User-Driven Capability Assessment of Taskable AI Systems

bit.ly/aia25-tutorial

Pulkit Verma

MIT CSAIL

Siddharth Srivastava

AAIR
Autonomous Agents and Intelligent Robots
ASU Arizona State University

Schedule

08:30 AM	Meet and Greet over Coffee
09:00 AM	Session 1 • Introduction and Motivation • Assessment through Model Learning • Assessment of Black-Box AI Systems in Stationary Settings
10:30 AM	Coffee Break
11:00 AM	Session 2 • Discovering Capabilities for Black-Box AI Assessment • AI Assessment in Adaptive Settings • Future Directions and Conclusion
12:30 PM	Lunch

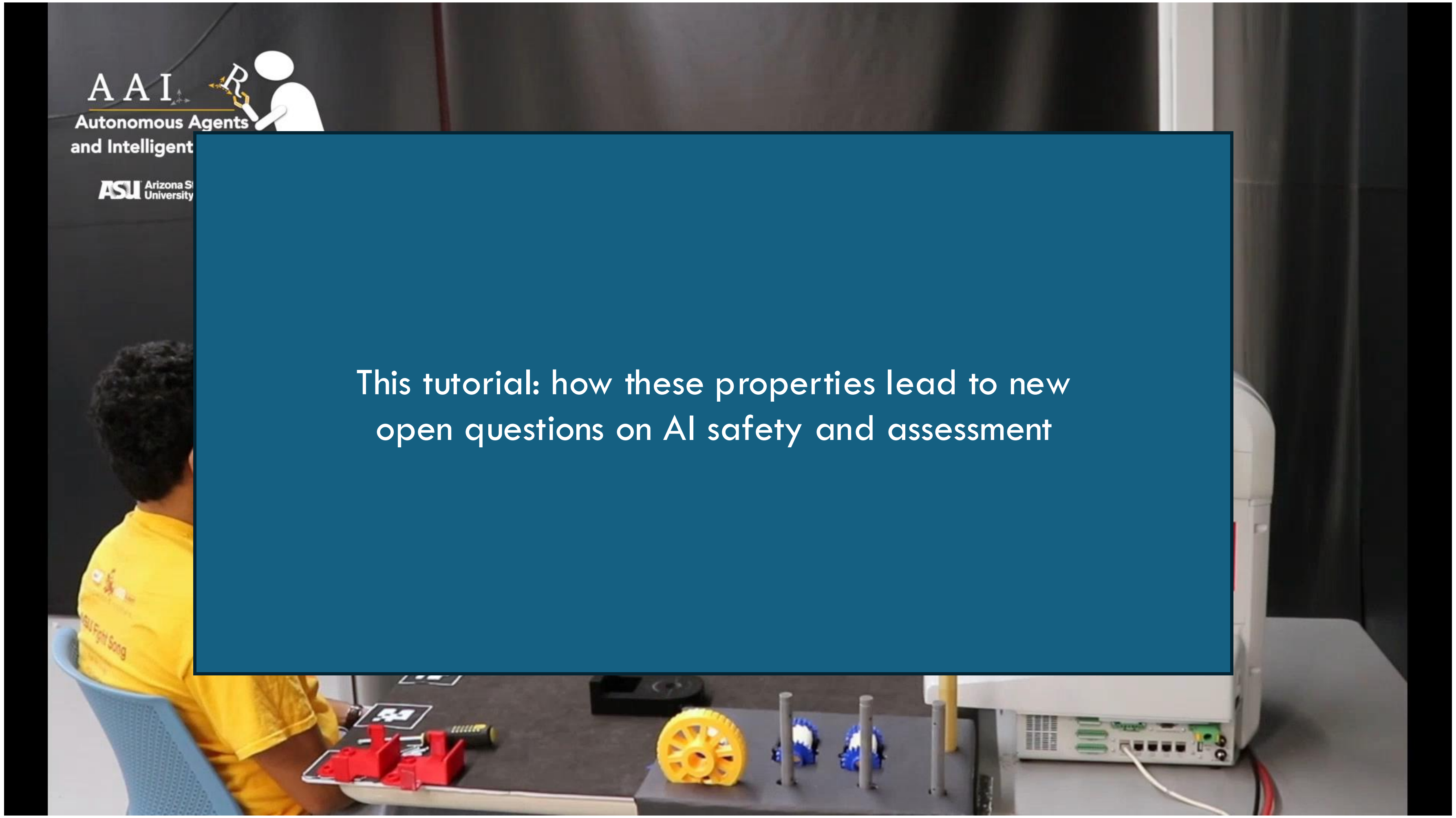
Taskable AI Systems

Expected to improve, adapt, learn, and achieve user-desired task

Video Link: bit.ly/taskable-ai

Taskable AI Systems

Expected to improve, adapt, learn, and achieve user-desired task



This tutorial: how these properties lead to new open questions on AI safety and assessment

Classical Notion of Verification

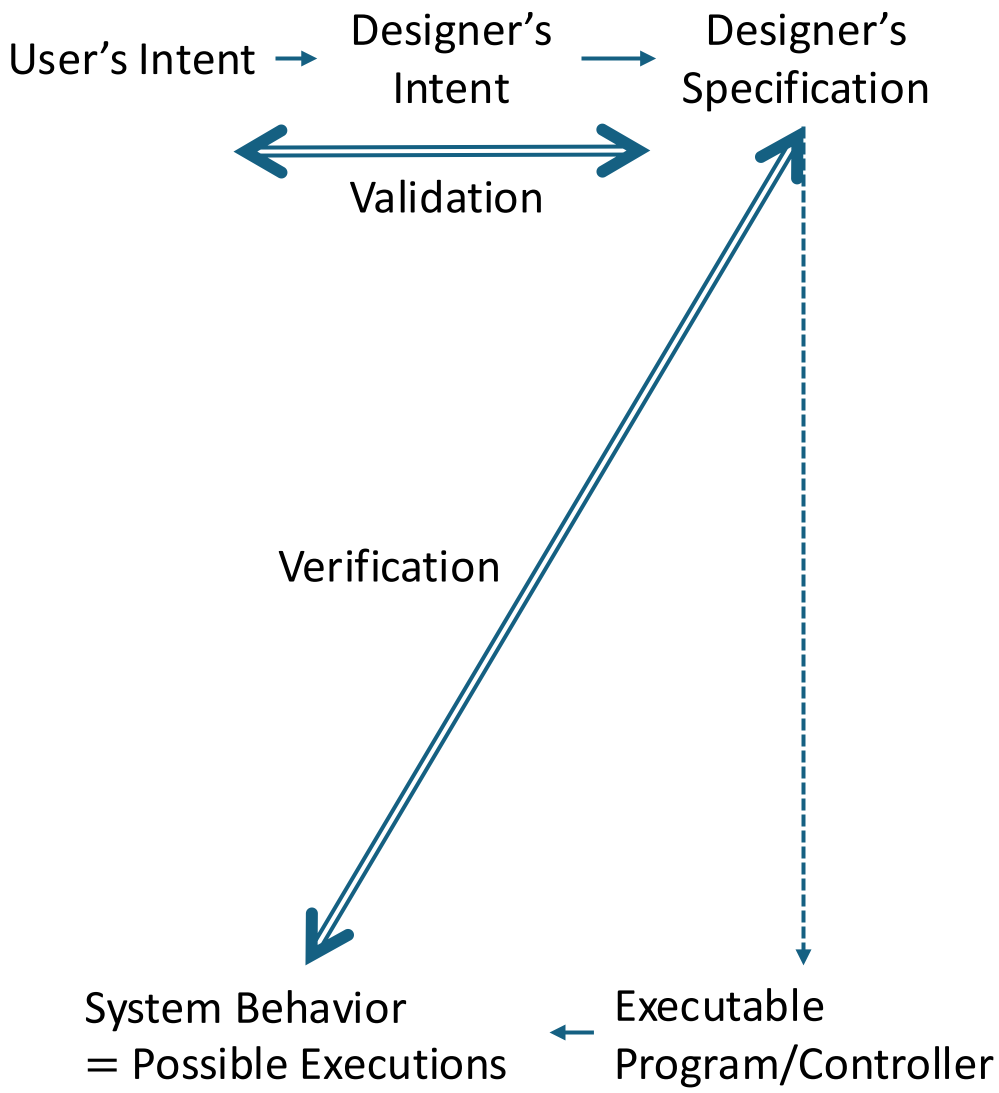
User's Intent → Designer's Intent → Designer's Specification

Executable Program/Controller → System Behavior = Possible Executions

Classical Notion of Verification

Input for Verification: Executable Program/Controller
(includes task spec)
+ Model/assumptions on env
+ Safety property

The designer plays a central role



Conventional Approach to Verification: Example

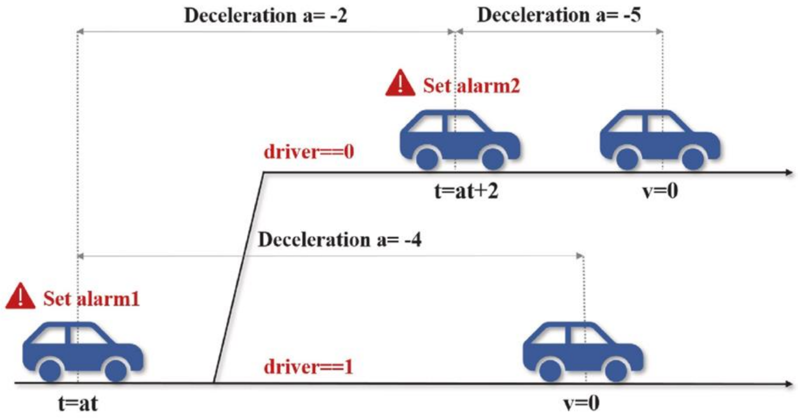
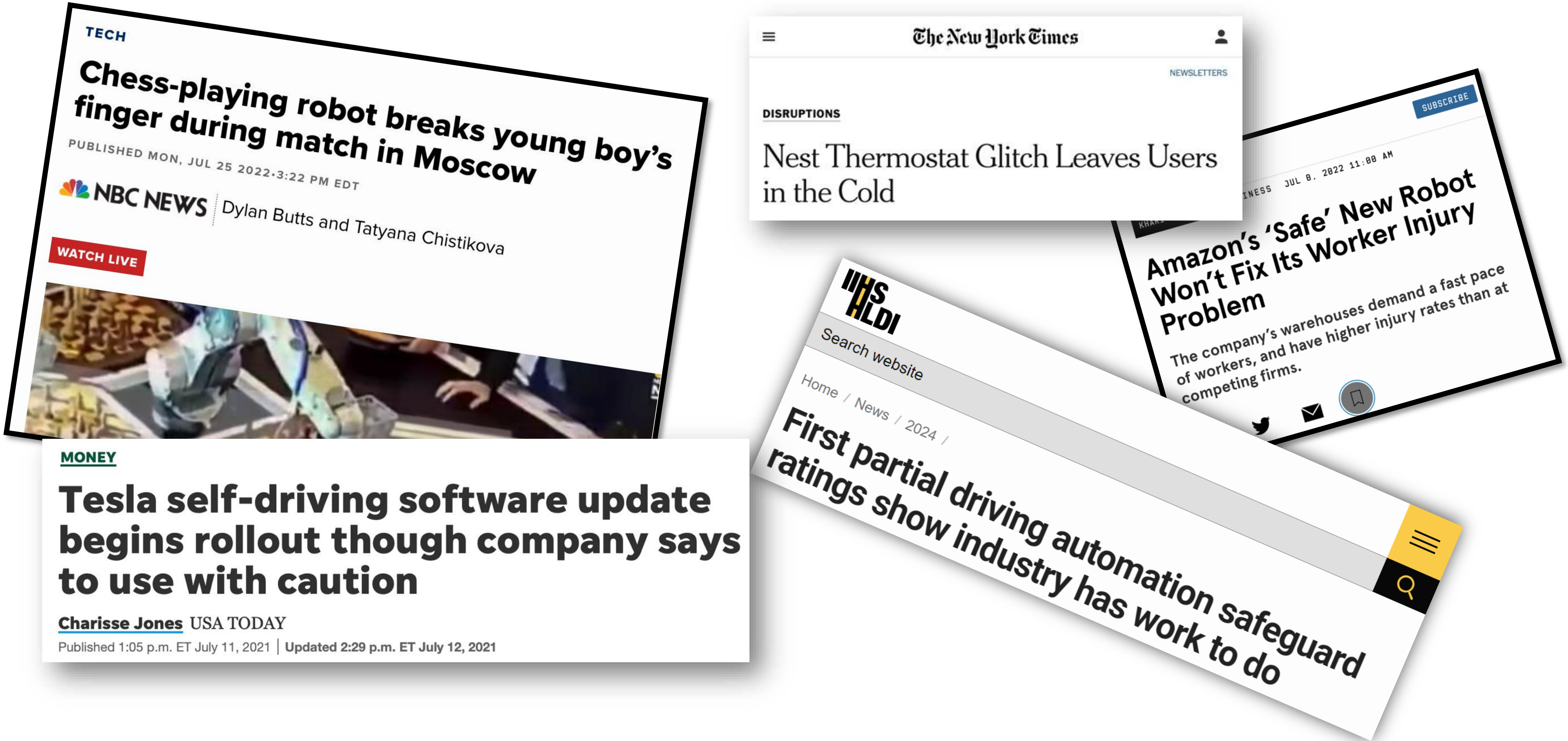


Fig. 8. Illustration of AEB.

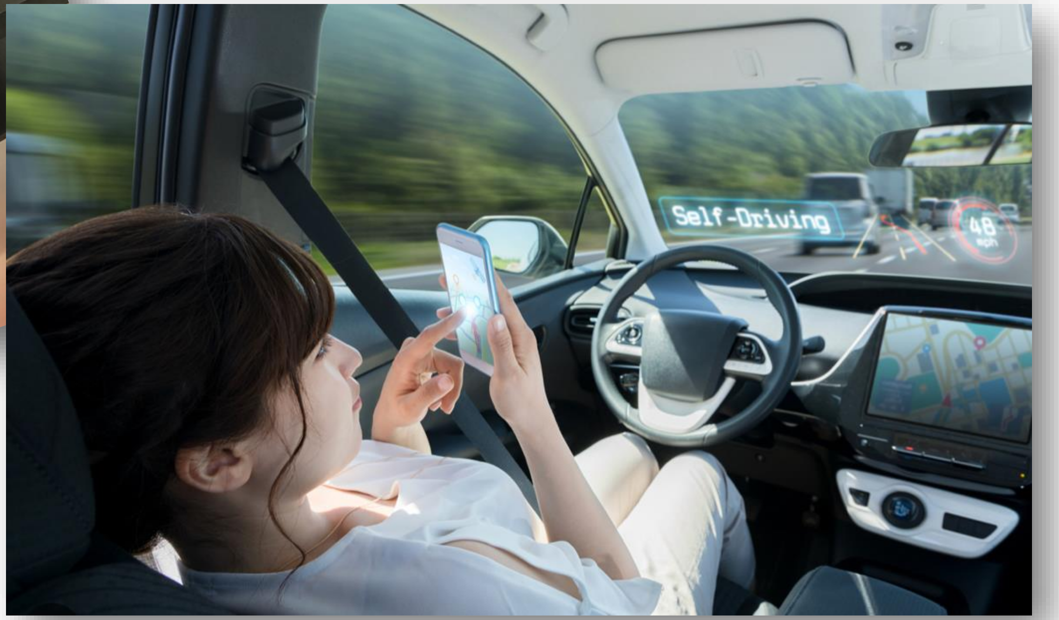
- **Property 1** is *Termination*; when the car reaches the $term$ location, its velocity must be 0. We set the forbidden states as $loc(Car1) == term \,\&\, v > 0$.
- **Property 2** is *VelocityLimit*; the velocity must always be in the range 0 to 20. The forbidden states of this property are defined as $v < 0 \,|\, v > 20$.
- **Property 3** is *Evolve*; we define this property to show the evolution of velocity.

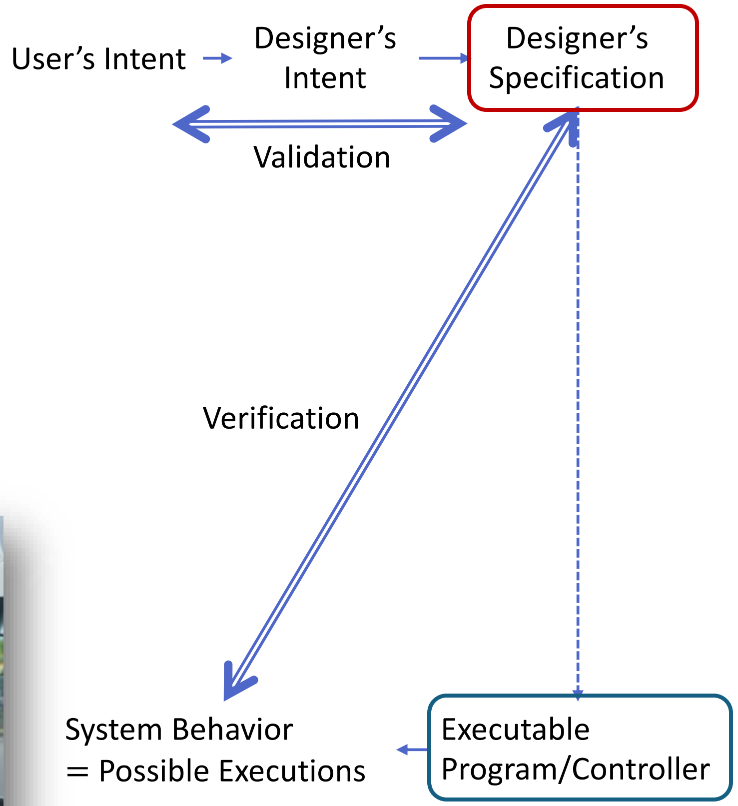


User-Aligned AI Assessment is a Different Problem:
How would a user know what their current AI system can do safely?

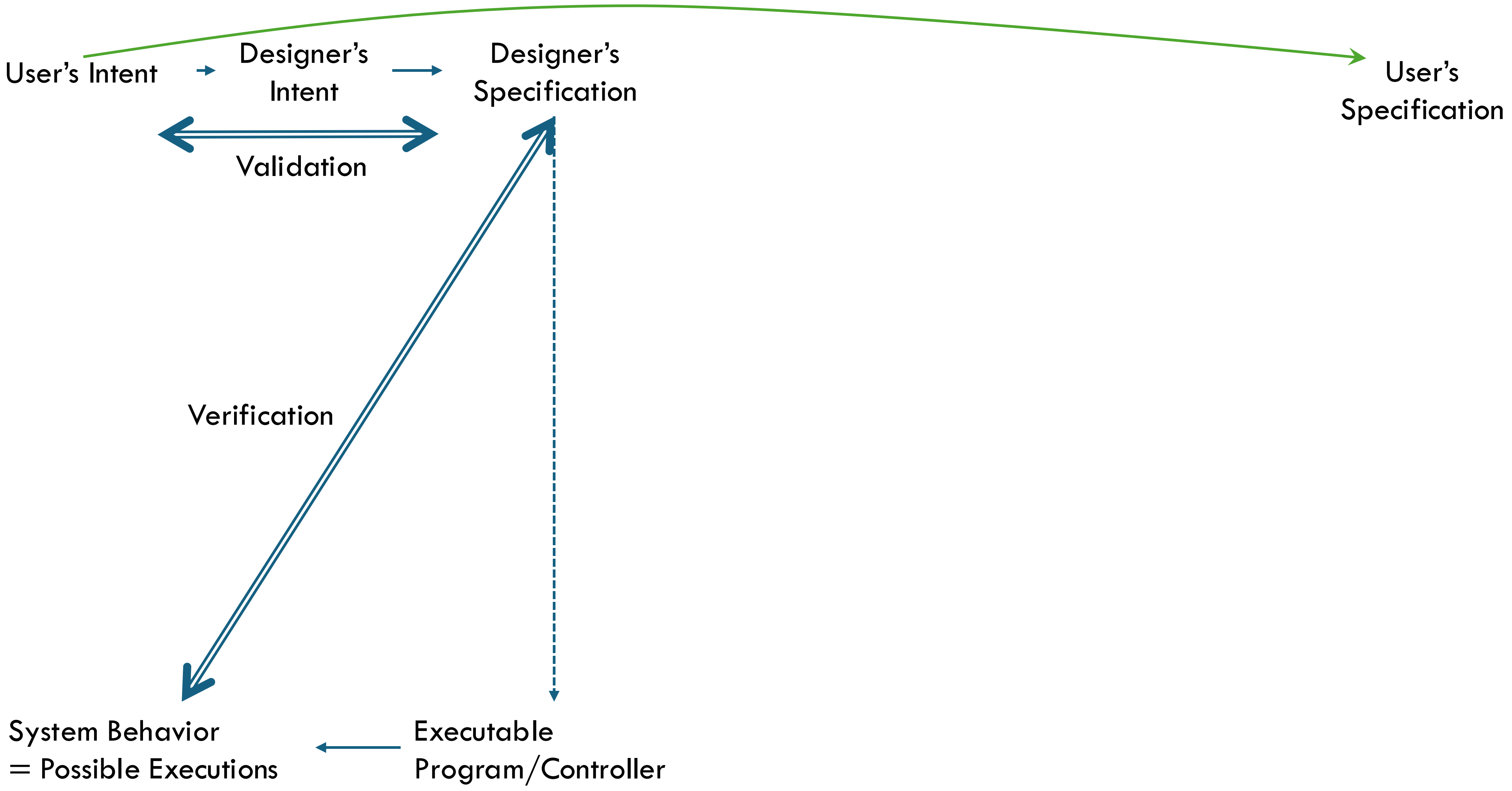
Taskable AI Systems

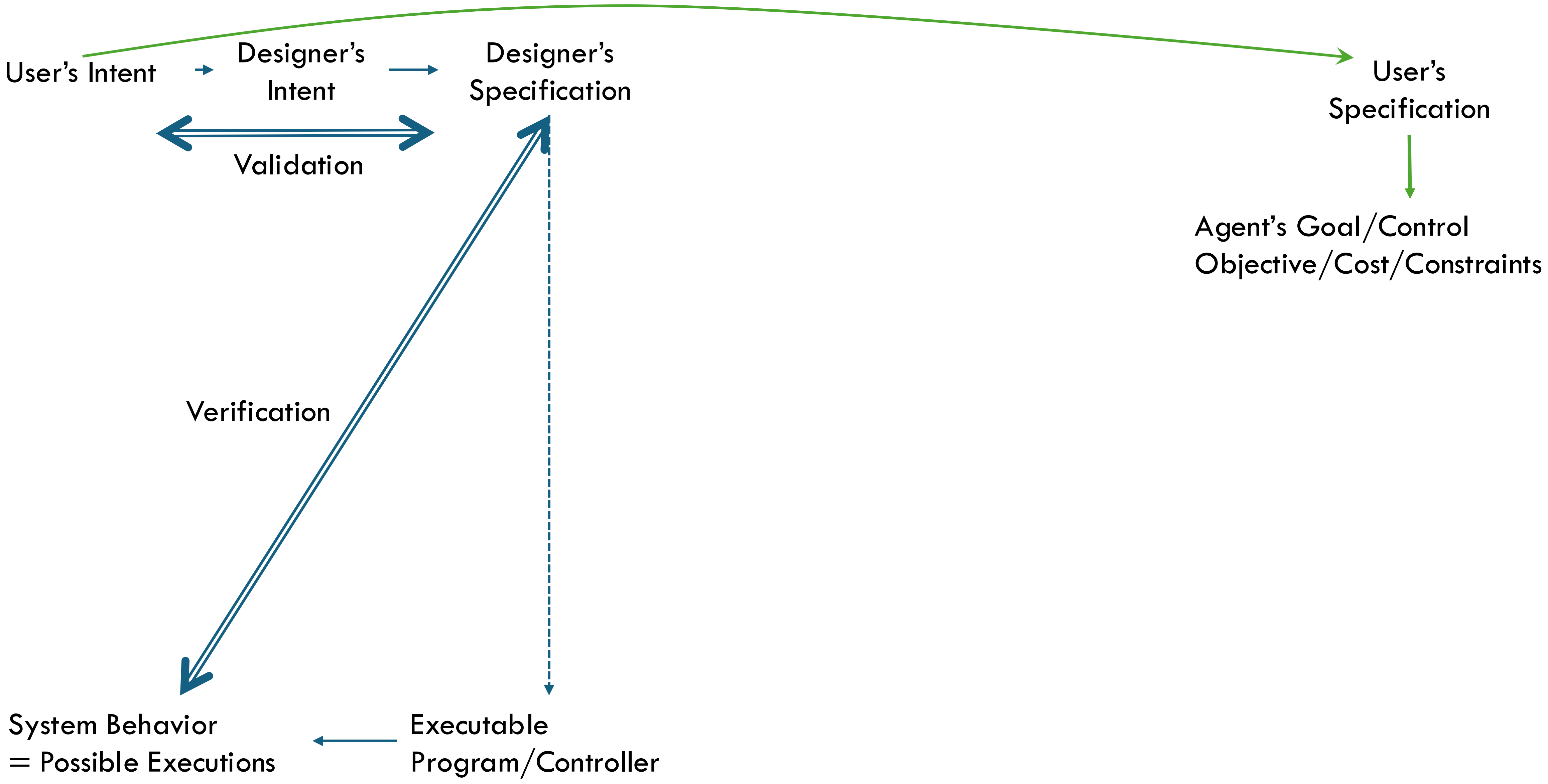
- What is the design spec?
- What is the program/controller?
- What should the safety property be?
- What should the user do when the system's behavior changes?

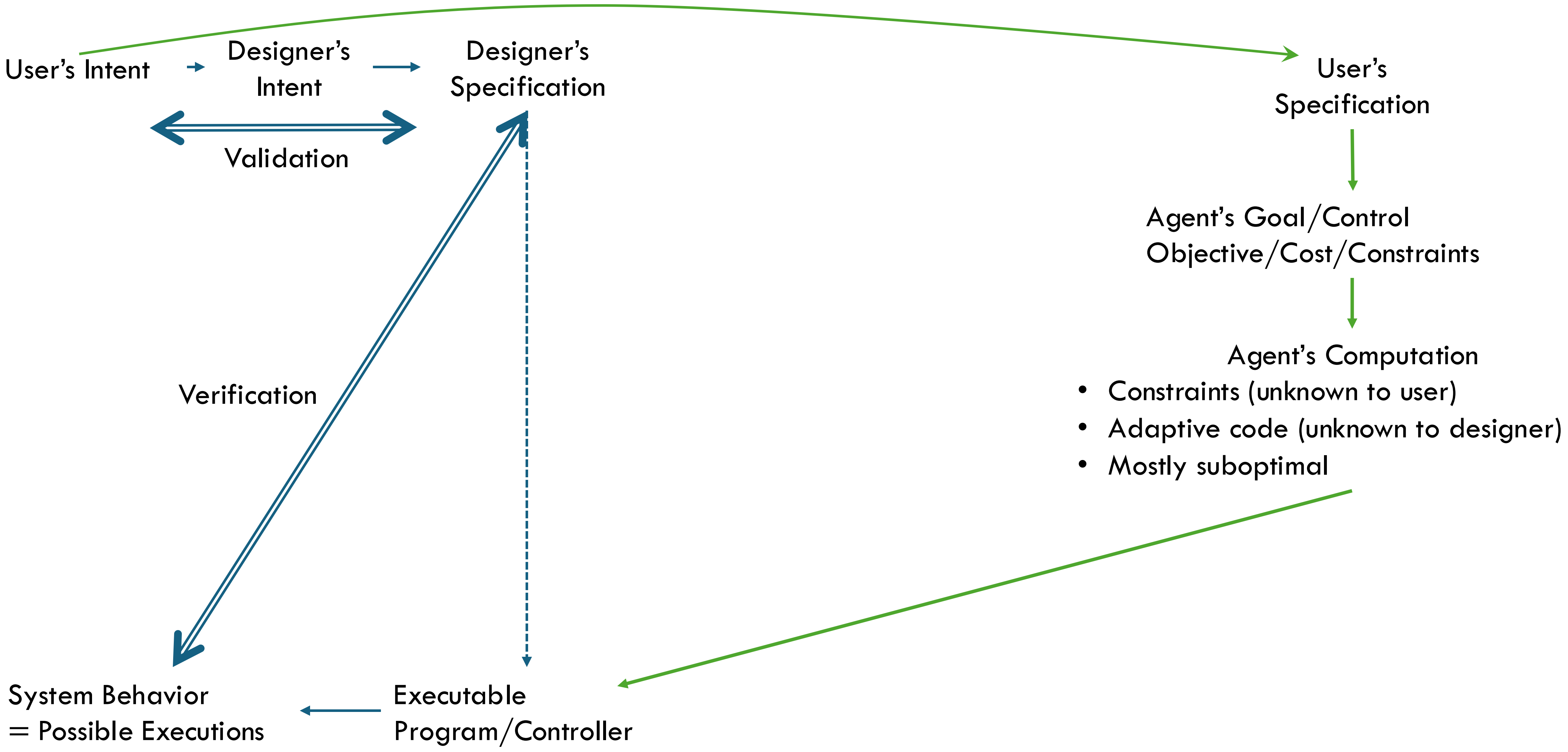


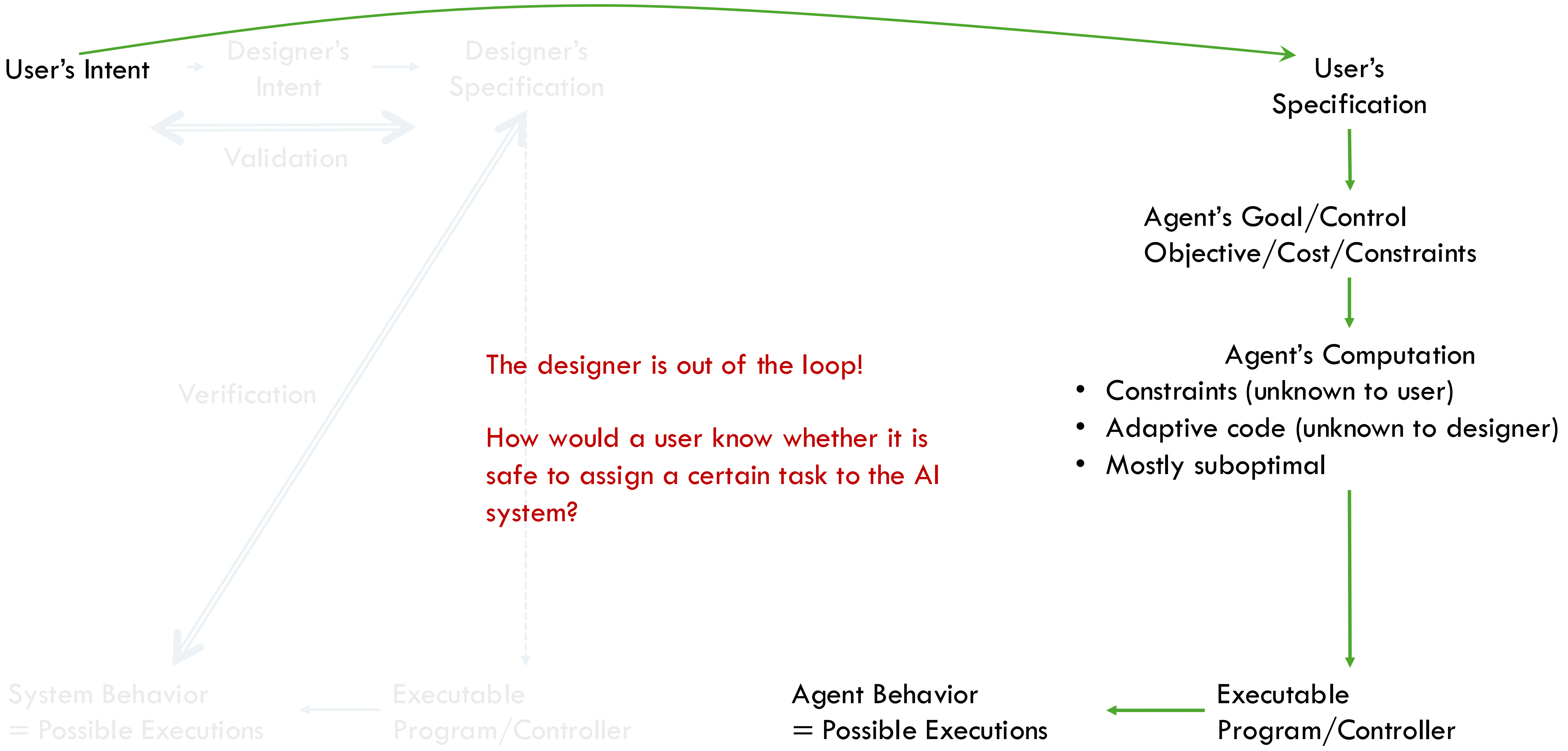


The AI Assessment Problem

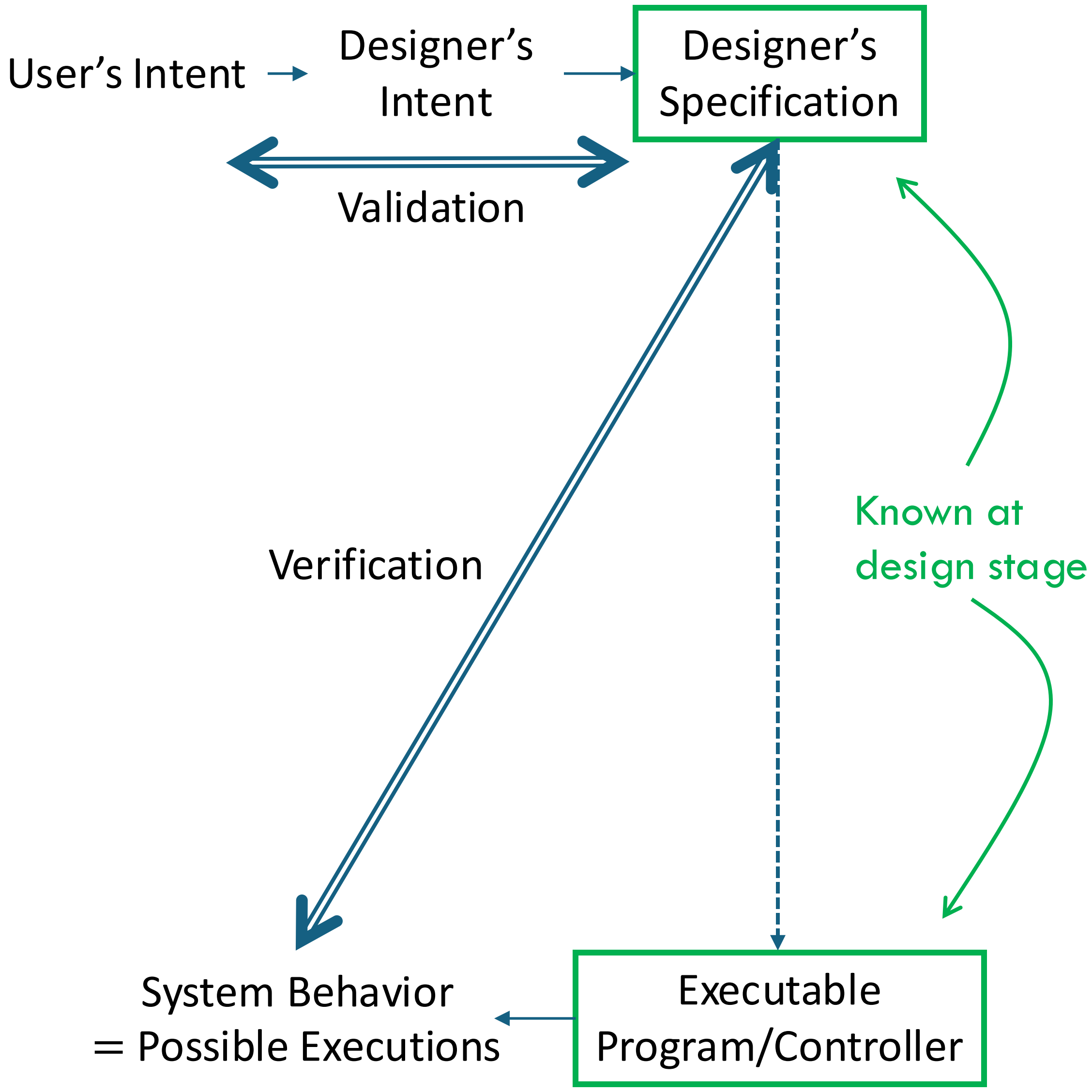

User's Intent
Designer's Intent
Designer's Specification
User's Specification
Validation
Verification
System Behavior = Possible Executions
Executable Program/Controller


User’s Intent
Designer’s Intent
Designer’s Specification
User’s Specification
Validation
Agent’s Goal/Control Objective/Cost/Constraints
Verification
System Behavior = Possible Executions
Executable Program/Controller

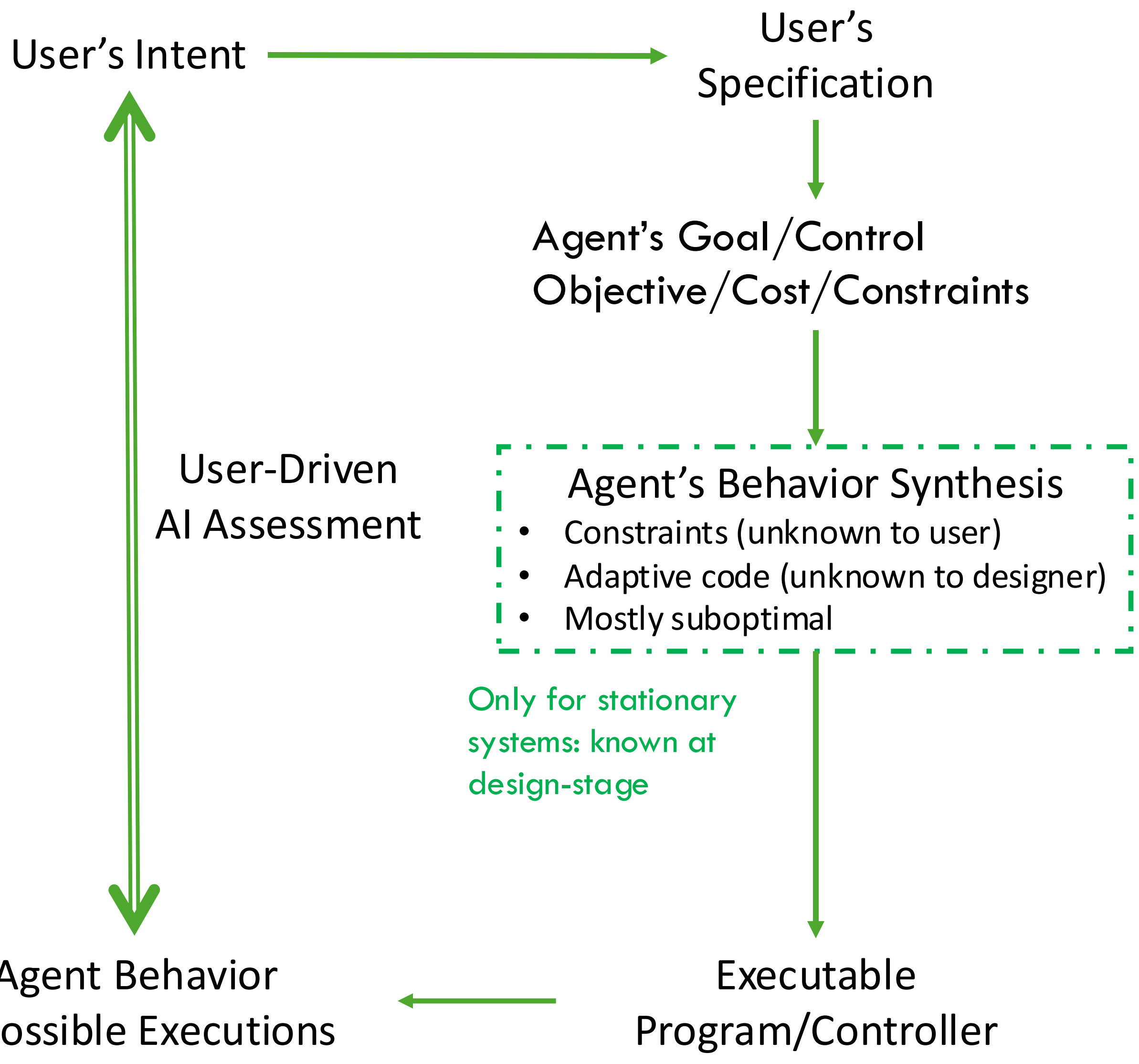

User's Intent
Designer's Intent
Designer's Specification
Validation
Verification
System Behavior = Possible Executions
Executable Program/Controller
User's Specification
Agent's Goal/Control Objective/Cost/Constraints
Agent's Computation
• Constraints (unknown to user)
• Adaptive code (unknown to designer)
• Mostly suboptimal


User's Intent
Designer's Intent
Designer's Specification
Validation
Verification
System Behavior = Possible Executions
Executable Program/Controller
User's Specification
Agent's Goal/Control Objective/Cost/Constraints
Agent's Computation
• Constraints (unknown to user)
• Adaptive code (unknown to designer)
• Mostly suboptimal
Executable Program/Controller
Agent Behavior = Possible Executions
The designer is out of the loop!
How would a user know whether it is safe to assign a certain task to the AI system?

Conventional Verification



Needed for AI Systems

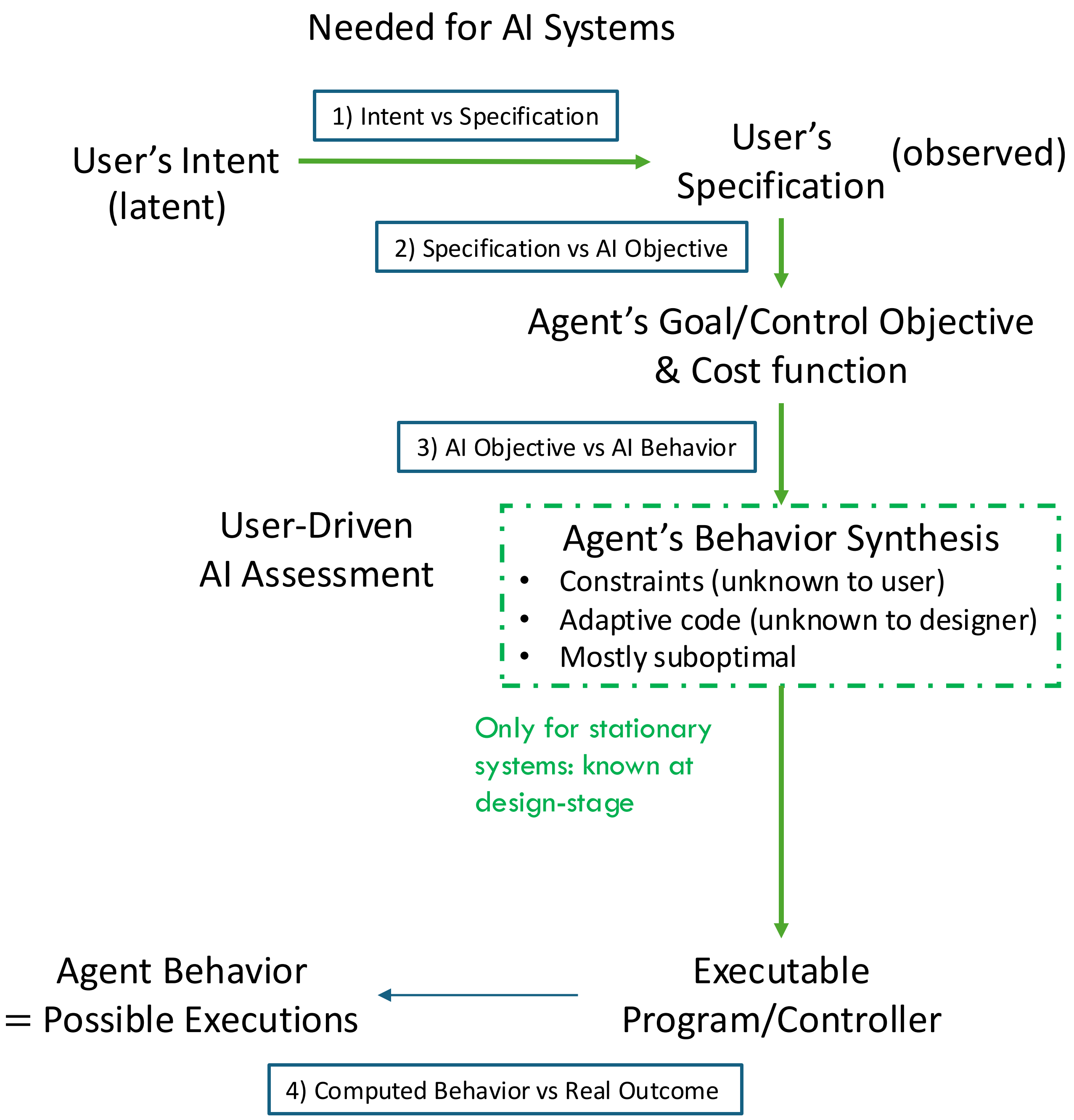


1. Translating a user's implicit intent to their explicit specification

2. Translating a user's specification to a formal representation of a goal or a utility function for the agent

3. Computing agent's behavior given a goal/utility function

4. The real results of executing the computed control/behavior



How do AI Safety Issues Fit in?

Reward Hacking

The agent optimizes reward but exploits flaws in the reward specification

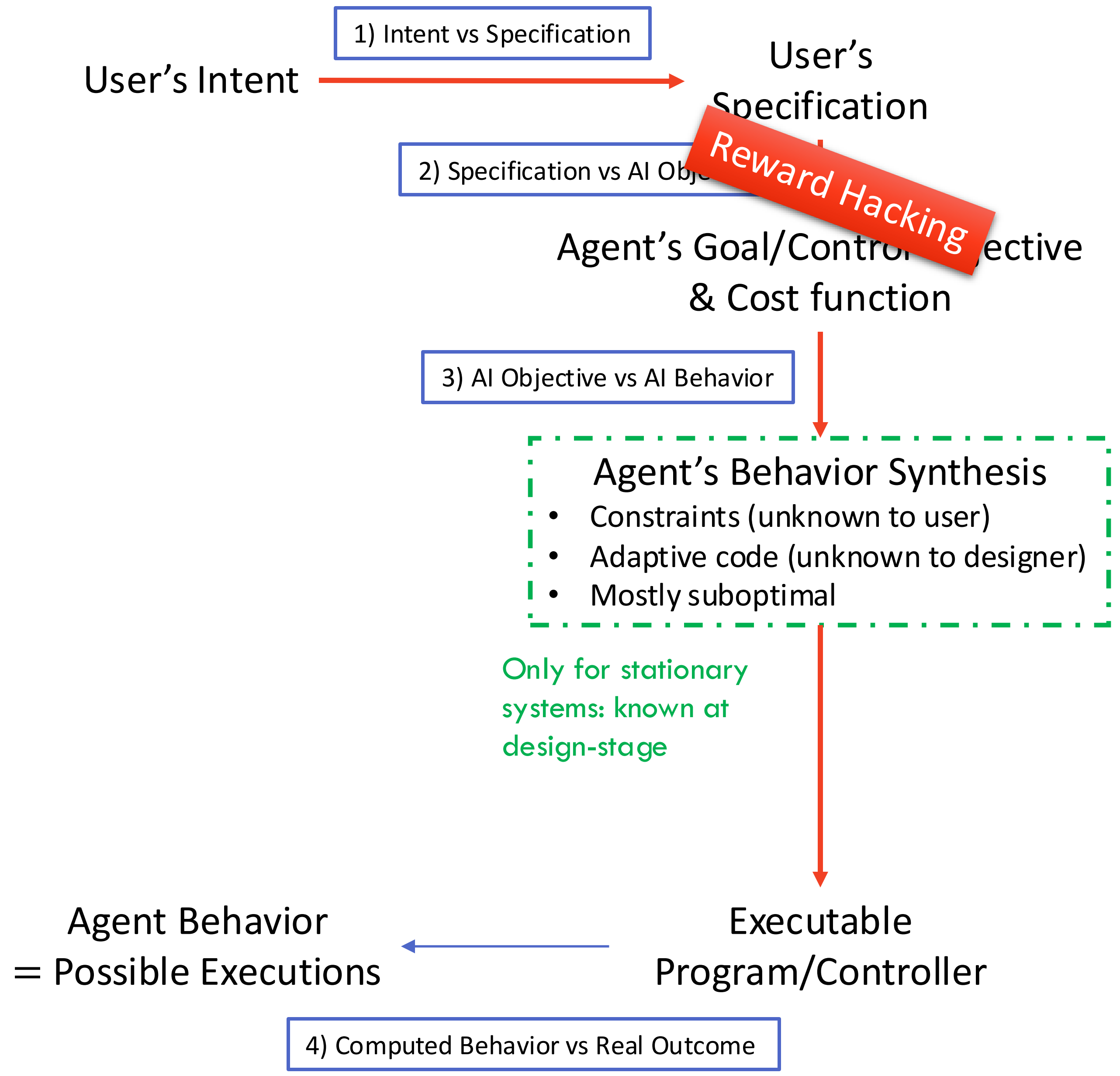
Wireheading

Reward Misspecification

Side Effects

Off-Switch

Needed for AI Systems



How do AI Safety Issues Fit in?

Reward Hacking

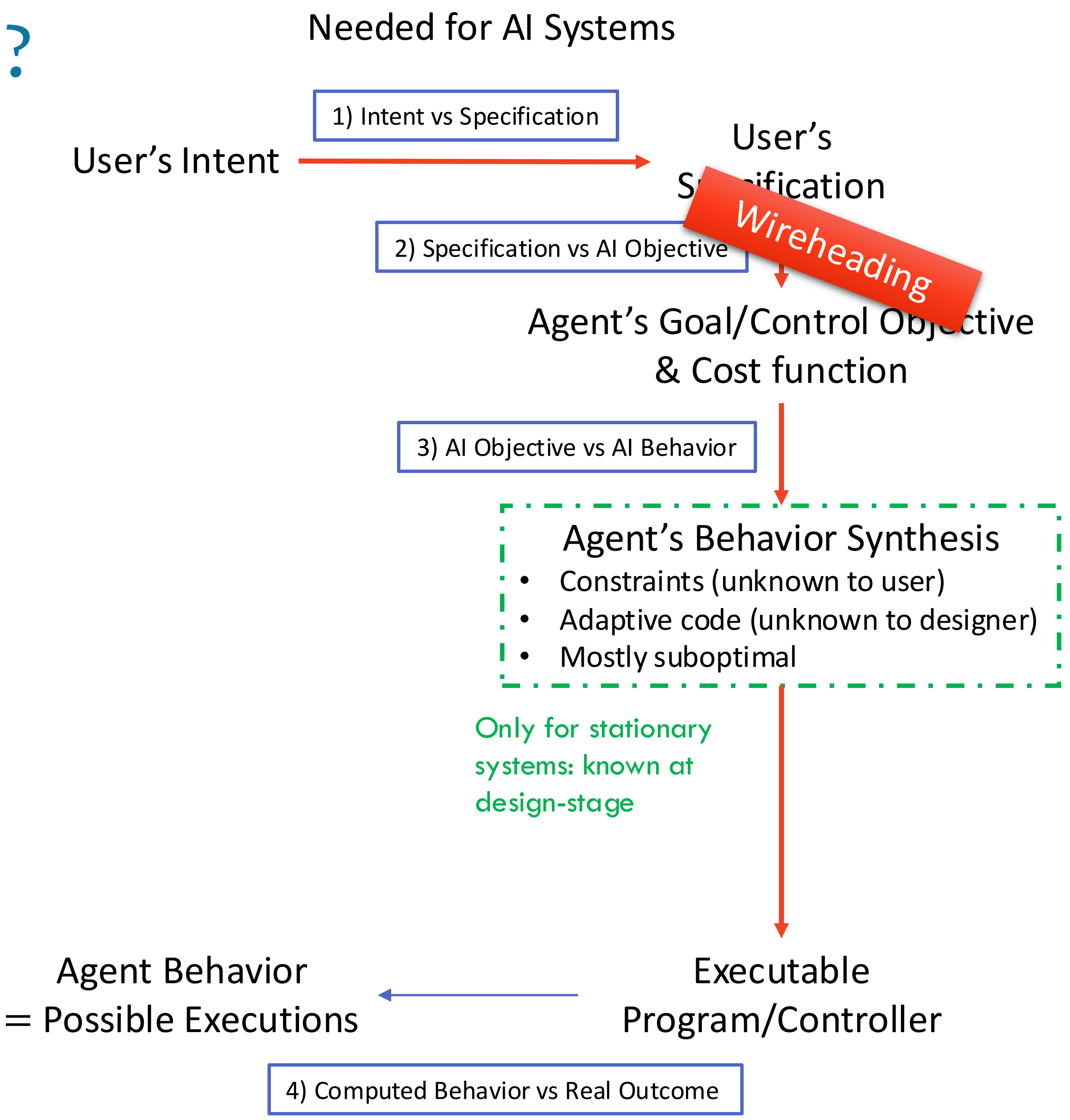
Wireheading

Agent manipulates its reward function. E.g., convince user; add noise to reward signal

Reward Misspecification

Side Effects

Off-Switch



How do AI Safety Issues Fit in?

Reward Hacking

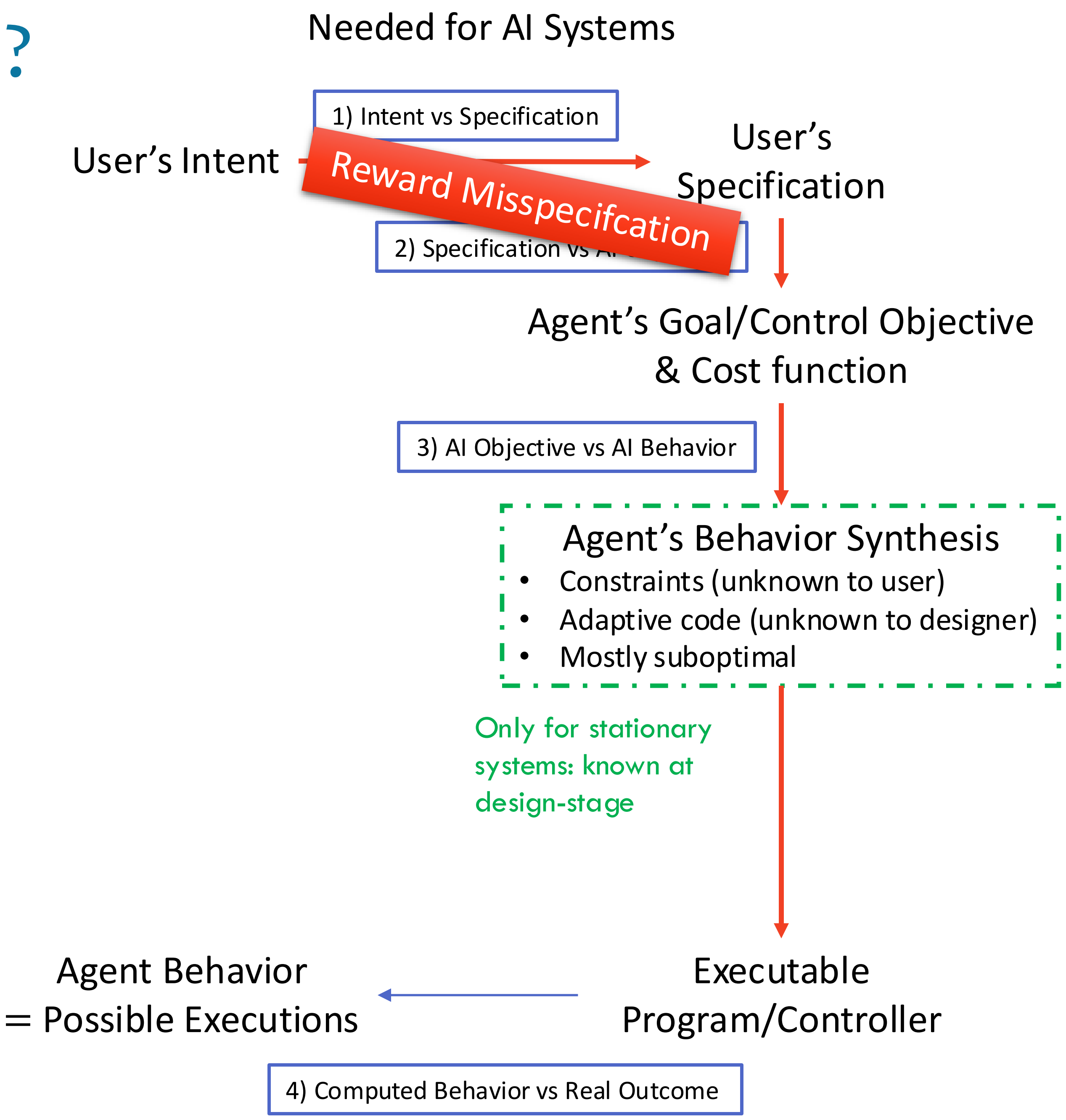
Wireheading

Reward Misspecification

User rewards observations, beliefs, or correlated features

Side Effects

Off-Switch



How do AI Safety Issues Fit in?

Reward Hacking

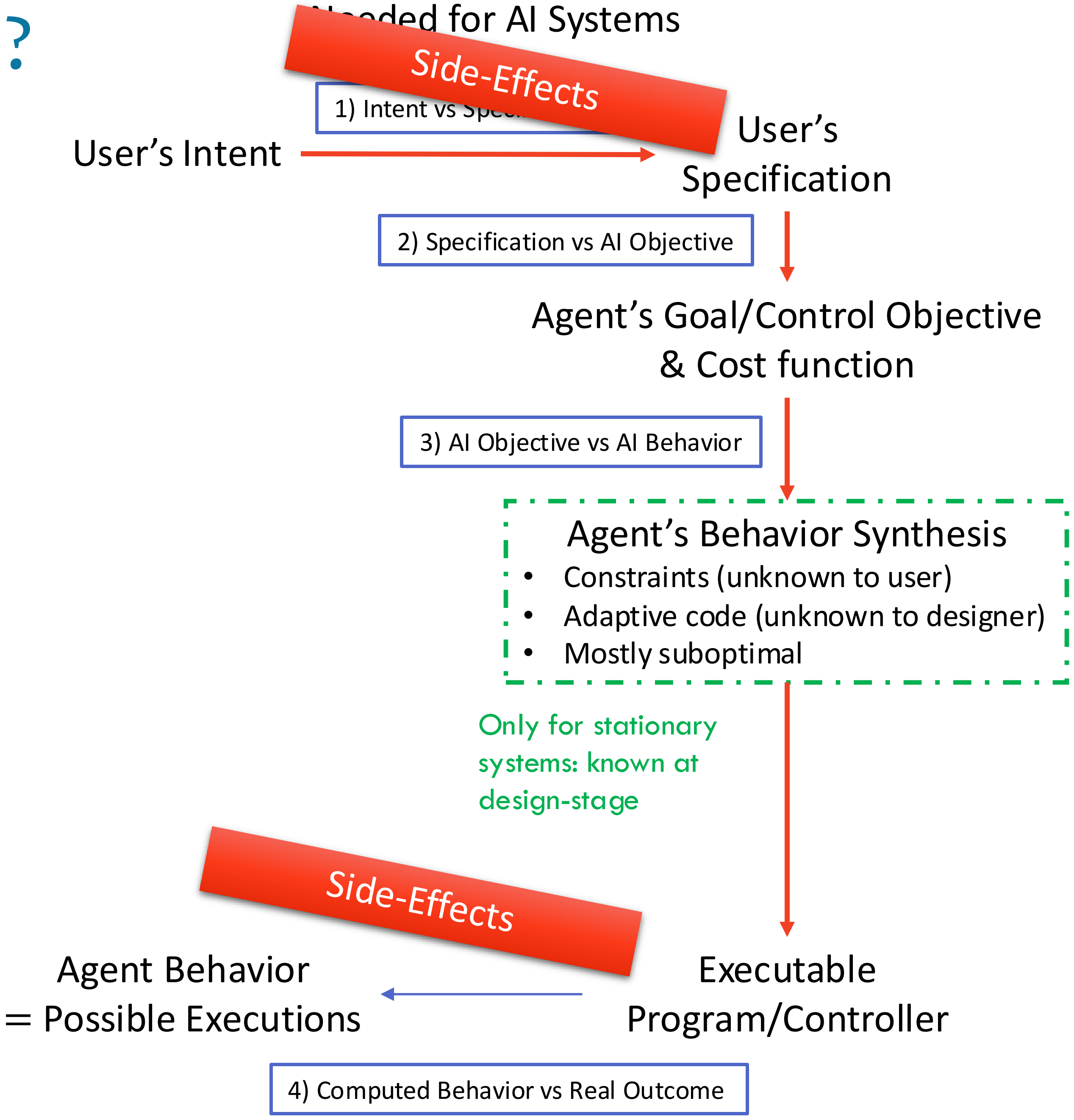
Wireheading

Reward Misspecification

Side Effects

Agent achieves objective, but with unexpected problems

Off-Switch



How do AI Safety Issues Fit in?

Reward Hacking

Wireheading

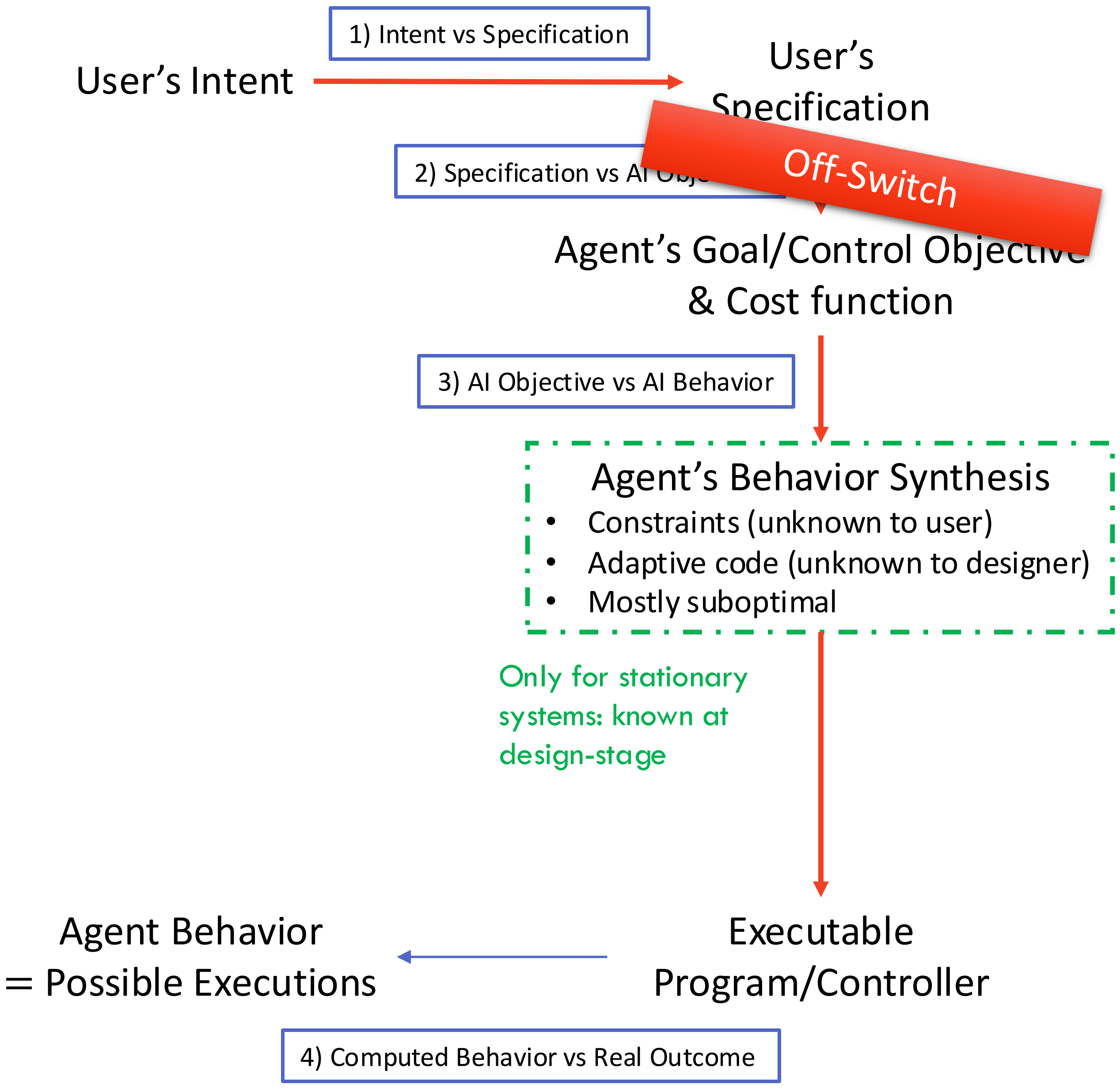
Reward Misspecification

Side Effects

Off-Switch

Agent doesn’t let the user turn it off

Needed for AI Systems



How do AI Safety Issues Fit in?

Reward Hacking

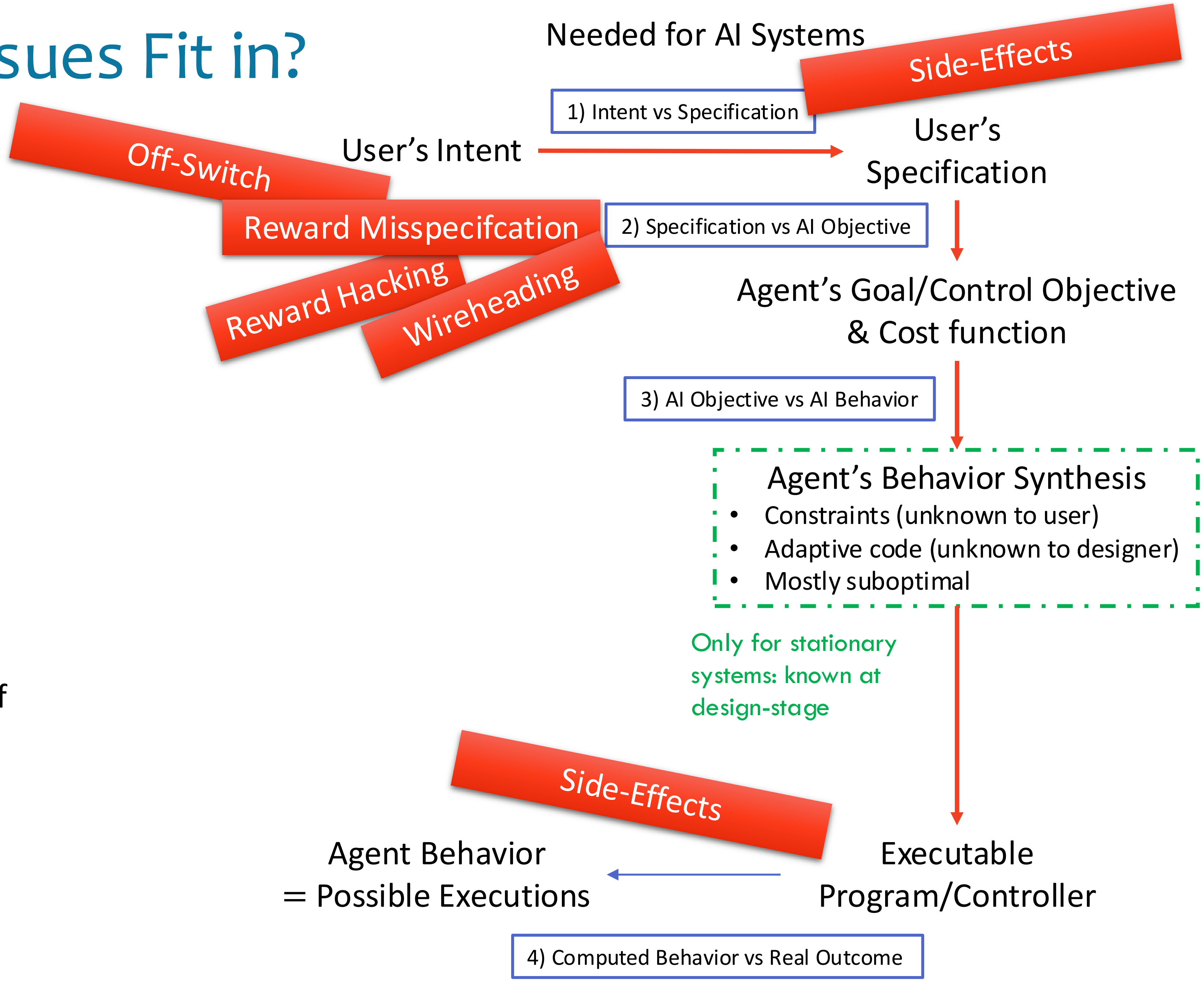
Wireheading

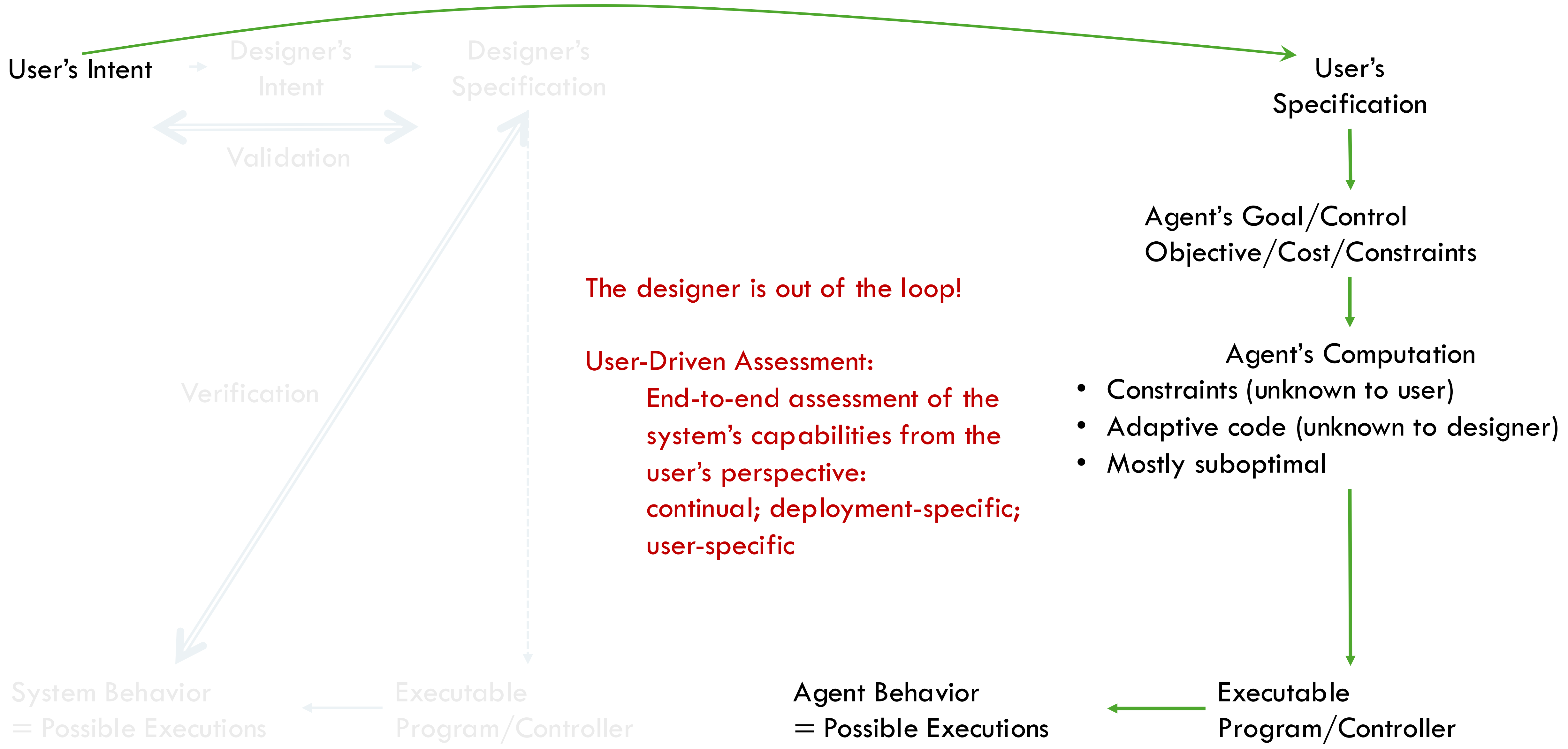
Reward Misspecification

Side Effects

Off-Switch

Agent doesn't let the user turn it off




User's Intent
Designer's Intent
Designer's Specification
Validation
Verification
System Behavior
= Possible Executions
Executable
Program/Controller
User's Specification
Agent's Goal/Control
Objective/Cost/Constraints
Agent's Computation
• Constraints (unknown to user)
• Adaptive code (unknown to designer)
• Mostly suboptimal
Agent Behavior
= Possible Executions
Executable
Program/Controller
The designer is out of the loop!
User-Driven Assessment:
End-to-end assessment of the system's capabilities from the user's perspective:
continual; deployment-specific; user-specific

Vocabulary + Semantics

Terms that the user understands
(e.g., “holding(x, gripper)”)

?

Query-Response
Protocol

Black-Box
AI

Arbitrary internal
implementation

Doesn’t know
user’s vocabulary

Vocabulary + Semantics

Terms that the user understands (e.g., “holding(x, gripper)”)

Interpretable model of Black-Box AI capabilities

Personalized AI Evaluator

(Query) instruction

(Response) result from sim

Black-Box AI

Arbitrary internal implementation

Doesn’t know user’s vocabulary

Assessment through Model Learning

Vocabulary + Semantics

Terms that the user understands (e.g., “holding(x, gripper)”)

Interpretable model of Black-Box AI capabilities

How does this model look like?

Personalized AI Evaluator

(Query) instruction

(Response) result from sim

Black-Box AI

Arbitrary internal implementation

Doesn’t know user’s vocabulary

Interpretable Description: PDDL/PPDDL

```
(:action open-door
   :parameters (?l1)
   :precondition (and
      (has_key)
      (player_at ?l1)
      (door_adjacent ?l1))
   :effect (probabilistic
      0.95 (and (door_open))
      0.05 (and (not (has_key))
                (game-over))
)
```

Precondition: This condition must be true for this action to execute

Effect: This is a set of conditions, one of which becomes true when this action is executed

Probabilities: Each set of effect has an associated probability with which that effect set is executed

Interpretable: Easily Convertible to Natural Language

```
(:action open-door
  :parameters (?l1)
  :precondition (and
     (has_key)
     (player_at ?l1)
     (door_adjacent ?l1))
  :effect (probabilistic
     0.95 (and (door_open))
     0.05 (and (not(has_key))
               (game-over))
)
```

The player can open the door when in location ?l1 if:

- It has the key
- The player is at location ?l1
- The door is adjacent to location ?l1

After executing that capability:

- With 95% probability, the door will open
- With 5% probability, the player will not have the key and the game will be over

Assessment using Passive Observations

$\langle s_0, a_1, s_1 \rangle$

$\langle s_1, a_2, s_2 \rangle$

$\vdots$

$\langle s_{n-1}, a_n, s_n \rangle$

[Input]

Learner

What kind of approaches these learners use?

```
(:action pick-up-beaker
   :parameters (?x - beaker)
     :precondition (and (ontable ?x)
        (not-in-use ?x)
        (handempty))
     :effect (and (not (ontable ?x))
         (not (handempty))
         (holding ?x)))

(:action put-on-shelf
   :parameters (?x - beaker)
     :precondition (holding ?x)
     :effect (and (not (holding ?x))
         (handempty)
         (onshelf ?x)))
```

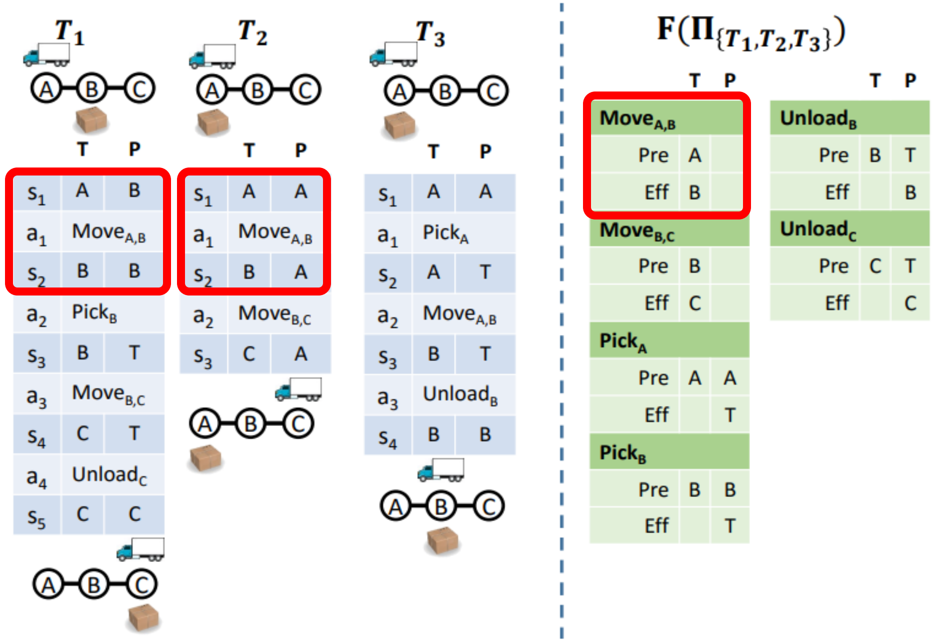
[PDDL Example]

Inference Rules based Learners

- Take intersection of all states where an action is applicable to create precondition.
- Take intersection of all states after executing an action to create effect.

$\langle s_0, a_1, s_1 \rangle$
$\langle s_1, a_2, s_2 \rangle$
$\vdots$
$\langle s_{n-1}, a_n, s_n \rangle$

[Input]




```
(:action move-A-B
  :parameters ()
  :precondition (and (at A))
  :effect (and (not (at A)) (at B))
)

(:action move-B-C
  :parameters ()
  :precondition (and (at B))
  :effect (and (not (at B)) (at C))
)
```

[PDDL Example]

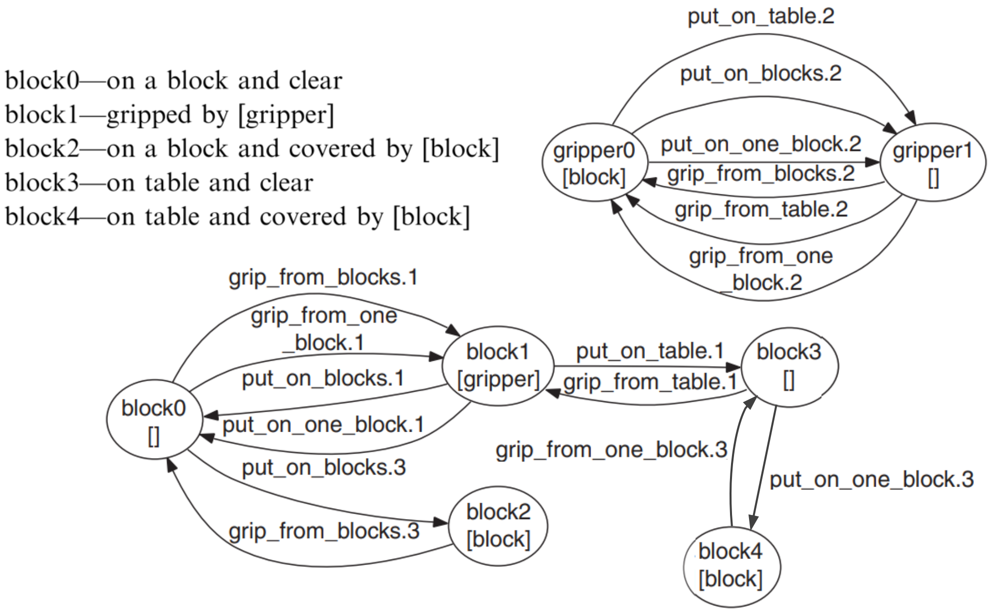
Stern et al. (IJCAI'17), SLAM - Juba et al. (KR'21)

Finite State Machine based Learners

- For each object type create a finite state machine.
- Create PDDL by combining them.

$$\langle s_0, a_1, s_1 \rangle$$
$$\langle s_1, a_2, s_2 \rangle$$
$$\vdots$$
$$\langle s_{n-1}, a_n, s_n \rangle$$

[Input]




```
(:action pick-up-beaker
   :parameters (?x - beaker)
     :precondition (and (ontable ?x)
        (not-in-use ?x)
        (handempty))
     :effect (and (not (ontable ?x))
         (not (handempty))
         (holding ?x)))

(:action put-on-shelf
   :parameters (?x - beaker)
     :precondition (holding ?x)
     :effect (and (not (holding ?x))
         (handempty)
         (onshelf ?x)))
```

[PDDL Example]

LOCM - Cresswell et al. (ICAPS’09), LOCM2 - Cresswell et al. (ICAPS’11, Know. Engg. Rev.’13),
LOP - Gregory et al. (ICAPS’15), NLOCM - Gregory et al. (ICAPS’16)

SAT based Learners

- Create a SAT problem using constraint axioms.
- Extract PDDL from the SAT problem's solution.

$$\langle s_0, a_1, s_1 \rangle$$
$$\langle s_1, a_2, s_2 \rangle$$
$$\vdots$$
$$\langle s_{n-1}, a_n, s_n \rangle$$

[Input]

1. $(par(p_k) \cap par(pri_i) = \phi) \wedge (par(p_k) \cap par(pre_{goal}) = \phi) \Rightarrow p_k \notin add_i \wedge p_k \notin del_i$
2. $pre_i \neq \phi \wedge add_i \neq \phi \wedge del_i \neq \phi$
3. $pre_i \cap add_i = \phi$
4. $del_i \subseteq pre_i$.

.
.
.

```
(:action pick-up-beaker
   :parameters (?x - beaker)
     :precondition (and (ontable ?x)
        (not-in-use ?x)
        (handempty))
     :effect (and (not (ontable ?x))
         (not (handempty))
         (holding ?x)))

(:action put-on-shelf
   :parameters (?x - beaker)
     :precondition (holding ?x)
     :effect (and (not (holding ?x))
         (handempty)
         (onshelf ?x)))
```

[PDDL Example]

ARMS -Yang et al. (AIJ 2007), Zhuo et al. (IJCAI'13)

Planning based Learners

- Create Planning problem using SAT-like rules.
- Extract correct PDDL from solution to the planning problem.

$$\langle s_0, a_1, s_1\rangle$$
$$\langle s_1, a_2, s_2\rangle$$
$$\vdots$$
$$\langle s_{n-1}, a_n, s_n\rangle$$

[Input]

→

```
(:action apply_stack
  :parameters (?o1 - object ?o2 - object)
  :precondition
   (and (or (not (pre_stack_on_v1_v1)) (on ?o1 ?o1))
        (or (not (pre_stack_on_v1_v2)) (on ?o1 ?o2))
        (or (not (pre_stack_on_v2_v1)) (on ?o2 ?o1))
        (or (not (pre_stack_on_v2_v2)) (on ?o2 ?o2))
                        . . .
        (or (not (pre_stack_handempty)) (handempty)))

  :effect
   (and (when (del_stack_on_v1_v1) (not (on ?o1 ?o1)))
        (when (del_stack_on_v1_v2) (not (on ?o1 ?o2)))
        (when (del_stack_on_v2_v1) (not (on ?o2 ?o1)))
        (when (del_stack_on_v2_v2) (not (on ?o2 ?o2)))
                        . . .
        (when (add_stack_holding_v1) (holding ?o1))
        (when (add_stack_holding_v2) (holding ?o2))
        (when (add_stack_handempty) (handempty))
        (when (modeProg) (not (modeProg)))))
```

→

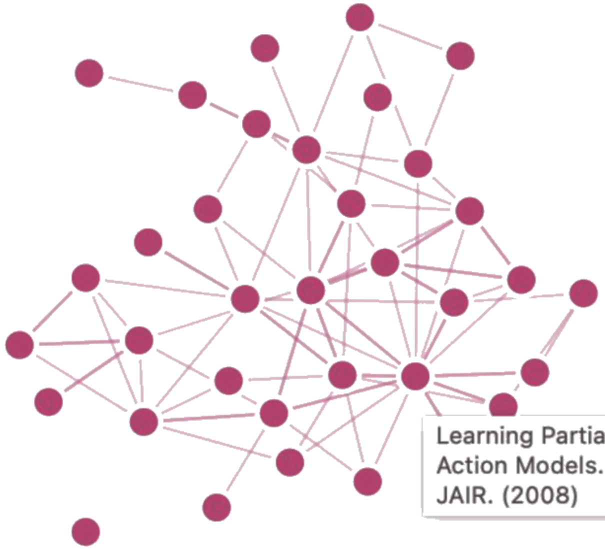
```
(:action pick-up-beaker
   :parameters (?x - beaker)
     :precondition (and (ontable ?x)
        (not-in-use ?x)
        (handempty))
     :effect (and (not (ontable ?x))
         (not (handempty))
         (holding ?x)))

(:action put-on-shelf
   :parameters (?x - beaker)
     :precondition (holding ?x)
     :effect (and (not (holding ?x))
         (handempty)
         (onshelf ?x)))
```

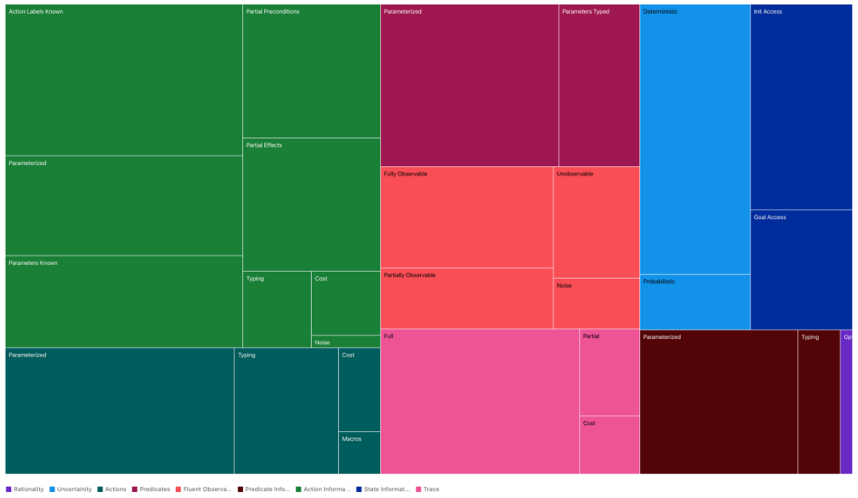
[PDDL Example]

FAMA – Aineto et al. (ICAPS'18, AIJ'19)

MACQ: Model Acquisition Toolkit

- Library of passive learning approaches
- Re-implementations of landm approaches
- Open source
- Visualization tools





Neighboring Papers

To get to this new paper, our AI thinks you should be looking at the following papers known to our system as the state of the art that immediately makes the new work possible. Each paper is tagged with features that need relaxation or extension to get to the new paper.

Efficient, Safe, and Probably Approximately Complete Learning of Action Models *by Stern, Roni, and Brendan Juba*. IJCAI (2017)

Learning Parameters / Data Features / Trace / Cost / True

Constructing Symbolic Representations for High-Level Planning *by Konidaris, George, Leslie Kaelbling, and Tomas Lozano-Perez*. AAAI (2014)

Learning Parameters / Data Features / State Information / Init Access / False
Learning Parameters / Data Features / Trace / Cost / True

Learning First-Order Representations for Planning from Black-Box States: New Results *by Rodriguez, Ivan D., Blai Bonet, Javier Romero, and Hector Geffner*. arXiv (2021)

Learning Parameters / Model Features / Actions / Parameterized / False
Learning Parameters / Model Features / Predicates / Parameterized / False
Learning Parameters / Data Features / Fluent Observability / Fully Observable / True
Learning Parameters / Data Features / Fluent Observability / Unobservable / False
Learning Parameters / Data Features / Fluent Observability / Noise / False
Learning Parameters / Data Features / Trace / Cost / True

Tutorial on Model Acquisition using MACQ

https://icaps23.icaps-conference.org/program/tutorials/model/

HOME | DATES | ATTENDING | CALLS | COMPETITIONS | SUBMISSIONS | PROGRAM | SCHEDULE | COMMITTEES | CODE OF CONDUCT | VENUE PHOTO

Model Acquisition in the Modern Era (Tutorial Materials)

Description

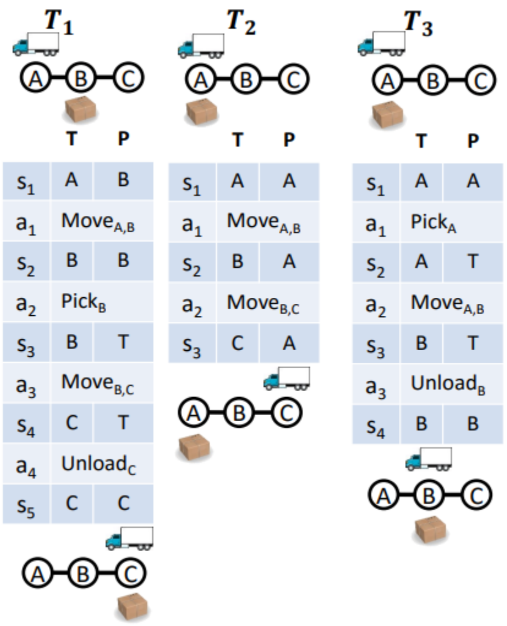
This tutorial will cover some of the landmark methods in the area of planning action model acquisition that our community has produced over the years. From OBSERVER in the early 90's to the modern forms of action-label-only LOCM techniques, we will cover both the concepts behind these approaches and grounded hands-on examples for attendees to try for themselves.

Limitations of Learning from Passive Observations

- Susceptible to incorrect or incomplete model learning.

- E.g., if all packages are brown in color, a possible precondition will be that the package must be brown to unload them.

- Such methods don’t capture correct causal relationships.



Active Acquisition of Observations

- Does not depend on third-party to provide observations.
- Strategy to acquire observations:
  - Directed Search: What action should I execute more to acquire more samples?

EXPO – Gil (ICML'93), IRALe – Rodrigues et al. (ILP'11), GLIB – Chitnis et al. (AAAI'21)

Online Learning of Action Models for PDDL Planning

Leonardo Lamanna, Alessandro Saetti, Luciano Serafini, Alfonso Emilio Gerevini, and Paolo Traverso

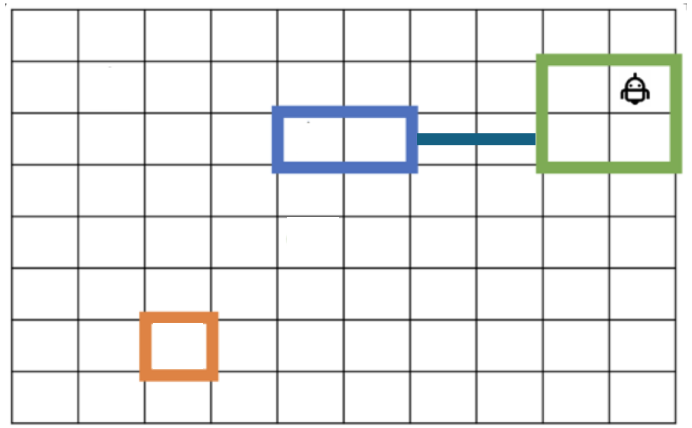
IJCAI 2021

Online Learning of Action Models for PDDL Planning

- **Assumptions:**
  - the set of predicates, operators and objects are known;
  - no negative preconditions and inconsistent effects;
  - full observability.

- **Two ways to learn from action executions:**
  - Learn from execution success.
  - Learn from execution failures.

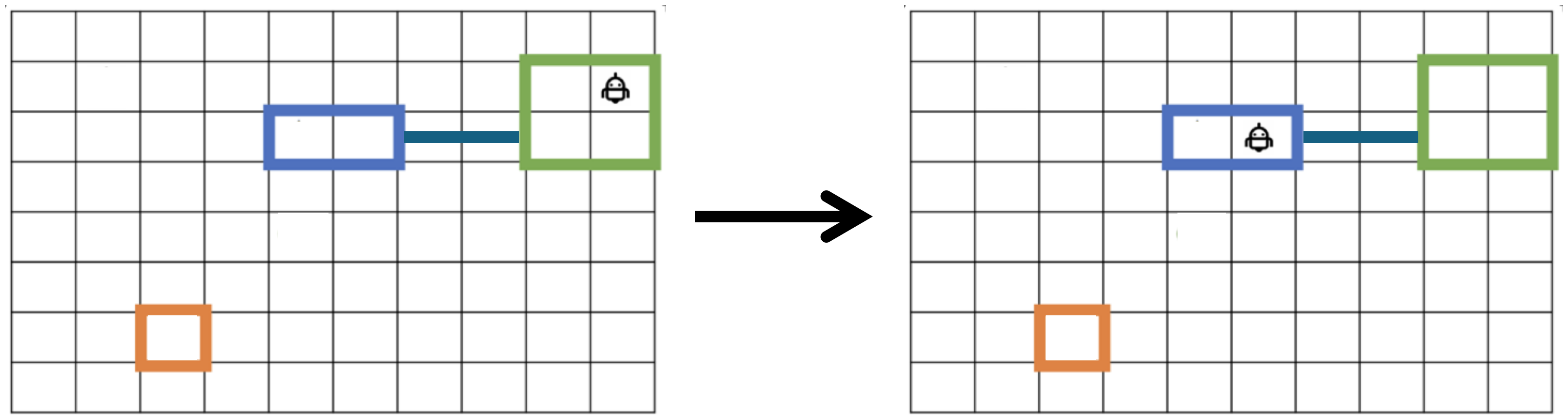
Learning from Action Execution Success



```
(:action move
  :parameters (?from ?to)
  :precondition (and
        (at ?from)
        (connected ?to ?from)
        (at ?to))
  :effect (and )
)
```

Learning from Action Execution Success

- **If action successful**
  - Remove incorrect preconditions.
  - Add necessary effects.



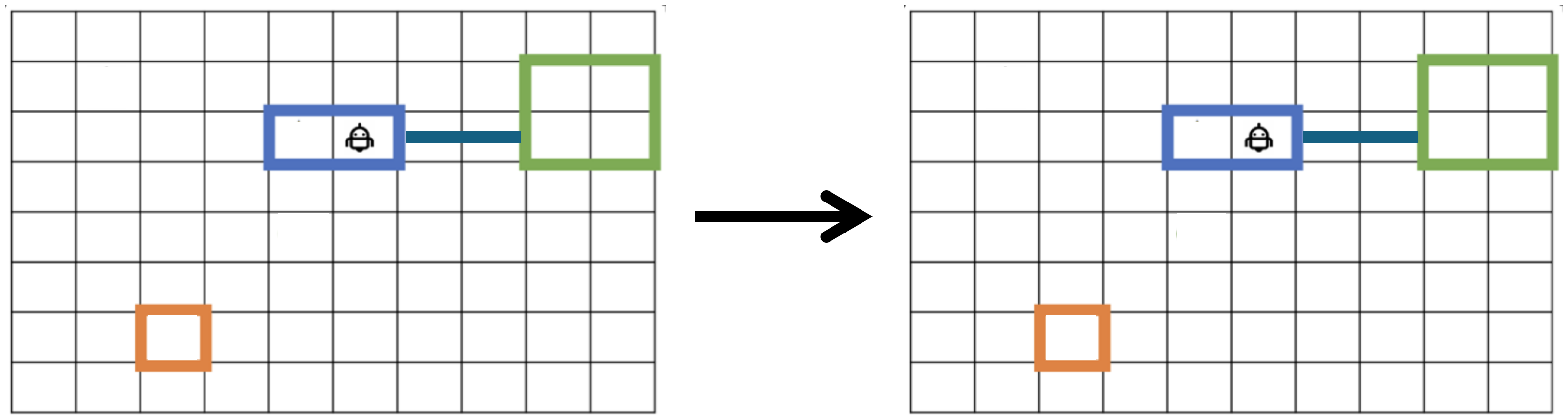
move(roomG roomB)

```
(:action move
  :parameters (?from ?to)
  :precondition (and
        (at ?from)
        (connected ?to ?from)
        (at ?to))
  :effect (and
        (at ?to)
        (not (at ?from)))
)
```

Learning from Action Execution Failure

- **If action failed**
  - Confirm preconditions.

```
(:action move
  :parameters (?from ?to)
  :precondition (and
        (at ?from)
        (connected ?to ?from))
  :effect (and
        (at ?to)
        (not (at ?from)))
)
```

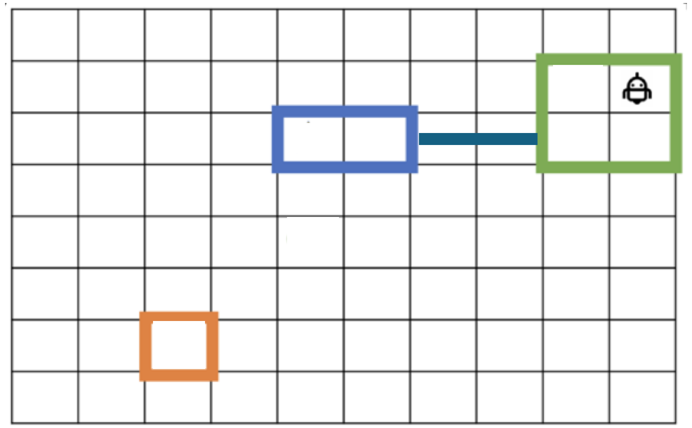


`move(roomB room0)`

OLAM Algorithm

```
(at robot roomG)
(connected roomG roomB)
⋮
```

PDDL State



Goal Specification

```
move(roomG roomB)
move(roomB roomO)
⋮
```

Plan π

$a = pop(\pi)$

execute(a)

Success

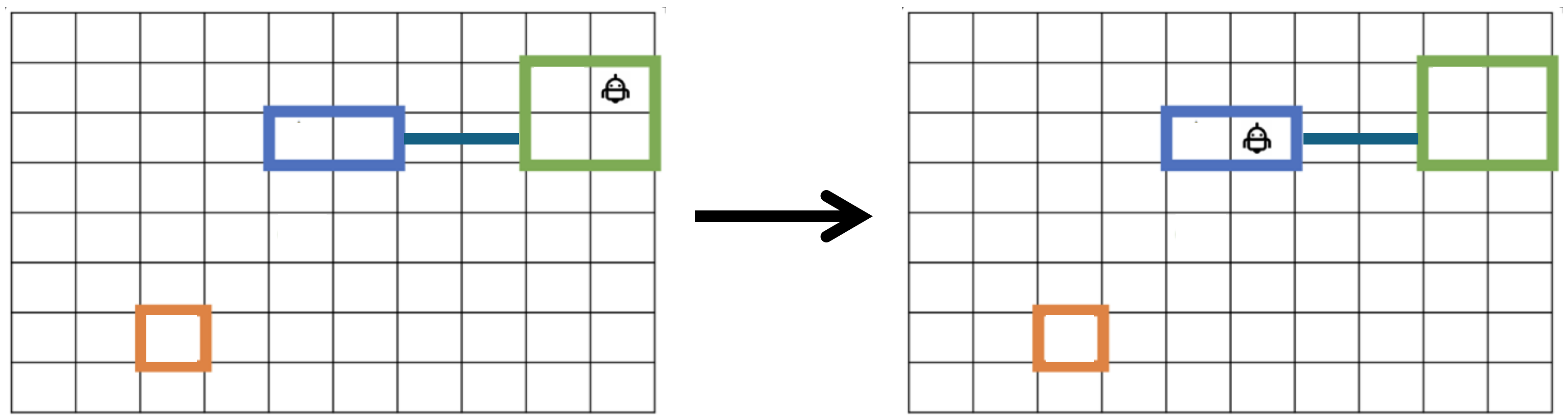
```
(:action move
  :parameters (?from ?to)
  :precondition (and
        (at ?from)
        (connected ?to ?from)
        (at ?to))
  :effect (and
        (at ?to)
        (not (at ?from))))
```

Failure

```
(:action move
  :parameters (?from ?to)
  :precondition (and
        (at ?from)
        (connected ?to ?from))
  :effect (and
        (at ?to)
        (not (at ?from)))
)
```

Goal Specification for OLAM

$s \xrightarrow{\pi} s'$



`move(roomG roomB)`

Goal Specification for OLAM

$$s \xrightarrow[\pi]{} s' \xrightarrow[op(c)]{} s''$$

Precondition:

- P^+: atoms true in s'
- P^-: atoms false in s' and are yet to be verified as necessary for executing $op(c)$

Effect:

- E^+: possible effects false in s' but can become true on executing $op(c)$
- E^-: possible effects true in s' but can become false on executing $op(c)$

Goal Specification for OLAM

Precondition:
- P^+: atoms true in s'
- P^-: atoms false in s' and are yet to be verified as necessary for executing $op(c)$

Effect:
- E^+: possible effects false in s' but can become true on executing $op(c)$
- E^-: possible effects true in s' but can become false on executing $op(c)$

$$\text{Goal} = \bigvee_{\substack{op(c) \in A \\ P^+P^-E^+E^- \textit{ satisfy (i -vi)}}} \left(\bigwedge_{p(c) \in P^+ \cup E^-} p(c) \wedge \bigwedge_{p(c) \in P^- \cup E^+} \neg p(c) \right)$$

(i) $P^- \cup E^+ \cup E^- \neq \emptyset$
(ii) $P^+ \cap P^- = \emptyset$
(iii) $P^+ \cup P^- = pre(op(c))$
(iv) $P^- \notin pre_{\perp}(op(c))\{\emptyset\}$
(v) $E^+ \subseteq eff_?^+(op(c))$
(vi) $E^- \subseteq eff_?^-(op(c))$

OLAM outperforms the baseline in accuracy

$$P = \frac{TP}{TP+FP}$$

$$R = \frac{TP}{TP+FN}$$

	OLAM			Fama		
Domain	Time	P	R	Time	P	R
blocksworld	5.03	**1**	**1**	510	**1**	**1**
driverlog	20.42	**0.93**	**1**	349	0.79	0.85
ferry	7.54	**0.94**	**1**	267	0.80	0.93
floortile	47.34	**0.83**	**1**	517	0.82	0.78
grid	36.92	**0.82**	**1**	306	0.81	0.74
gripper	3.50	**1**	**1**	165	0.86	0.93
hanoi	2.38	**0.88**	**1**	818	**0.88**	0.86
miconic	4.24	**1**	**1**	200	0.81	**1**
n-puzzle	1.97	**0.88**	**1**	23	0.86	**1**
parking	183.94	**0.89**	**1**	895	0.84	0.84
rover	154.10	**0.83**	**0.84**	629	0.51	0.53
satellite	11.26	**1**	**1**	65	0.70	0.89
transport	74.98	**0.95**	**1**	280	0.80	0.89

GLIB: Efficient Exploration for Relational Model-Based Reinforcement Learning via Goal-Literal Babbling

Rohan Chitnis, Tom Silver, Joshua Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez

AAAI 2021

Exploration via Goal-Literal Babbling (GLIB)

1. Sample (babble) a conjunctive goal that has not yet been seen
   i. Max number of literals in conjunction is a hyperparameter
   ii. Whether the goals are lifted or ground is a hyperparameter

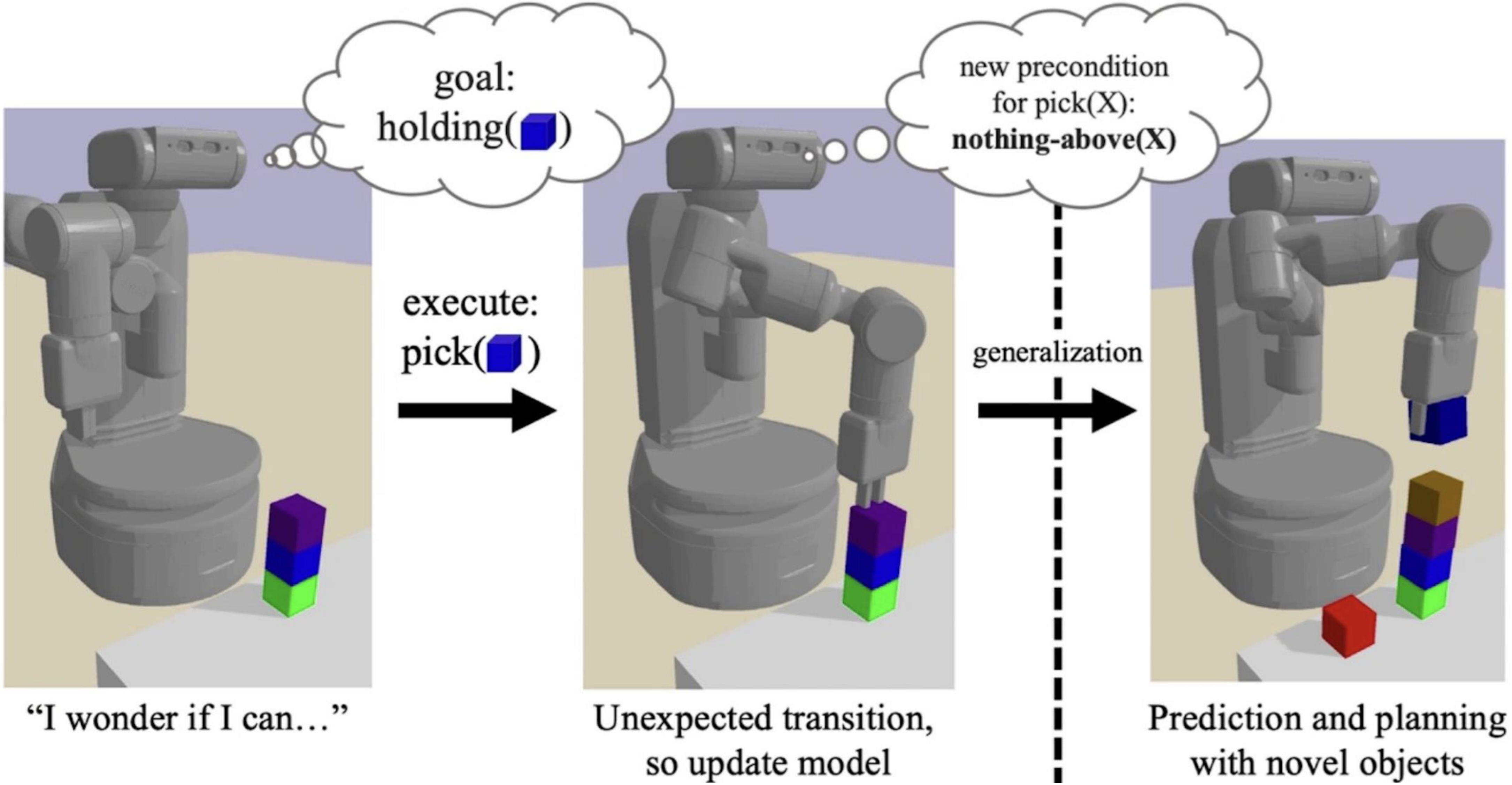
2. Plan to achieve the goal using the current (wrong) operators

3. Execute the plan to acquire data

4. Use the resulting data to improve the operators

5. Repeat

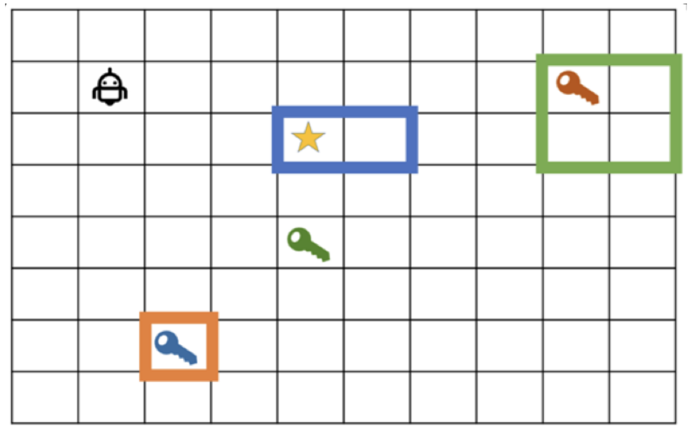
GLIB can find errors and update the model



Exploration via Goal-Literal Babbling (GLIB)

- Sample a novel (goal, action) pair.
- If we can't sample a goal that yields a non-empty plan after several tries, fall back to taking a random action.
- Ground goals (GLIB-G) vs. lifted goals (GLIB-L): GLIB-G tends to under-generalize while GLIB-L tends to over-generalize.

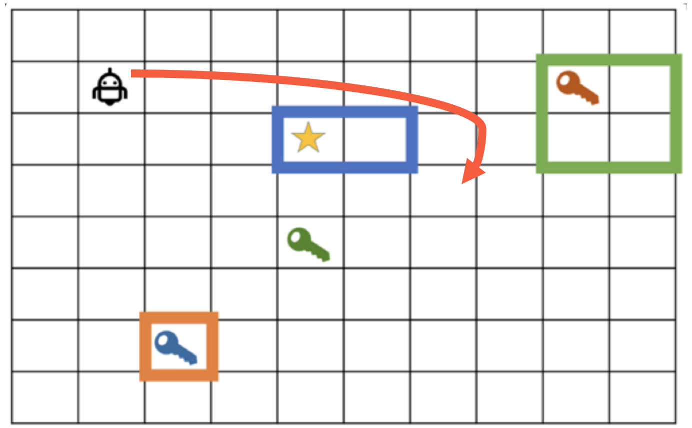
Exploration in GLIB-L



No goals achievable

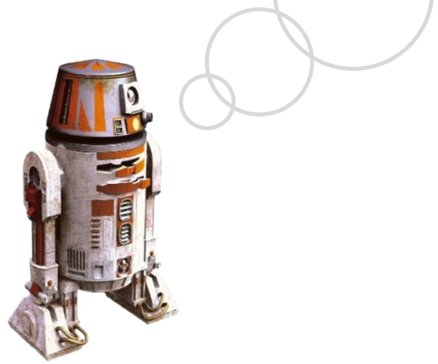
```
(:action move
  :parameters (?from ?to)
  :precondition ( )
  :effect ( )
)
```

Exploration in GLIB-L



No goals achievable

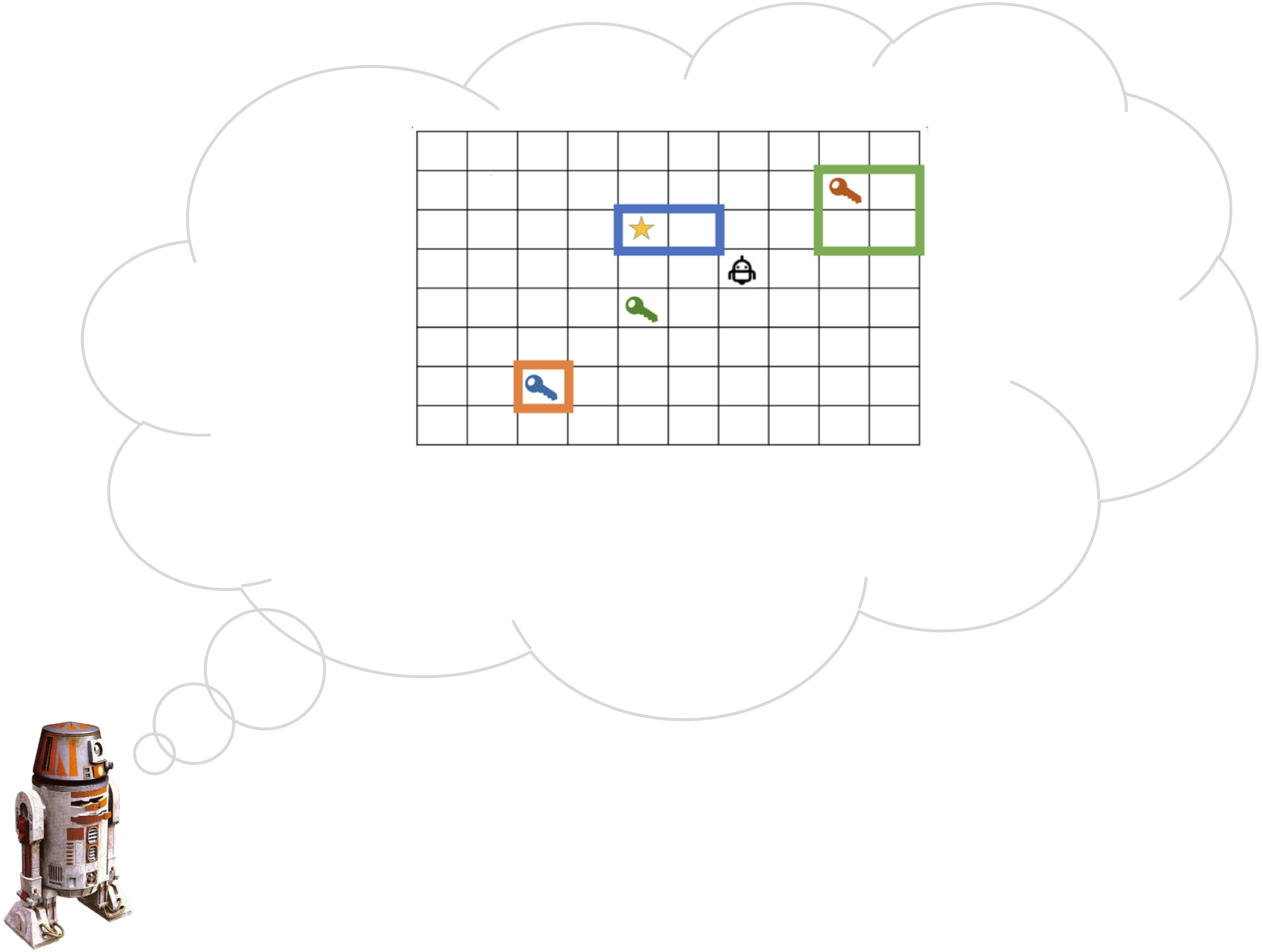
Sample random action: move (7-2, 5-7)



```
(:action move
  :parameters (?from ?to)
  :precondition ( )
  :effect ( )
)
```

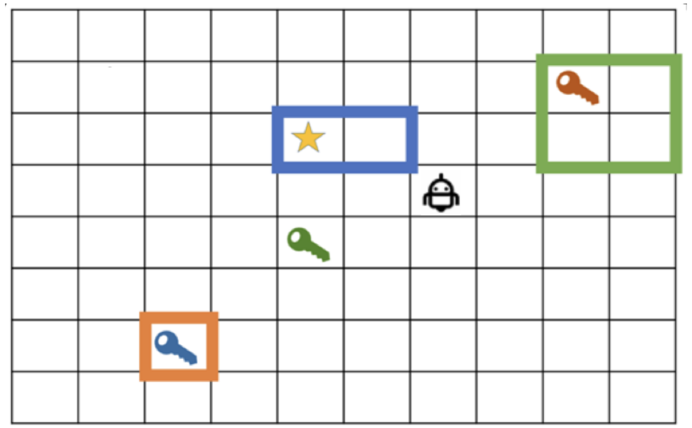

54

Exploration in GLIB-L



```
(:action move
  :parameters (?from ?to)
  :precondition (and
        (at ?from))
  :effect (and
        (not (at ?from))
        (at ?to))
)
```

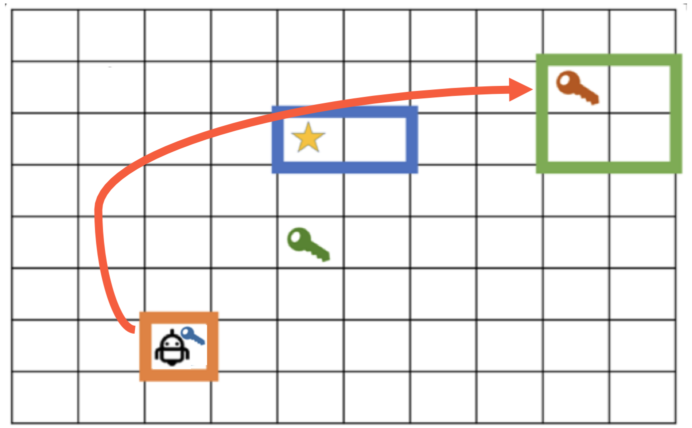
Exploration in GLIB-L



Babble Goal: at(2-3) ∧ keyat(keyB, 2-3)
with final action: pick(2-3, keyB)

```
(:action move
  :parameters (?from ?to)
  :precondition (and
          (at ?from))
  :effect (and
          (not (at ?from))
          (at ?to))
)
```

Exploration in GLIB-L

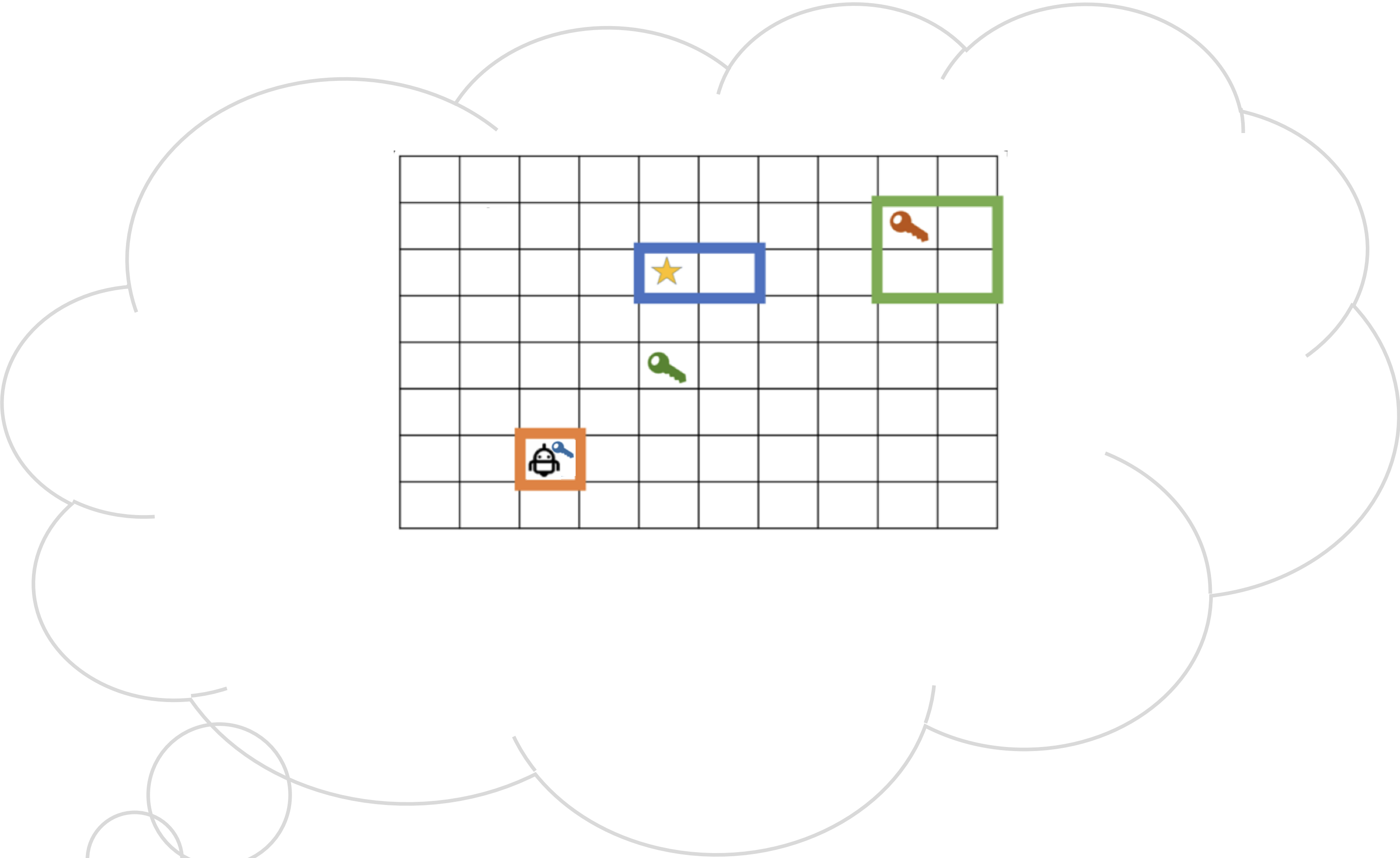


Babble Goal: at(7-9) ∧ locked(roomG)
with final action: move(7-9, 6-5)

Plan: ⟨move(2-~~6~~,7-9), move(7-9,6-5)⟩

```
(:action move
   :parameters (?from ?to)
   :precondition (and
         (at ?from))
   :effect (and
         (not (at ?from))
         (at ?to))
)
(:action pick
   :parameters (?loc ?room)
   :precondition (and
         (keyat ?loc)
         (at ?loc)
         (keyforroom ?room))
   :effect (and
         (not (keyat ?loc))
         (not (locked ?room)))
)
```

Exploration in GLIB-L



```
(:action move
  :parameters(?from ?to ?room)
  :precondition (and
        (at ?from)
        (inroom ?to ?room)
        (not (locked ?room))
  : effect (and
        (not (at ?from))
        (at ?to))
)
(:action pick
  :parameters (?loc ?room)
  :precondition (and
        (keyat ?loc)
        (at ?loc)
        (keyforroom ?room))
  :effect (and
        (not (keyat ?loc))
        (not (locked ?room)))
)
```

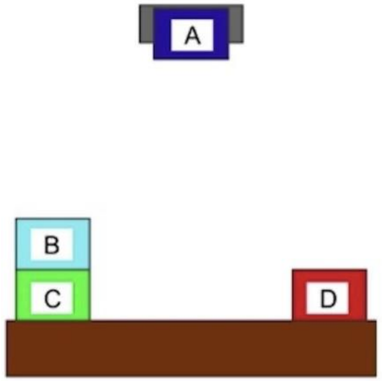
Theoretical Properties of GLIB

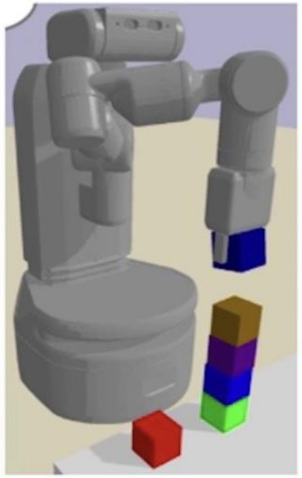
- **Theorem**: Under mild assumptions about the environment, planner, and operator learning algorithm, GLIB will visit all reachable transitions infinitely often in the limit.
- **Corollary**: The model learned using GLIB will converge almost surely to the ground truth model over the space of reachable transitions.

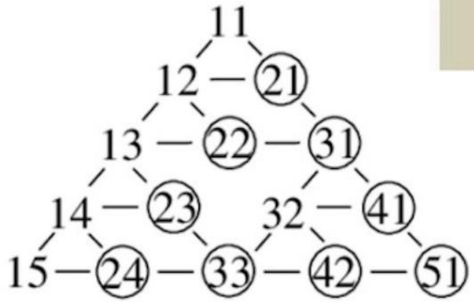
Empirical Evaluation

- Measured the following as a function of the number of interactions with the environment.
  - Prediction accuracy of the learned operators
  - Planning performance of the learned operators on a hand-designed test set of goals

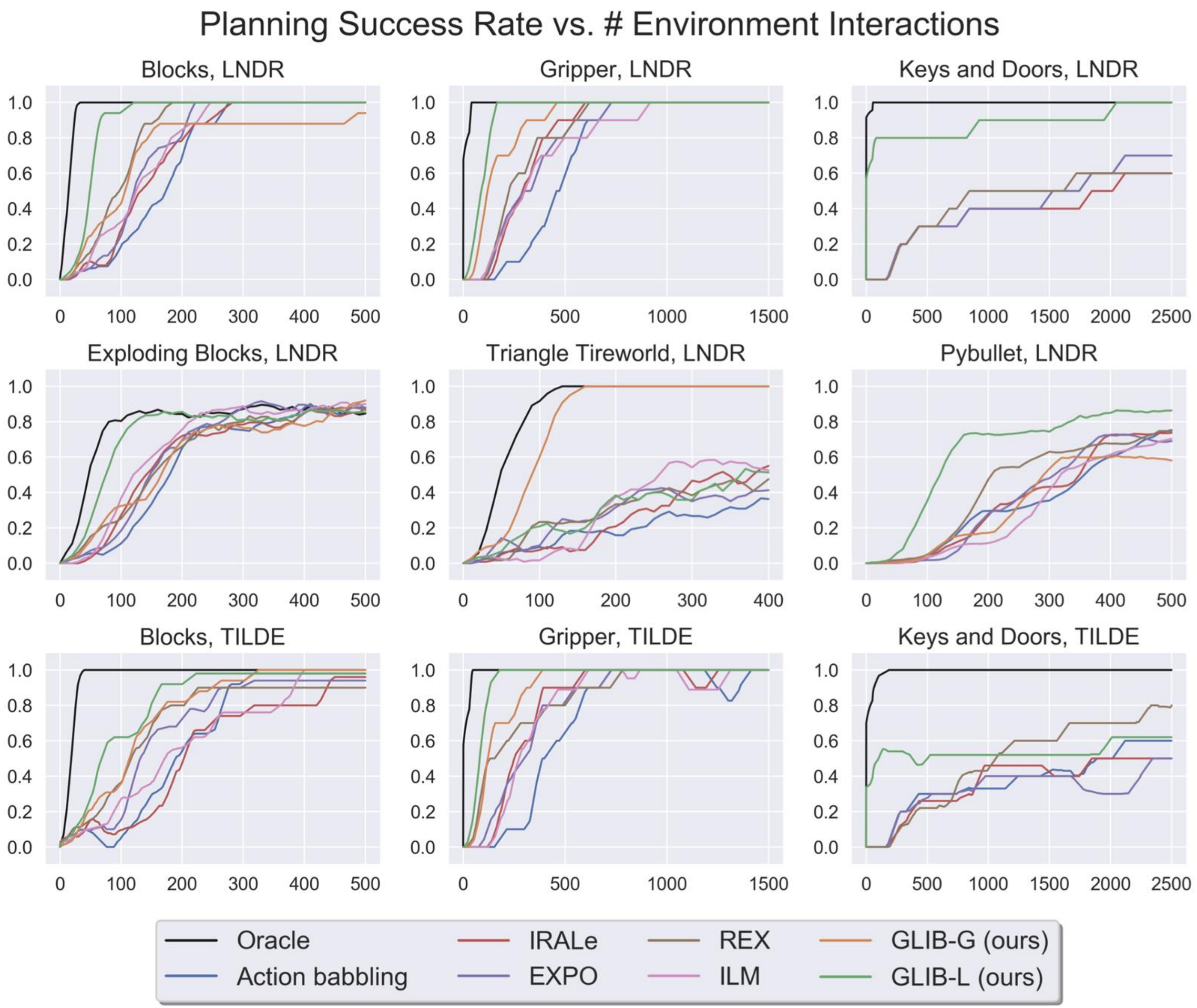
- Baselines: SOTA algorithms for exploration in relational model-based RL.
  - REX (Lang 2012), ILM (Ng 2019), IRALe (Rodrigues 2011), EXPO (Gil 1994)







GLIB is sample efficient



Assessment of Black-Box AI Systems in Stationary Settings

Vocabulary + Semantics
Terms that the user understands
(e.g., “holding(x, gripper)”)

Interpretable model
of
Black-Box AI
capabilities

Personalized
AI Evaluator

(Query)
instruction

(Response) result
from sim

Black-Box
AI

Arbitrary internal
implementation

Doesn’t know
user’s vocabulary

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

Pulkit Verma, Shashank Rao Marpally, and Siddharth Srivastava

AAAI 2021

Deterministic and Stationary Setting

Input

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

Output

- PPDDL-like description of each capability.

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).

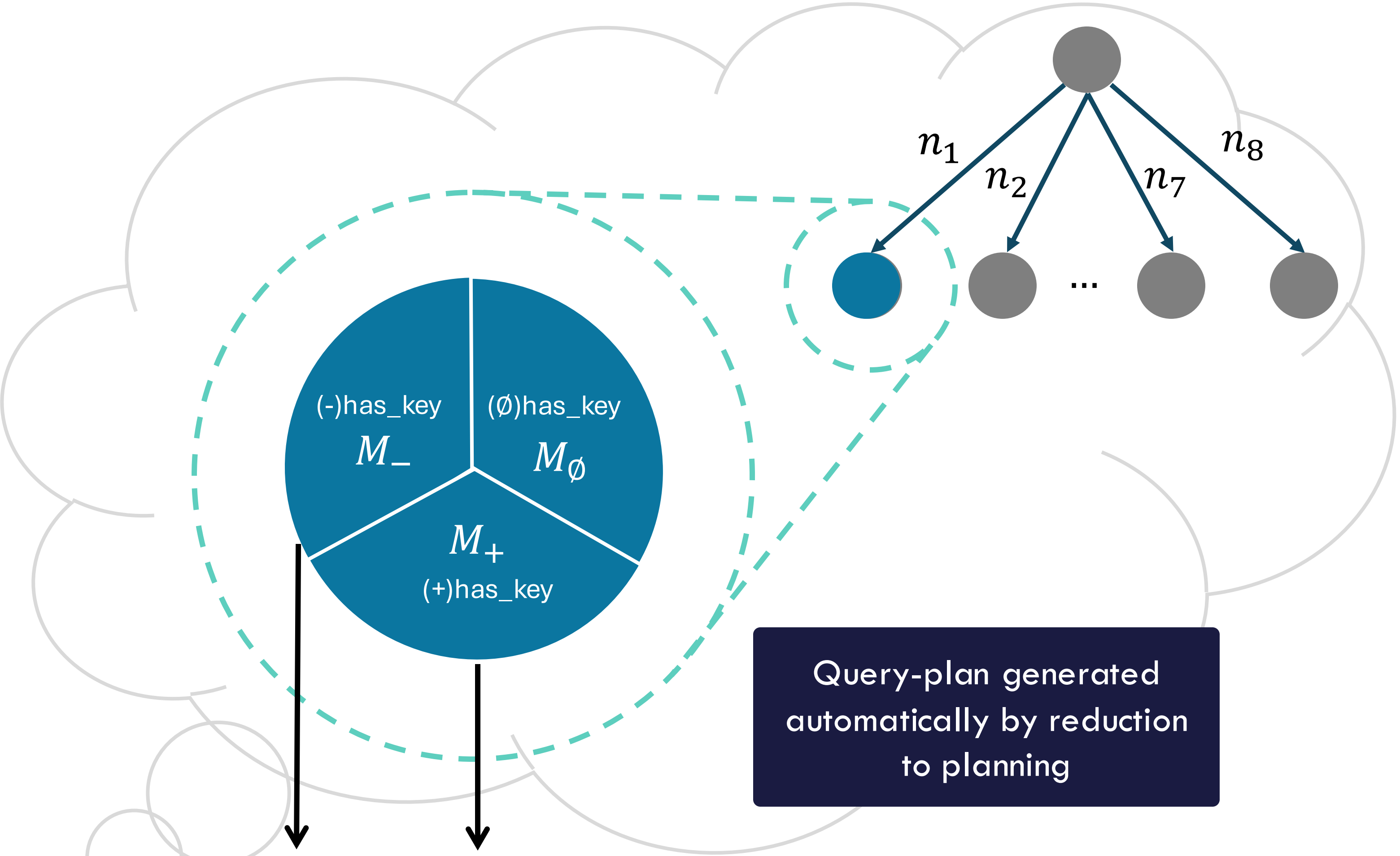
```
(:action open-door
  :parameters (?l1)
  :precondition (and
    (+/-/∅)(has_key)
    (+/-/∅)(door_open)
    (+/-/∅)(door_adjacent ?l1)
    (+/-/∅)(player_at ?l1))
  :effect (and
    (+/-/∅)(has_key)
    (+/-/∅)(door_open)
    (+/-/∅)(door_adjacent ?l1)
    (+/-/∅)(player_at ?l1))
```

Simple Queries

	Plan Outcome Queries	State Reachability Query
Query	In state s_I, what will happen if you execute the plan $\pi = \langle c_1, \dots, c_n \rangle$?	Can you go from state s_I to state s_F?
Response	I can execute first ℓ steps of the plan, ending up in state s_F.	Yes / No.

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

Hierarchical Query Synthesis



Generate a
distinguishing query:
Q such that $Q(M_-) \neq Q(M_+)$

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

Query Synthesis as Planning

Models differ in only one predicate in precondition or effect.

M_-

```
(:action open-door
  :parameters (?loc)
  :precondition (and
    (p1) (p2))
  :effect (and
    (p3)
    (not (has-key)))
```

M_+

```
(:action open-door
  :parameters (?loc)
  :precondition (and
    (p1) (p2))
  :effect (and
    (p3)
    (has-key))
```

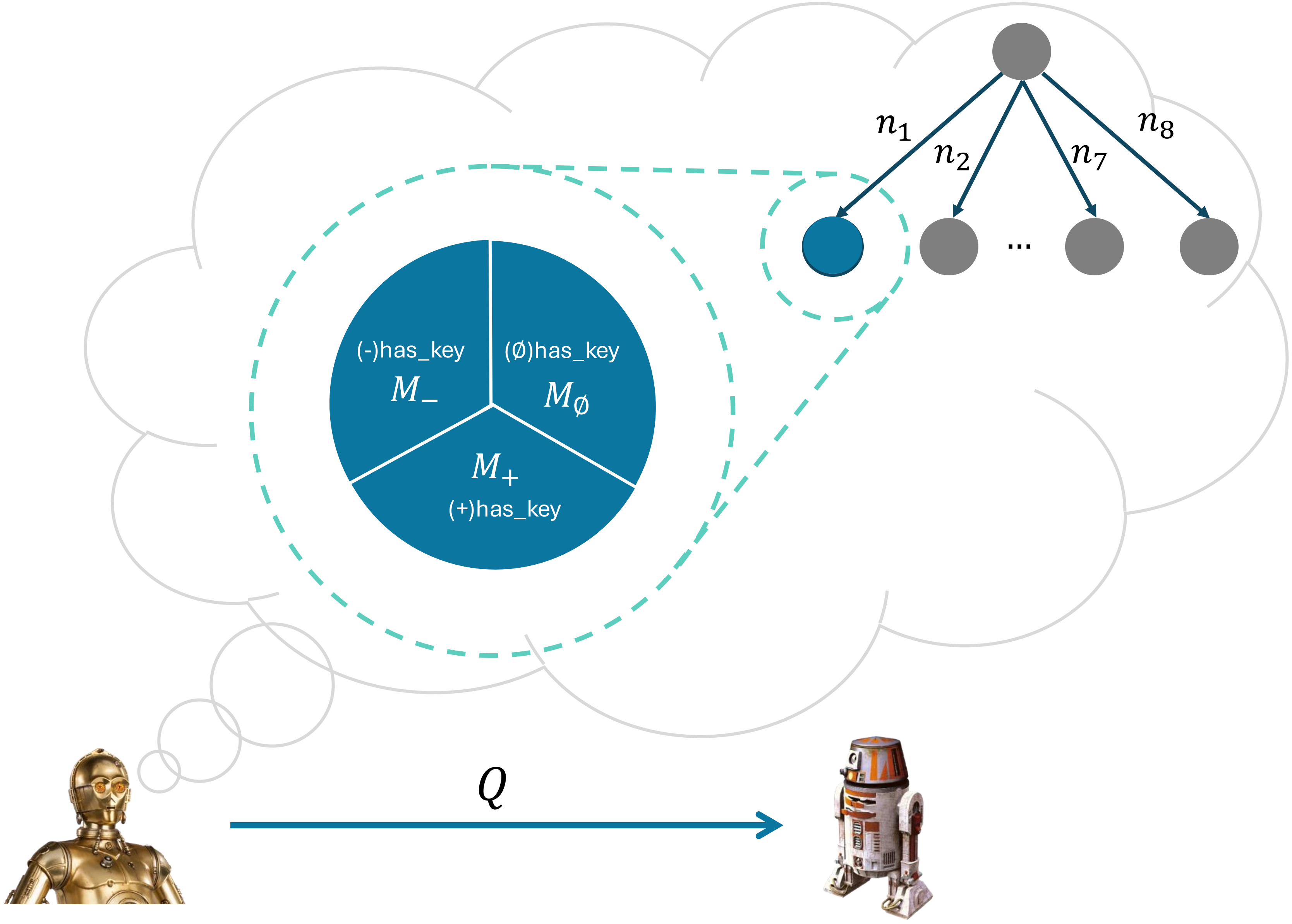
open-door (?location ?item)
  precondition:
    $(\text{precondition})_{M_-} \lor (\text{precondition})_{M_+}$
  effect:
    $((\text{precondition})_{M_-} \land !(\text{precondition})_{M_+} \rightarrow (\text{goal}))$
    $(!(\text{precondition})_{M_-} \land (\text{precondition})_{M_+} \rightarrow (\text{goal}))$
    $((\text{precondition})_{M_-} \land (\text{precondition})_{M_+} \rightarrow ((\text{effect})_{M_-} \land (\text{effect})_{M_+}))$

Consolidated capability used to generate the Planning Domain

If the precondition of only one model is satisfied, the goal is reached.

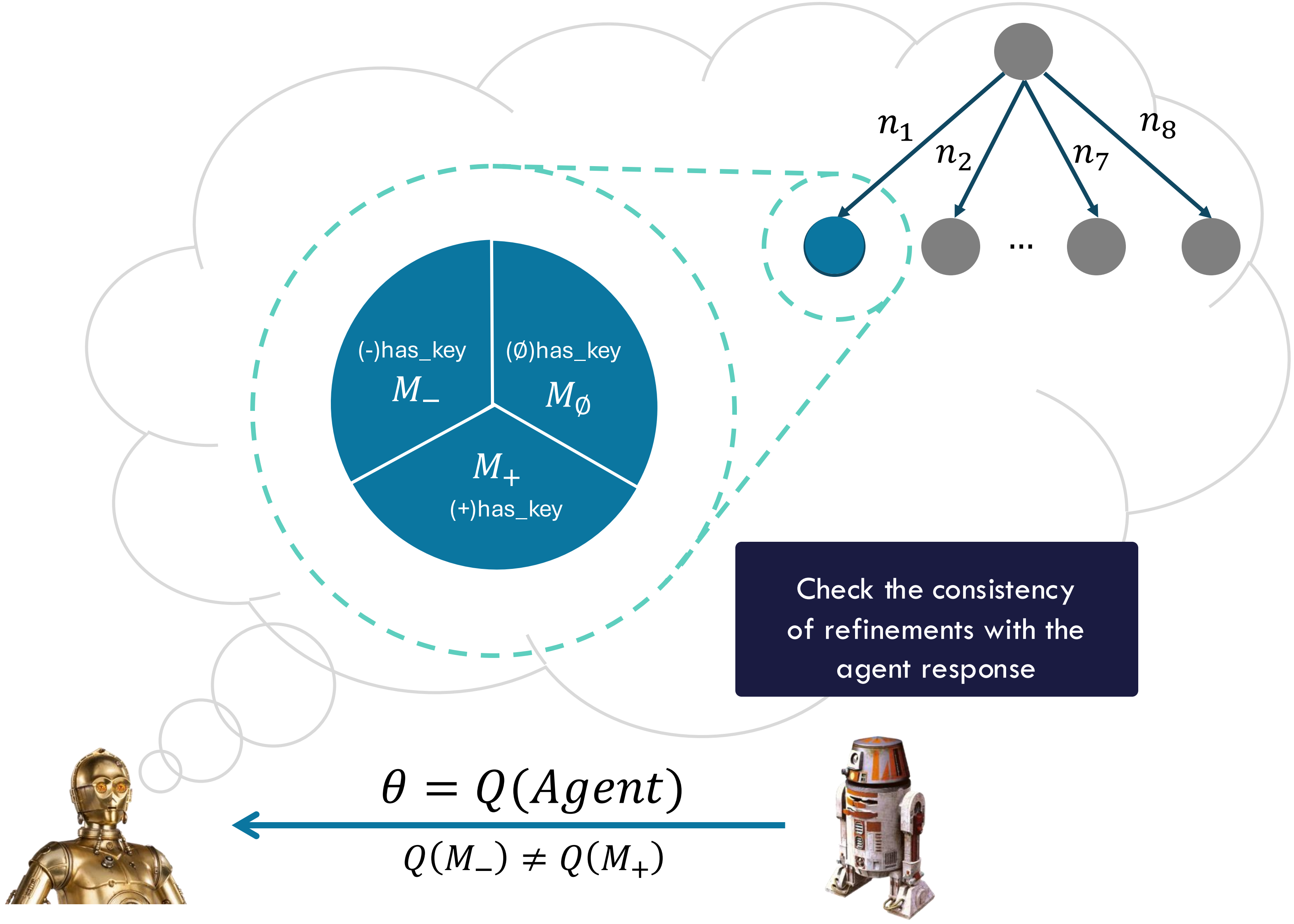
If the preconditions of both models are satisfied, apply the effects of both.

Hierarchical Query Synthesis



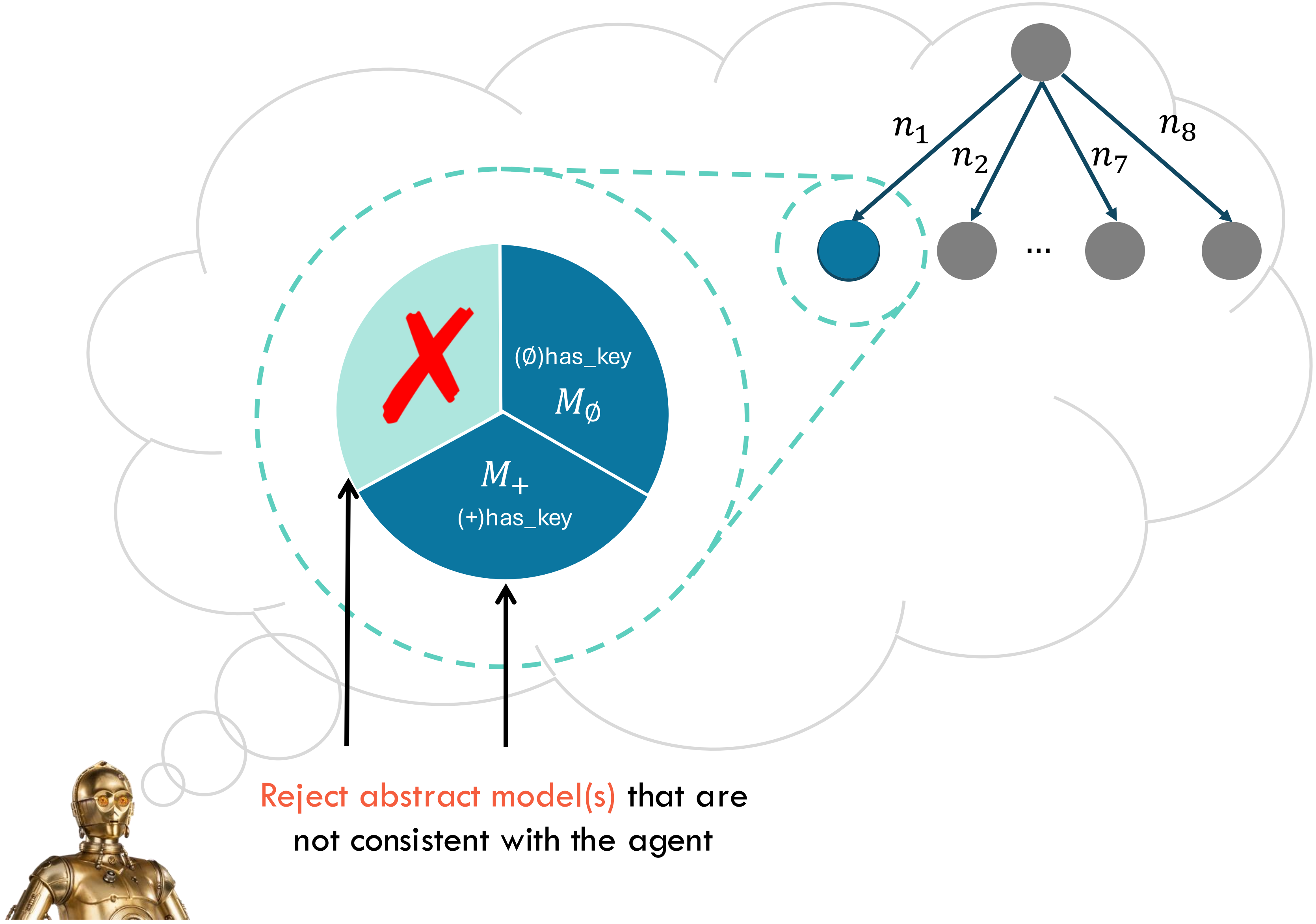

```
(:action open-door
  :parameters (?l1)
  :precondition (and
n1  (+/-/Ø)(has_key)
n2  (+/-/Ø)(door_open)
n3  (+/-/Ø)(door_adjacent ?l1)
n4  (+/-/Ø)(player_at ?l1))
  :effect (and
n5  (+/-/Ø)(has_key)
n6  (+/-/Ø)(door_open)
n7  (+/-/Ø)(door_adjacent ?l1)
n8  (+/-/Ø)(player_at ?l1))
```

Hierarchical Query Synthesis




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

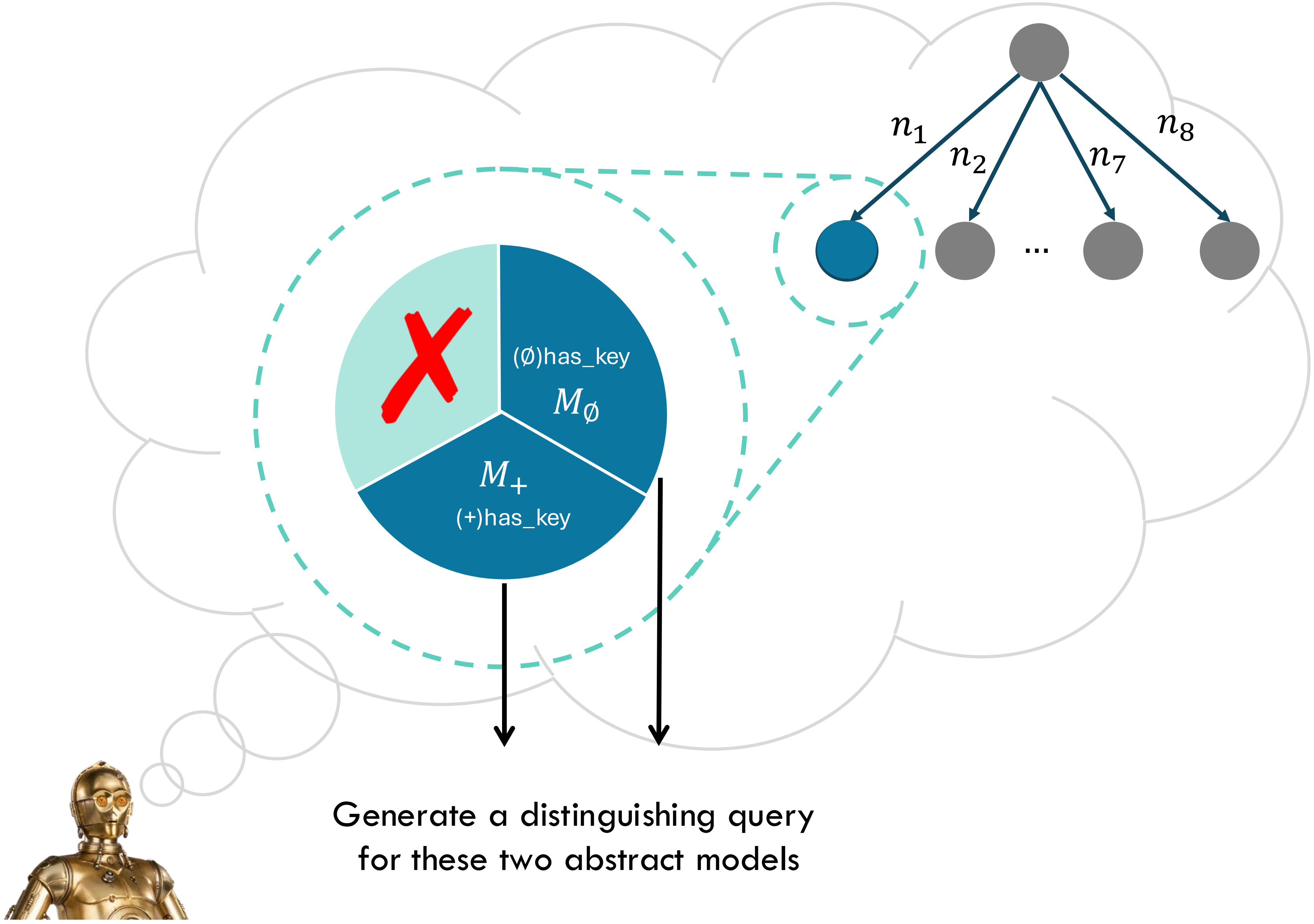
Hierarchical Query Synthesis



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

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

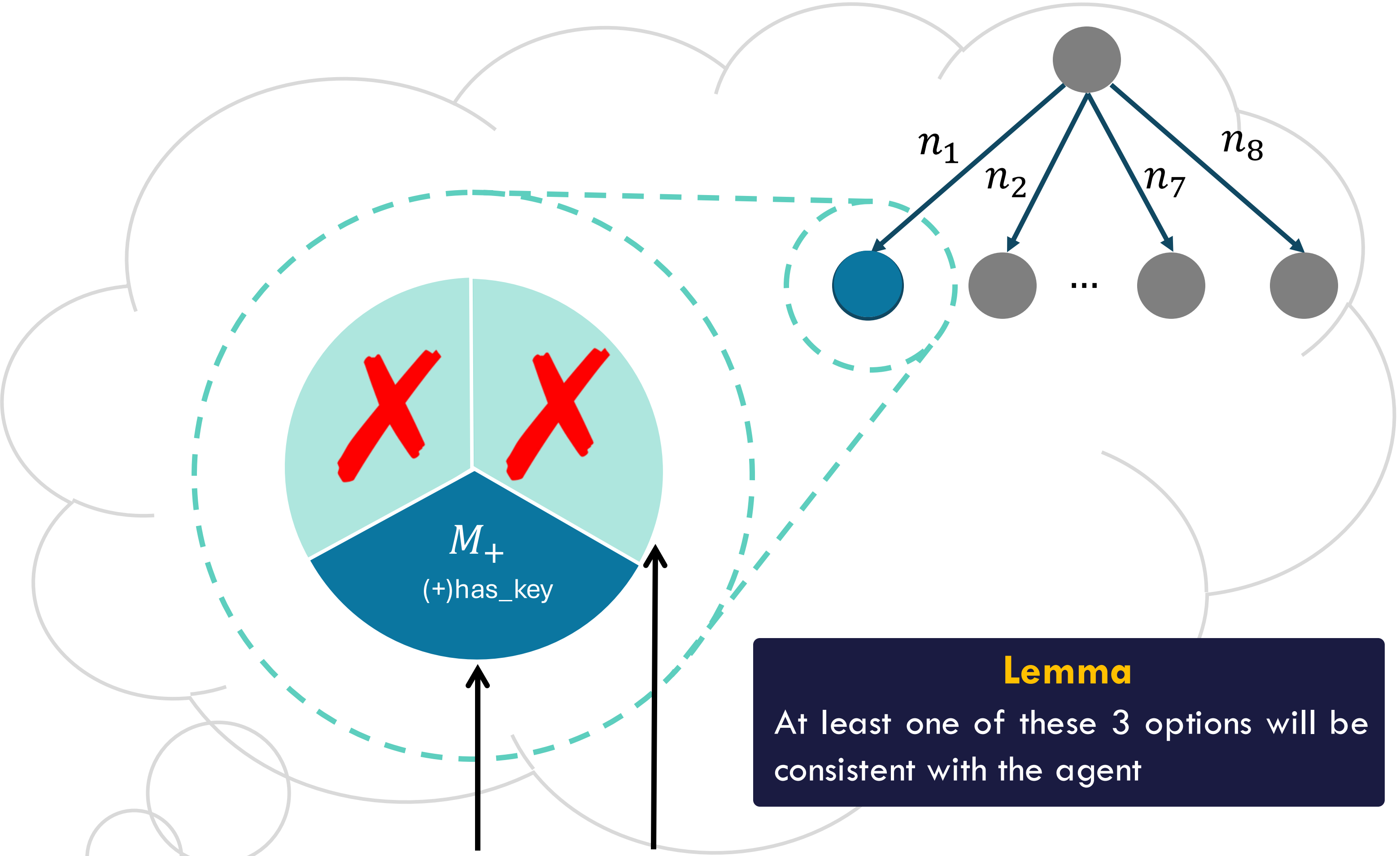
Hierarchical Query Synthesis



Generate a distinguishing query
for these two abstract models

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

Hierarchical Query Synthesis



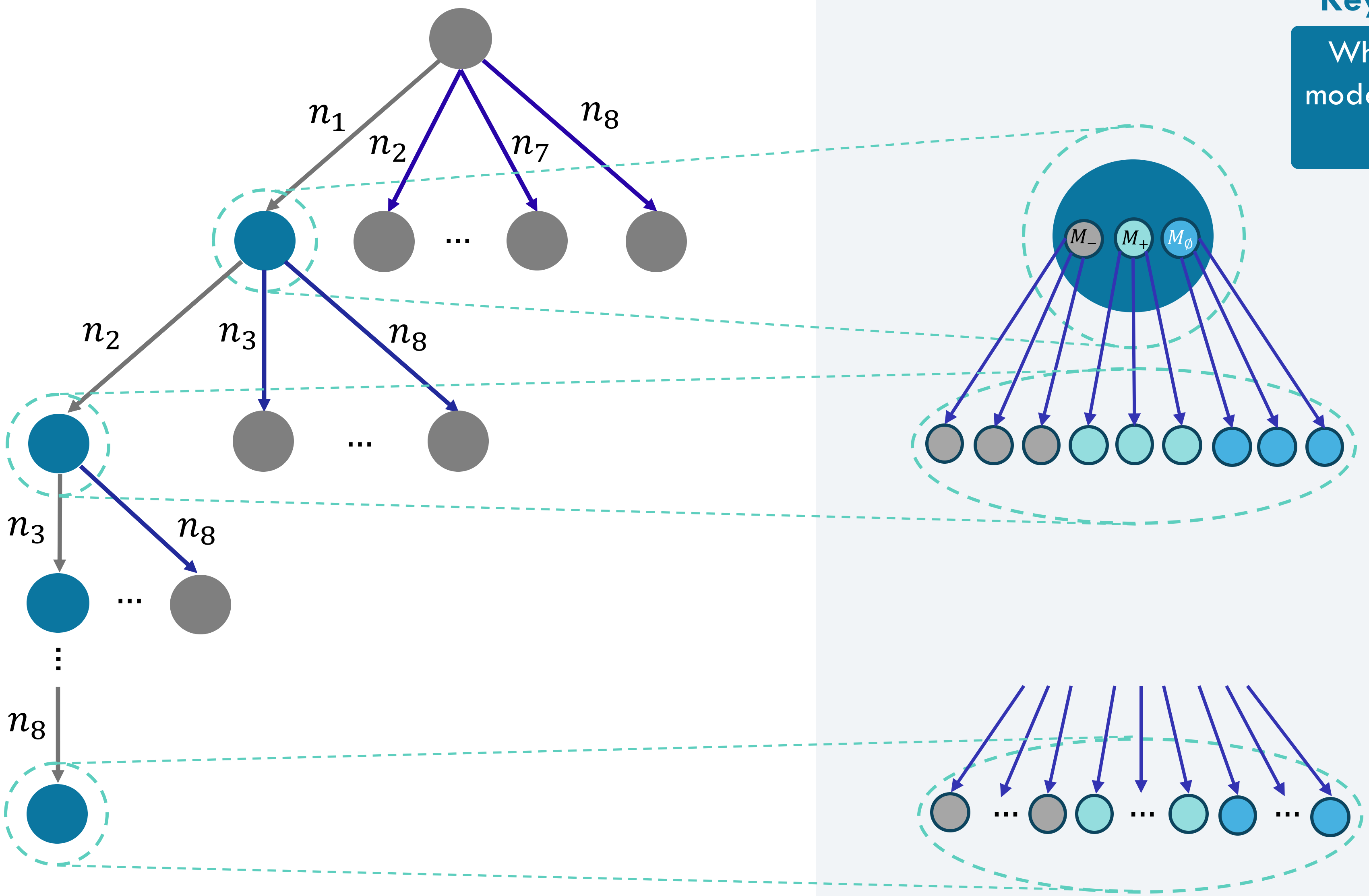
Lemma

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

Reject the abstract model
that is not 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))
```

Hierarchical Query Synthesis



Key feature of the algorithm

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

Active Learning

Deterministic and Stationary Setting

Input

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

Output

- PDDL-like description of each capability.

Assumptions

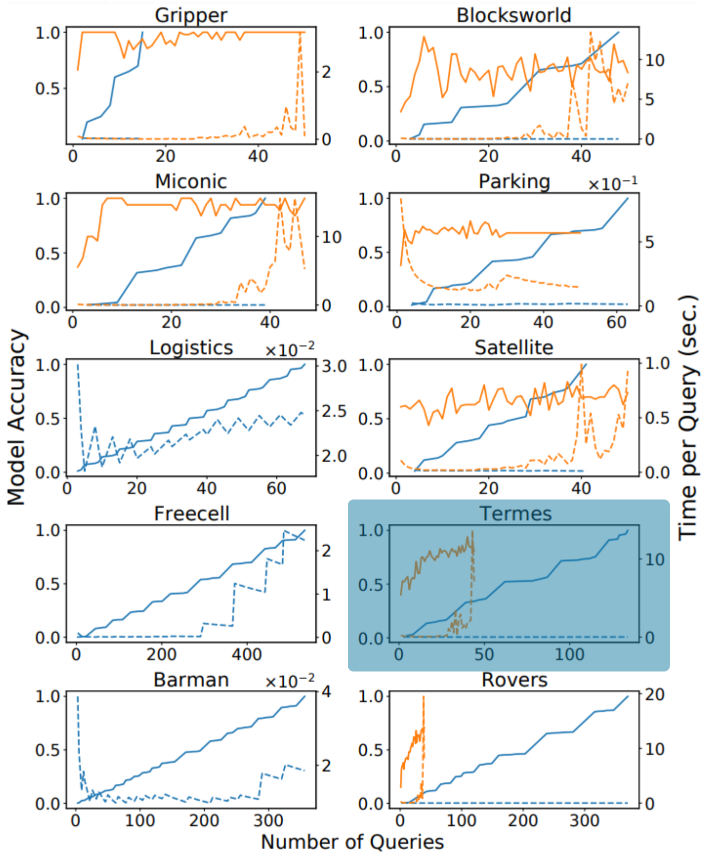
- User's vocabulary matches simulator's vocabulary.
- Black-Box AI provides a list of capabilities.
- Stationary agent model.
- Deterministic environment.
- Fully observable setting.

AAM learns Accurate Model with fewer Queries

- Asses by learning the model and compare with ground truth.
- Baseline†: A passive learner (FAMA) that observes agent behavior

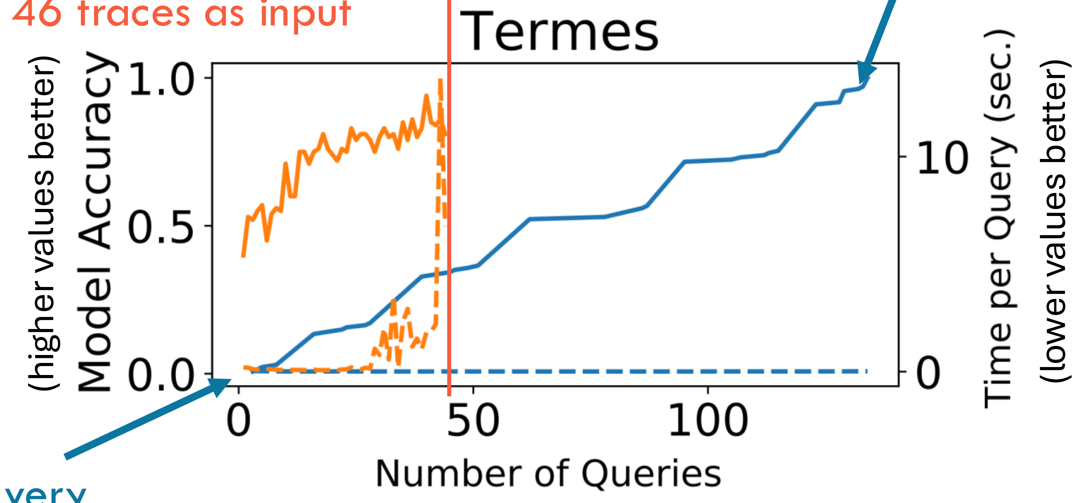
Random deterministic planning agent from IPC

Accuracy: — AAM — FAMA
Time: - - - AAM - - - FAMA



FAMA ran out of memory with 46 traces as input

AAM learned the correct model with 134 queries



AAM takes very less time

AAM learns Accurate Deterministic Models

- Theorem (*termination*) : The algorithm terminates after a finite number of iterations.

- Theorem (*soundness*): The resulting (set of) model(s) is(are) functionally equivalent to the ground truth model.

Autonomous Capability Assessment of Sequential Decision-Making Systems in Stochastic Settings

Pulkit Verma, Rushang Karia, and Siddharth Srivastava

NeurIPS 2023

Stochastic and Stationary Setting

Input

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

Output

- PPDDL-like description of each capability.

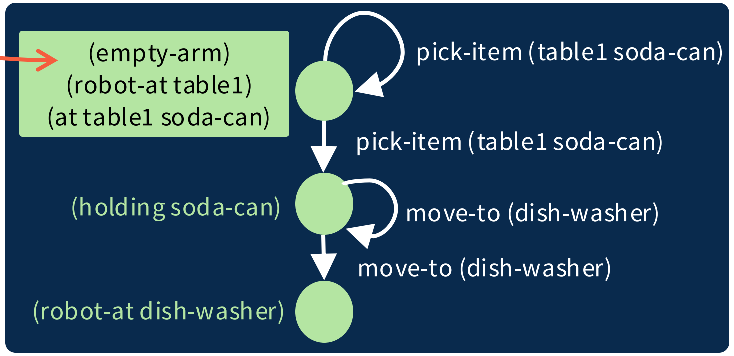
Assumptions

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

Changes for Stochastic Settings

New Queries

Initial State



Policy: Generated Autonomously by Reduction to Non-Deterministic Planning

What happens if you start in the given initial state and follow this partial policy?

Assumptions

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

Changes for Stochastic Settings

Step 1: Learn a Non-Deterministic Model

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

Apply Maximum Likelihood Estimation

on the observed data (query responses)

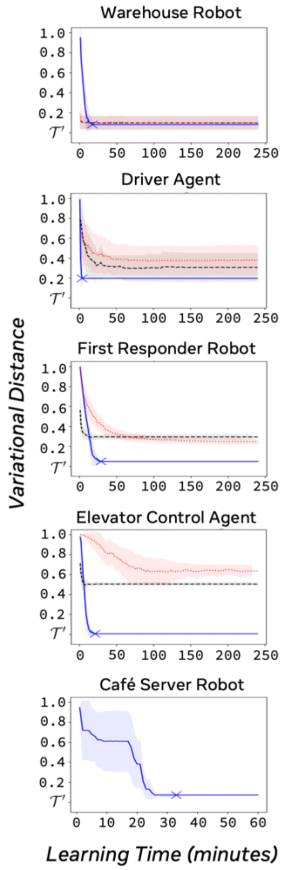
Step 2: Convert to Probabilistic Model

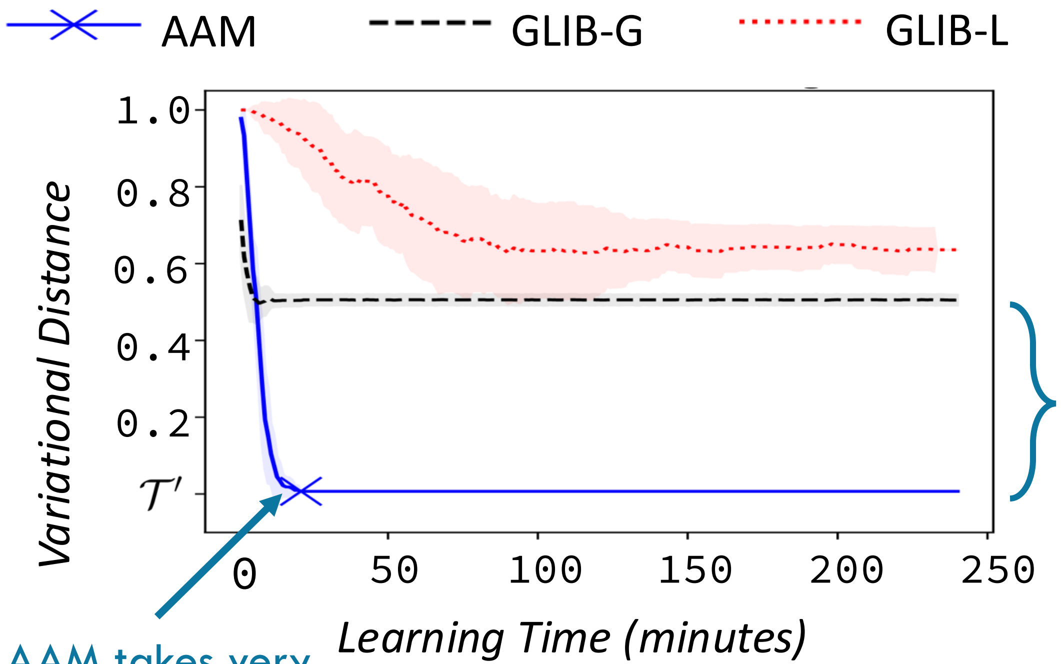
```
(:action open-door
  :parameters (?l1)
  :precondition (and
    (+/-/Ø)(has_key)
    (+/-/Ø)(door_open)
    (+/-/Ø)(door_adjacent ?l1)
    (+/-/Ø)(player_at ?l1))
  :effect (probabilistic
    0.xx (and
      (+/-/Ø)(has_key)
      (+/-/Ø)(door_open)
      (+/-/Ø)(door_adjacent ?l1)
      (+/-/Ø)(player_at ?l1))
    0.yy (and
      (+/-/Ø)(has_key)
      (+/-/Ø)(door_open)
      (+/-/Ø)(door_adjacent ?l1)
      (+/-/Ø)(player_at ?l1)))
```

AAM learns accurate probabilistic models faster

- Baseline: directed exploration approach (GLIB)
- Increase the time taken to learn the model.

Random probabilistic planning agent from IPC





AAM learns a much better model than GLIB

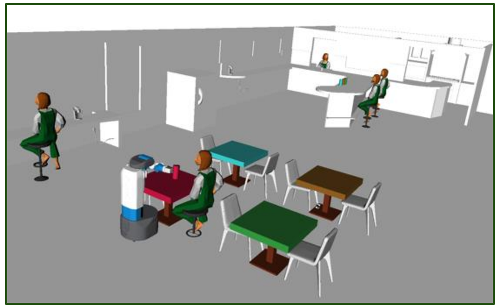
AAM takes very less time

†Chitnis, R.; Silver, T.; Tenenbaum, J.; Kaelbling, L. P.; Lozano-Perez, T. GLIB: Efficient Exploration for Relational MBRL via Goal-Literal Babbling. AAAI 2021.

AAM learns accurate models for Continuous Domains

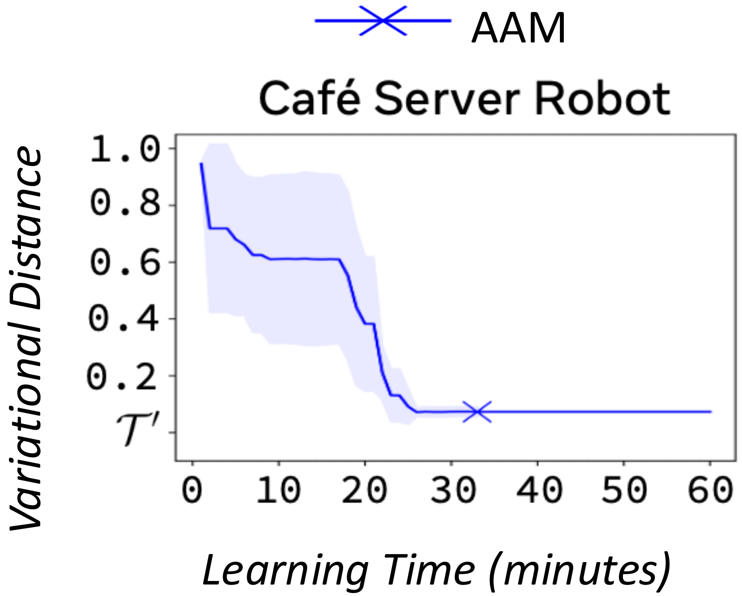
- Use Task and Motion Planning (TMP) to convert actions into motion plans.
- Increase the time taken to learn the model.

Probabilistic planning agent using OpenRave and TMP



	x	y	z	θ	φ	ψ
robot-base	1.0	-3.2	4.7	0.9	1.3	3.1
soda-can1	6.0	-2.8	3.5	8.3	6.7	9.2
⋮					⋮	
table4	-2.1	4.1	1.9	3.7	9.5	4.8

(empty-arm)
(robot-at table1)
(at table1 soda-can)



AAM learns Accurate Probabilistic Models

- Theorem (*soundness and completeness*):
  The intermediate non-deterministic model (after step 1) is sound and complete w.r.t. the ground truth model.

- Theorem (*probabilistic correctness*):
  The resulting probabilistic model is correct w.r.t. the ground truth model.

Coffee Break