IJCAI/2023 MACAO

TECHNION | The Henry and Marilyn Taub Faculty of Computer Science

AAIR Autonomous Agents and Intelligent Robots

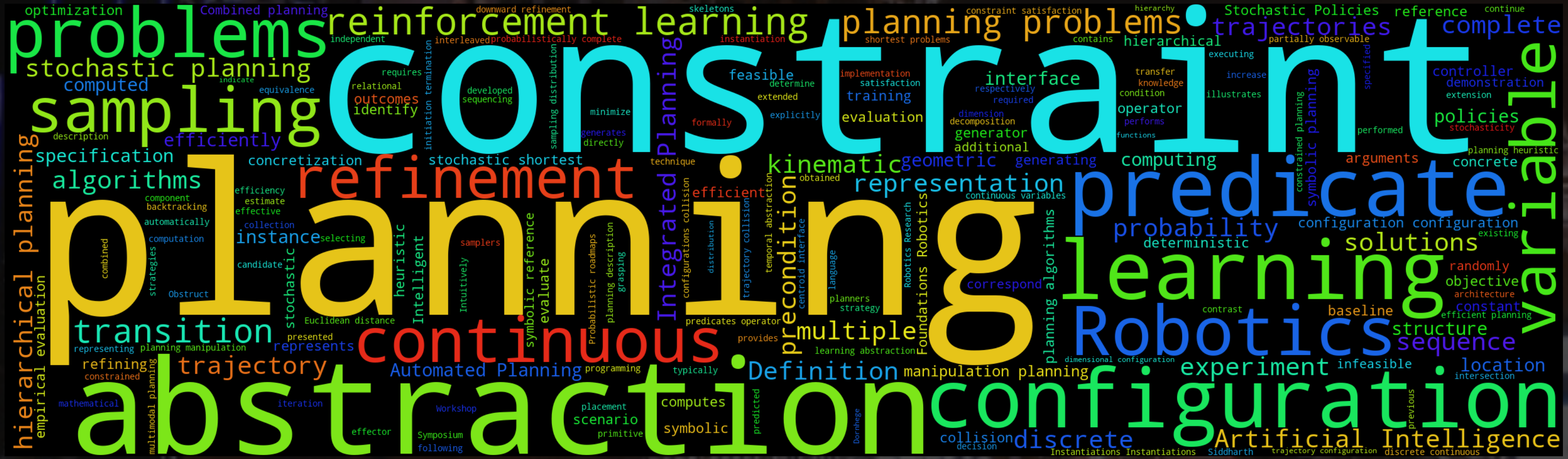
ASU Arizona State University

IJCAI 2023 Tutorial

Integrated Task and Motion Planning

From Foundations to Research Frontiers

19th August, 2023



Naman Shah
Arizona State University

Sarah Keren
Technion

Siddharth Srivastava
Arizona State University


AAIR
Autonomous Agents
and Intelligent Robots
ASU Arizona State University
Next-level problem solvers

Outline

1. Background: Why Task and Motion Planning?
2. Abstraction as a Foundation for TMP
3. Modern Abstraction-Based Approaches
4. Research Frontier: Neuro-Symbolic Abstraction Learning for TMP

Fundamental Problem: Long-Horizon Planning

S, set of states

A, set of actions

$T: S \times A \rightarrow \mu S$, action transition function

$R: S \times A \rightarrow \mathbb{R}$, costs and utility of states, actions (can express goals and some forms of preferences*)

Automated Planning/Sequential Decision Making:

What should the robot do to maximize R (achieve goal) over multiple time steps?

Task and Motion Planning: Longer Horizons, Uncertainty

Formulation as SDM problems: $S = ?\ A = ?$

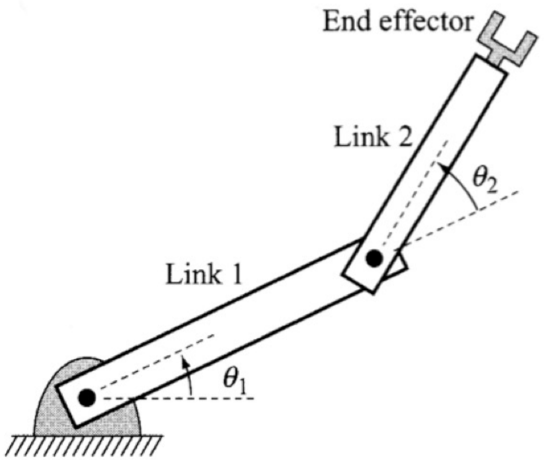
[Srivastava, Gupta, Zilberstein, Abbeel and Russell, AAAI 2015]

[Shah, Vasudevan, Kumar, Srivastava, ICRA 2020]

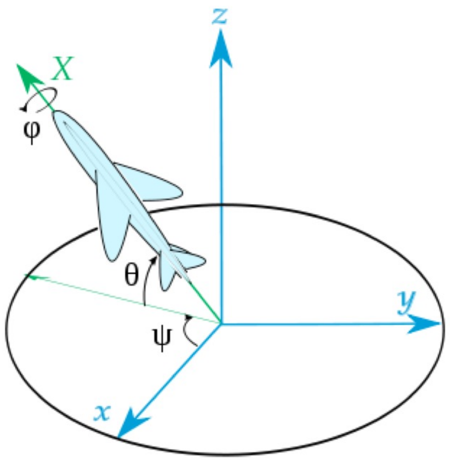
Configuration Space (C-Space): State Space for a Robot in an Environment

Configuration: A complete specification of the position of every point in the system

C-Space: Space of all possible system configurations



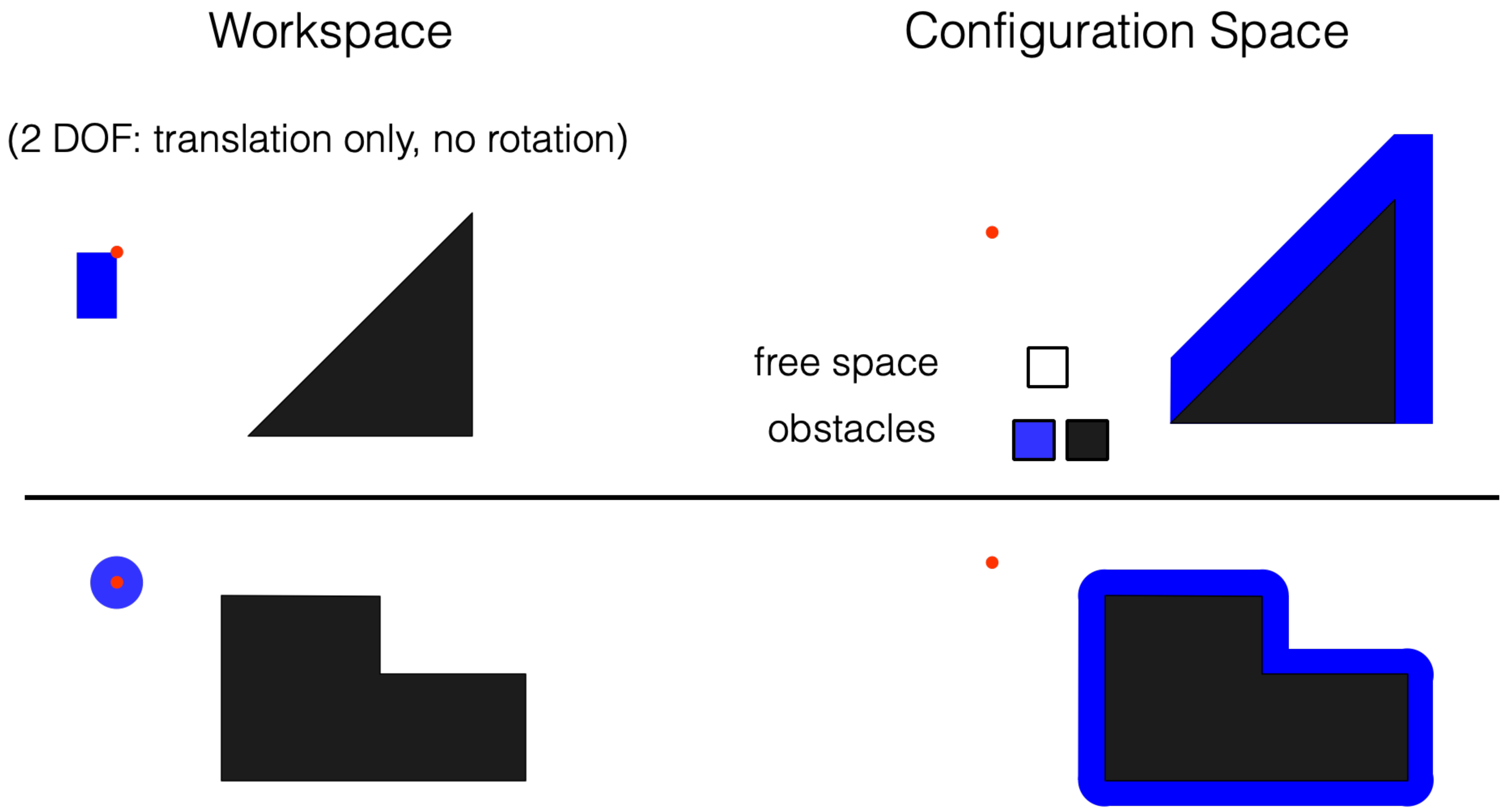
2 - Dim



6 - Dim

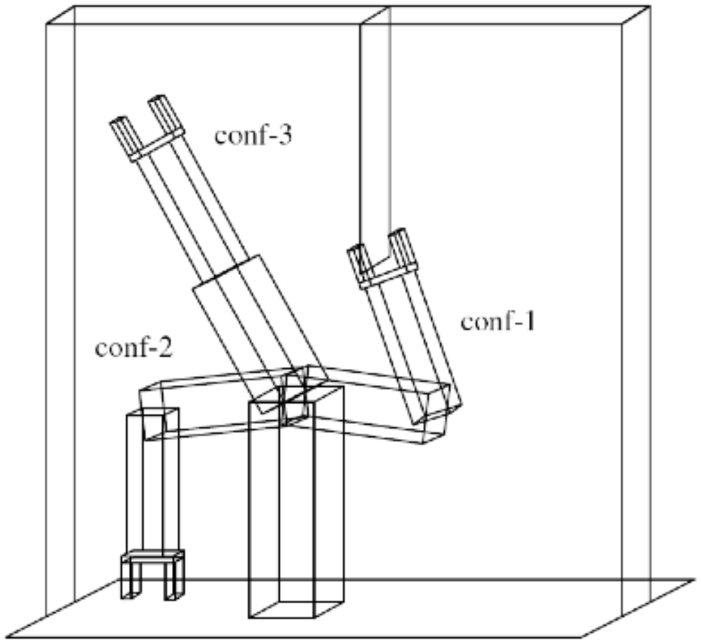
20 - Dim

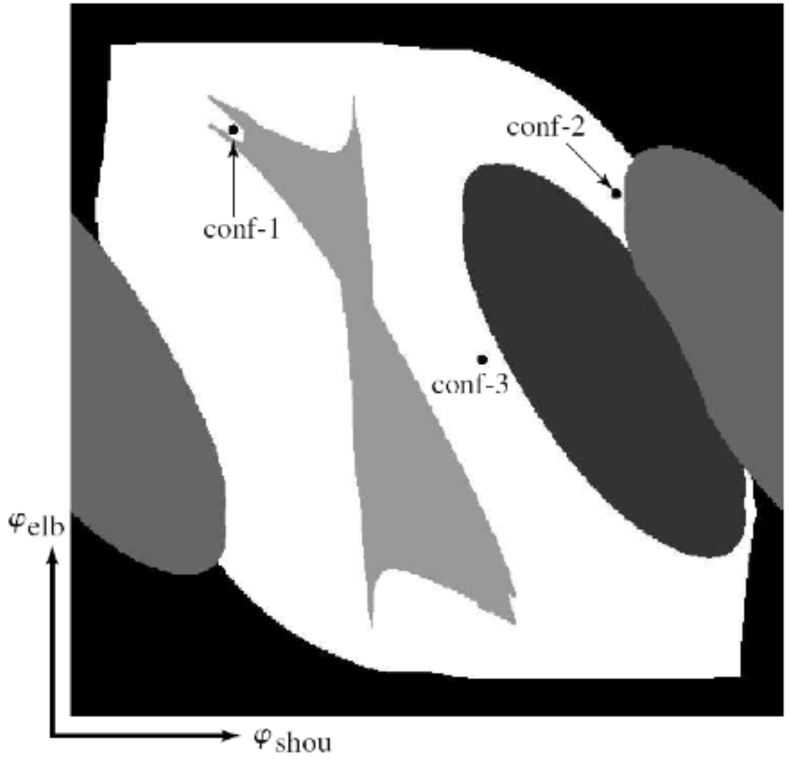
Configuration Space (C-Space)



[Source: CS287, Pieter Abbeel, UC Berkeley]

Configuration Space (C-Space)



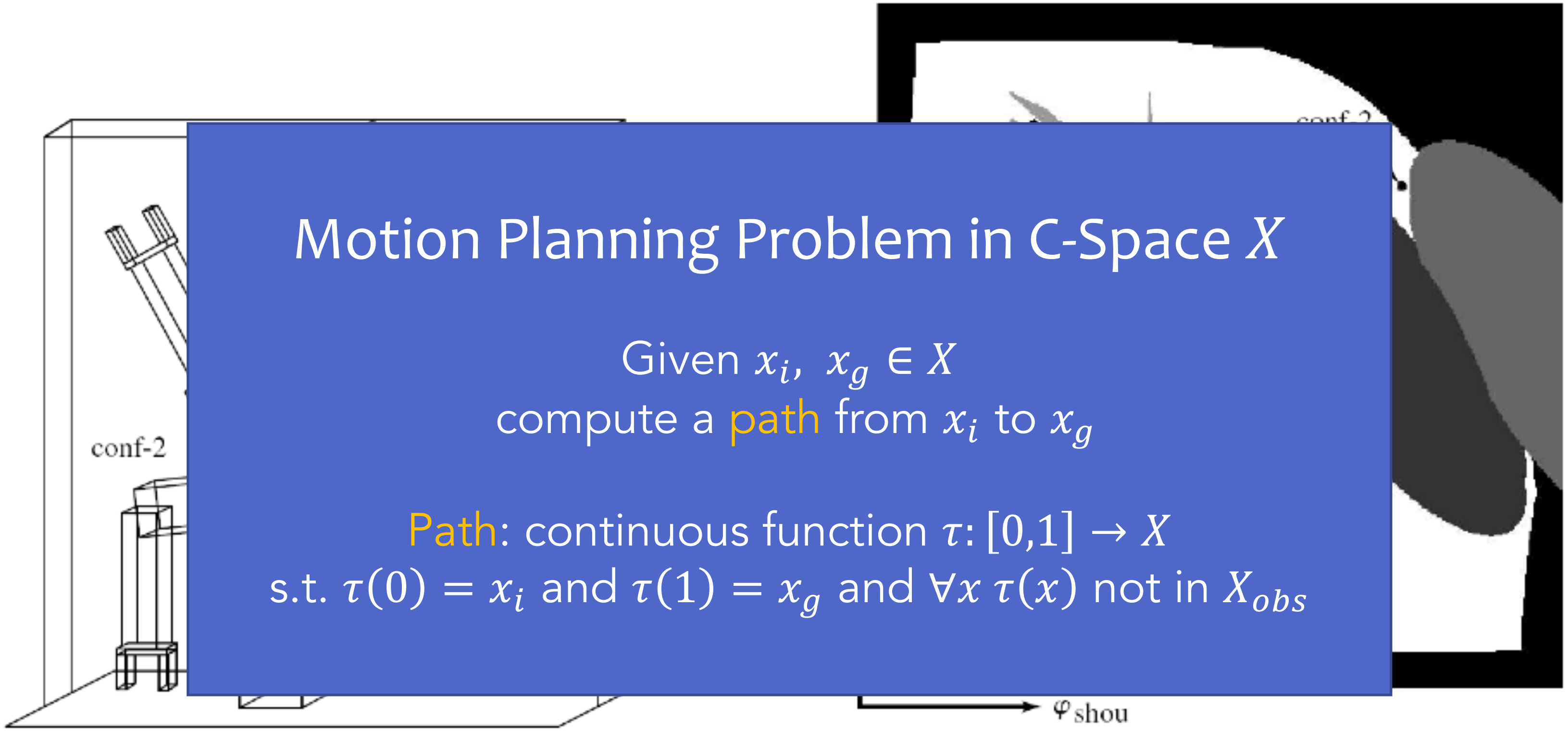


Obstacles reduce free space

Computing obstacle boundaries in C-Space provably exponential

[Source: CS287, Pieter Abbeel, UC Berkeley]

Configuration Space (C-Space)



Motion Planning Problem in C-Space X

Given $x_i,\ x_g \in X$
compute a path from x_i to x_g

Path: continuous function τ: $[0,1] \rightarrow X$
s.t. $\tau(0) = x_i$ and $\tau(1) = x_g$ and $\forall x\ \tau(x)$ not in X_{obs}

Obstacles reduce free space

Computing obstacle boundaries in C-Space provably exponential

[Source: CS287, Pieter Abbeel, UC Berkeley]

Examples

PR2: Two 8 DoF arms + 1 DoF height + 3 DoF base

YuMi: Two 7-DoF arms

Formulation as Motion Planning Problems:

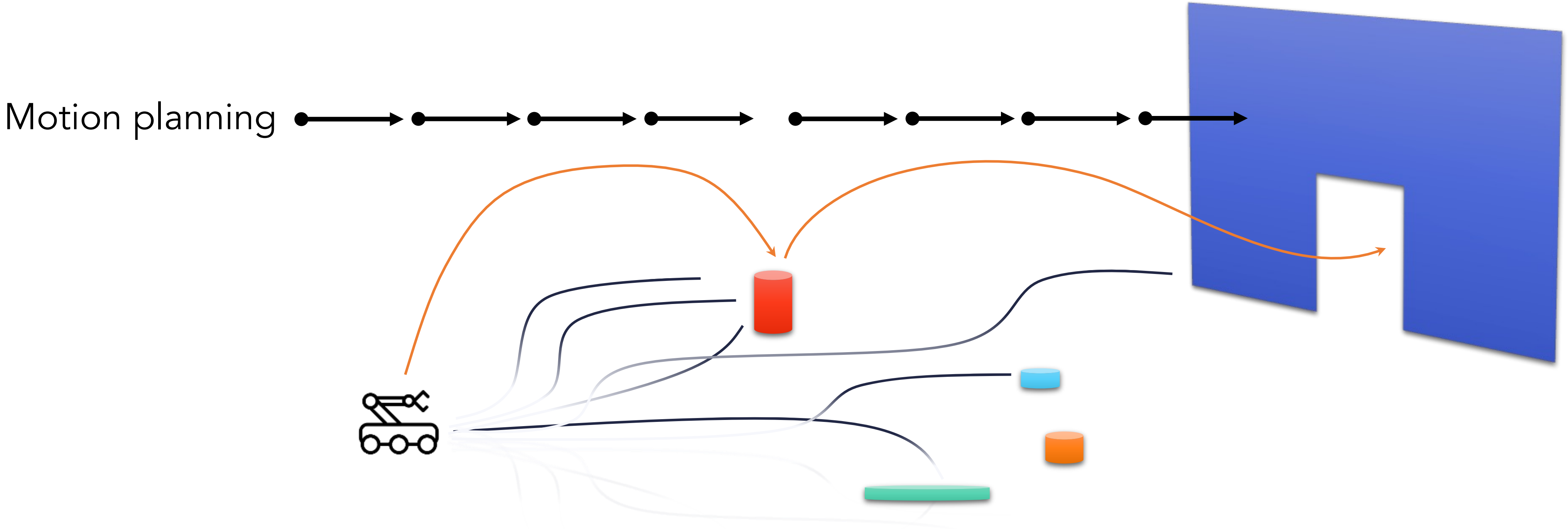
State (Config) Space X =?

x_i, x_g =?

What happens to the C-space when the robot picks up a plank?

Pure Motion Planning is Not Enough!

- Motion planning – which path (of waypoints) should the robot take?
    - But which motion planning problem should it solve? different pickups ⇒ different c-spaces
    - Where would motion planning goals in each C-space come from?
    - Clearly, motion planning is not enough



Would Pure Task-Planning Do the Trick?

Can a "higher-level" planner help us compute the strategy?

Then we could refine each action in the plan into a motion plan

Higher-level planning is typically done over states described using features, or properties

E.g., #clothes on table

IsHolding(robot, basket)

...

+

Actions describing how and when robot can change these properties

Would Pure Task-Planning Do the Trick?

Example of a high-level action:

```
(:action pickup
  :parameters (?obj ?gripper)
  :precondition (and (empty ?gripper)
                     (ontable ?obj))
  :effect (and (not (empty ?gripper))
               (not (ontable ?obj)
               (in ?obj, ?gripper)))))
```

SDM problem: which sequence of actions will lead to the goal?

The Shakey Robot

ARTIFICIAL INTELLIGENCE 189

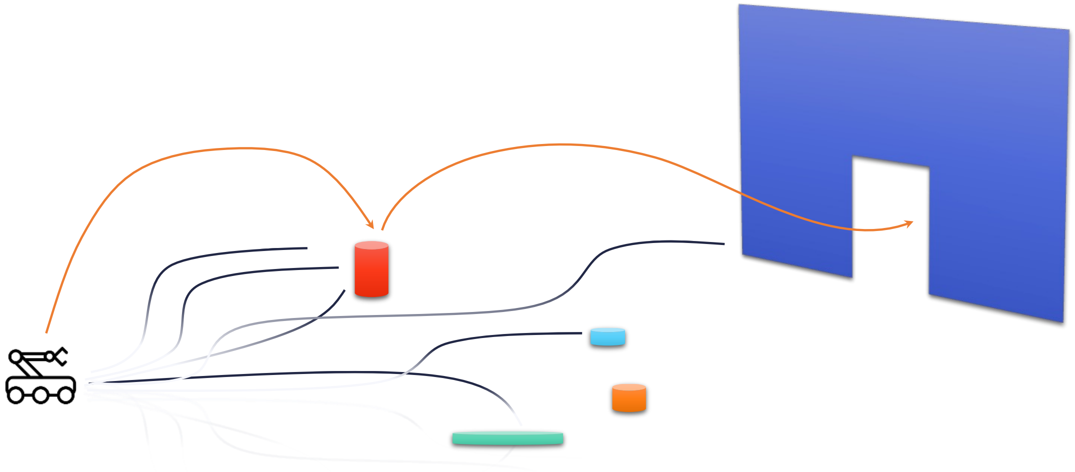
STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving[1]

Richard E. Fikes
Nils J. Nilsson
Stanford Research Institute, Menlo Park, California

Recommended by B. Raphael

Presented at the 2nd IJCAI, Imperial College, London, England, September 1–3, 1971.

How do These High-Level Actions Connect to the C-Space?



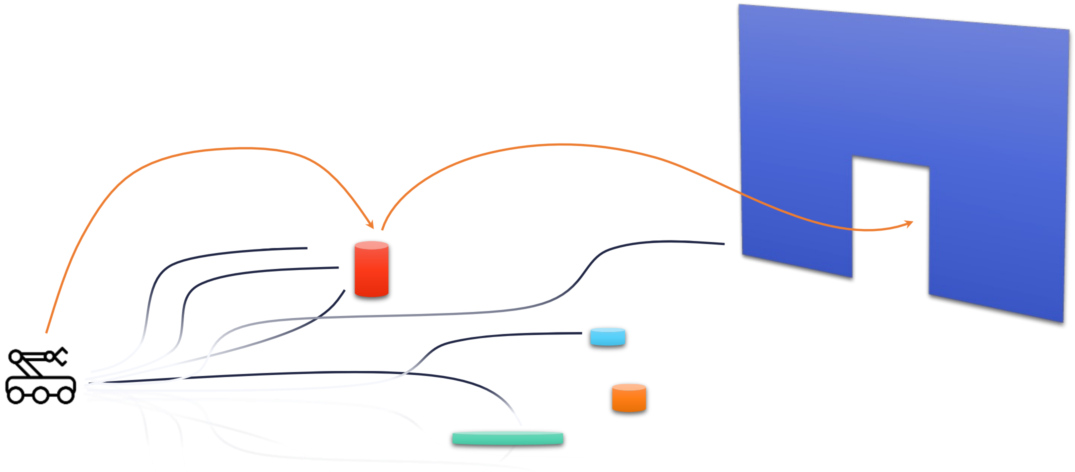
GoTo(l)
Pickup(x)
PutDown(x)

Temporal Abstraction
≡ Abstract Actions,
Macros,
Options…

At(x, l)
InGripper(x)
AtDestination(x)

State
Abstraction

How do These High-Level Actions Connect to the C-Space?



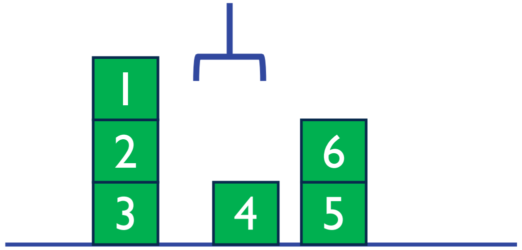
GoTo(l)
Pickup(x)
PutDown(x)

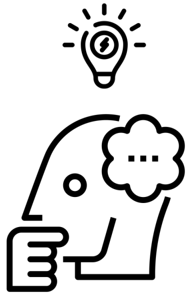
Temporal Abstraction
≡ Abstract Actions,
Macros,
Options…

At(x, l)
InGripper(x)
AtDestination(x)

State
Abstraction

Sometimes Intuitive Abstractions are … Perfect





Pickup(x)
PutDown(x)

OnTable(3)
On(1,2)
…

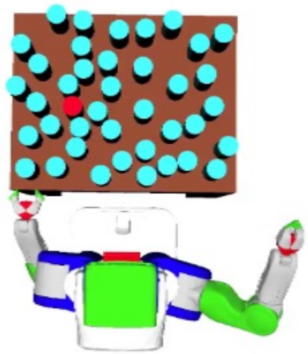
```
(:action pickup
  :parameters (?obj ?gripper)
  :precondition (and (empty ?gripper)
                     (ontable ?obj) (clear ?obj))
  :effect (and (not (empty ?gripper))
               (not (ontable ?obj))
               (in ?obj, ?gripper) (not (clear ?obj)) ))))
```

But Human Intuition has its Limits

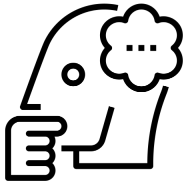
Pickup the red can!

Can pickup only from the side

Obstructions depend on choice of movement trajectory



?!



```
(:action pickup
  :parameters (?obj ?gripper)
  :precondition (and (empty ?gripper)
                     (ontable ?obj) (clear ?obj)
  :effect (and (not (empty ?gripper))
               (not (ontable ?obj)
               (in ?obj, ?gripper) (not (clear ?obj)) ))))
```

Prevailing Abstraction

Pickup(x)
PutDown(x)

Abstract Actions

OnTable(3)
~~On(1,2)~~
OnTable(2)
OnTable(1)
…

Abstract State

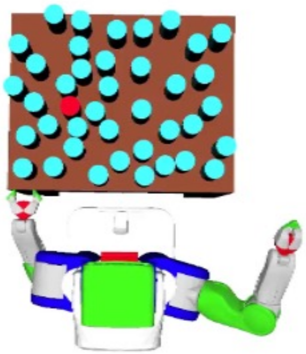
[Srivastava et al., ICRA 2014; Srivastava et al., AAAI 2016]

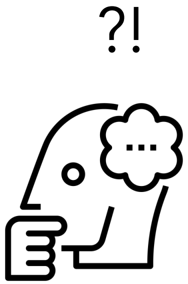
But Human Intuition has its Limits

Pickup the red can!

Can pickup only from the side

Obstructions depend on choice of movement trajectory





Prevailing Abstraction

Pickup(x)
PutDown(x)

Abstract Actions

OnTable(3)
~~On(1,2)~~
OnTable(2)
OnTable(1)
…

Abstract State

Abstract model *thinks* this is a trivial problem

Solutions from abstract model: Mostly infeasible

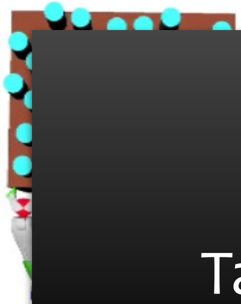
[Srivastava et al., ICRA 2014; Srivastava et al., AAAI 2016]

But Human Intuition has its Limits

Pickup the red can!

Can pickup only from the side

Obstructions depend on choice of movement trajectory



Task Planning is Not Enough!

Task Planning followed by motion planning is also not enough!

Prevailing Abstraction

Pickup(x)
own(x)

Abstract Actions

able(3)
~~,2)~~
able(2)
able(1)

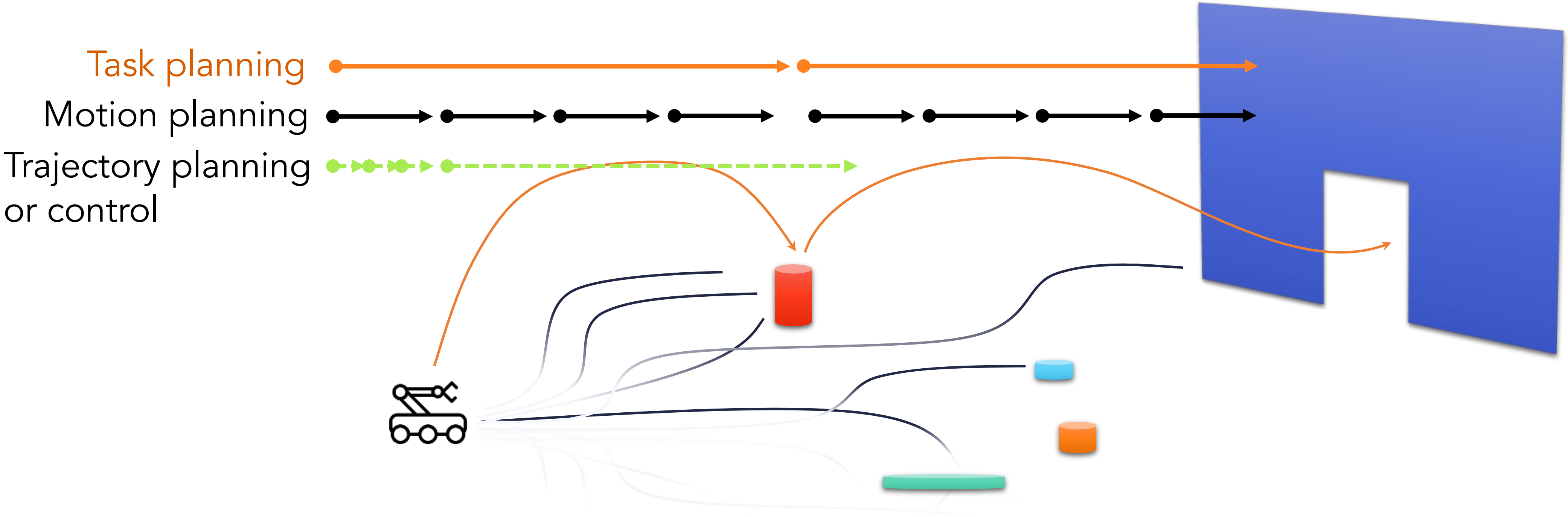
Abstract State

[Srivastava et al., ICRA 2014; Srivastava et al., AAAI 2016]

Summary: We Need to Integrate Task and Motion Planning!

- Task planning – given a task planning problem, computes the high-level action the robot should perform at each step
  - But that action may have no feasible motion plan (recall: cluttered table)
- Motion planning – given a motion planning problem, computes the path that the robot should take
  - But which motion planning problem should it solve?

(Trajectory planning – selects the control inputs that should go to the robot's motors)



Summary: We Need to Integrate Task and Motion Planning!

- Task planning – given a task planning problem, computes the high-level action the robot should perform at each step
  - But that action may have no feasible motion plan (recall: cluttered table)
- Motion planning – given a motion planning problem, computes the path that the robot should take
  - But which motion planning problem should it solve?
- (Trajectory planning – selects the control inputs that should go to the robot's motors?)

Technical Problems That Characterize Integrated Task and Motion Planning

High-level (abstract) models are imprecise! They scale to long horizons at the expense of low-level constraints

➢ Which HL action will have a feasible motion plan at a point in time?

Each HL action (e.g., pickup) defines uncountably infinite Motion Planning Problems!

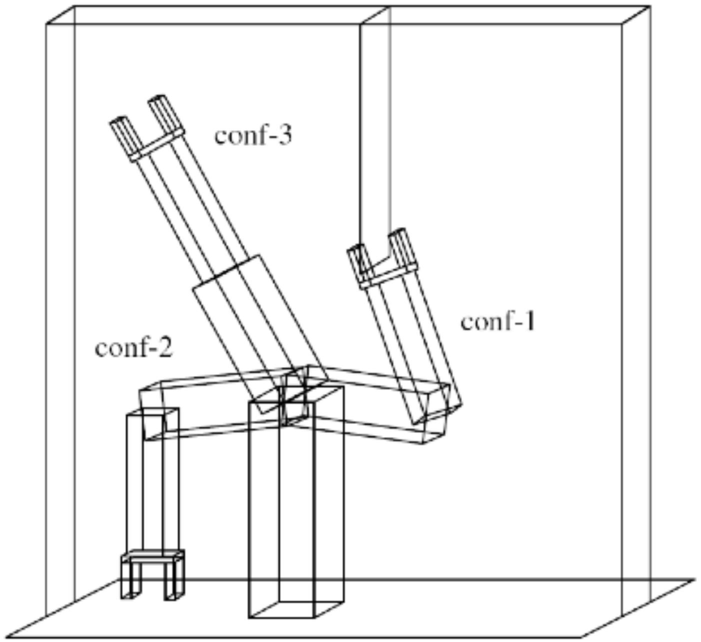
➢ Which MP should be solved?

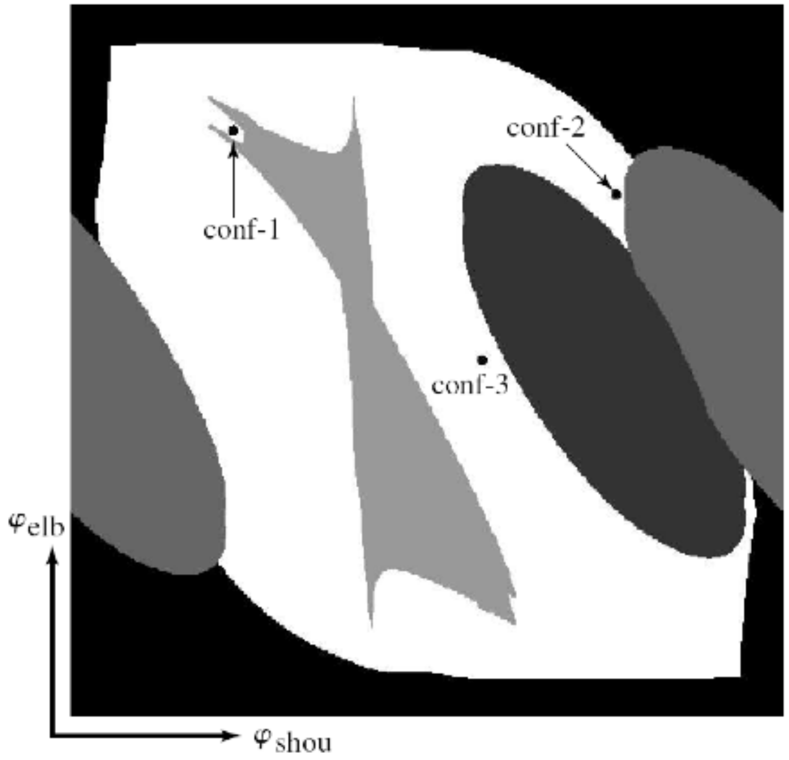
Formalized later in the tutorial

Outline

1. Background: Why Task and Motion Planning?
   - Task Planning is Not Enough
   - Motion Planning is Not Enough
   - Foundations of Motion Planning
2. Abstraction as a Foundation for TMP
3. Abstraction-based Approaches
4. Research Frontier: Neuro-Symbolic Abstraction Learning for TMP

Recall: Configuration Space (C-Space)





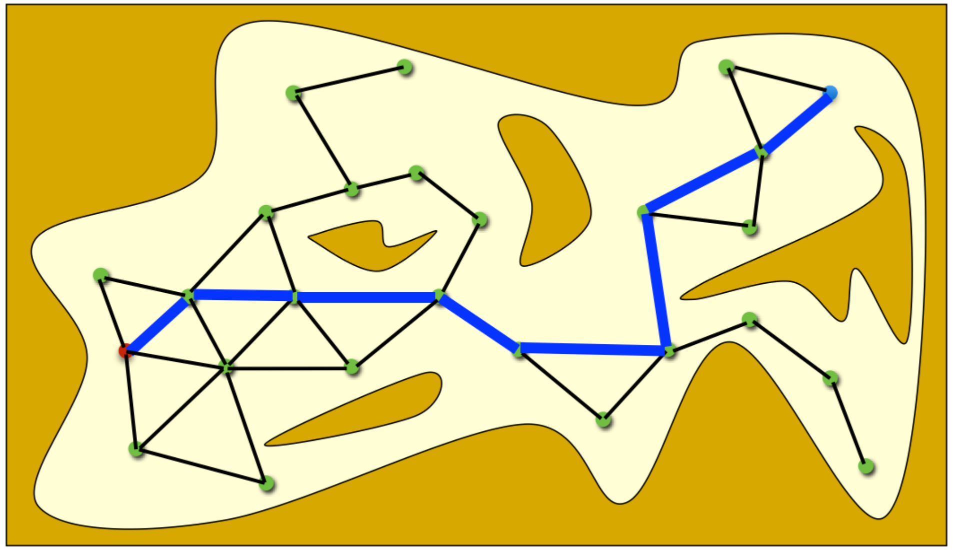
Obstacles reduce free space

Computing obstacle boundaries in C-Space provably exponential

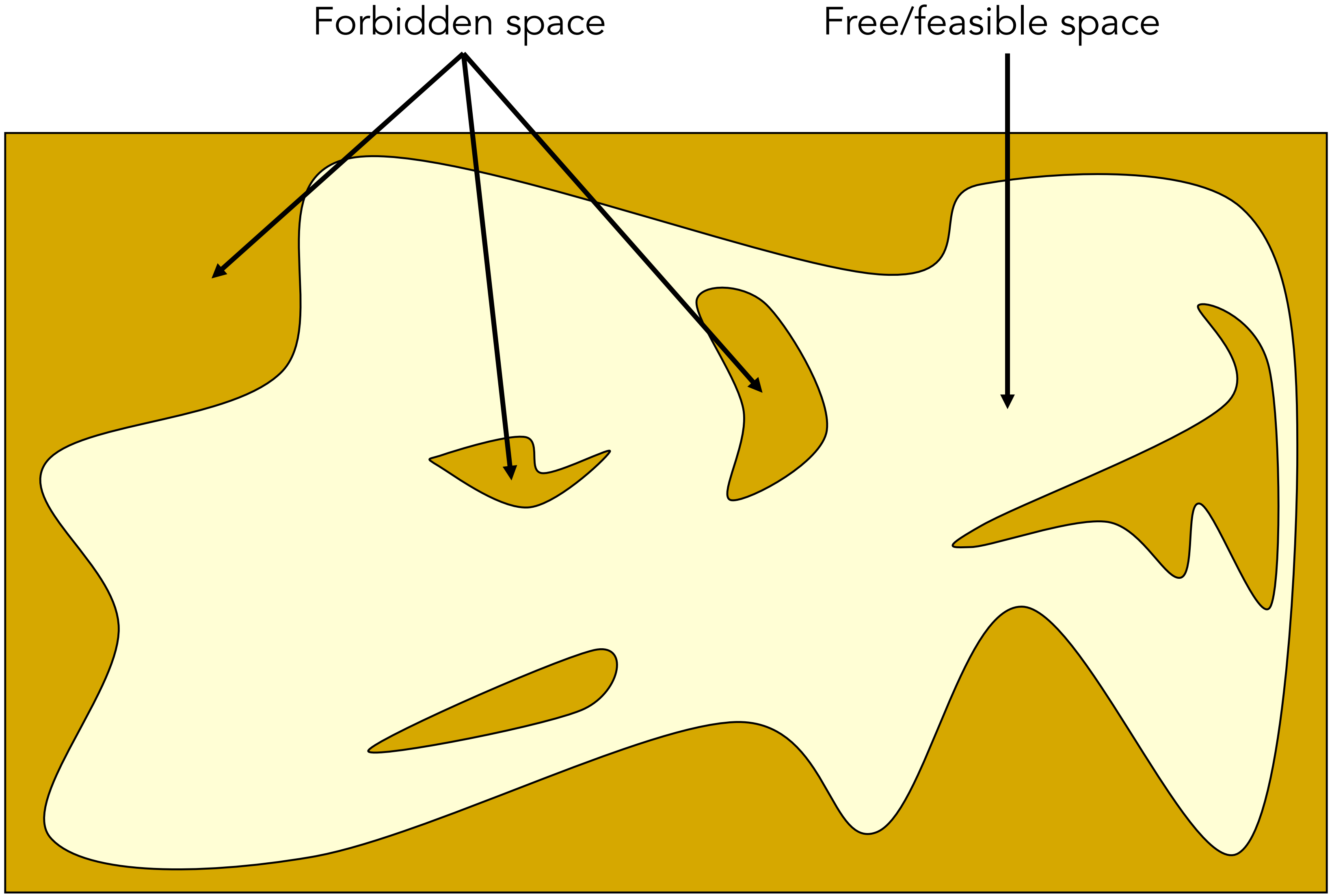
[Source: CS287, Pieter Abbeel, UC Berkeley]

Sampling-based Motion Planning

- Sampling-based solutions sample the C-Space instead of explicitly computing it
  - Probabilistic Roadmap (PRM)
  - Rapidly-exploring random tree (RRT)
- Simply need to know if the robot is in collision (workspace query)



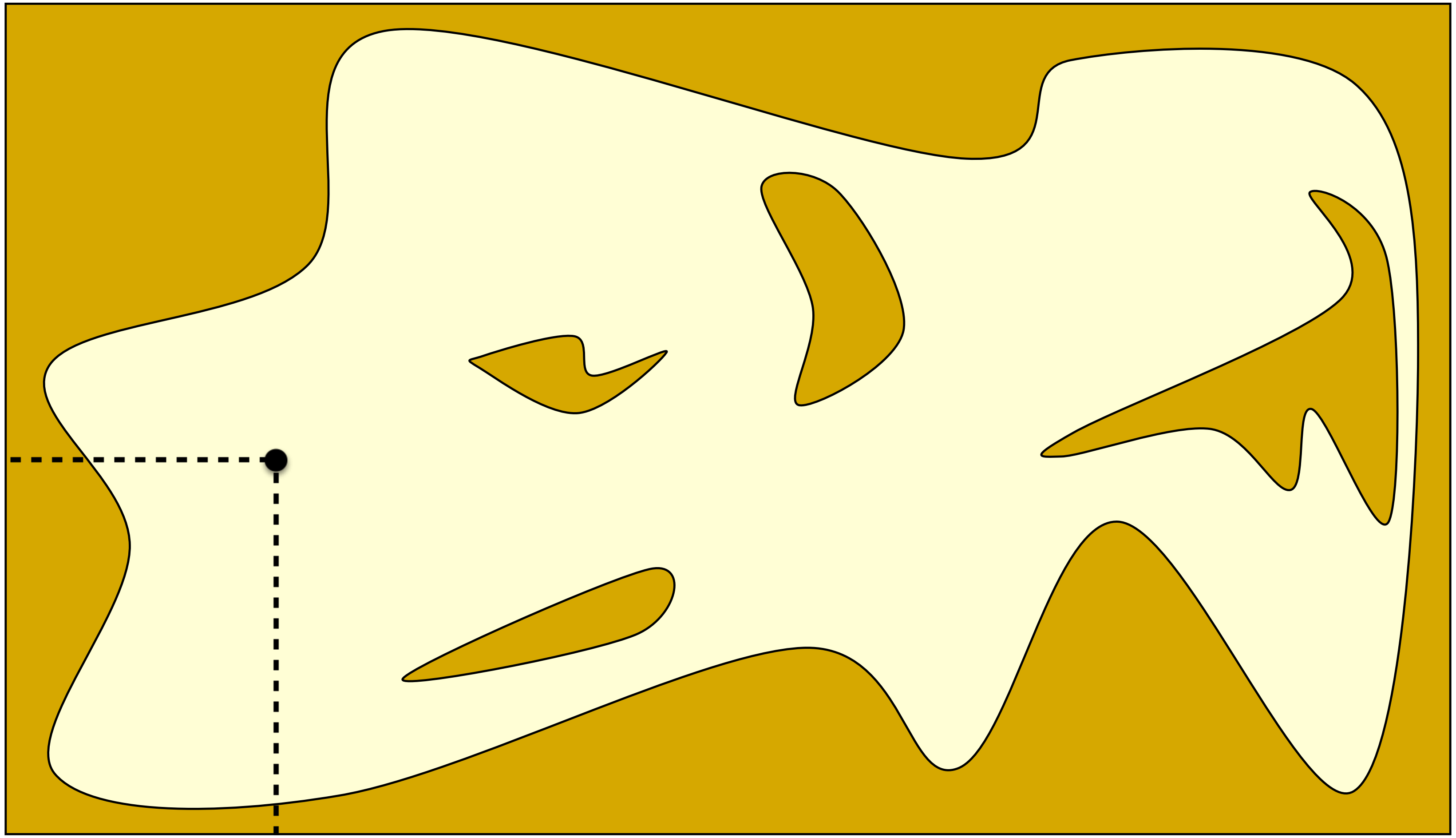
Probabilistic Roadmaps (PRM)



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Roadmaps (PRM)

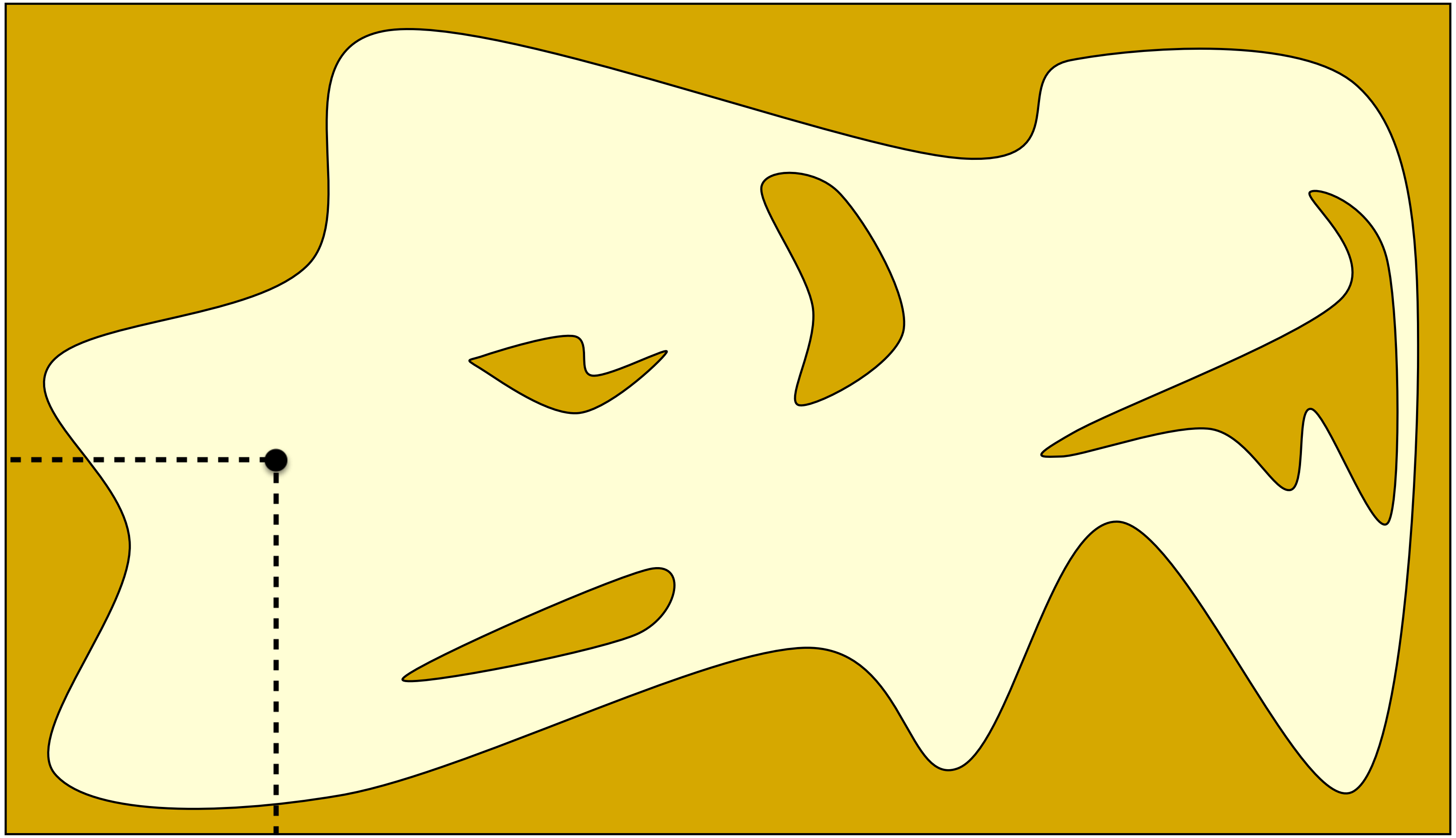
Configurations are sampled by picking coordinates at random



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Roadmaps (PRM)

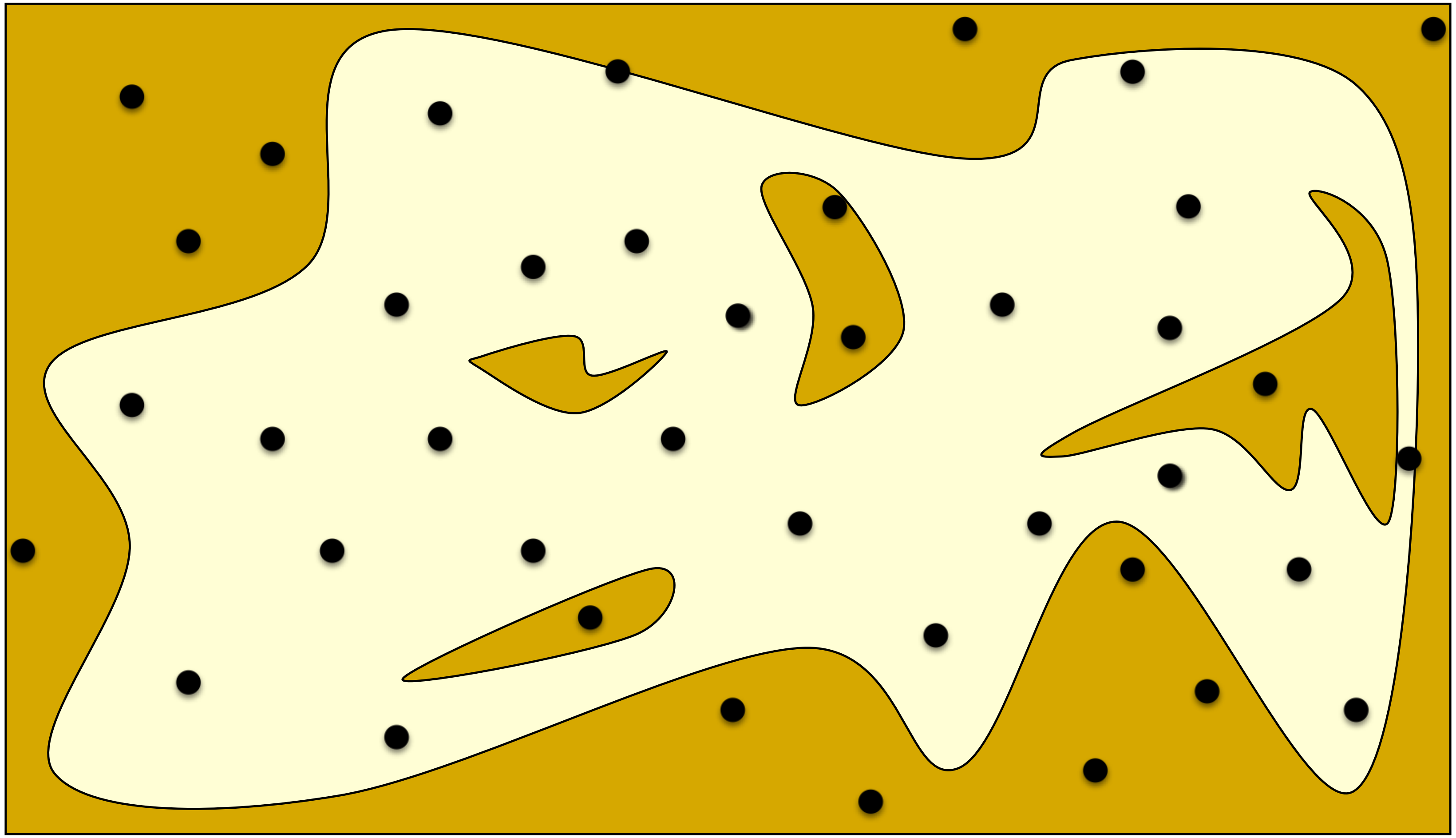
Configurations are sampled by picking coordinates at random



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Roadmaps (PRM)

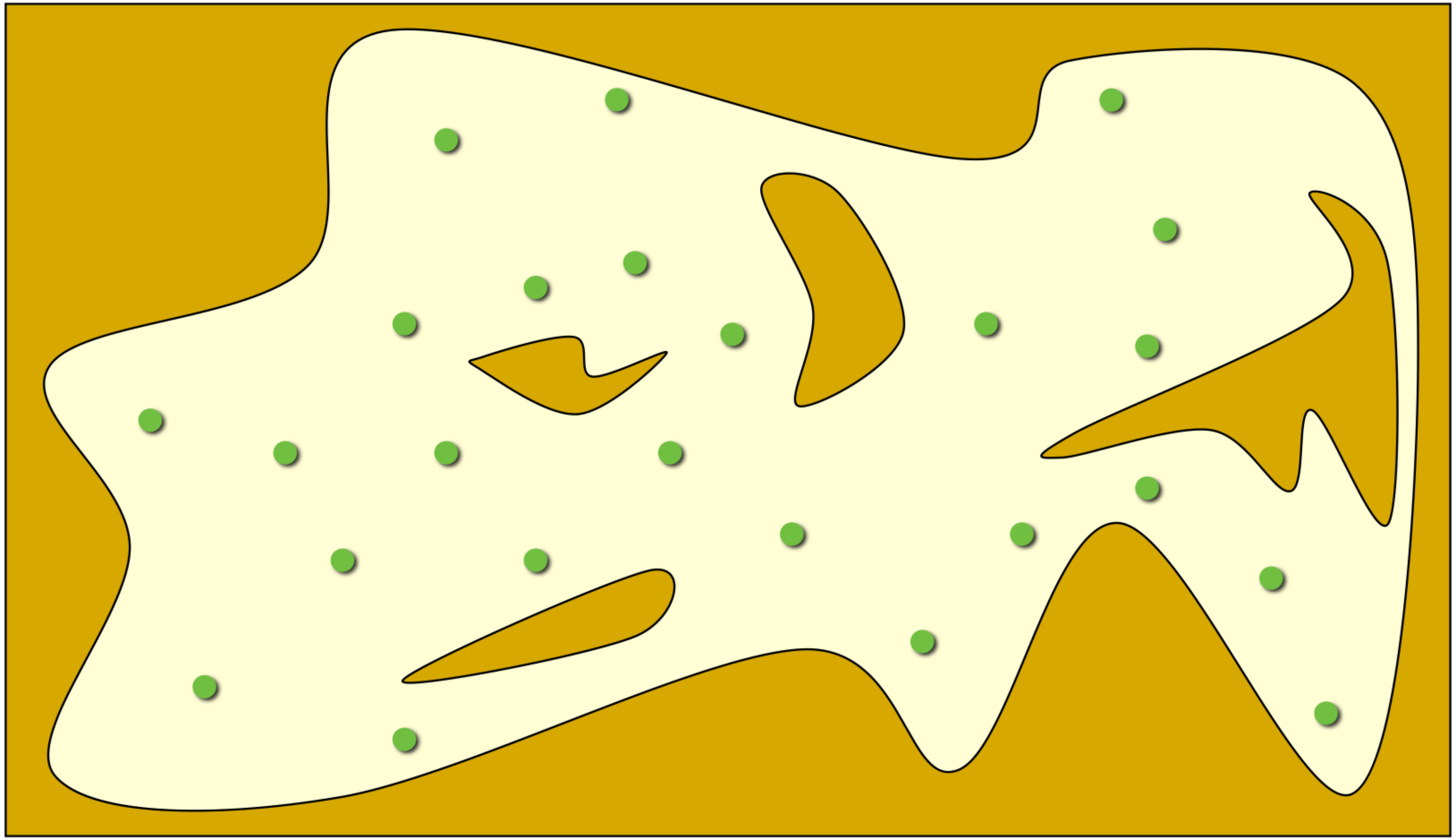
Configurations are sampled by picking coordinates at random



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Roadmaps (PRM)

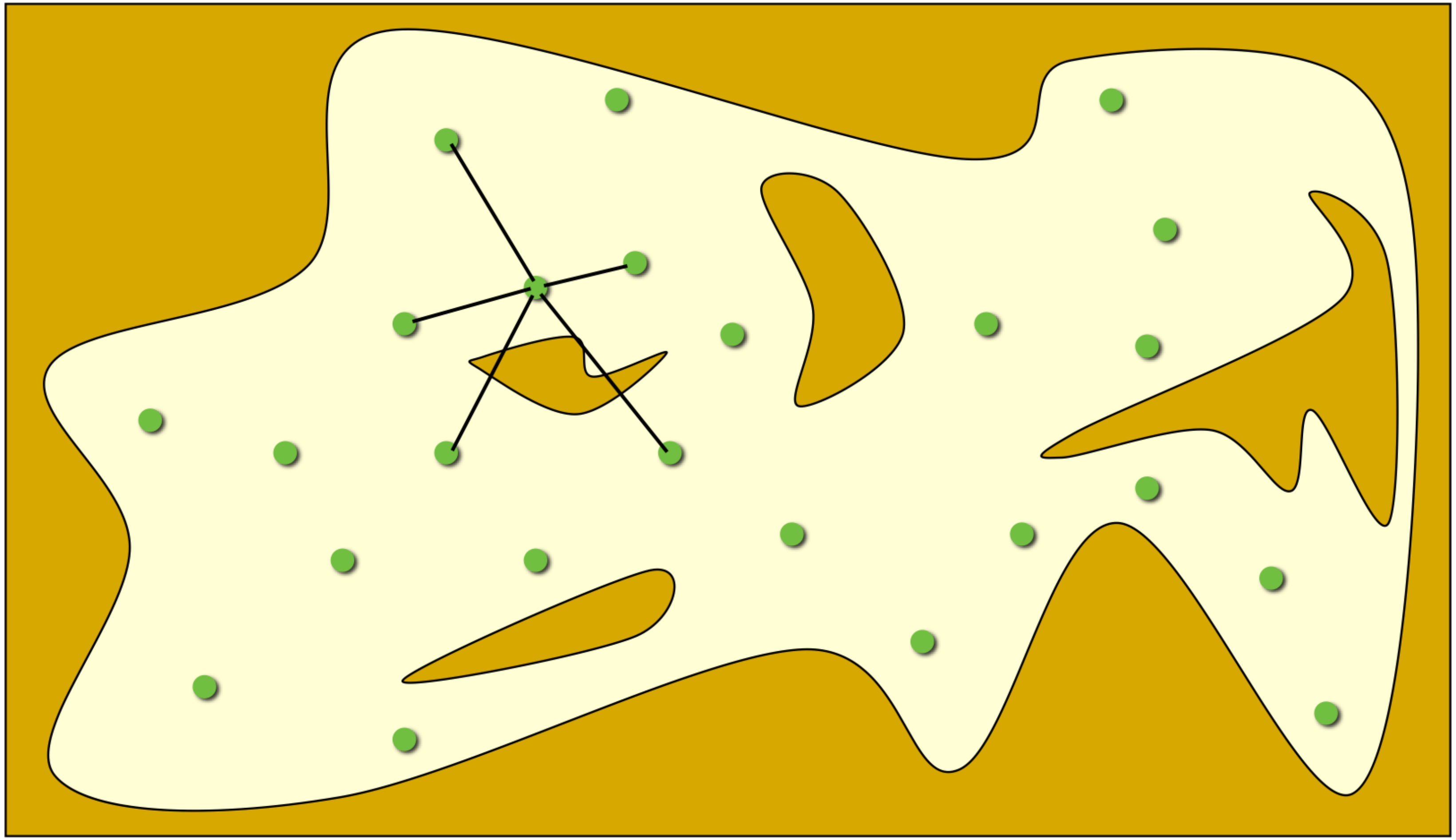
Sampled configurations are tested for collisions



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Roadmaps (PRM)

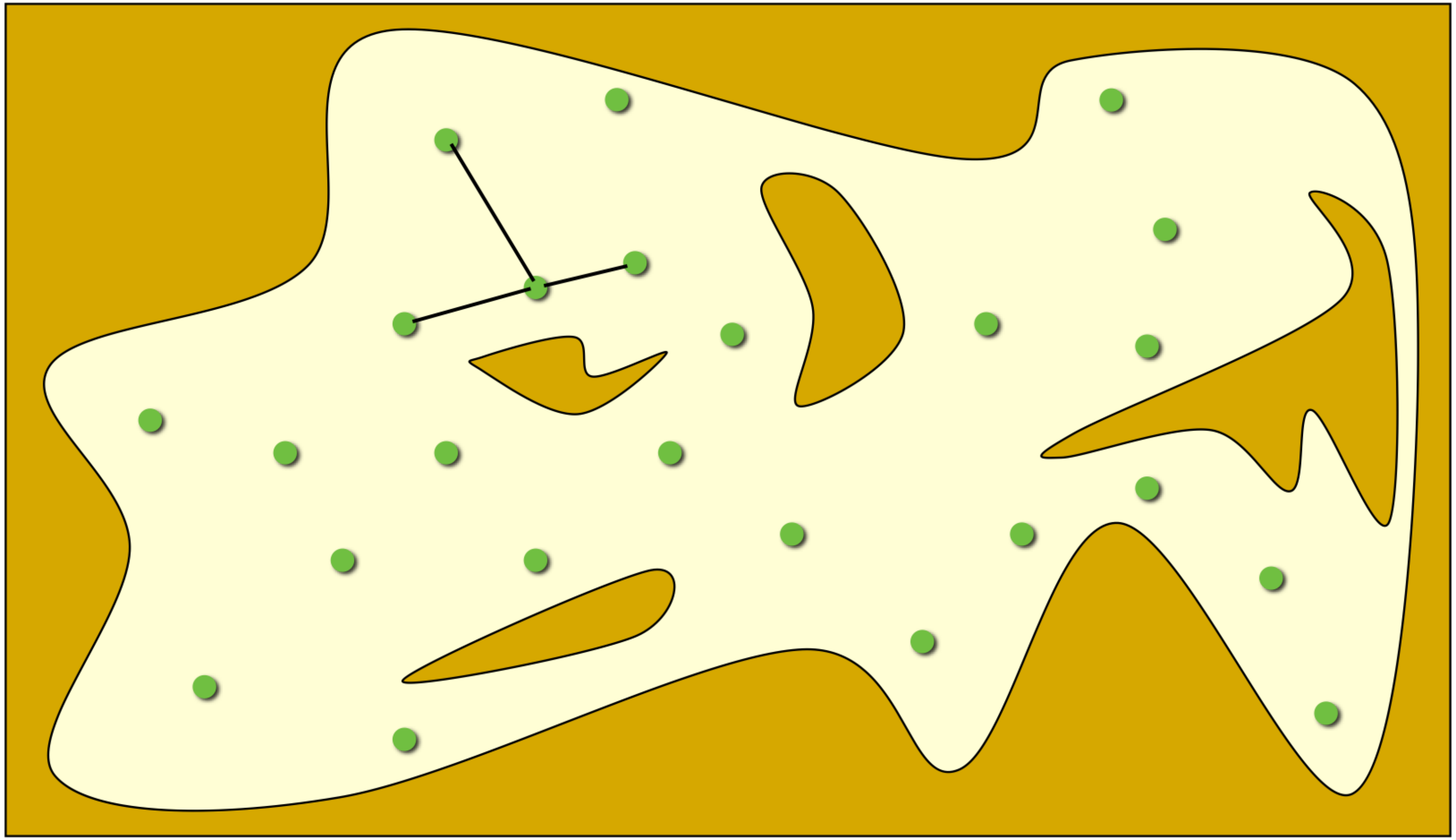
Each milestone is linked to its nearest neighbors by straight paths



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Roadmaps (PRM)

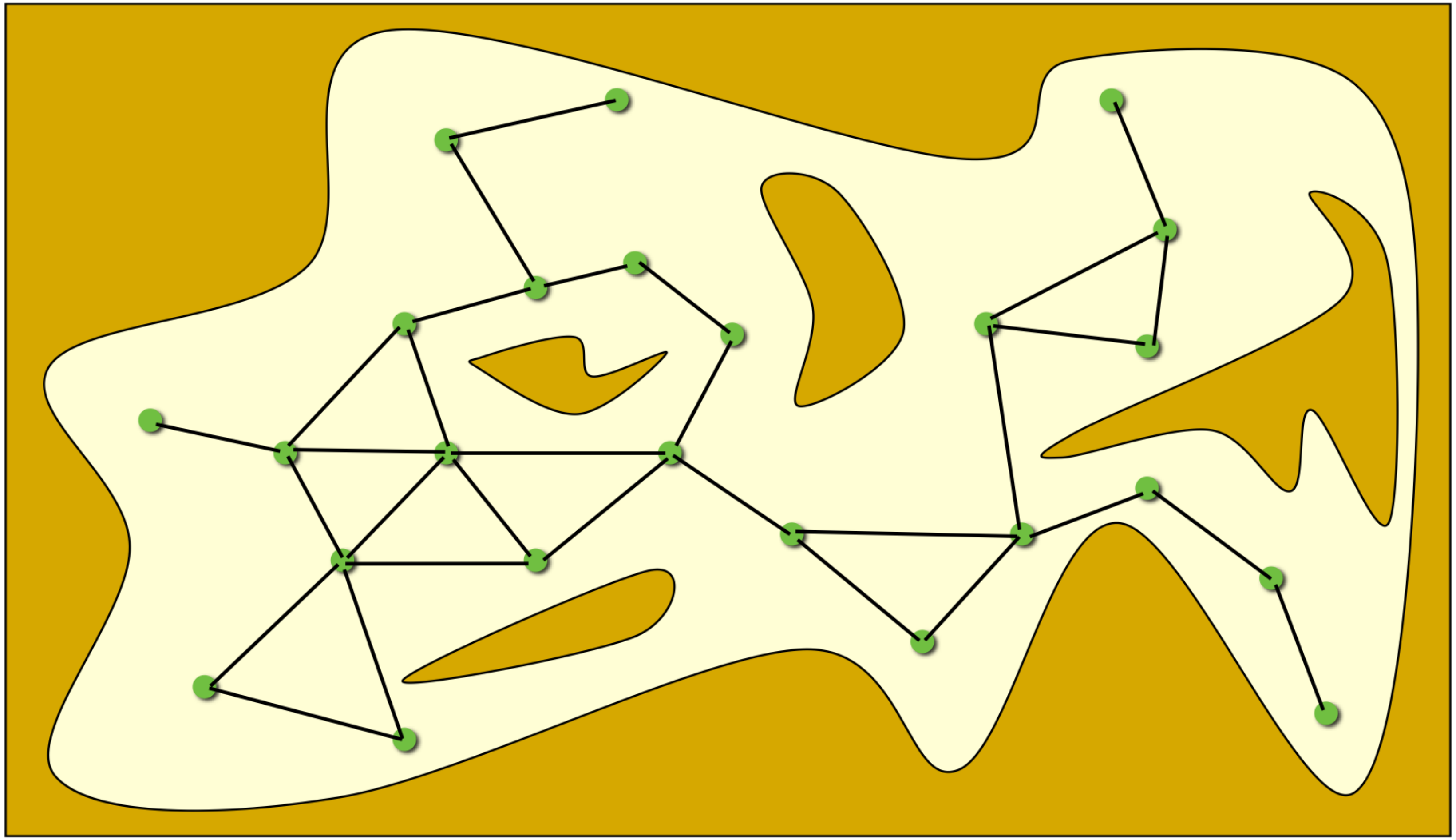
PRM is searched for a path from start (s) to goal (g)



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Roadmaps (PRM)

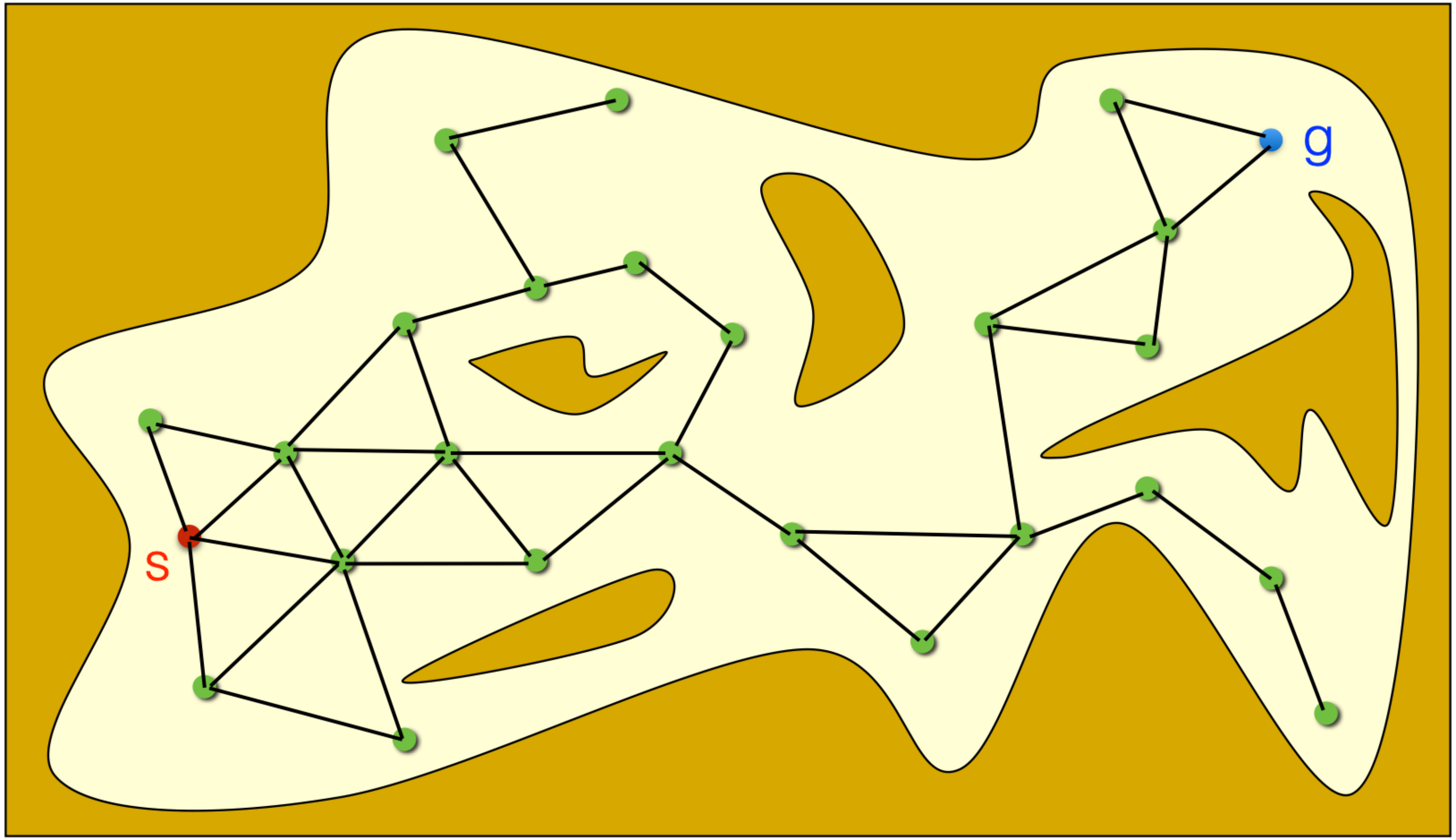
Collision-free edges are retained as local paths to form PRM



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Roadmaps (PRM)

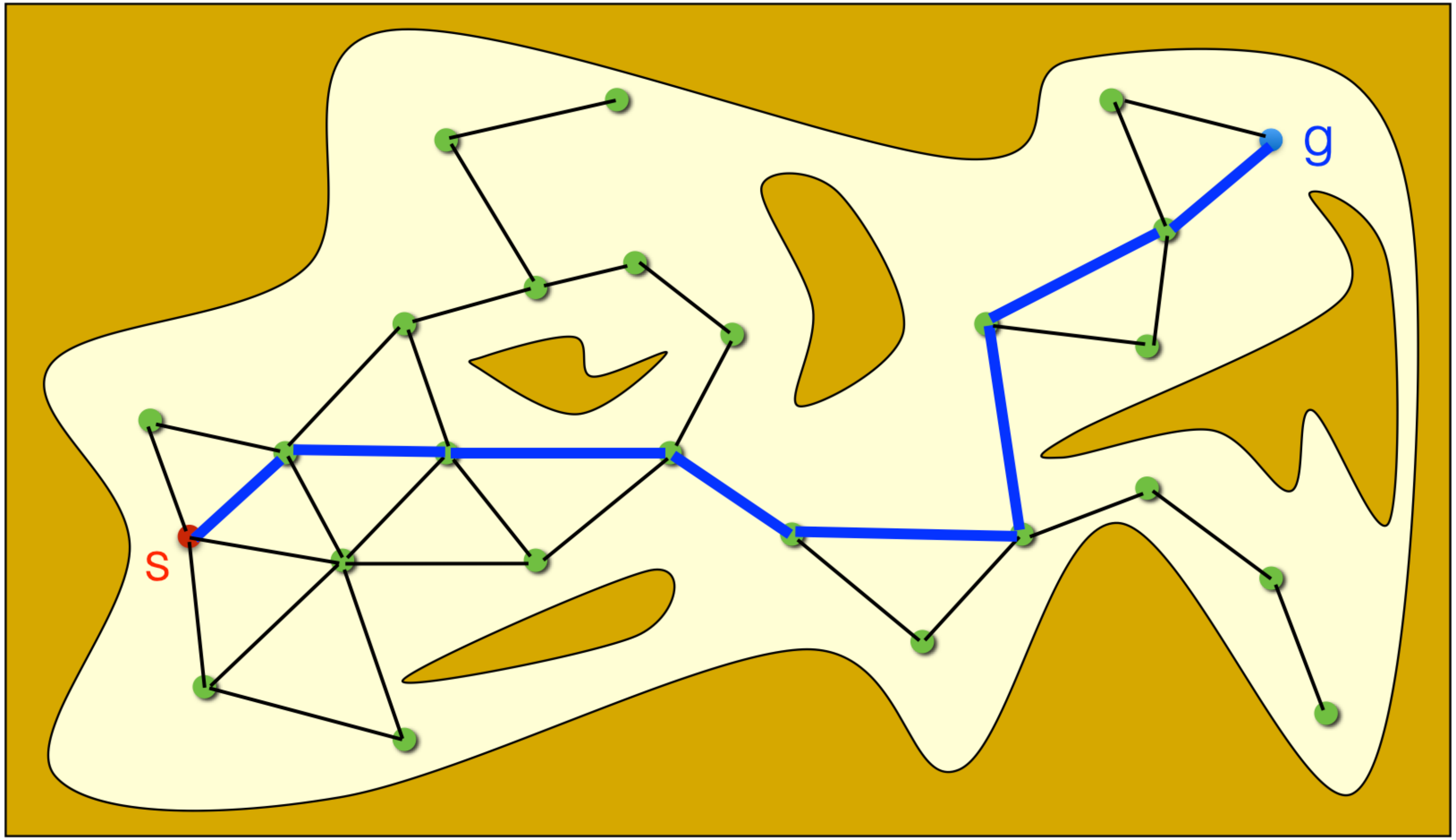
Start and goal configurations are included as milestones



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Roadmaps (PRM)

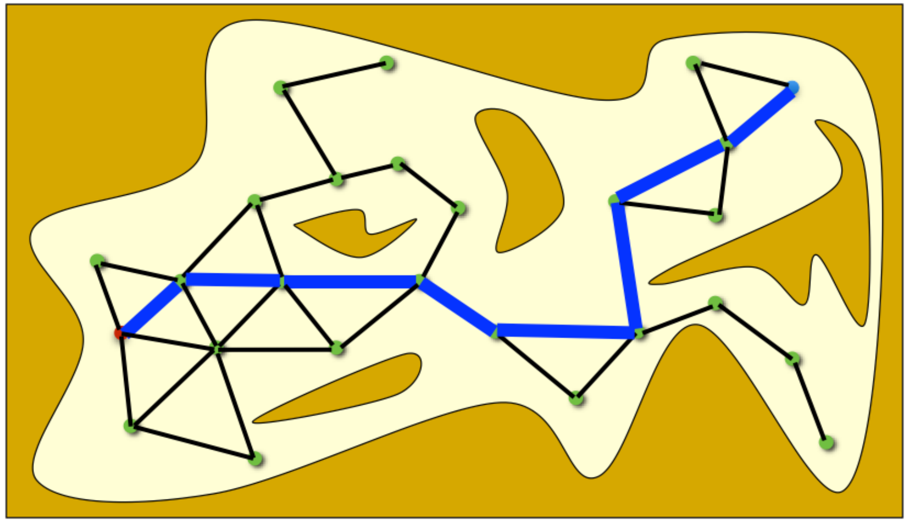
Collision-free edges are retained as local paths to form PRM



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Roadmaps (PRM)

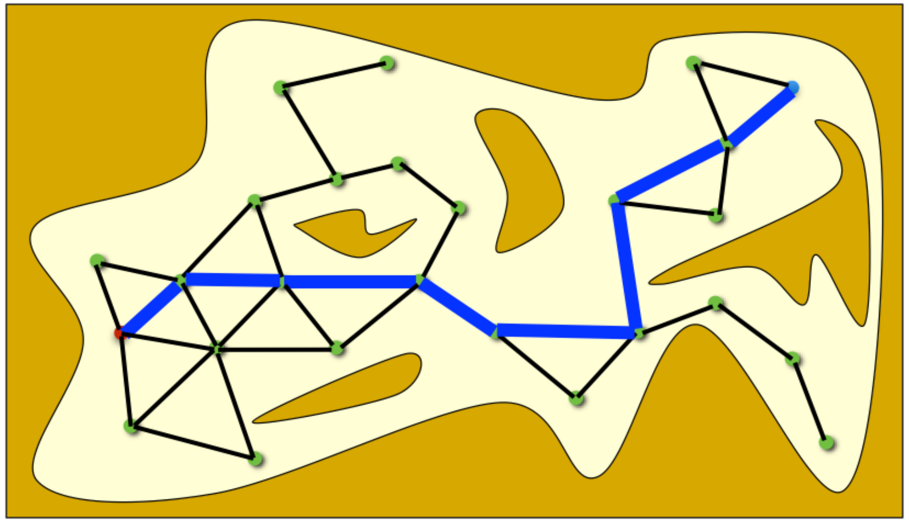
- Recap
  - Randomly sample configurations from C-Space
  - Connect them to nearest neighbors (if no collisions with obstacles)
  - Two primitive procedures (workspace collision query):
    - Check if configuration is in free space
    - Check if an edge is in free space
- PRM can be used for multiple queries



How is this different from vanilla search problems?

Probabilistic Roadmaps (PRM)

- Recap
  - Randomly sample configurations from C-Space
  - Connect them to nearest neighbors (if no collisions with obstacles)
  - Two primitive procedures (workspace collision query):
    - Check if configuration is in free space
    - Check if an edge is in free space
- PRM can be used for multiple queries



How is this different from vanilla search problems?

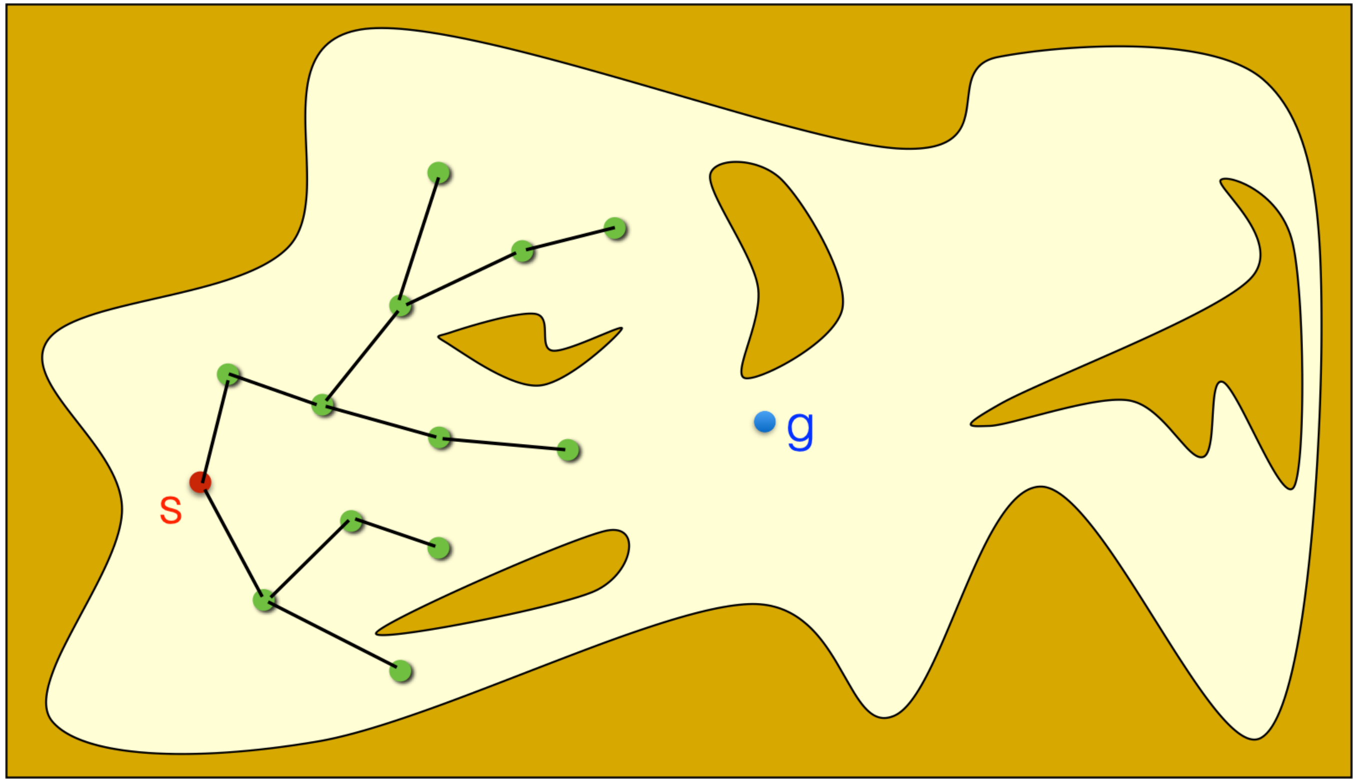
- Uncountably infinite state space (hence the need for sampling)

- Connectivity needs to be computed on the fly

+ Known metric as a starting point

Rapidly-Exploring Random Trees (RRT)

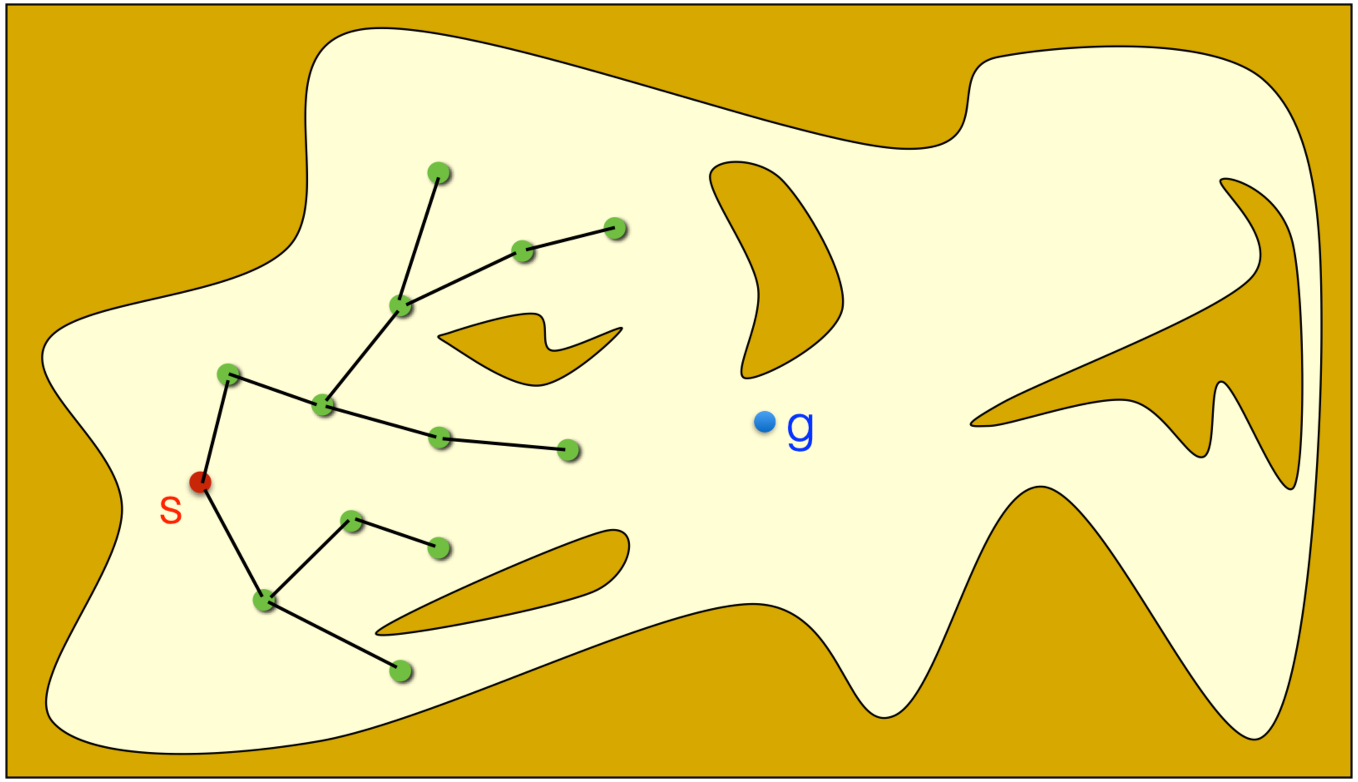
Data structure: T = (nodes V, edges E)



[Source: CS287, Pieter Abbeel, UC Berkeley]

Rapidly-Exploring Random Trees (RRT)

1. $x \leftarrow \texttt{Sample}()$ // Sample configuration from C-space
2. $v \leftarrow \texttt{Nearest}(T, x)$ // Find nearest node in the tree to sample
3. $v' \leftarrow \texttt{Extend}(v, x)$ // Try extension nearest node towards sample
4. If $(\texttt{ObstacleFree}(v, v'))$ then // If extension does not collide with obstacles
5. $\quad V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$ // Add new node and edge to the tree



[Source: CS287, Pieter Abbeel, UC Berkeley]

Rapidly-Exploring Random Trees (RRT)

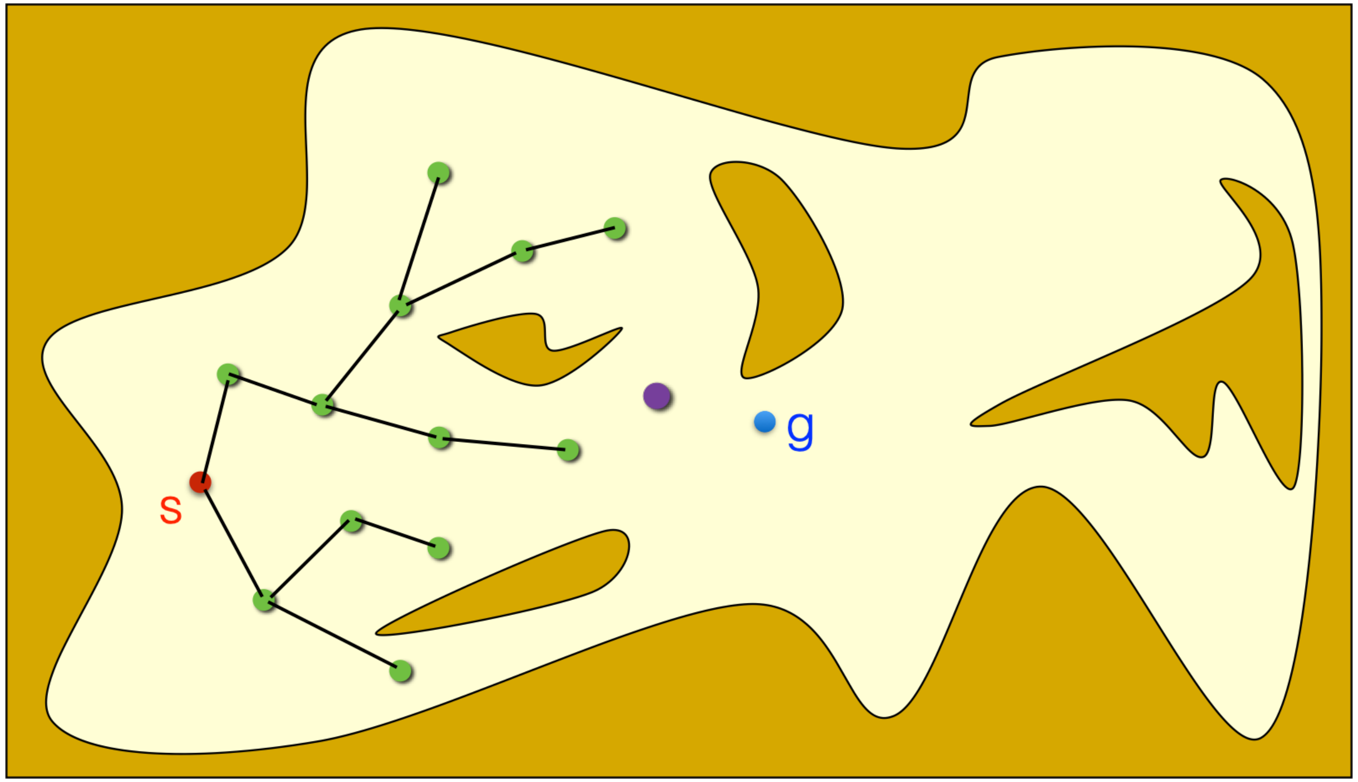
$\boxed{1.\ x \leftarrow \texttt{Sample}()}$ // Sample configuration from C-space

2. $v \leftarrow \texttt{Nearest}(T, x)$ // Find nearest node in the tree to sample

3. $v' \leftarrow \texttt{Extend}(v, x)$ // Try extension nearest node towards sample

4. If $(\texttt{ObstacleFree}(v, v'))$ then // If extension does not collide with obstacles

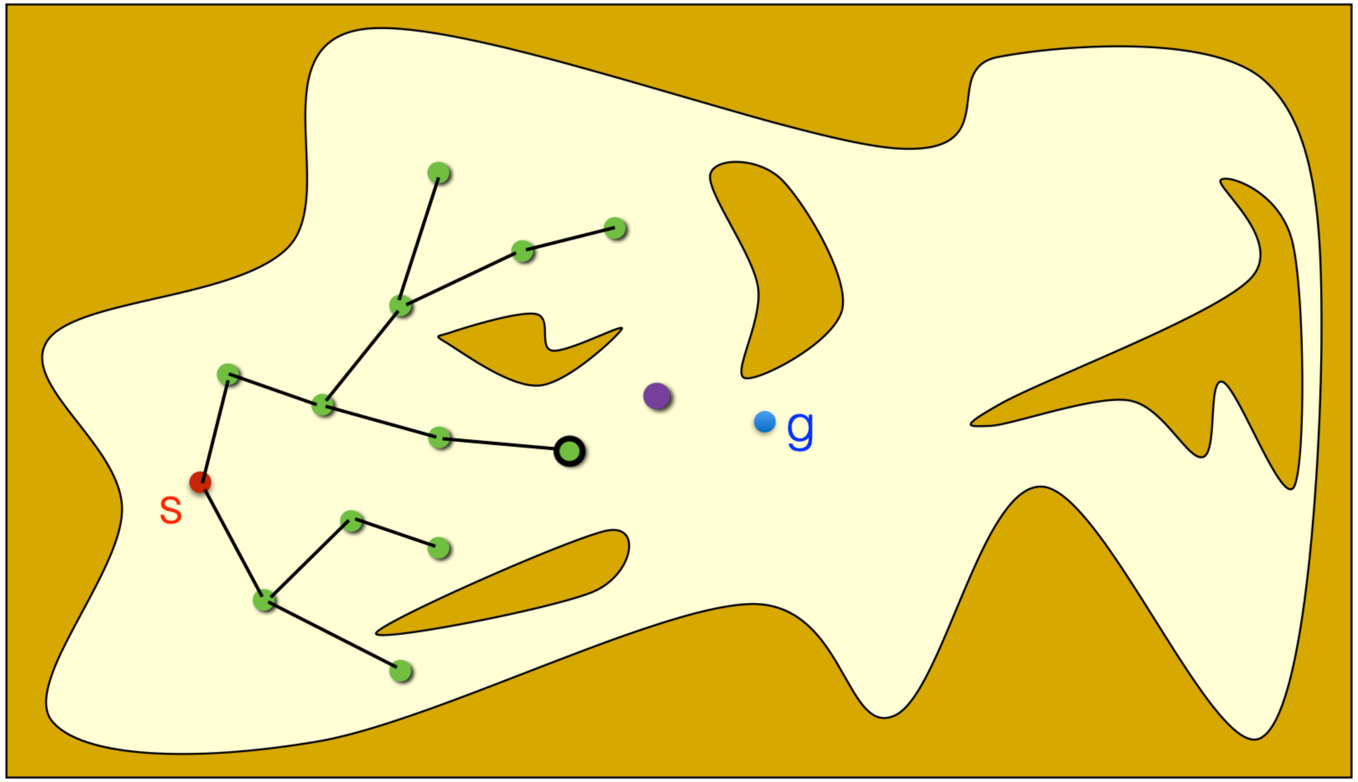
5. $\quad V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$ // Add new node and edge to the tree



[Source: CS287, Pieter Abbeel, UC Berkeley]

Rapidly-Exploring Random Trees (RRT)

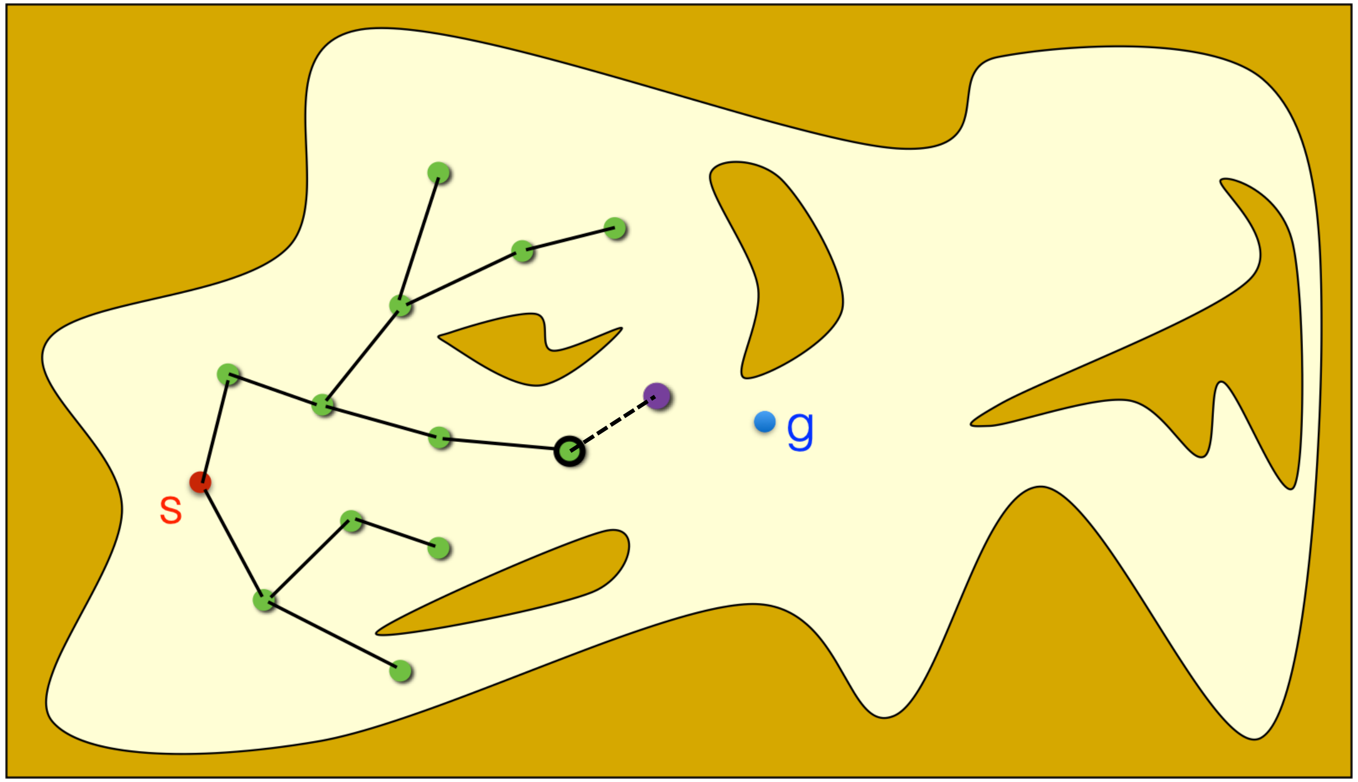
1. $x \leftarrow \texttt{Sample}()$ — // Sample configuration from C-space
2. $\boxed{v \leftarrow \texttt{Nearest}(T, x)}$ — // Find nearest node in the tree to sample
3. $v' \leftarrow \texttt{Extend}(v, x)$ — // Try extension nearest node towards sample
4. If $(\texttt{ObstacleFree}(v, v'))$ then — // If extension does not collide with obstacles
5. $\quad V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$ — // Add new node and edge to the tree



[Source: CS287, Pieter Abbeel, UC Berkeley]

Rapidly-Exploring Random Trees (RRT)

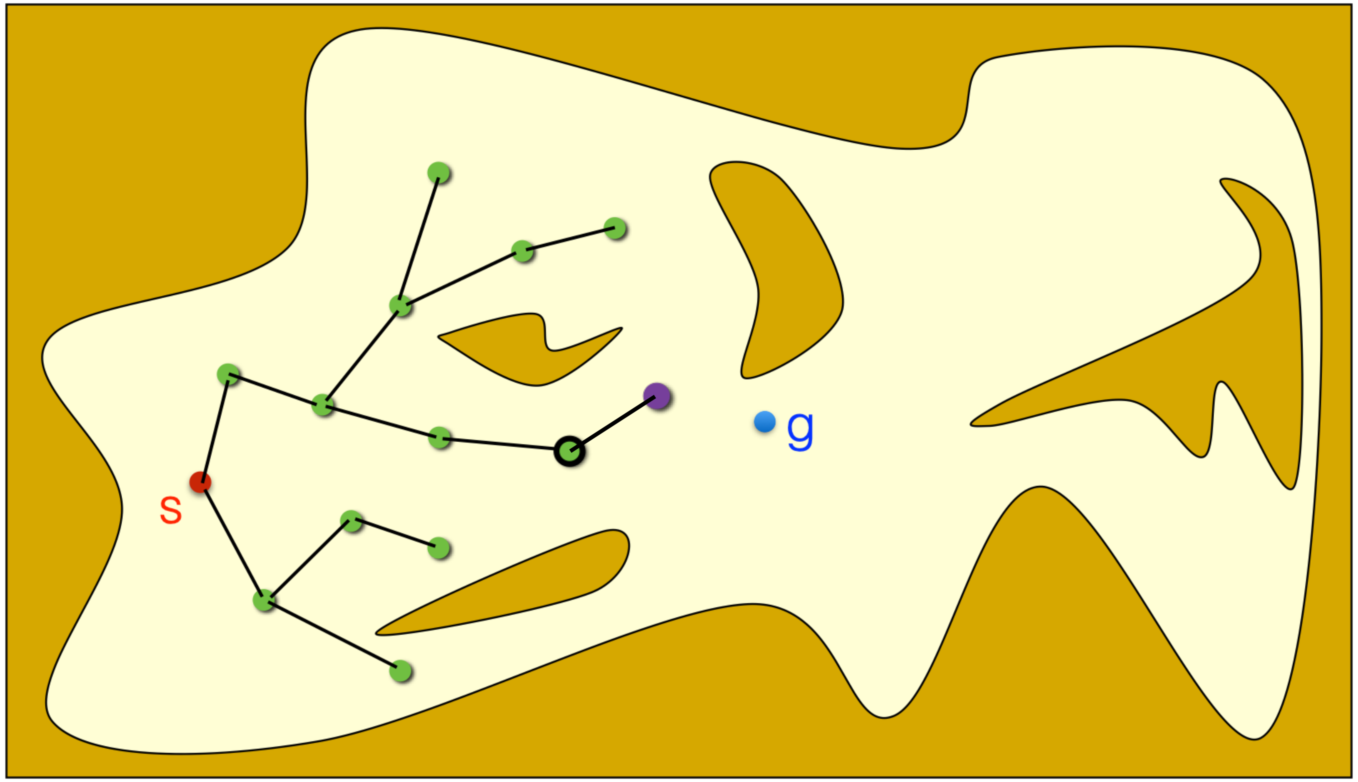
1. $x \leftarrow \texttt{Sample}()$ // Sample configuration from C-space
2. $v \leftarrow \texttt{Nearest}(T, x)$ // Find nearest node in the tree to sample
3. $\boxed{v' \leftarrow \texttt{Extend}(v, x)}$ // Try extension nearest node towards sample
4. If $(\texttt{ObstacleFree}(v, v'))$ then // If extension does not collide with obstacles
5. $\quad V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$ // Add new node and edge to the tree



[Source: CS287, Pieter Abbeel, UC Berkeley]

Rapidly-Exploring Random Trees (RRT)

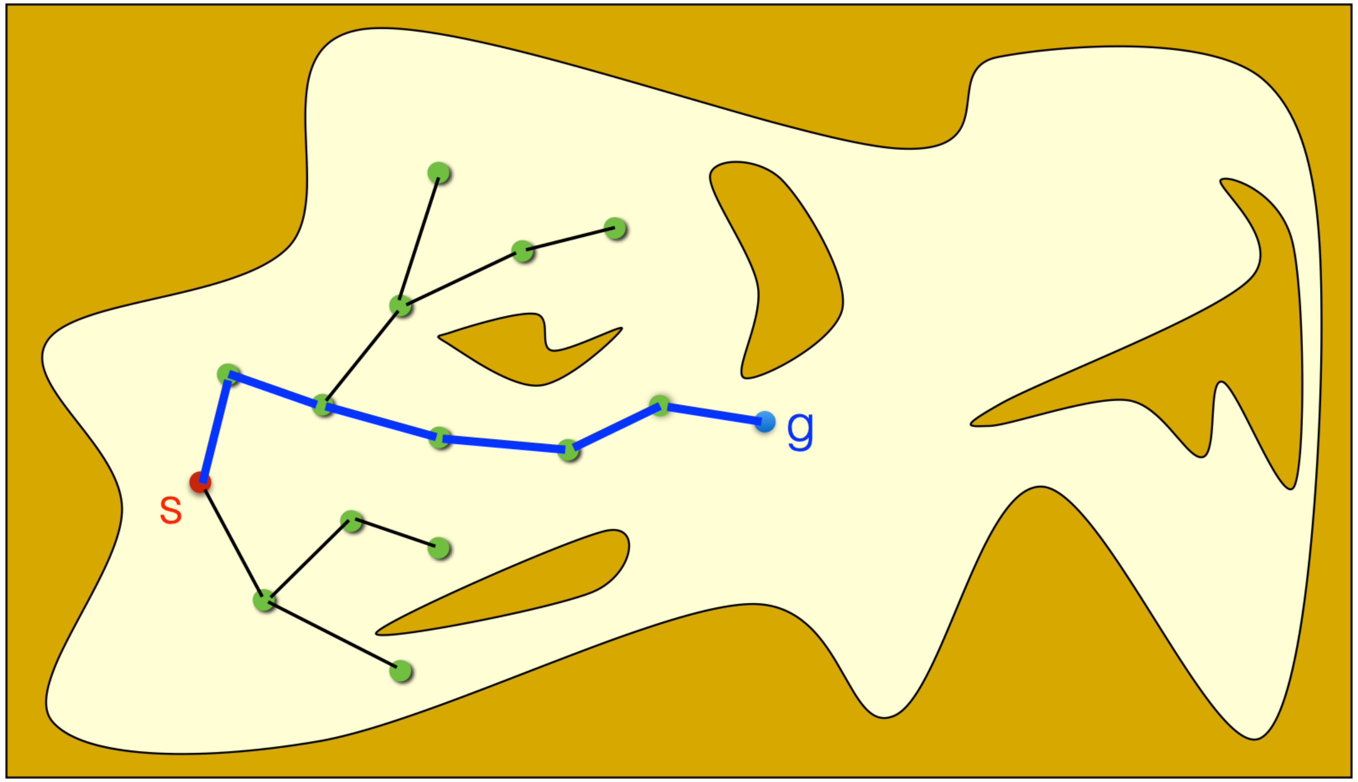
$x \leftarrow \texttt{Sample}()$	// Sample configuration from C-space
$v \leftarrow \texttt{Nearest}(T, x)$	// Find nearest node in the tree to sample
$v' \leftarrow \texttt{Extend}(v, x)$	// Try extension nearest node towards sample
If $(\texttt{ObstacleFree}(v, v'))$ then	// If extension does not collide with obstacles
$\quad V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$	// Add new node and edge to the tree



[Source: CS287, Pieter Abbeel, UC Berkeley]

Rapidly-Exploring Random Trees (RRT)

1. $x \leftarrow$ `Sample()` // Sample configuration from C-space
2. $v \leftarrow$ `Nearest`(T, x) // Find nearest node in the tree to sample
3. $v' \leftarrow$ `Extend`(v, x) // Try extension nearest node towards sample
4. If (`ObstacleFree`(v, v')) then // If extension does not collide with obstacles
5. $\quad V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$ // Add new node and edge to the tree



[Source: CS287, Pieter Abbeel, UC Berkeley]

Early Approaches

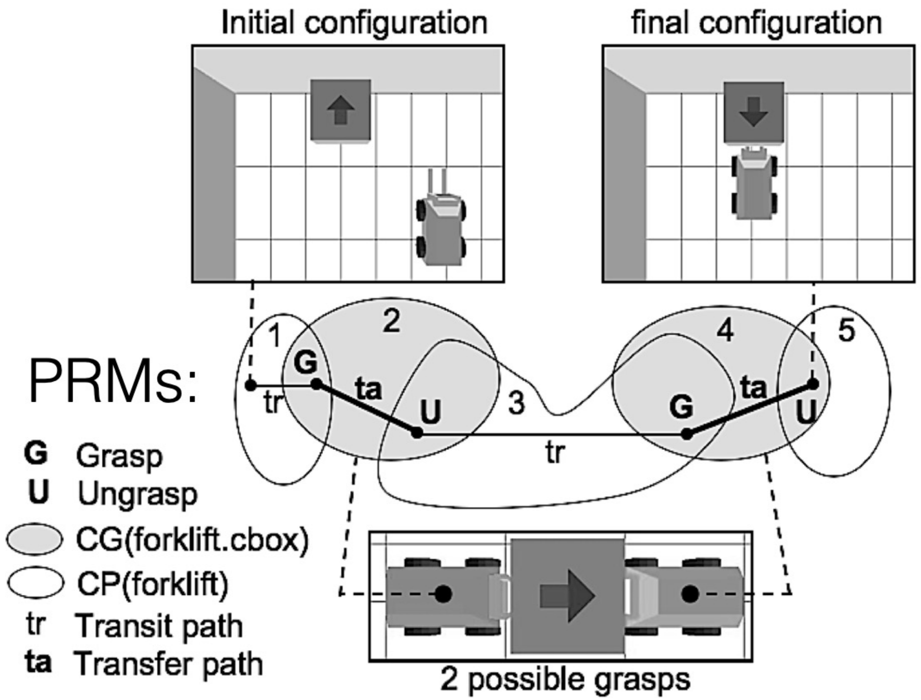
Early Characterization of Key Challenges: aSyMov

- Approach built upon PRMs
- *Articulated key technical problem:* picking and placing objects are discrete events, change the "robot", c-space
  - Picking an object leads to a new, composed robot
- Different composed versions of the robot have different C-spaces
- Solution Idea:
  - Maintain separate (projected) PRMs for different versions of the robot
  - Link them up based on actions such as pick and place
- Motion planning after a pick-up would use the PRM for the composed robot
- Where does task planning come in?
  - Goal specified in PDDL-like language
  - Task plan is used as a heuristic in PRM expansion

Cambon, S., Alami, R., & Gravot, F. (2009). A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research*

aSyMov: Example Problem

- PRM for robot + box is disconnected
  - Components correspond to different grasps
- PRM for robot alone is also disconnected.
  - Components correspond to different box positions
- Need a path through linked points
- Grow a PRM per action; bias expansion by using high-level plan cost as a heuristic for c-states
- High-level model may be abstract (inaccurate) but used in a limited manner – not updated



High-Level Summary

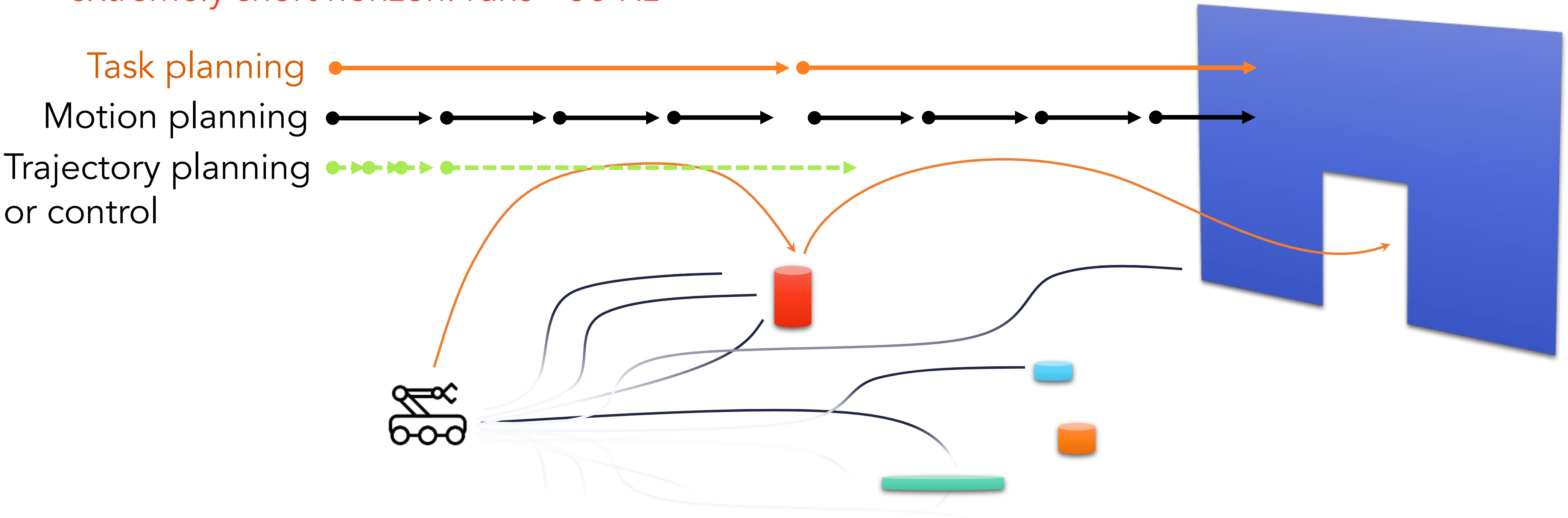
	Search Space	High Level Reasoning	Low Level Reasoning	High-level	High Level Language
aSyMov	Single	Any TP	PRM/RRT	Symbolic	PDDL

Outline

1. Background: Why Task and Motion Planning?
2. Abstraction as a Foundation for TMP
3. Modern Abstraction-Based Approaches
4. Research Frontier: Neuro-Symbolic Abstraction Learning for TMP

Recall Problem Hierarchies

- Task planning – given a task planning problem, computes the high-level action the robot should perform at each step
  - Long horizon: each action takes usually a few minutes to complete.
- Motion planning – given a motion planning problem, computes the path that the robot should take
  - short horizon: each waypoint usually takes a few seconds to be achieved
- (Trajectory planning – selects the control inputs that should go to the robot's motors)
  - extremely short horizon: runs ~50 Hz



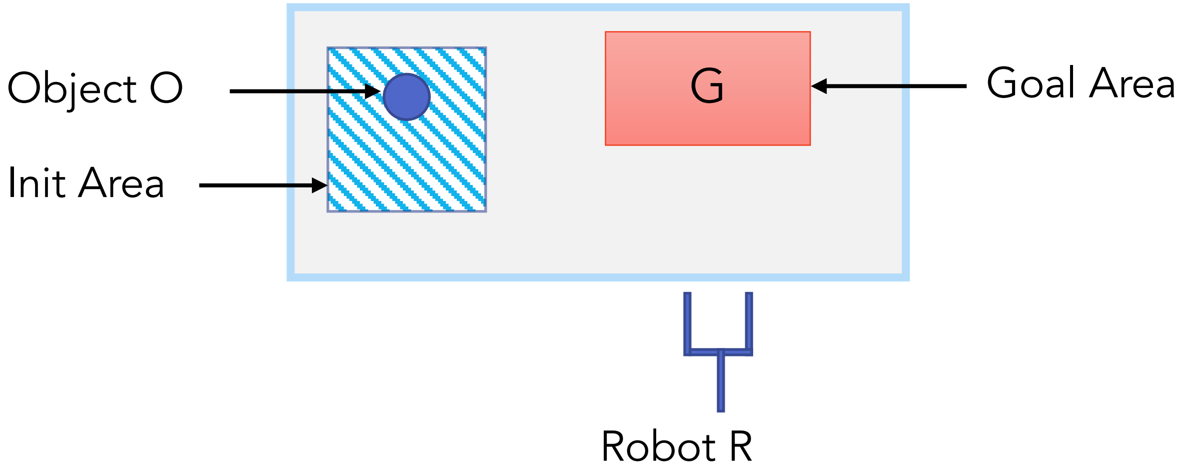
Can we better utilize this hierarchy?

Can we somehow exploit this hierarchy?

Ans: **Yes – and the key concept is "abstractions"**

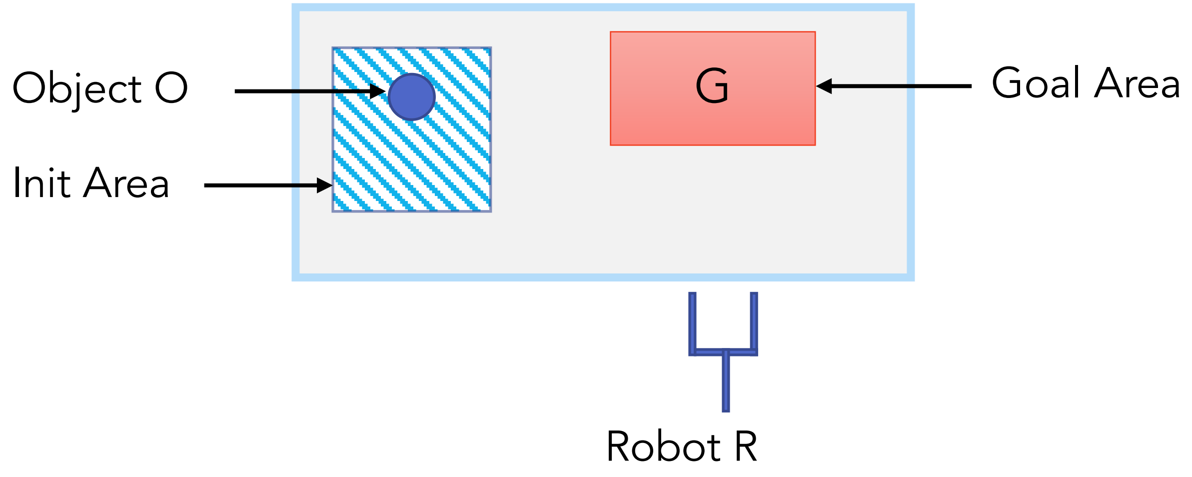
Domain for Robot Planning

```
(:action Move
 :parameter ?robot ?location ?trajectory
 :precondition
       not At(?robot ?location)
       Collision-free(?trajectory)
 :effect
       At(?robot ?location)
 )
```



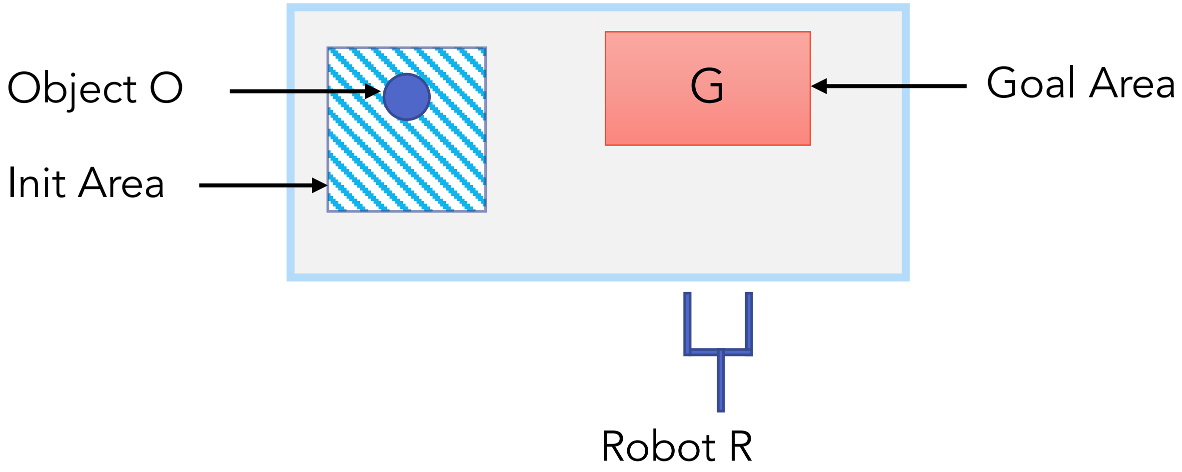
Domain for Robot Planning

```
(:action Move
 :parameter ?robot ?location ?trajectory
 :precondition
      not At(?robot ?location)
      Collision-free(?trajectory)
 :effect
      At(?robot ?location)
 )
```



Key challenges: infinitely many facts, infinite branching factor

Abstraction: State Abstraction



E.g., At(O,Init) is True iff $\exists l \in BlueArea$ s.t. pose(o,l) = True.

Abstraction: State Abstraction

First-order logic queries from concrete vocabulary V_l to abstract vocabulary V_h where $V_h \subset V_l$.

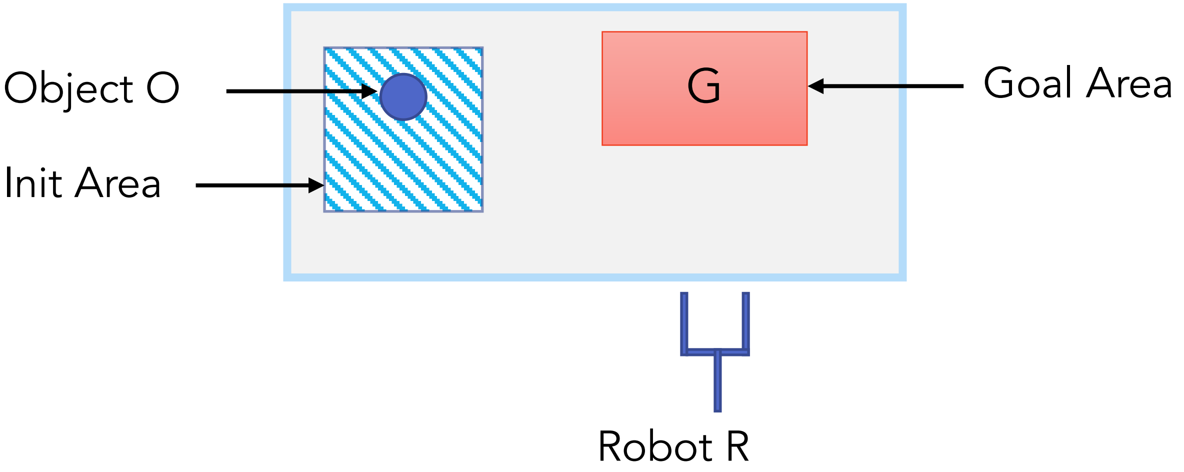
The query $[r]_{s_h}(\overline{o_1}, \ldots, \overline{o_n}) = True$ iff

$\exists o_1, \ldots, o_n$ such that $o_i \in \rho(\overline{o_i})$ and

$[\varphi_r^{\alpha_\rho}(o_1, \ldots, o_n)]_{s_l} = True.$

Collection function

Symbolic Reference



E.g., At(O,Init) is True iff $\exists l \in BlueArea$ s.t. pose(o,l) = True.

Here, $\rho(Init) = BlueArea = \{p_1, \ldots, p_n\}$

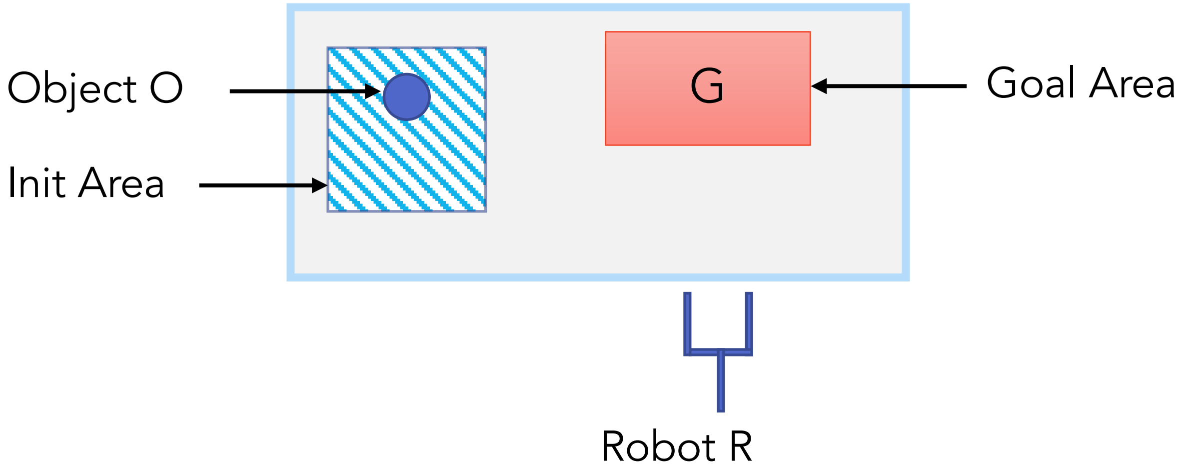
Abstraction: Symbolic Actions

First-order logic queries from concrete vocabulary V_l to abstract vocabulary V_h where $V_h \subset V_l$.

The query $[r]_{s_h}(\,\overline{o_1}, \dots, \overline{o_n}\,) = \mathit{True}$ iff

$\exists o_1, \dots, o_n$ such that $o_i \in \rho(\overline{o_i})$ and

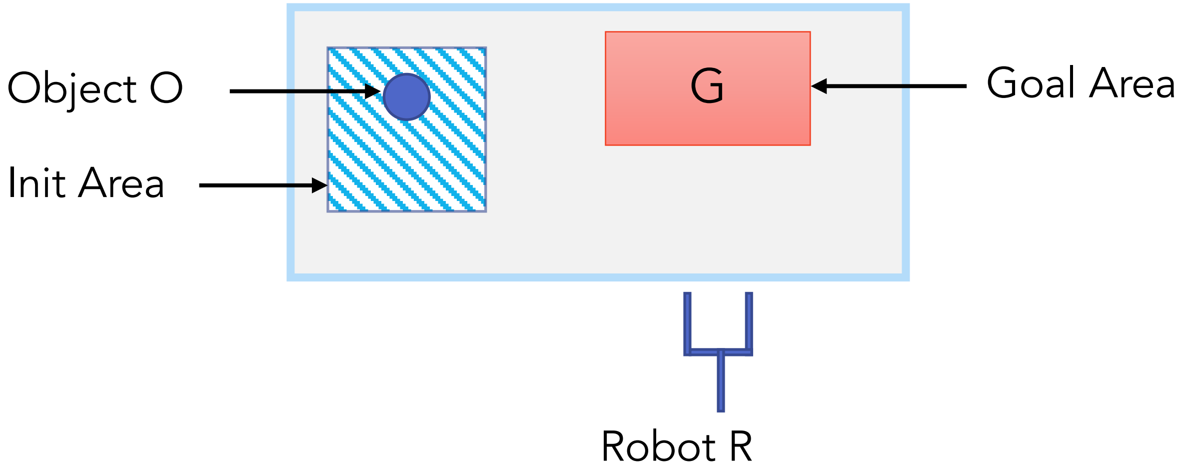
$[\varphi_r^{\alpha_\rho}(o_1, \dots, o_n)]_{s_l} = \mathit{True}$.




```
(:action Move
 :parameter ?robot ?location ?trajectory
 :precondition
       not At(?robot ?location)
       Collision-free(?trajectory)
 :effect
       At(?robot ?location)
 )
```

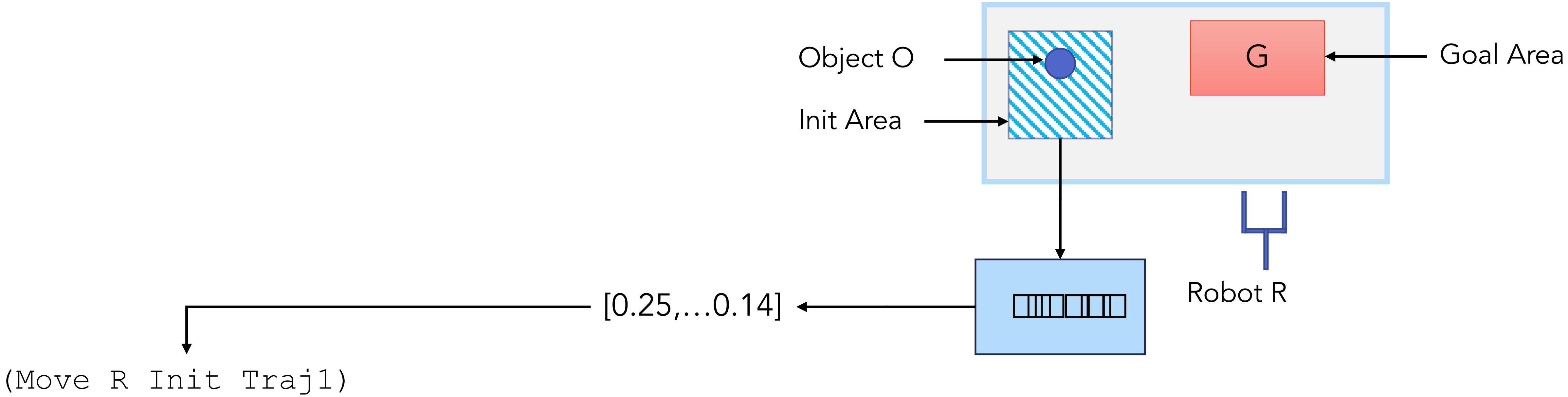
Symbolic arguments that can be instantiated with symbolic references for high-level planning

Abstraction: State Abstraction

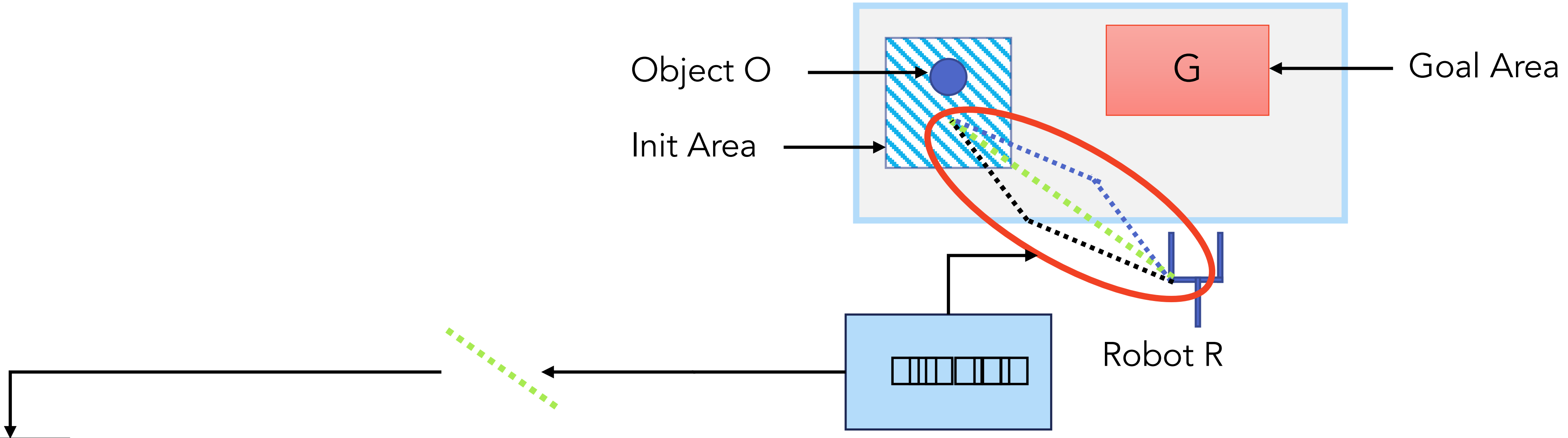


`(Move R Init Traj1)`

Abstraction: State Abstraction

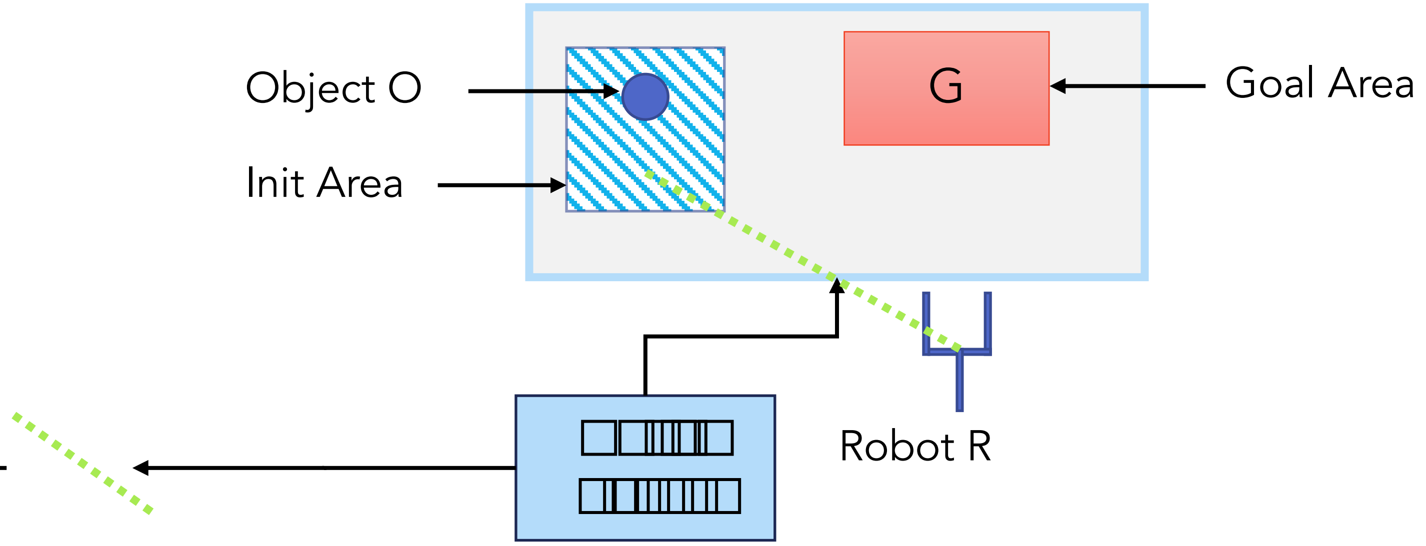


Abstraction: Refining Symbolic Action



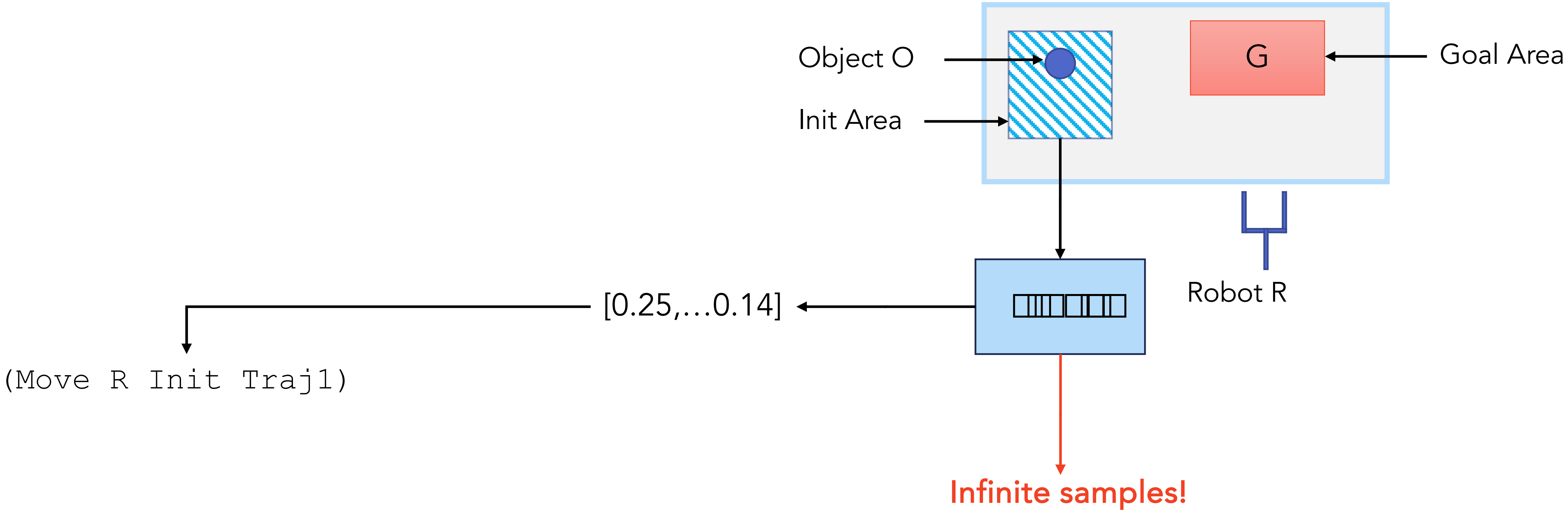
`(Move R Init Traj1)`

Abstraction: Refining Symbolic Action

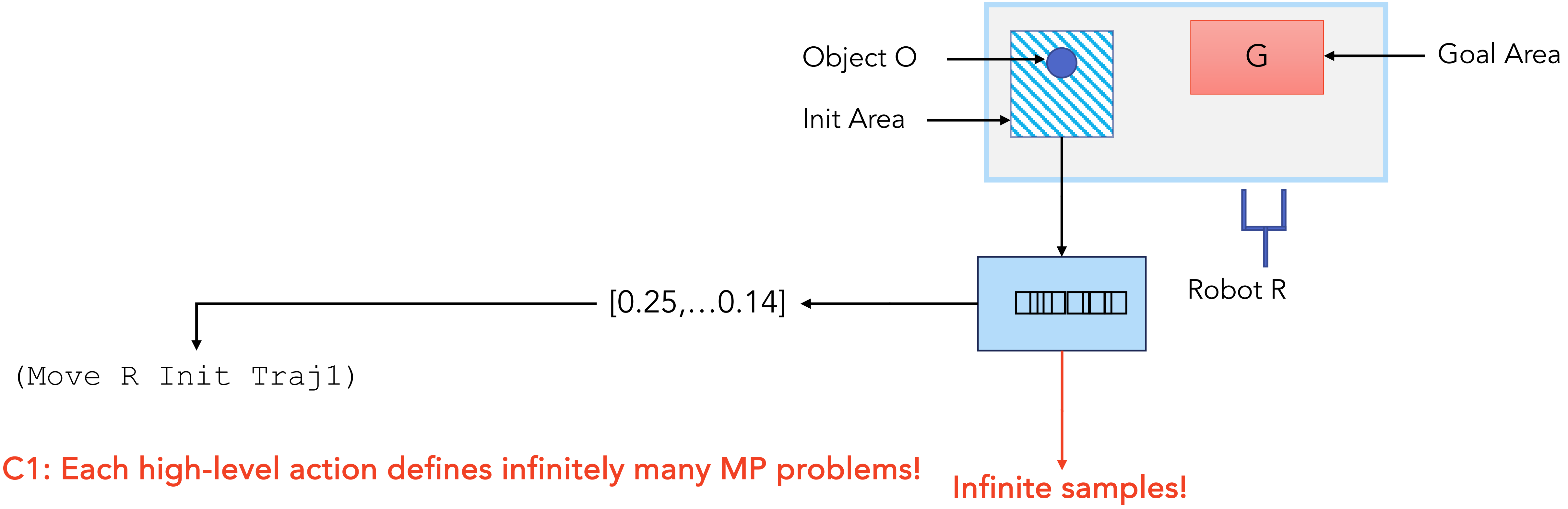


(Move R Init $\boxed{\texttt{Traj1}}$)

Challenge 1 – Which MP to Solve?



Challenge 1 – Which MP to Solve?

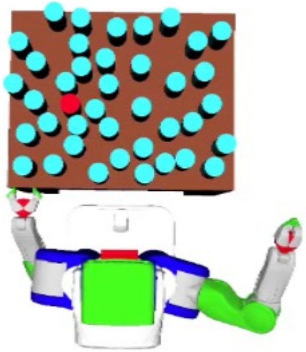


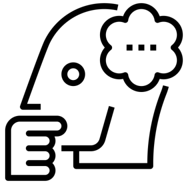
C1: Each high-level action defines infinitely many MP problems!

Challenge 2 – Which MP Will be Solvable?

Can pickup only from the side

Obstructions depend on choice of movement trajectory





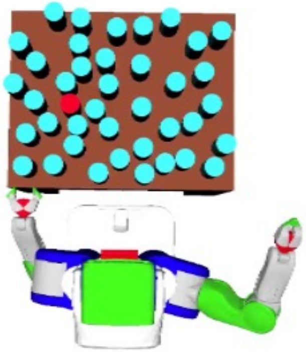
No way to verify which trajectories are collision-free

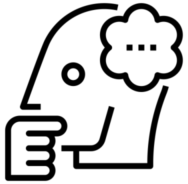
```
(:action Move
 :parameter ?robot ?location ?trajectory
 :precondition
        not At(?robot ?location)
        Collision-free(?trajectory)
 :effect
        At(?robot ?location)
 )
```

Challenge 2 – Which MP Will be Solvable?

Can pickup only from the side

Obstructions depend on choice of movement trajectory





No way to verify which trajectories are collision-free

```
(:action Move
 :parameter ?robot ?location ?trajectory
 :precondition
       not At(?robot ?location)
       Collision-free(?trajectory)
 :effect
       At(?robot ?location)
 )
```

C2: Loss of information in abstraction → Actions in plan that can not refinable

Task and Motion Planning Problem

O, universe of objects and object poses (implicitly defined)

$P = P_{sym} \cup \bar{P}_h$, set of symbolic predicates and interpretations (definitions in geometric constraints)

Generates S, set of abstract states

$A = A_{sym} \cup \bar{A}_h$, set of abstract actions

$T: S \times A \rightarrow \mu S$, action transition function

$R: S \times A \rightarrow \mathbb{R}$, costs and utility of states, actions (can express goals and some forms of preferences*)

γ, a concretization function (often in the form of samplers or generators)

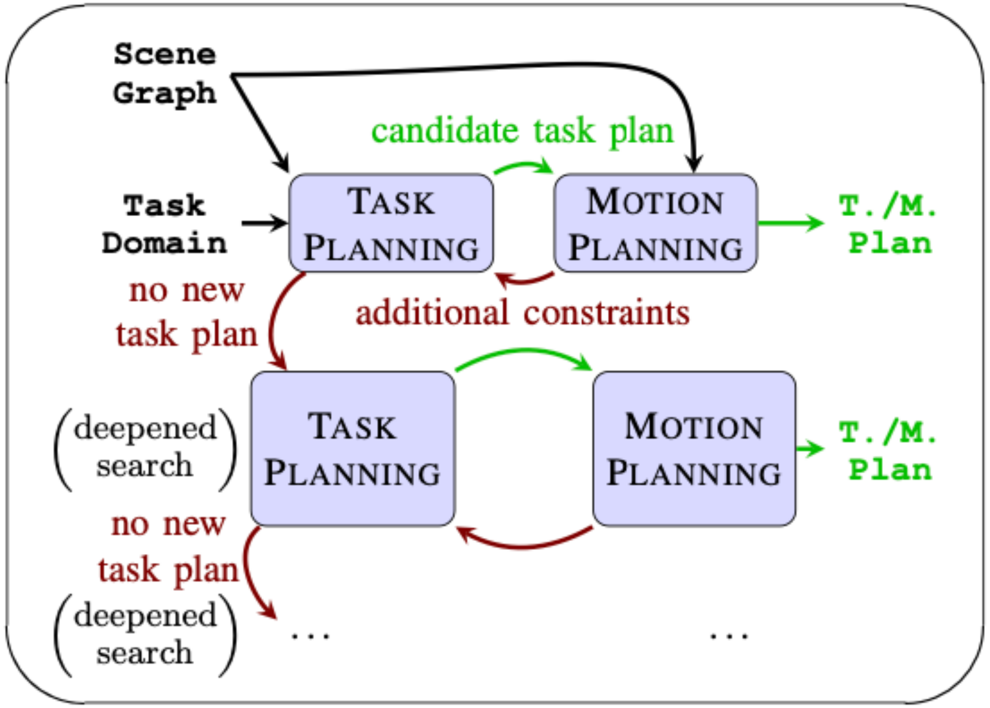
Task and motion planning:

Compute a sequence of actions from A that maximizes the utility R and that can be executed in the given model.

Optimization and SMT Based Approaches

IDTMP

- Core idea: Use advances in SAT/SMT solvers to perform hybrid search for TMP
- State variables for different objects and use poses as values
- Convert each high-level action to low-level motion planning problem.



Nedunuri, Srinivas, et al. "SMT-based synthesis of integrated task and motion plans from plan outlines." *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014.

Dantam, N. T., Kingston, Z. K., Chaudhuri, S., & Kavraki, L. E. (2018). An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research*,

IDTMP: SMT Planning

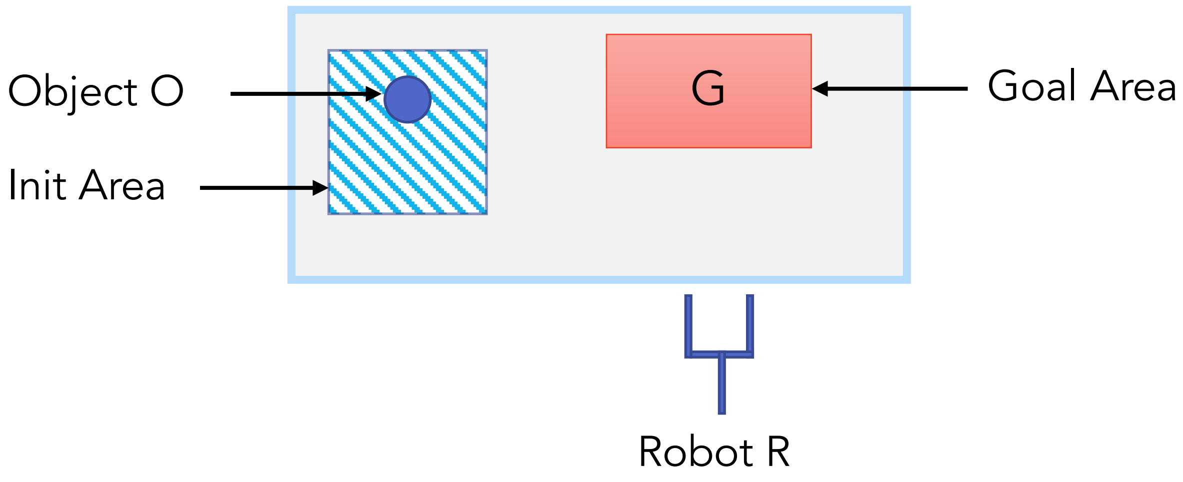
- SMT = satisfiability module theories
- Model the problem as Boolean satisfiability problem

IDTMP: SMT Planning

- SMT = satisfiability module theories
- Model the problem as Boolean satisfiability problem

State variables:
Pose of the object $P_0 \in R^2$
Configuration of the robot $C_R \in \mathcal{X}$

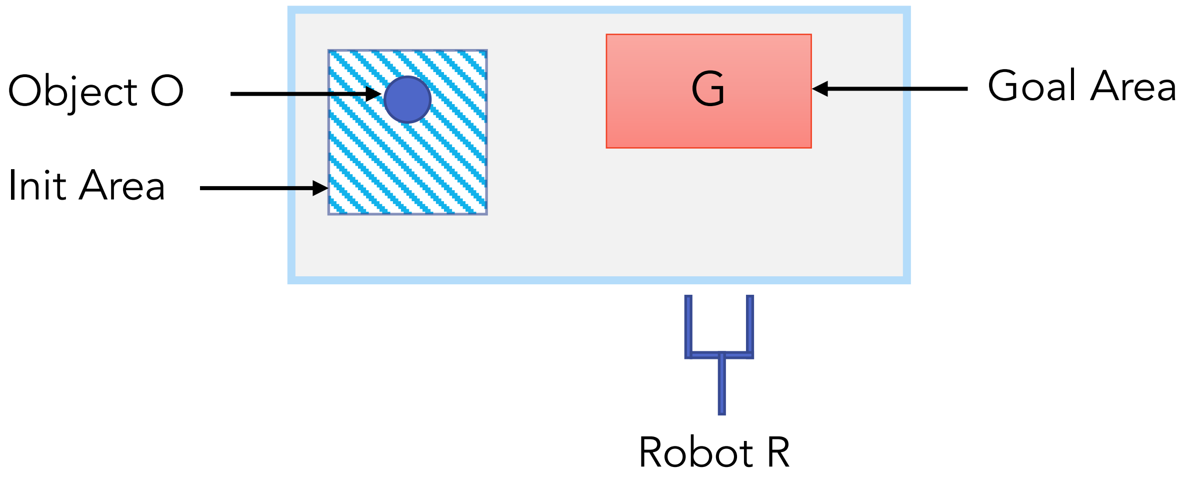
Actions:
Pick
Place
Move



IDTMP: SMT Planning

State variables:
Pose of the object $P_0 \in R^2$
Configuration of the robot $C_R \in \mathcal{X}$

Actions:
Pick
Place
Move



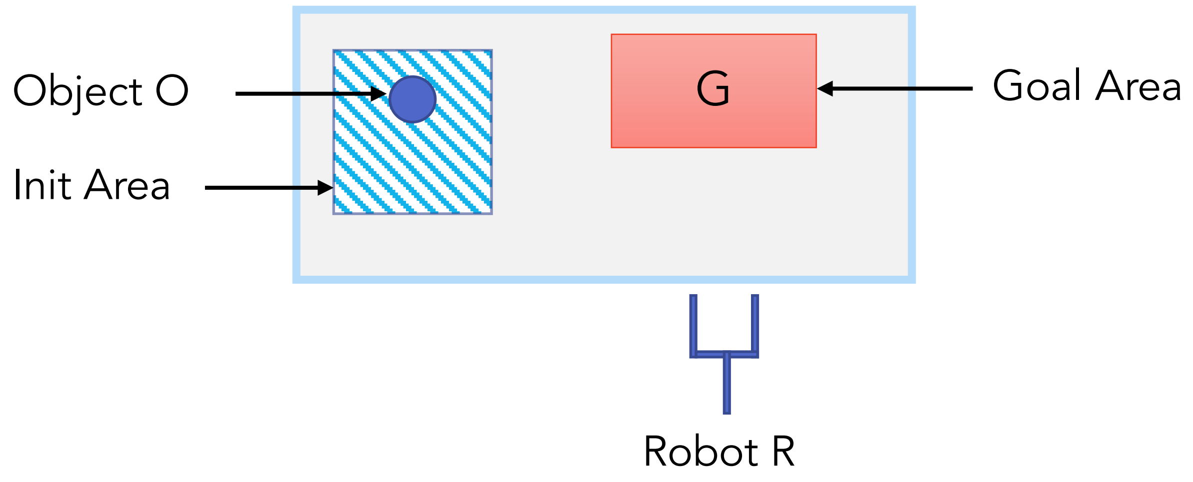
$P = \{p_0, \ldots, p_n\}$ (a set of state variables)

$A = \{a_0, \ldots, a_k\}$ (a set of actions)

IDTMP: SMT Planning

State variables:
Pose of the object $P_0 \in R^2$
Configuration of the robot $C_R \in \mathcal{X}$

Actions:
Pick
Place
Move



$P = \{p_0, \ldots, p_n\}$ (a set of state variables)

$A = \{a_0, \ldots, a_k\}$ (a set of actions)

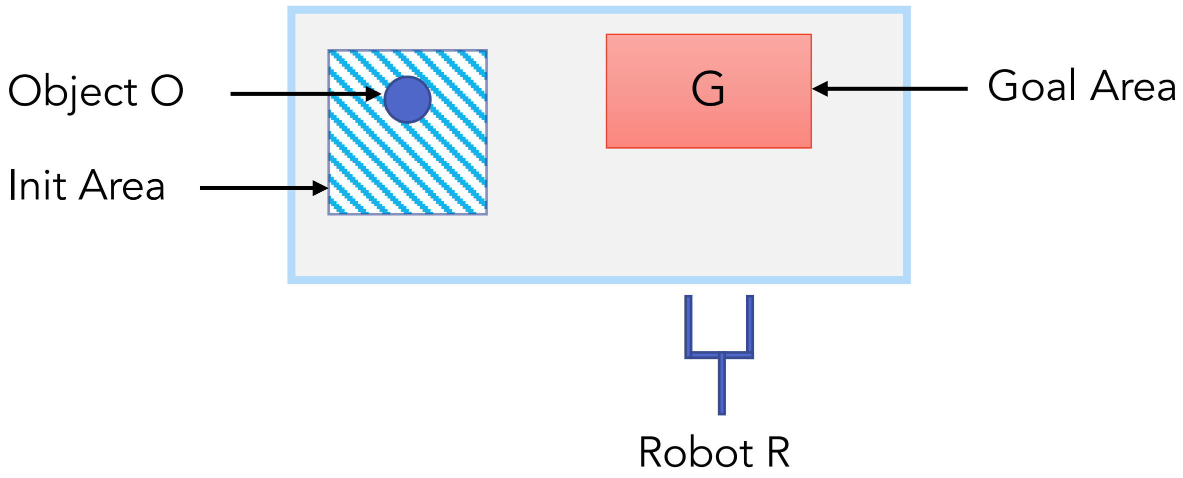
Three constraints:

1. $a_i^k \Rightarrow Pre(a_i)^k \wedge Eff(a_i)^{k+1}$ // If an action is taken at the step k, the its precondition and effect must hold

IDTMP: SMT Planning

State variables:
Pose of the object $P_0 \in R^2$
Configuration of the robot $C_R \in \mathcal{X}$

Actions:
Pick
Place
Move



$P = \{p_0, \dots, p_n\}$ (a set of state variables)

$A = \{a_0, \dots, a_k\}$ (a set of actions)

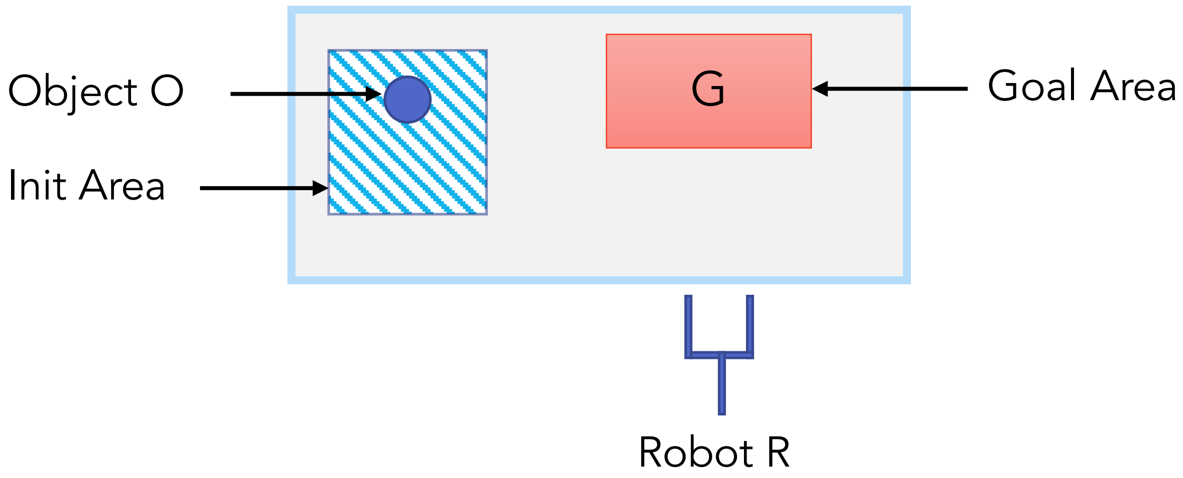
Three constraints:

1. $a_i^k \Rightarrow Pre(a_i)^k \wedge Eff(a_i)^{k+1}$ // If an action is taken at the step k, the its precondition and effect must hold
2. $(p_i^k = p_i^{k+1}) \vee (a_j^k \vee \dots \vee a_l^k)$ // Variables that are not changed by the actions remains unchanged

IDTMP: SMT Planning

State variables:
Pose of the object $P_0 \in R^2$
Configuration of the robot $C_R \in \mathcal{X}$

Actions:
Pick
Place
Move



$P = \{p_0, \dots, p_n\}$ (a set of state variables)

$A = \{a_0, \dots, a_k\}$ (a set of actions)

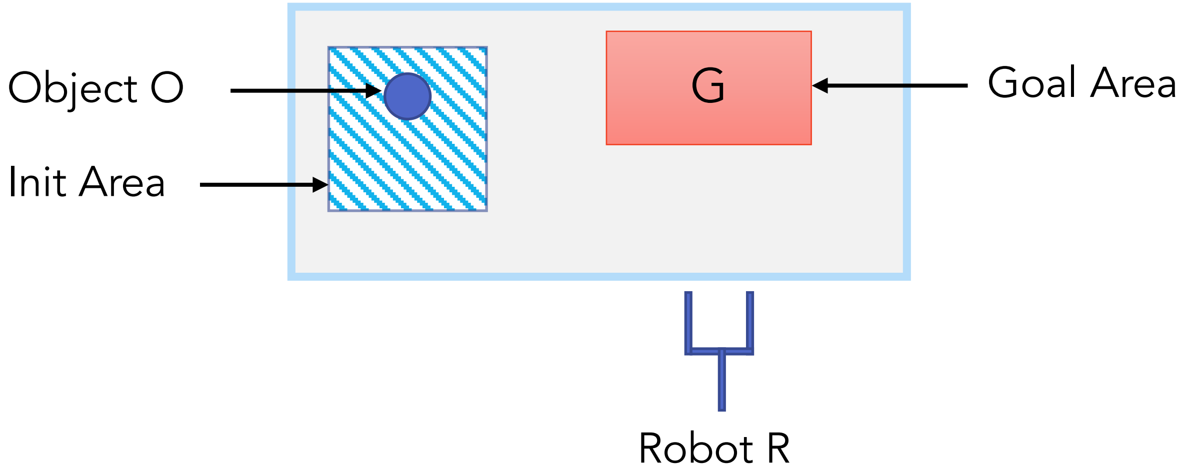
Three constraints:

1. $a_i^k \Rightarrow Pre(a_i)^k \wedge Eff(a_i)^{k+1}$ // If an action is taken at the step k, the its precondition and effect must hold
2. $(p_i^k = p_i^{k+1}) \vee (a_j^k \vee \dots \vee a_l^k)$ // Variables that are not changed by the actions remains unchanged
3. $a_i^k \Rightarrow \neg(a_0^k \vee \dots a_{i-1}^k \vee \dots a_{i+1}^k \vee \dots a_l^k)$ // Only one action can be taken at the given step

IDTMP: SMT Planning

State variables:
Pose of the object $P_0 \in R^2$
Configuration of the robot $C_R \in \mathcal{X}$

Actions:
Pick
Place
Move



$P = \{p_0, \dots, p_n\}$ (a set of state variables)

$A = \{a_0, \dots, a_k\}$ (a set of actions)

Three constraints:

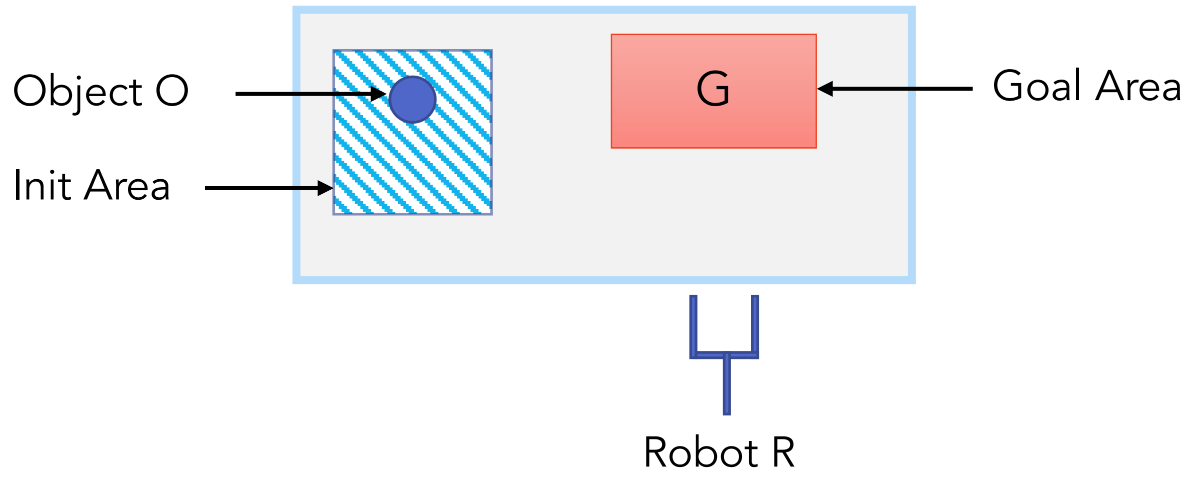
1. $a_i^k \Rightarrow Pre(a_i)^k \wedge Eff(a_i)^{k+1}$ // If an action is taken at the step k, the its precondition and effect must hold
2. $(p_i^k = p_i^{k+1}) \vee (a_j^k \vee \dots \vee a_l^k)$ // Variables that are not changed by the actions remains unchanged
3. $a_i^k \Rightarrow \neg\, (a_0^k \vee \quad \dots a_{i-1}^k \vee \quad \dots a_{i+1}^k \vee \quad \dots a_l^k)$ // Only one action can be taken at the given step

SMT formula using p_i and a_i // $p_i \in P$ (a set of state variables) and $a_i \in A$ (a set of actions)

IDTMP: SMT Planning

State variables:
Pose of the object $P_0 \in R^2$
Configuration of the robot $C_R \in \mathcal{X}$

Actions:
Pick
Place
Move



$P = \{p_0, \ldots, p_n\}$ (a set of state variables)
$A = \{a_0, \ldots, a_k\}$ (a set of actions)
Three constraints:

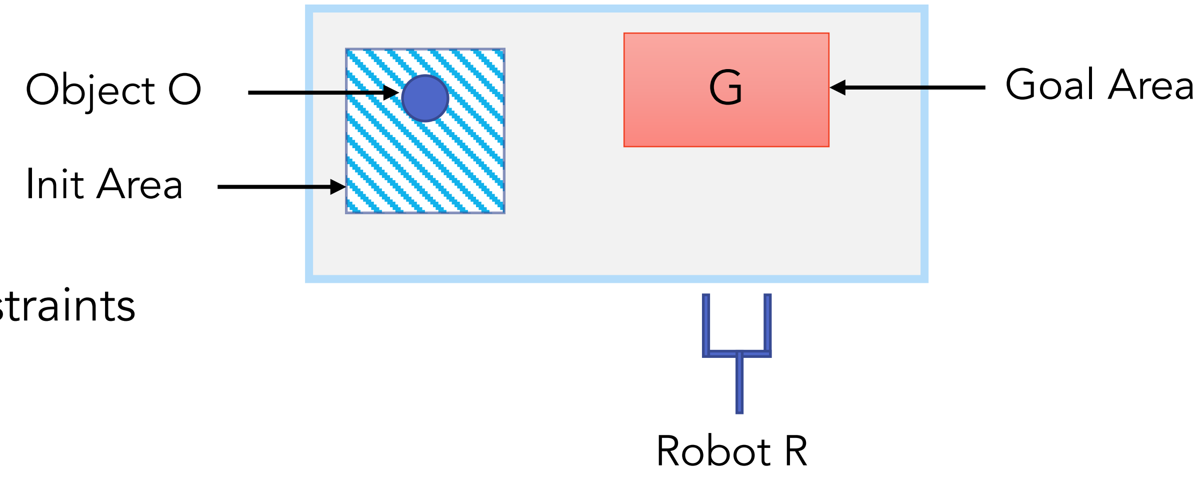
1. $a_i^k \Rightarrow Pre(a_i)^k \wedge Eff(a_i)^{k+1}$ // If an action is taken at the step k, the its precondition and effect must hold
2. $(p_i^k = p_i^{k+1}) \vee (a_j^k \vee \ldots \vee a_l^k)$ // Variables that are not changed by the actions remains unchanged
3. $a_i^k \Rightarrow \neg(a_0^k \vee \ldots a_{i-1}^k \vee \ldots a_{i+1}^k \vee \ldots a_l^k)$ // Only one action can be taken at the given step

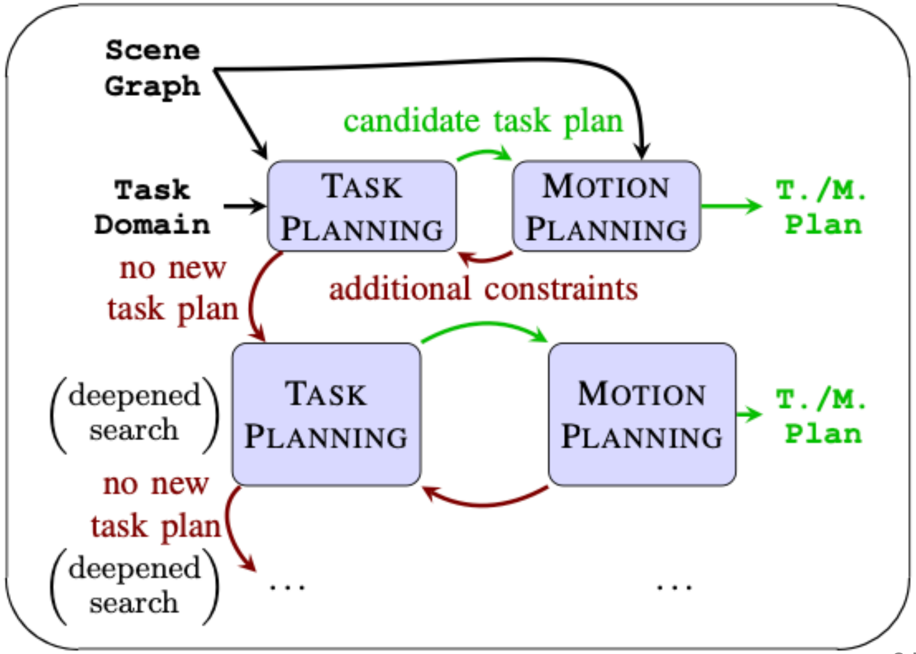
SMT formula using p_i and a_i // $p_i \in P$ (a set of state variables) and $a_i \in A$ (a set of actions)

SMT used
1) solve SMT formula consisting of discrete action and Boolean variables
2) maintain dynamic constraints such as action a_i is not applicable at step t

IDTMP: Overall Approach

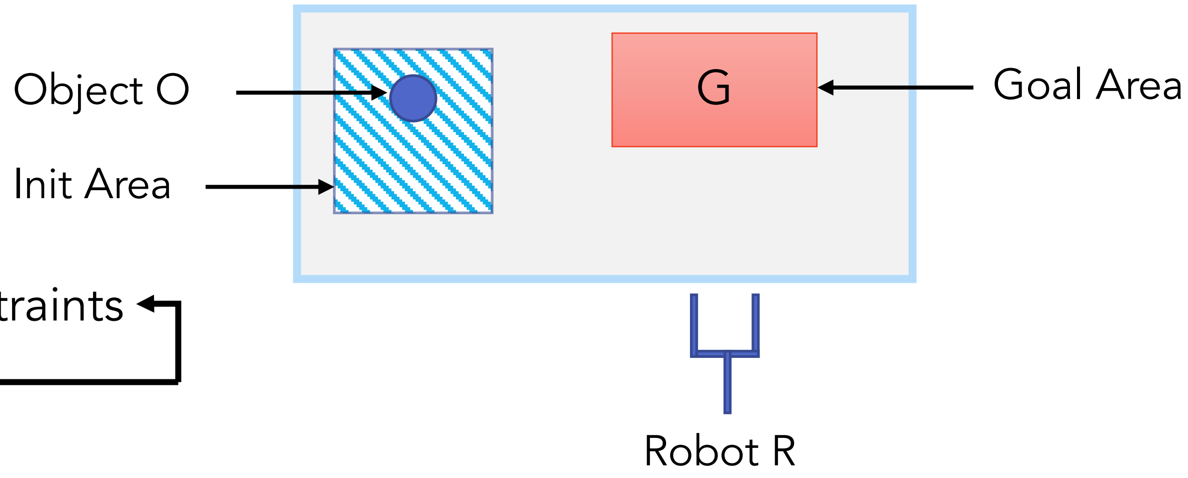
- Plan_length = 1
- Constraints = initial constraints
- Compute task plan for the current plan length and constraints

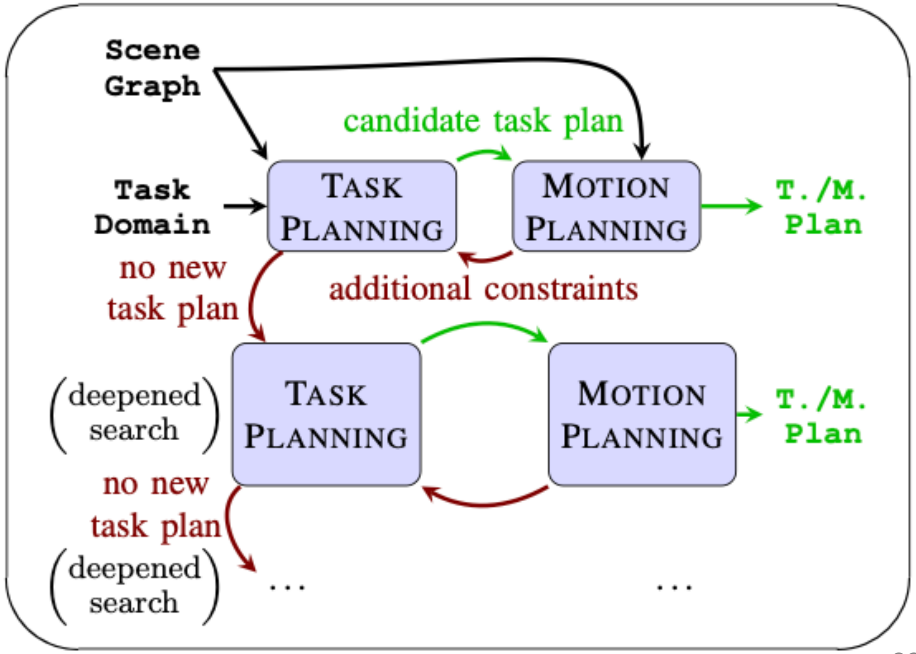




IDTMP: Overall Approach

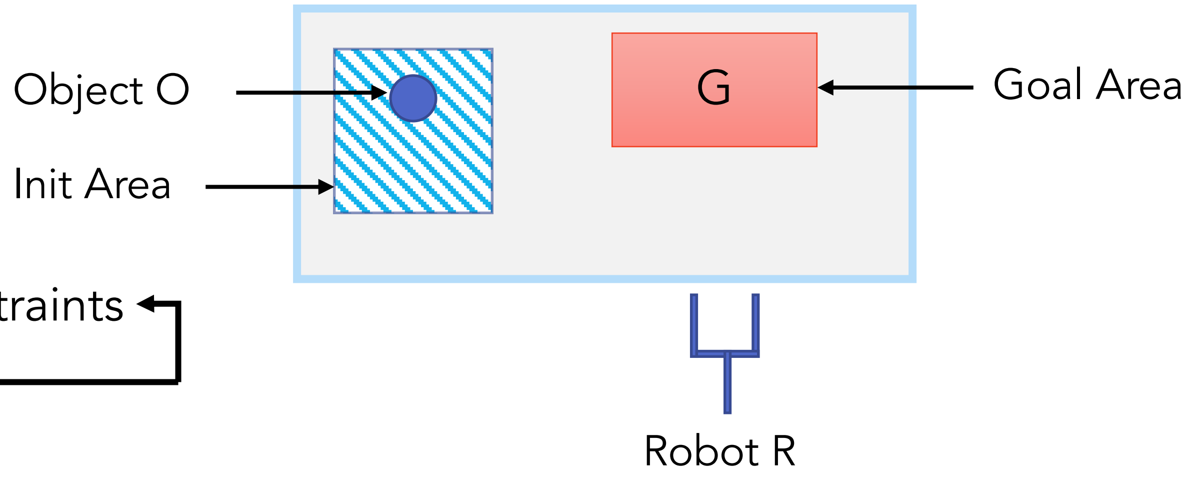
- Plan_length = 1
- Constraints = initial constraints
- Compute task plan for the current plan length and constraints
- If no task plan found: plan_length + 1 and retry

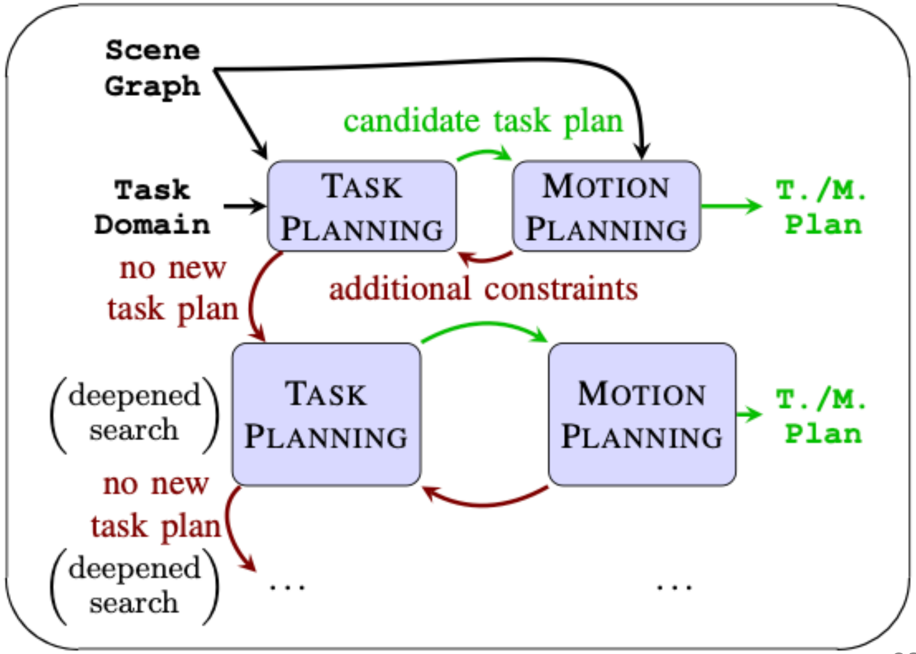




IDTMP: Overall Approach

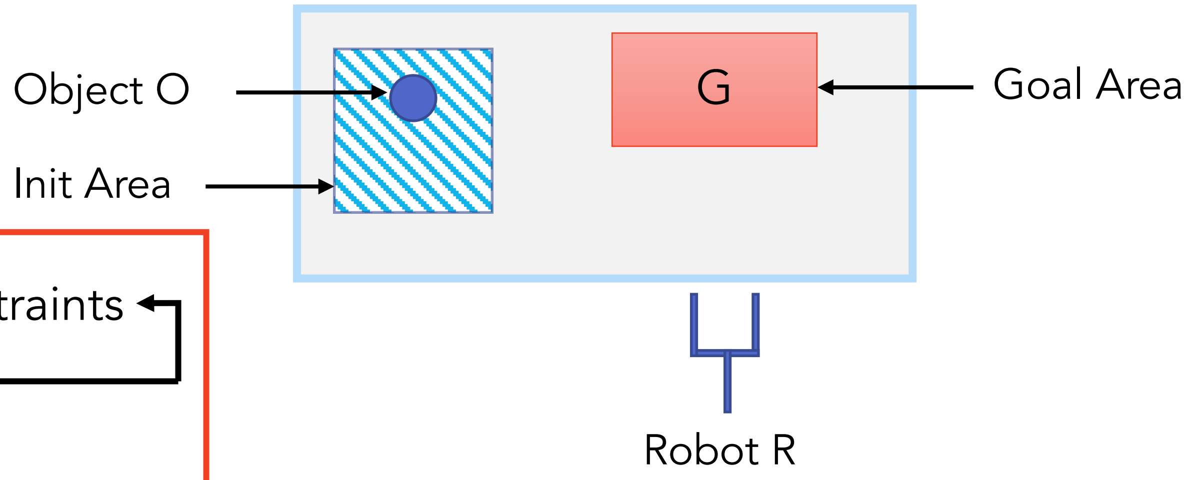
- Plan_length = 1
- Constraints = initial constraints
- Compute task plan for the current plan length and constraints
- If no task plan found: plan_length + 1 and retry
- If a task plan is found:
    - For every action in task plan:
        - Compute a motion plan

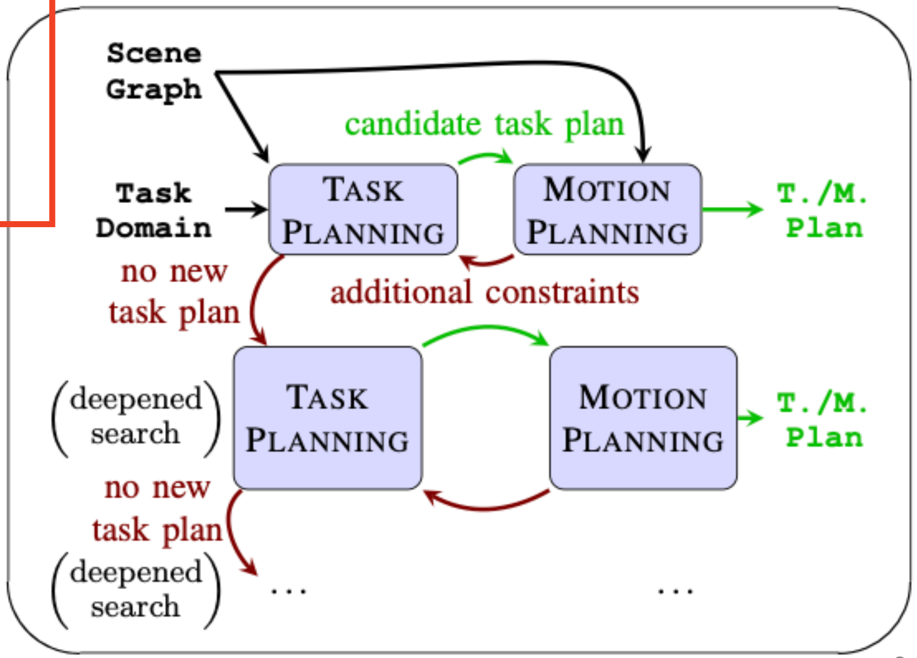




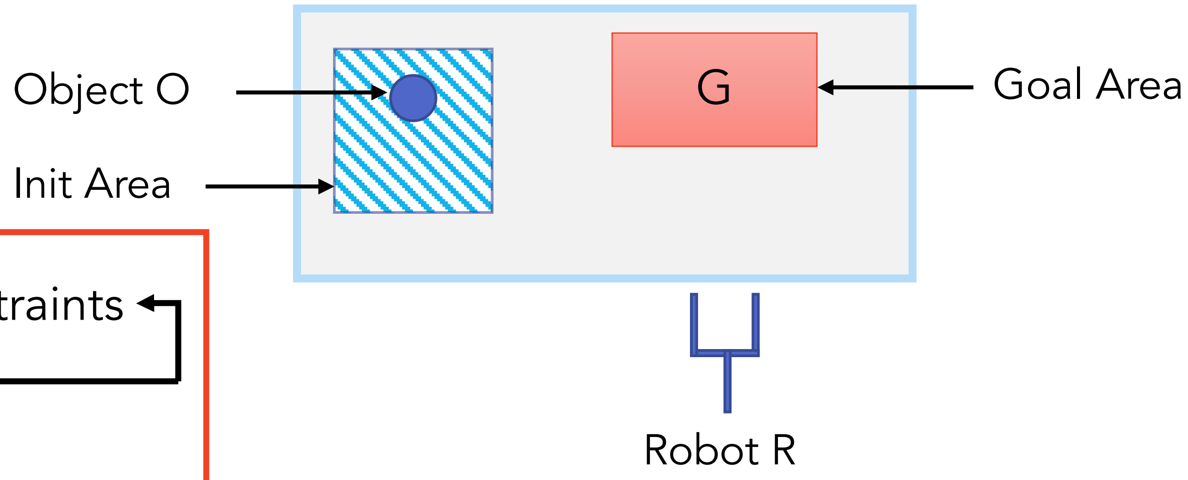
IDTMP: Overall Approach

- Plan_length = 1
- Constraints = initial constraints
- Compute task plan for the current plan length and constraints
- If no task plan found: plan_length + 1 and retry
- If a task plan is found:
  - For every action in task plan:
    - Compute a motion plan
      - If no motion plan:
        - Add new constraints

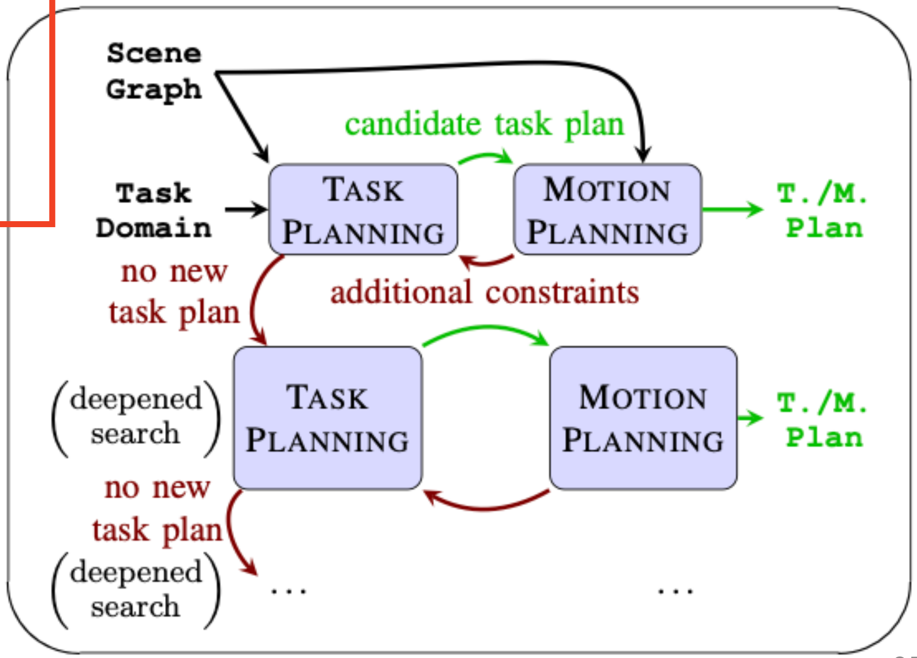




IDTMP: Overall Approach



- Plan_length = 1
- Constraints = initial constraints
- Compute task plan for the current plan length and constraints
- If no task plan found: plan_length + 1 and retry
- If a task plan is found:
  - For every action in task plan:
    - Compute a motion plan
      - If no motion plan:
        - Add new constraints



If a_i^k does not have a motion plan → disallow a_i at step k.

IDTMP: Experiments

Incremental Task and Motion Planning: A Constraint-Based Approach

Neil T. Dantam, Zachary K. Kingston,
Swarat Chaudhuri, and Lydia E. Kavraki

January 2016

RICE

IDTMP: Summary

- Inputs
  - SMT domain with hybrid (continuous and symbolic) variable and actions
- Properties
  - Probabilistically complete – if the low-level motion planner is probabilistically complete.
- How does it handle C1: SMTs reasoning for instantiating continuous variables
- How does it handle C2: SMTs reasoning for instantiations of discrete variables and adding new constraints

Nedunuri, Srinivas, et al. "SMT-based synthesis of integrated task and motion plans from plan outlines." *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014.

Dantam, N. T., Kingston, Z. K., Chaudhuri, S., & Kavraki, L. E. (2018). An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research*,

High-Level Summary

	Search Space	High Level Planner	Low Level Reasoning	High-level	High Level Language	P1: Infinite Motion Plans	P2: Downward Refinability
aSyMov	Single	Any TP	PRM/RRT	Symbolic	PDDL	N/A	N/A
IDTMP	Dual	SMT	Any MP	Variables with continuous domains	SAS	SMT	SMT

Outline

1. Background: Why Task and Motion Planning?
2. Abstraction as a Foundation for TMP
3. Abstraction-based Approaches
4. Research Frontier: Neuro-Symbolic Learning for TMP

Hierarchical Planning in the Now: HPN

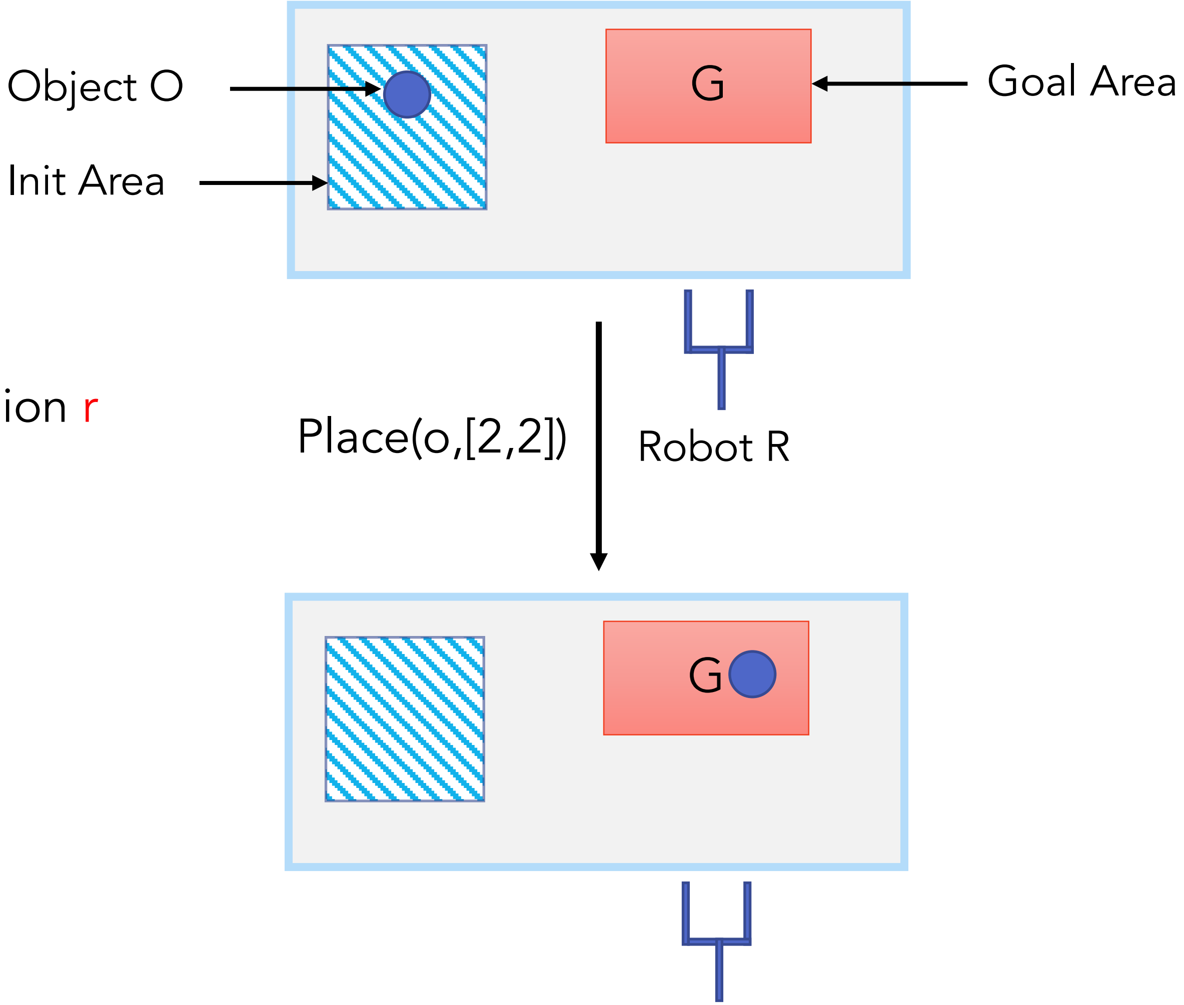
Central Idea:

- Don’t abstract the state; plan over fluents & actions with continuous arguments
- For high-level reasoning: Use regression of geometric fluents
- For efficiency: Use an operator-abstraction hierarchy
- For refinement: Interleave refinement with execution

Kaelbling, Leslie Pack, and Tomás Lozano-Pérez. "Hierarchical task and motion planning in the now." *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011.

HPN: States and Actions

- State represented by a set of fluents with possibly continuous arguments
  - Subroutines used to dynamically evaluate fluents
  - No need to represent complete state descriptions
- Examples of fluents with continuous arguments (1-d):
  - In(o, r): object o is completely inside region r
  - ObjLoc(o, l): left edge of object o is at location l
  - ClearX(r, x): only objects ∈ x possibly overlap with region r
- Actions:
  - **Place**(o, l_{target}) causes ObjLoc(o, l_{target});
    - requires ClearX(sweptVol(o, l_{init}, l_{target}))
  - **In**(o, r): ramification action for the fluent In(o, r)
  - **Clear**(r, x): ramification action for the fluent Clear(r, x)



HPN: Action Specifications

- Action specification without hierarchy:
  - Arguments include current subgoal (maintained during regression)
  - Effects, preconditions including argument-choice constraints

PickPlace((o, l_{target}), s_{now} , γ):

effect: ObjLoc(o, l_{target})
choose: l_{start} $\in$ {s_{now}[o].loc} $\cup$

generateLocsInRegions((o,{warehouse,stove,sink}),s_{now},γ)

//operator instantiations have to be considered for each generated value of l_{start}

//Allows l_{start} to be generated using γ for efficient regression

pre: ObjLoc(o, l_{start}), ClearX(sweptVol(o, l_{start},l_{target}),{o})

- Precondition + choose essentially generates the next subgoal during regression
- Use operator-specific regression subroutines for geometric fluents (provided as input)

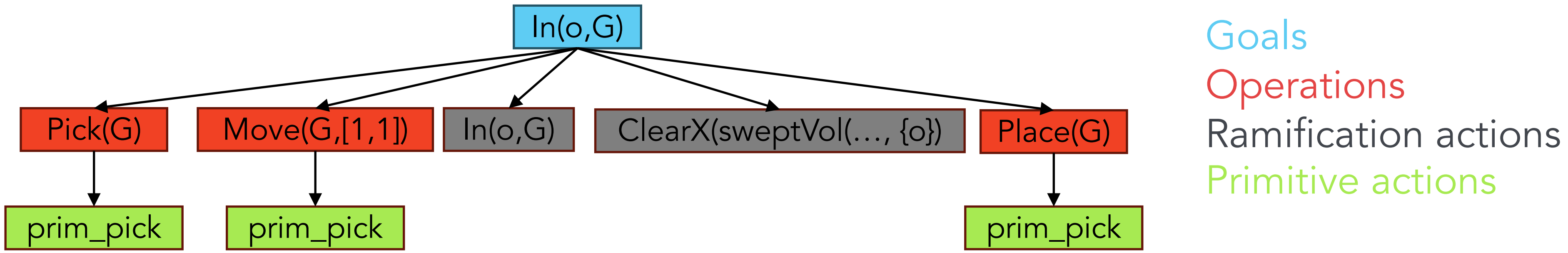
HPN: Regression Algorithm

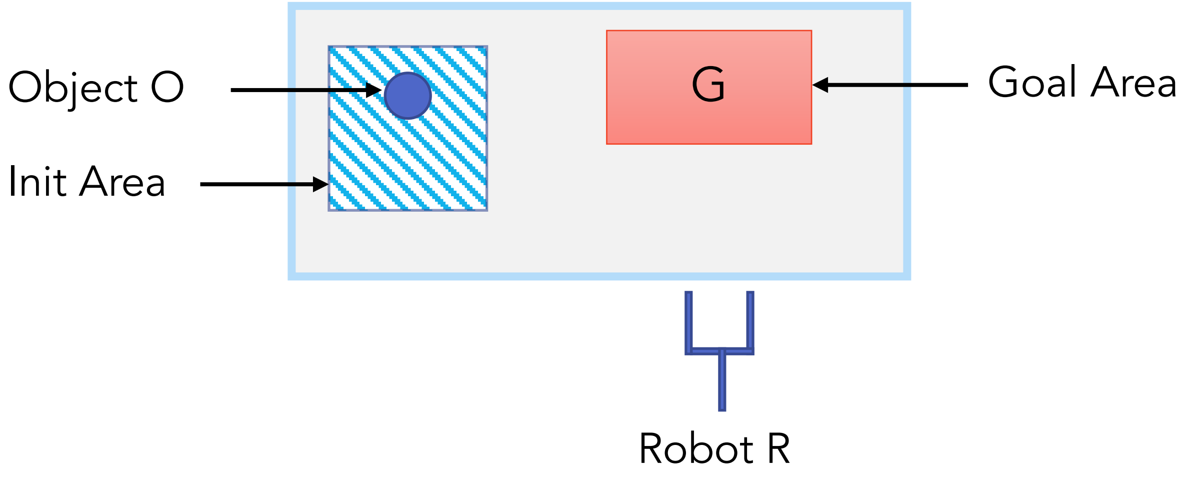
- Carry out goal regression using goal state, preimage computation methods for each operator, geometric fluent
- Search using A*; heuristic = number of goal fluents that are not true in the preimage

Pick((o),s,γ):
  effect: Holding(o)
  pre: AtGraspPose(o) [1]
  prim: prim_pick

Place((o),s,γ):
  effect: Holding(o) [1]
  pre: ¬Holding(o)
  prim: prim_place

Move((o,l_{end}),s,γ):
  choose: l_{start}
  effect: ObjectLoc(l_{end})
  pre: ObjectLoc(l_{start}) [1]
    Holding(o) [2]
    ClearX(sweptVol(o, l_{start}, l_{end}), {o}) [3]
  prim: prim_move





HPN: Hierarchical Action Representation

- Use a hierarchy defined using precondition postponement to reduce the horizon during regression
- Suppose operator o has preconditions $p_1, \ldots, p_n$, effect r
- Operator with p_n postponed
  - o_postponed:
    - precon = $p_1, \ldots, p_{n-1}$
    - expansion:
      - Achieve p_n while maintaining $p_1, \ldots, p_{n-1}$;
      - Then execute o
- Additional side-effects of achieving p_n may have to be declared
- Define hierarchy by associating abstraction-level with each precondition

Pick((o),s,γ):
  effect: Holding(o)
  pre: AtGraspPose(o) [1]
  prim: prim_pick

Place((o),s,γ):
  effect: Holding(o) [1]
  pre: ¬Holding(o)
  prim: prim_place

Move((o, l_{end}),s,γ):
  choose: l_{start}
  effect: ObjectLoc(l_{end})
  pre: ObjectLoc(l_{start}) [1]
    Holding(o) [2]
    ClearX(sweptVol(o, l_{start}, l_{end}), {o}) [3]
  prim: prim_move

HPN: Hierarchical Action Representation

- Use a hierarchy defined using precondition postponement to reduce the horizon during regression
- Suppose operator o has preconditions $p_1, \ldots, p_n$, effect r
- Operator with p_n postponed
  - o_postponed:
    - precon = $p_1, \ldots, p_{n-1}$
    - expansion:
      - Achieve p_n while maintaining $p_1, \ldots, p_{n-1}$;
      - Then execute o
- Additional side-effects of achieving p_n may have to be declared
- Define hierarchy by associating abstraction-level with each precondition

Pick((o),s,γ):
 effect: Holding(o)
 pre: AtGraspPose(o) [1]
 prim: prim_pick

Place((o),s,γ):
 effect: Holding(o) [1]
 pre: ¬Holding(o)
 prim: prim_place

Move((o, l_{end}),s,γ):
 choose: l_{start}
 effect: ObjectLoc(l_{end})
 pre: ObjectLoc(l_{start}) [1]
  Holding(o) [2]
  ClearX(sweptVol(o, l_{start}, l_{end}), {o}) [3]
 prim: prim_move

HPN: Main Algorithm

- Repeat depth-first refinement + execution
  - Primitives are executed as they are generated ("planning in the now")

$\textsc{hpn}(s_{now}, \gamma, \alpha, world)$:

$\quad p = \textsc{plan}(s_{now}, \gamma, \alpha)$

$\quad$ **for** (ω_i, g_i) **in** p

$\quad\quad$ **if** $\textsc{isPrim}(\omega_i)$

$\quad\quad\quad world.\textsc{execute}(\omega_i,\ s_{now})$

$\quad\quad$ **else**

$\quad\quad\quad \textsc{hpn}(s_{now},\ g_i,\ \textsc{nextLevel}(\alpha,\ \omega_i),\ world)$

HPN: Overall Approach

```
HPN(s_now, γ, α, world):
    p = PLAN(s_now, γ, α)
    for (ω_i, g_i) in p
        if ISPRIM(ω_i)
            world.EXECUTE(ω_i, s_now)
        else
            HPN(s_now, g_i, NEXTLEVEL(α, ω_i), world)
```

HPN: Overall Approach

Pick((o),s,γ):

effect: Holding(o)

pre: AtGraspPose(o) [1]

prim: prim_pick

Place((o),s,γ):

effect: ¬ Holding(o) [1]

pre: Holding(o)

prim: prim_place

Move((o,l_{end}),s,γ):

choose: l_{start}

effect: ObjectLoc(l_{end})

pre: ObjectLoc(l_{start}) [1]

Holding(o) [2]

ClearX(sweptVol(o, l_{start}, l_{end}), {o}) [3]

prim: prim_move

In((o,R),s,γ):

choose: l

effect: In(o, R)

pre: ObjectLoc(l) [1]

¬Holding(o) [2]

$\textsc{hpn}(s_{now}, \gamma, \alpha, world)$:

$p = \textsc{plan}(s_{now}, \gamma, \alpha)$

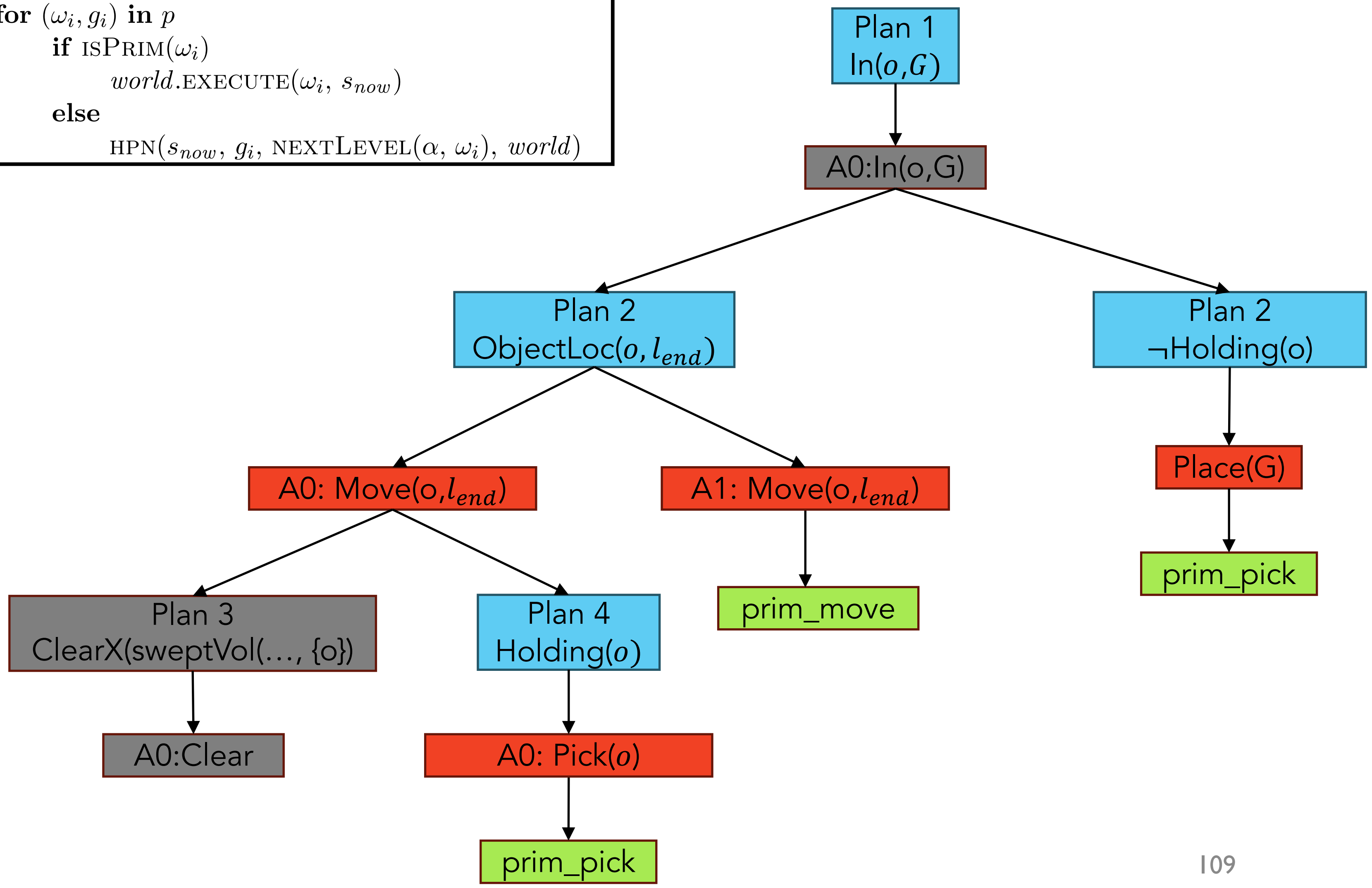
for (ω_i, g_i) **in** p

if $\textsc{isPrim}(\omega_i)$

$world.\textsc{execute}(\omega_i,\ s_{now})$

else

$\textsc{hpn}(s_{now},\ g_i,\ \textsc{nextLevel}(\alpha,\ \omega_i),\ world)$



HPN: Overall Approach

Pick((o),s,γ):

effect: Holding(o)

pre: AtGraspPose(o) [1]

prim: prim_pick

Place((o),s,γ):

effect: ¬ Holding(o) [1]

pre: Holding(o)

prim: prim_place

Move((o,l_{end}),s,γ):

choose: l_{start}

effect: ObjectLoc(l_{end})

pre: ObjectLoc(l_{start}) [1]

Holding(o) [2]

ClearX(sweptVol(o, l_{start}, l_{end}), {o}) [3]

prim: prim_move

In((o,R),s,γ):

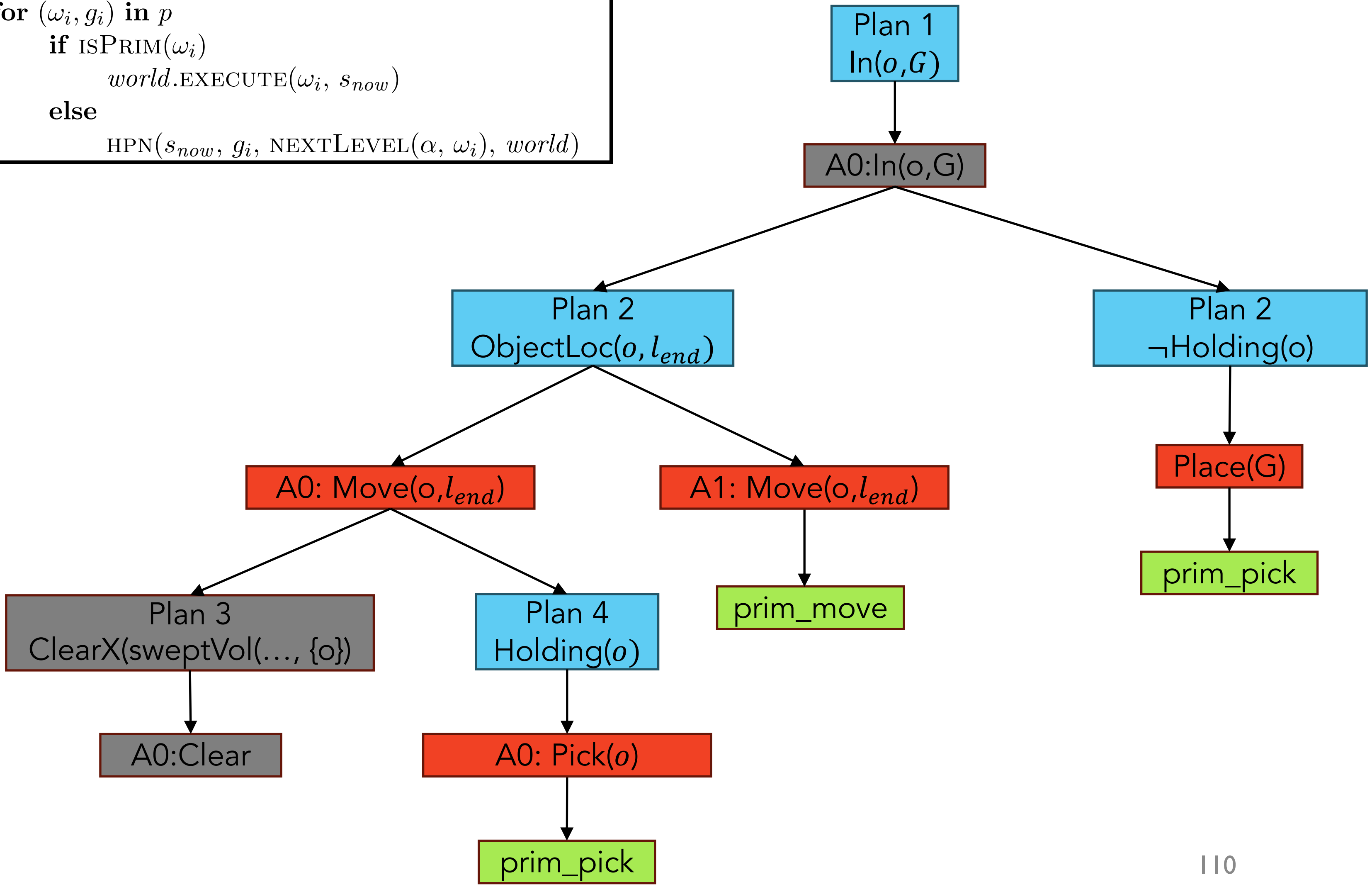
choose: l

effect: In(o, R)

pre: ObjectLoc(l) [1]

¬Holding(o) [2]

```
HPN(s_now, γ, α, world):
    p = PLAN(s_now, γ, α)
    for (ω_i, g_i) in p
        if ISPRIM(ω_i)
            world.EXECUTE(ω_i, s_now)
        else
            HPN(s_now, g_i, NEXTLEVEL(α, ω_i), world)
```



HPN: Overall Approach

Pick$((o), s, \gamma)$:

effect: Holding(o)

pre: AtGraspPose(o) [1]

prim: prim_pick

Place$((o), s, \gamma)$:

effect: ¬ Holding(o) [1]

pre: Holding(o)

prim: prim_place

Move$((o, l_{end}), s, \gamma)$:

choose: l_{start}

effect: ObjectLoc(l_{end})

pre: ObjectLoc(l_{start}) [1]

Holding(o) [2]

ClearX(sweptVol(o, l_{start}, l_{end}), {o}) [3]

prim: prim_move

In$((o, R), s, \gamma)$:

choose: l

effect: In(o, R)

pre: ObjectLoc(l) [1]

¬Holding(o) [2]

$\text{HPN}(s_{now}, \gamma, \alpha, \mathit{world})$:

$p = \text{PLAN}(s_{now}, \gamma, \alpha)$

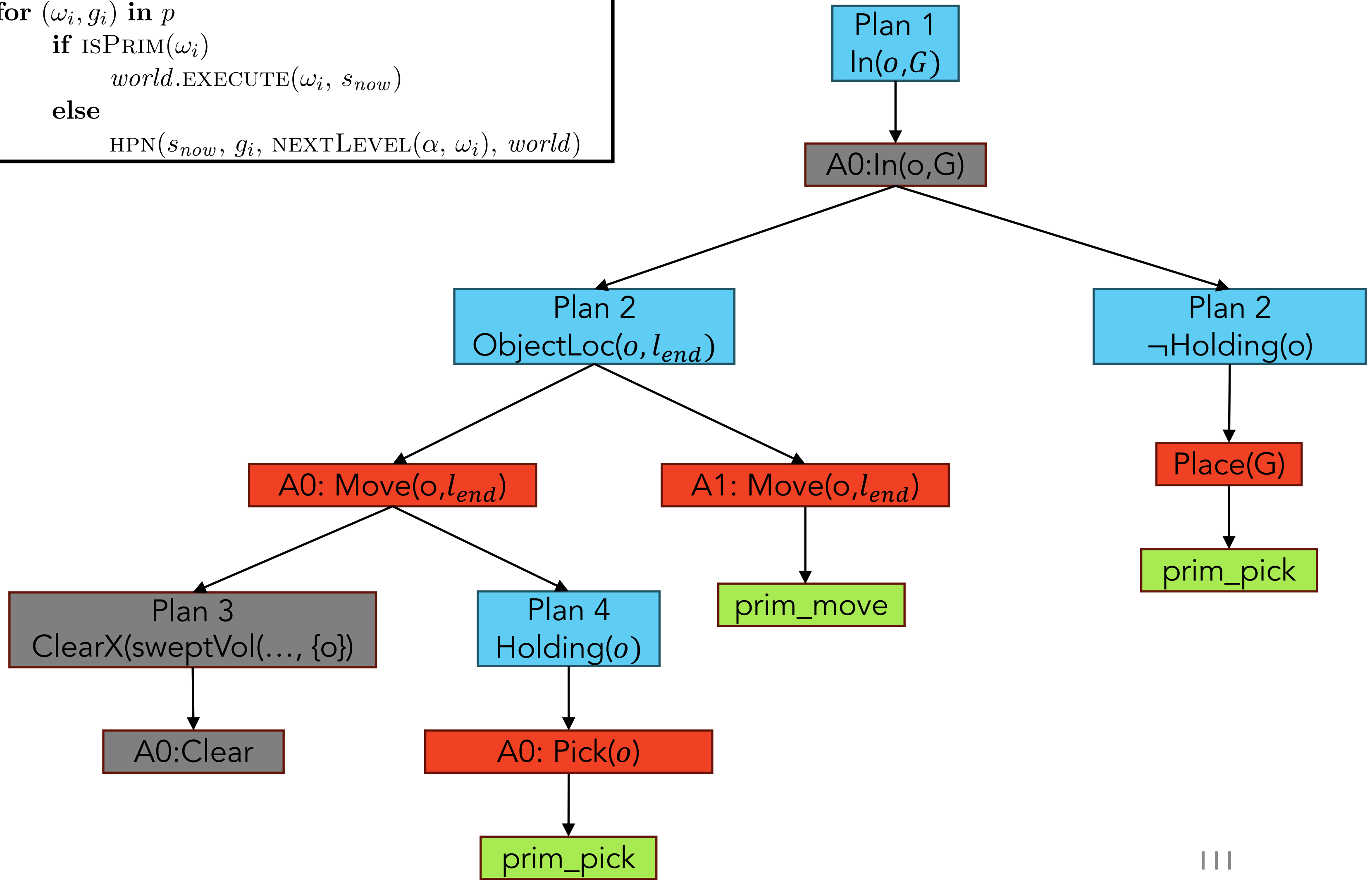
for (ω_i, g_i) **in** p

if $\text{ISPRIM}(\omega_i)$

$\mathit{world}.\text{EXECUTE}(\omega_i,\ s_{now})$

else

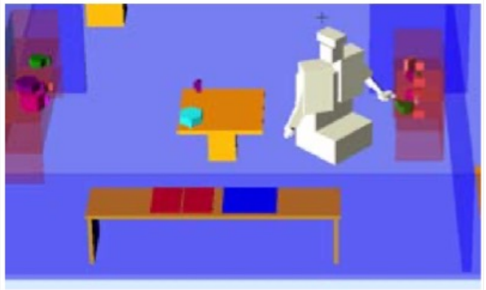
$\text{HPN}(s_{now},\ g_i,\ \text{NEXTLEVEL}(\alpha,\ \omega_i),\ \mathit{world})$



HPN: Algorithmic Details & Optimizations

- Use generators to make choice of arguments more efficient
- Algorithm commits to subgoals generated at higher level of abstraction when refining
  - Need to ensure subgoal feasibility
  - Approximate the generation of feasible results of “choose” operations using limited logical reasoning
    - Ensures logical consistency of fluents based on domain-specific integrity constraints
- Achieving postponed preconditions can lead to additional effects in abstract operators
  - Can declare approximations/conservative versions of side-effects with actions

HPN: Experiments



(a) A plate (intended for the first food item) has been placed on the table, and the robot is getting food from refrigerator.

(d) Starting to tidy up; both pans are in the sink.

(f) In order to enable picking up the pink cup in the right warehouse, the objects in the warehouse are re-arranged.

(h) All the objects in the right warehouse have been placed in refrigerator; two dirty cups left out on various tables are in the sink.

HPN: Summary

- Input:
  - Correct and complete primitive action definitions using geometric predicates
  - Operator-specific regression functions for geometric properties
  - Pose generators
  - For efficiency, can use additional input:
    - Pose generators that make use of current subgoal
    - Precondition levels to obtain a hierarchy, declaration of operator side-effects
    - Limited logical reasoning in pose generators
- Properties:
  - Complete if the problem has no dead-ends, action preconditions are accurate
    - Motion planners terminate and return solutions when preconditions hold
- Approach for C1: through generators and regression (backward-search)
- Approach for C2: Regression over geometric fluents

Kaelbling, Leslie Pack, and Tomás Lozano-Pérez. "Hierarchical task and motion planning in the now." *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011.

High-Level Summary

	Search Space	High Level Planner	Low Level Reasoning	High-level	High Level Language	P1: Infinite Motion Plans	P2: Downward Refinability
aSyMov	Single	Any TP	PRM/RRT	Symbolic	PDDL	N/A	N/A
IDTMP	Dual	SMT	Any MP	Variables with continuous domains	SAS	SMT	SMT
HPN	Dual	HPN-specific regression planner	Any MP	Hybrid	HPN-specific	generators	Regression over geometric fluents

TMP through an Interface Layer

- Geometric planning is hard → Symbolic high-level
- Off-the-shelf task planner
- Off-the-shelf motion planner
- Interface layer that communicates between task planner and motion planner
  - Converts each task level action to a motion planning problem
  - Converts motion planning failures as facts over symbols & refines abstract information

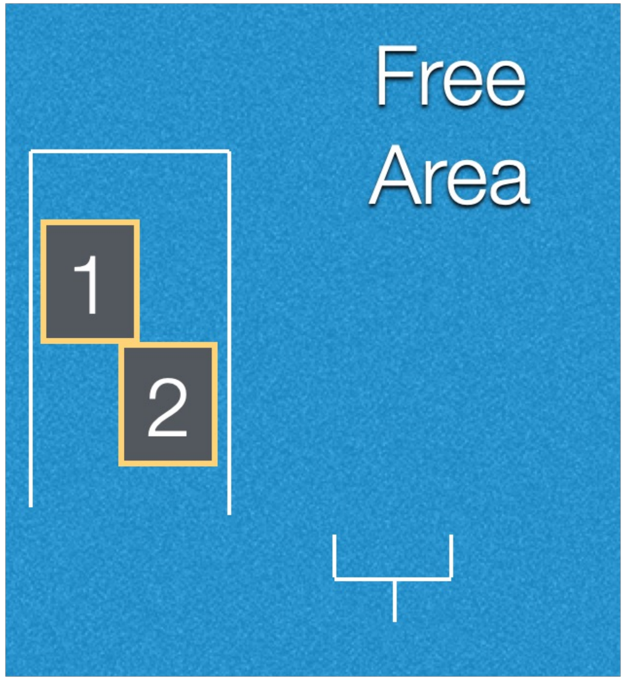
Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., & Abbeel, P. (2014, May). Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*

TMP through an Interface Layer: Example

PickUp(obj o1, pose p1, pose p2, pose p3, path p):

precondition: Empty(gripper) ∧ GripperAt(p1) ∧
At(o1, p3) ∧ IsGraspingPose(p2, o1, p3)
∧ path(p, p1, p2) ∧ ∀o2 ¬ *Obstructs*(o2, p)

effect: Holding(o1) ∧ ¬ At(o1, p3) ∧
¬ Empty(gripper) ∧ GripperAt(p2)



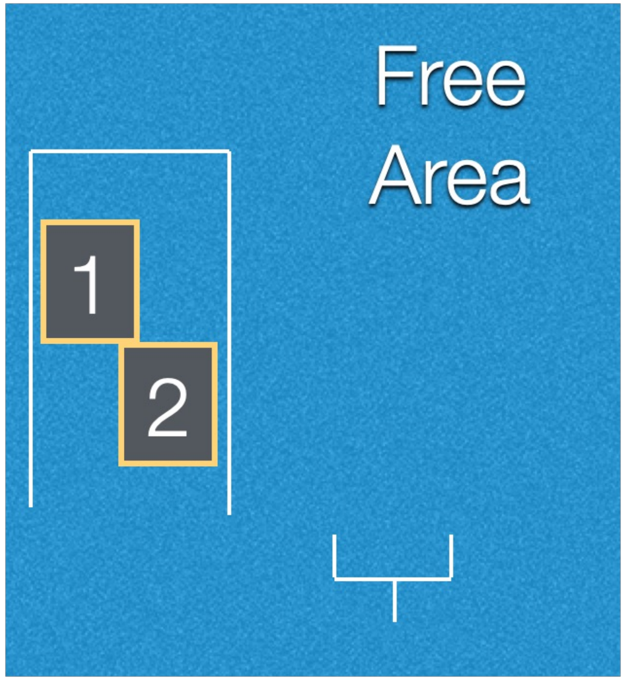
TMP through an Interface Layer: Example

G

PickUp(obj o1, pose p1, pose p2, pose p3, path p):

precondition: Empty(gripper) ∧ GripperAt(p1) ∧
At(o1, p3) ∧ IsGraspingPose(p2, o1, p3)
∧ path(p, p1, p2) ∧ ∀o2 ¬ *Obstructs*(o2, p)

effect: Holding(o1) ∧ ¬ At(o1, p3) ∧
¬ Empty(gripper) ∧ GripperAt(p2)



TMP through an Interface Layer: Example

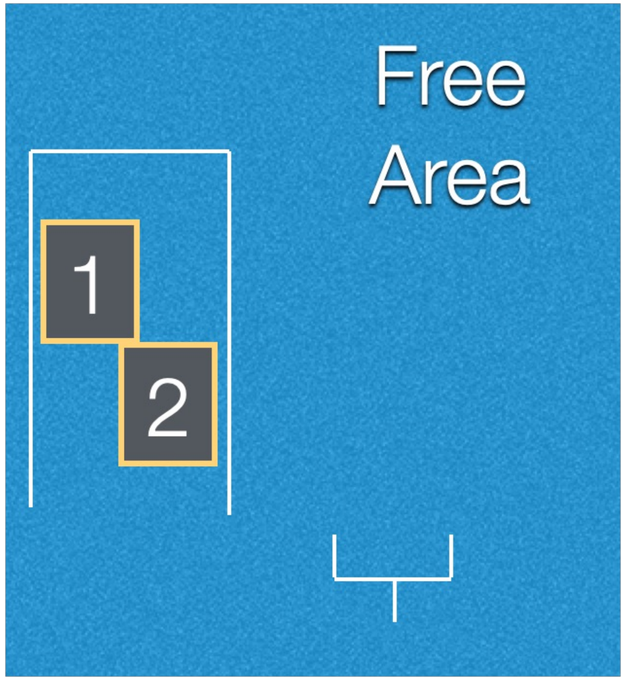
PickUp(obj o1, **pose p1, pose p2, pose p3, path p**):

precondition: Empty(gripper) ∧ GripperAt(p1) ∧
At(o1, p3) ∧ IsGraspingPose(p2, o1, p3)
∧ path(p, p1, p2) ∧ ∀o2 ¬ *Obstructs*(o2, p)

effect: Holding(o1) ∧ ¬ At(o1, p3) ∧
¬ Empty(gripper) ∧ GripperAt(p2)

Symbolic references

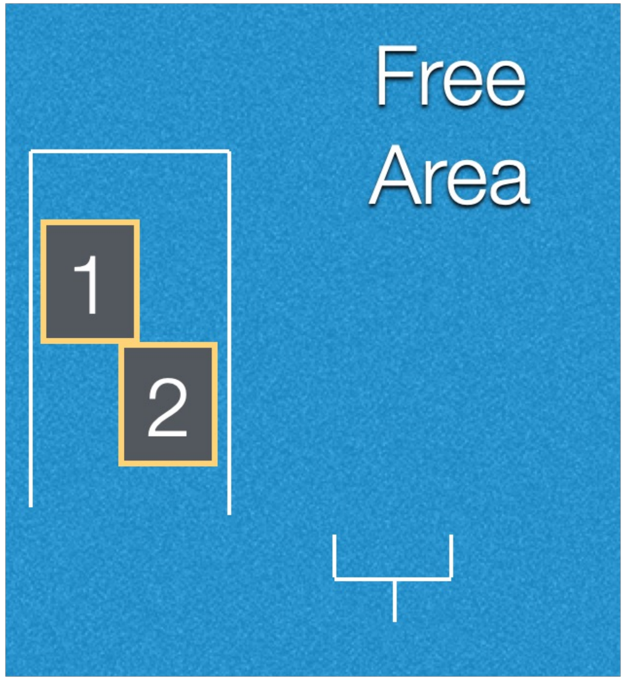
- High level intuitive plan:
  - pick block1 after going to *block1's grasping pose* along *a trajectory*



TMP through an Interface Layer: Example

Interface level:
Searches for an instantiation of block1's grasping pose that is reachable via a feasible (collision-free) trajectory

…finds no feasible trajectory



Symbolic references

- High level intuitive plan:
  - pick block1 after going to *block1's grasping pose* along *a trajectory*

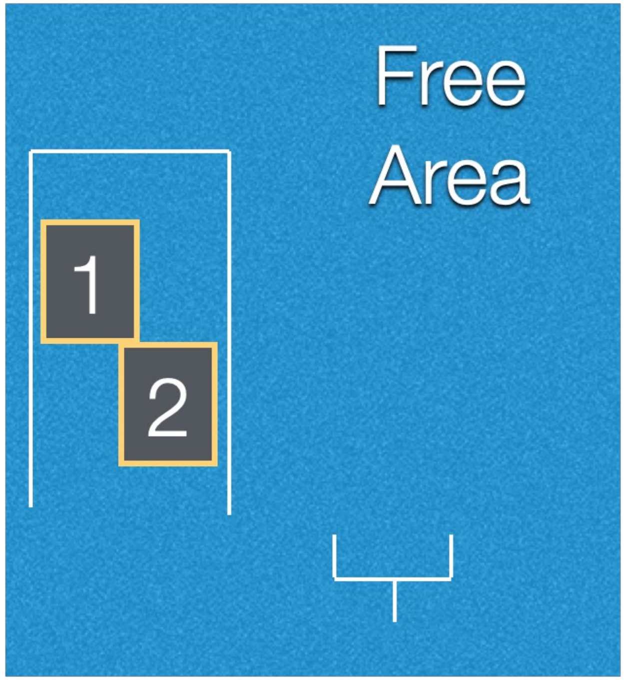
TMP through an Interface Layer: Example

Interface level:
Searches for an instantiation of block1's grasping pose that is reachable via a feasible (collision-free) trajectory

…finds no feasible trajectory

Symbolic references

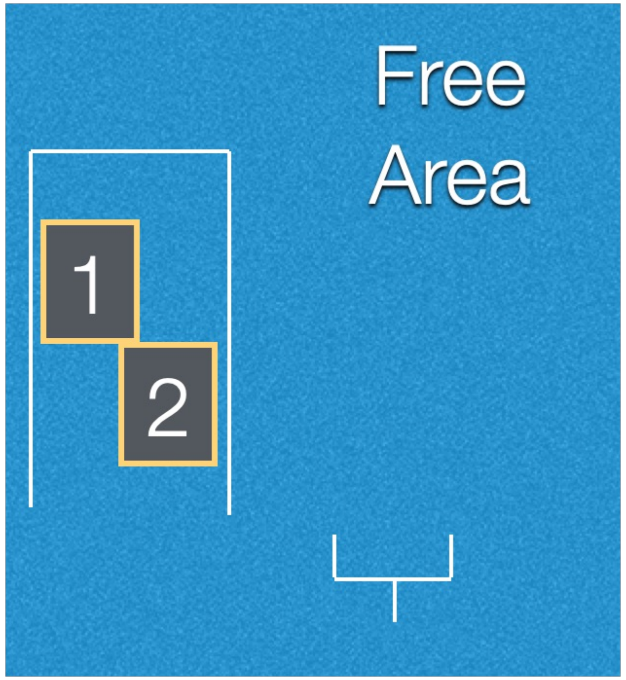
- High level intuitive plan:
  - ~~pick block1 after going to *block1's grasping pose* along *a trajectory*~~



Fix values for references, report reason for failure:
"block2 obstructs *block1's grasping pose* along *a trajectory*"

TMP through an Interface Layer: Example

PickUp(obj o1, **pose p1, pose p2, pose p3, path p**):
precondition: Empty(gripper) ∧ GripperAt(p1) ∧
At(o1, p3) ∧ IsGraspingPose(p2, o1, p3)
∧ path(p, p1, p2) ∧ ∀o2 ¬ *Obstructs*(o2, p)
effect: Holding(o1) ∧ ¬ At(o1, p3) ∧
¬ Empty(gripper) ∧ GripperAt(p2)



Symbolic references

- High level intuitive plan:
  - ~~pick block1 after going to *block1's grasping pose* along *a trajectory*~~

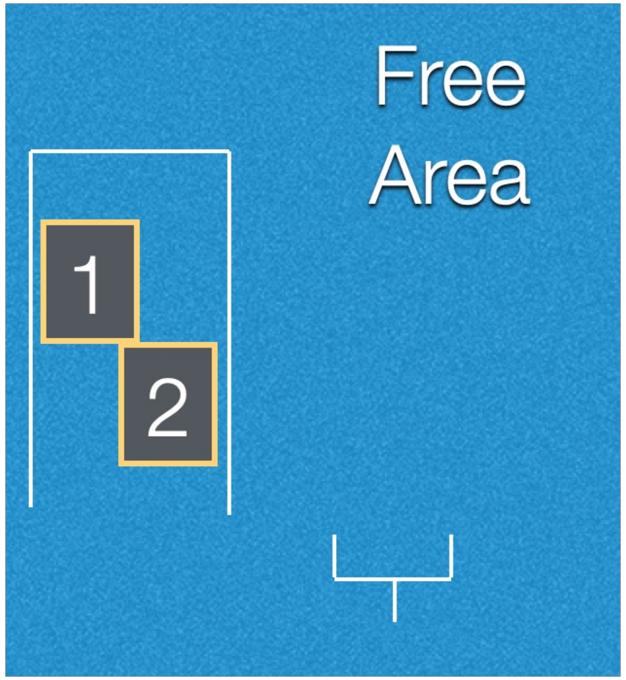
Fix values for references, report reason for failure:
"block2 obstructs *block1's grasping pose* along *a trajectory*"

TMP through an Interface Layer: Example

PickUp(obj o1, **pose p1, pose p2, pose p3, path p**):

precondition: Empty(gripper) ∧ GripperAt(p1) ∧
At(o1, p3) ∧ IsGraspingPose(p2, o1, p3)
∧ path(p, p1, p2) ∧ ∀o2 ¬ *Obstructs*(o2, p)

effect: Holding(o1) ∧ ¬ At(o1, p3) ∧
¬ Empty(gripper) ∧ GripperAt(p2)



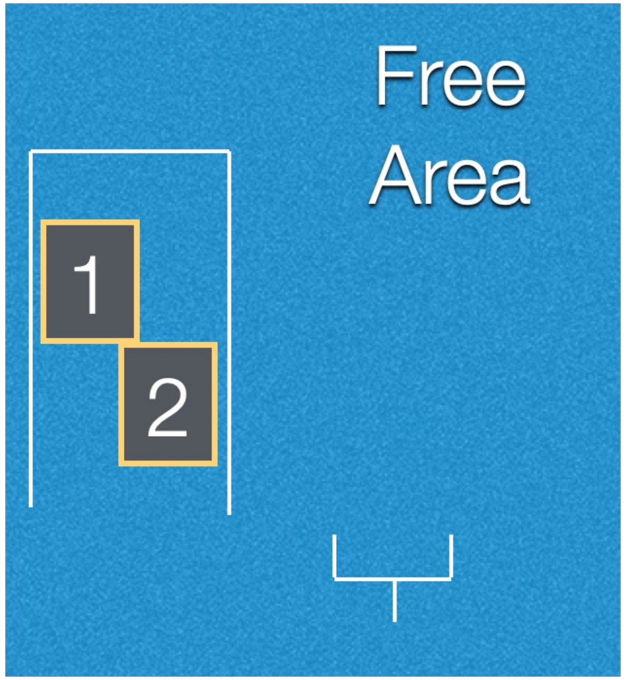
Discrete state += Obstructs(block2, path(initLoc, gp(block1))

TMP through an Interface Layer: Example

PickUp(obj o1, **pose p1, pose p2, pose p3, path p**):

precondition: Empty(gripper) ∧ GripperAt(p1) ∧
At(o1, p3) ∧ IsGraspingPose(p2, o1, p3)
∧ path(p, p1, p2) ∧ ∀o2 ¬ *Obstructs*(o2, p)

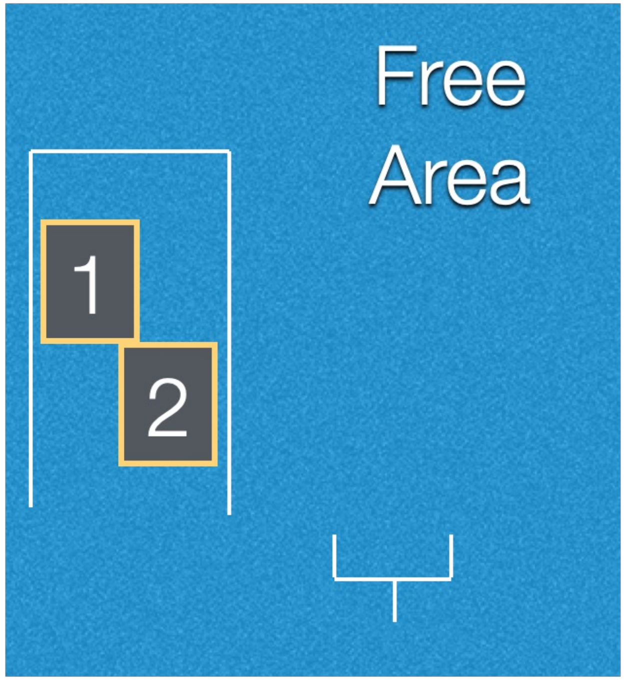
effect: Holding(o1) ∧ ¬ At(o1, p3) ∧
¬ Empty(gripper) ∧ GripperAt(p2)



Discrete state += Obstructs(block2, path(initLoc, gp(block1))

TMP through an Interface Layer: Example

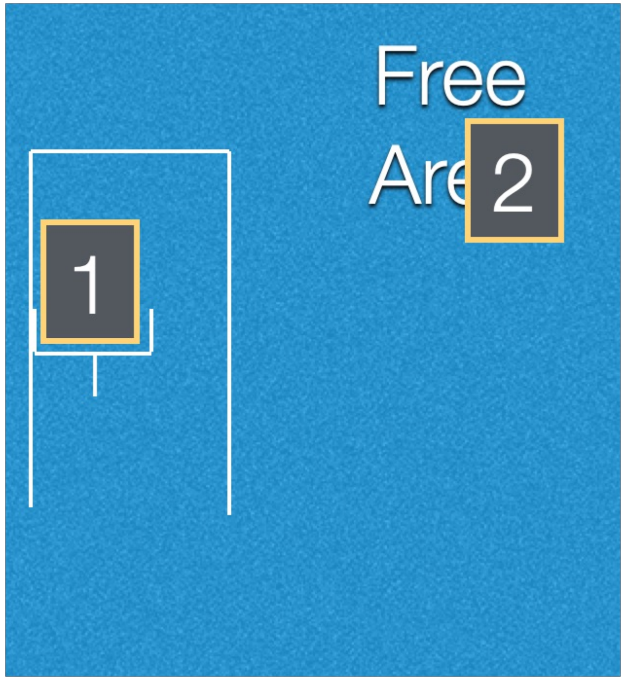
PickUp(obj o1, **pose p1, pose p2, pose p3, path p**):
precondition: Empty(gripper) ∧ GripperAt(p1) ∧
At(o1, p3) ∧ IsGraspingPose(p2, o1, p3)
∧ path(p, p1, p2) ∧ ∀o2 ¬ *Obstructs*(o2, p)
effect: Holding(o1) ∧ ¬ At(o1, p3) ∧
¬ Empty(gripper) ∧ GripperAt(p2)



- High level intuitive plan:
    - ~~pick block1 after going to *block1's grasping pose…*~~
- REPLAN
    - pick block2 after going to *block2's grasping pose…*
    - release block2 after going to *release pose for free area…*
    - pick block1 after going to *block1's grasping pose…*

TMP through an Interface Layer: Example

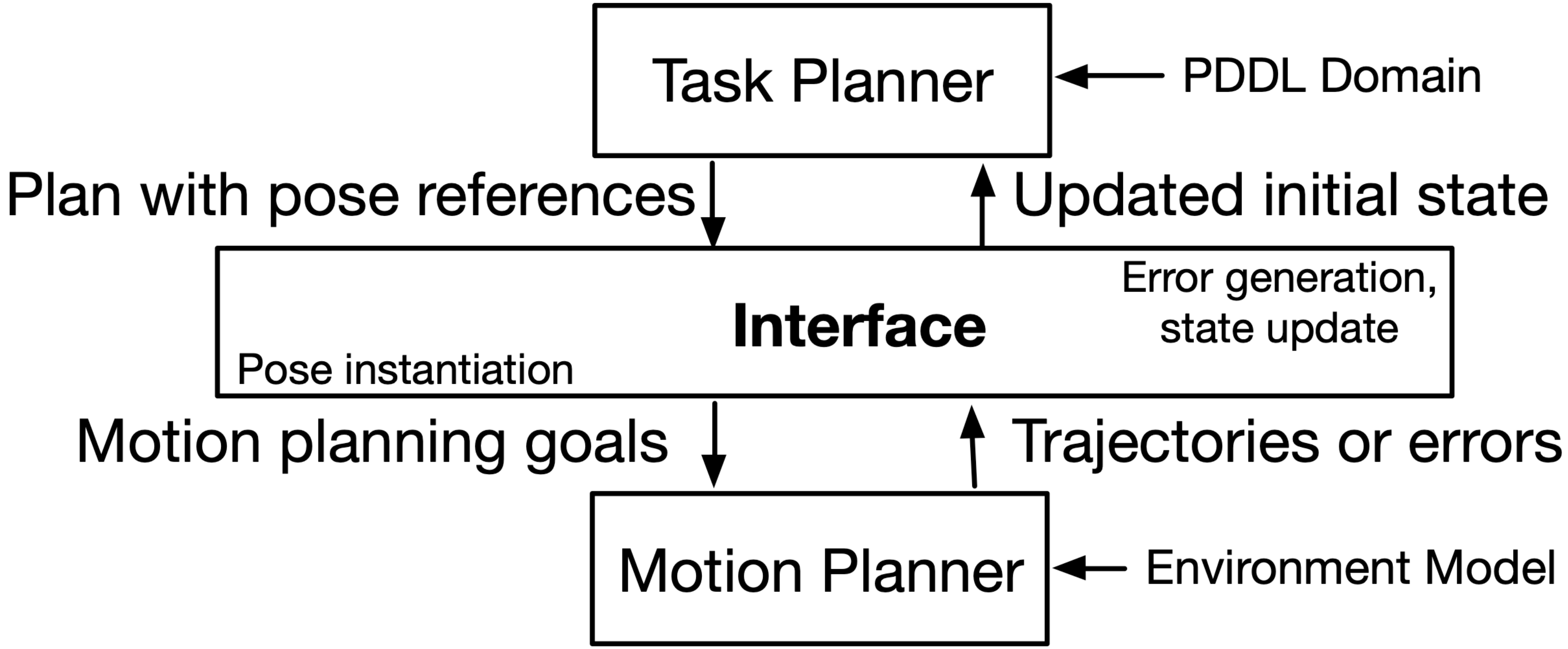
PickUp(obj o1, **pose p1, pose p2, pose p3, path p**):
precondition: Empty(gripper) ∧ GripperAt(p1) ∧
At(o1, p3) ∧ IsGraspingPose(p2, o1, p3)
∧ path(p, p1, p2) ∧ ∀o2 ¬ *Obstructs*(o2, p)
effect: Holding(o1) ∧ ¬ At(o1, p3) ∧
¬ Empty(gripper) ∧ GripperAt(p2)



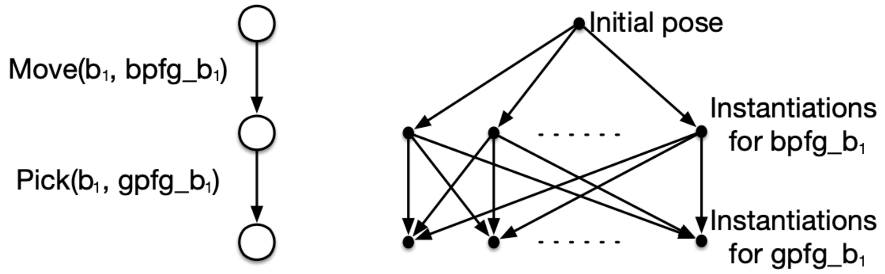
Goal Reached!!

- High level intuitive plan:
  - ~~pick block1 after going to *block1's grasping pose…*~~
- REPLAN
  - pick block2 after going to *block2's grasping pose…*
  - release block2 after going to *release pose for free area…*
  - pick block1 after going to *block1's grasping pose…*

TMP through an Interface Layer: Complete Algorithm

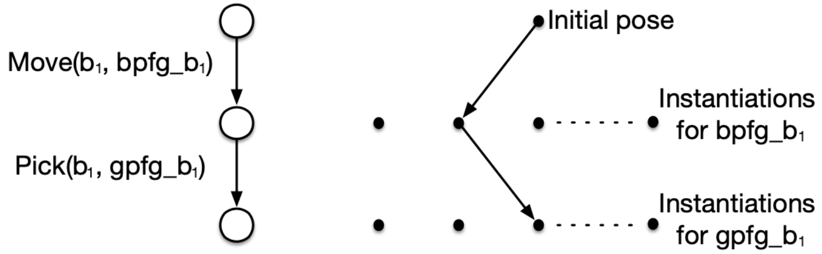


TMP through an Interface Layer: Complete Algorithm

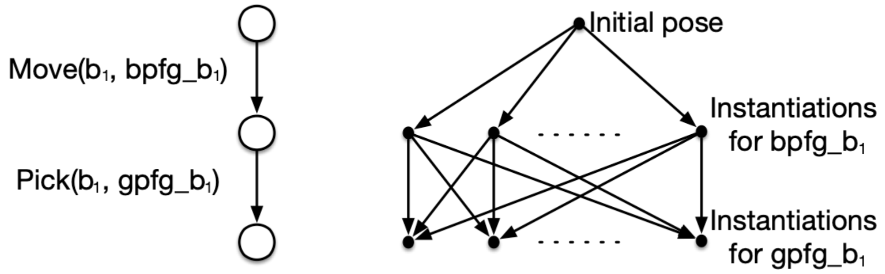


Iteratively try all instantiations

Stop when a valid instantiation is found for all the symbolic references.

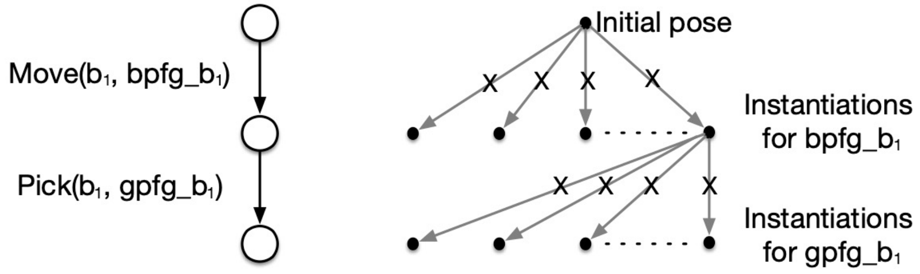


TMP through an Interface Layer: Complete Algorithm

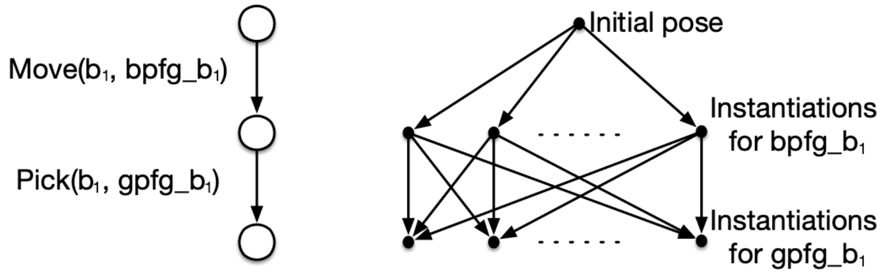


Iteratively try all instantiations

What if that fails?

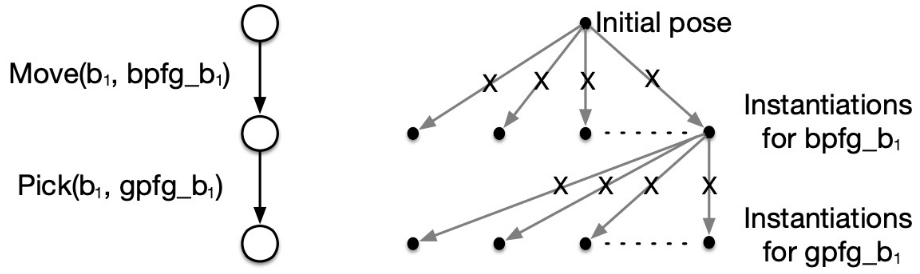


TMP through an Interface Layer: Complete Algorithm

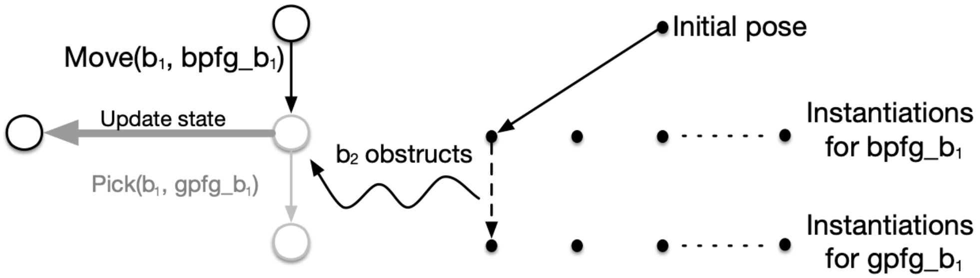


Iteratively try all instantiations

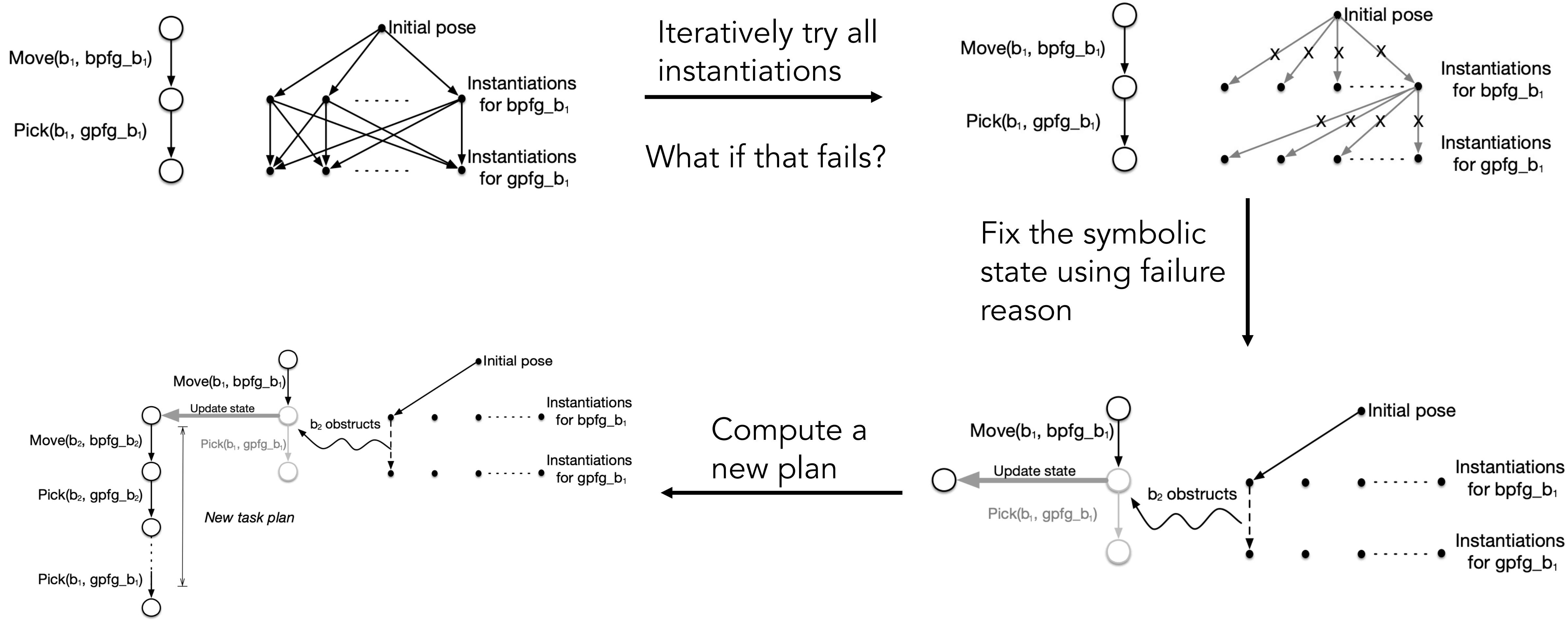
What if that fails?



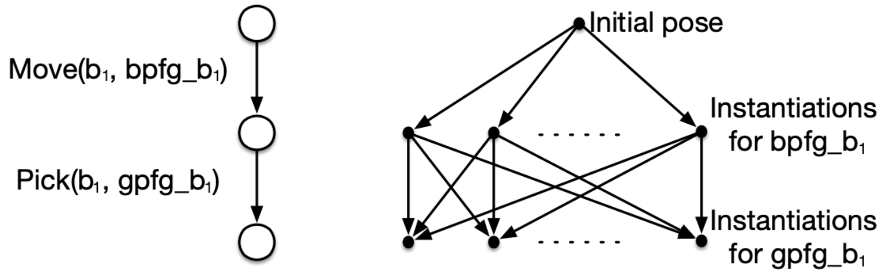
Fix the symbolic state using failure reason



TMP through an Interface Layer: Complete Algorithm

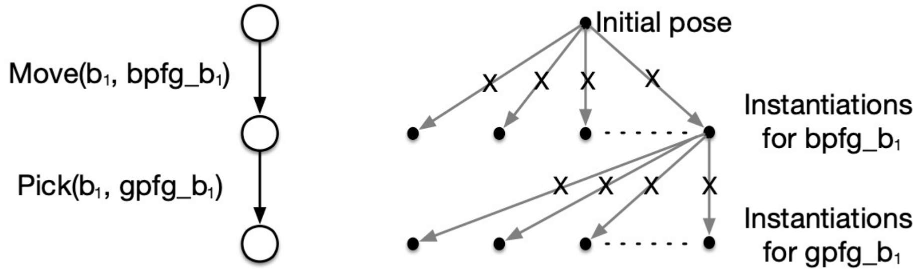


TMP through an Interface Layer: Complete Algorithm

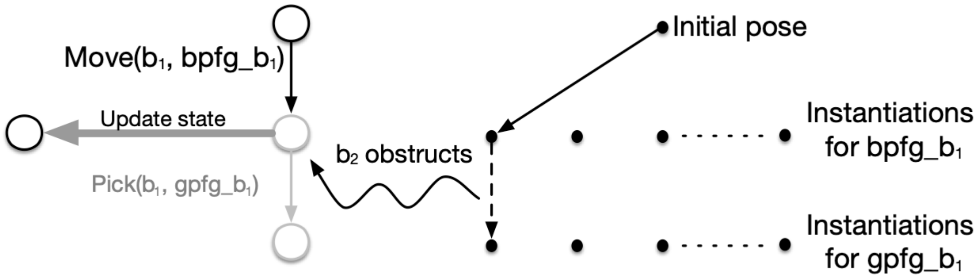


Iteratively try all instantiations

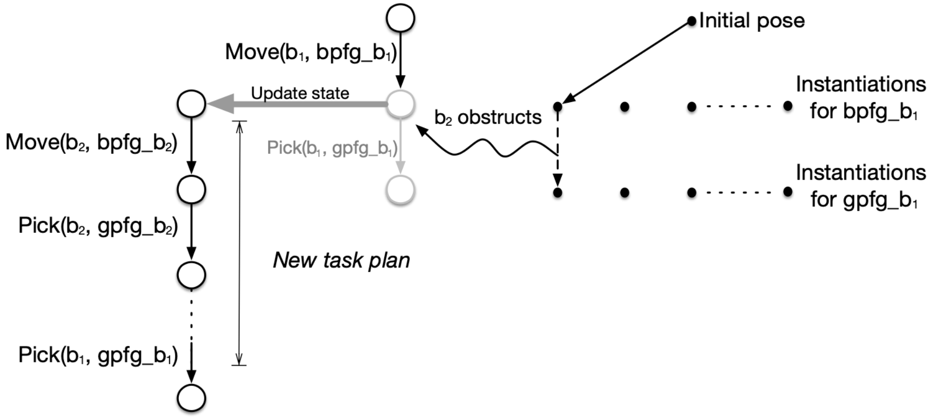
What if that fails?



Fix the symbolic state using failure reason

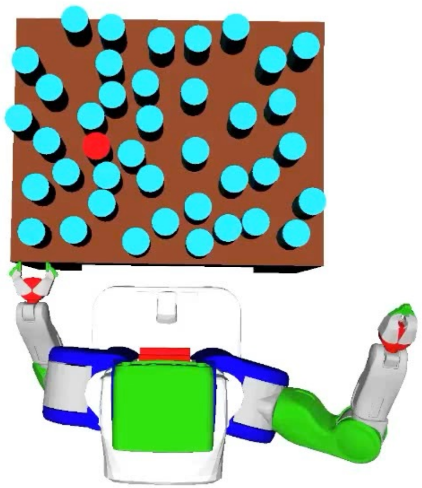


Compute a new plan



TMP through an Interface Layer: Experiments

- Several objects obstruct the target object
- Most of these objects are themselves obstructed by other objects
- No designated free space
- Geometric predicates constructed by the interface layer: obstructs(pose, obj1, obj2)



TMP through an Interface Layer: Experiments

- 3 pairs of noodle & soup bowls, predefined destinations
- Tray available, but utilization not
- Task planner has to decide whether to use the tray based on plan costs
- Geometric predicates constructed by interface layer:
  - smaller(obj1, obj2);
  - wrong_side(gripper, pose) — used for determining which hand to use

TMP through an Interface Layer: Summary

- Inputs
  - PDDL domain with references instead of continuous values
  - Generators for instantiating references
  - Subroutines for infeasibility detection and expression, given a motion plan
- Properties
  - Probabilistically complete
  - Each task planner invocation gets
    - Small branching factor
    - States have only a relevant subset of facts for current instantiation
- How does it handle P1: through symbolic references, forward-search, and lazily invoking the motion planner
- How does it handle P2: by communicating errors to the high-level planner and computing new plans

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., & Abbeel, P. (2014, May). Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*

High-level Summary

	Search Space	High Level Planner	Low Level Reasoning	High-level	High Level Language	P1: Infinite Motion Plans	P2: Downward Refinability
aSyMov	Single	Any TP	PRM/RRT	Symbolic	PDDL	N/A	N/A
IDTMP	IDTMP	Dual	SMT	Any MP	Variables with continuous domains	SAS	SMT
HPN	Dual	HPN-specific regression planner	Any MP	Hybrid	HPN-specific	generators	Regression over geometric fluents
Symbolic Interface	Dual	Any TP	Any MP	Symbolic	PDDL	Pose generators	Identifying errors

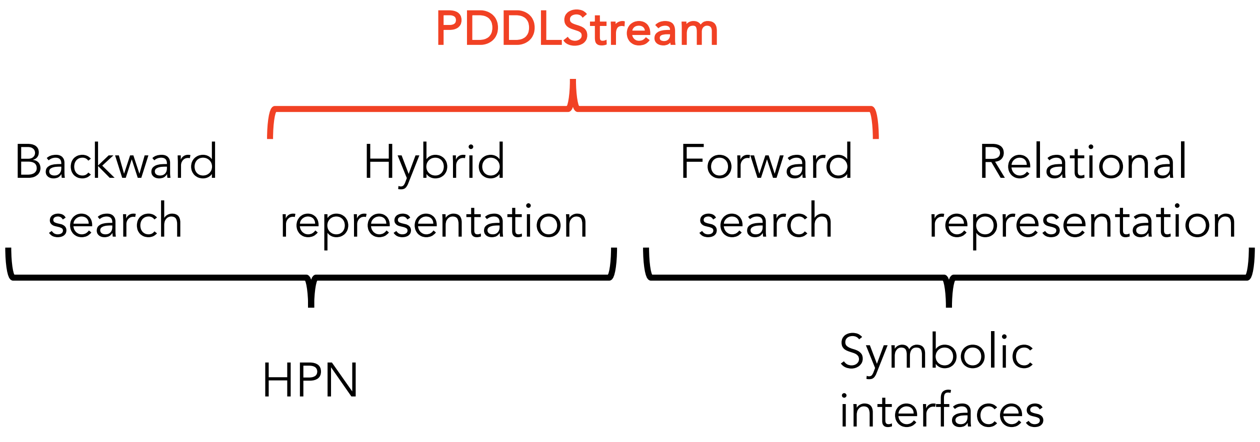
PDDLStream

- Forward-search over hybrid representation
- Modifies the PDDL representation

Garrett, C. R., Lozano-Pérez, T., & Kaelbling, L. P. (2020, June). Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

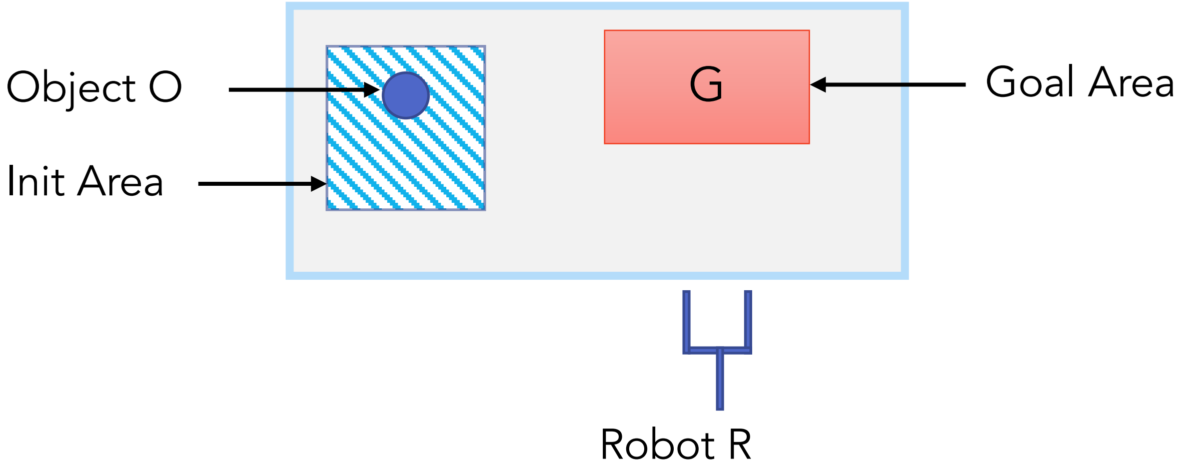
PDDLStream

- Forward-search over hybrid representation
- Modifies the PDDL representation



PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation

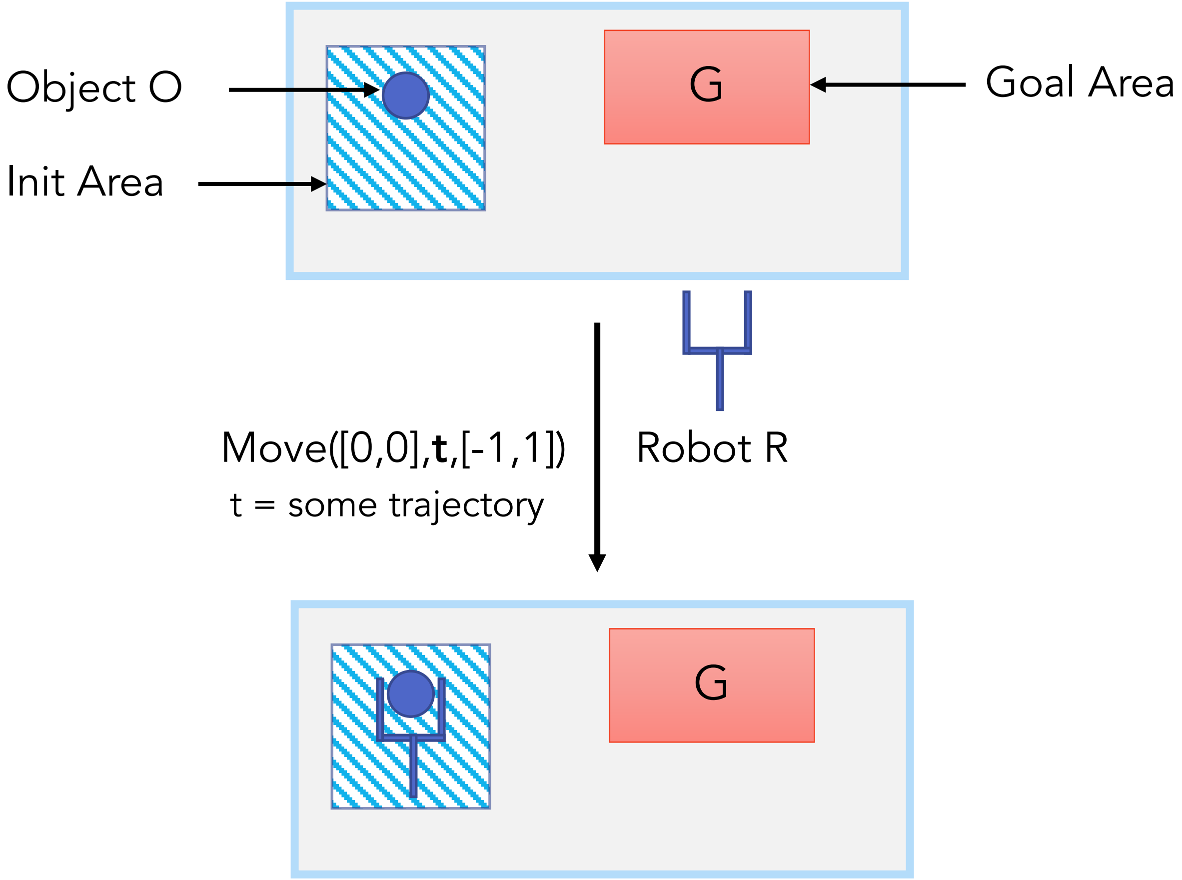



```
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1)
(forall(?b)(imply (Block ?b) (Safe ?t ?b))))
 :eff (and (AtConf ?q2) (not (AtConf ?q1))
       (incr (total-cost) (Dist ?t)))
```

PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation

```
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1)
(forall(?b)(imply (Block ?b) (Safe ?t ?b))))
 :eff (and (AtConf ?q2) (not (AtConf ?q1))
      (incr (total-cost) (Dist ?t)))
```

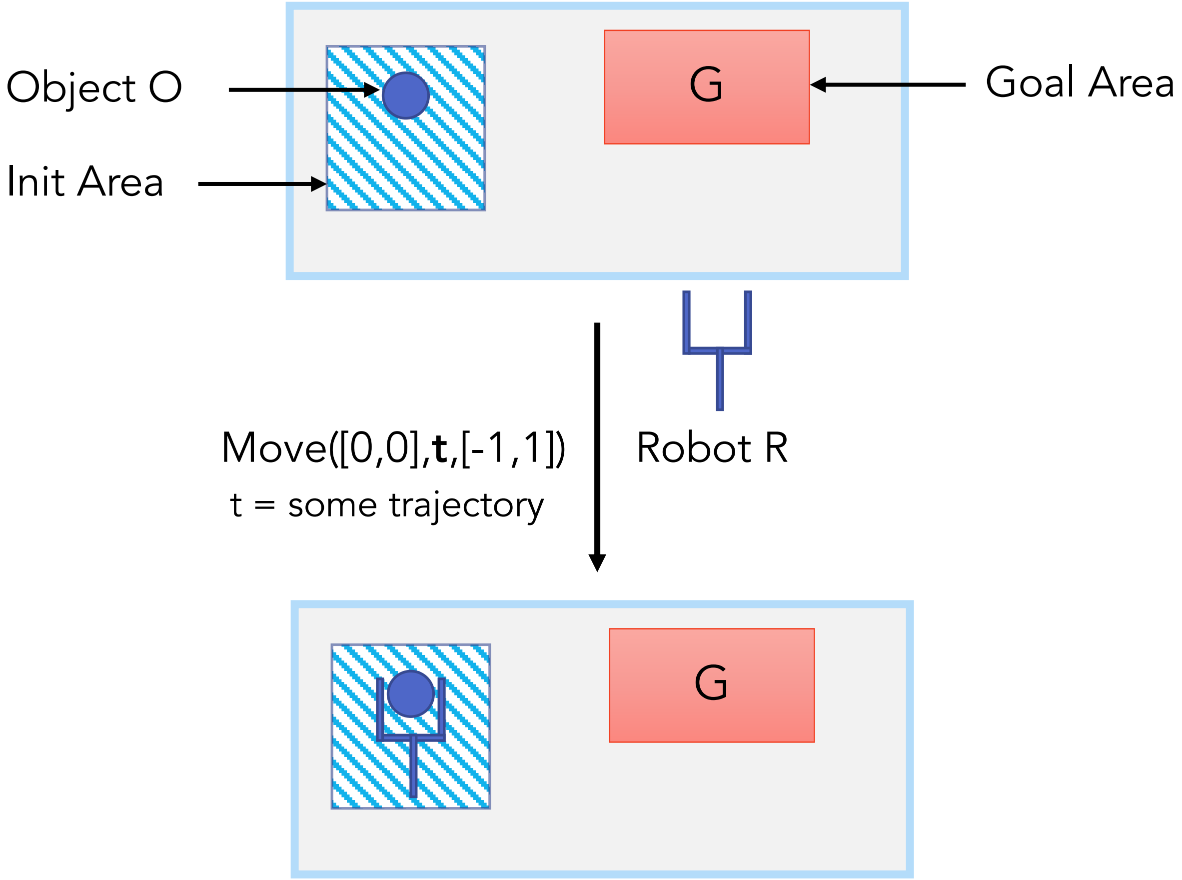


PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation

Requires continuous parameters

```
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1)
(forall(?b)(imply (Block ?b) (Safe ?t ?b))))
 :eff (and (AtConf ?q2) (not (AtConf ?q1))
      (incr (total-cost) (Dist ?t)))
```

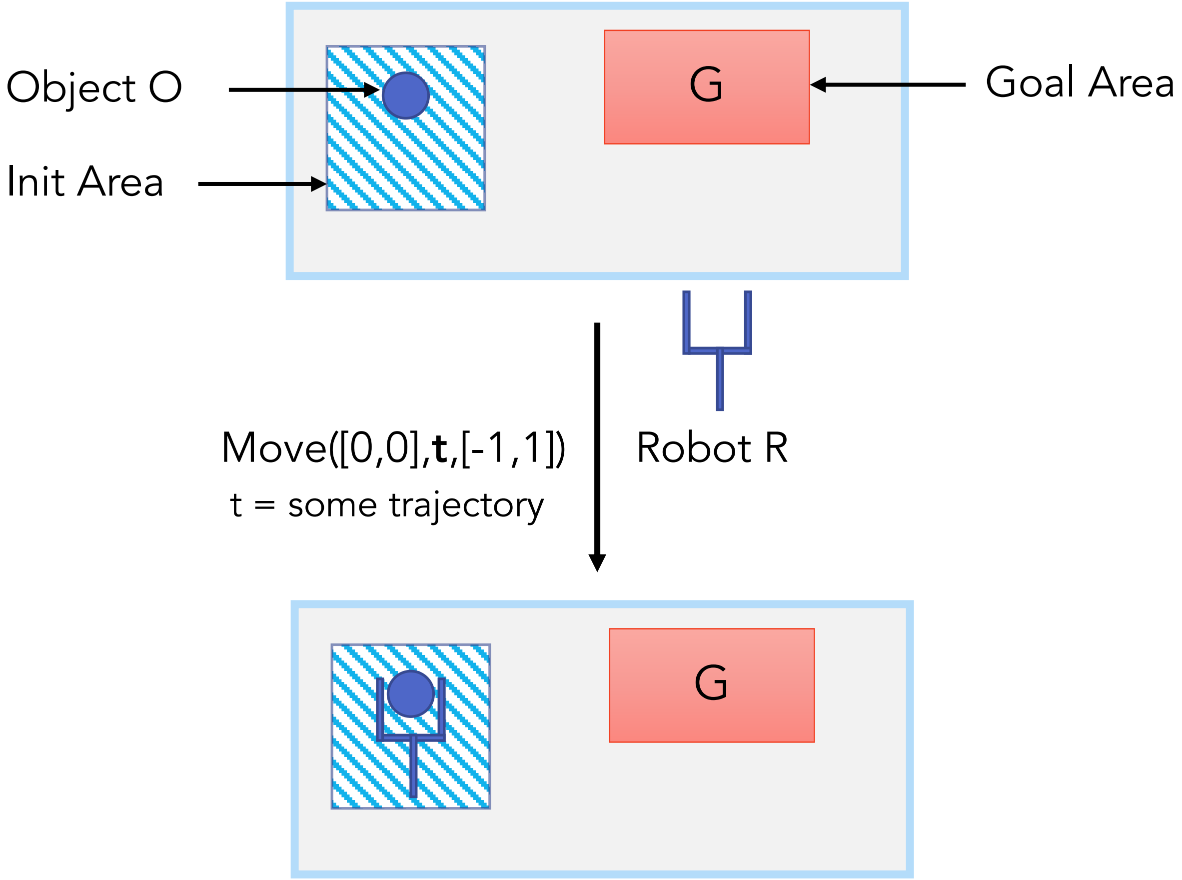


PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation
- Main two concepts:
  - Streams

```
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1)

 :eff (and (AtConf ?q2) (not (AtConf ?q1))
      (incr (total-cost) (Dist ?t)))
```

```
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
(Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```

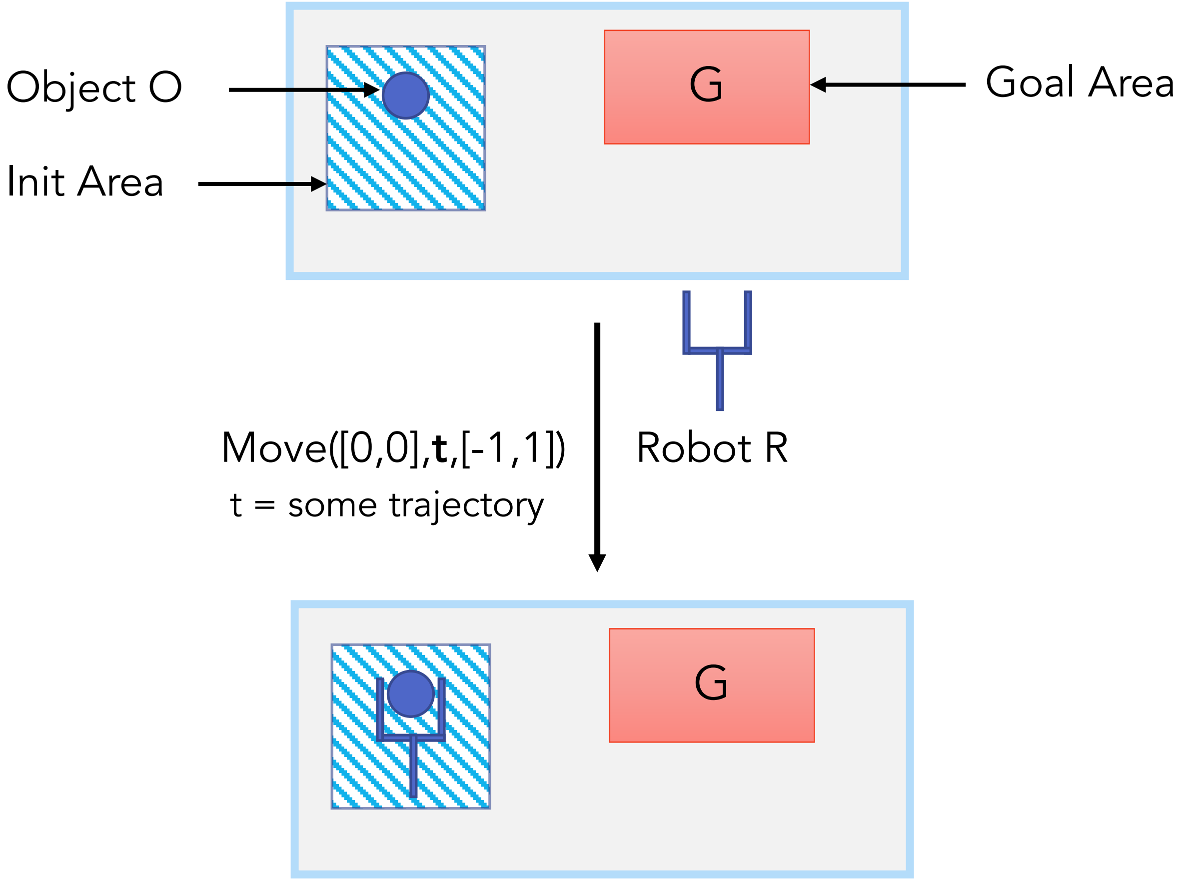


PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation
- Main two concepts:
    - Streams ~ Samplers or Generators

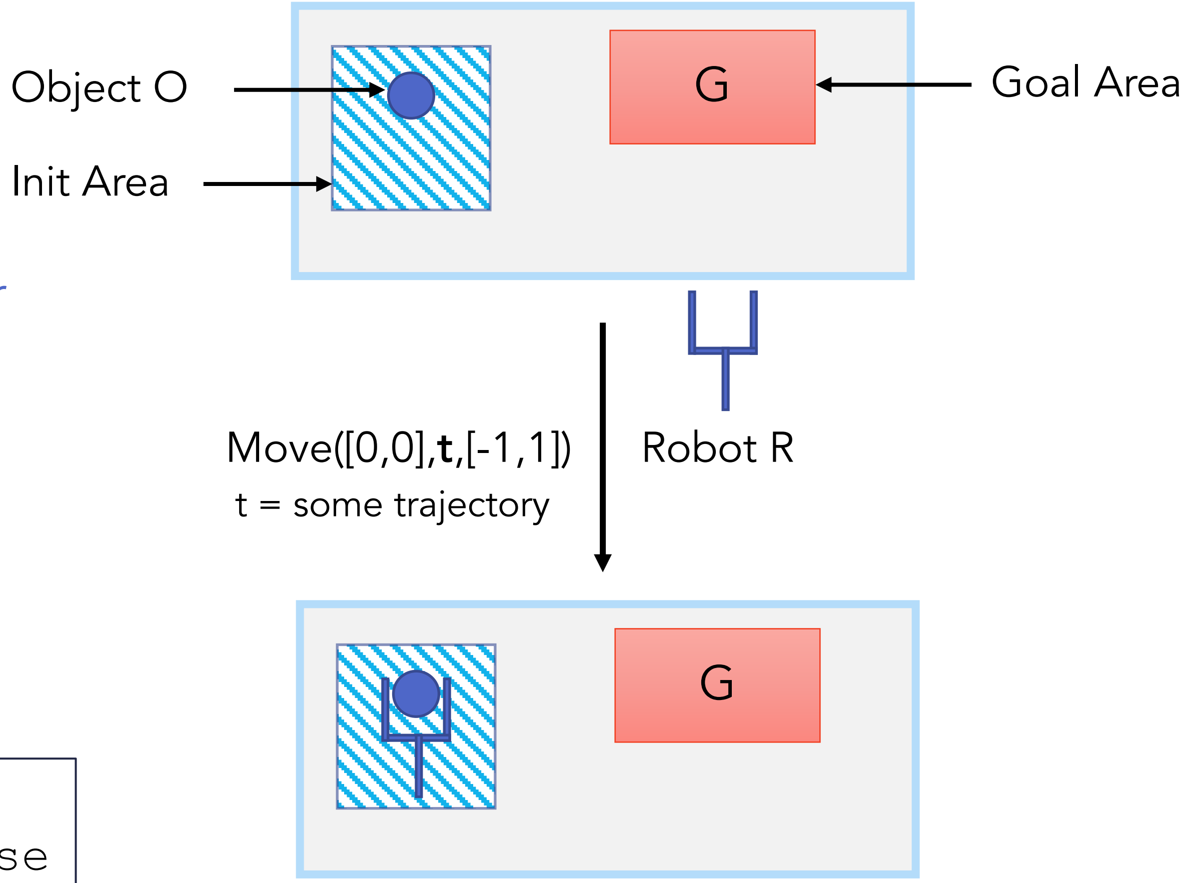
```
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1)

 :eff (and (AtConf ?q2) (not (AtConf ?q1))
      (incr (total-cost) (Dist ?t)))
```

```
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
(Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```



PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation
- Main two concepts:
  - Streams ~ Samplers or Generators
    - Procedural component: A conditional generator




```
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1)

 :eff (and (AtConf ?q2) (not (AtConf ?q1))
      (incr (total-cost) (Dist ?t)))
```

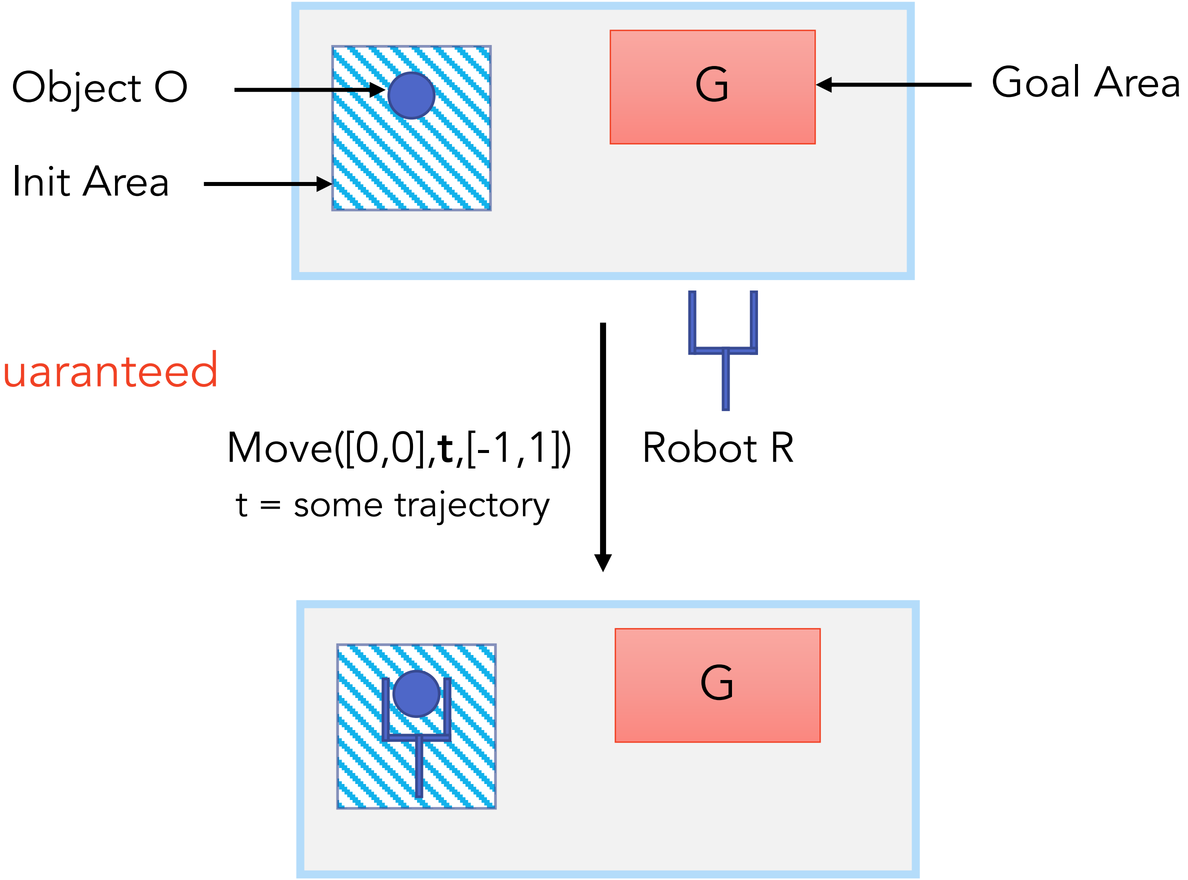
```
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
(Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```

```
def motion(q1, q2):
#code to sample a pose
.....
return traj
```

PDDLStream: Streams

- Forward-search over hybrid representation
- Modifies the PDDL representation
- Main two concepts:
  - Streams ~ **Samplers or Generators**
    - Procedural component: a conditional generator
    - Declarative component: add facts that can be guaranteed

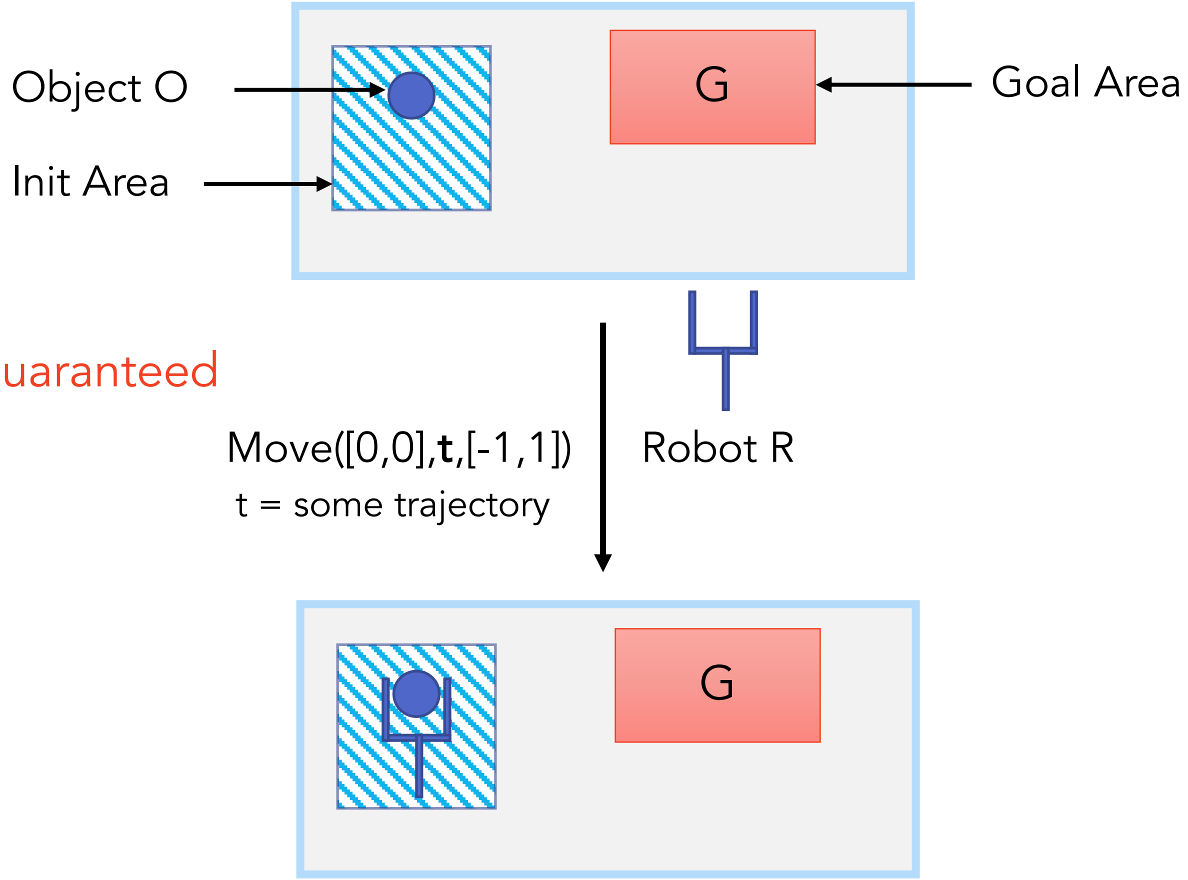
```
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1)

 :eff (and (AtConf ?q2) (not (AtConf ?q1))
      (incr (total-cost) (Dist ?t)))


(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
 (Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```



PDDLStream: Streams

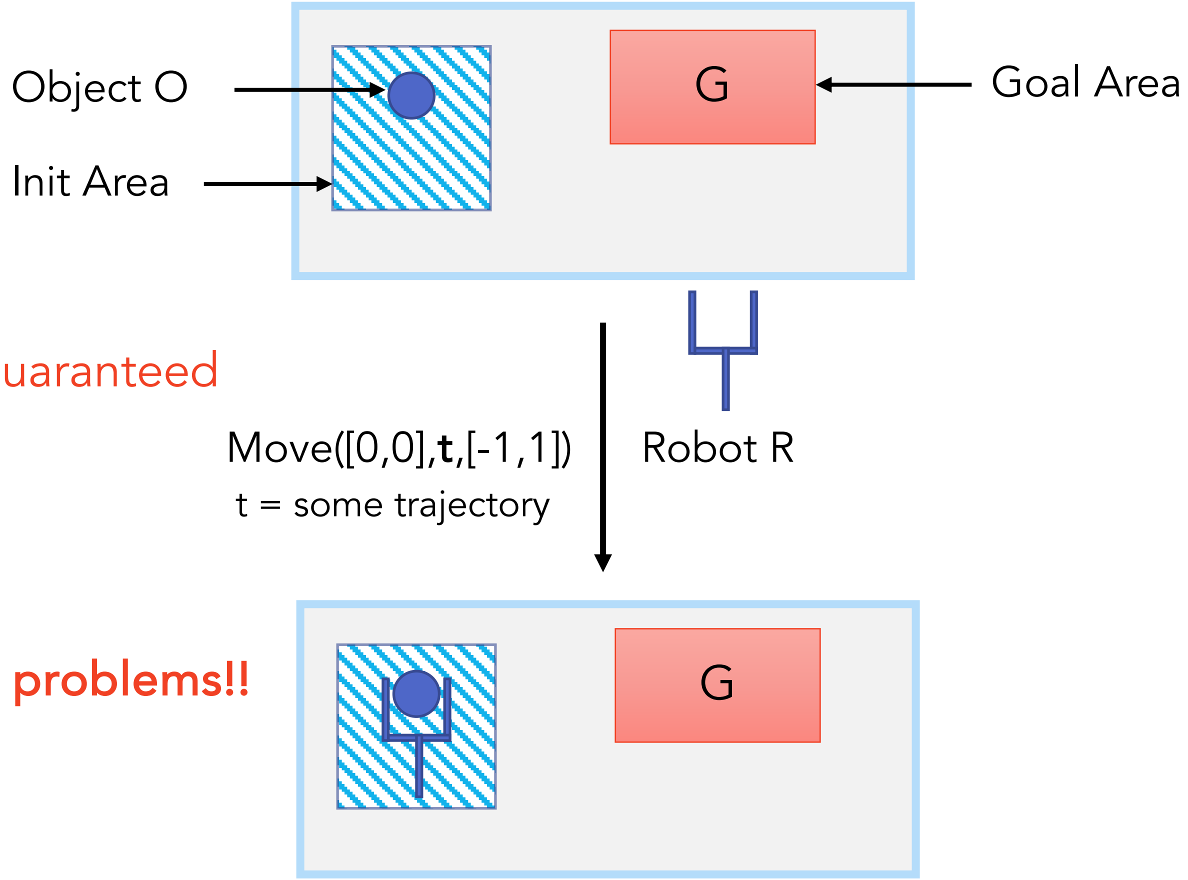
- Forward-search over hybrid representation
- Modifies the PDDL representation
- Main two concepts:
    - Streams ~ **Samplers or Generators**
        - Procedural component: a conditional generator
        - Declarative component: add facts that can be guaranteed

```
(:action move
 :param (?q1 ?t ?q2)
 :pre (and (Motion ?q1 ?t ?q2) (AtConf ?q1)

 :eff (and (AtConf ?q2) (not (AtConf ?q1))
      (incr (total-cost) (Dist ?t)))


(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
(Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```



PDDLStream: C1

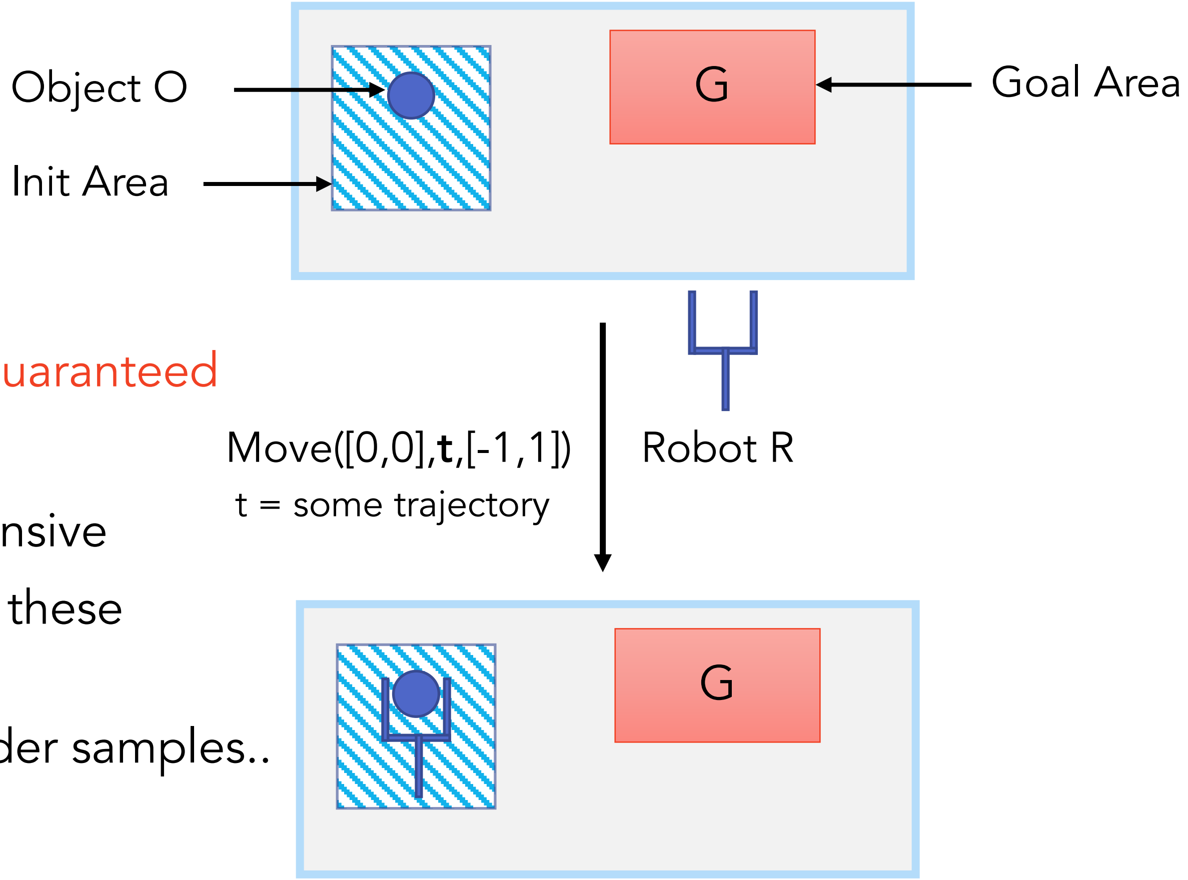
- Forward-search over hybrid representation
- Modifies the PDDL representation
- Main two concepts:
  - Streams ~ **Samplers or Generators**
    - Procedural component: a conditional generator
    - Declarative component: add facts that can be guaranteed



C1: Infinite groundings → Infinitely many motion planning problems!!

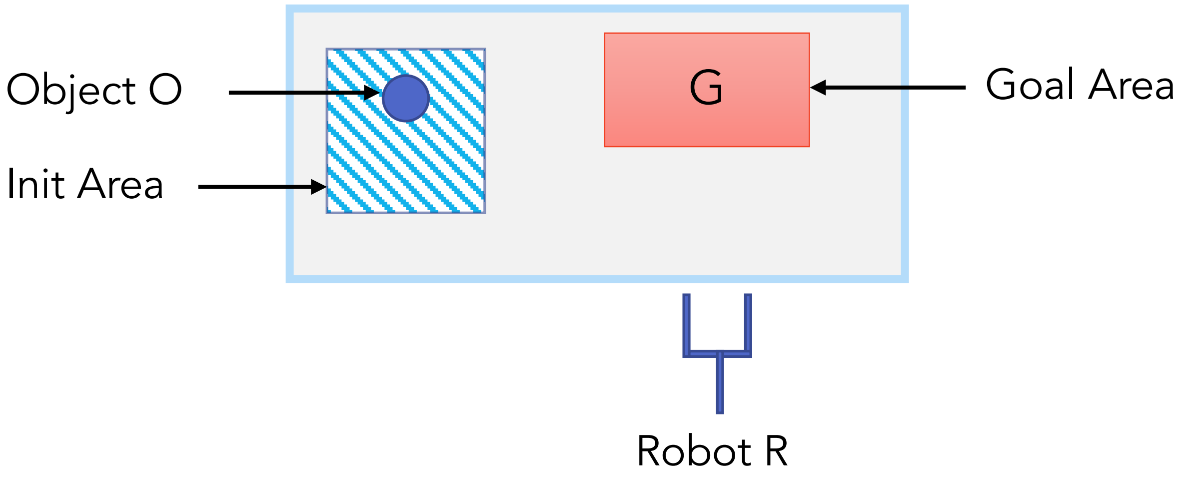
PDDLStream: Optimistic Samples

- Forward-search over hybrid representation
- Modifies the PDDL representation
- Main two concepts:
  - Streams ~ **Samplers or Generators**
    - Procedural component: a conditional generator
    - Declarative component: add facts that can be guaranteed
  - Optimistic samples ~ **Symbolic References**
    - Evaluating streams to generate samples is expensive
    - High-level planning would need a thousands of these calls for even simple problems
    - So solution? → Generate "optimistic" placeholder samples..
      Assumed to be valid



PDDLStream: Overall Approach

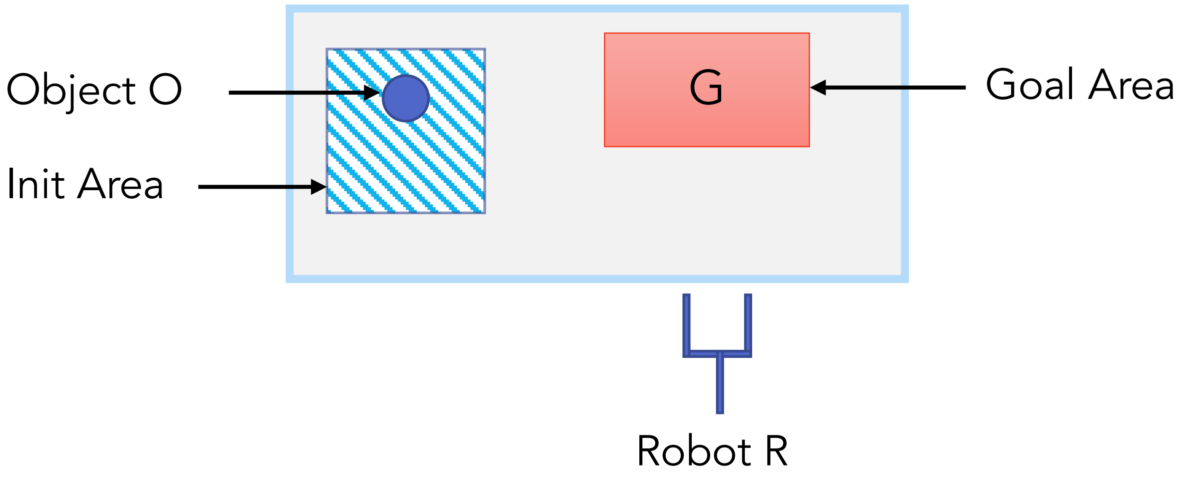
- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples (Tackling C1)




```
(:stream poses
 :inp (?b ?r)
 :dom (and (Block ?b)
 (Region ?r))
 :out (?p)
 :cert (and (Pose ?b ?p)
 (Contain ?b ?p ?r)))
(:stream grasps
 :inp (?b)
 :dom (Block ?b)
 :out (?g)
 :cert (Grasp ?b ?g))
(:stream cfree
 :inp (?t ?b ?p)
 :dom (and (Traj ?t)
 (Pose ?b ?p))
 :cert (CFree ?t ?b ?p))
(:stream ik
 :inp (?b ?p ?g)
 :dom (and
 (Pose ?b ?p)
 (Grasp ?b ?g))
 :out (?q)
 :cert (and (Conf ?q)
 (Kin ?b ?p ?g ?q)))
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
 (Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```

PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples



Only depends on objects that does not require any stream → Required depth = 0

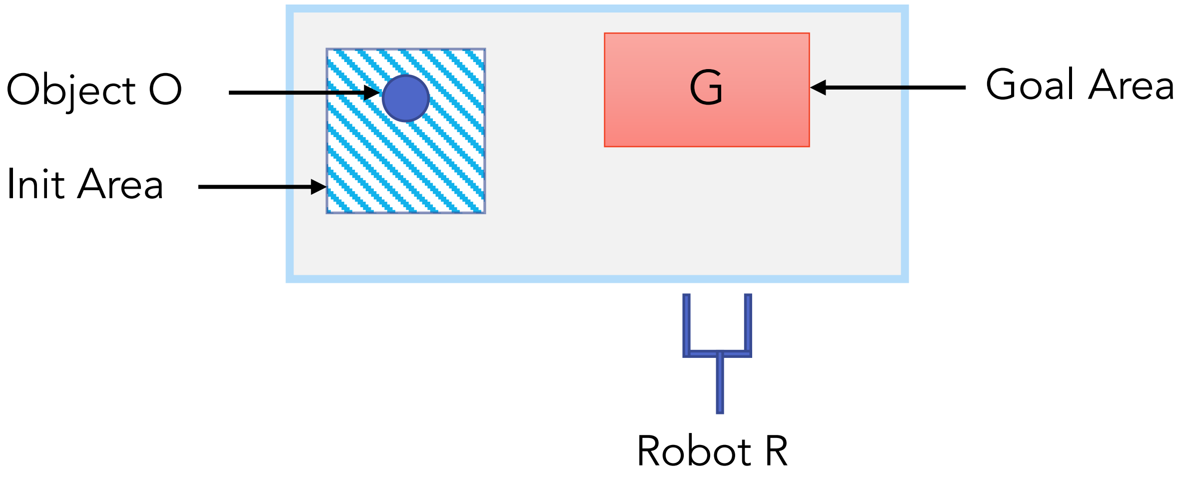
Achievable optimistic samples at level 0: #p1_0 [(O, Init)]
#p2_0[(O, Goal)]
#g1_0[(G, Goal)]

```
(:stream poses
 :inp (?b ?r)
 :dom (and (Block ?b)
 (Region ?r))
 :out (?p)
 :cert (and (Pose ?b ?p)
 (Contain ?b ?p ?r)))
(:stream grasps
 :inp (?b)
 :dom (Block ?b)
 :out (?g)
 :cert (Grasp ?b ?g))
(:stream cfree
 :inp (?t ?b ?p)
 :dom (and (Traj ?t)
 (Pose ?b ?p))
 :cert (CFree ?t ?b ?p))
```

```
(:stream ik
 :inp (?b ?p ?g)
 :dom (and
 (Pose ?b ?p)
 (Grasp ?b ?g))
 :out (?q)
 :cert (and (Conf ?q)
 (Kin ?b ?p ?g ?q)))
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
 (Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```

PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples



Requires input that is generated using streams of depth = 0 → Required depth = 1

At level 1: #q1_1 [(O,#p1_0, #g_0)]

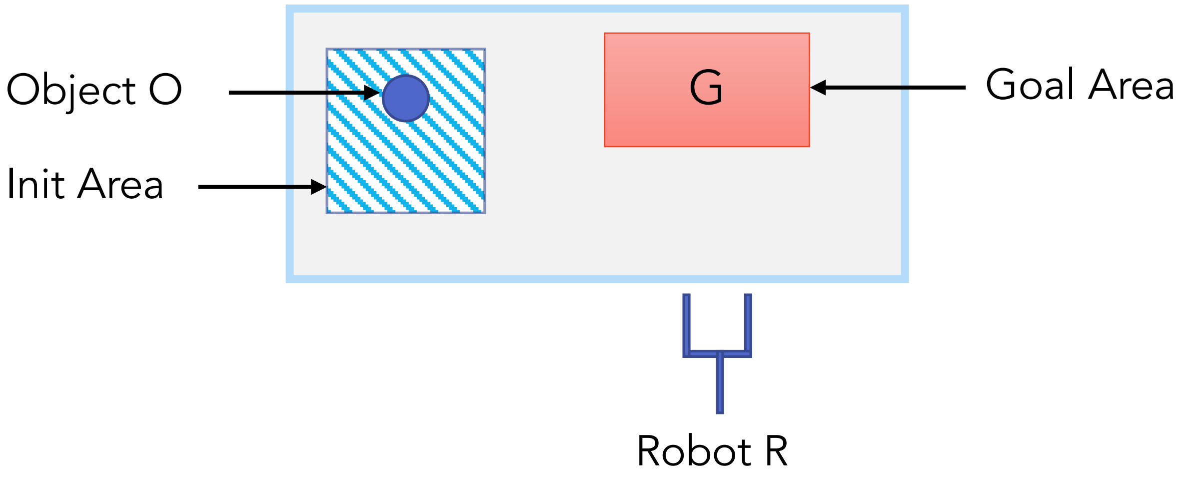
#q2_1 [(O,#p2_0, #g_0)]

….

```
(:stream poses
 :inp (?b ?r)
 :dom (and (Block ?b)
 (Region ?r))
 :out (?p)
 :cert (and (Pose ?b ?p)
 (Contain ?b ?p ?r)))
(:stream grasps
 :inp (?b)
 :dom (Block ?b)
 :out (?g)
 :cert (Grasp ?b ?g))
(:stream cfree
 :inp (?t ?b ?p)
 :dom (and (Traj ?t)
 (Pose ?b ?p))
 :cert (CFree ?t ?b ?p))
(:stream ik
 :inp (?b ?p ?g)
 :dom (and
 (Pose ?b ?p)
 (Grasp ?b ?g))
 :out (?q)
 :cert (and (Conf ?q)
 (Kin ?b ?p ?g ?q)))
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
 (Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```

PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples



Requires input that is generated using streams of depth = 0 → Required depth = 1

At level 1: #q1_1 [(O,#p1_0, #g_0)]

#q2_1 [(O,#p2_0, #g_0)]

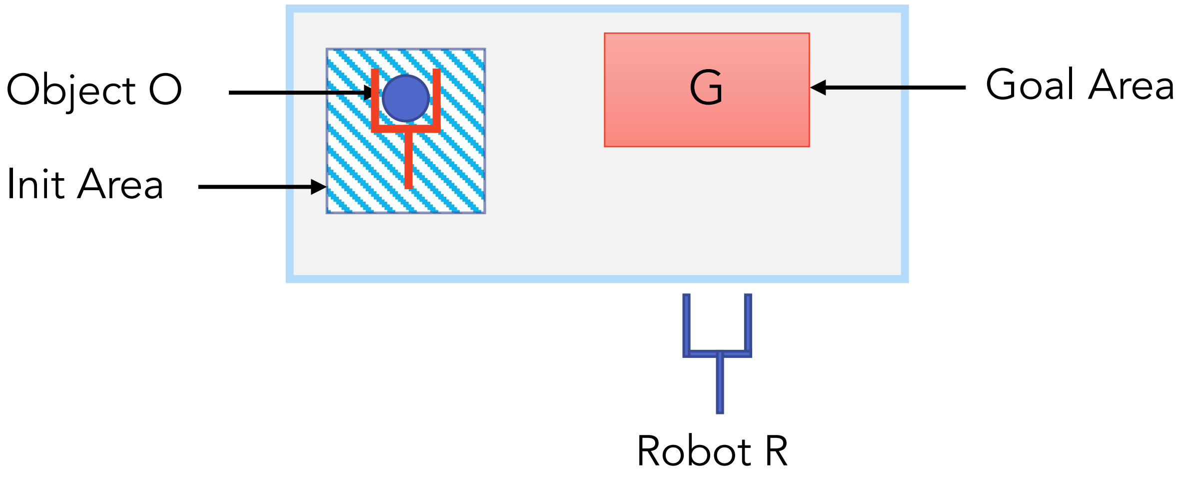
....

```
(:stream poses
 :inp (?b ?r)
 :dom (and (Block ?b)
 (Region ?r))
 :out (?p)
 :cert (and (Pose ?b ?p)
 (Contain ?b ?p ?r)))
(:stream grasps
 :inp (?b)
 :dom (Block ?b)
 :out (?g)
 :cert (Grasp ?b ?g))
(:stream cfree
 :inp (?t ?b ?p)
 :dom (and (Traj ?t)
 (Pose ?b ?p))
 :cert (CFree ?t ?b ?p))
```

```
(:stream ik
 :inp (?b ?p ?g)
 :dom (and
 (Pose ?b ?p)
 (Grasp ?b ?g))
 :out (?q)
 :cert (and (Conf ?q)
 (Kin ?b ?p ?g ?q)))
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
 (Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```

PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples
  - Compute an abstract plan using the optimistic samples



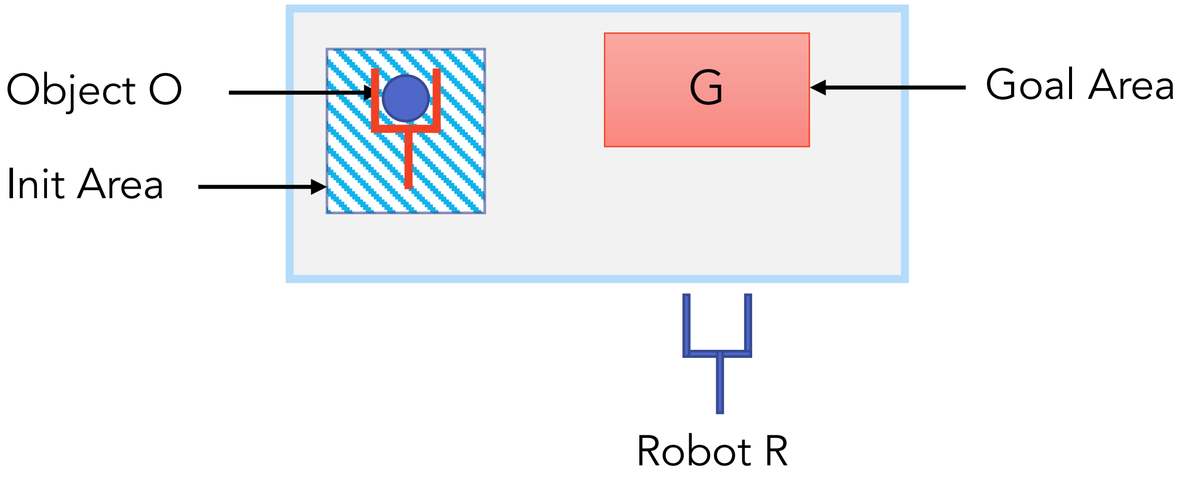
We need:
(Motion …) and (Kin …)

Min depth = 2

```
(:stream poses
 :inp (?b ?r)
 :dom (and (Block ?b)
 (Region ?r))
 :out (?p)
 :cert (and (Pose ?b ?p)
 (Contain ?b ?p ?r)))
(:stream grasps
 :inp (?b)
 :dom (Block ?b)
 :out (?g)
 :cert (Grasp ?b ?g))
(:stream cfree
 :inp (?t ?b ?p)
 :dom (and (Traj ?t)
 (Pose ?b ?p))
 :cert (CFree ?t ?b ?p))
(:stream ik
 :inp (?b ?p ?g)
 :dom (and
 (Pose ?b ?p)
 (Grasp ?b ?g))
 :out (?q)
 :cert (and (Conf ?q)
 (Kin ?b ?p ?g ?q)))
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
 (Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```

PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
    - Generate achievable optimistic samples
    - Compute an abstract plan using the optimistic samples
    - If no plan found
        - Increase depth and repeat.



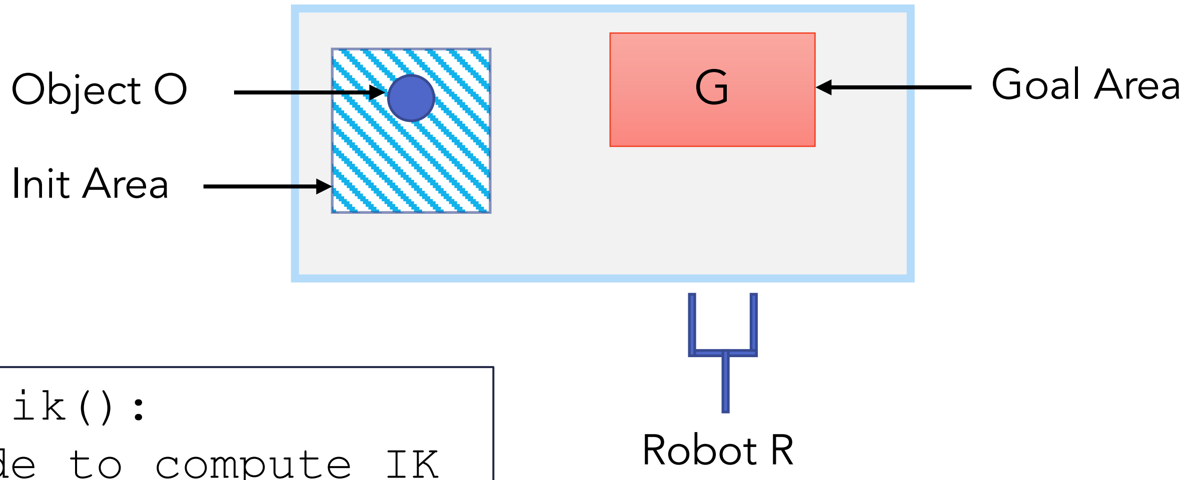
We need:
(Motion …) and (Kin …)

Min depth = 2

```
(:stream poses
 :inp (?b ?r)
 :dom (and (Block ?b)
 (Region ?r))
 :out (?p)
 :cert (and (Pose ?b ?p)
 (Contain ?b ?p ?r)))
(:stream grasps
 :inp (?b)
 :dom (Block ?b)
 :out (?g)
 :cert (Grasp ?b ?g))
(:stream cfree
 :inp (?t ?b ?p)
 :dom (and (Traj ?t)
 (Pose ?b ?p))
 :cert (CFree ?t ?b ?p))
(:stream ik
 :inp (?b ?p ?g)
 :dom (and
 (Pose ?b ?p)
 (Grasp ?b ?g))
 :out (?q)
 :cert (and (Conf ?q)
 (Kin ?b ?p ?g ?q)))
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
(Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```

PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples
  - Compute an abstract plan using the optimistic samples
  - If no plan found
    - Increase dept and repeat.
  - If a plan is found
    - Use streams to instantiate optimistic samples with real values




```
def ik():
#code to compute IK
.....
```

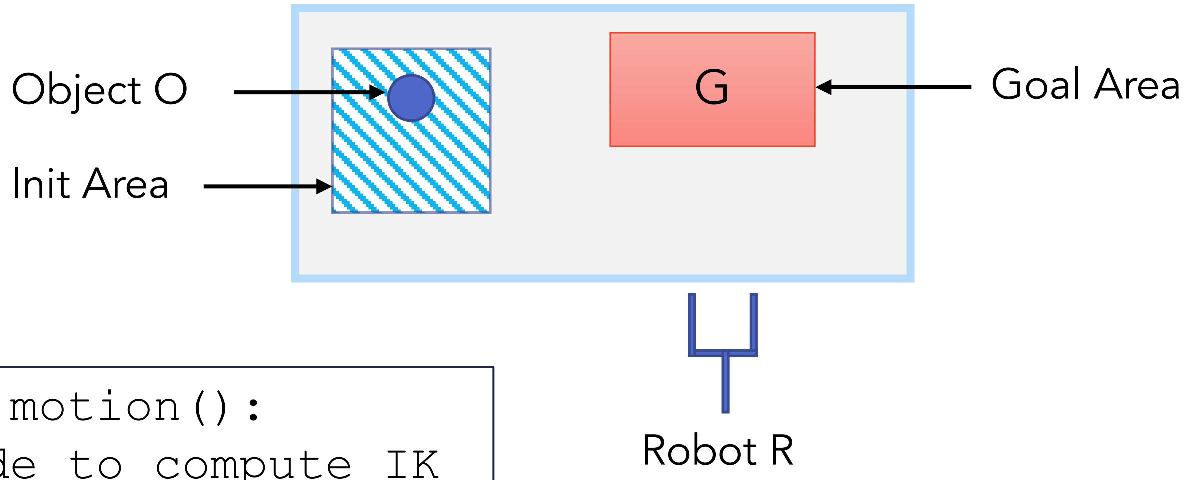
```
(Move #q0 #t0 #q1)
(Pick b1 #p0 g #q1)
(Move #q1 #t2 #q2)
(Place b #p2 g #q2)
```

```
(:stream poses
 :inp (?b ?r)
 :dom (and (Block ?b)
 (Region ?r))
 :out (?p)
 :cert (and (Pose ?b ?p)
 (Contain ?b ?p ?r)))
(:stream grasps
 :inp (?b)
 :dom (Block ?b)
 :out (?g)
 :cert (Grasp ?b ?g))
(:stream cfree
 :inp (?t ?b ?p)
 :dom (and (Traj ?t)
 (Pose ?b ?p))
 :cert (CFree ?t ?b ?p))
```

```
(:stream ik
 :inp (?b ?p ?g)
 :dom (and
 (Pose ?b ?p)
 (Grasp ?b ?g))
 :out (?q)
 :cert (and (Conf ?q)
 (Kin ?b ?p ?g ?q)))
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
(Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
:dom (Traj ?t))
```

PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples
  - Compute an abstract plan using the optimistic samples
  - If no plan found
    - Increase dept and repeat.
  - If a plan is found
    - Use streams to instantiate optimistic samples with real values
    - If no instantiation found
      - Disable stream at the current depth and replan with the same depth




```
def motion():
#code to compute IK
.....
```

```
(Move #q0 #t0 #q1)
(Pick b1 #p0 g #q1)
(Move #q1 #t2 #q2)
(Place b #p2 g #q2)
```

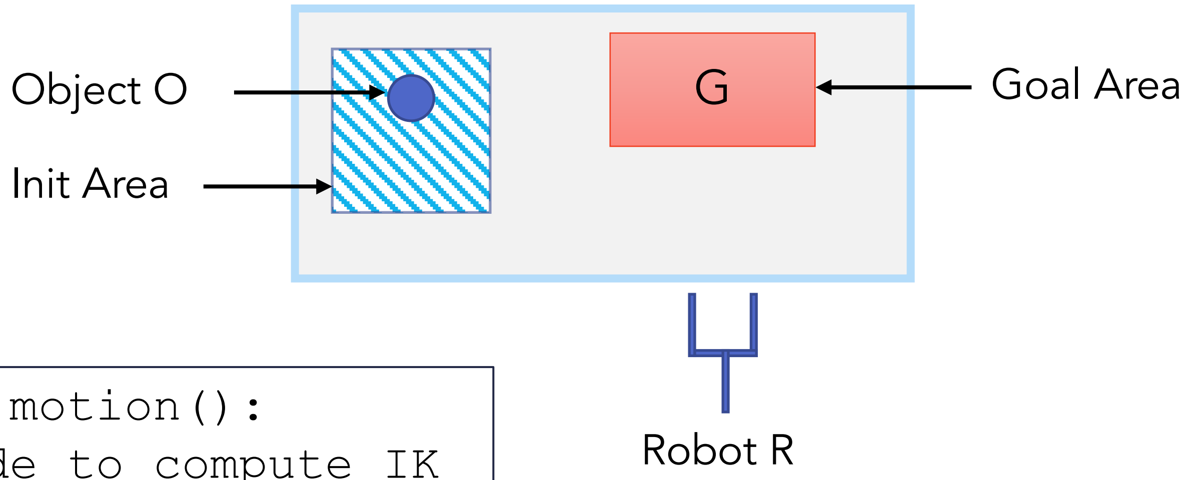
```
(:stream poses
 :inp (?b ?r)
 :dom (and (Block ?b)
 (Region ?r))
 :out (?p)
 :cert (and (Pose ?b ?p)
 (Contain ?b ?p ?r)))
(:stream grasps
 :inp (?b)
 :dom (Block ?b)
 :out (?g)
 :cert (Grasp ?b ?g))
(:stream cfree
 :inp (?t ?b ?p)
 :dom (and (Traj ?t)
 (Pose ?b ?p))
 :cert (CFree ?t ?b ?p))
```

```
(:stream ik
 :inp (?b ?p ?g)
 :dom (and
 (Pose ?b ?p)
 (Grasp ?b ?g))
 :out (?q)
 :cert (and (Conf ?q)
 (Kin ?b ?p ?g ?q)))
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
 (Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```

PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples
  - Compute an abstract plan using the optimistic samples
  - If no plan found
    - Increase dept and repeat.
  - If a plan is found
    - Use streams to instantiate optimistic samples with real values
    - If no instantiation found
      - Disable stream at the current depth and replan with the same depth

Forces task planner to give a new plan!!




```
def motion():
#code to compute IK
.....
```

```
(Move #q0 #t0 #q1)
(Pick b1 #p0 g #q1)
(Move #q1 #t2 #q2)
(Place b #p2 g #q2)
```

Akin to the backtracking and IDTMP

Constraint approach

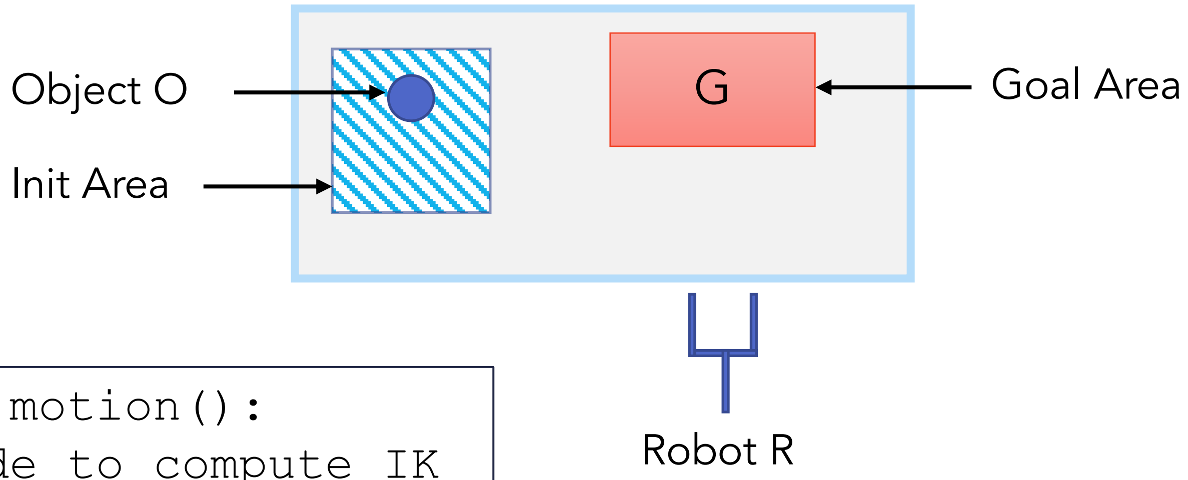
```
(:stream poses
 :inp (?b ?r)
 :dom (and (Block ?b)
 (Region ?r))
 :out (?p)
 :cert (and (Pose ?b ?p)
 (Contain ?b ?p ?r)))
(:stream grasps
 :inp (?b)
 :dom (Block ?b)
 :out (?g)
 :cert (Grasp ?b ?g))
(:stream cfree
 :inp (?t ?b ?p)
 :dom (and (Traj ?t)
 (Pose ?b ?p))
 :cert (CFree ?t ?b ?p))
```

```
(:stream ik
 :inp (?b ?p ?g)
 :dom (and
 (Pose ?b ?p)
 (Grasp ?b ?g))
 :out (?q)
 :cert (and (Conf ?q)
 (Kin ?b ?p ?g ?q)))
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
(Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```

PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples
  - Compute an abstract plan using the optimistic samples
  - If no plan found
    - Increase dept and repeat.
  - If a plan is found
    - Use streams to instantiate optimistic samples with real values
    - If no instantiation found
      - Disable stream at the current depth and replan with the same depth

depth+=1




```
def motion():
#code to compute IK
.....
```

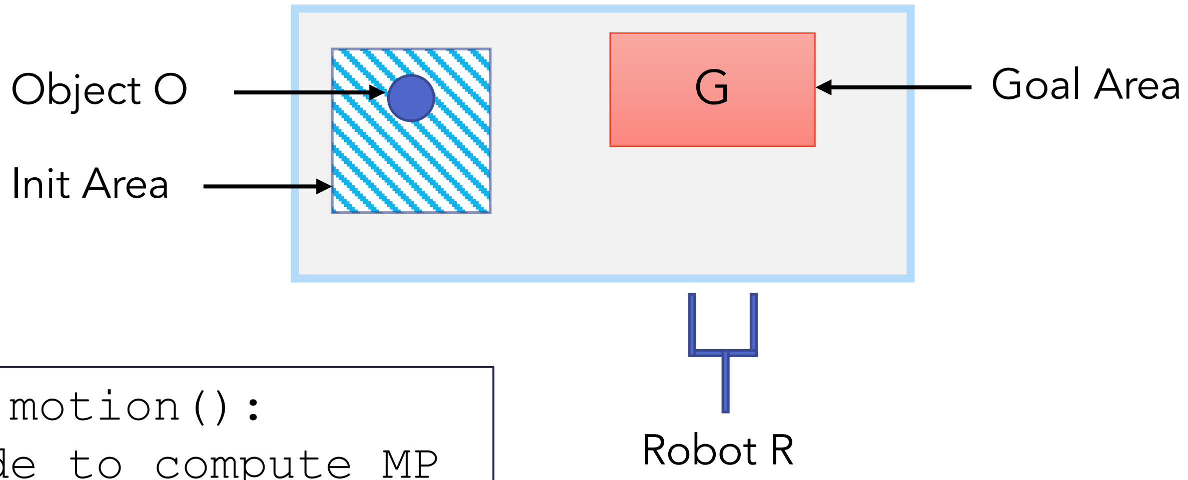
```
(Move #q0 #t0 #q1)
(Pick b1 #p0 g #q1)
(Move #q1 #t2 #q2)
(Place b #p2 g #q2)
```

```
(:stream poses
 :inp (?b ?r)
 :dom (and (Block ?b)
 (Region ?r))
 :out (?p)
 :cert (and (Pose ?b ?p)
 (Contain ?b ?p ?r)))
(:stream grasps
 :inp (?b)
 :dom (Block ?b)
 :out (?g)
 :cert (Grasp ?b ?g))
(:stream cfree
 :inp (?t ?b ?p)
 :dom (and (Traj ?t)
 (Pose ?b ?p))
 :cert (CFree ?t ?b ?p))
(:stream ik
 :inp (?b ?p ?g)
 :dom (and
 (Pose ?b ?p)
 (Grasp ?b ?g))
 :out (?q)
 :cert (and (Conf ?q)
 (Kin ?b ?p ?g ?q)))
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
(Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
:dom (Traj ?t))
```

Forces task planner to give a new plan!!

PDDLStream: Overall Approach

- Depth = 0
- For the current dept:
  - Generate achievable optimistic samples
  - Compute an abstract plan using the optimistic samples
  - If no plan found
    - Increase dept and repeat.
  - If a plan is found
    - Use streams to instantiate optimistic samples with real values
    - If no instantiation found
      - Disable stream at the current depth and replan with the same depth




```
def motion():
#code to compute MP
.....
```

```
(Move #q0 #t0 #q1)
(Pick b1 #p0 g #q1)
(Move #q1 #t2 #q2)
(Place b #p2 g #q2)
```

```
(:stream poses
 :inp (?b ?r)
 :dom (and (Block ?b)
 (Region ?r))
 :out (?p)
 :cert (and (Pose ?b ?p)
 (Contain ?b ?p ?r)))
(:stream grasps
 :inp (?b)
 :dom (Block ?b)
 :out (?g)
 :cert (Grasp ?b ?g))
(:stream cfree
 :inp (?t ?b ?p)
 :dom (and (Traj ?t)
 (Pose ?b ?p))
 :cert (CFree ?t ?b ?p))
(:stream ik
 :inp (?b ?p ?g)
 :dom (and
 (Pose ?b ?p)
 (Grasp ?b ?g))
 :out (?q)
 :cert (and (Conf ?q)
 (Kin ?b ?p ?g ?q)))
(:stream motion
 :inp (?q1 ?q2)
 :dom (and (Conf ?q1)
 (Conf ?q2))
 :out (?t)
 :cert (and (Traj ?t)
(Motion ?q1 ?t ?q2)))
(:function (Dist ?t)
 :dom (Traj ?t))
```

To ensure completeness:
Adaptively switch between computing new plan and refinements.

PDDLStream: Experiments

PDDLStream: Summary

- Inputs
  - PDDL domain with streams
  - Procedural function for streams -- conditional generators
- Properties
  - Forward search using hybrid representation
  - Probabilistically complete – guarantees it by switching between computing refinements and finding new plans
- Key Ideas:
  - C1: through optimistic samples and lazy-querying streams
  - C2: by forcing the planner to generate new plans until it finds a plan that is refinable.

Garrett, C. R., Lozano-Pérez, T., & Kaelbling, L. P. (2020, June). Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

High-Level Summary

	Search Space	High Level Planner	Low Level Reasoning	High-level	High Level Language	P1: Infinite Motion Plans	P2: Downward Refinability
aSyMov	Single	Any TP	PRM/RRT	Symbolic	PDDL	N/A	N/A
IDTMP	Dual	SMT	Any MP	Variables with continuous domains	SAS	SMT	SMT
HPN	Dual	HPN-specific regression planner	Any MP	Hybrid	HPN-specific	generators	Regression over geometric fluents
Symbolic Interface	Dual	Any TP	Any MP	Symbolic	PDDL	Pose generators	Identifying errors
PDDLStream	Dual	Any TP	Any MP	Hybrid	PDDL	optimistic samples	Iterating over all task plans

Limitations

TMP through an extension layer [Srivastava et al. 2014]
Incremental TMP [Dantam et al. 2018]

PDDLStream [Garrett et al. 2020]

Works only for deterministic problems

Limitations

TMP through an extension layer [Srivastava et al. 2014]
Incremental TMP [Dantam et al. 2018]
PDDLStream [Garrett et al. 2020]

Works only for deterministic problems

What if robot's actions are stochastic?

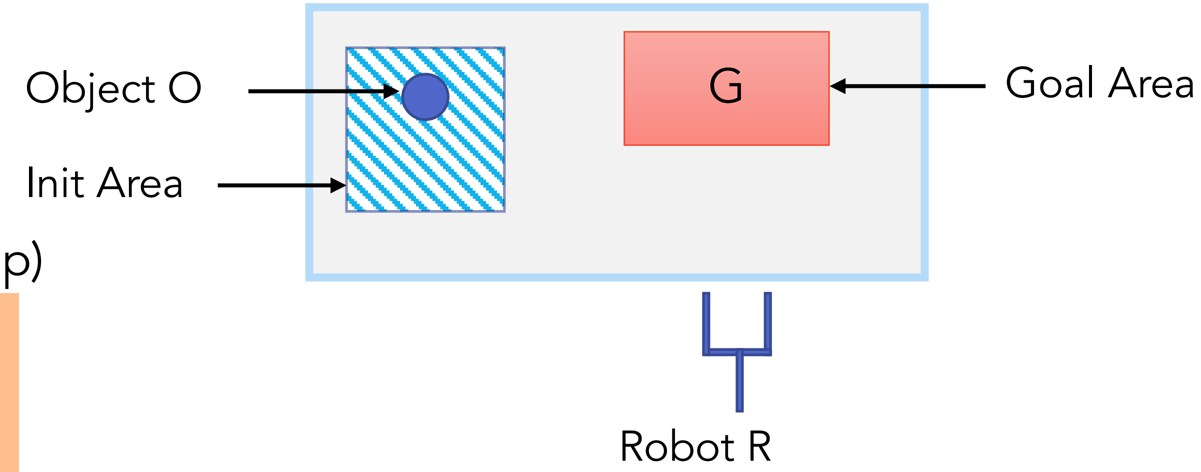
Task and Motion Planning Under Uncertainty

PickUp(obj o1, pose p1, pose p2, pose p3, path p):

precondition: Empty(gripper) ∧ GripperAt(p1) ∧ At(o1, p3) ∧ IsGraspingPose(p2, o1, p3) ∧ path(p, p1, p2) ∧ ∀o2 ¬ *Obstructs*(o2, p)

effect: 0.8 Holding(o1) ∧ ¬ At(o1, p3) ∧ ¬ Empty(gripper) ∧ GripperAt(p2)

0.2 [No change]



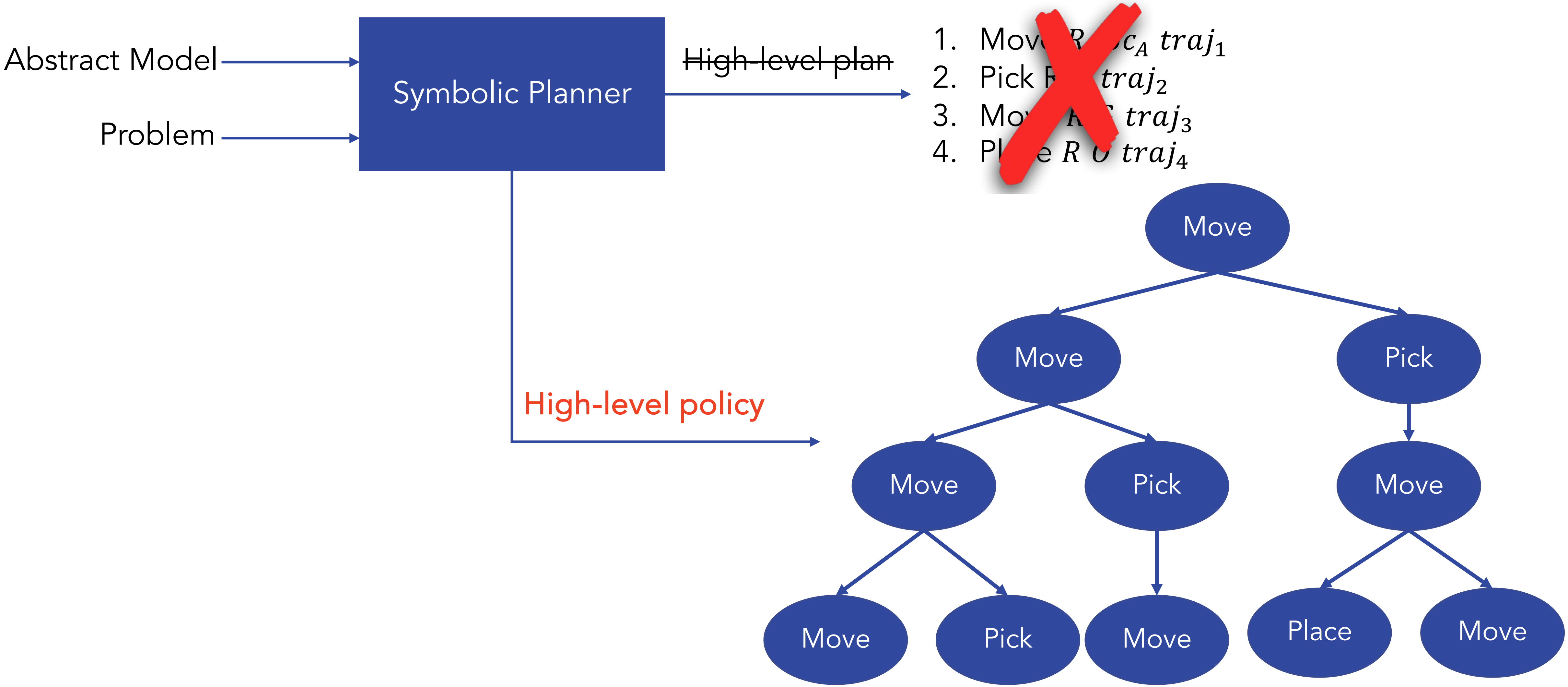
Task and Motion Planning Under Uncertainty

Abstract Model → Symbolic Planner
Problem → Symbolic Planner

Symbolic Planner → High-level plan →

1. Mov R c_A $traj_1$
2. Pick R $traj_2$
3. Mo R $traj_3$
4. Pl e R $traj_4$

Task and Motion Planning Under Uncertainty



STAMP

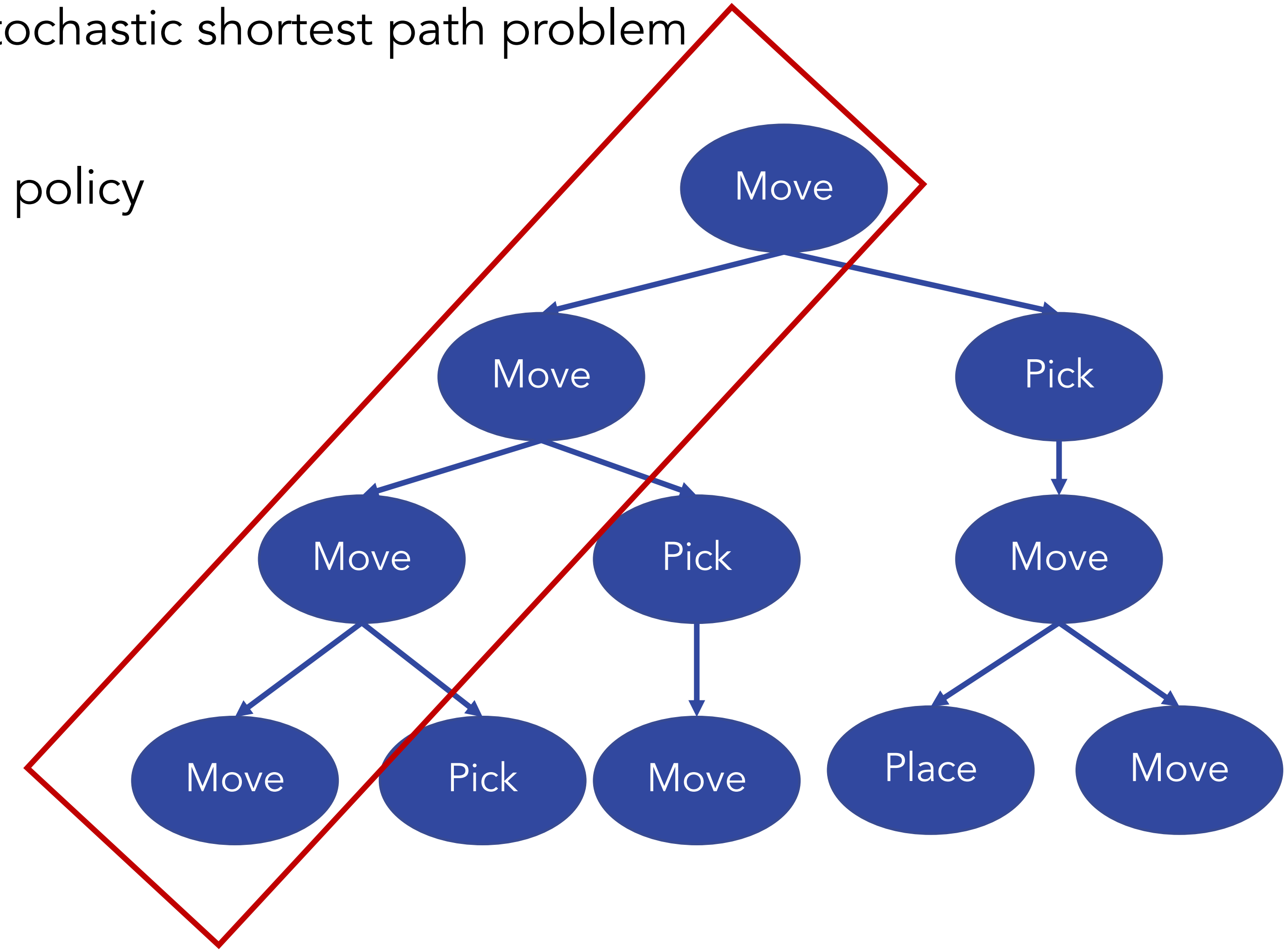
- High-level domain: ~~classical planning PDDL domain~~
  PPDDL domain for a stochastic shortest path problem

Shah, N., Vasudevan, D. K., Kumar, K., Kamojjhala, P., & Srivastava, S. (2020, May). Anytime integrated task and motion policies for stochastic environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*

STAMP

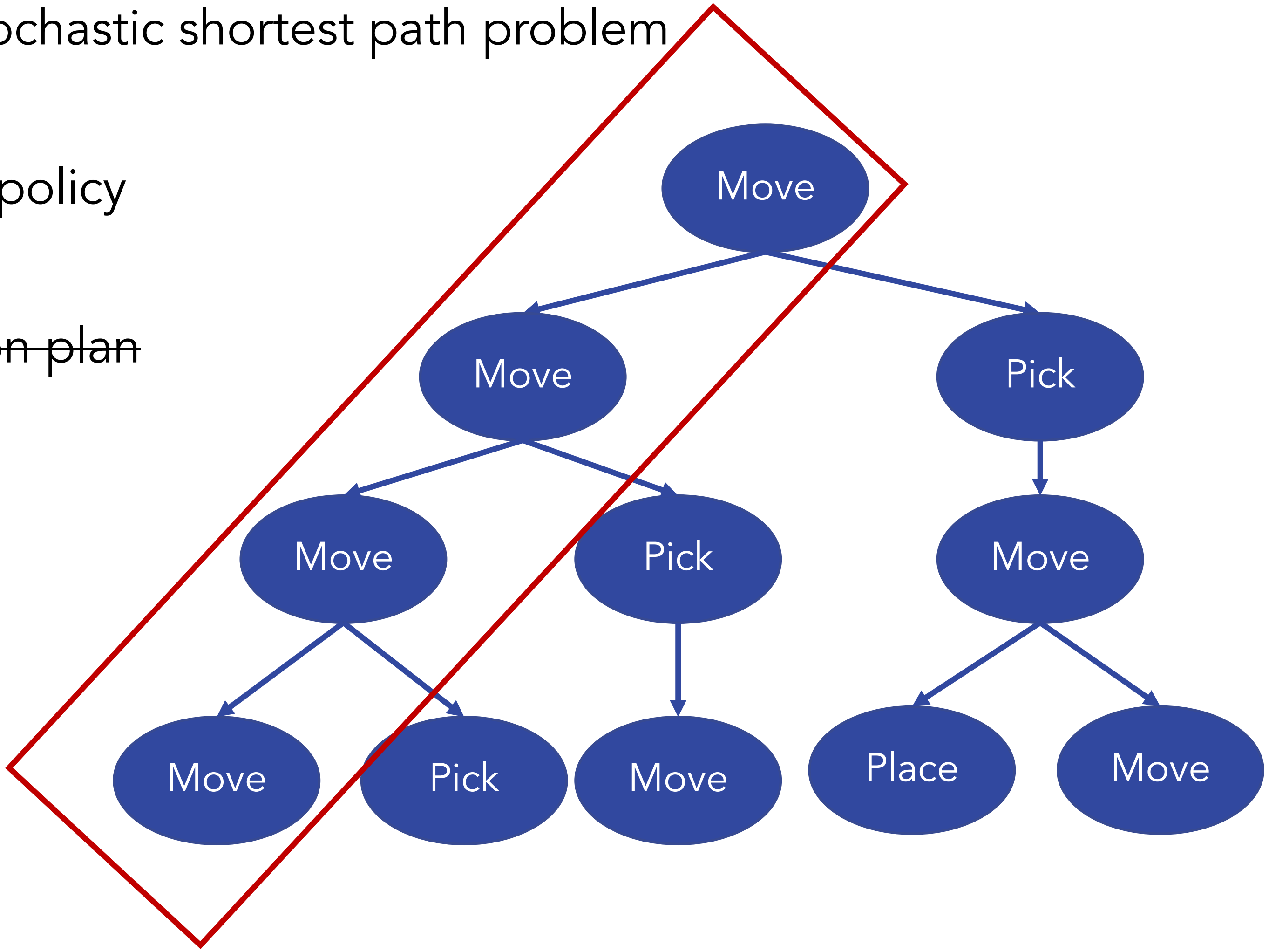
- High-level domain: ~~classical planning PDDL domain~~
  PPDDL domain for a stochastic shortest path problem

- Use an SSP solver to compute a branching policy



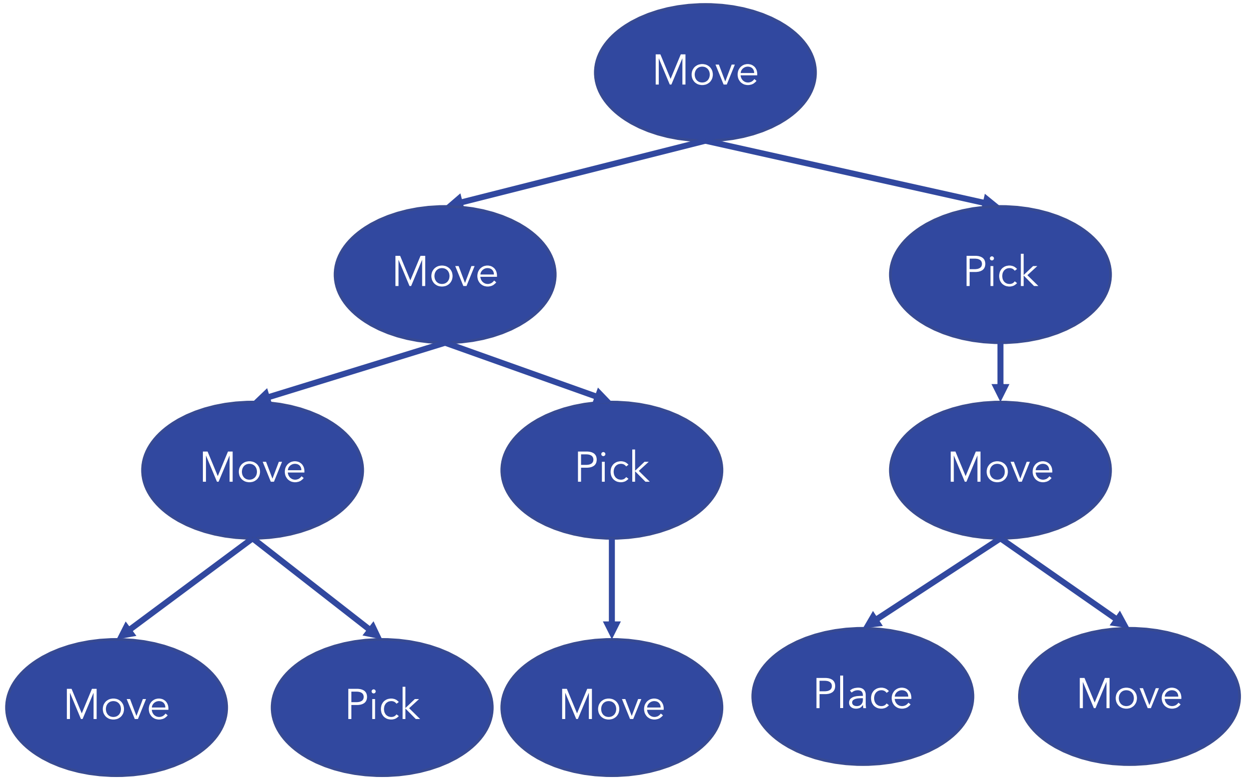
STAMP

- High-level domain: ~~classical planning PDDL domain~~
  PPDDL domain for a stochastic shortest path problem
- Use an SSP solver to compute a branching policy
- ~~Refine the plan to compute task and motion plan~~
  Refine the **entire policy** to compute
  task and motion policy



STAMP: Dealing with #branches

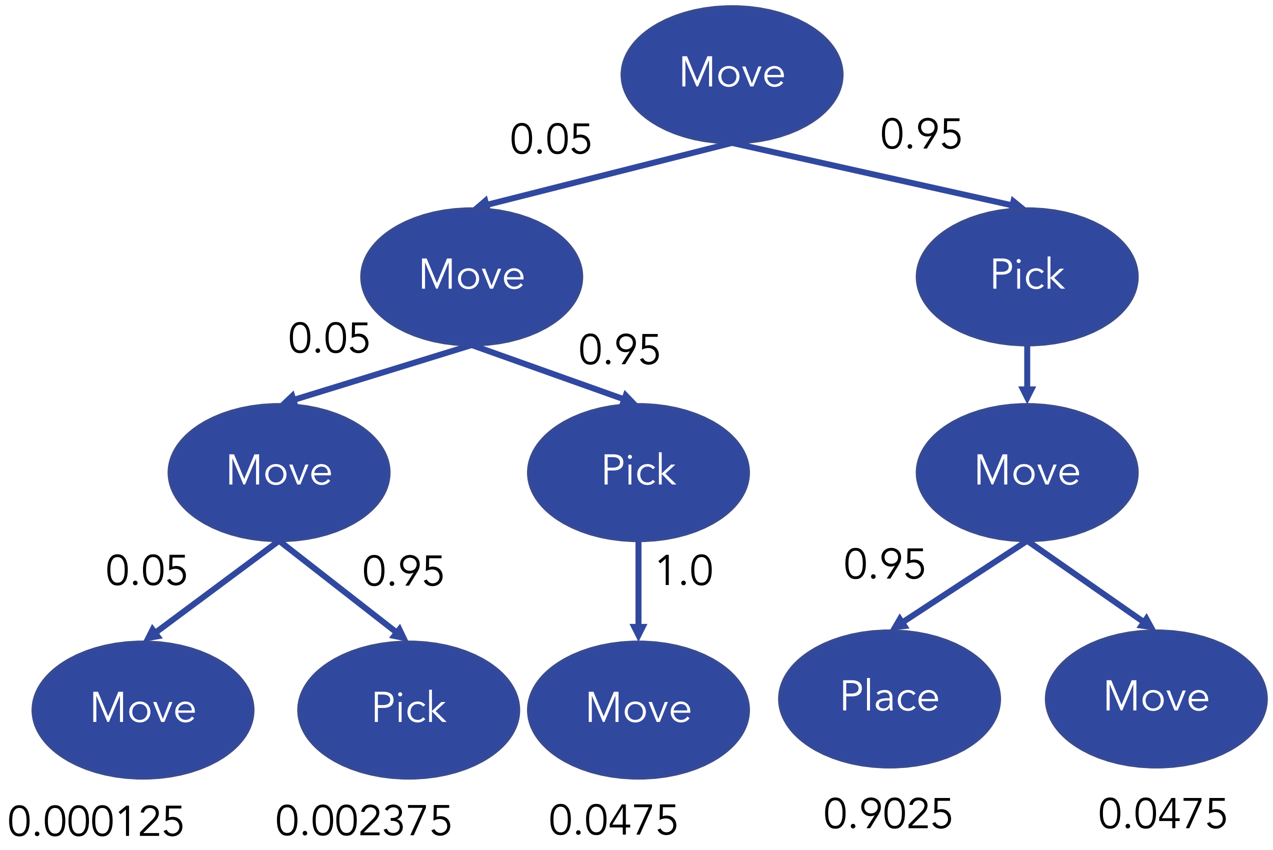
- Too many branches: Waiting to refine the entire policy tree would be inefficient



STAMP: Dealing with #branches

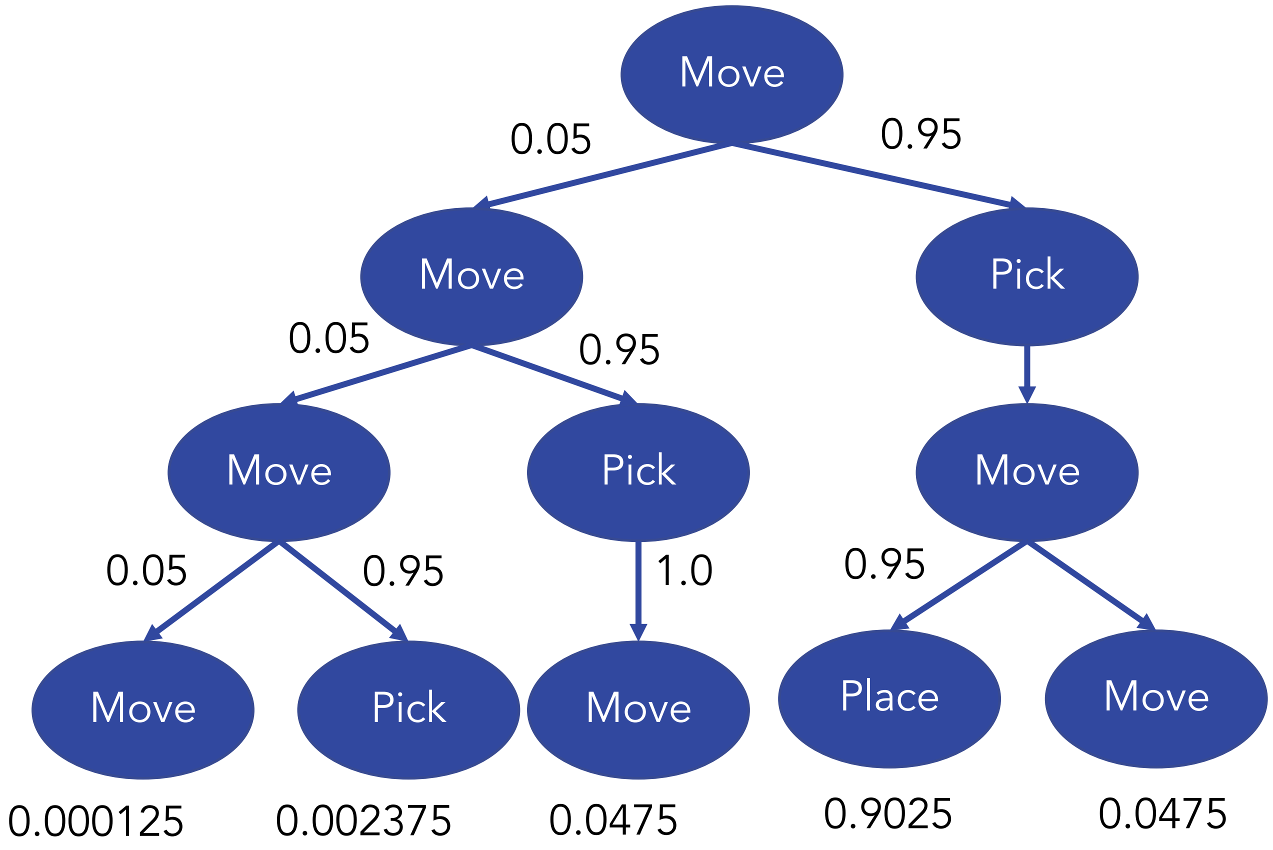
- Too many branches: Waiting to refine the entire policy tree would be inefficient

- Intuitive idea: knapsack problem with computation as cost and probability of encountering as value

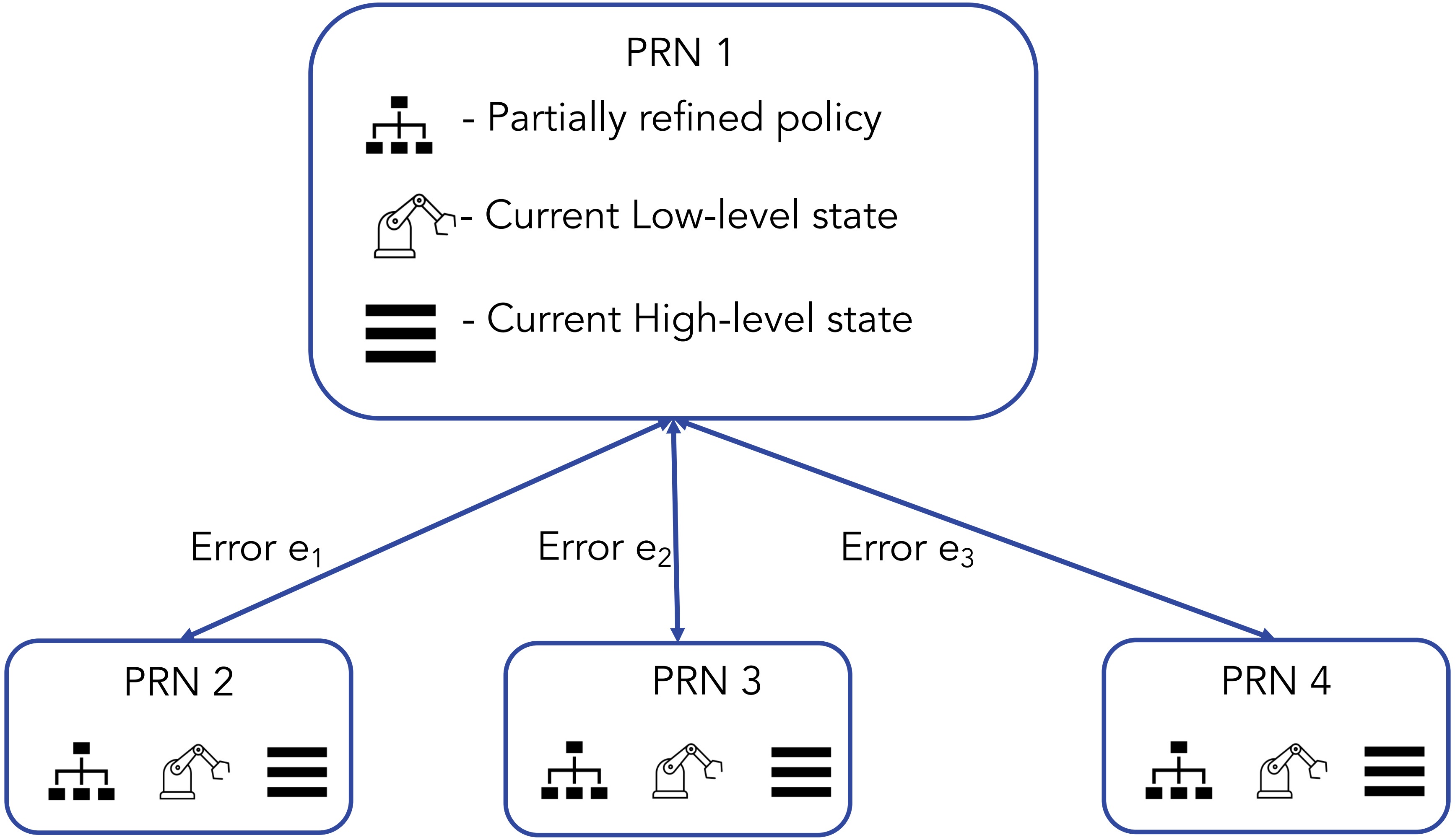


STAMP: Dealing with #branches

- Too many branches: Waiting to refine the entire policy tree would be inefficient
- Intuitive idea: knapsack problem with computation as cost and probability of encountering as value



STAMP: HPlan Algorithm

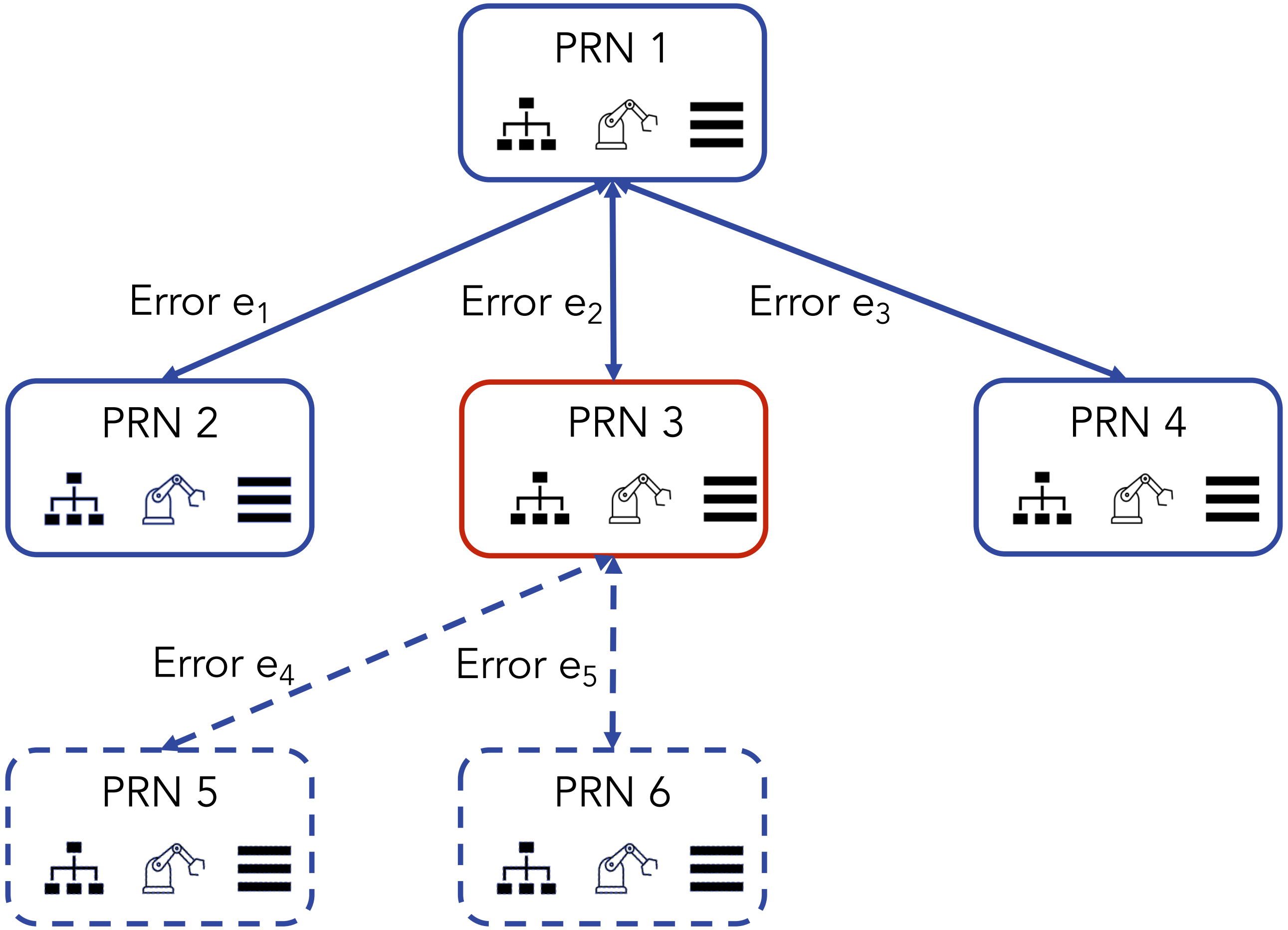


STAMP: HPlan Algorithm

The overall algorithm works as follows:

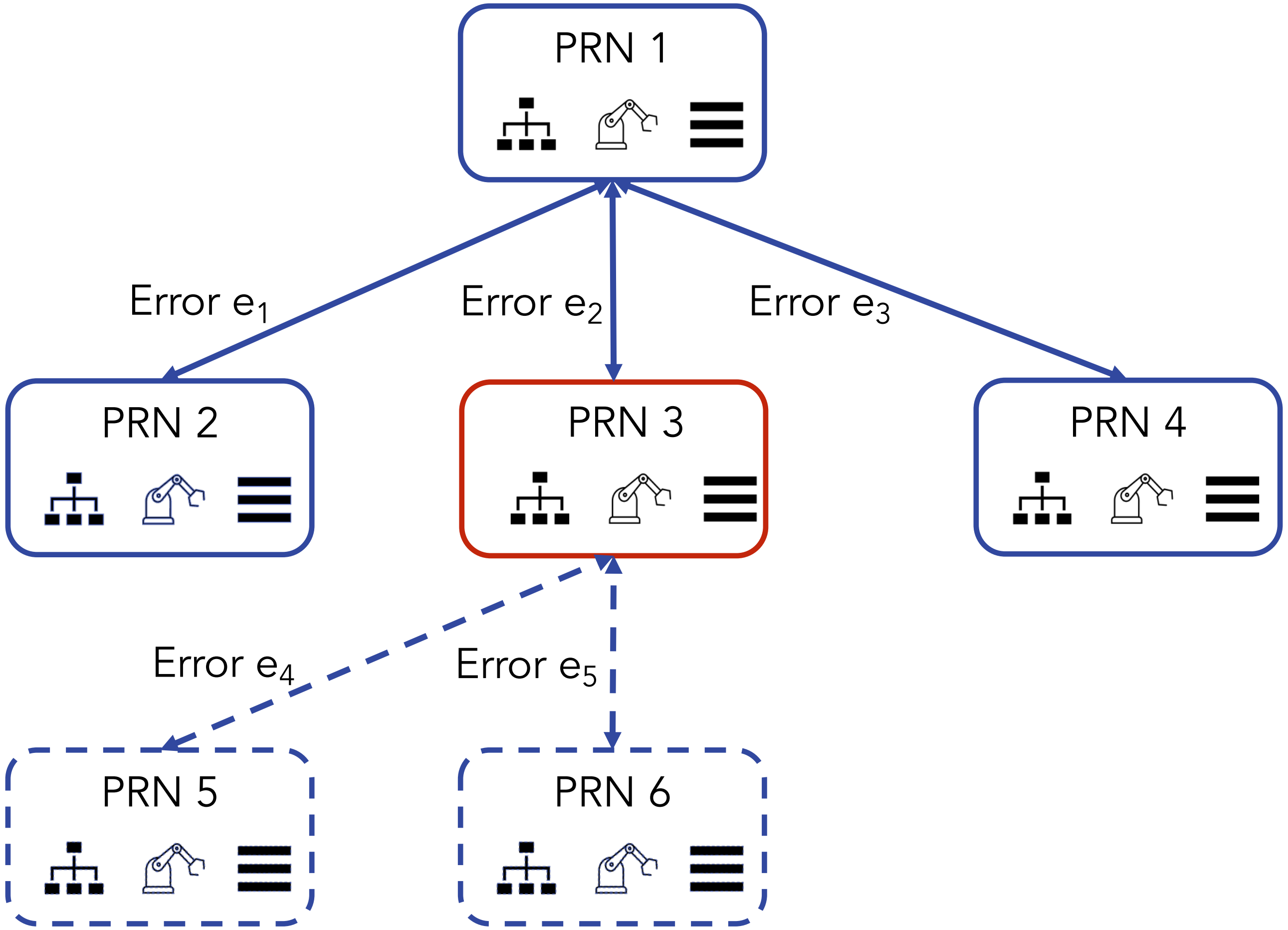
- Select a node from PRG.
- Compute an abstract policy.
- Select one of the following:
    - Explore
    - Expand the PRG

Repeat until a policy is fully refined in one of the PRG nodes.



STAMP: Why a New Refinement Algorithm?

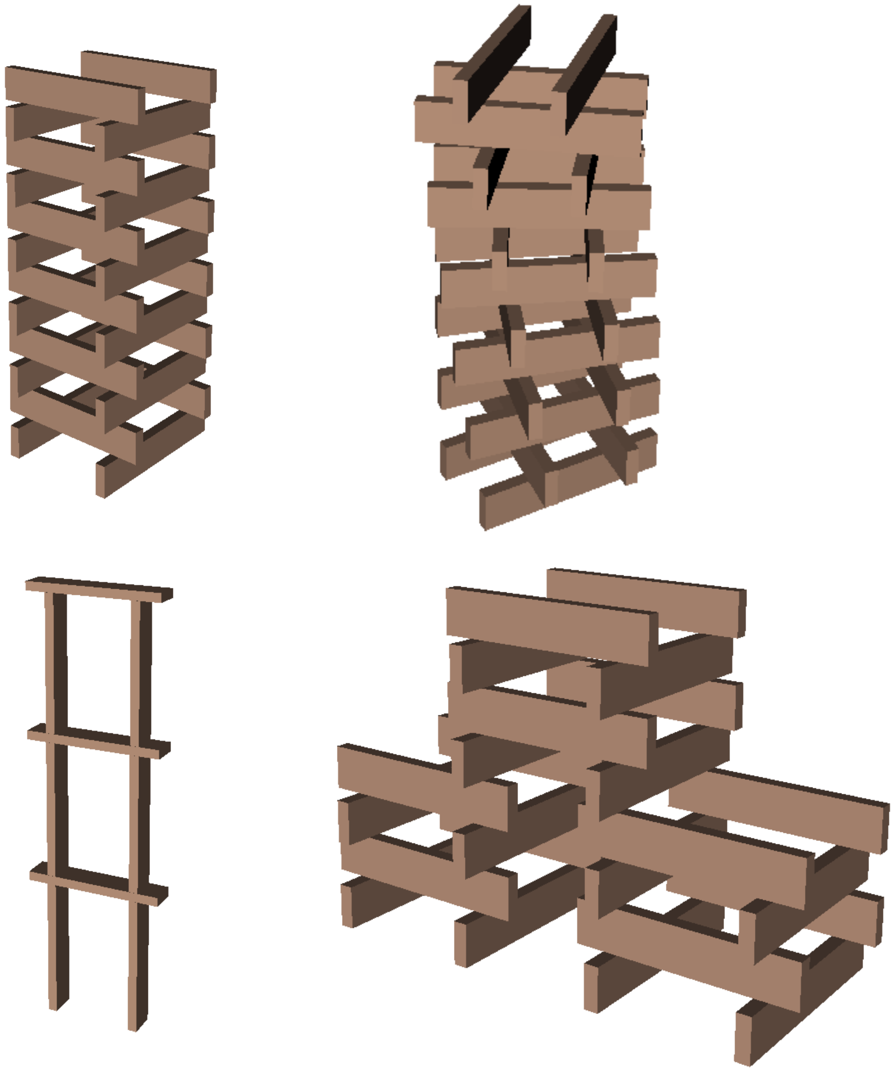
Time-based switching between nodes allows maintaining multiple abstract models and prevents getting stuck into a single abstract model.

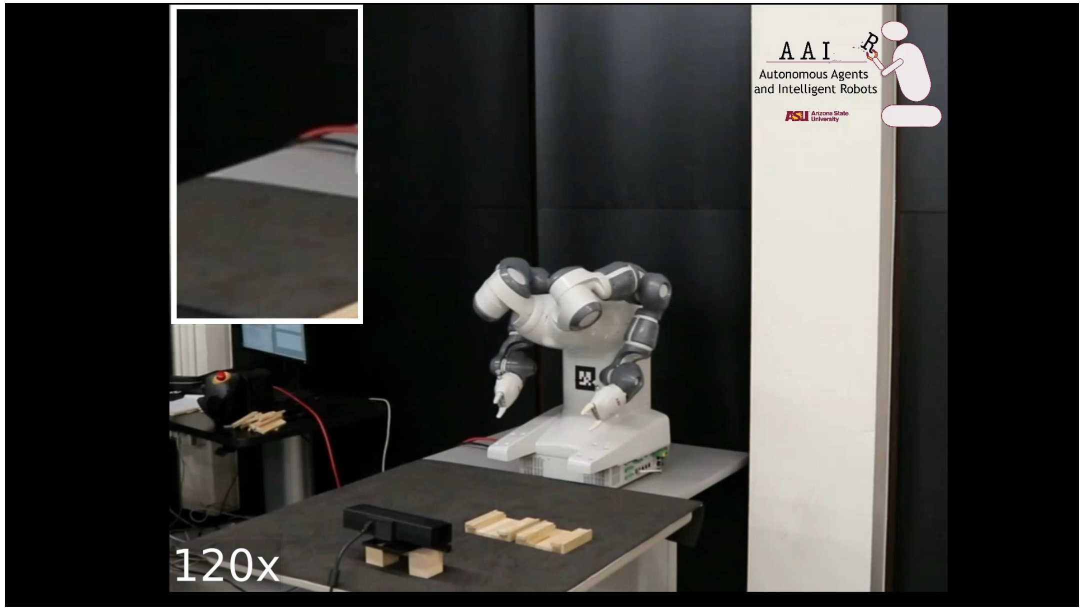


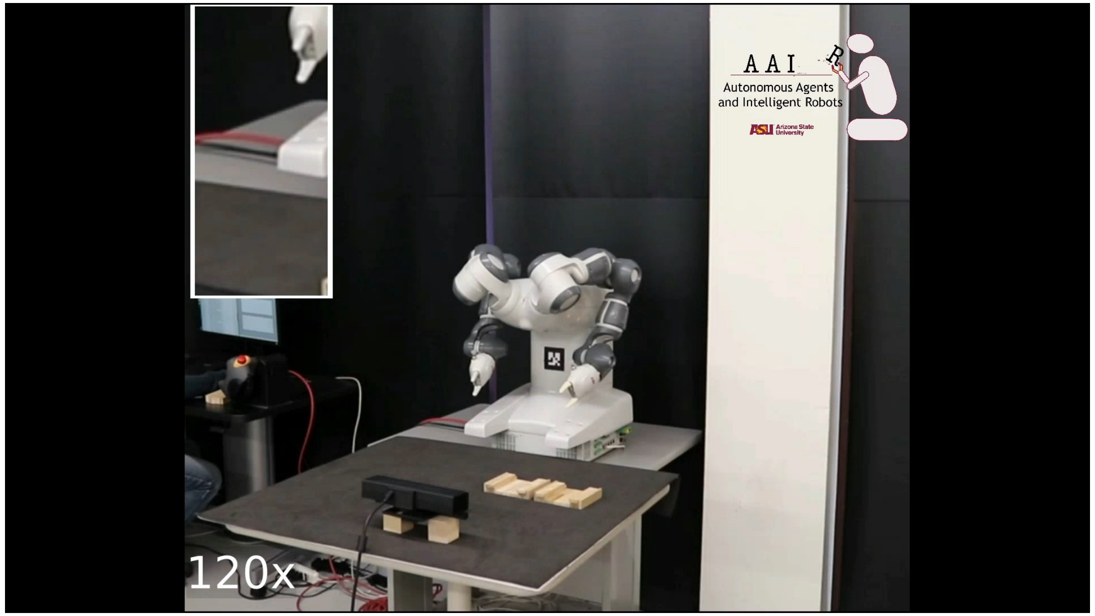
Theorem: If there exists a proper policy that reaches the goal within horizon h with probability p, and has feasible low-level refinement, then the algorithm will find it with probability *1.0* in the limit of infinite samples.

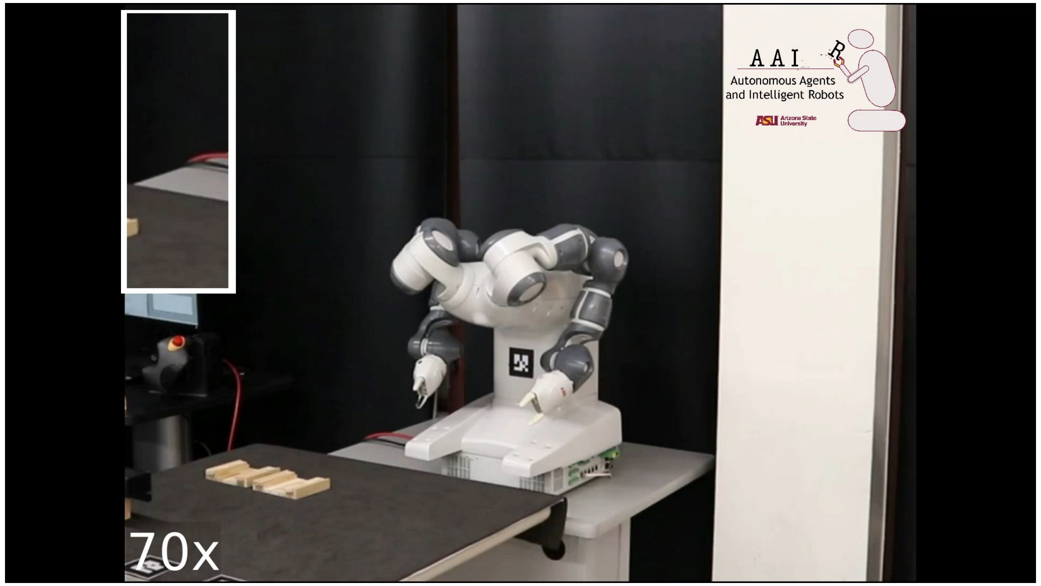
STAMP: Experiments

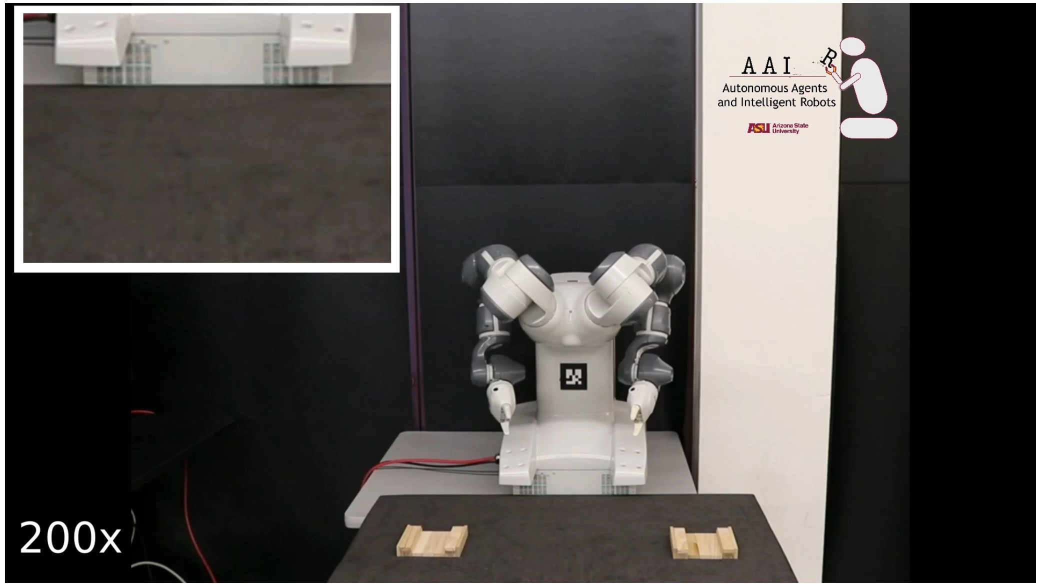
- Problem: build a desired structure using Keva planks.
  - Target design is expressed as a goal condition
- Stochasticity:
  - User may place the plank on one of two different locations
- Robot: Yumi IRB 14000




AAI
Autonomous Agents
and Intelligent Robots
120x


AAI
Autonomous Agents
and Intelligent Robots
120x


AAI
Autonomous Agents
and Intelligent Robots
70x


AAI
Autonomous Agents
and Intelligent Robots
200x

STAMP: Summary

- Inputs
  - PPDDL domain with references instead of continuous values and possibly stochastic actions
  - Generators for instantiating references
  - Subroutines for infeasibility detection and expression, given a motion plan
- Properties
  - **Handles Stochasticity**
  - + all the properties of "TMP using Interface layer"
- How does it handle P1: through symbolic references, forward-search, and lazily invoking the motion planner
- How does it handle P2: by communicating errors to the high-level planner and computing new plans

Shah, N., Vasudevan, D. K., Kumar, K., Kamojjhala, P., & Srivastava, S. (2020, May). Anytime integrated task and motion policies for stochastic environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*

High-Level Summary

	Search Space	High Level Planner	Low Level Reasoning	High-level	High Level Language	P1: Infinite Motion Plans	P2: Downward Refinability	Stochastic
aSyMov	Single	Any TP	PRM/RRT	Symbolic	PDDL	N/A	N/A	No
HPN	Dual	HPN-specific regression planner	Any MP	Hybrid	HPN-specific	generators	Regression over geometric fluents	No
IDTMP	Dual	SMT	Any MP	Variables with continuous domains	SAS	SMT	SMT	No
Symbolic Interface	Dual	Any TP	Any MP	Symbolic	PDDL	Pose generators	Identifying errors	No
PDDLStream	Dual	Any TP	Any MP	Hybrid	PDDL	optimistic samples	Iterating over all task plans	No
STAMP	Dual	Any TP	Any MP	Symbolic	PPDDL	Pose generators	Identifying errors	Yes

Limitations

Approach	Input
aSyMov	PDDL
IDTMP	PDDL + module to convert symbolic action to motion planning problems
HPN	Action descriptions + preimage for each action
Symbolic Interface	PDDL + generators
PDDLStream	PDDL + streams (generators)
STAMP	PDDL + generators

Limitations

Approach	Input
aSyMov	PDDL
IDTMP	PDDL + module to convert symbolic action to motion planning problems
HPN	Action descriptions + preimage for each action
Symbolic Interface	PDDL + generators
PDDLStream	PDDL + streams (generators)
STAMP	PDDL + generators

Requires

1) State abstractions
2) Action abstractions
3) Action descriptions
4) Samples or generators

Outline

1. Background: Why Task and Motion Planning?
2. Abstraction as a Foundation for TMP
3. Abstraction-based Approaches
4. Research Frontier: Neuro-Symbolic Learning for TMP

What needs to be learned?

1. State abstractions
2. Temporal abstractions
    1. Identifying actions
    2. Learning action descriptions
3. Learning samplers / generators for action refinements

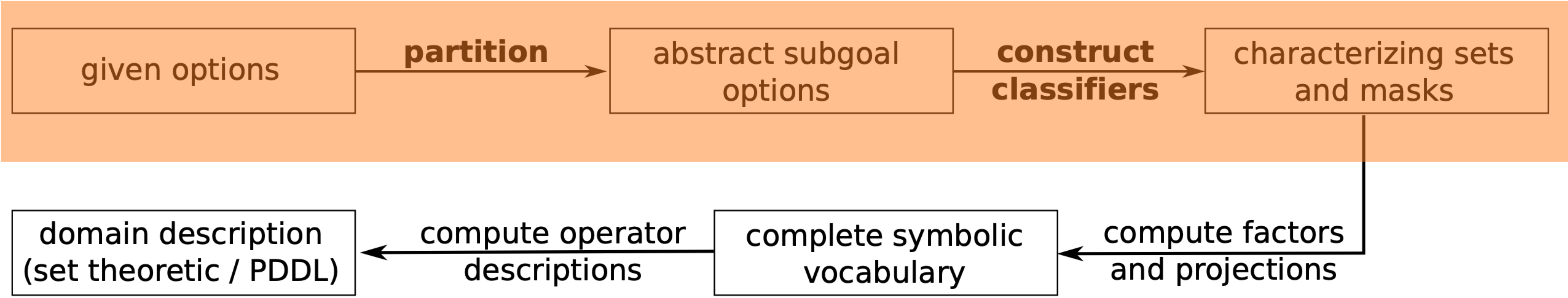
Learning State Abstractions Given High-level Actions

Skills-to-Symbols

- Core idea: Learned symbolic model in a PDDL representation
- Input: State variables with low-level continuous values
- Output: A symbolic PDDL model
- What is given:
  - Options masks -- set of low-level state variables relevant to an option
  - Abstract goal options -- Options that achieves termination sets with probability 1.0 expressed using only relevant state variables

Konidaris, G., Kaelbling, L. P., & Lozano-Perez, T. (2018). From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*

Skills-to-symbols



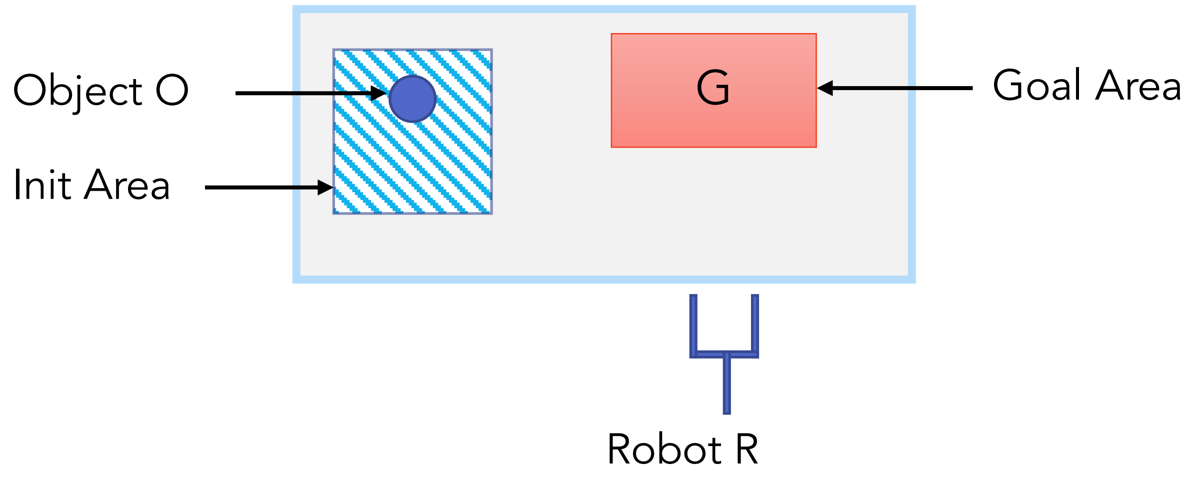
Skills-to-symbols: Computing Factors

Variables

$s_1 = \ Pose_r$ = Robot pose

$s_2 = \ Pose_o$ = Object pose

$s_3 = \ Attached$ = Whether the object is picked or not



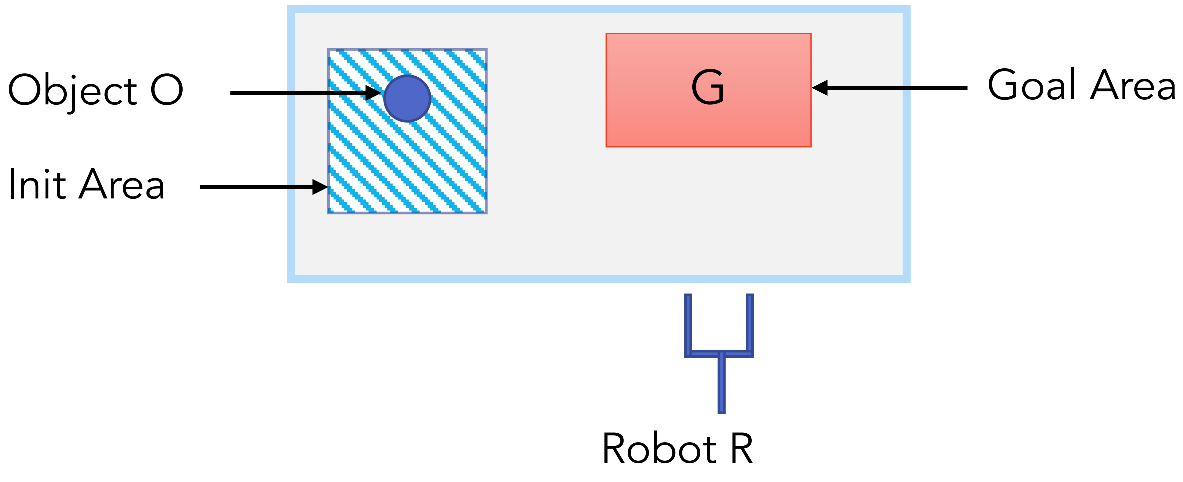
Skills-to-symbols: Computing Factors

Variables

$s_1 = \ Pose_r$ = Robot pose

$s_2 = \ Pose_o$ = Object pose

$s_3 = \ Attached$ = Whether the object is picked or not



Options

o_1 = Move (r) : s_1 // moves the robot

o_2 =Grab (o). : s_1, s_2 // grabs the object

o_3 = UnGrab(o). : s_1, s_2 // releases the object

o_4 = Transport(o) : $s_1,\ s_3$ // moves the object

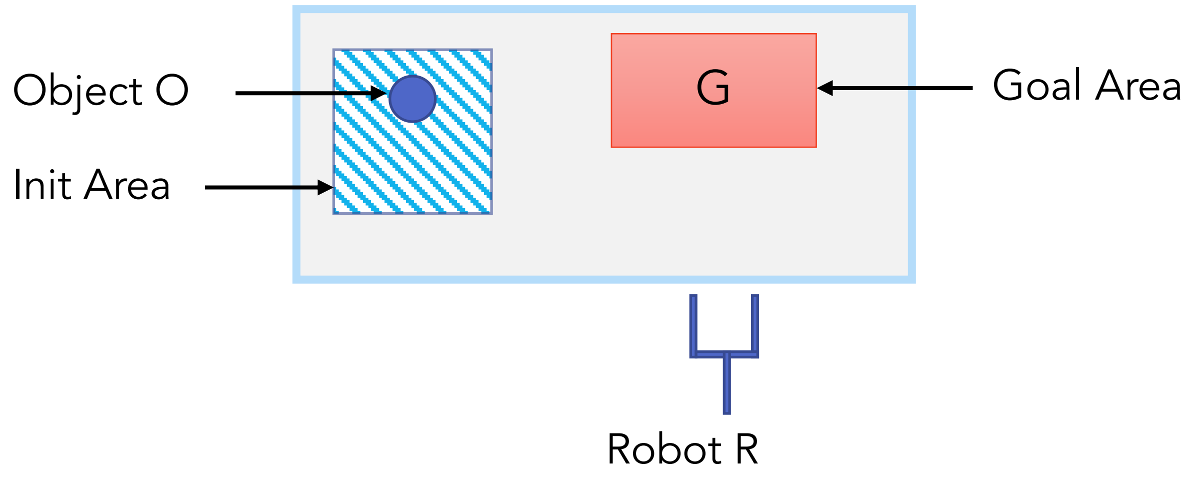
Skills-to-symbols: Computing Factors

Variables

$s_1 = \ Pose_r$ = Robot pose

$s_2 = \ Pose_o$ = Object pose

$s_3 = \ Attached$ = Whether the object is picked or not



Options

o_1 = Move (r) : s_1 // moves the robot

o_2 =Grab (o). : s_1, s_2 // grabs the object

o_3 = UnGrab(o). : s_1, s_2 // releases the object

o_4 = Transport(o) : $s_1, \ s_3$ // moves the object

Variable	Option
s_1	o_1, o_4
s_2	o_2, o_3
s_3	o_4

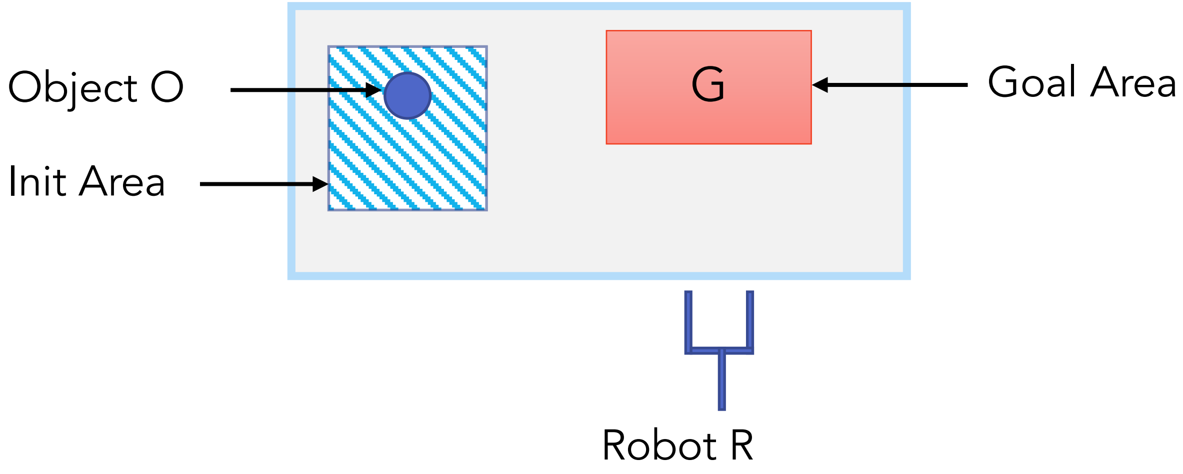
Skills-to-symbols: Computing Factors

Variables

$s_1 = \ Pose_r$ = Robot pose

$s_2 = \ Pose_o$ = Object pose

$s_3 = \ Attached$ = Whether the object is picked or not



Options

o_1 = Move (r) : s_1 // moves the robot

o_2 =Grab (o). : s_1, s_2 // grabs the object

o_3 = UnGrab(o). : s_1, s_2 // releases the object

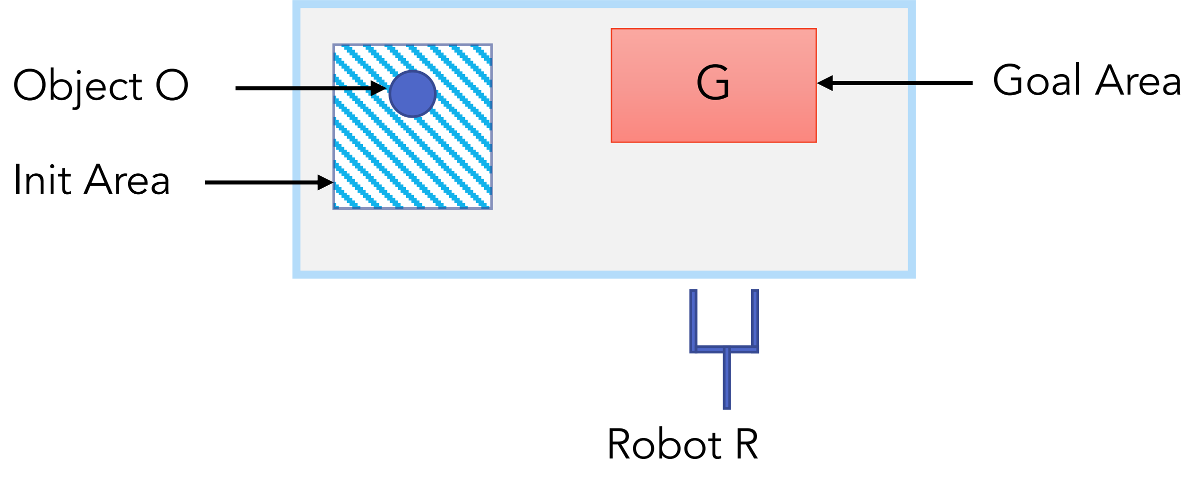
o_4 = Transport(o) : $s_1,\ s_3$ // moves the object

	Variable	Option
f_1	s_1	$o_1,\ o_4$
f_2	s_2	o_2, o_3
f_3	s_3	o_4

Skills-to-symbols: Computing Factors

Variables

$s_1 = \ Pose_r$ = Robot pose
$s_2 = \ Pose_o$ = Object pose
$s_3 = \ Attached$ = Whether the object is picked or not



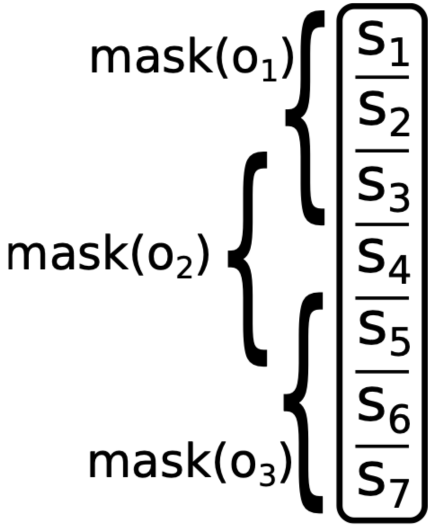
Options

o_1 = Move (r) : s_1 // moves the robot
o_2 =Grab (o). : s_1, s_2 // grabs the object
o_3 = UnGrab(o). : s_1, s_2 // releases the object
o_4 = Transport(o) : s_1, s_3 // moves the object

	Variable	Option
f_1	s_1	o_1, o_4
f_2	s_2	o_2, o_3
f_3	s_3	o_4

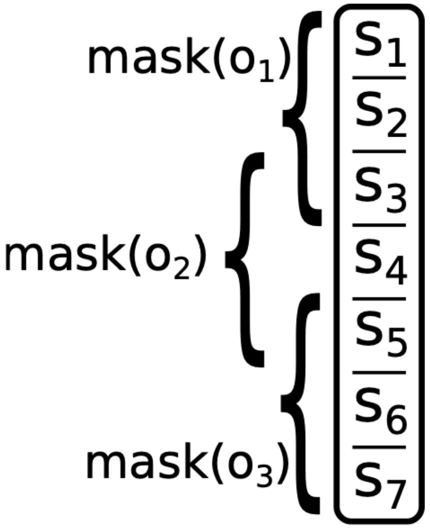
Simple case!!

Skills-to-symbols: Computing Factors



Factor	State Variables	Options
f_1	s_1, s_2	o_1
f_2	s_3	o_1, o_2
f_3	s_4	o_2
f_4	s_5	o_2, o_3
f_5	s_6, s_7	o_3

Skills-to-symbols: Assigning Propositions



Factor	State Variables	Options
f_1	$s_1,\ s_2$	o_1
f_2	s_3	$o_1,\ o_2$
f_3	s_4	o_2
f_4	s_5	$o_2,\ o_3$
f_5	$s_6,\ s_7$	o_3

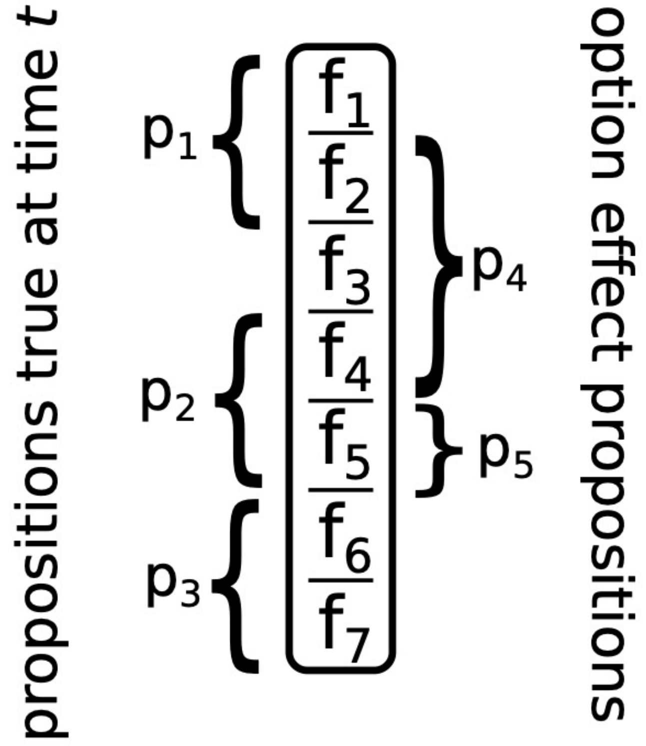
Identify propositions:

1. Identify independent factors -- factors that can be used to decompose the effect
   create a proposition for each independent factor

2. For the remaining collection of factor -- create a proposition

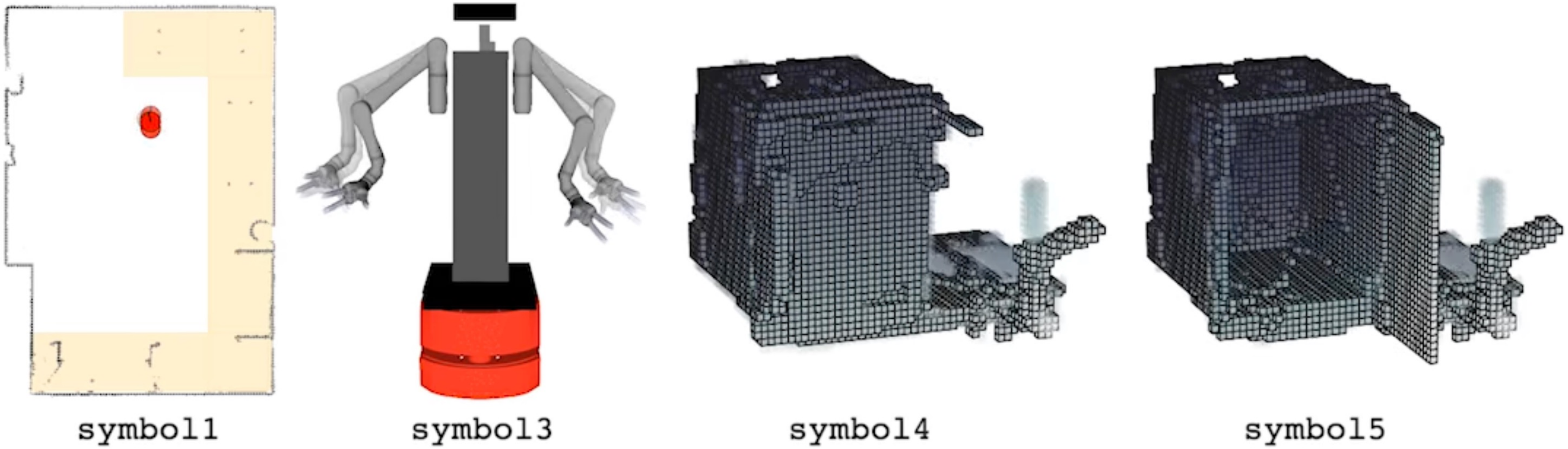
Skills-to-symbols: Learning Symbolic Operators

- For each option:
    - Identify the added and removed propositions in the effect set (termination set)
    - Identify the propositions in the propositions in the initiation set

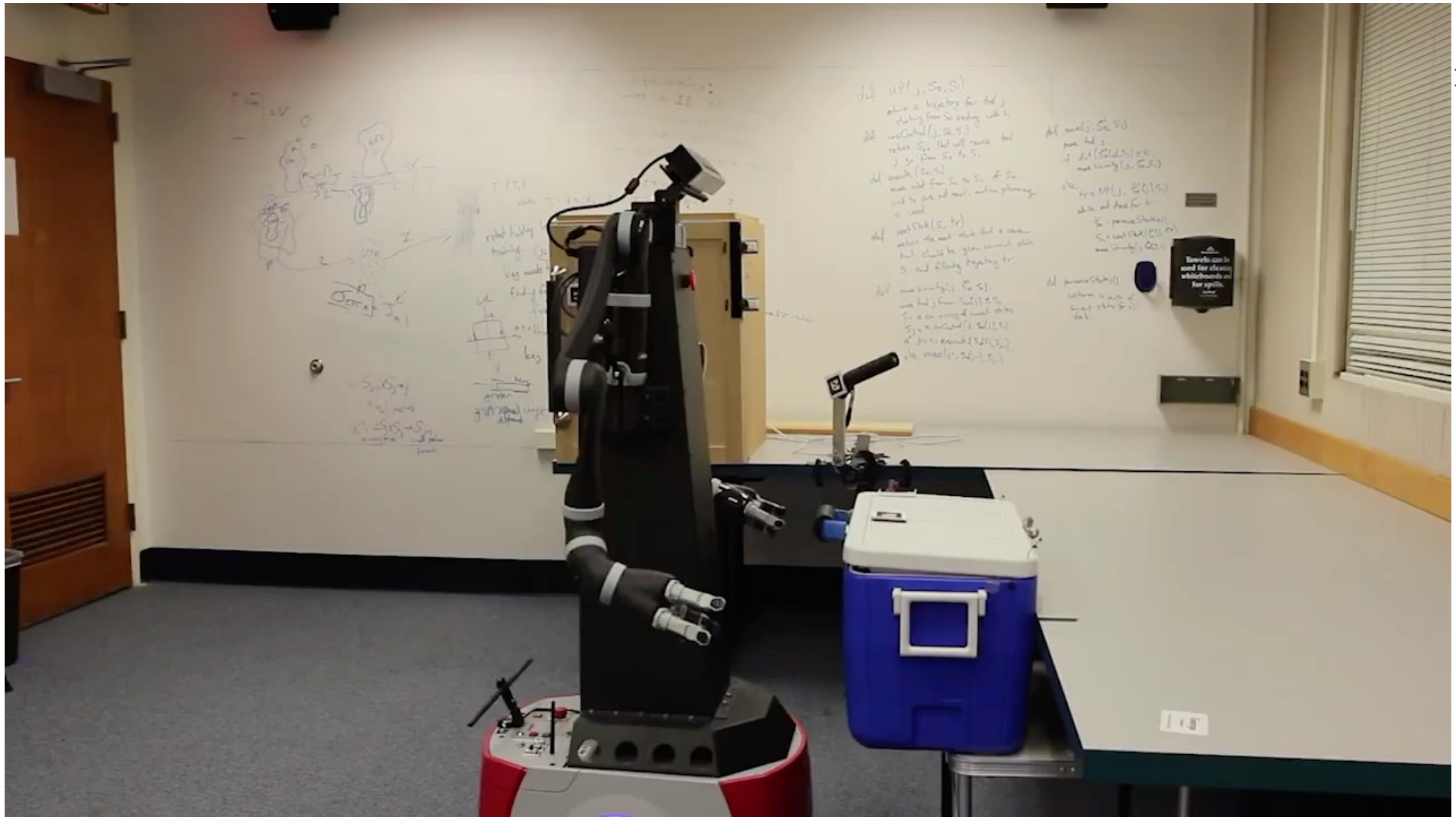


Skills-to-symbols: Example

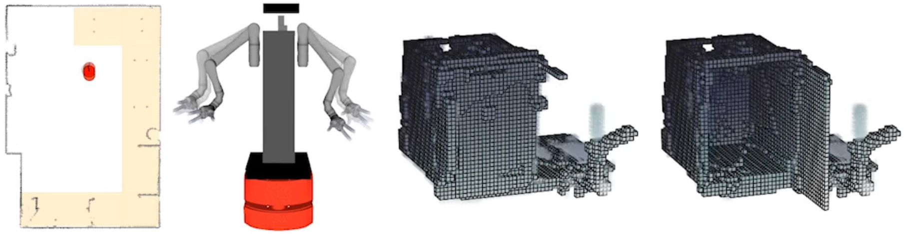
```
(:action cupboard_open1
 :parameters ()
 :precondition (and (symbol1) (symbol3) (symbol4))
 :effect       (and (symbol5) (not (symbol4))
                    (decrease (reward) 67.44))
)
```



Skills-to-symbols: Example



```
(:action cupboard_open1
 :parameters ()
 :precondition (and (symbol1) (symbol3) (symbol4))
 :effect       (and (symbol5) (not (symbol4))
                    (decrease (reward) 67.44))
)
```



symbol1 symbol3 symbol4 symbol5

High-Level Summary

	Abstract States	Abstract Actions	Low-level behaviors	Samplers	C-Space changes
Skills to Symbols	Learned	Input	Input	Input	Yes

Learning High-Level Actions Given State Abstractions

Learning Symbolic Operators

- Core idea: Learn descriptions (precondition and effects) of high-level actions for task and motion planning

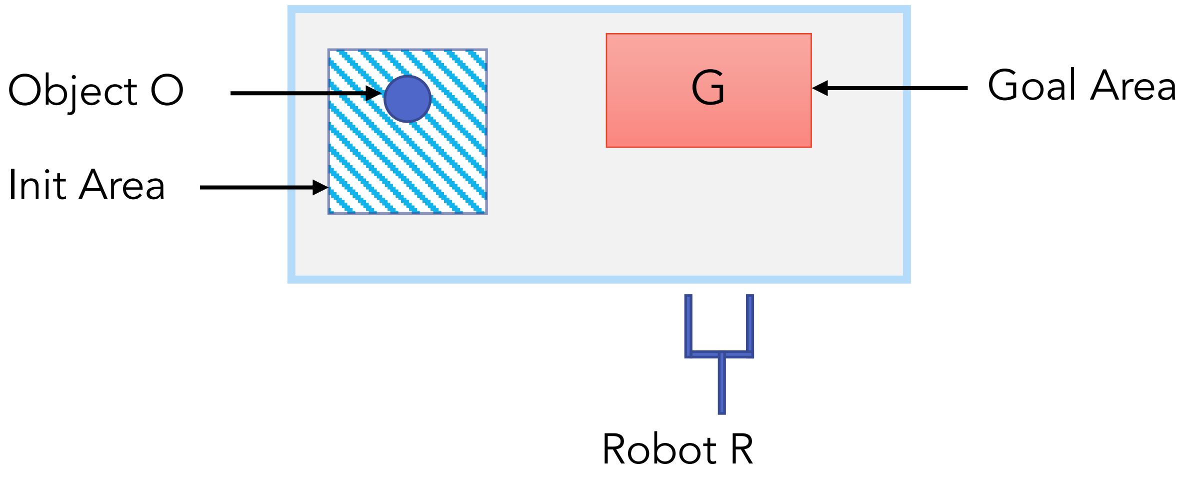
- What is provided:
  - State abstractions -- predicates and interpretation of predicates
  - Controllers -- low-level behaviors parameterized by typed objects
  - Samplers -- generators for discretizing high-level arguments
  - Data -- $\{x_i,\ a_i,\ x_{i+1}\}$ for a set of training tasks

Silver, T., Chitnis, R., Tenenbaum, J., Kaelbling, L. P., & Lozano-Pérez, T. (2021, September). Learning symbolic operators for task and motion planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*

Learning Symbolic Operators: Approach

What is given?

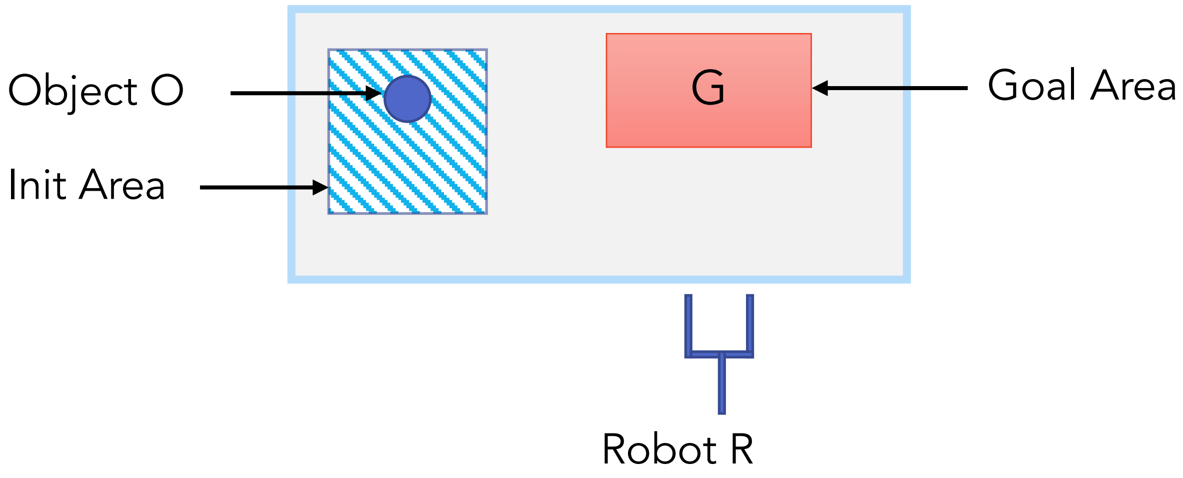
Data	Controllers	Predicates
x_0, c_1, x_1	Move(o)	At(?o – obj ?a –area)
….	Pick(o)	RobotAt(?a – area)
x_{n-1}, c_n, x_n	Place(o)	Holding(?o – obj)



Learning Symbolic Operators: Approach

What is given?

Data	Controllers	Predicates
x_0, c_1, x_1	Move(o)	At(?o – obj ?a –area)
$\ldots.$	Pick(o)	RobotAt(?a – area)
x_{n-1}, c_n, x_n	Place(o)	Holding(?o – obj)



1. Convert low-level states into abstract states in the data

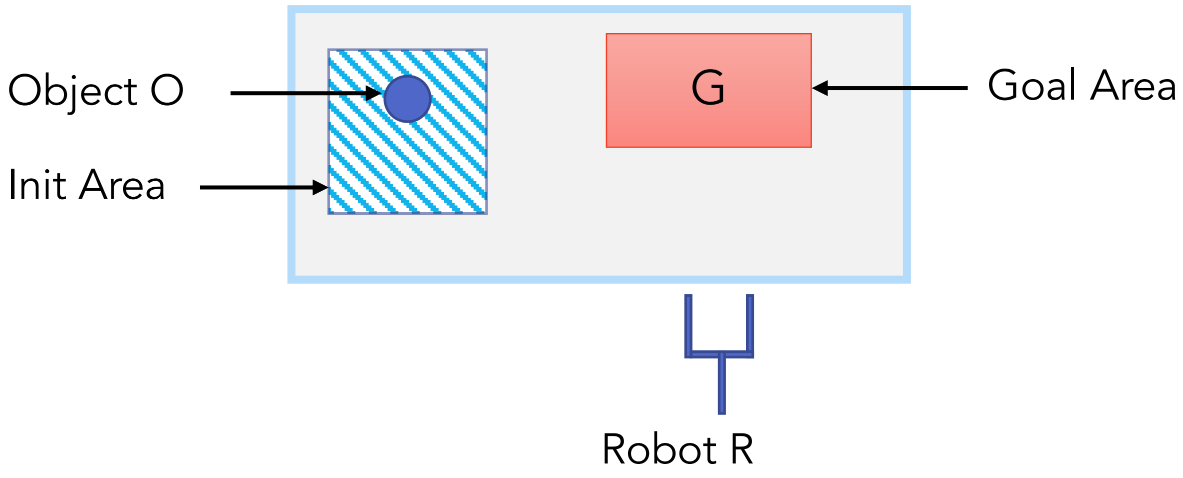
Data		Abstract Data
x_0, c_1, x_1	$\longrightarrow$	s_0, c_1, s_1
$\ldots.$		$\ldots.$
x_{n-1}, c_n, x_n		$\boxed{s_{n-1}}, c_n, s_n$

At(o,init)
RobotAt(a1)
¬Holding(o)

Learning Symbolic Operators: Approach

What is given?

Data	Controllers	Predicates
x_0, c_1, x_1	Move(o)	At(?o – obj ?a –area)
….	Pick(o)	RobotAt(?a – area)
x_{n-1}, c_n, x_n	Place(o)	Holding(?o – obj)



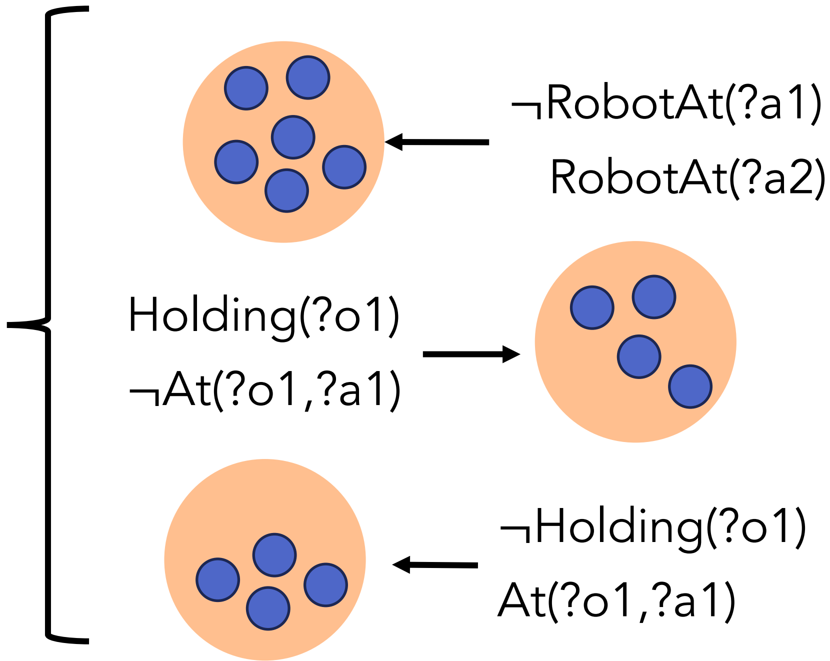
1. Convert low-level states into abstract states in the data
2. Cluster transitions using the lifted effects

Abstract Data

s_0, c_1, s_1

….

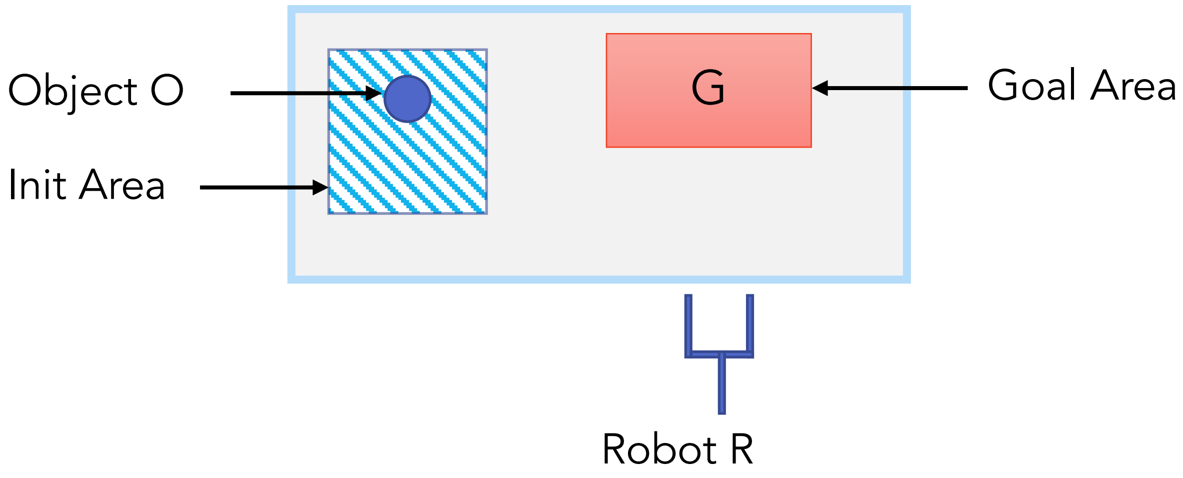
s_{n-1}, c_n, s_n



Learning Symbolic Operators: Approach

What is given?

Data	Controllers	Predicates
x_0, c_1, x_1	Move(o)	At(?o – obj ?a –area)
….	Pick(o)	RobotAt(?a – area)
x_{n-1}, c_n, x_n	Place(o)	Holding(?o – obj)



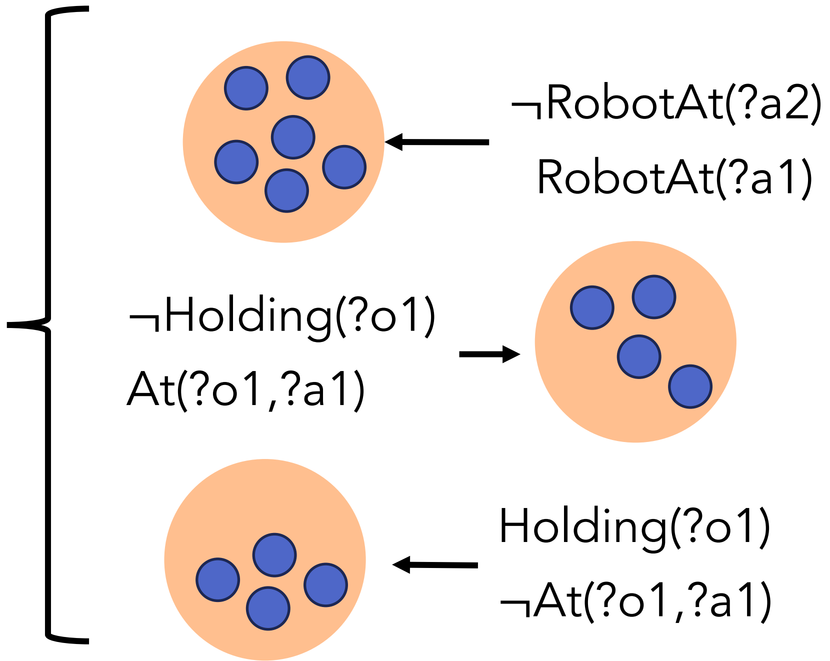
1. Convert low-level states into abstract states in the data
2. Cluster transitions using the lifted effects
3. Learn preconditions for each cluster

Abstract Data

s_0, c_1, s_1

….

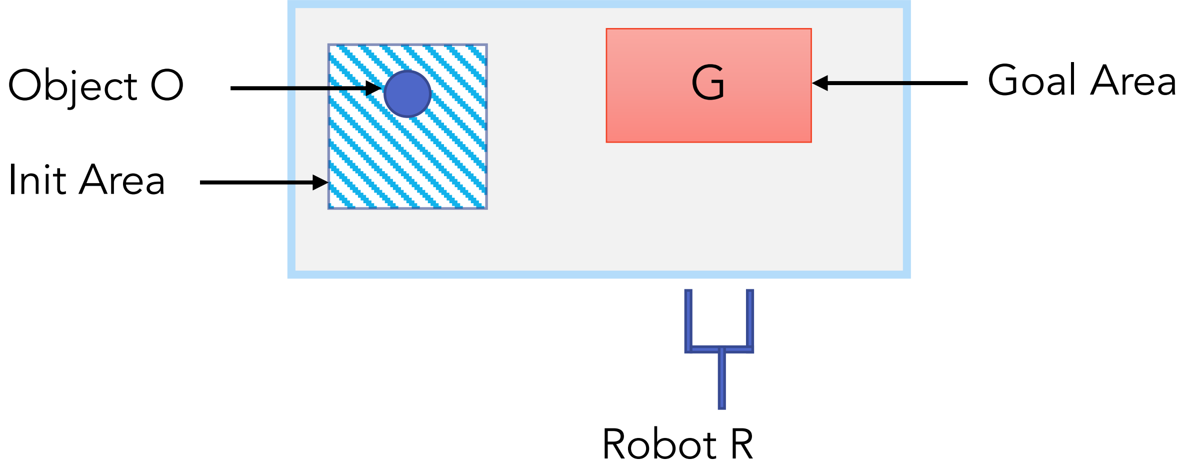
s_{n-1}, c_n, s_n



Learning Symbolic Operators: Approach

What is given?

Data	Controllers	Predicates
x_0, c_1, x_1	Move(o)	At(?o – obj ?a –area)
….	Pick(o)	RobotAt(?a – area)
x_{n-1}, c_n, x_n	Place(o)	Holding(?o – obj)



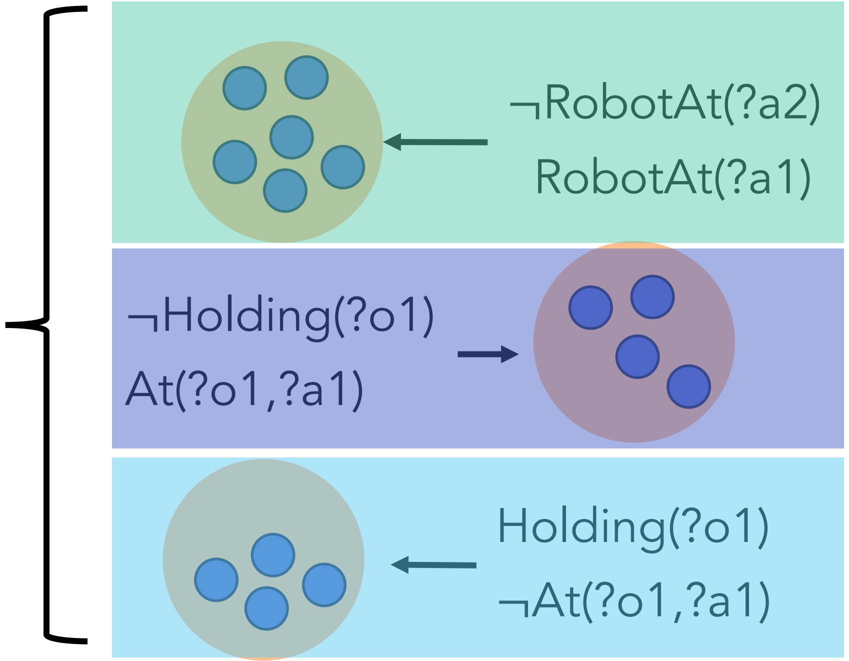
1. Convert low-level states into abstract states in the data
2. Cluster transitions using the lifted effects
3. Learn preconditions for each cluster
4. Make an operator for each cluster

Abstract Data

s_0, c_1, s_1

….

s_{n-1}, c_n, s_n



TMP with Learned High-level Actions

TMP framework requires:

- Abstract states
- Abstract actions
- Low-level behaviors / motion planner
- Samplers / generators

TMP with Learned High-level Actions

TMP framework requires:

Abstract states -- Input

Abstract actions -- Learned

Low-level behaviors / motion planner -- Input

Samplers / generators -- Input

High-Level Summary

	Abstract States	Abstract Actions	Low-level behaviors	Samplers	C-Space changes
Skills to Symbols	Learned	Input	Input	Input	Yes
[illegible], 2021	Input	Learned	Input	Input	Yes

Learning NSRTs

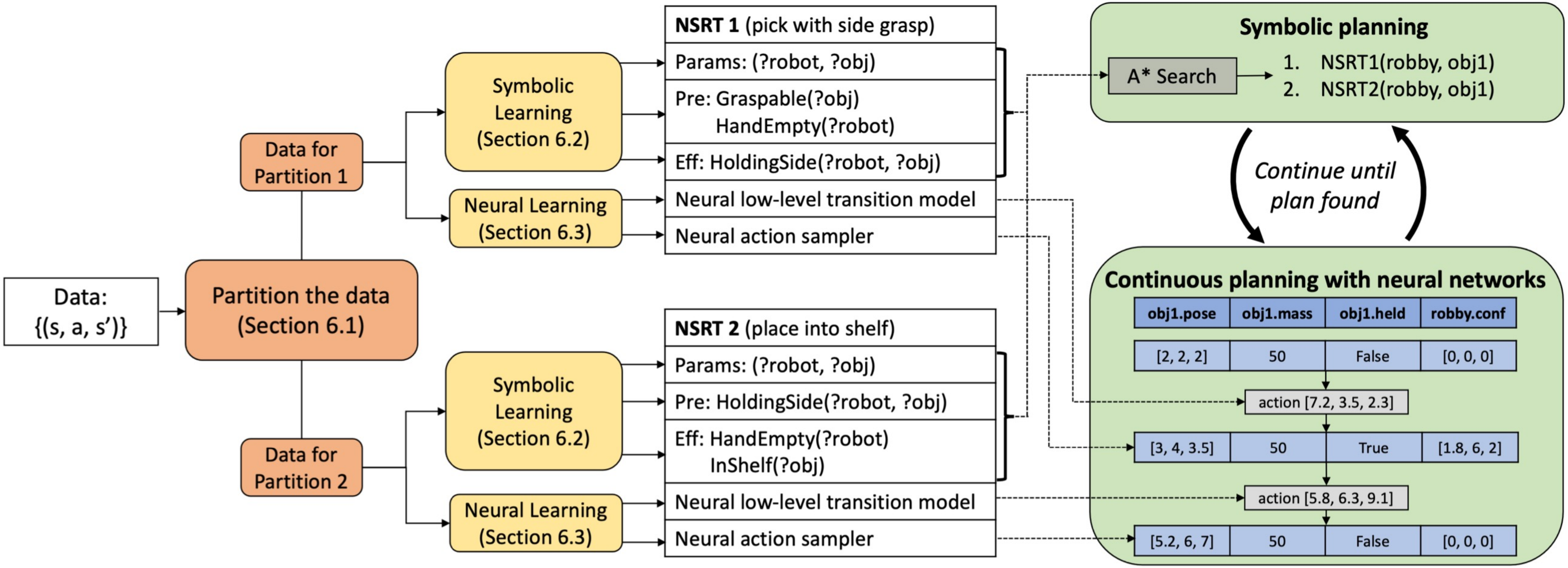
- Core idea: learn high-level abstract actions in the form of NSRTs
- NSRTs = Neuro-symbolic relational transitions

- NSRTs include:
  - High-level action components: parameters, preconditions, effects
  - Low-level reactive policy: $\pi(a|x)$

- What is provided:
  - Predicates -- state abstraction
  - $f: X \times A \to X$ a known low-level deterministic transition function

Silver, T., Athalye, A., Tenenbaum, J. B., Lozano-Pérez, T., & Kaelbling, L. P. (2023, March). Learning Neuro-Symbolic Skills for Bilevel Planning. In *Conference on Robot Learning*. PMLR

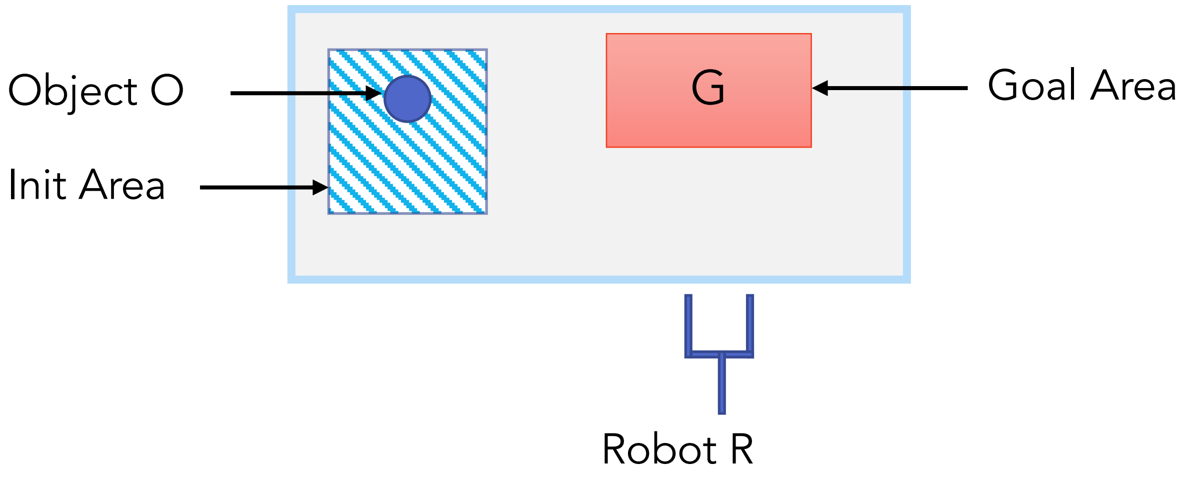
Learning NSRTs: Overall Approach



Learning NSRTs: Creating Data Partitions and Effects

What is given?

Data	Predicates
x_0, a_1, x_1	At(?o – obj ?a –area)
$\ldots.$	RobotAt(?a – area)
x_{n-1}, a_n, x_n	Holding(?o – obj)

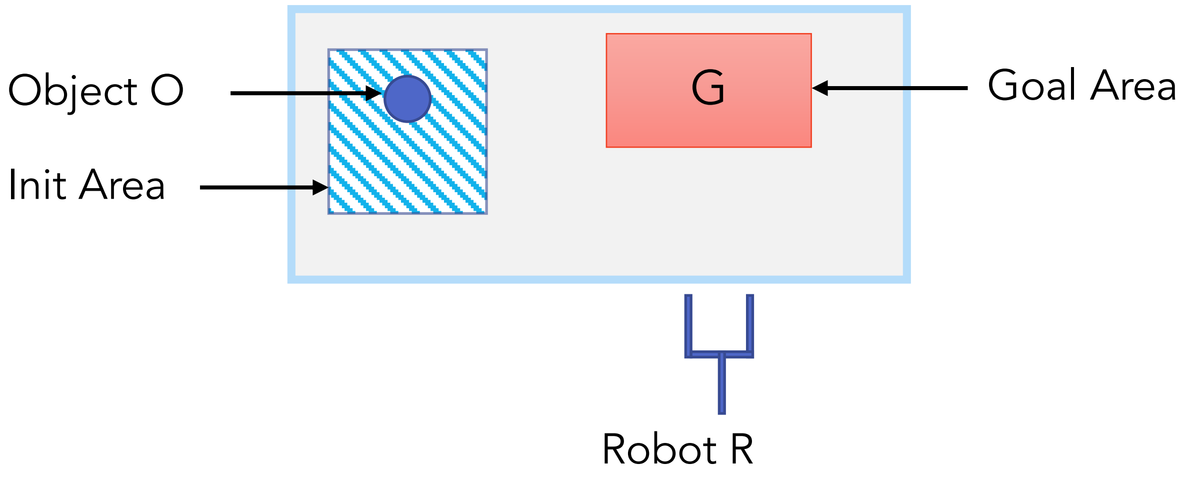


Data		Abstract Data	Partition 1
x_0, a_1, x_1	$\longrightarrow$	s_0, a_1, s_0	s_0, a_1, s_0
$\ldots.$		$\ldots.$	$\ldots.$
x_{n-1}, a_n, x_n		s_{n-1}, a_n, s_n	s_0, a_n, s_1

Learning NSRTs: Creating Data Partitions and Effects

What is given?

Data	Predicates
x_0, a_1, x_1	At(?o – obj ?a –area)
$\dots.$	RobotAt(?a – area)
x_{n-1}, a_n, x_n	Holding(?o – obj)



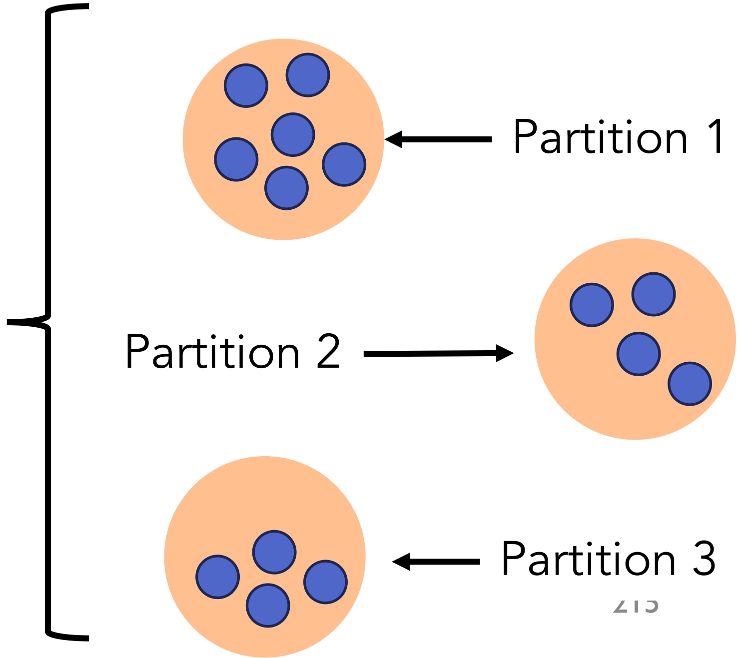
Data		Abstract Data
x_0, a_1, x_1	$\longrightarrow$	s_0, a_1, s_0
$\dots.$		$\dots.$
x_{n-1}, a_n, x_n		s_{n-1}, a_n, s_n

Abstract Data

s_0, a_1, s_1

$\dots.$

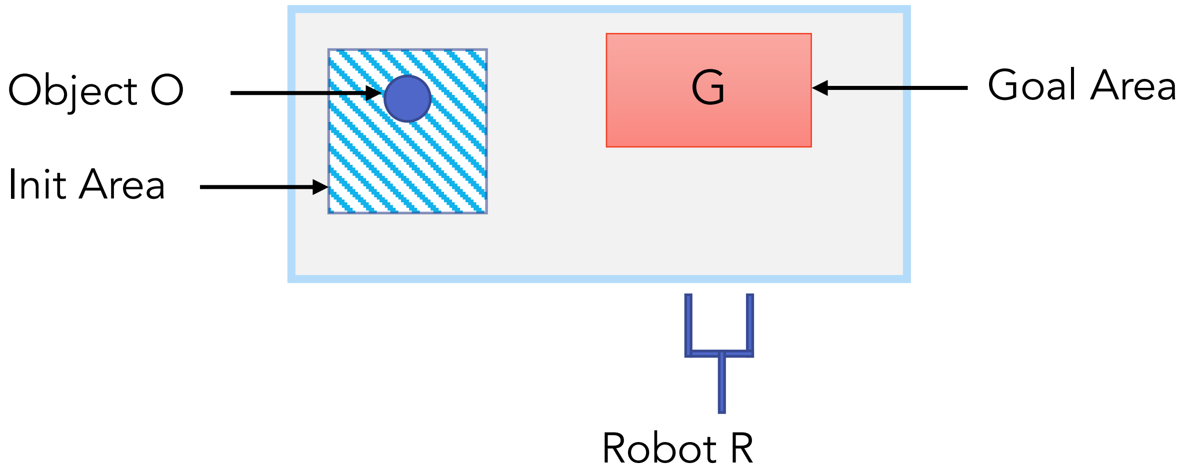
s_{n-1}, a_n, s_n



Learning NSRTs: Learning Preconditions

What is given?

Data	Predicates
x_0, a_1, x_1	At(?o – obj ?a –area)
….	RobotAt(?a – area)
x_{n-1}, a_n, x_n	Holding(?o – obj)



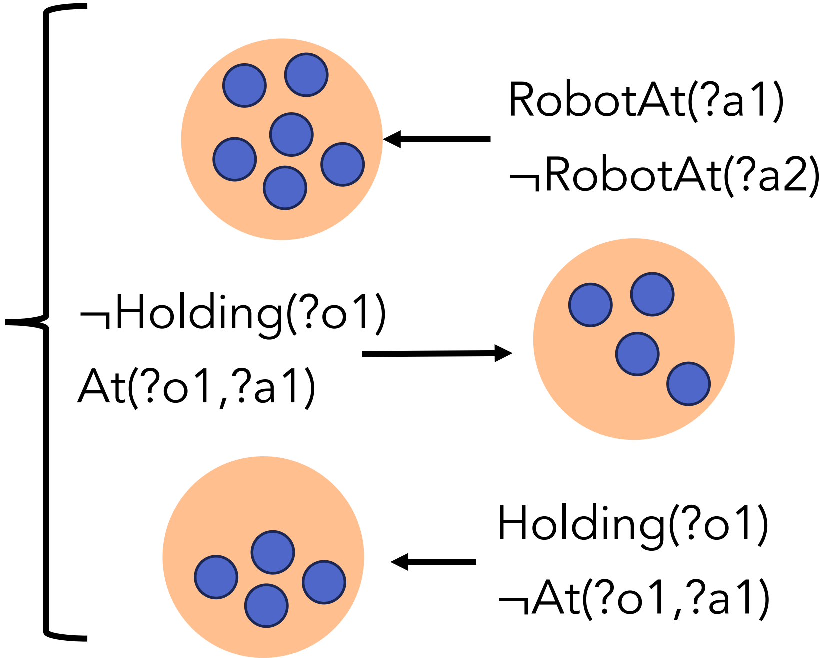
Data		Abstract Data
x_0, a_1, x_1	$\longrightarrow$	s_0, a_1, s_0
….		….
x_{n-1}, a_n, x_n		s_{n-1}, a_n, s_n

Abstract Data

s_0, a_1, s_1

….

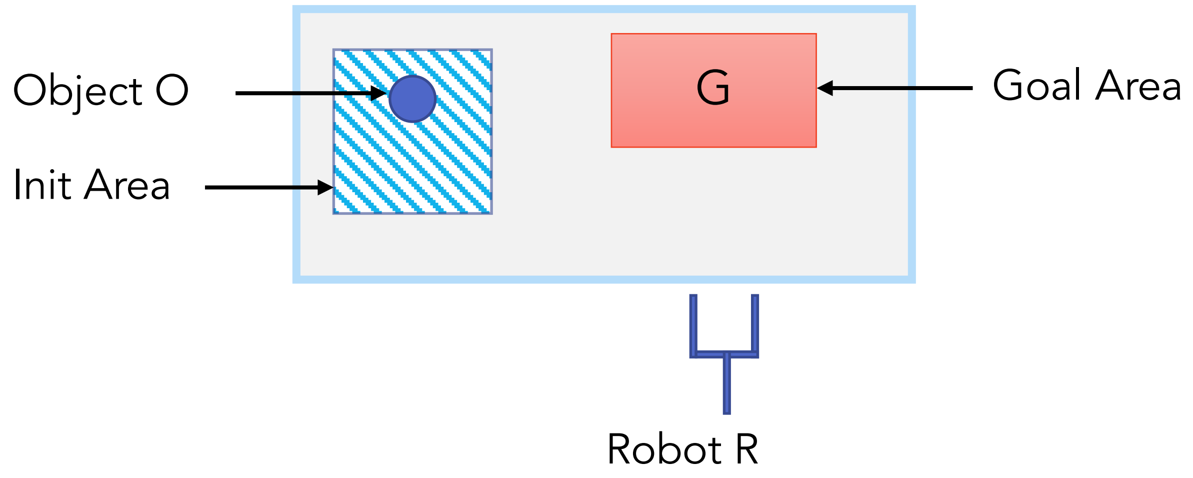
s_{n-1}, a_n, s_n



Learning NSRTs: Reactive Low-Level Policy

What is given?

Data	Predicates
x_0, a_1, x_1	At(?o – obj ?a –area)
….	RobotAt(?a – area)
x_{n-1}, a_n, x_n	Holding(?o – obj)



Partition 1		Low-level Data		
s_0, a_1, s_0	→	x_0, a_1, x_1	→	Learn a regression model $\pi(a\|x)$
….		….		
s_0, a_n, s_1		x_{n-1}, a_n, s_n		

High-Level Summary

	Abstract States	Abstract Actions	Low-level behaviors	Samplers	C-Space changes
Skills to Symbols	Learned	Input	Input	Input	Yes
[illegible], 2021	Input	Learned	Input	Input	Yes
[illegible], 2022	Input	Learned	Learned	Learned	Yes

Learning State and Action Abstractions

HARP: A Neuro-Symbolic Motion Planner

- Core idea: Learn to zero-shot create state and action abstractions simultaneously using *critical regions*

- What is provided?
  - A set of training environments
  - A random problem generators (random initial and goal configuration)
  - A motion planner

- What is learned?
  - A method to **zero-shot** create **state** and **action** abstractions for **unseen environments**

Shah, N., & Srivastava, S. (2022, May). Using Deep Learning to Bootstrap Abstractions for Hierarchical Robot Planning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*

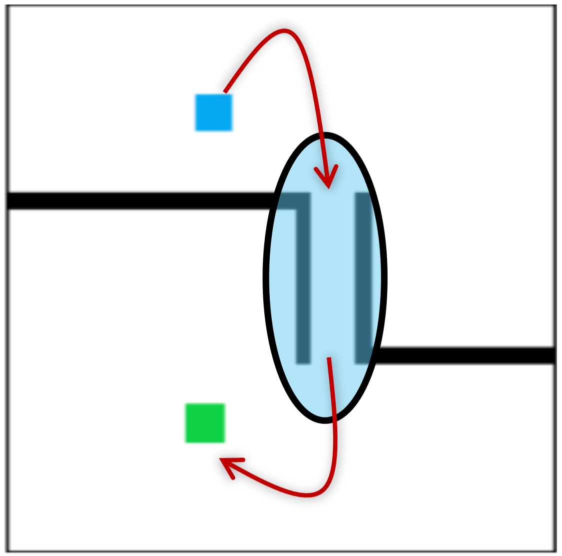
Critical Regions

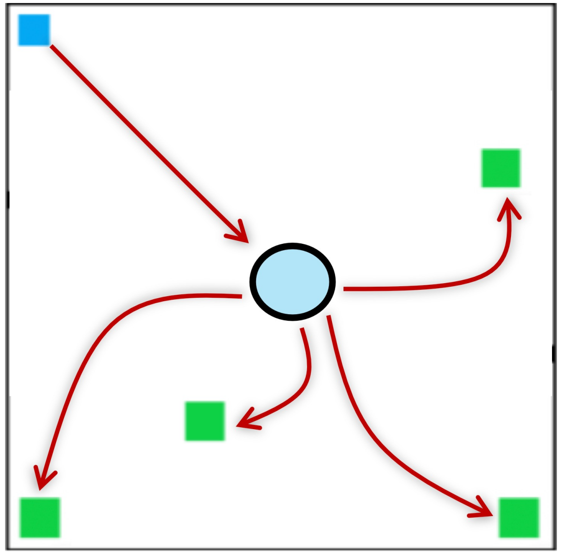
Given a class of motion planning problem M, criticality of an open set r in the C-space:

$$\mu(r) = \lim_{s_n \to^+ v} \frac{f(r)}{v(s_n)}$$

$f(x)$ = fraction of solution plans that pass through x // captures hubs
$v(x)$ = measure of x under uniform sampling density //captures bottlenecks.

[Molina et al., 2020, ICRA]

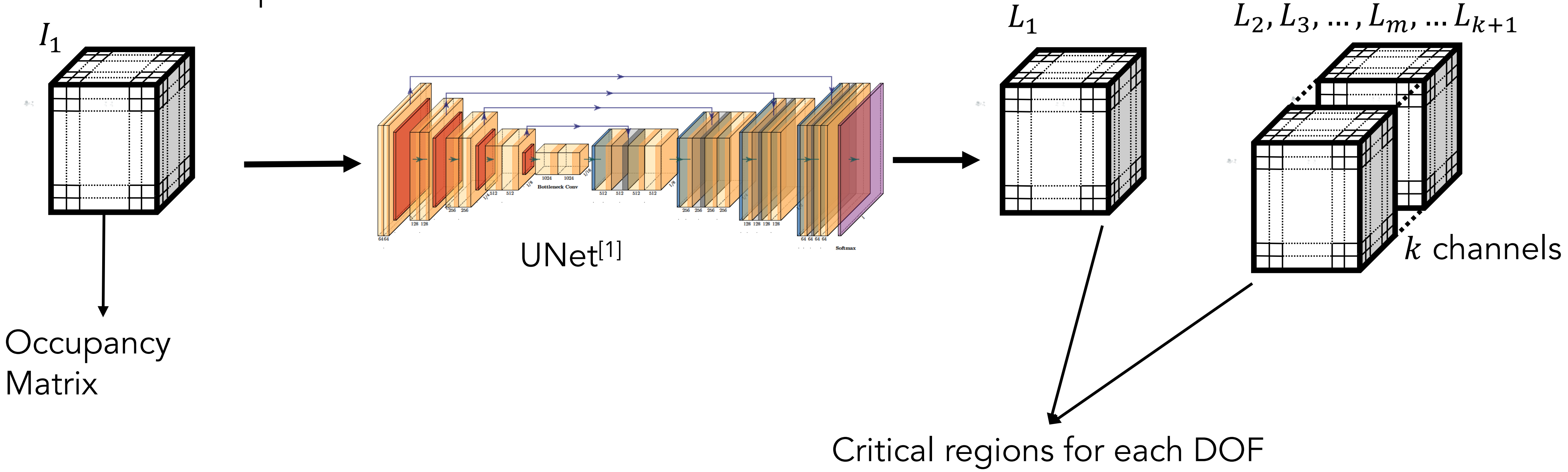




HARP: Learning to Zero-Shot Predict CRs

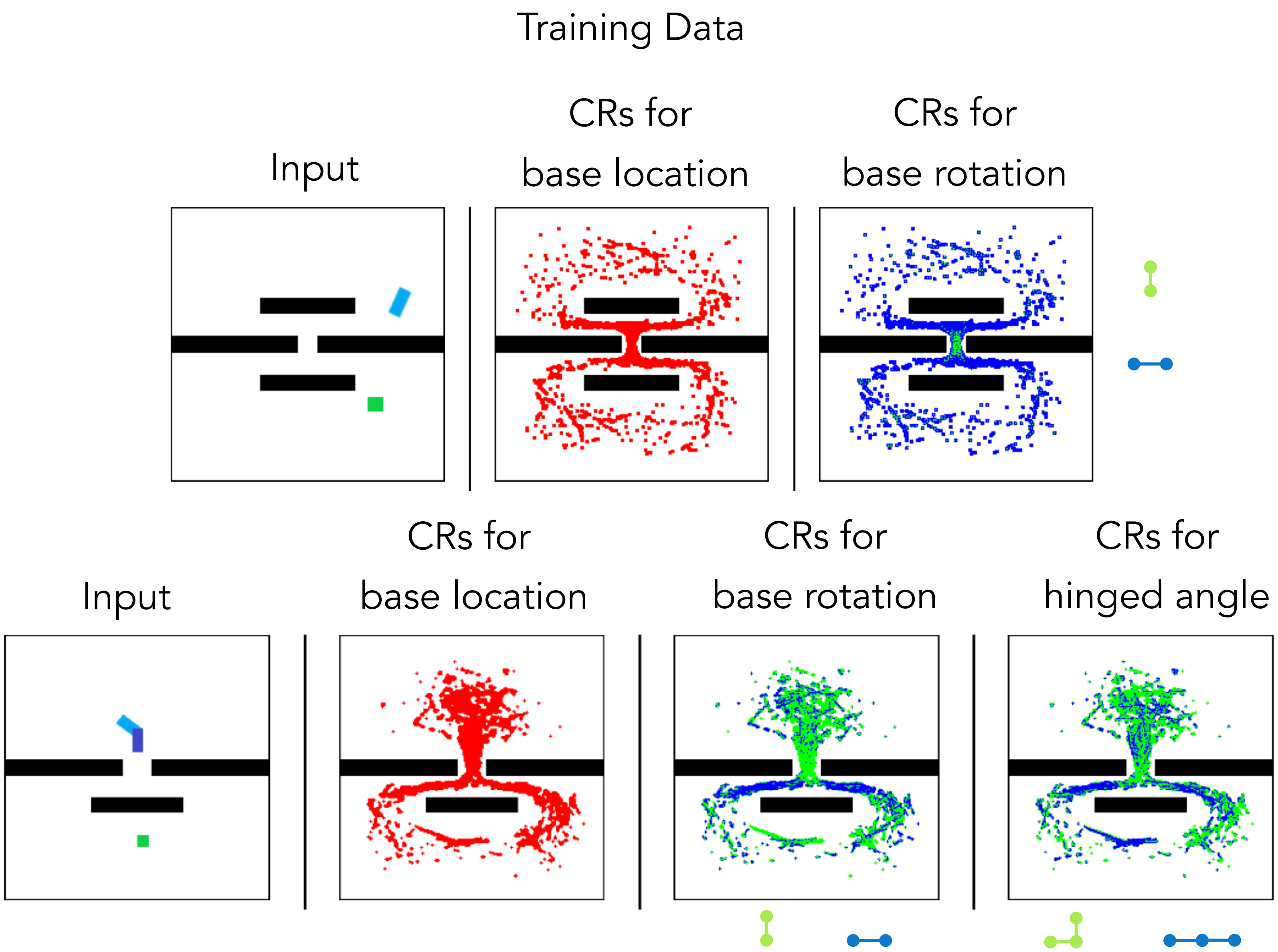
n = #degrees of freedom (DOFs)
k = #DOFs that are not determined by the end-effector's location in the workspace

end-effector = base link for navigation
end-effector = gripper link for manipulation



[1] O. Ronneberger, P.Fischer, and T. Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Proc. MICCAI, 2015

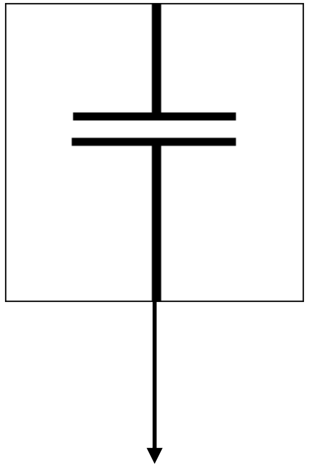
HARP: Training Data



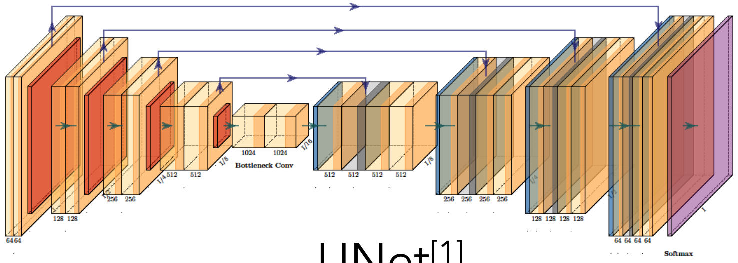
HARP: Learning to Zero-Shot Predict CRs

n = #degrees of freedom (DOFs)
k = #DOFs that are not determined by the end-effector's location in the workspace

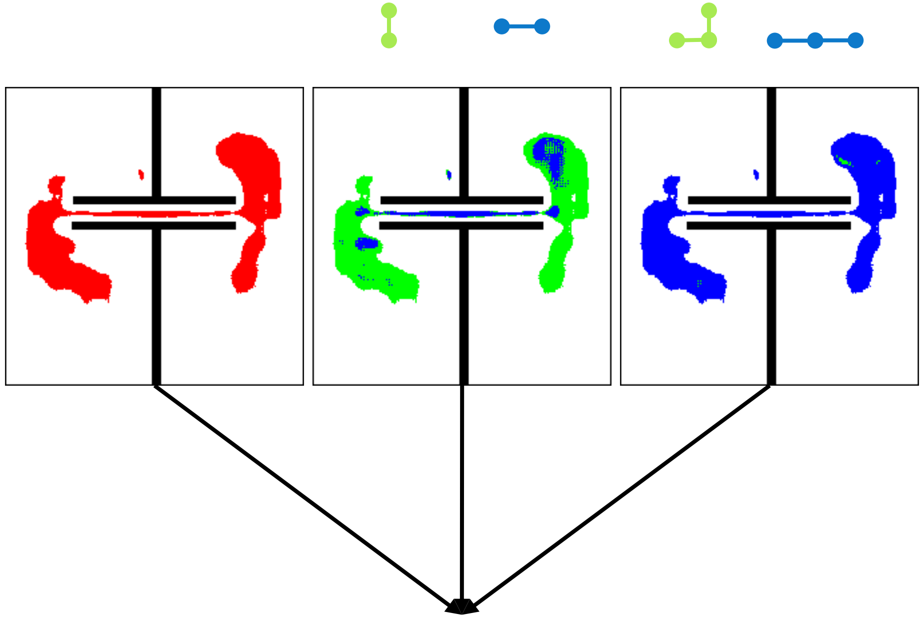
end-effector = base link for navigation
end-effector = gripper link for manipulation



Occupancy Matrix



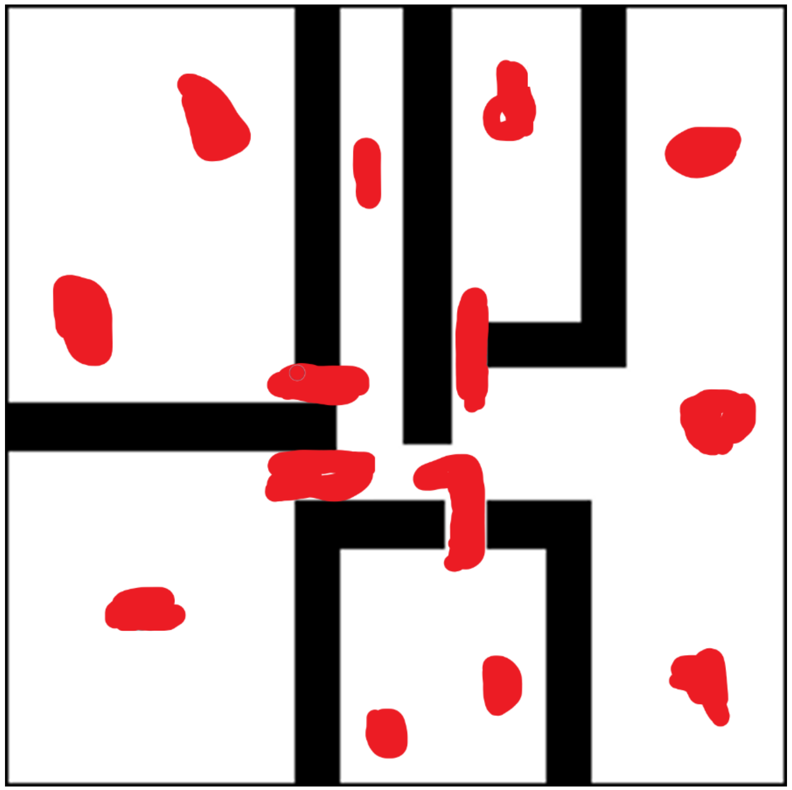
UNet[1]



Critical regions for each DOF

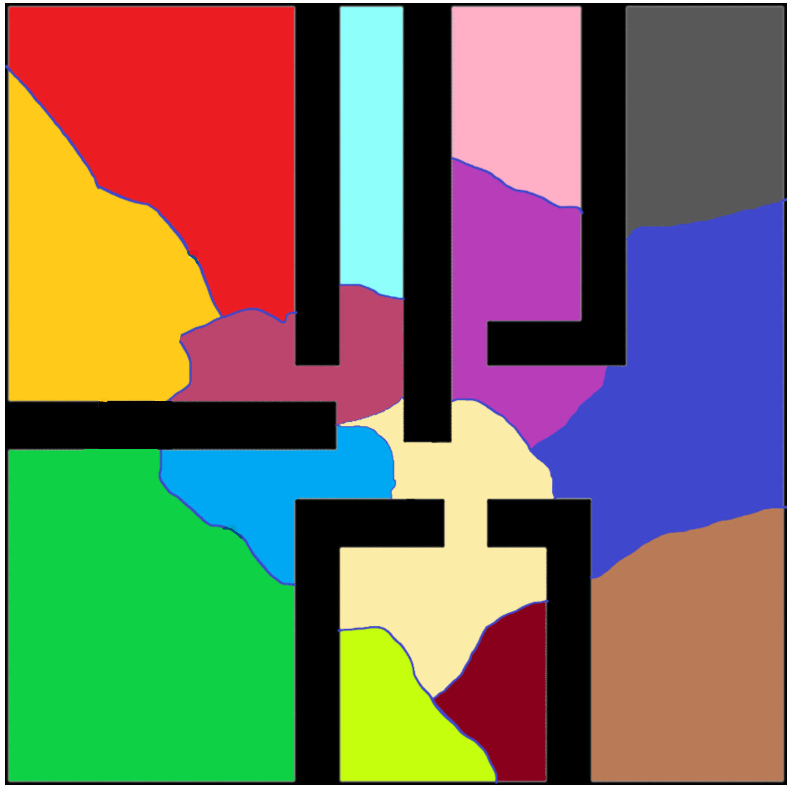
[1] O. Ronneberger, P.Fischer, and T. Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Proc. MICCAI, 2015

HARP: Constructing Zero-Shot Abstractions



- Given a robot and new environment,
  Predict CRs

HARP: Constructing Zero-Shot Abstractions

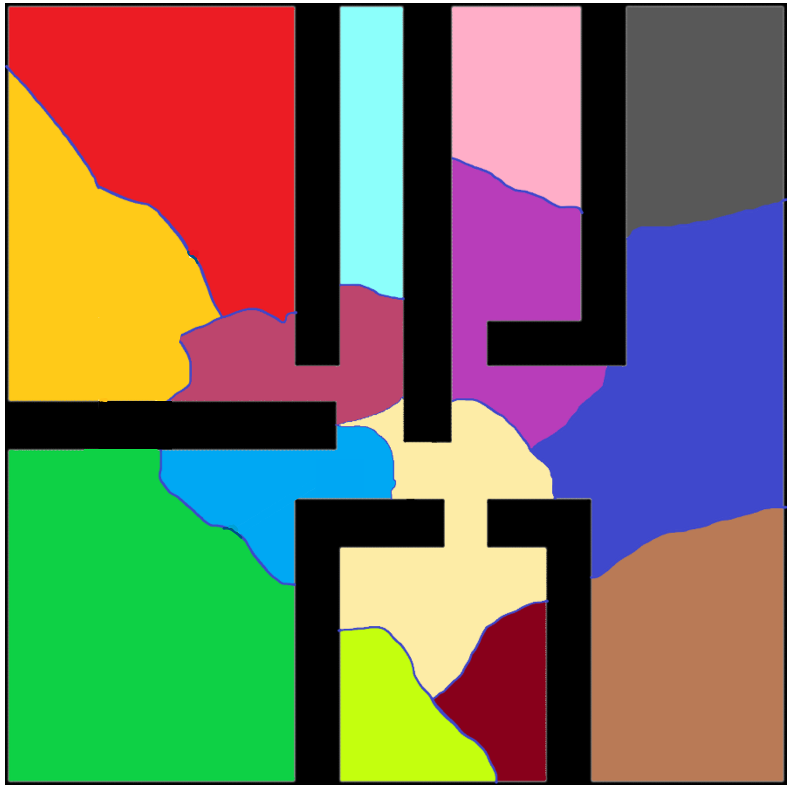


- Given a robot and new environment,

  Predict CRs

  Construct Voronoi diagrams around CRs

HARP: Constructing Zero-Shot Abstractions



2D projection of RBVDs

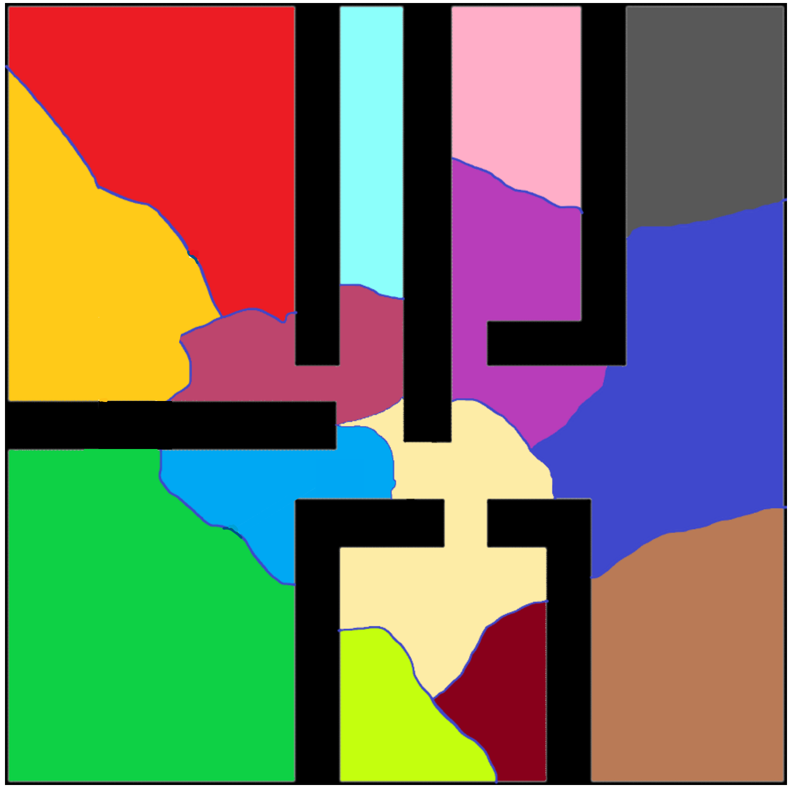
- Given a robot and new environment,

  Predict CRs

  Construct Voronoi diagrams around CRs

  Abstract states = Voronoi cells

HARP: Constructing Zero-Shot Abstractions



2D projection of RBVDs

- Given a robot and new environment,

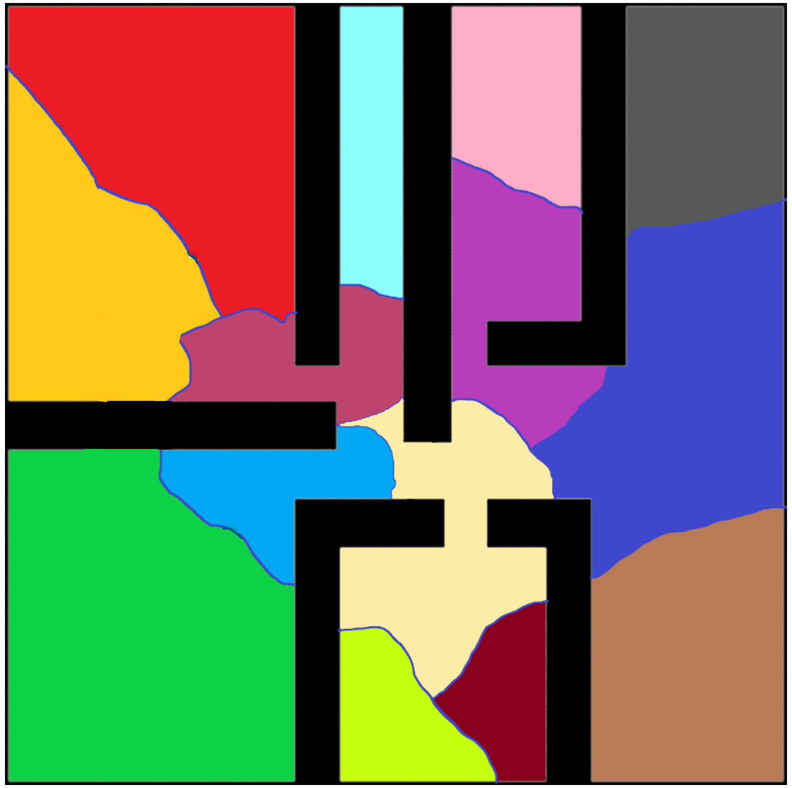
  Predict CRs

  Construct Voronoi diagrams around CRs

  Abstract states = Voronoi cells

  Abstract actions = transitions between abstract states

HARP: Hierarchical Motion Planning



2D projection of RBVDs

- Given a robot and new environment,

  Predict CRs

  Construct Voronoi diagrams around CRs

  Abstract states = Voronoi cells

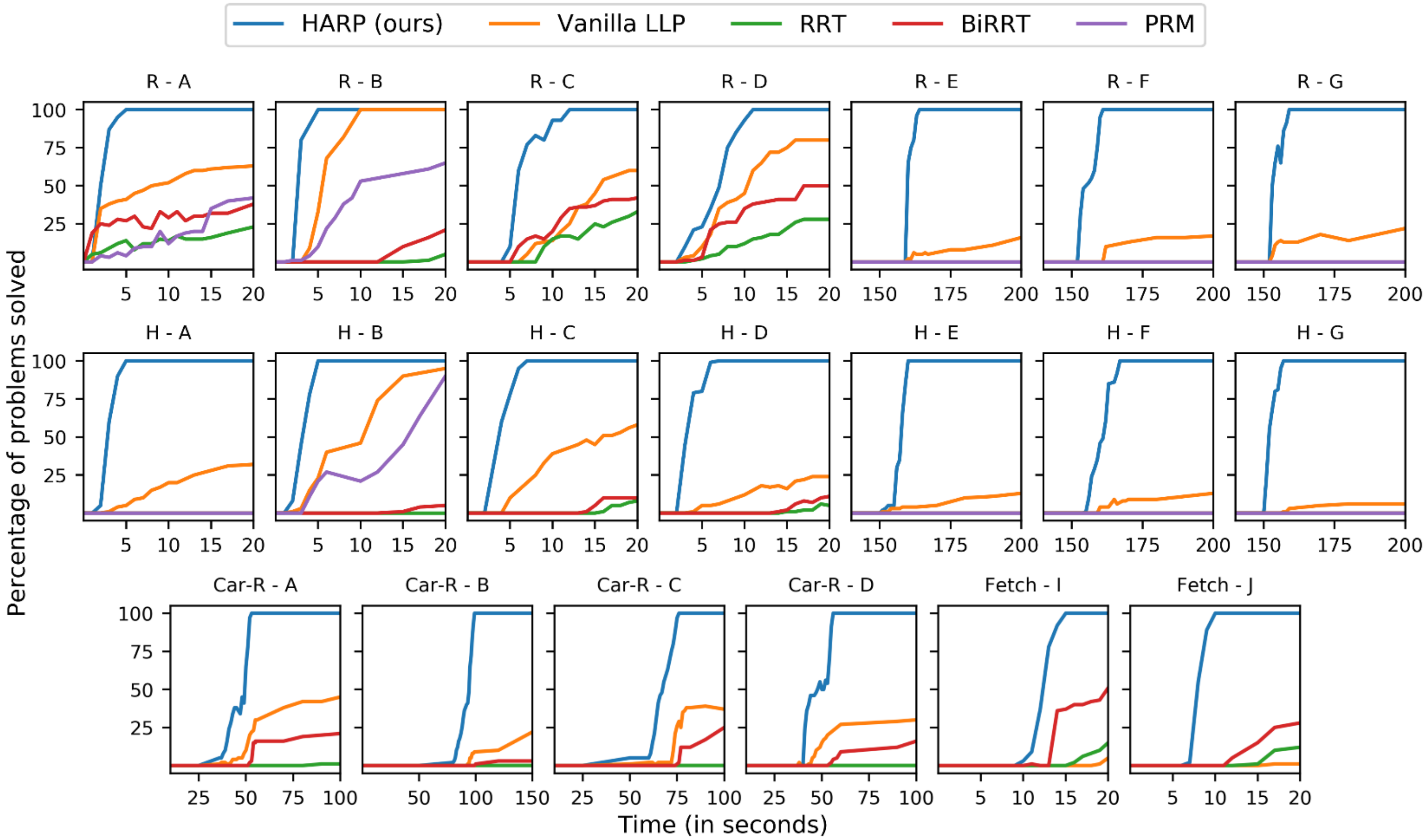
  Abstract actions = transitions between abstract states

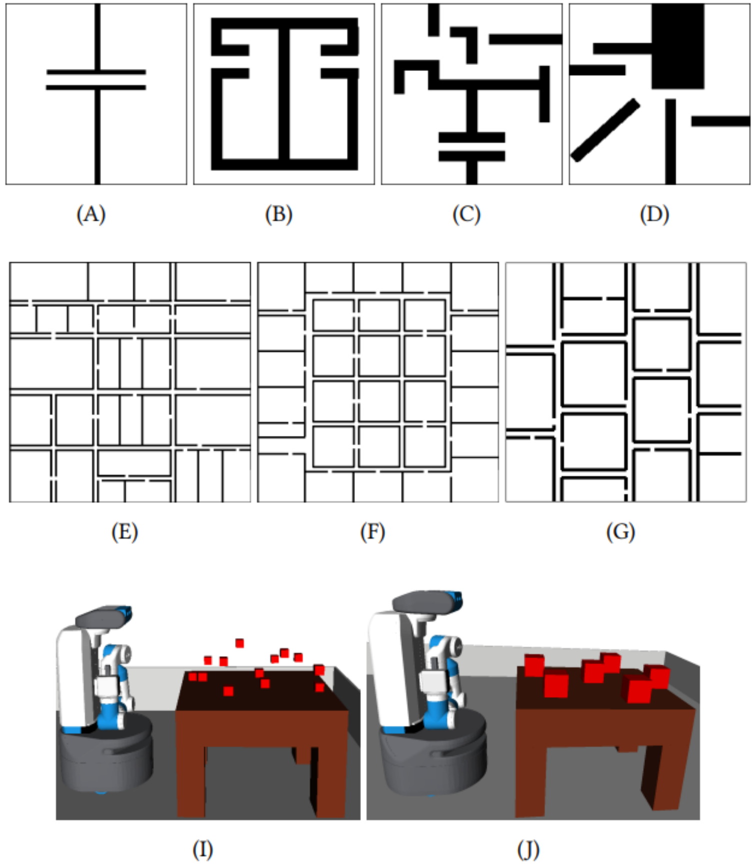
  Hierarchical motion planning using:

  a high-level multi-source bi-directional beam search

  a multi-source multi-directional LLP mp

HARP: Experiments





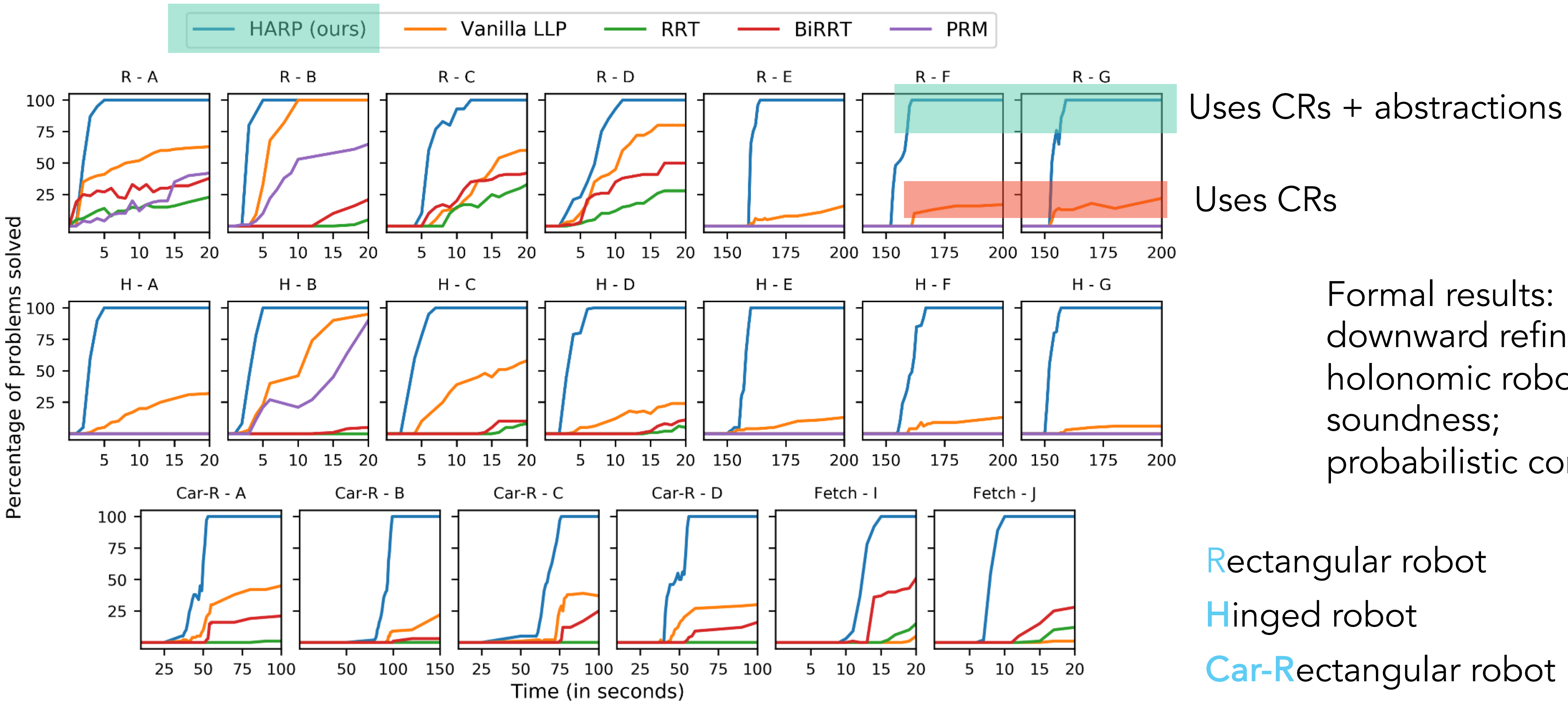
Rectangular robot

Hinged robot

Car-Rectangular robot

Fetch 8-DOF manipulator

HARP: Experiments



Formal results:
downward refinability for holonomic robots;
soundness;
probabilistic completeness

Rectangular robot

Hinged robot

Car-Rectangular robot

Fetch 8-DOF manipulator

SHARP: What if Robot Dynamics are Stochastic?

Objective: ~~Compute a motion plan~~

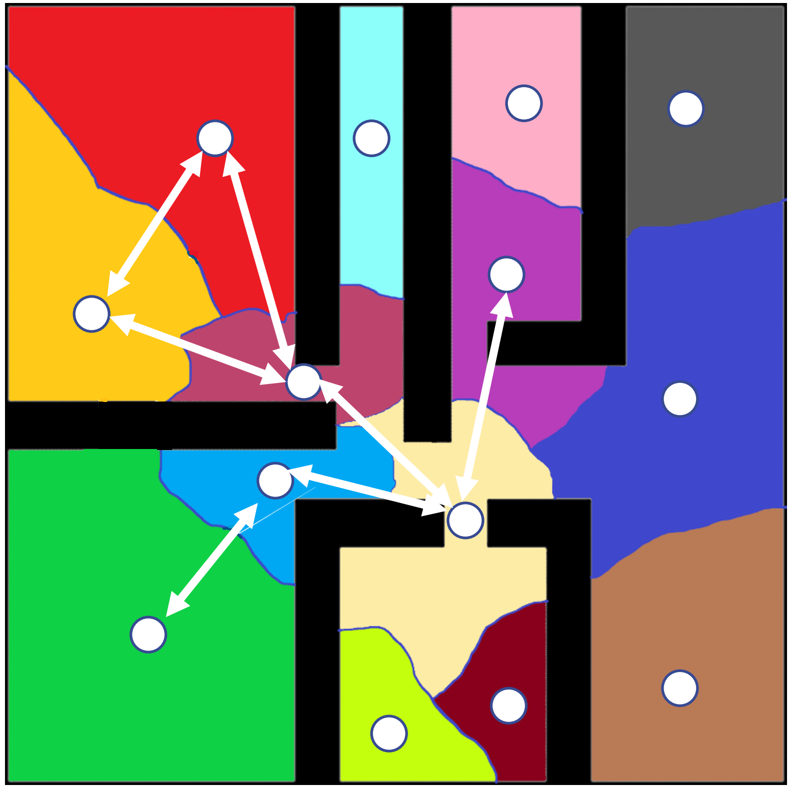
Compute a motion policy

SHARP: What if Robot Dynamics are Stochastic?

Objective: ~~Compute a motion plan~~

Compute a motion policy

High-level actions = Options



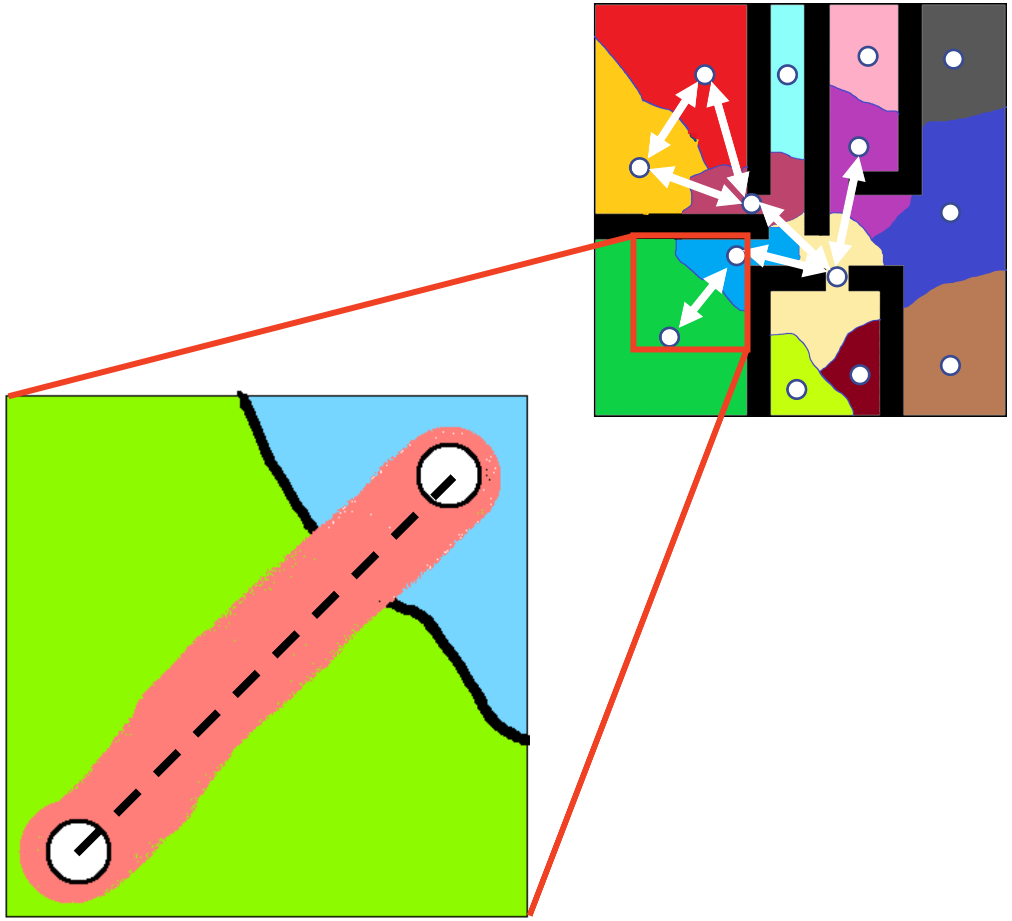
SHARP: What if Robot Dynamics are Stochastic?

Objective: ~~Compute a motion plan~~

Compute a motion policy

High-level actions = Options

Option guide = autogenerated dense pseudo-reward



SHARP: What if Robot Dynamics are Stochastic?

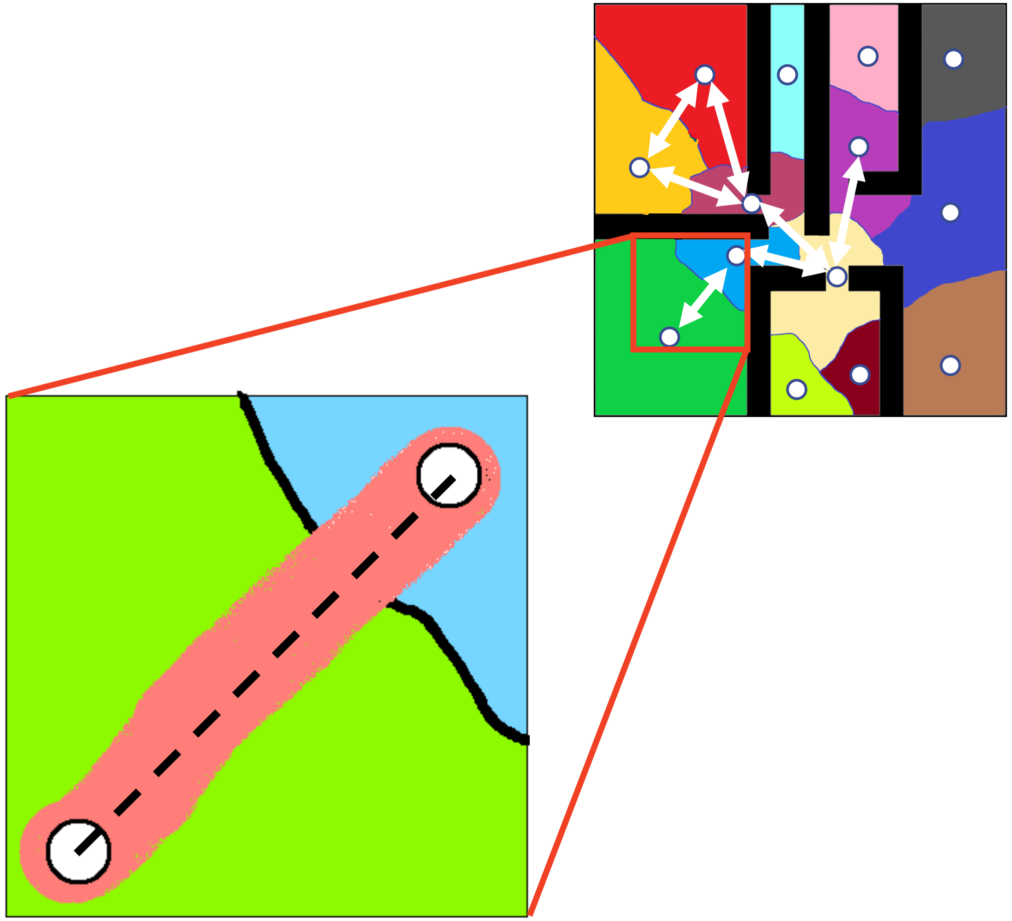
Objective: ~~Compute a motion plan~~

Compute a motion policy

High-level actions = Options

Option guide = autogenerated dense pseudo-reward

Autogenerated reward shaping



SHARP: What if Robot Dynamics are Stochastic?

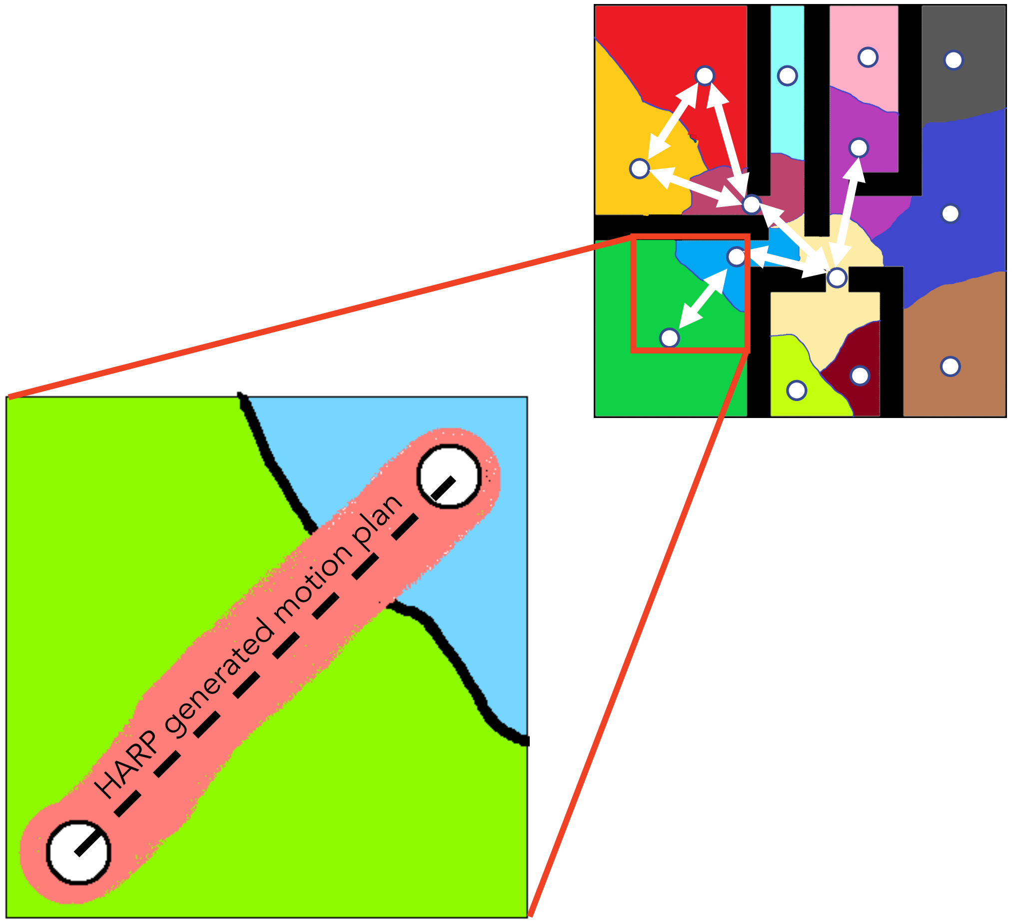
Objective: ~~Compute a motion plan~~

Compute a motion policy

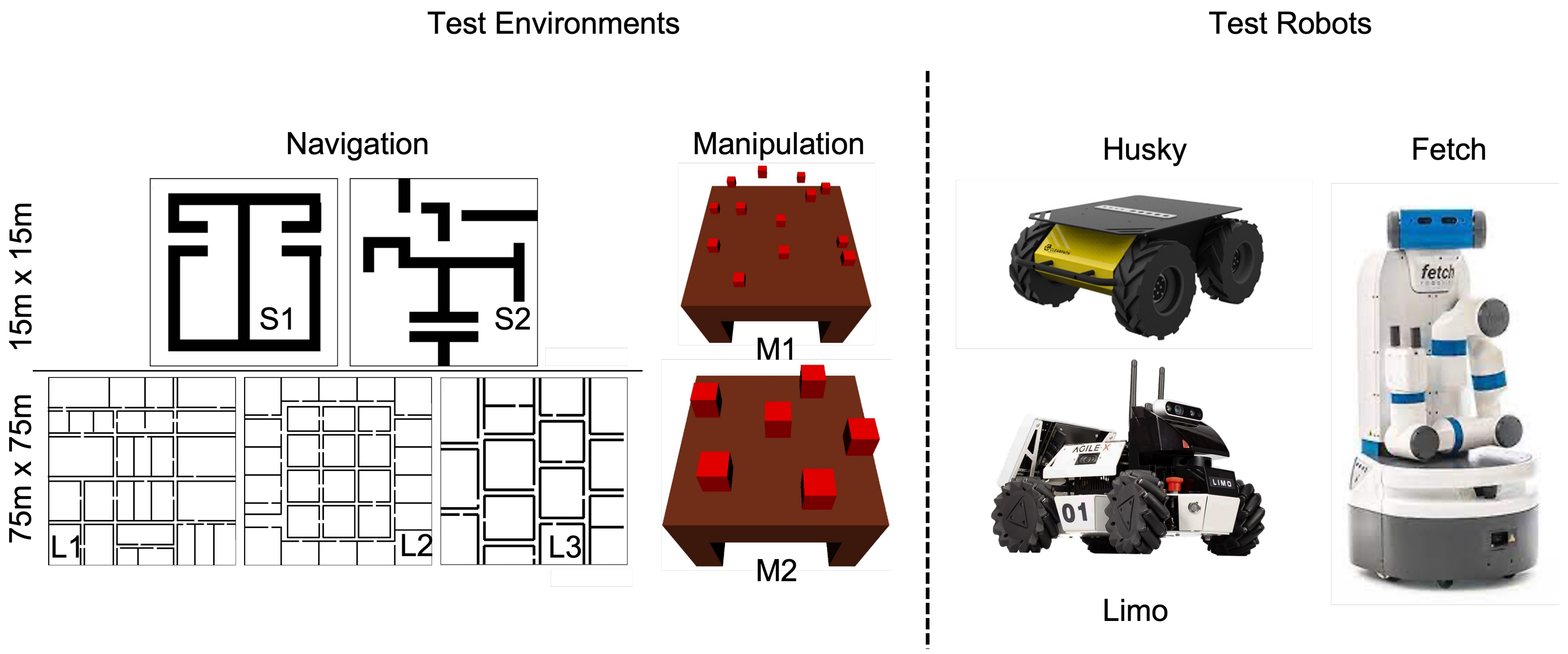
High-level actions = Options

Option guide = autogenerated dense pseudo-reward

Autogenerated reward shaping

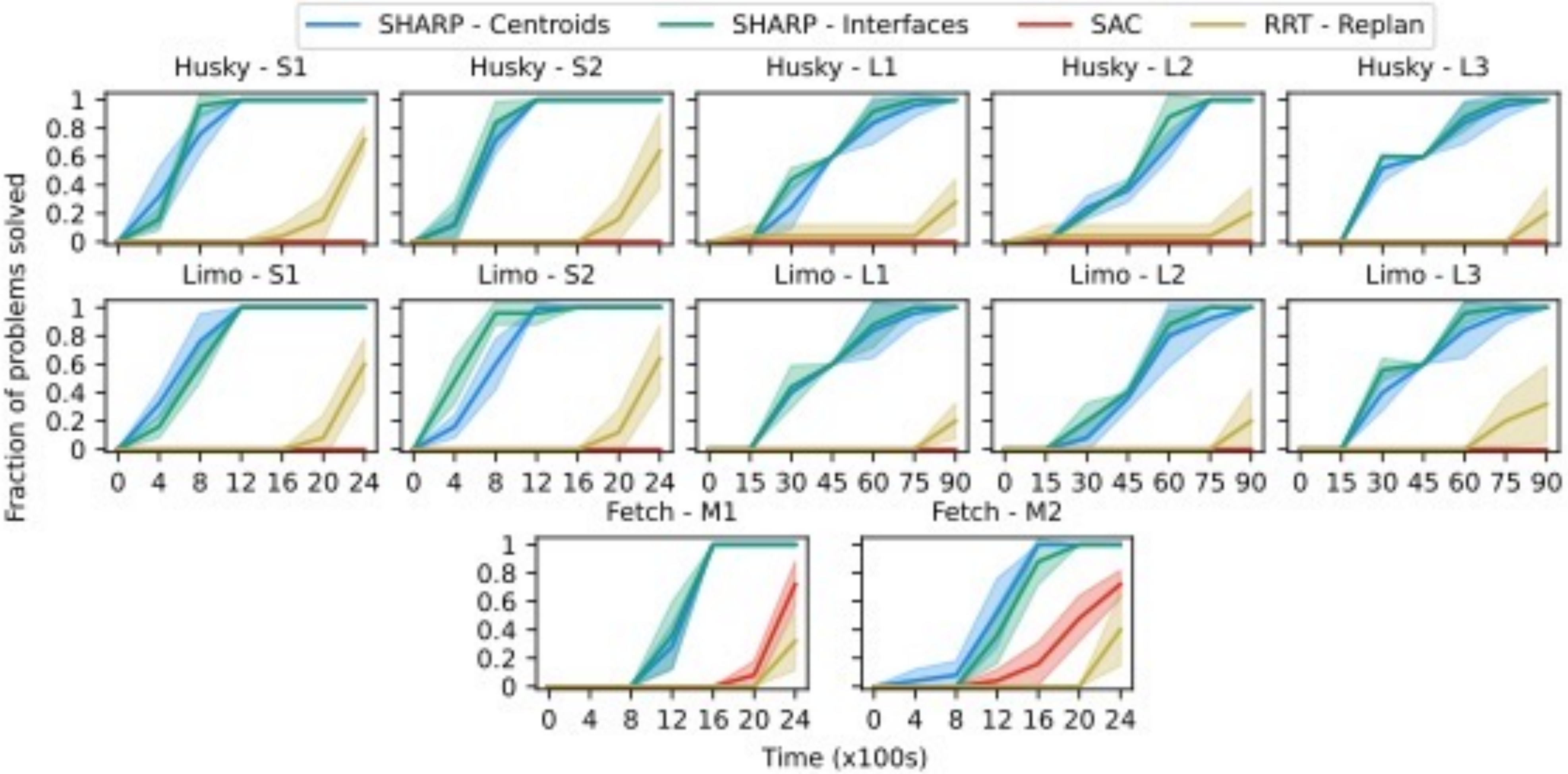


SHARP: Experiments

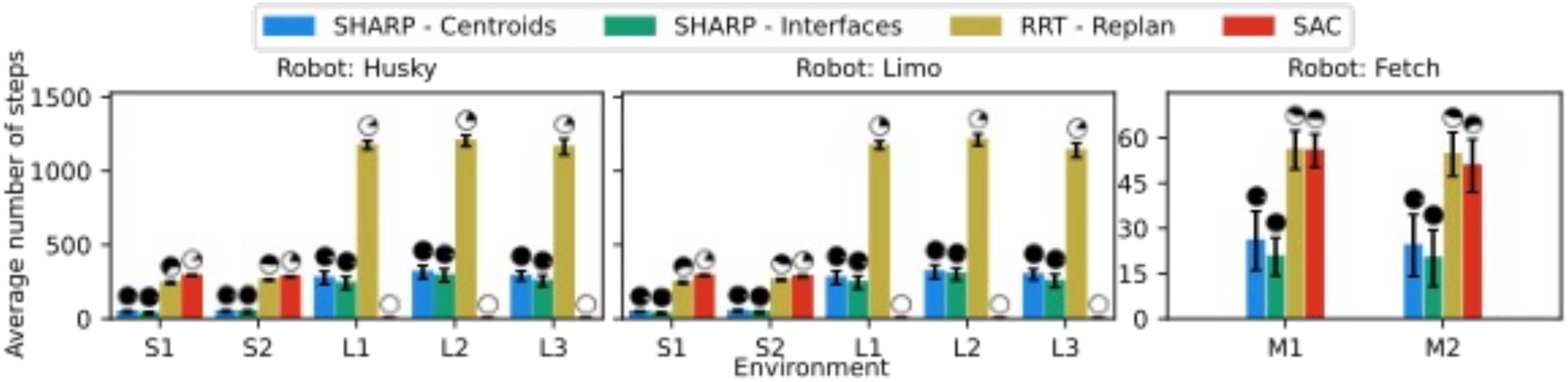


SHARP: Results

SHARP performs well (times include creation of state and action abstractions).
Next-best: RRT-replan! Other baselines struggle to learn
Hypothesis: not suited for stochasticity, long horizons, sparse rewards



SHARP: Results



Bars indicate solution length in number of steps (lower is better)
Pies indicate % success (darker is better)

High-Level Summary

	Abstract States	Abstract Actions	Low-level behaviors	Samplers	C-Space changes
Skills to Symbols	Learned	Input	Input	Input	Yes
[illegible], 2021	Input	Learned	Input	Input	Yes
[illegible], 2022	Input	Learned	Learned	Learned	Yes
HA[illegible]/ [illegible]HA[illegible]	Learned	Learned	Learned / Computed	Learned	**No**

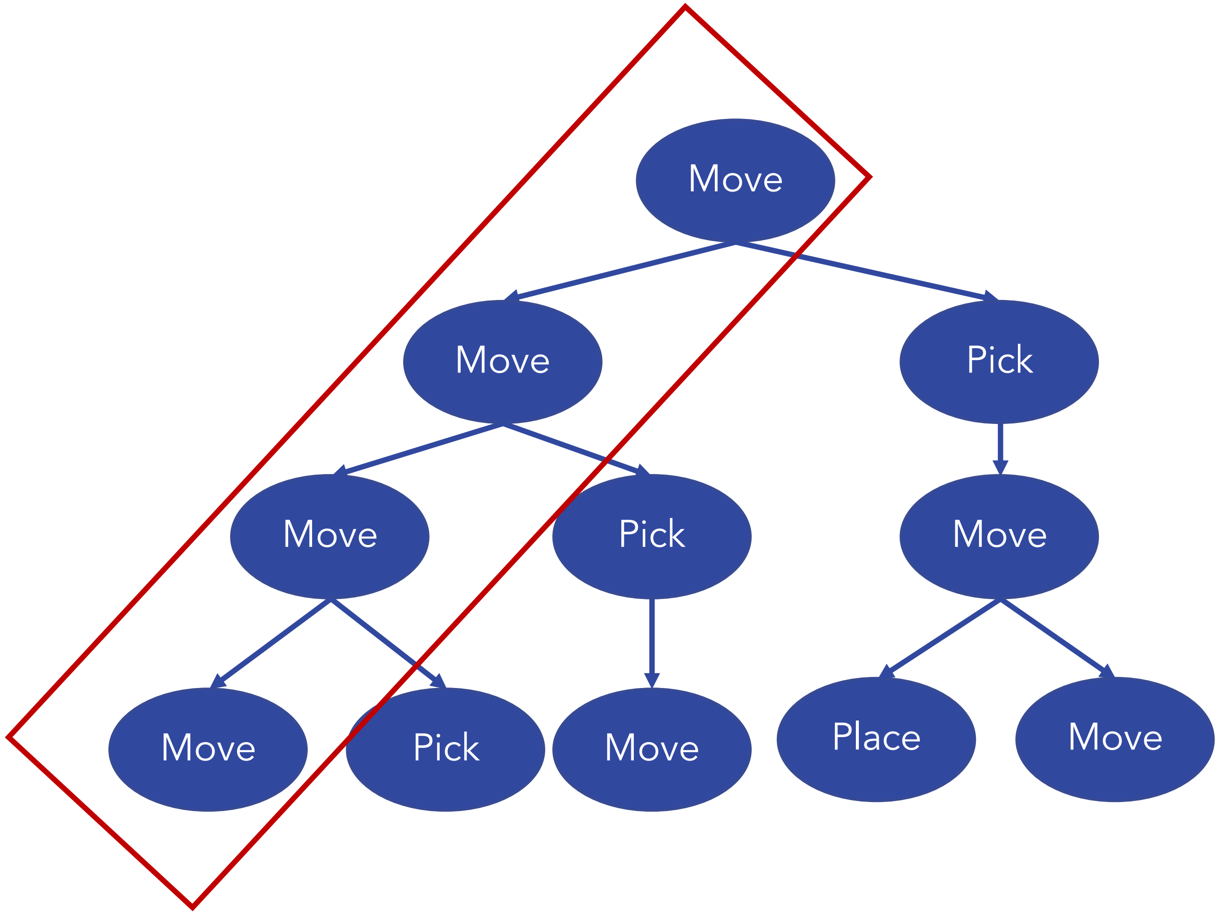
Outline

1. Background: Why Task and Motion Planning?
2. Abstraction as a Foundation for TMP
3. Abstraction-based Approaches
4. Research Frontier: Neuro-Symbolic Learning for TMP

What next?

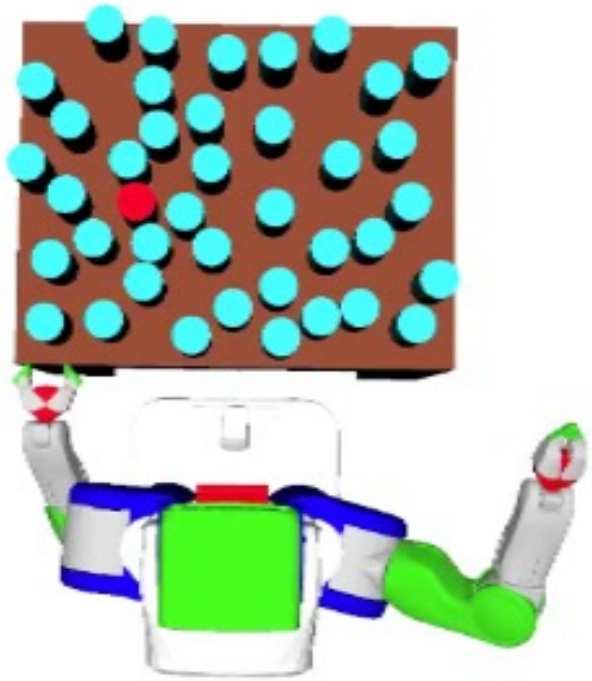
Open Questions

- Stochasticity at the low-level



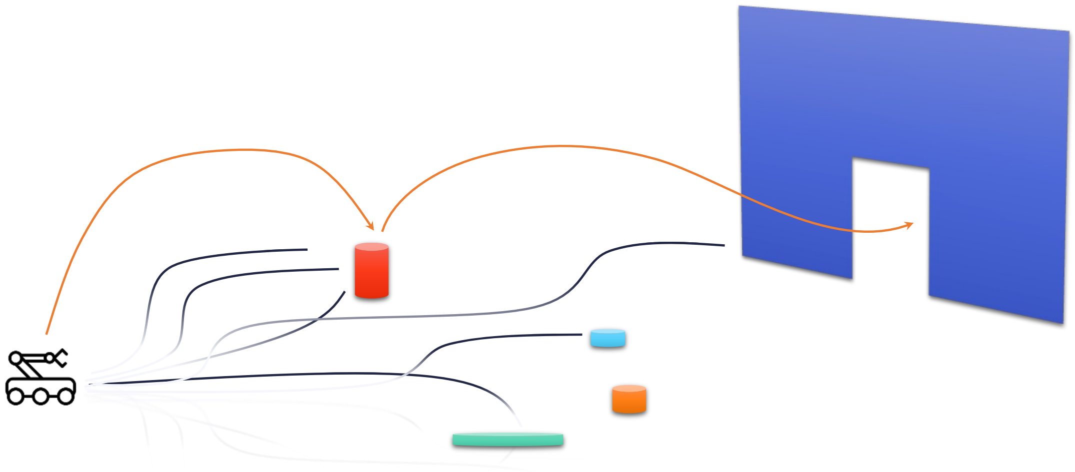
Open Questions

- Stochasticity at the low-level
- Partial observability



Open Questions

- Stochasticity at the low-level
- Partial observability
- Learning abstractions for TMP -- What if **neither state nor actions** are not provided?



GoTo(l)
Pickup(x)
PutDown(x)

Temporal Abstraction
≡ Abstract Actions,
Macros,
Options…

At(x, l)
InGripper(x)
AtDestination(x)

State
Abstraction

Open Questions

- Stochasticity at the low-level
- Partial observability
- Learning abstractions for TMP -- What if state **and** actions are not provided?
- Orthogonal direction: Pose estimation (understanding low-level observational data)