Planning with Hierarchy and Abstraction

Tom Silver
Robot Planning Meets Machine Learning
Princeton University
Fall 2025

Let’s Play a Review Game

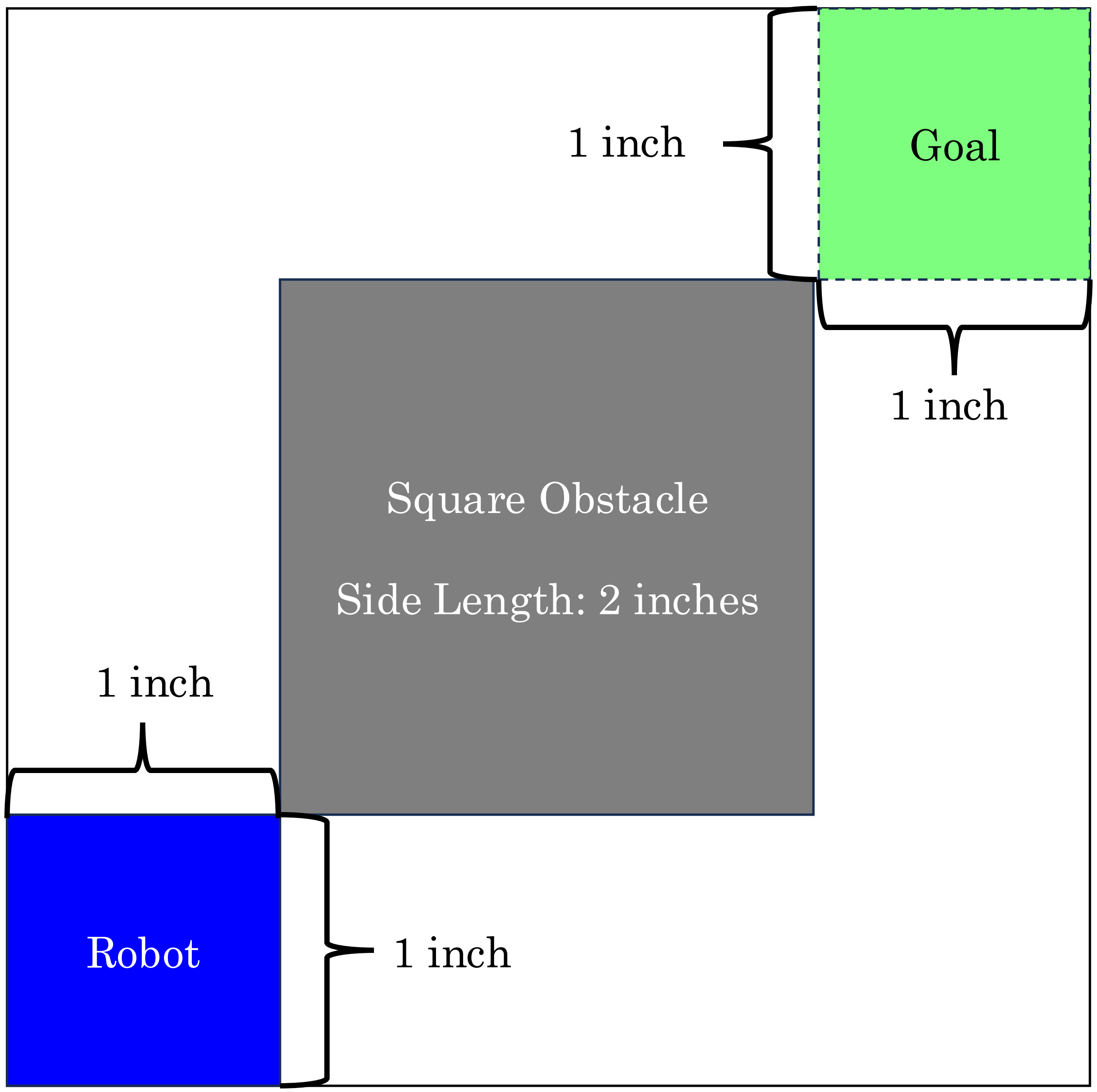
Bar trivia rules

- Break up into teams of 3-5
- Give your team a great name
- I will ask questions
- You will discuss quietly with your team
- Write down your answer
- Hold it up when I say so

Question 1

What is the expected number of nodes that RRT will create in the example on the right?

1. Less than 5
2. Between 5 and 25
3. Between 25 and 100
4. Greater than 100 / no limit



Question 2

Consider a POMDP with 2 states, 2 observations, and 2 actions.

Suppose that our initial belief state is uniform.

Is it possible that the corresponding Belief MDP has an infinite number of reachable states?

Question 3

True or False: if a POMDP has a deterministic transition distribution and a deterministic observation distribution, then there exists some policy for the agent that would lead to an absolutely certain belief state (some state has 100%).

Question 4

True or False: for any classical planning problem, if a solution exists, then a solution also exists in the delete relaxed problem.

Question 5

True or false: in classical planning, given an *optimal* heuristic, the number of nodes *expanded* by A* is equal to the number of actions in the output plan.

Question 6

Which of the following is true about MCTS, but not about RTDP?

1. Requires only simulator access to MDP
2. Focuses on “promising” parts of AODAG
3. Adds one new state node at each iteration
4. Backpropagates values after each iteration
5. Uses rollout heuristic to estimate leaf node values
6. Uses greedy policy to select nodes to expand

You may select multiple.

Question 7

Which of the following bandit exploration strategies are guaranteed to try all arms infinitely often in the limit?

1. Uniform random
2. Exploit only
3. Epsilon-greedy (for nontrivial epsilon)
4. UCB

You may select multiple.

Question 8

Is there any bug in this code, and if so, which line?

```
 1 def value_iteration(
 2     states: List[State],
 3     actions: List[Action],
 4     transitions: Dict[Tuple[State, Action], List[Transition]],
 5     gamma: float = 0.95,
 6     theta: float = 1e-6,
 7 ) -> Dict[State, float]:
 8     """Returns state values."""
 9     V = {s: 0.0 for s in states}
10
11     while True:
12         delta = 0.0
13         for s in states:
14             q_values = []
15             for a in actions:
16                 exp_return = 0.0
17                 for p, s_next, r in transitions[(s, a)]:
18                     exp_return += p * r + gamma * V[s_next]
19                 q_values.append(exp_return)
20
21             v_new = max(q_values) if q_values else V[s]
22             delta = max(delta, abs(v_new - V[s]))
23             V[s] = v_new
24
25         if delta < theta:
26             break
27
28     return V
```

Question 9

What are the three kinds of MDP time horizons?

Question 10 (Tiebreak)

List any algorithms we have covered in this course. The most recalled wins.

Planning with Hierarchy and Abstraction

Tom Silver

Robot Planning Meets Machine Learning

Princeton University

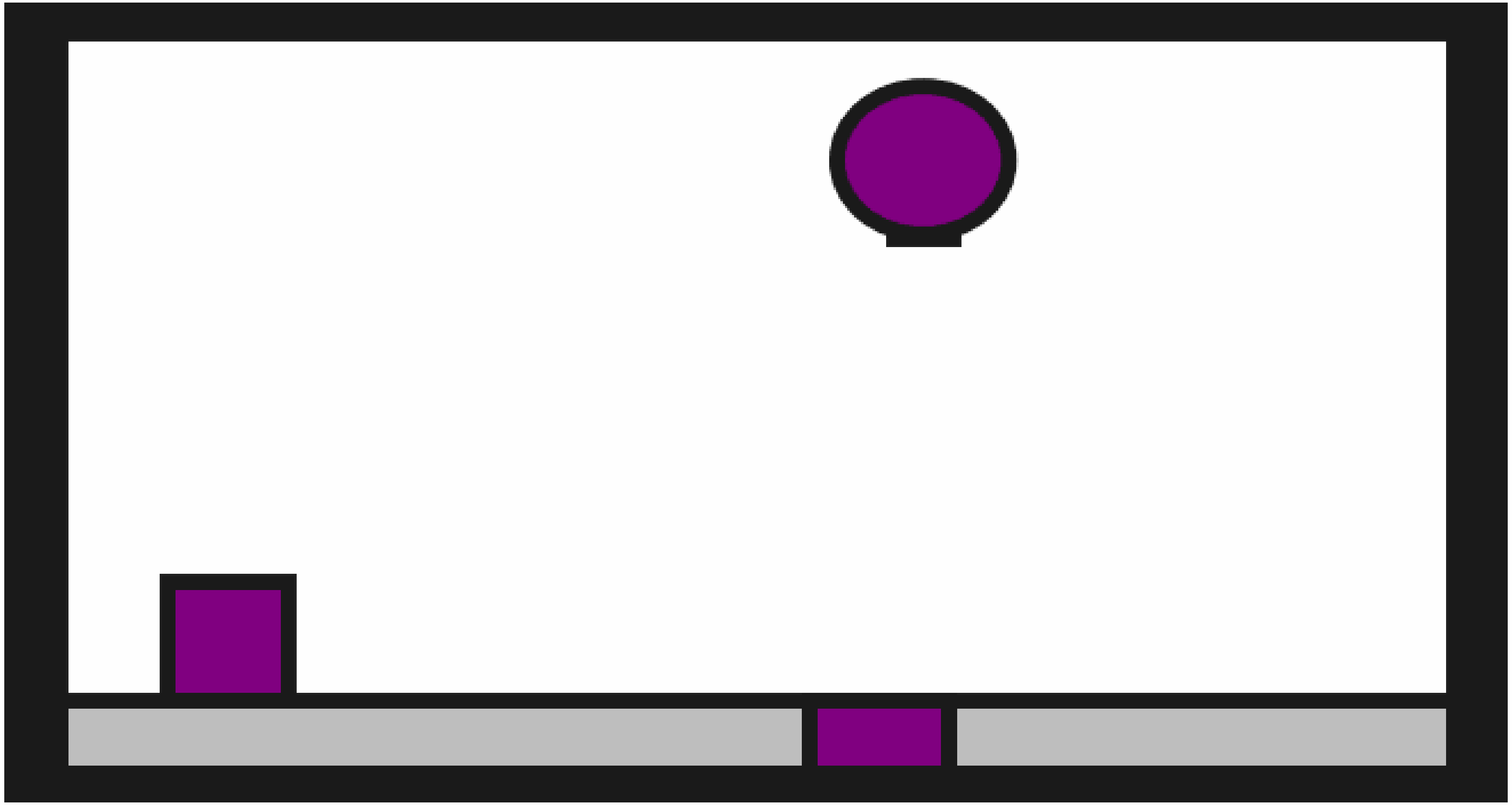
Fall 2025

Recap and Preview

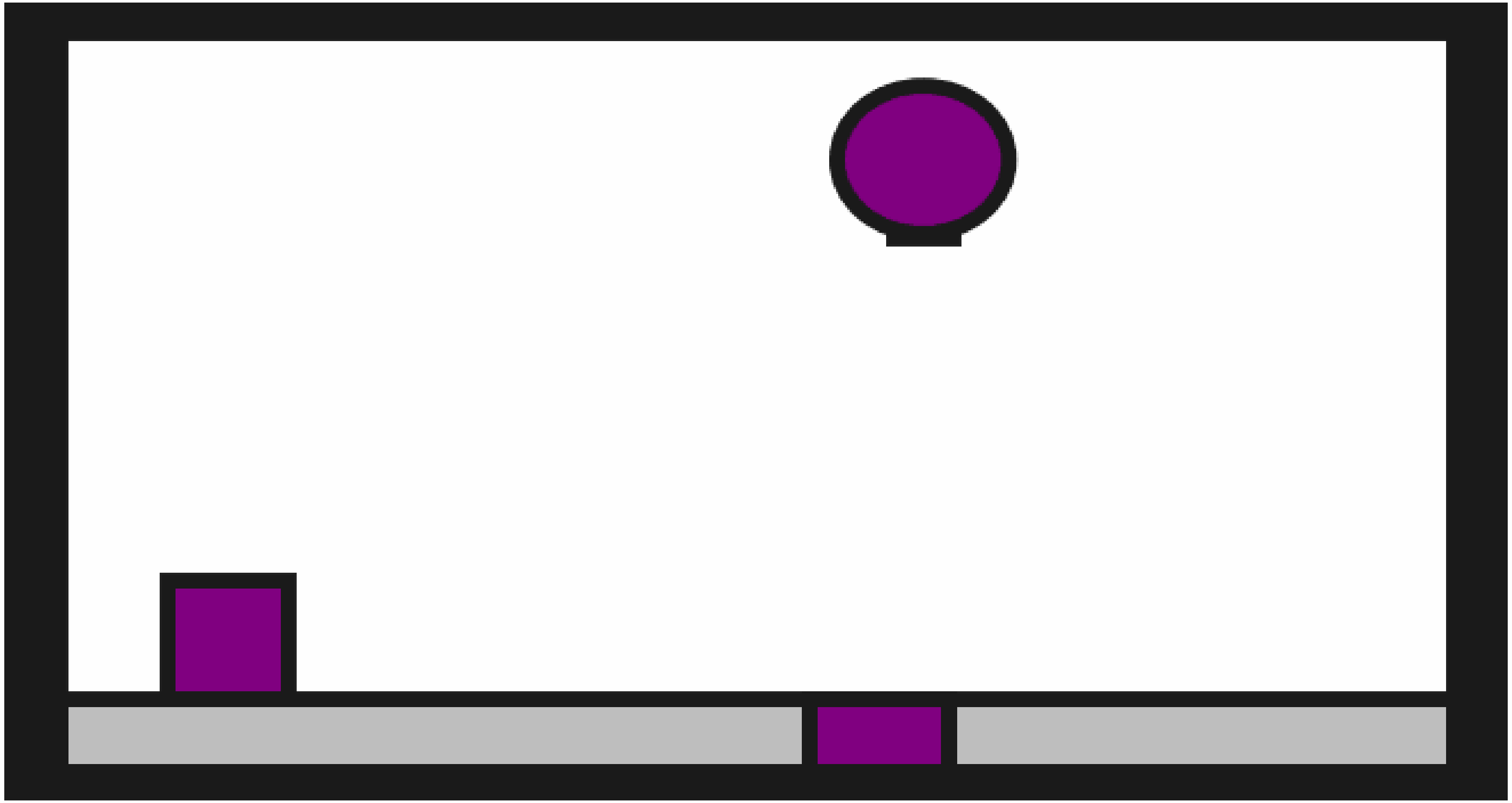
Last time: planning in continuous state and action spaces

This time: same problem setting, new tools: abstractions!

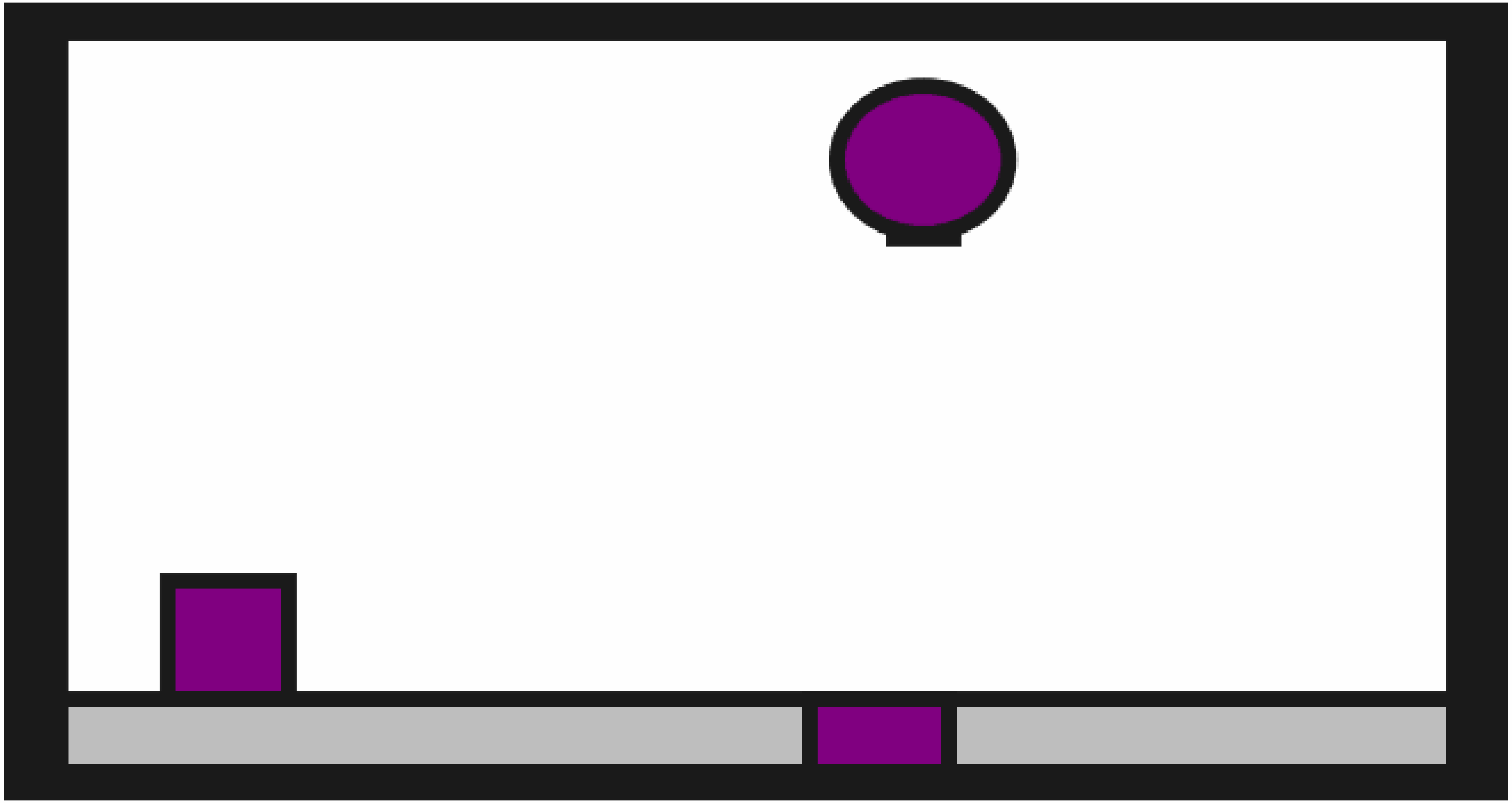
Human Demo



Random Actions



Task Distribution



Example

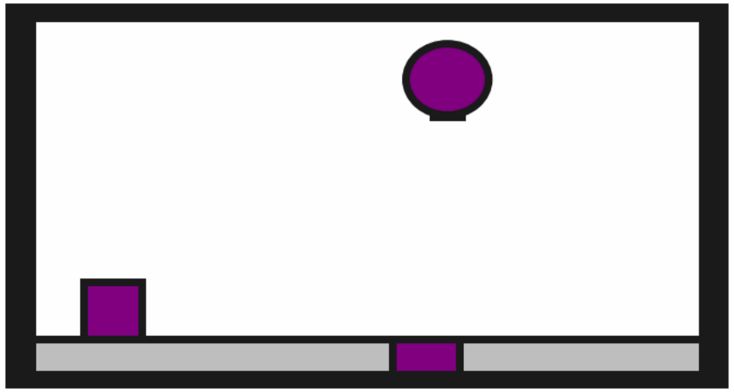
Which planners could we try? Would they work?

State space: Robot config, block pose (8D)

Action space: Pose change, vacuum (5D)

Transition function: Apply action but disallow collisions (no change)

Cost function: -1 until block on target

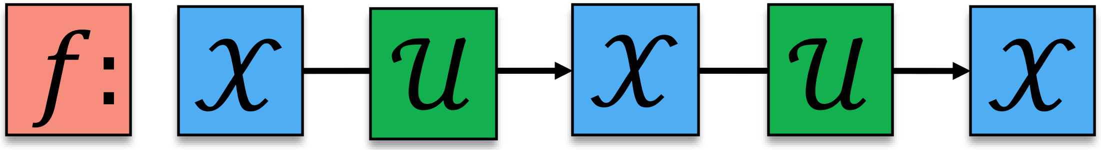


A Two-Level Hierarchy

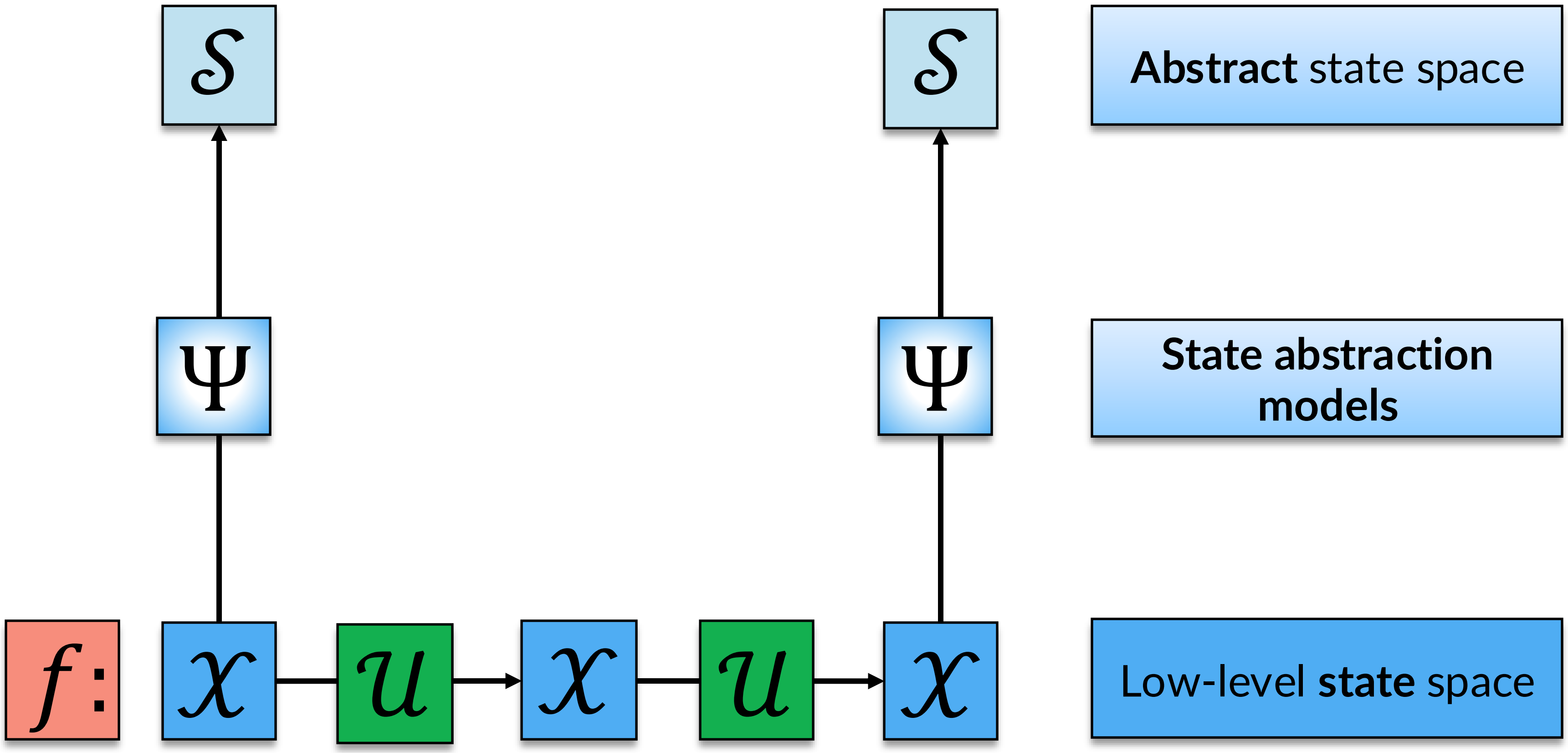
Low-level **transition** model

Low-level **action** space

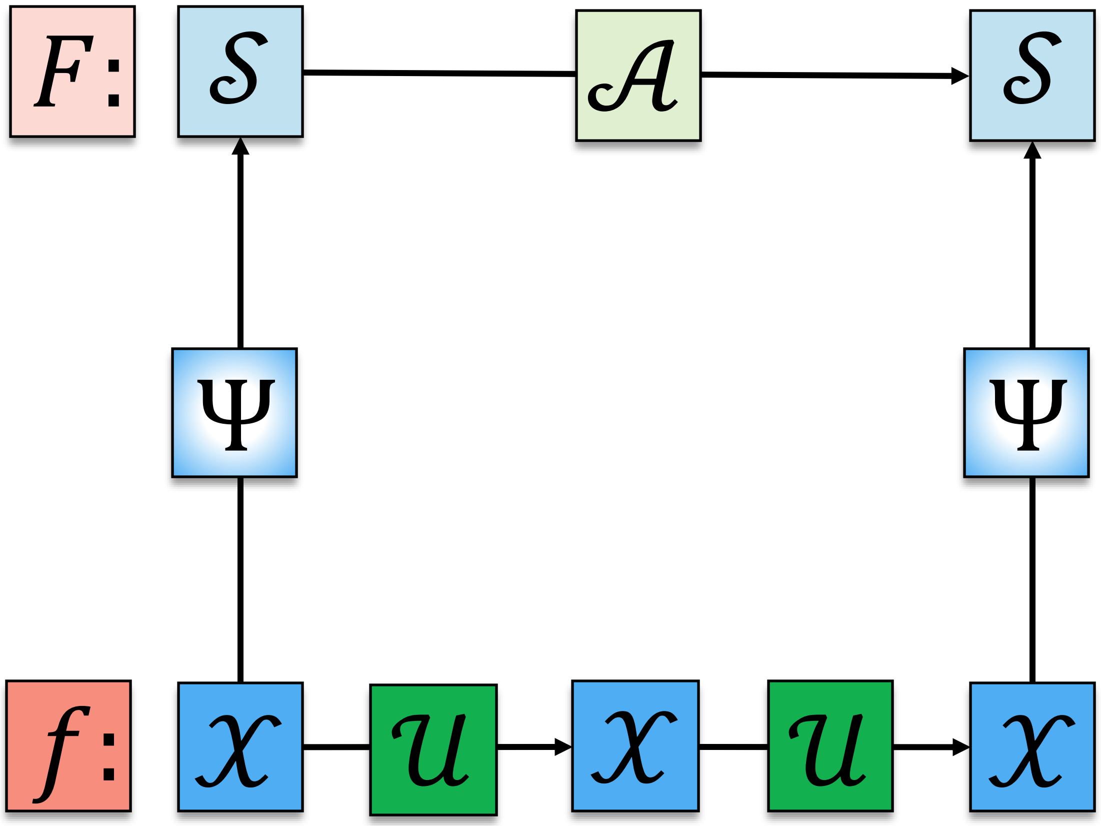
Low-level **state** space



A Two-Level Hierarchy



A Two-Level Hierarchy

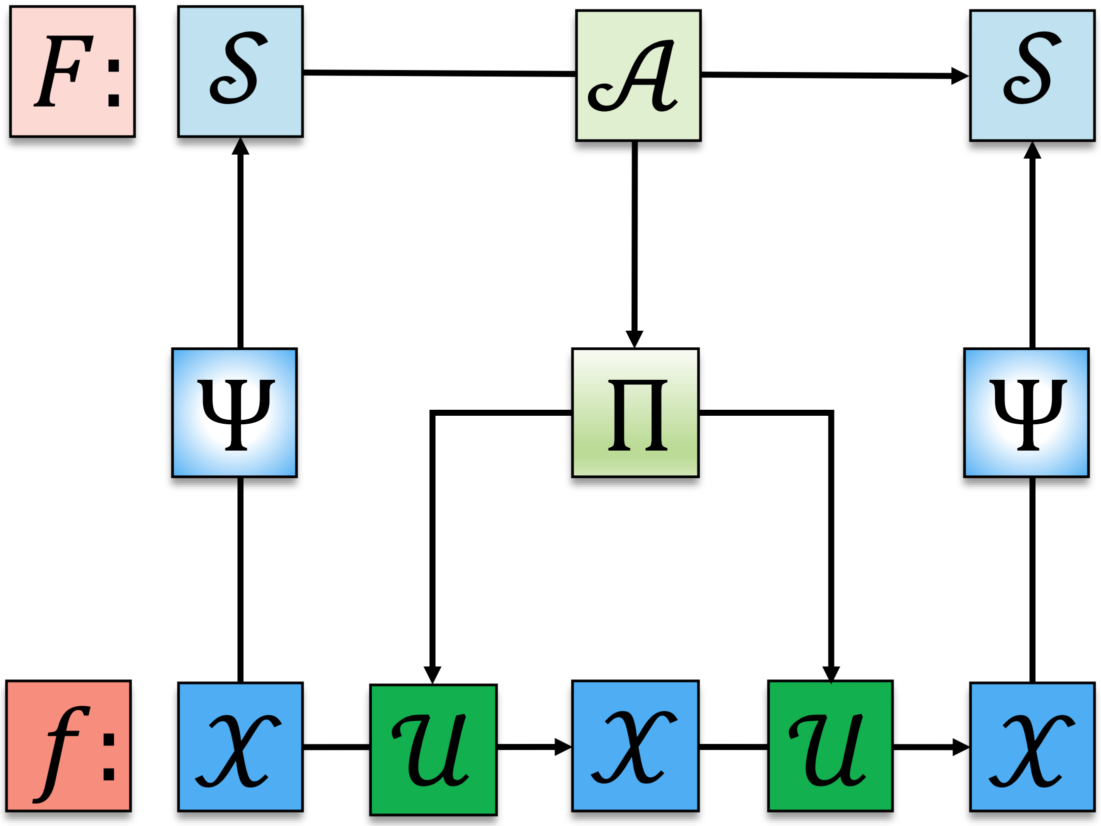


Abstract state space

Abstract action space

Abstract transition model

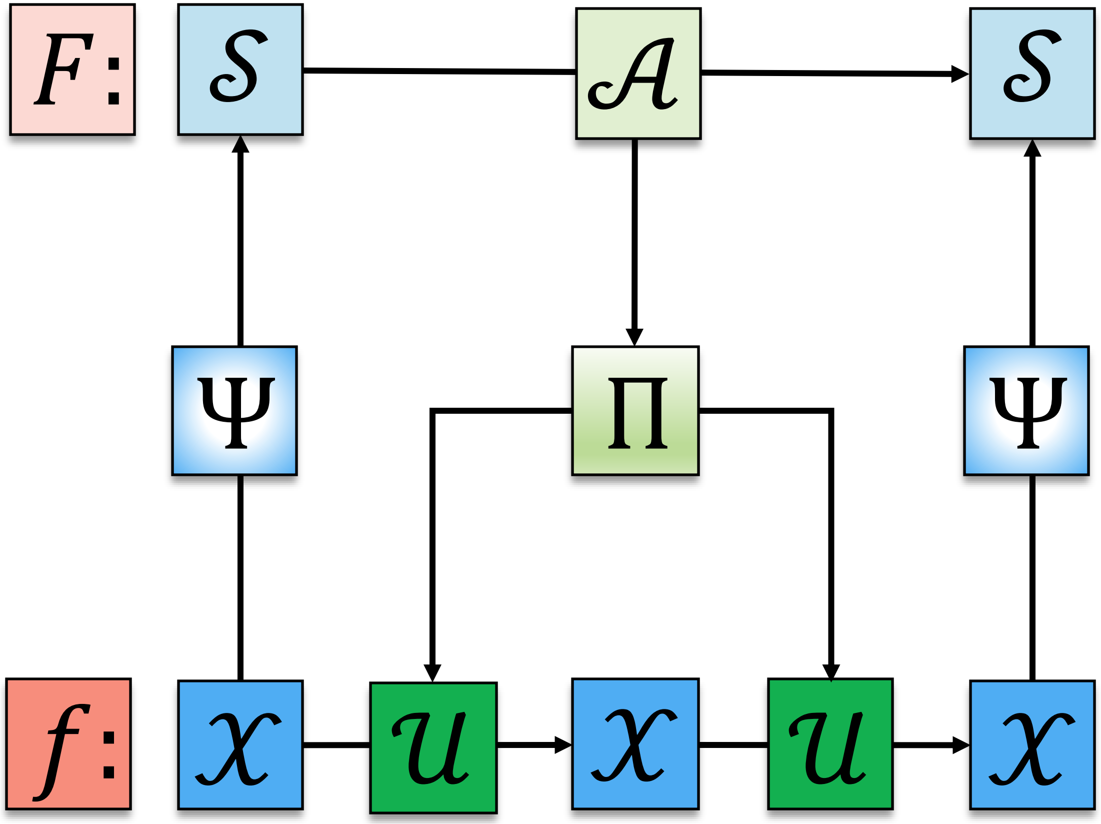
A Two-Level Hierarchy



Abstract action space

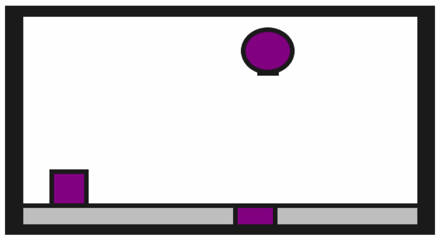
Action abstraction models

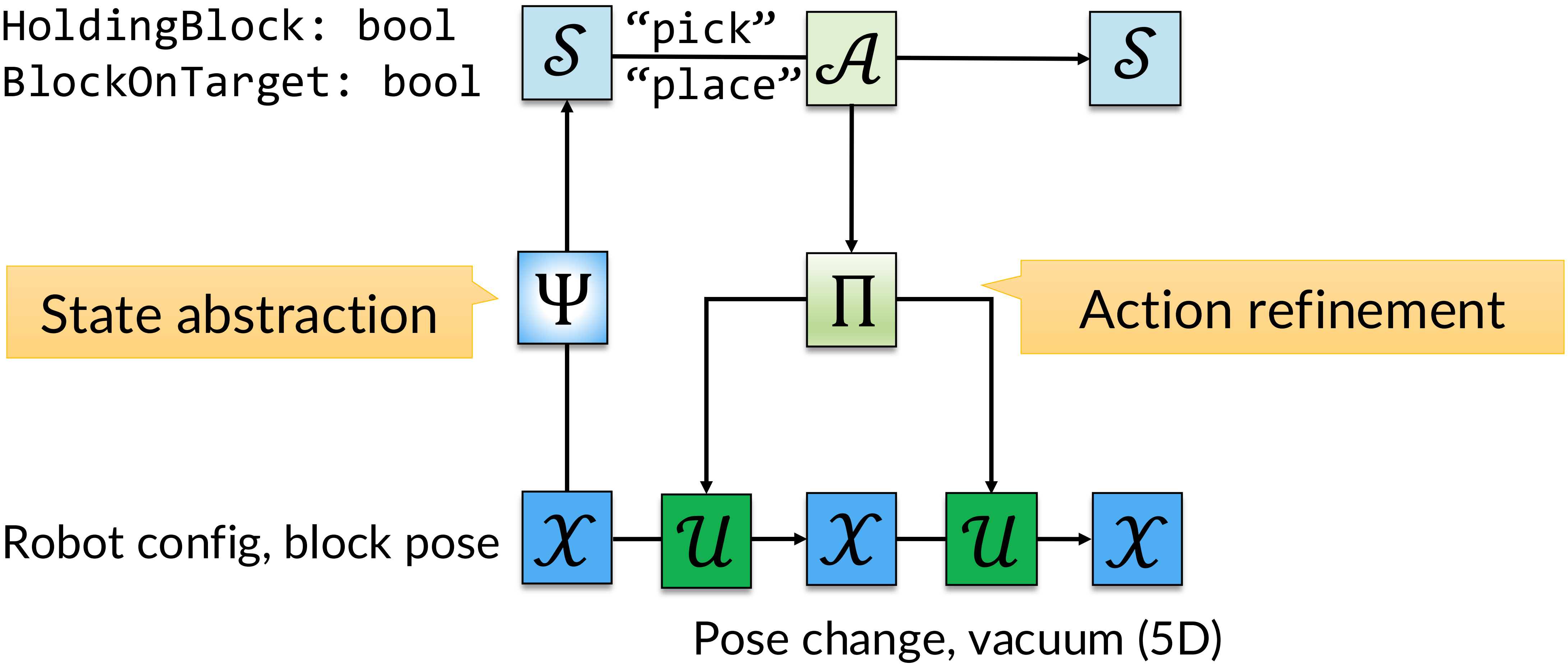
A Two-Level Hierarchy

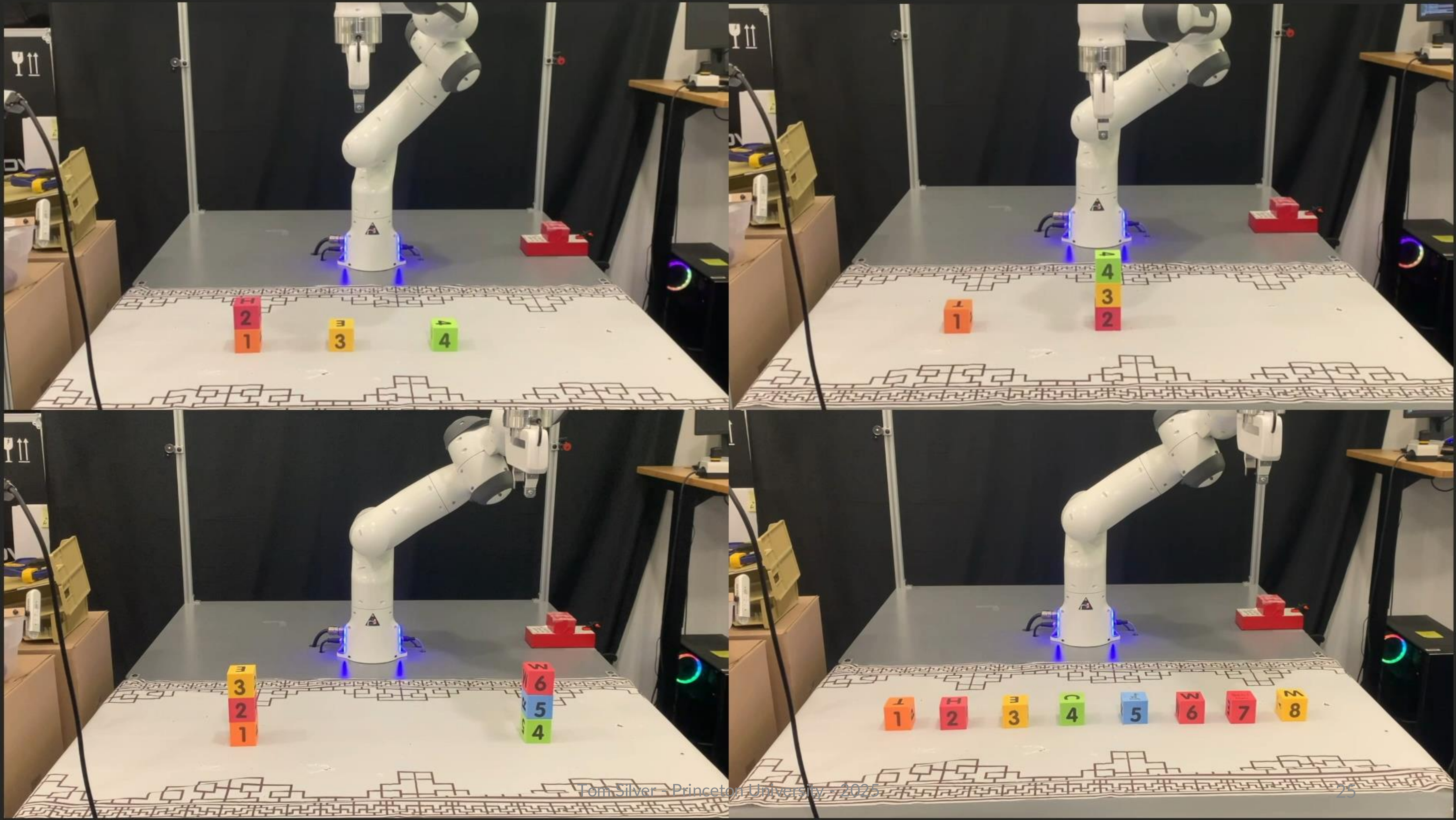


Maybe it's easier to plan up here!

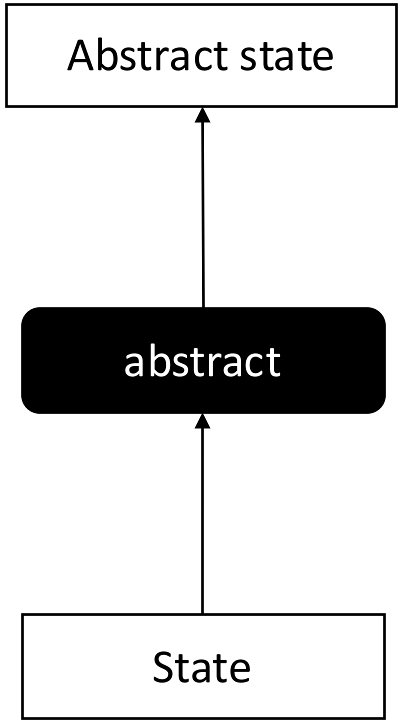
Abstractions in Example







State Abstraction with Predicates

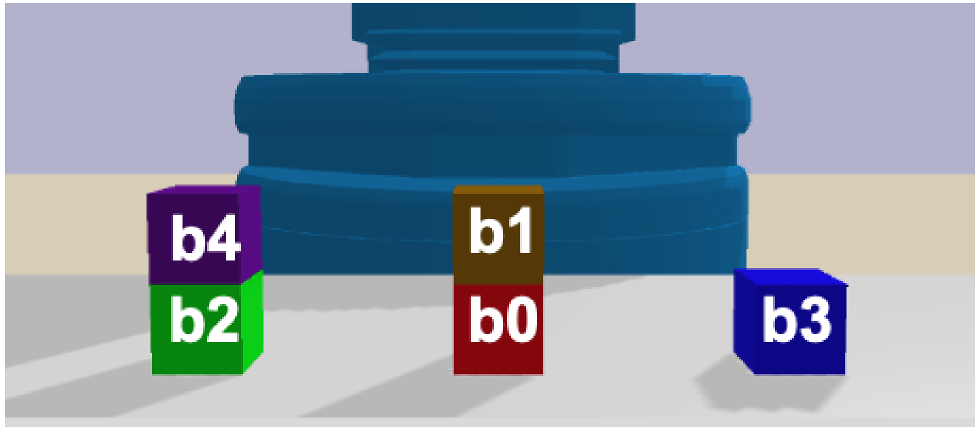



```
OnTable(b2), On(b4, b2)
OnTable(b0), On(b1, b0)
OnTable(b3)
```

	x	y	z	size
rob	0.63	0.11	0.94	0.5
b0	0.74	0.11	0.00	0.1
b1	0.75	0.10	0.20	0.1
b2	0.50	0.11	0.00	0.1
b3	0.99	0.12	0.00	0.1
b4	0.51	0.11	0.20	0.1

```
def classifyOnTable(state, ?block):
    state[?block].z < 1e-5

def classifyOn(state, ?top, ?bot):
    (state[?top].z - state[?bot].z -
     state[?bot].size) < 1e-5 &
    (state[?top].x - state[?bot].x) < 1e-5 &
    (state[?top].y - state[?bot].y) < 1e-5 &
```



Operators as Abstract Actions

Arguments

List of typed variables

Preconditions

What must be true in order to use this operator?

Add/Delete Effects

How is the abstract state changed by this operator?

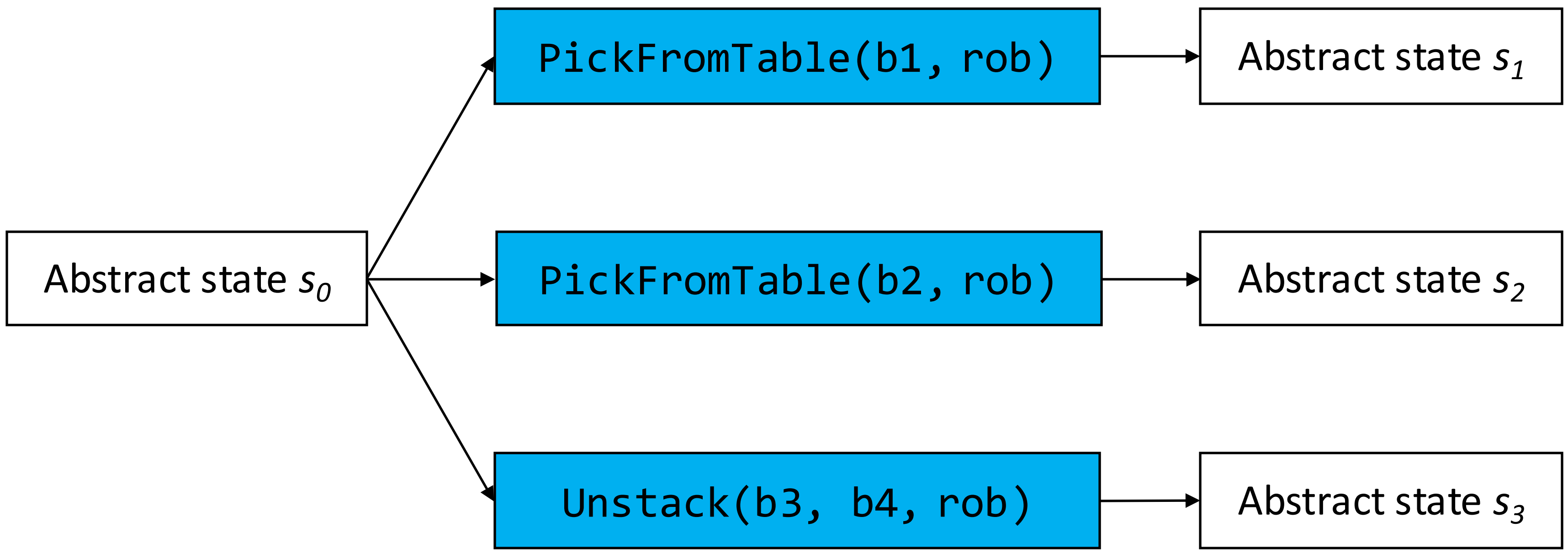
Operator-PickFromTable:

Arguments: [?b - block, ?r - robot]

Preconditions: {GripperOpen(?r),
OnTable(?b)}

Add effects: {Holding(?b)}
Delete effects: {GripperOpen(?r),
OnTable(?b)}

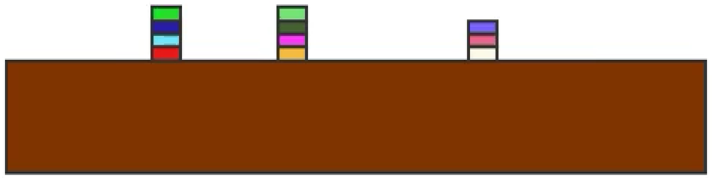
An Abstract Transition Model



An abstract (partial) transition model

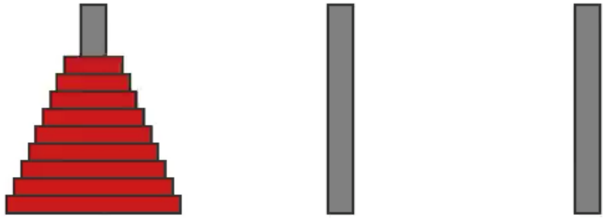
Why Predicates and Operators?

If we have predicates and operators, then we can use very powerful off-the-shelf symbolic planners!



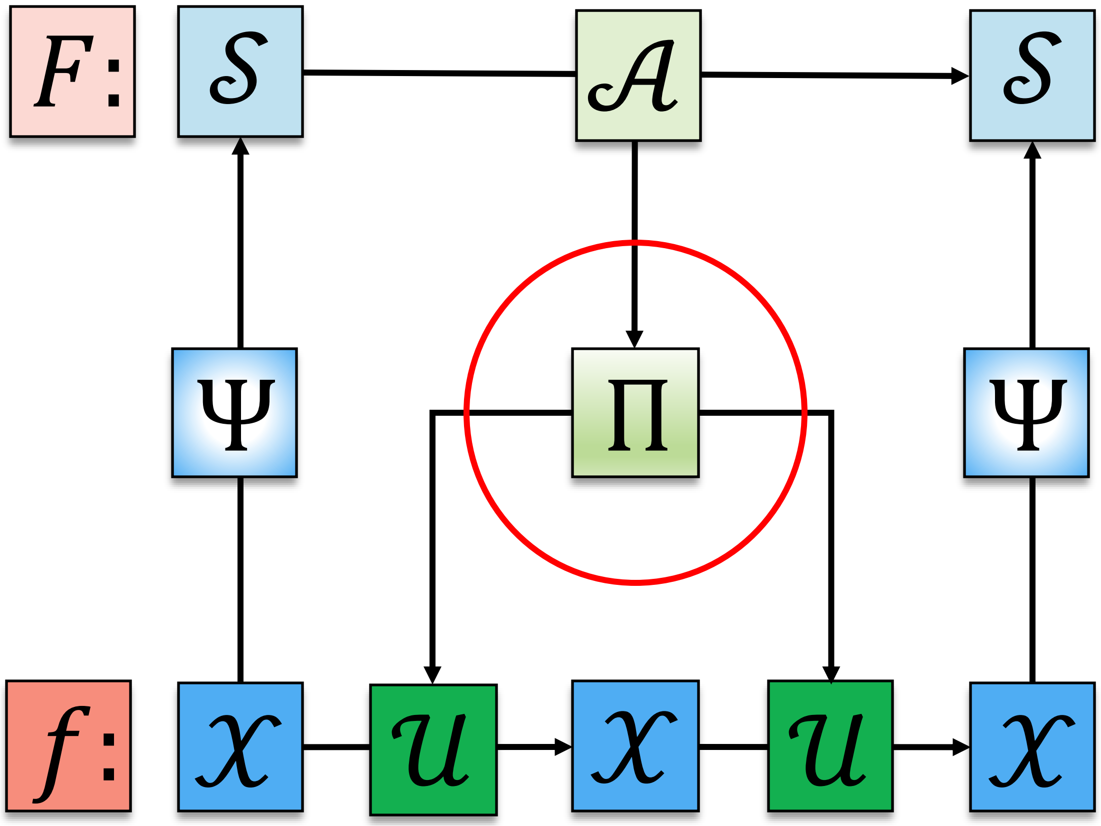
Blocks World
Plan length: 28
Planning time: 0.12 s

Sokoban
Plan length: 167
Planning time: 0.25 s

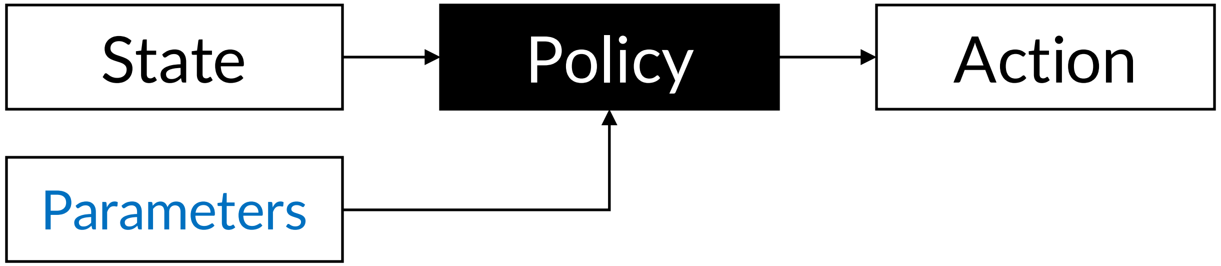


Hanoi
Plan length: 579
Planning time: 0.22 s

Planning with Fast Downward (https://www.fast-downward.org)
Rendering and simulation with PDDLGym (https://github.com/**tomsilver**/pddlgym)


F:
$\mathcal{S}$
$\mathcal{A}$
$\mathcal{S}$
Ψ
Π
Ψ
f:
$\mathcal{X}$
$\mathcal{U}$
$\mathcal{X}$
$\mathcal{U}$
$\mathcal{X}$

Policies as Abstract Action Models



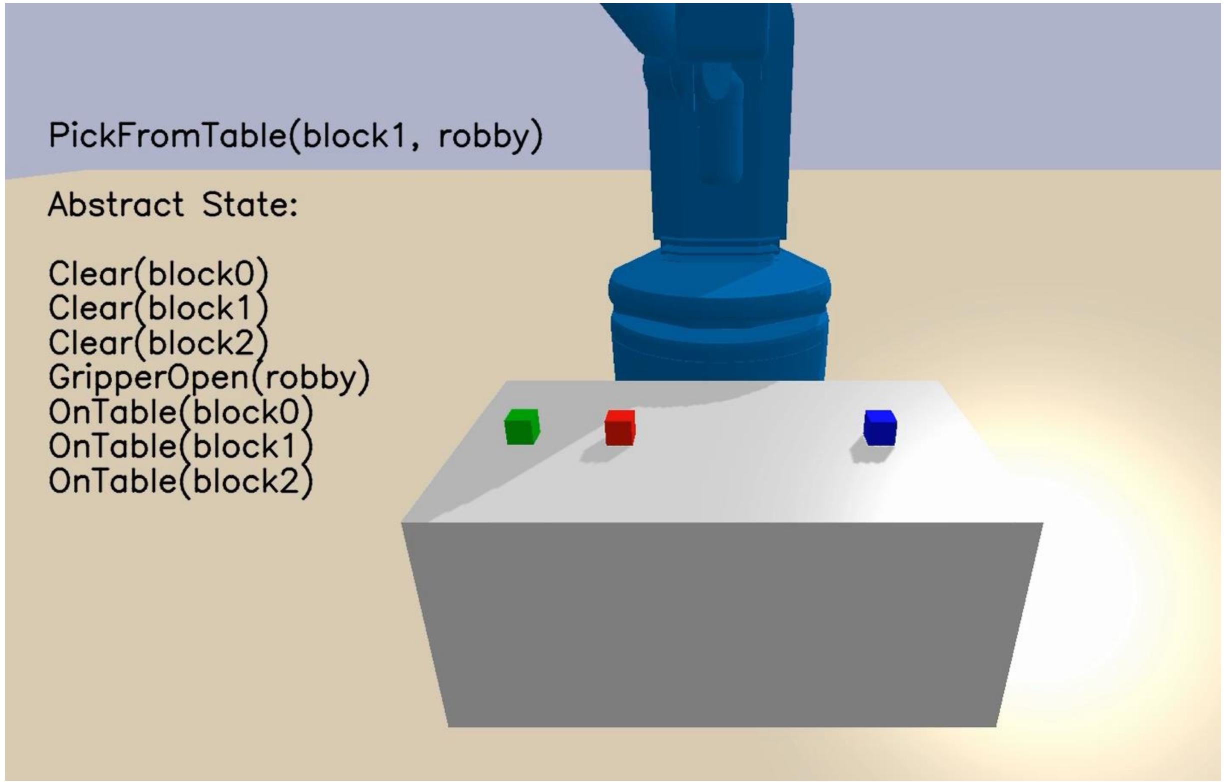
Same as operator

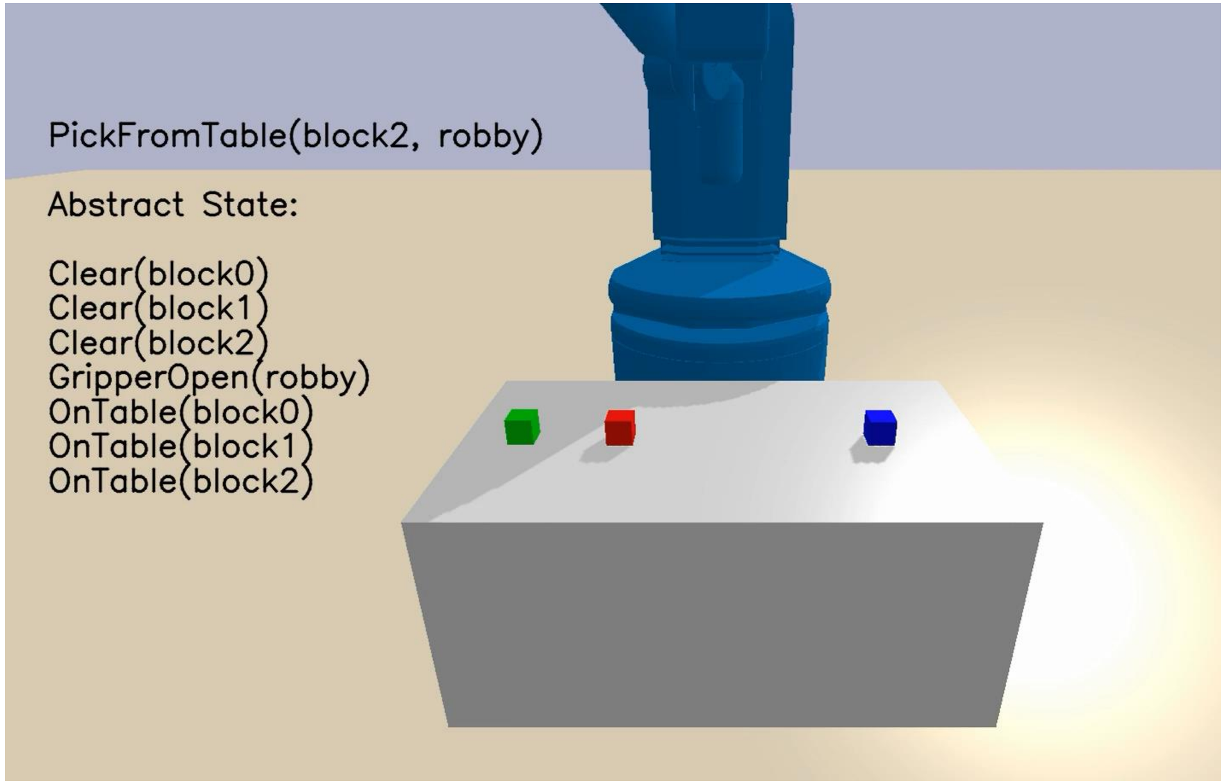
```
def policyPickFromTable(state, ?b, ?r):
    dx = (state[?b].x - state[?r].x)
    dy = (state[?b].y - state[?r].y)
    dz = (state[?b].z - state[?r].z)
    return [dx, dy, dz]
```

Simplified example

The policy should *achieve* the operator effects when the operator preconditions hold

Example Policy Executions



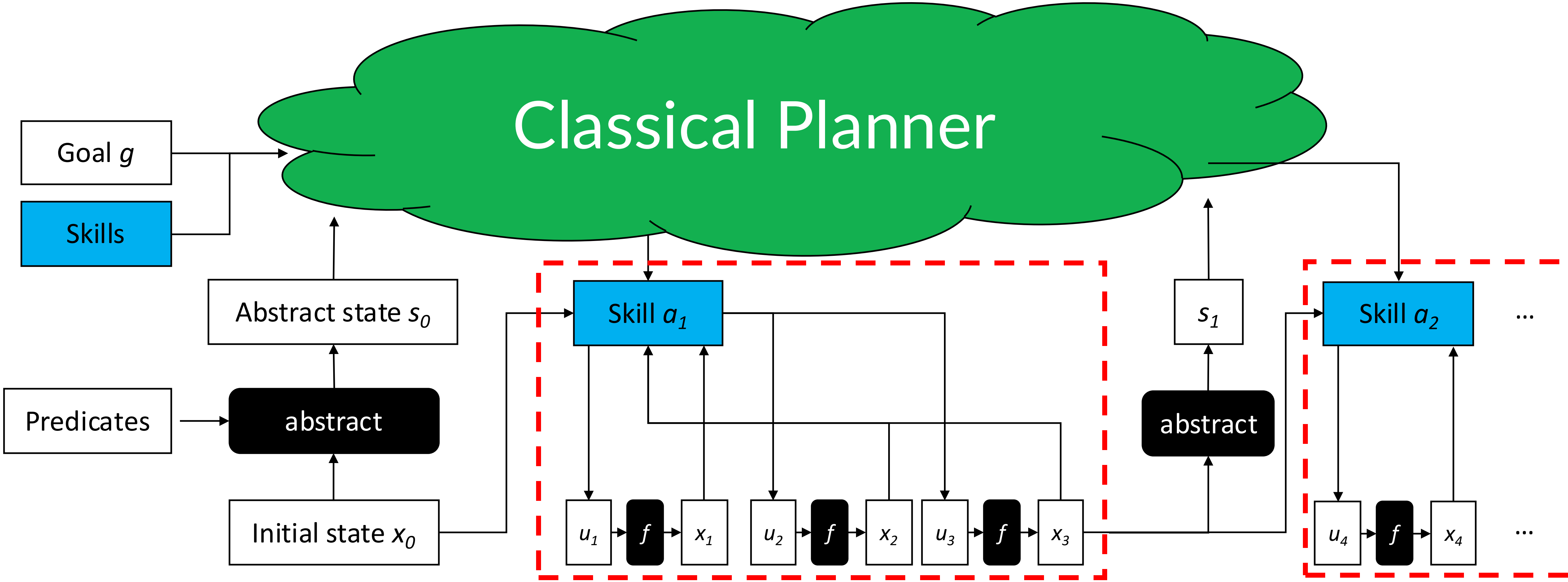


Skills: abstract actions that bring the robot from one abstract state to another

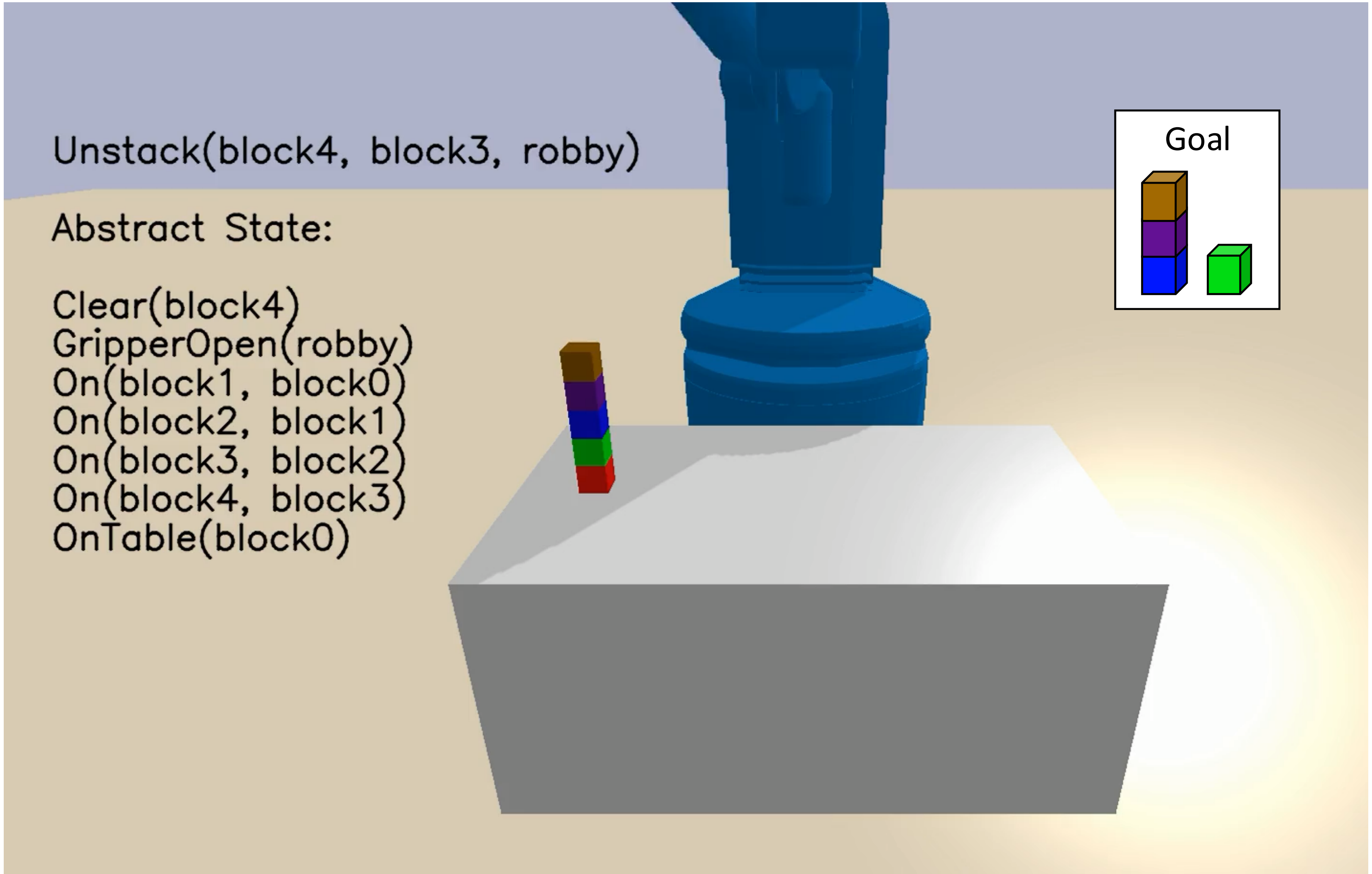
What abstract state transition?

How should I get there?

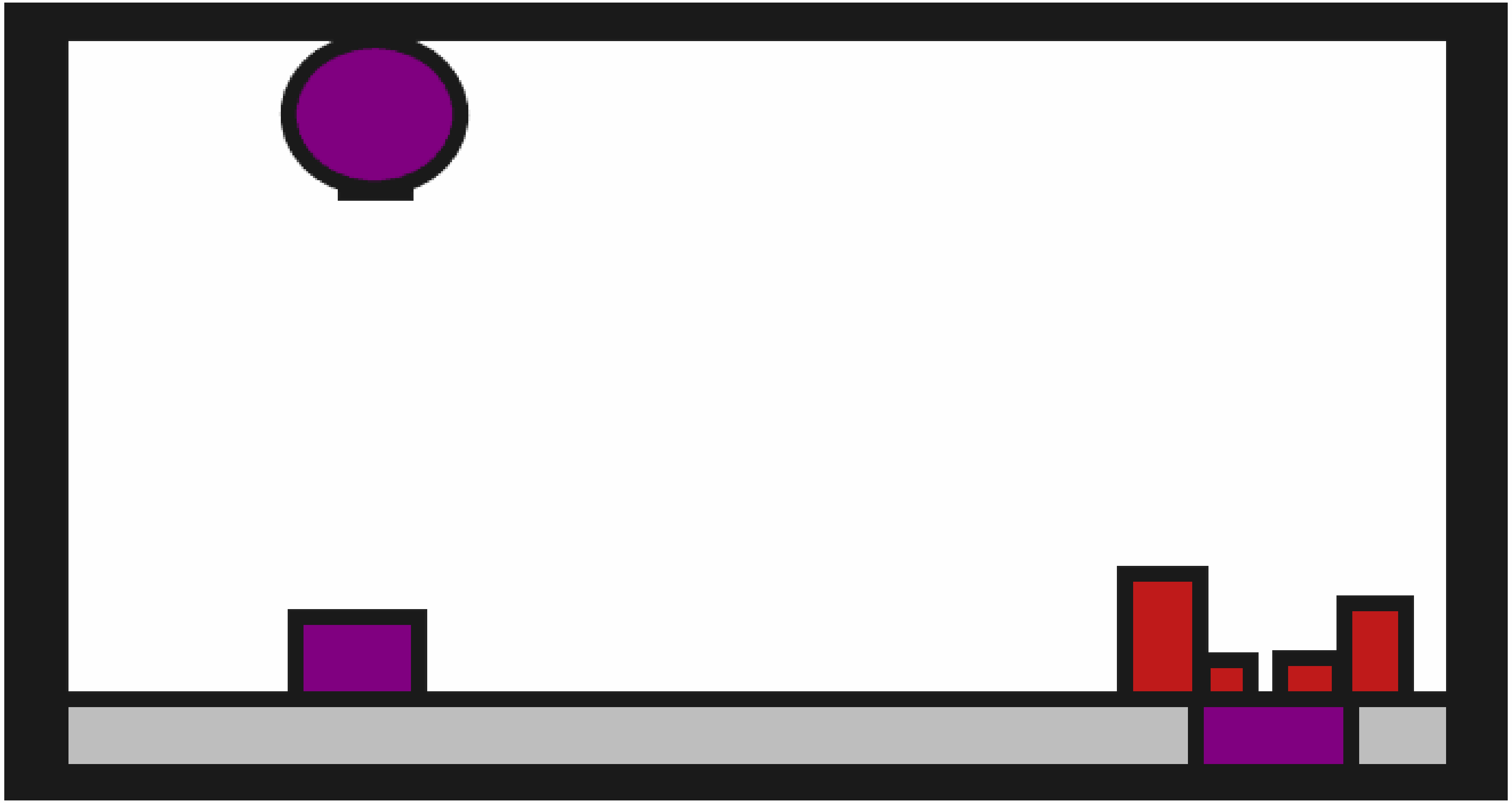
A skill has an **operator** and a **policy**

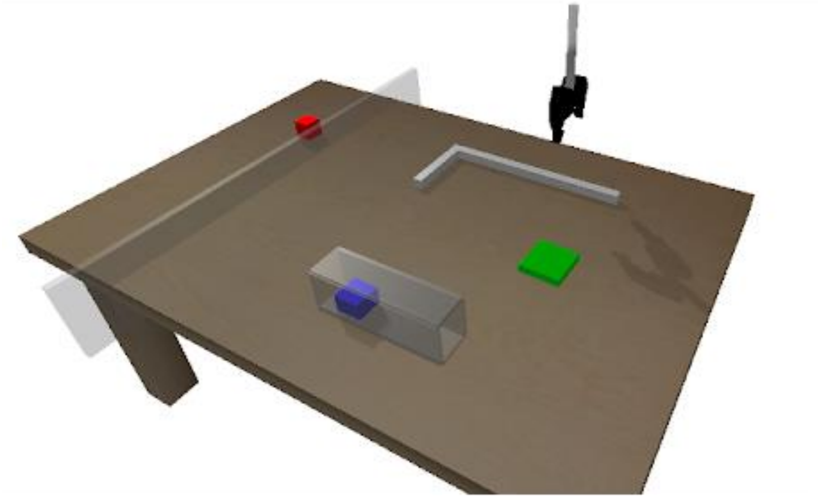


Action abstraction via skills


Unstack(block4, block3, robby)
Abstract State:
Clear(block4)
GripperOpen(robby)
On(block1, block0)
On(block2, block1)
On(block3, block2)
On(block4, block3)
OnTable(block0)
Goal

The abstractions might be liars…





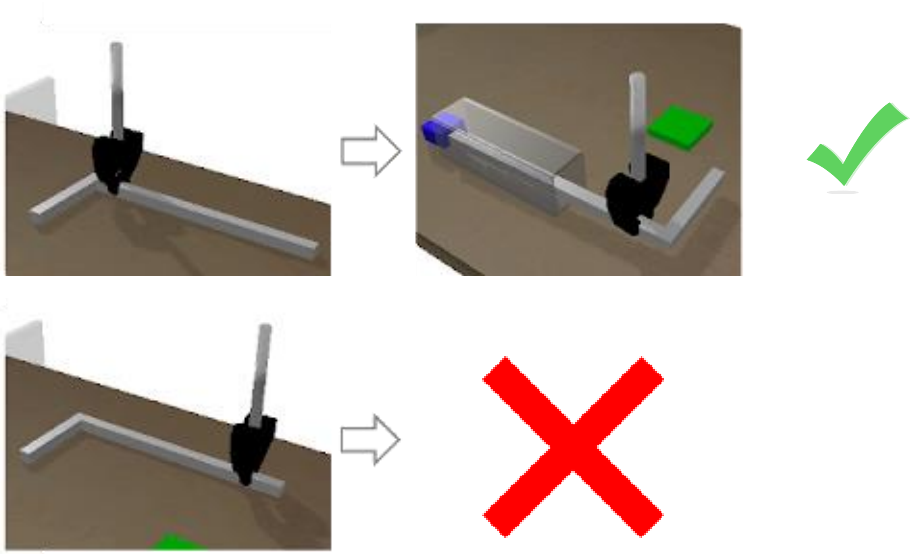
“Deep Affordance Foresight: Planning Through What Can Be Done in the Future.” Danfei Xu, Ajay Mandlekar, Roberto Martin-Martin , Yuke Zhu, Silvio Savarese and Li Fei-Fei. ICRA 2021.

Operator-PushOutOfTube:

Arguments: [, ,]

Preconditions: {Holding(,),
InTube( )}

Add effects: {OutOfTube( )}
Delete effects: {InTube( )}



Possible Conclusions from this Example

1. Insufficient predicates → learn new predicates

   HoldingBottom, HoldingTop, etc.

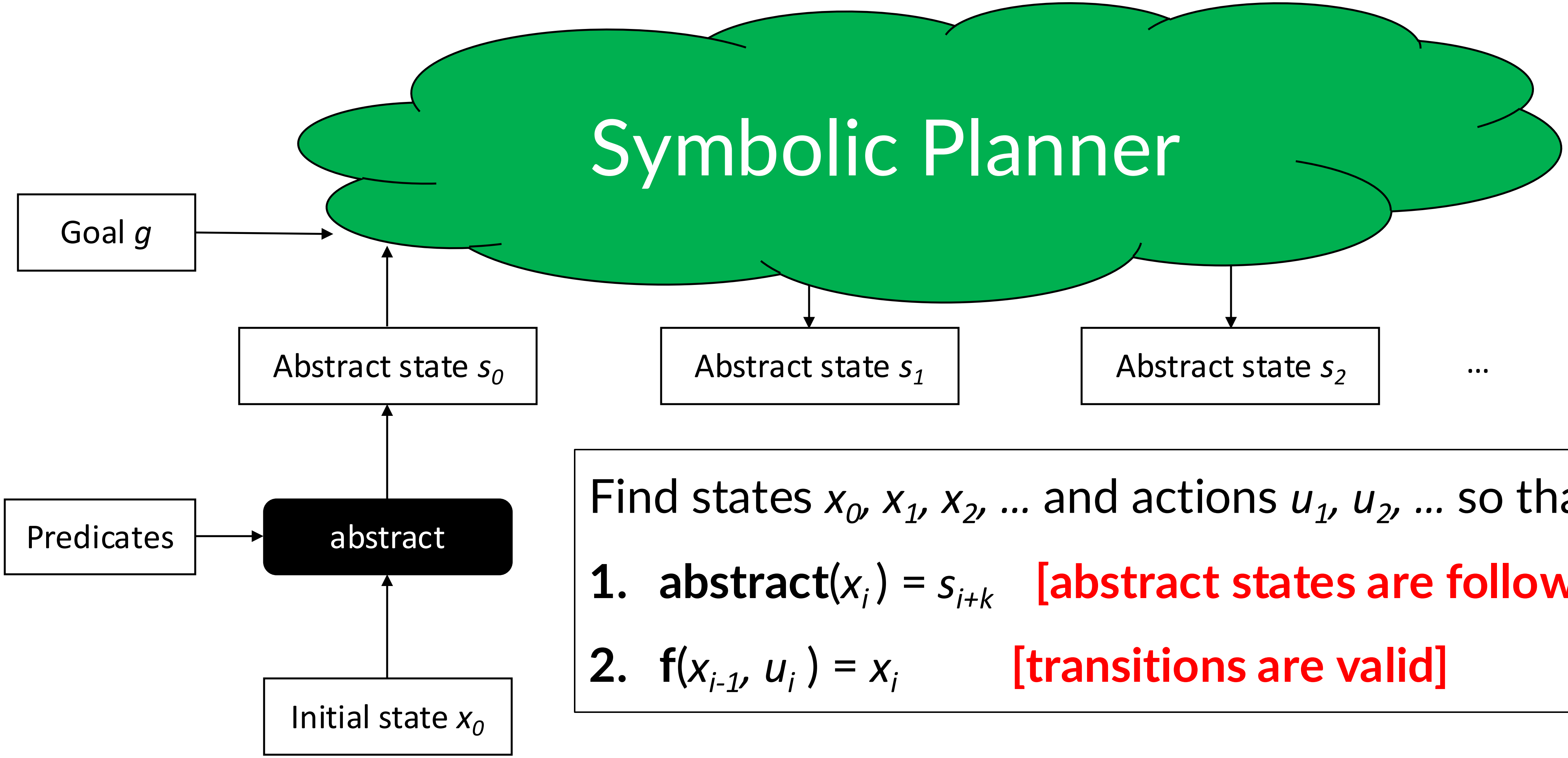
2. Insufficient policies → learn better policies

   Put down the tool and regrasp if needed

3. Insufficient planner → be less trusting of the abstractions

   View abstractions as *guidance* for low-level planning

Bilevel Planning: View Abstractions as *Constraints*

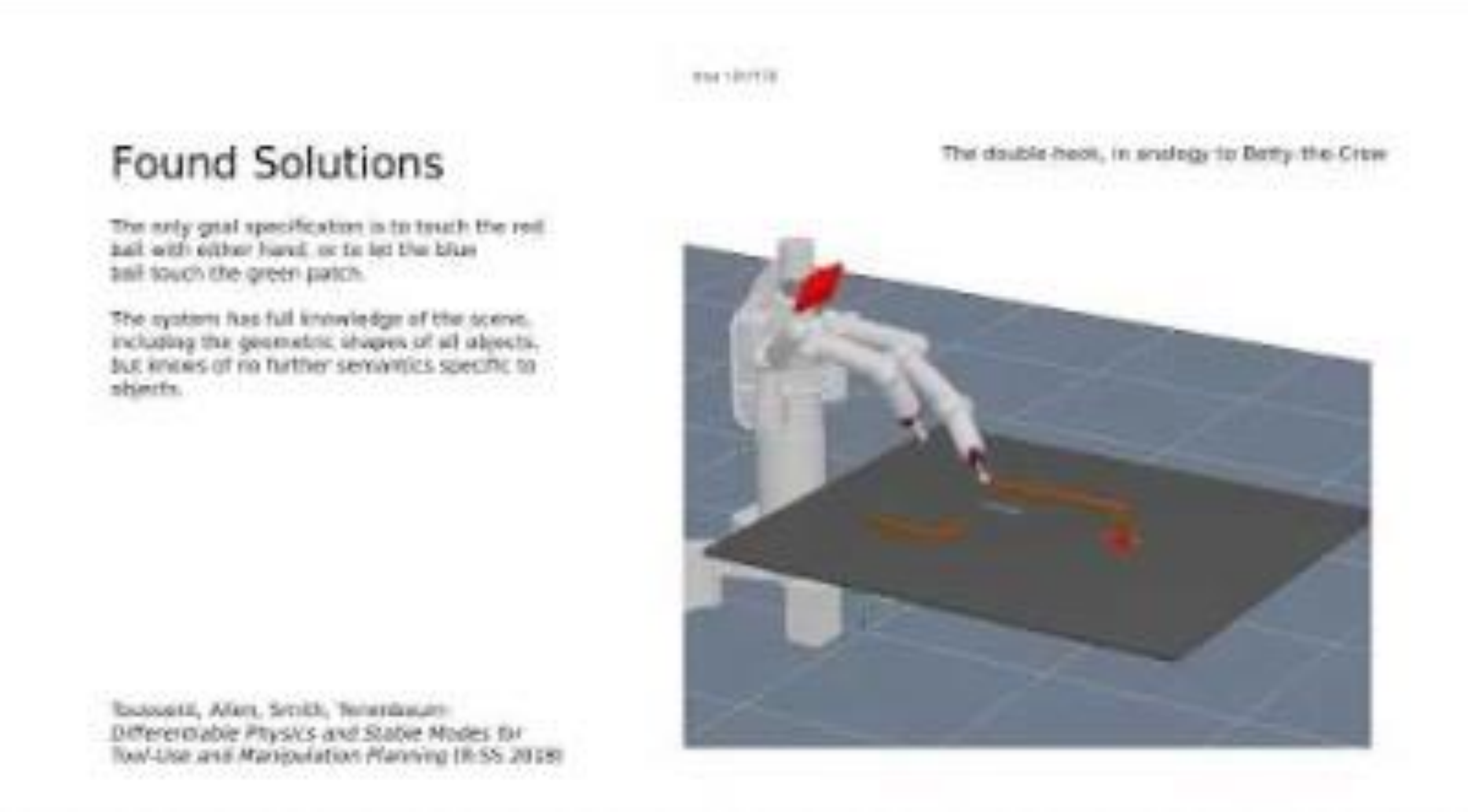


Find states $x_0, x_1, x_2, \ldots$ and actions $u_1, u_2, \ldots$ so that

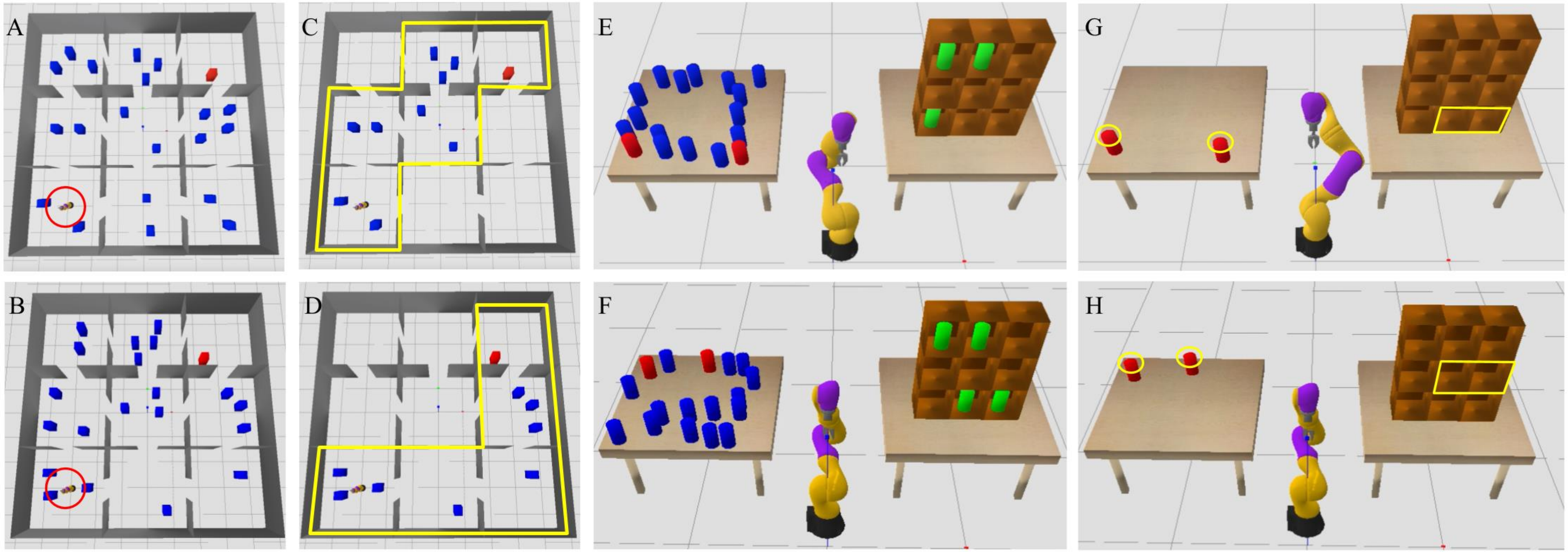
1. **abstract**$(x_i) = s_{i+k}$ **[abstract states are followed]**
2. **f**$(x_{i-1}, u_i) = x_i$ **[transitions are valid]**

Logic-Geometric Programming

Toussaint (2015)



Side Note: Constraints Can Help Planning in Multiple Ways



From Chitnis*, Silver*, et al. (2020)

Logic-Geometric Programming

Toussaint (2015)

Possible issues:

1. Optimizing in low-level state and action space remains hard for long-horizon problems

Recall from last lecture

General Trick 2: Optimize Splines Instead

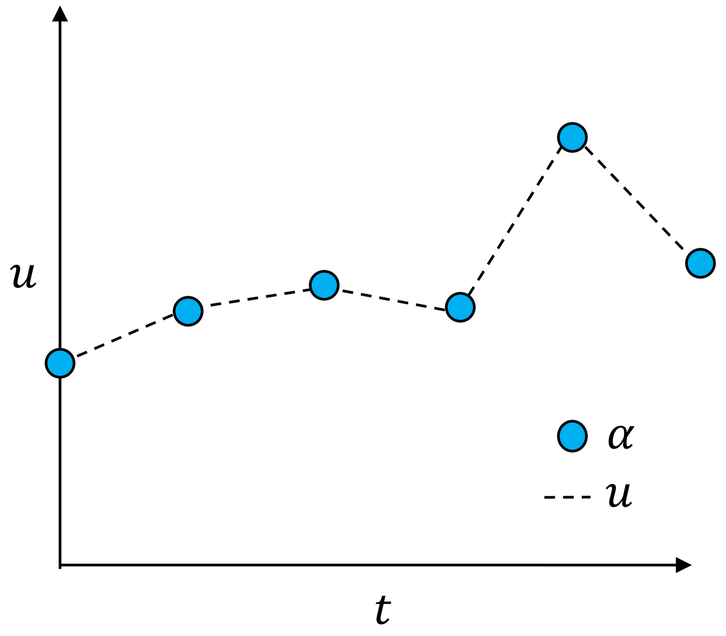
Optimizing $(u_0, u_1, \dots, u_{H-1})$ is slow for large H

Instead, optimize over lower-dimensional α:

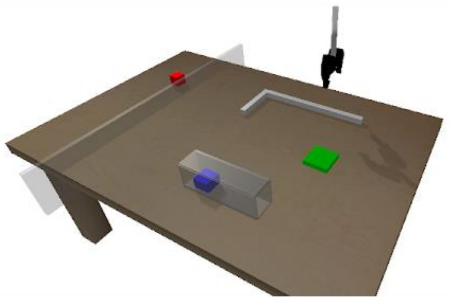
$$u_t = f(t, \boldsymbol{\alpha}) \text{ where } \boldsymbol{\alpha} \in \mathbb{R}^d \text{ and } d \ll mH$$

Common: think of α as "action waypoints" and interpolate between them

For example, linear splines (see right)



Parameterized Skill Policies



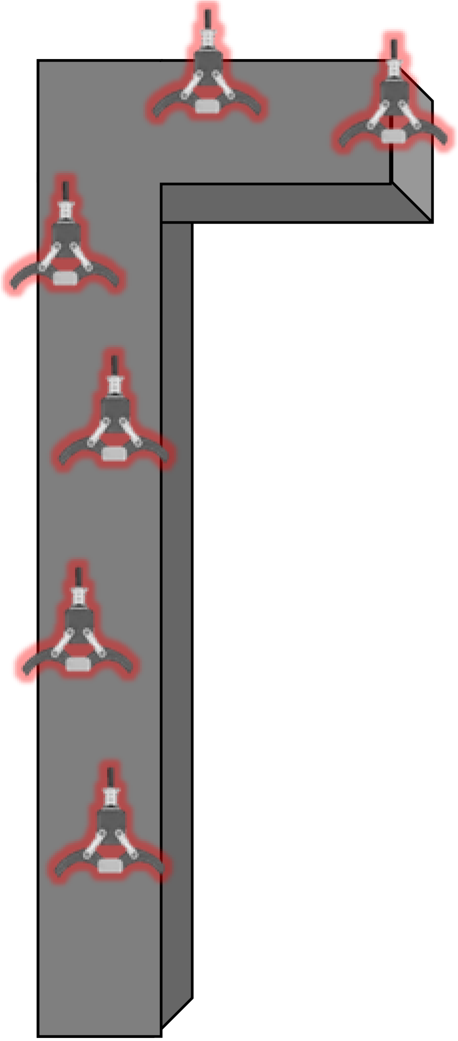
Operator-PickTool:

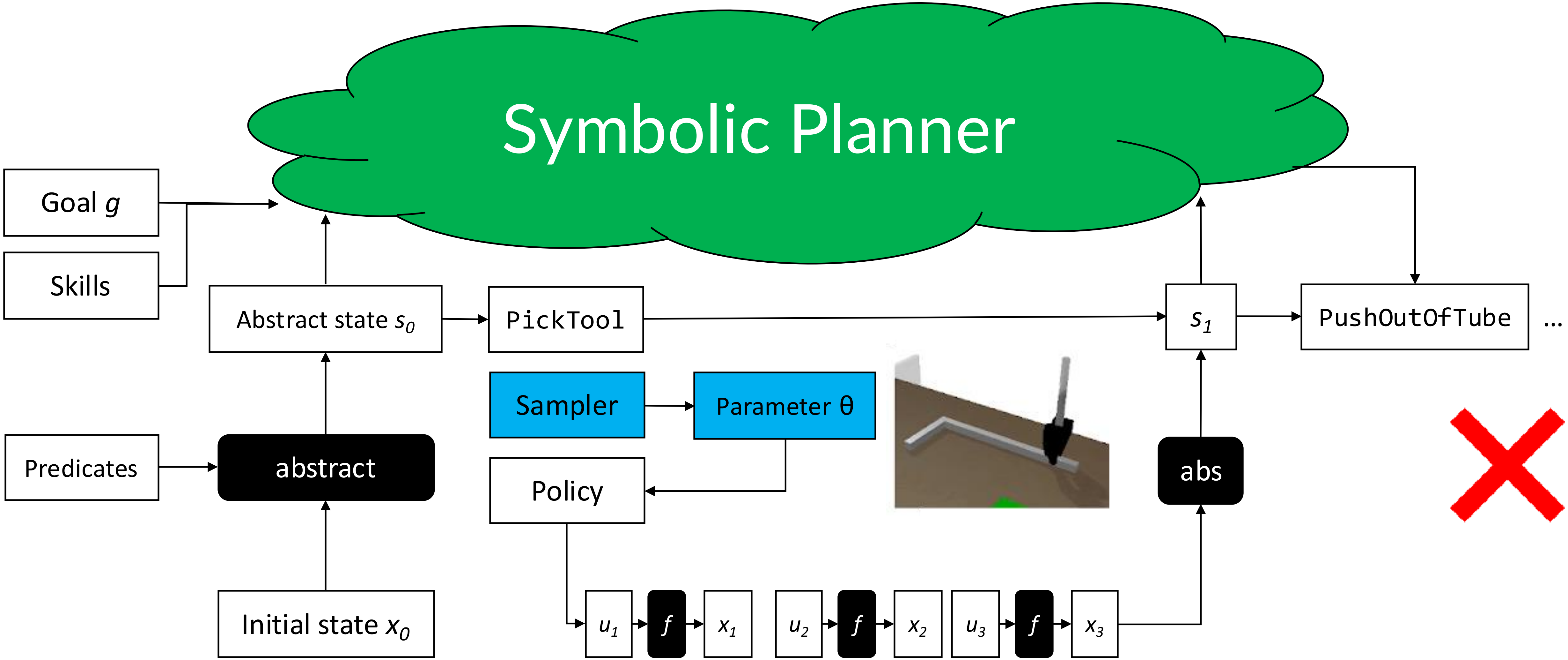
Arguments: [,]

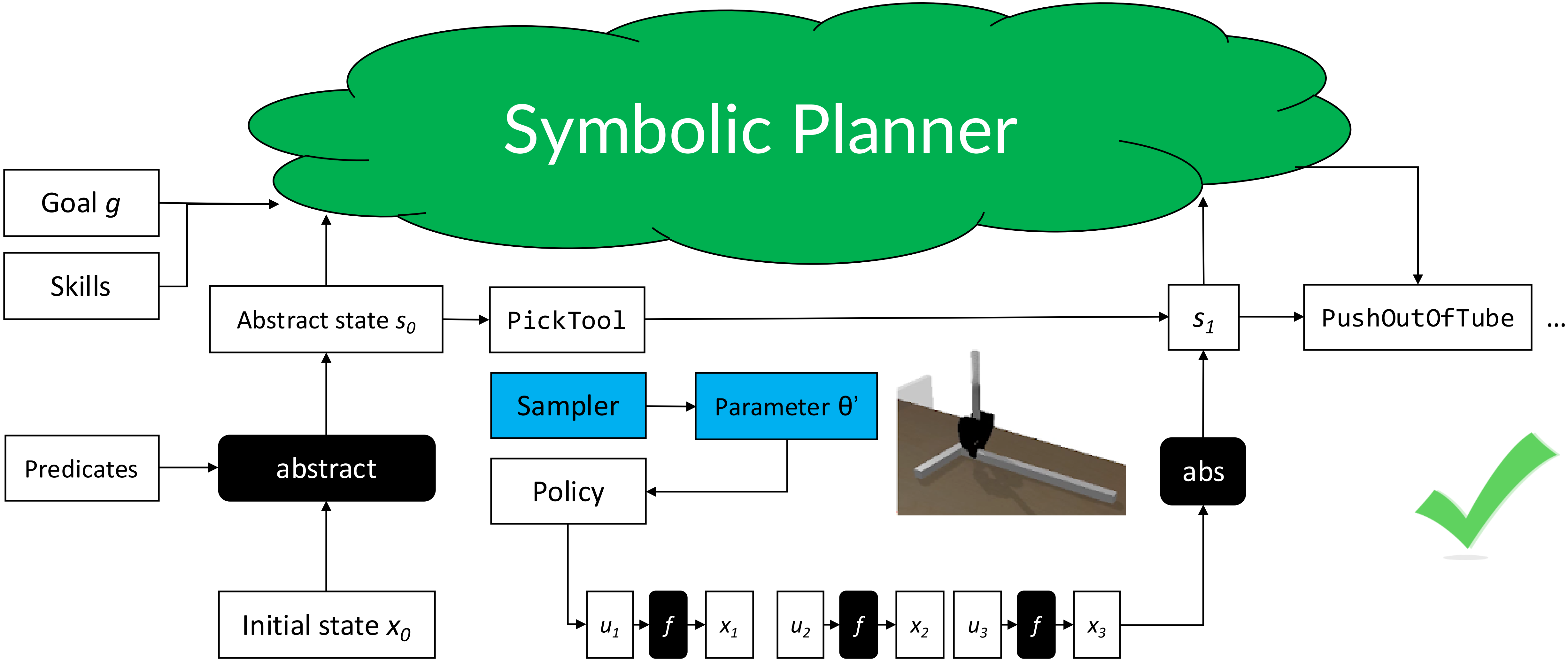
Preconditions: {GripperOpen( )}

Add effects: {Holding(,)}
Delete effects: {GripperOpen( )}

Different Parameters




Symbolic Planner
Goal g
Skills
Abstract state s_0
PickTool
s_1
PushOutOfTube
...
Sampler
Parameter θ
Policy
Predicates
abstract
abs
Initial state x_0
u_1
f
x_1
u_2
f
x_2
u_3
f
x_3



Symbolic Planner
Goal g
Skills
Abstract state s_0
PickTool
s_1
PushOutOfTube
...
Sampler
Parameter θ'
Policy
Predicates
abstract
abs
Initial state x_0
u_1
f
x_1
u_2
f
x_2
u_3
f
x_3

Logic-Geometric Programming

Toussaint (2015)

Possible issues:

1. Optimizing in low-level state and action space remains hard for long-horizon problems

2. **We still may have "contract disputes"…**

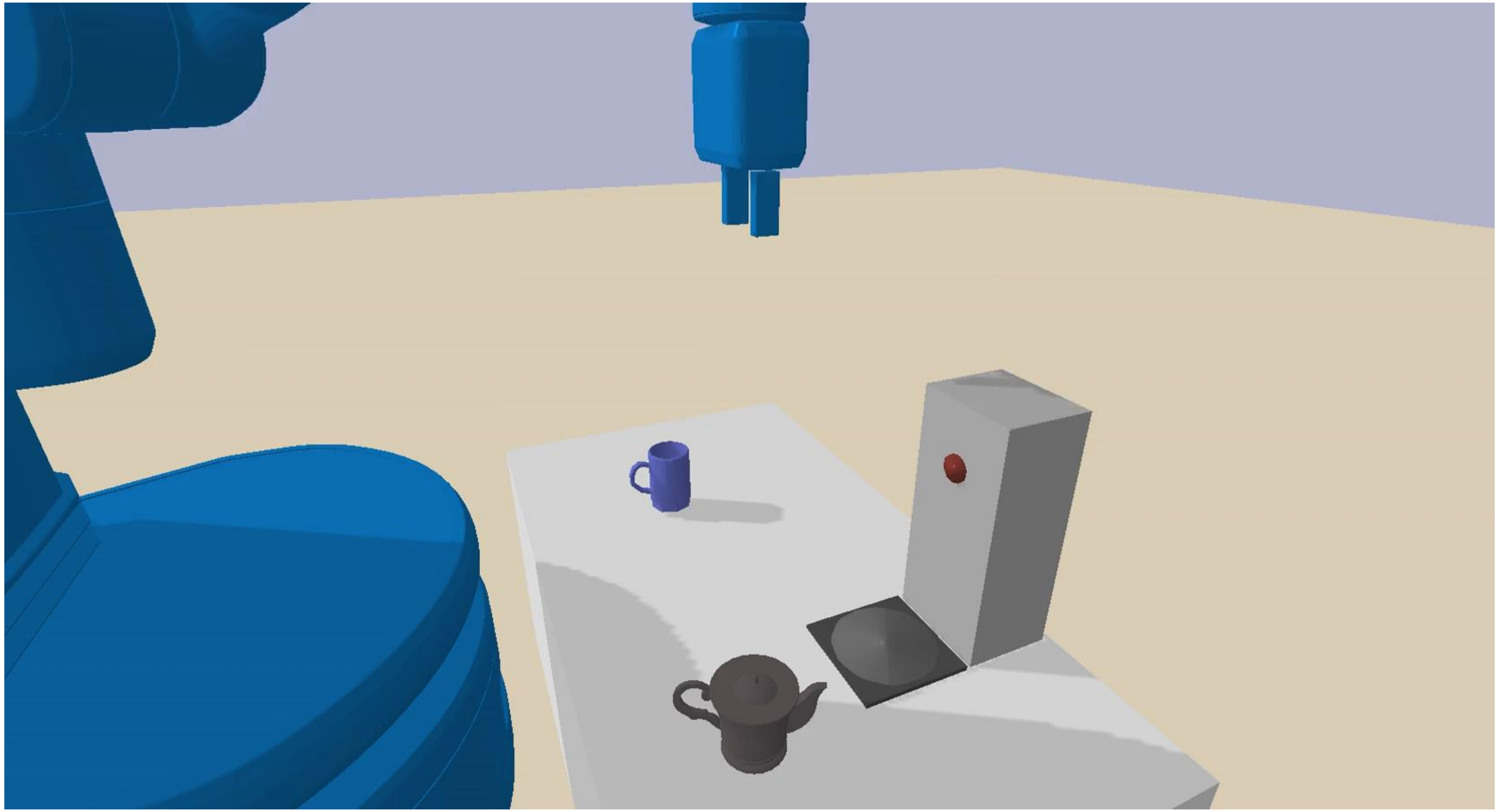
The abstractions may be *pathological* liars...

An abstract plan may not be refinable at all!

Coffee Domain

Abstract Plan

- Grasp handle
- Place on plate
- Turn plate on
- Pick up pot
- Pour into cup

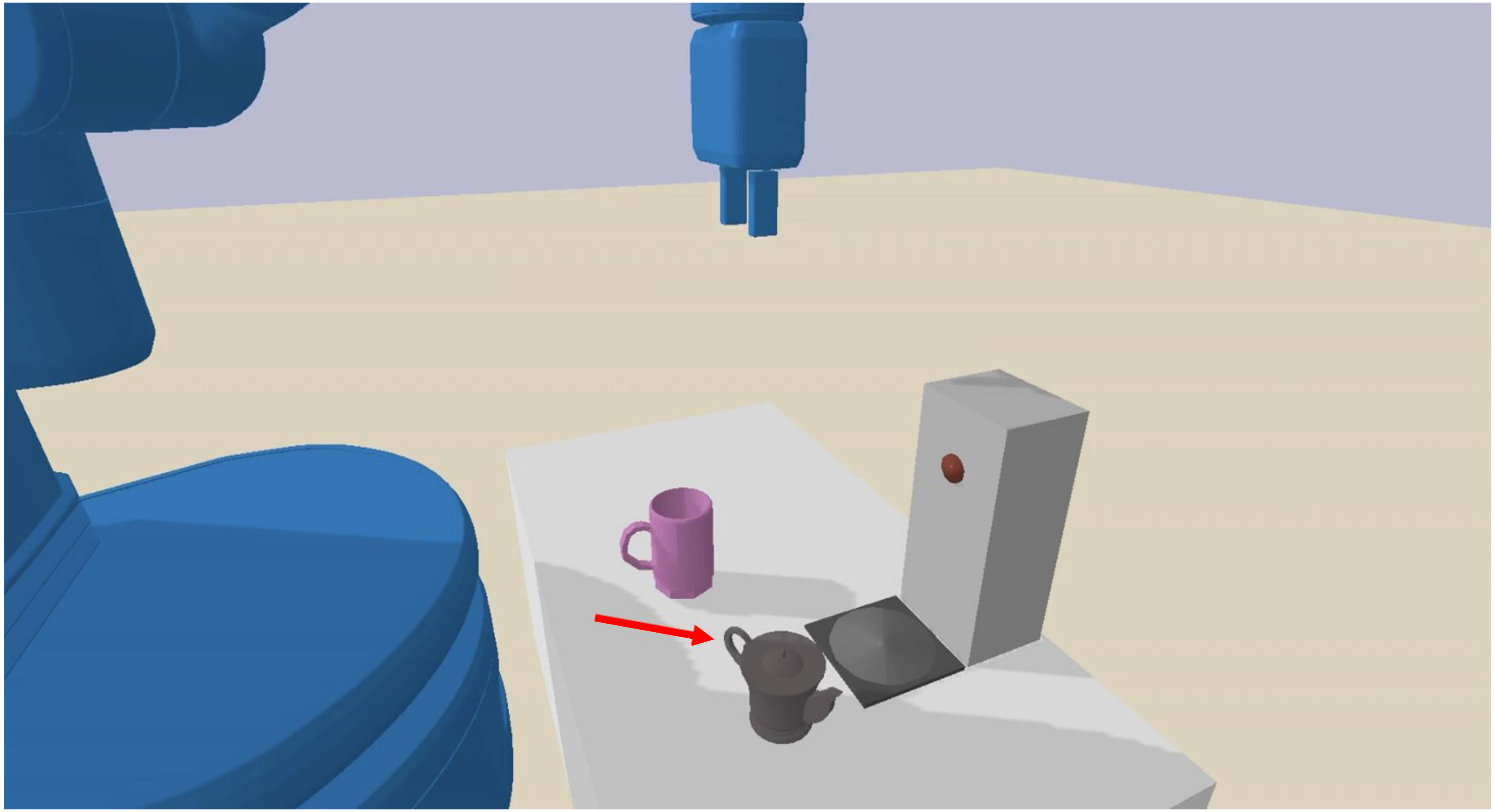


We need a different abstract plan!

Coffee Domain

Abstract Plan

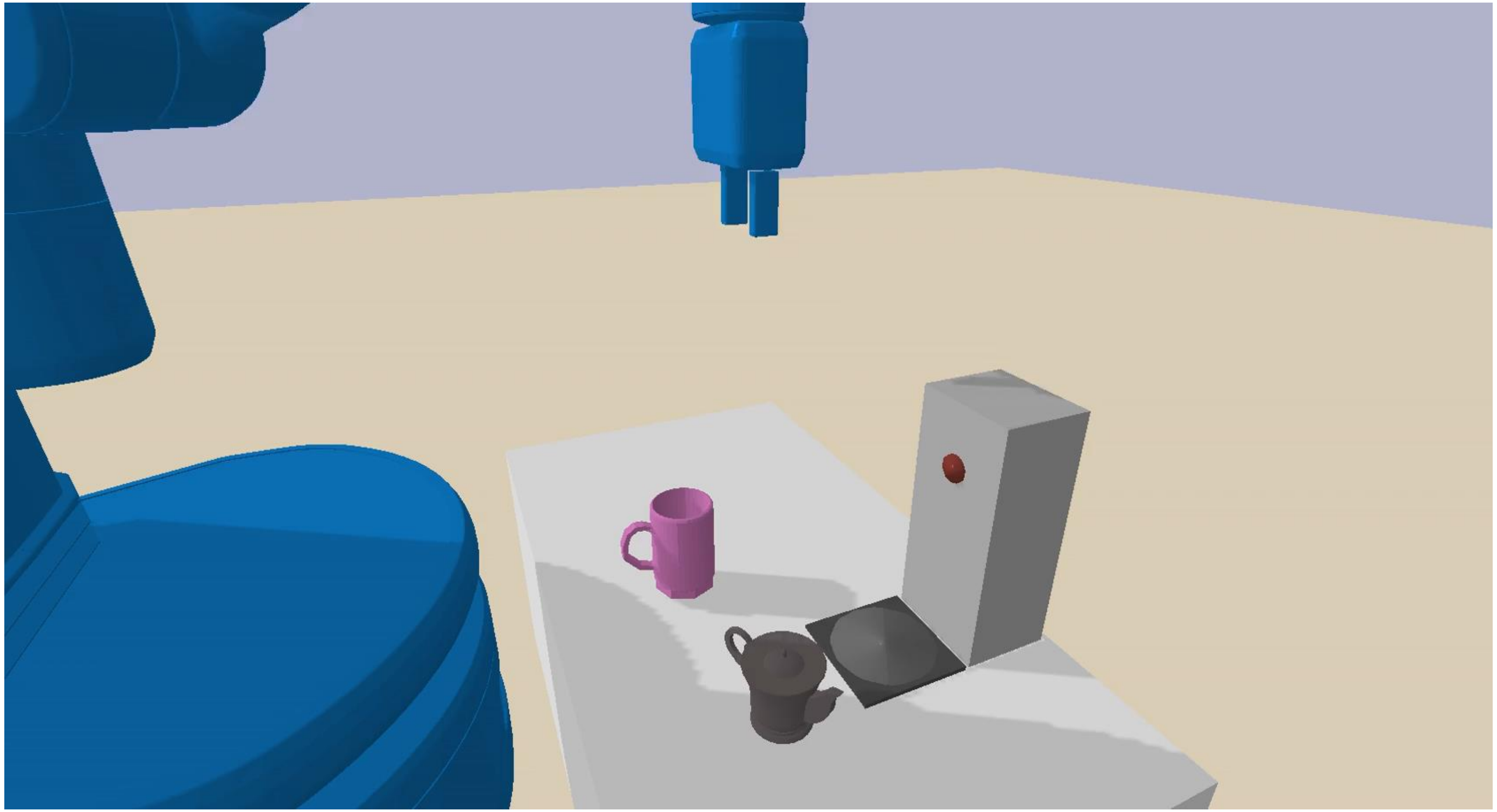
- ~~Grasp handle~~
- Place on plate
- Turn plate on
- Pick up pot
- Pour into cup



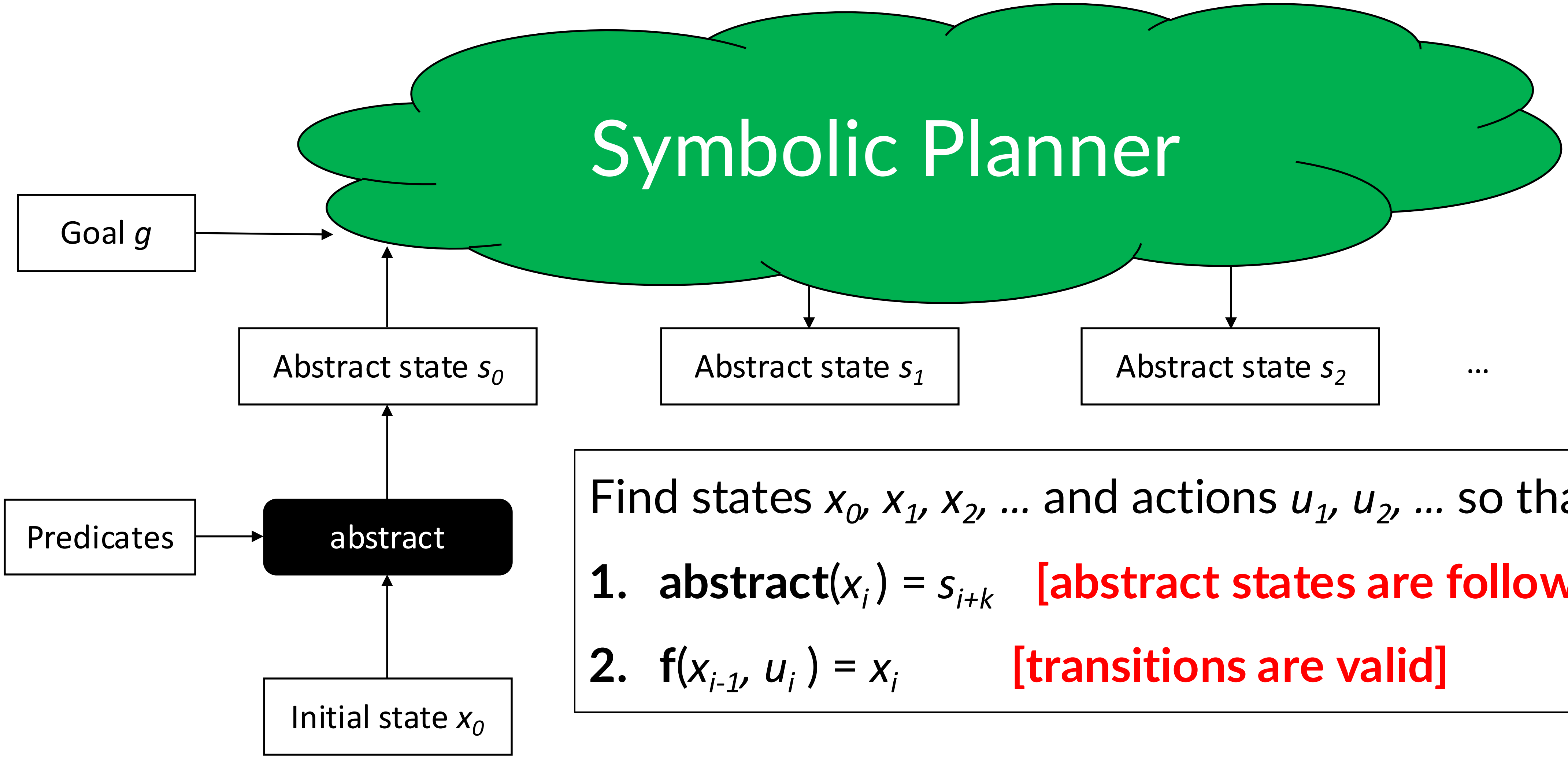
Coffee Domain

Abstract Plan

- Rotate pot
- Grasp handle
- Place on plate
- Turn plate on
- Pick up pot
- Pour into cup



One Remedy: Try Multiple Abstract Plans



Find states x_0, x_1, x_2, … and actions u_1, u_2, … so that

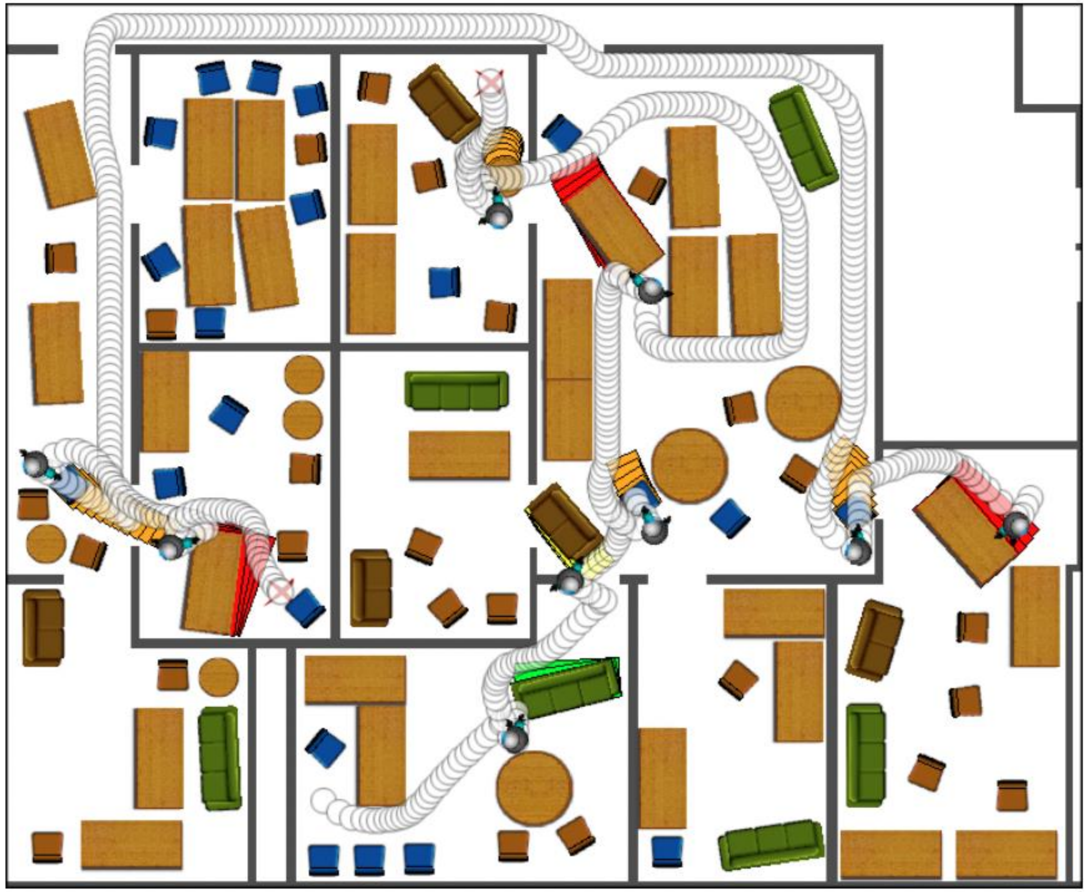
1. **abstract**$(x_i) = s_{i+k}$ **[abstract states are followed]**
2. **f**$(x_{i-1}, u_i) = x_i$ **[transitions are valid]**

Better: Use Feedback from Refinement Failure to Influence Task Planning

Example: "Navigation Among Moveable Obstacles (NAMO)"

(Stilman & Kuffner 2004)

(Simplified explanation)
When a collision is encountered during refinement, make a plan to move the collided object out of the way first



Another Approach: Sample *then* Search

- Extends ideas from sample-based motion planning (RRT, PRM)
- Instead of sampling just robot configurations, sample…
  - Candidate grasps
  - Candidate positions of objects
  - …
- Sample in a factored and conditional way
  - Example: conditioned on a future object position, sample a grasp
  - Conditioned on a grasp, sample a robot base position
  - Can sample “forward”, “backward”, or any-which-way in time
- See: PDDLStream (Garrett et al. 2018)

Summary: Task and Motion Planning (TAMP)

- Plan with state and action abstractions

- Use relational abstractions (e.g., PDDL) when possible

- Beware that the abstractions might be “liars”
  - TAMP is most interesting in this case!

- Use the abstractions as “guidance” for planning

- Closely related to hierarchical RL