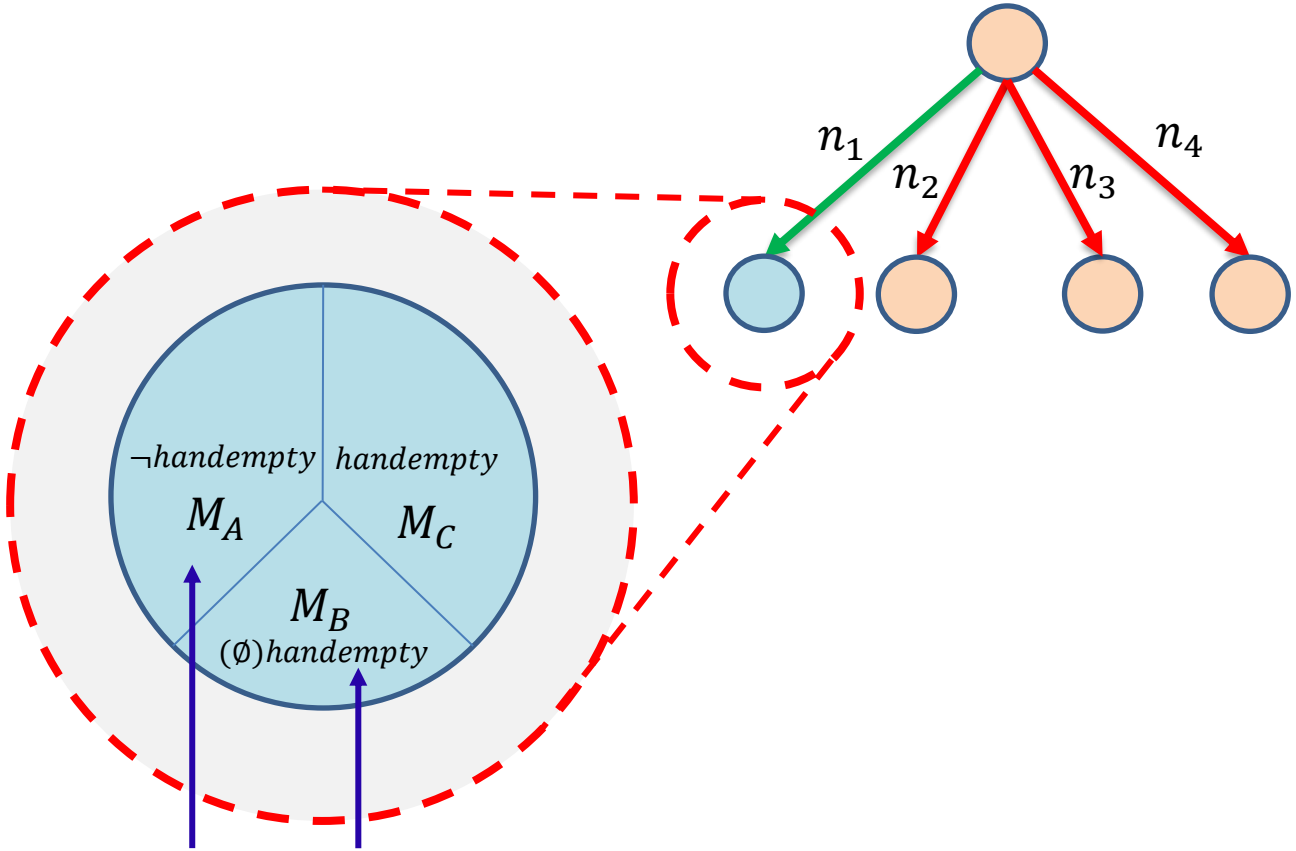



```
(:action pickup
  :parameters (?ob)
  :precondition (and (+/-/∅) (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

$\mathcal{R}$ ←

Check the consistency of refinements with the agent response

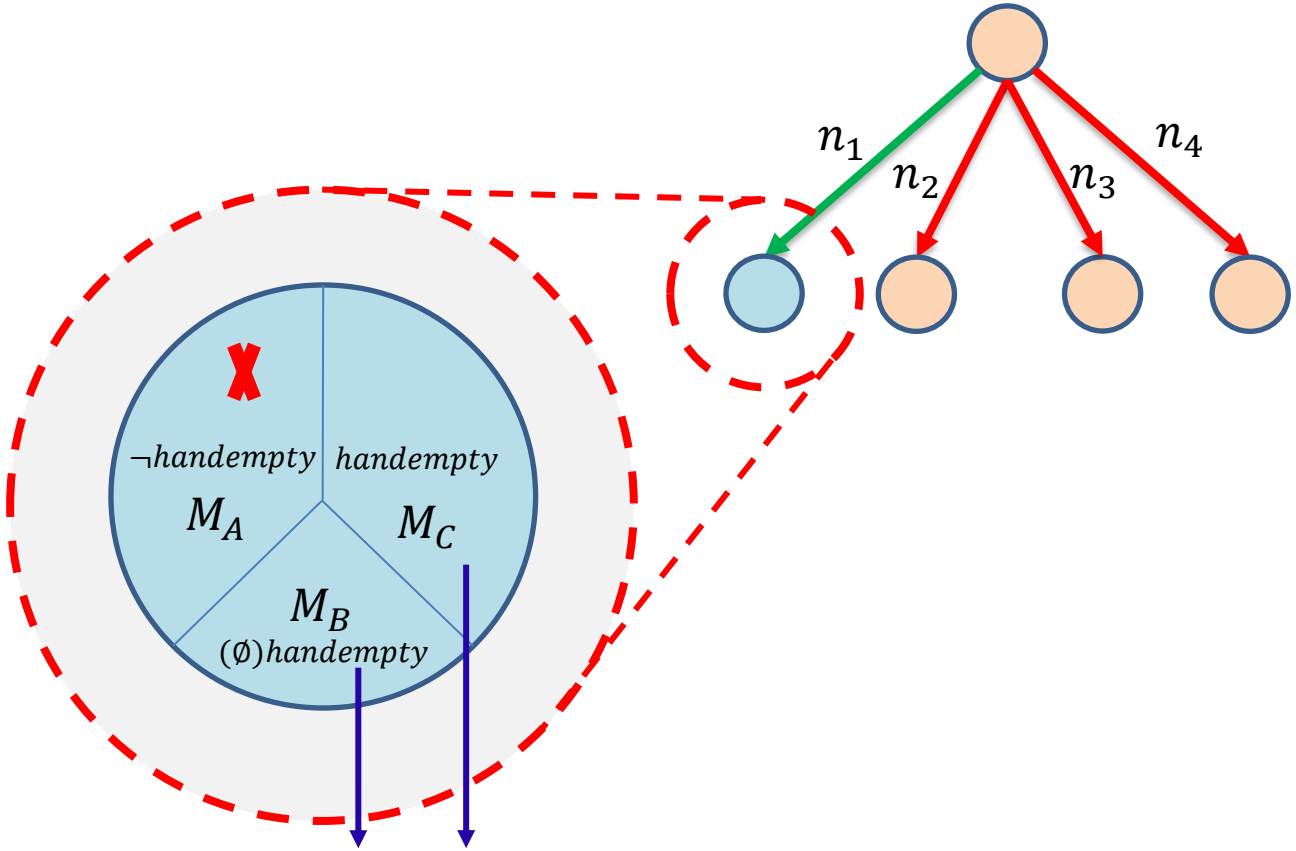
Algorithm



Reject one (or both) of the refinement(s) which is (are) not consistent with the agent

```
(:action pickup
  :parameters (?ob)
  :precondition (and (+/-/∅) (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

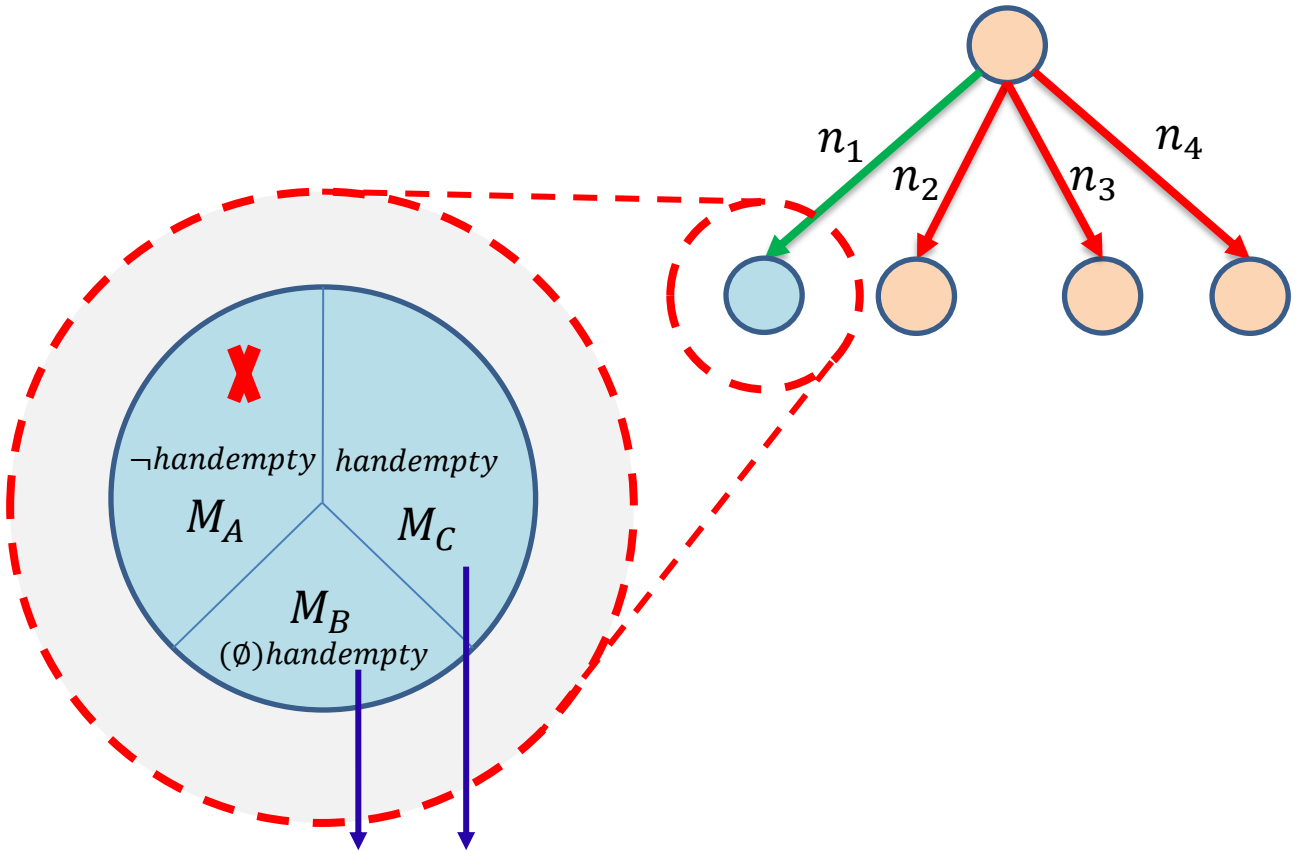
Algorithm



Generate a query that can differentiate between these two refinements,i.e., $\boldsymbol{Q}$ such that $\boldsymbol{Q}(M_B) \neq \boldsymbol{Q}(M_C)$

```
(:action pickup
  :parameters (?ob)
  :precondition (and (+/-/∅) (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

Algorithm

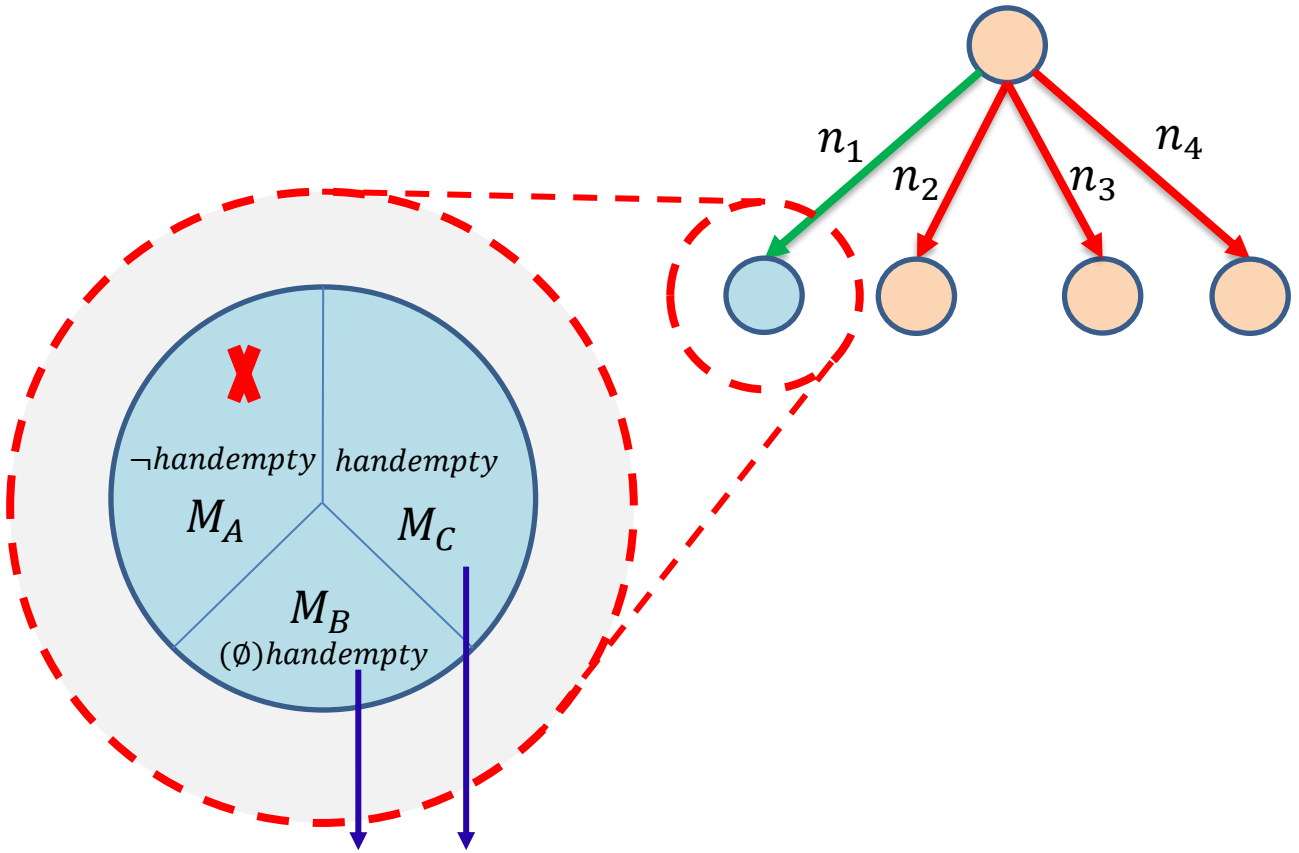



```
(:action pickup
  :parameters (?ob)
  :precondition (and (+/-/∅) (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

Q →

Pose the query to the agent

Algorithm

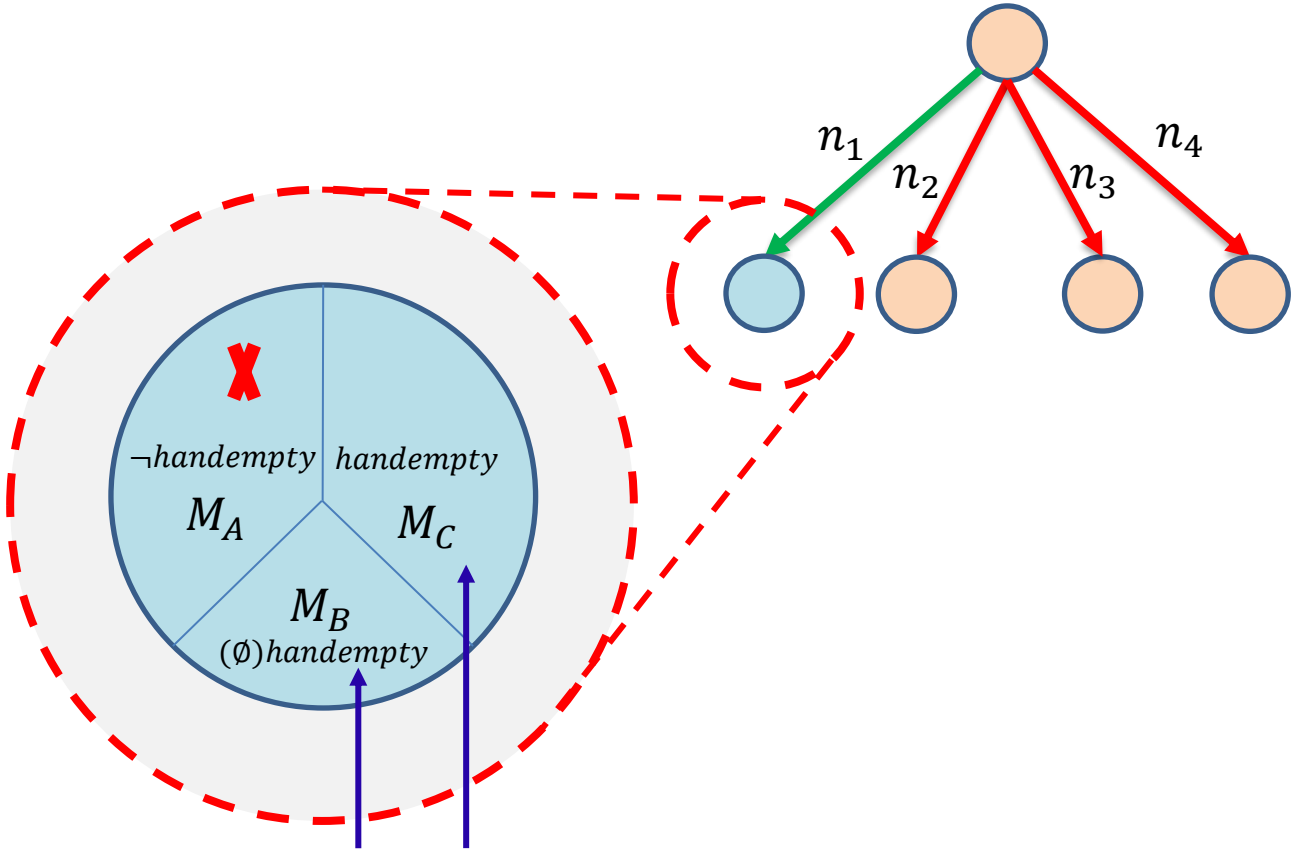



```
(:action pickup
  :parameters (?ob)
  :precondition (and (+/-/∅) (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

$\mathcal{R}$ →

Check the consistency of refinements with the agent response

Algorithm

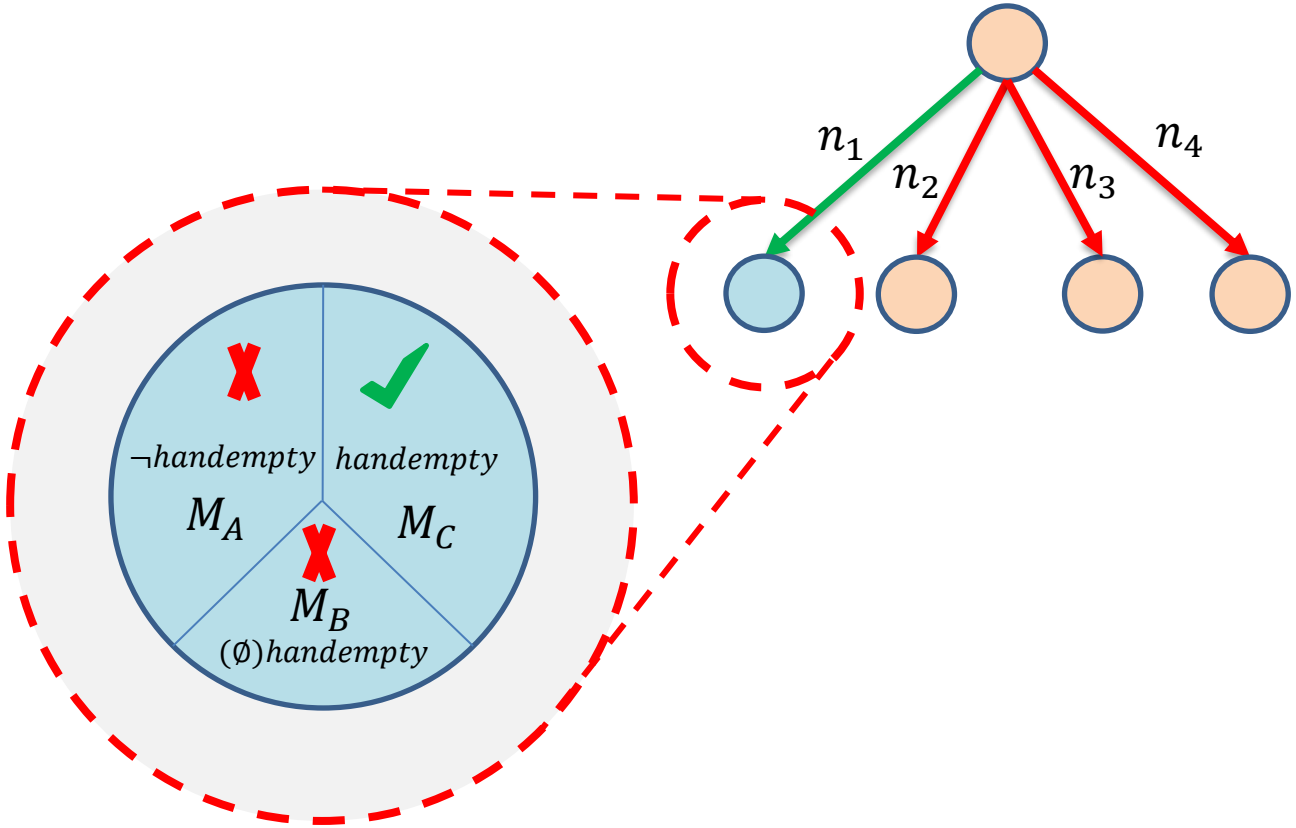


Reject the refinement
that is not consistent with the agent

```
(:action pickup
  :parameters (?ob)
  :precondition (and (+/-/∅) (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

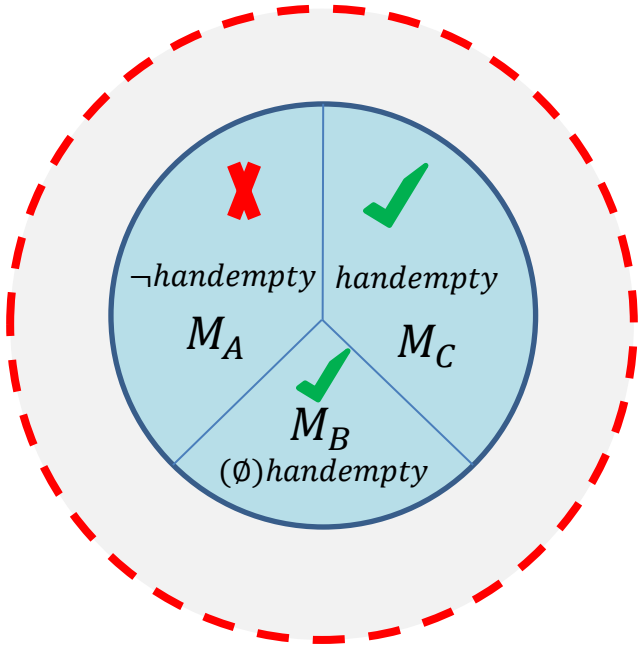
Lemma: At least one of the refinements will be consistent with the agent

Algorithm




```
(:action pickup
  :parameters (?ob)
  :precondition (and (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

Algorithm

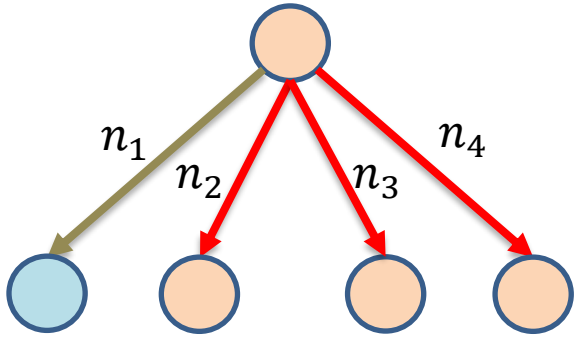


We keep both refinements in such cases.

Sometimes we are unable to prune because the models are functionally equivalent.

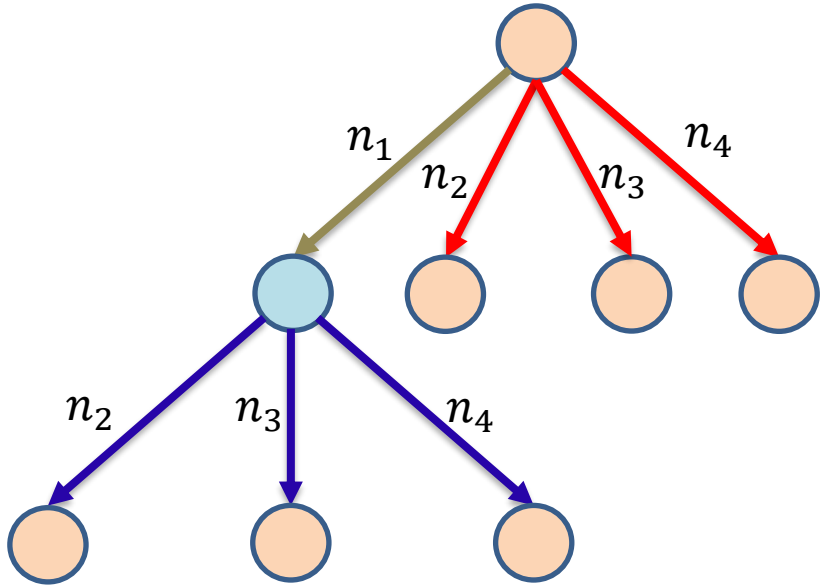
```
(:action pickup
  :parameters (?ob)
  :precondition (and (handempty)
                     (ontable ?ob))
  :effect (and (handempty)
               (not)(ontable ?ob)))
```

Algorithm



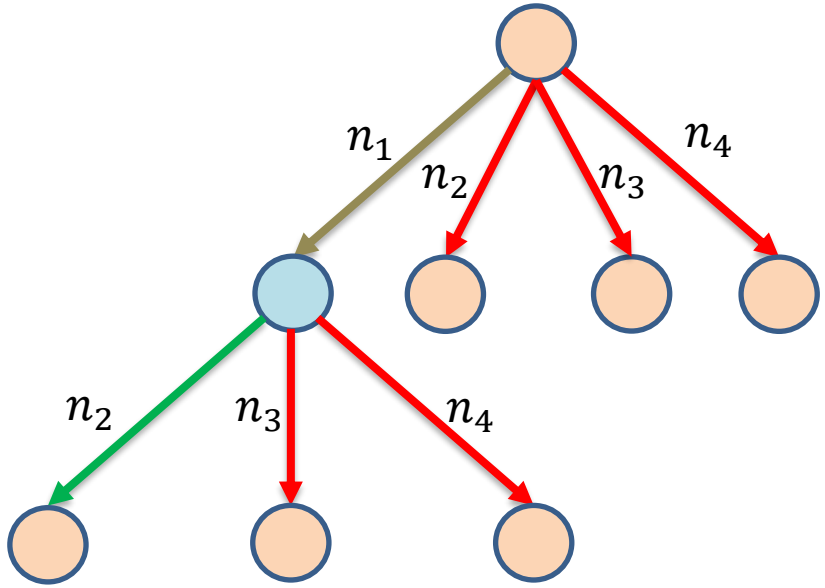

```
(:action pickup
  :parameters (?ob)
  :precondition (and (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

Algorithm




```
(:action pickup
  :parameters (?ob)
  :precondition (and (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

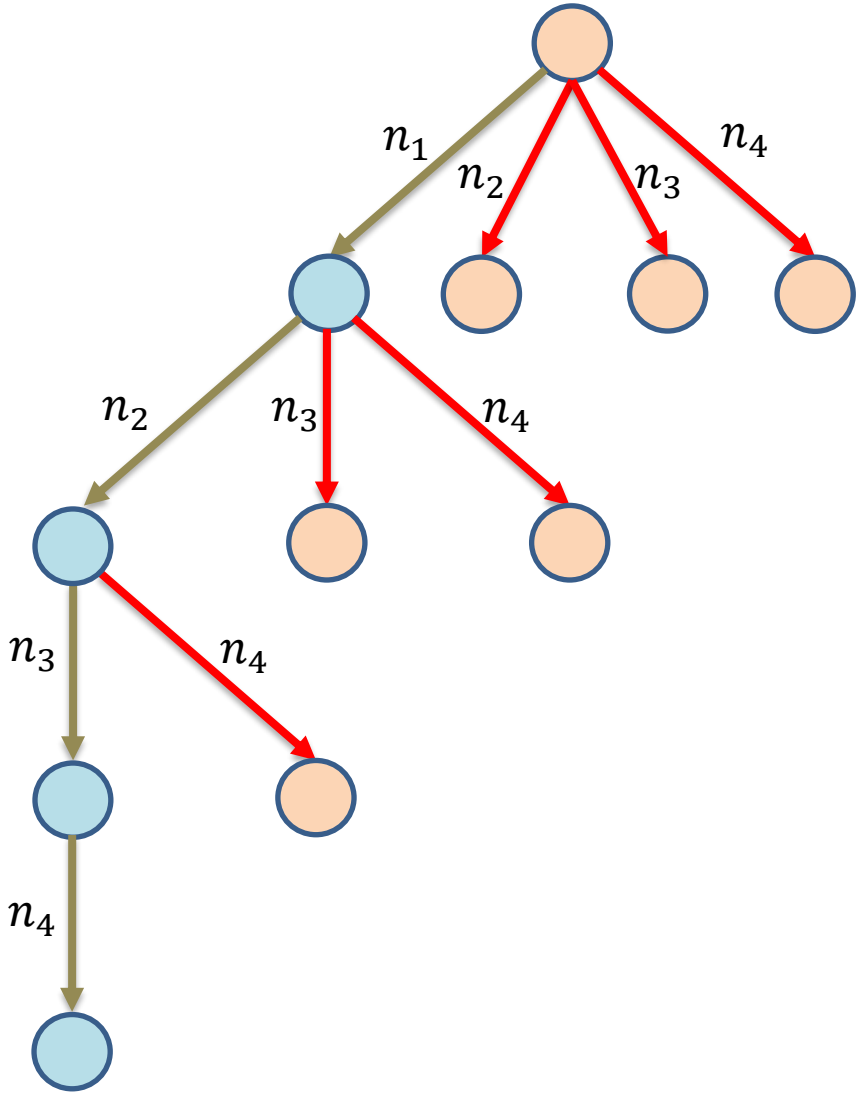
Algorithm




```
(:action pickup
  :parameters (?ob)
  :precondition (and (handempty)
                     (+/-/∅) (ontable ?ob))
  :effect (and (+/-/∅) (handempty)
               (+/-/∅) (ontable ?ob)))
```

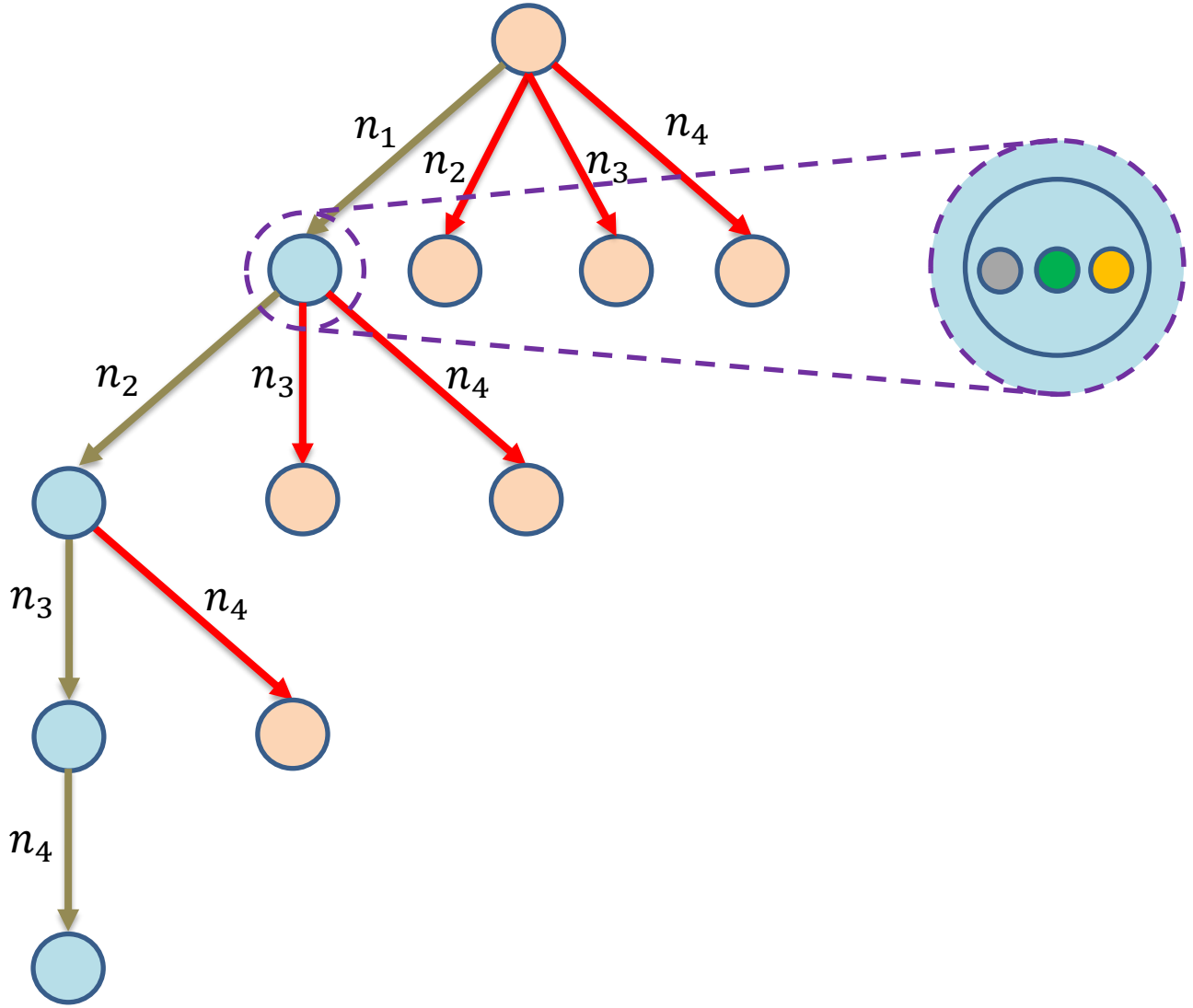
Repeat the whole process till we fix all the predicates in all the actions in their correct form.

Algorithm

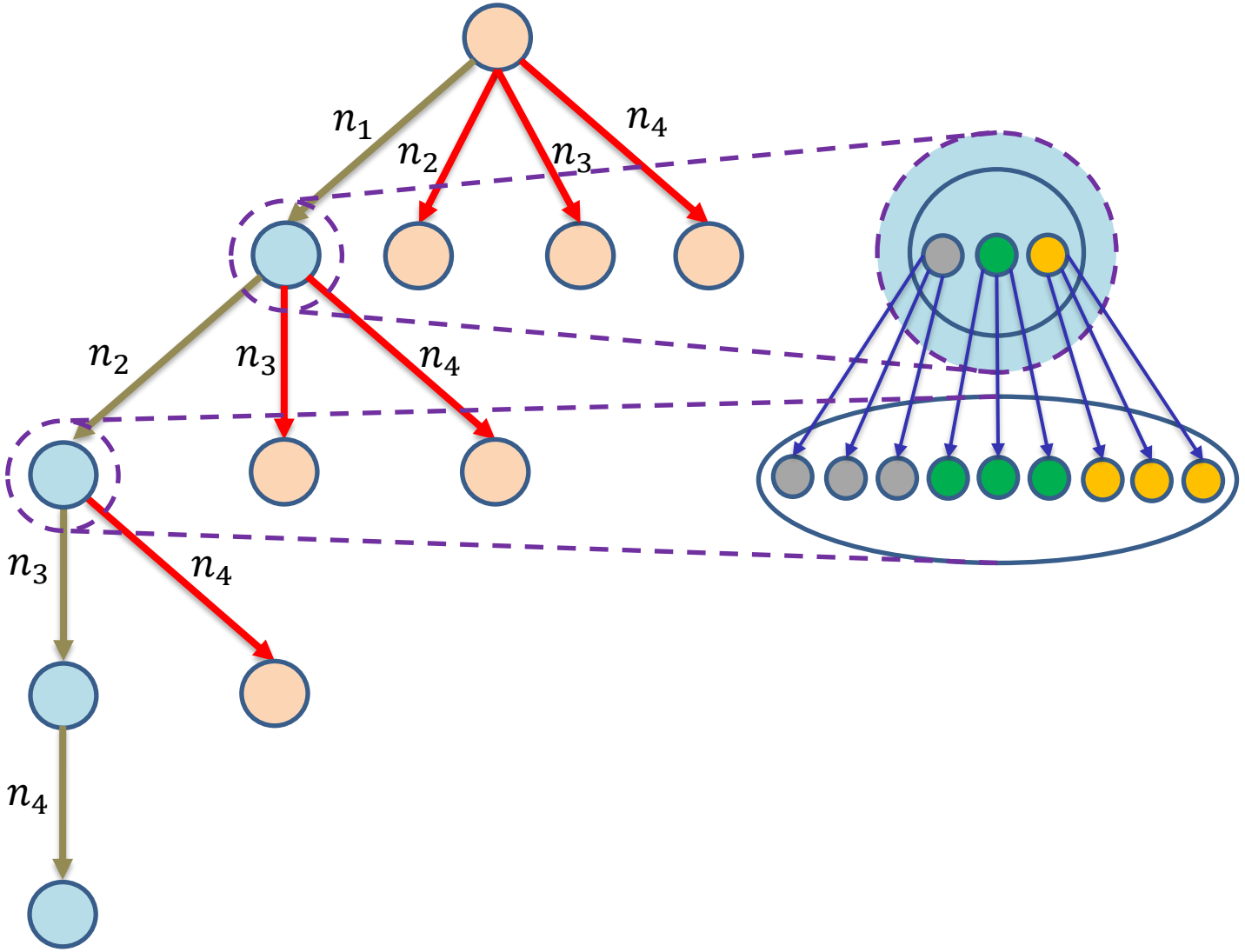



```
(:action pickup
  :parameters (?ob)
  :precondition (and (handempty)
                     (ontable ?ob))
  :effect (and (not)(handempty)
               (not)(ontable ?ob)))
```

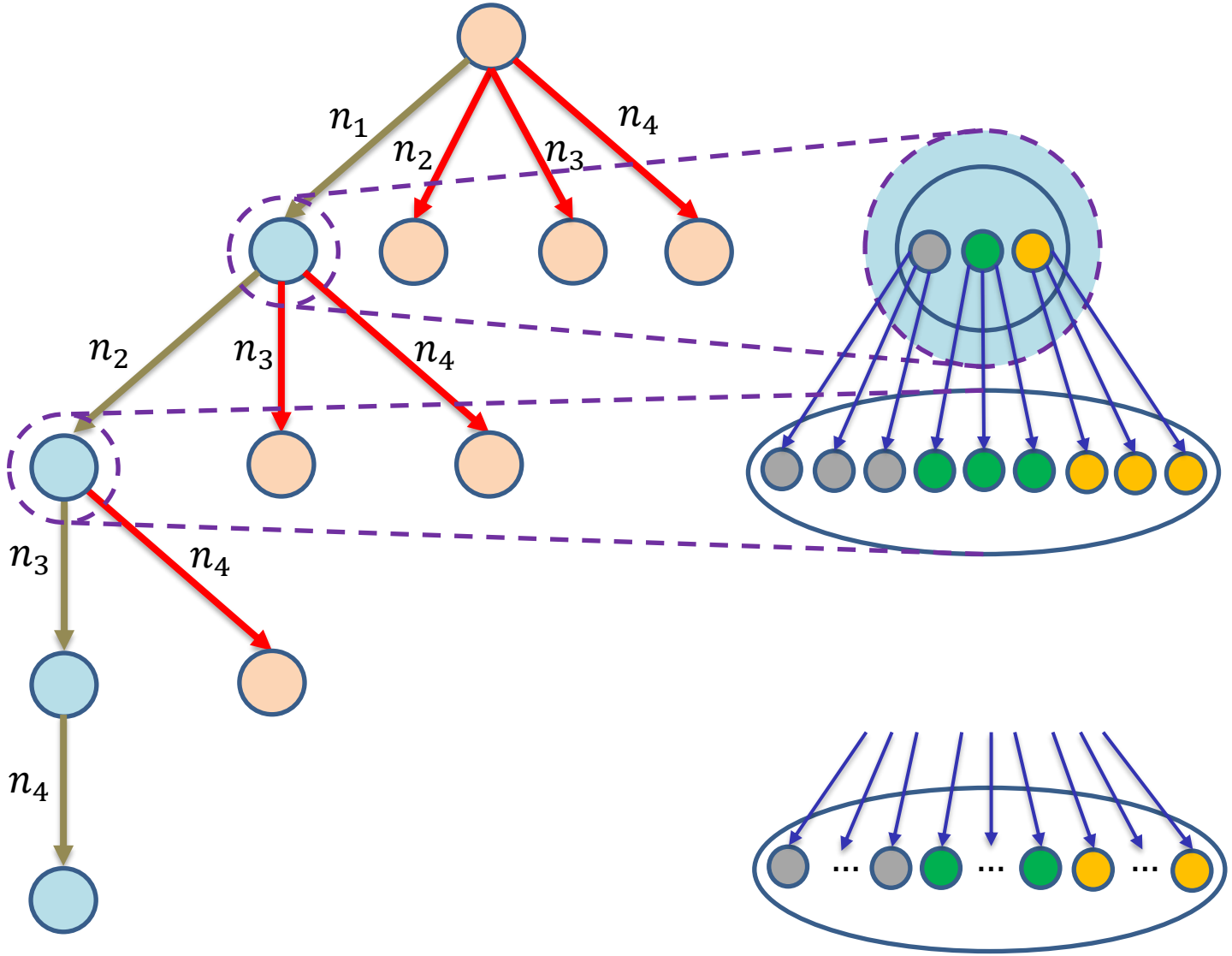
Algorithm



Algorithm



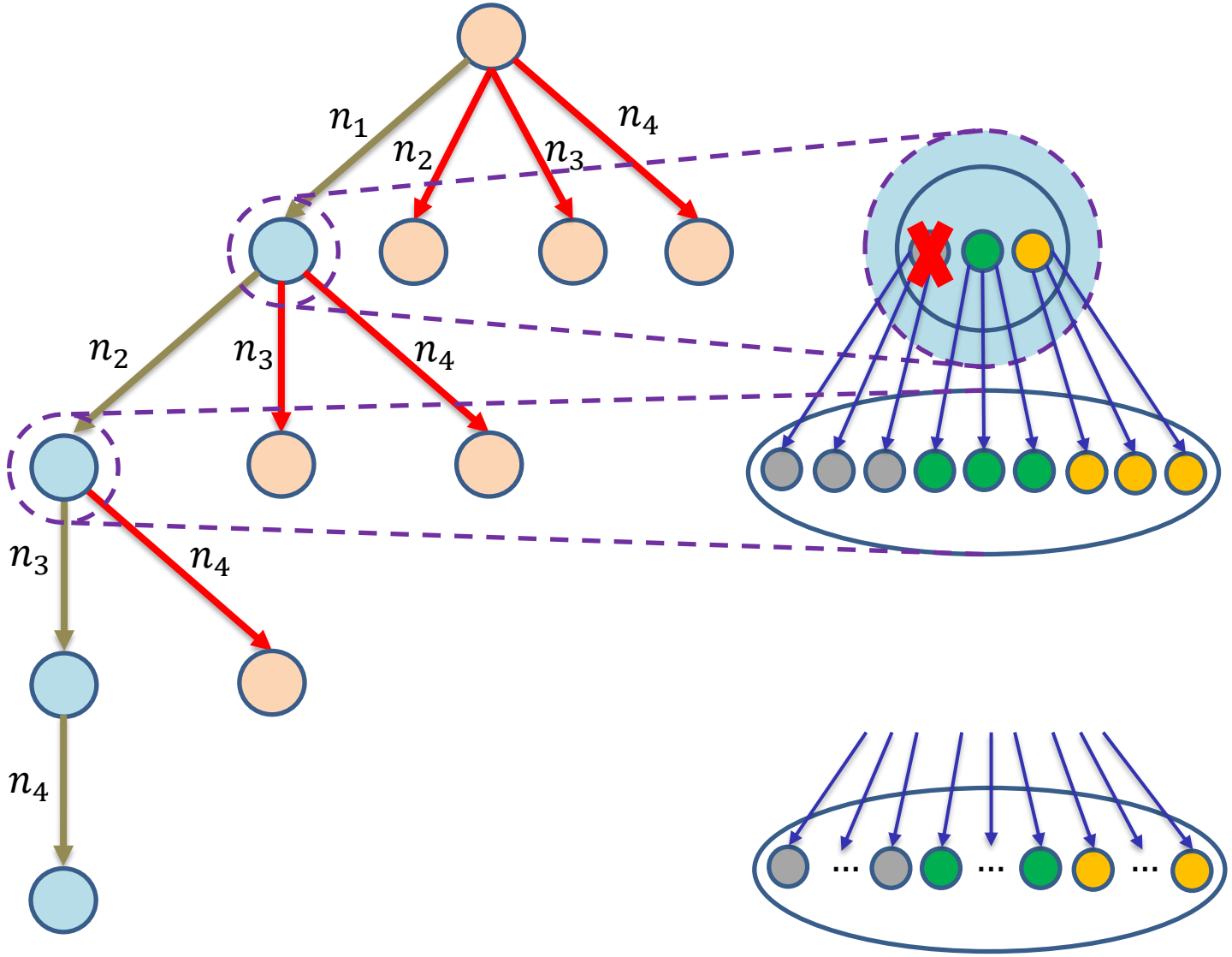
Algorithm



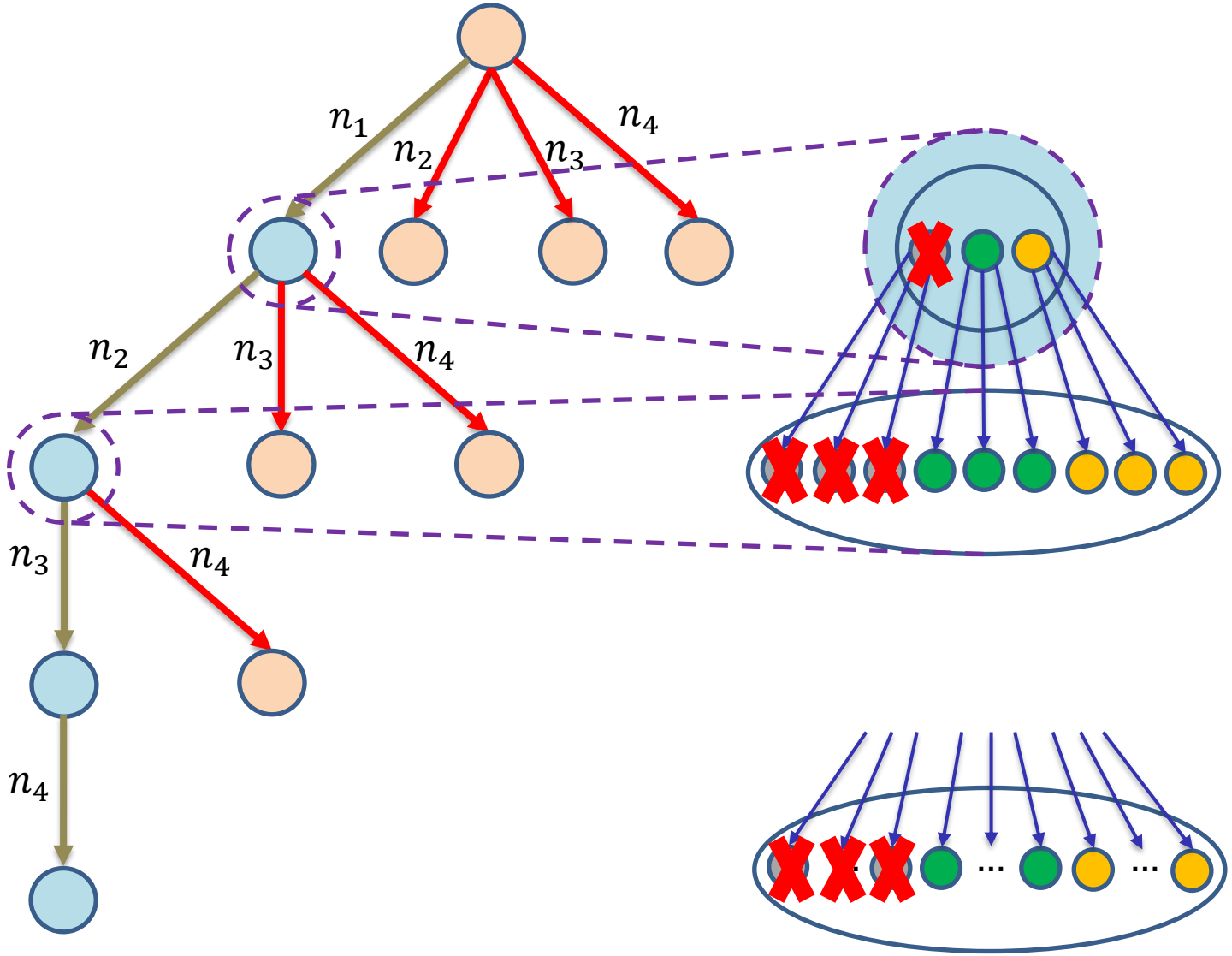
$9^{|\mathbb{P}|\times|\mathbb{A}|}$ because each predicate can appear in both precondition and effect in 3 possible ways: as a positive literal, as a negative literal, or not present.

$9^{|\mathbb{P}|\times|\mathbb{A}|}$ possible models at the most concrete node.

Algorithm



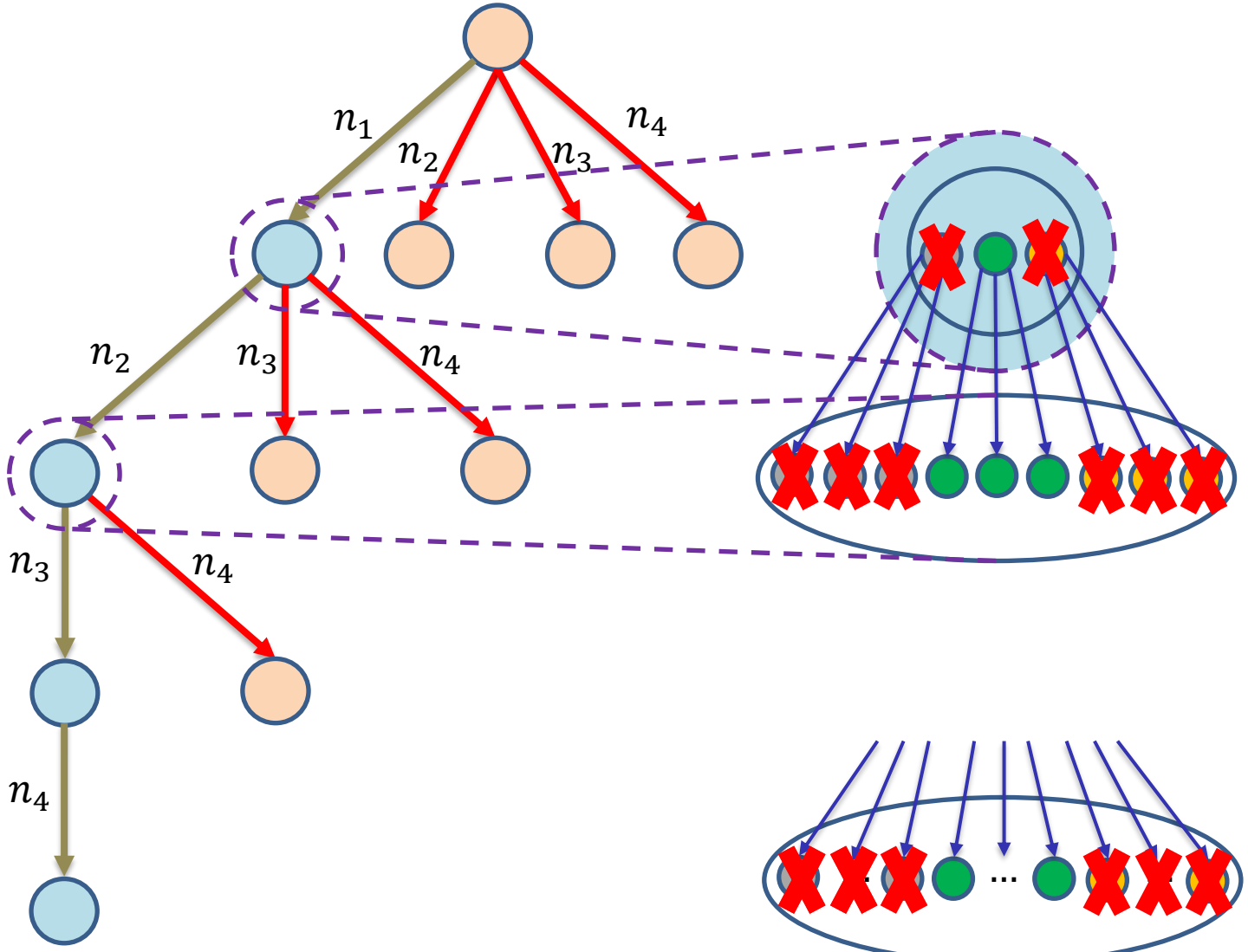
Algorithm



Key feature of the algorithm

Each time we prune an abstracted model, we prune a very large number of models at the most concrete node.

Algorithm



Key feature of the algorithm

Each time we prune an abstracted model, we prune a very large number of models at the most concrete node.

By the time we reach here, only 3 models remain in *most of the cases*.

Empirical Evaluation

Setup

- Randomly generate an agent and environment (domain+problem instance) from the IPC benchmark suite.
- Algorithm doesn't get this information.
- Evaluate the performance of agent interrogation in estimating that agent's relational STRIPS-like model.

Results

Number of queries generated in our approach vs naïve baseline

Domain	$\lvert\mathbb{P}\rvert$	$\lvert\mathbb{A}\rvert$	$\lvert\mathcal{Q}_{naive}\rvert$	$\lvert\mathcal{Q}\rvert$	Time/$\mathcal{Q}$ (sec)
gripper	5	3	15×2^5	37	0.14
blocks-world	9	4	36×2^9	92	1.73
elevator	10	4	40×2^{10}	109	5.91
logistics	11	6	66×2^{11}	98	11.62
parking	18	4	72×2^{18}	173	12.01
satellite	17	5	85×2^{17}	127	19.53
openstacks	10	12	120×2^{10}	203	11.28

Results are averages of 10 random runs

Conclusion

The proposed approach:

- Needs no prior knowledge of the agent model.
- Only requires an agent to have rudimentary query answering capabilities.
- Works for non-stationary environments.
  - Able to quickly obtain relevant, independent pieces of information needed to estimate new model.

Questions?