Robot Autonomy

Lecture 11: Task Planning

Oliver Kroemer

Complex Tasks

- Often need to perform complex tasks
  - Travel to another city
  - Replace the tire on a car
- Difficult to reasoning about tasks directly at c-space level
  - Matching nuts and bolts together
- Consist of multiple steps that need to be performed
  - Lift car, remove bolts, remove tire, place new tire…
  - Can use previous planners for individual steps
- Plan for entire task at the more abstract symbolic level

Task Planning Problem

- The robot is given:
  - An initial state
  - A goal characterisation (set of goal states)
  - A set of potential actions

- The robot needs to generate:
  - A sequence of actions that transforms the initial state into a goal state

- Need to model the state and the actions to plan

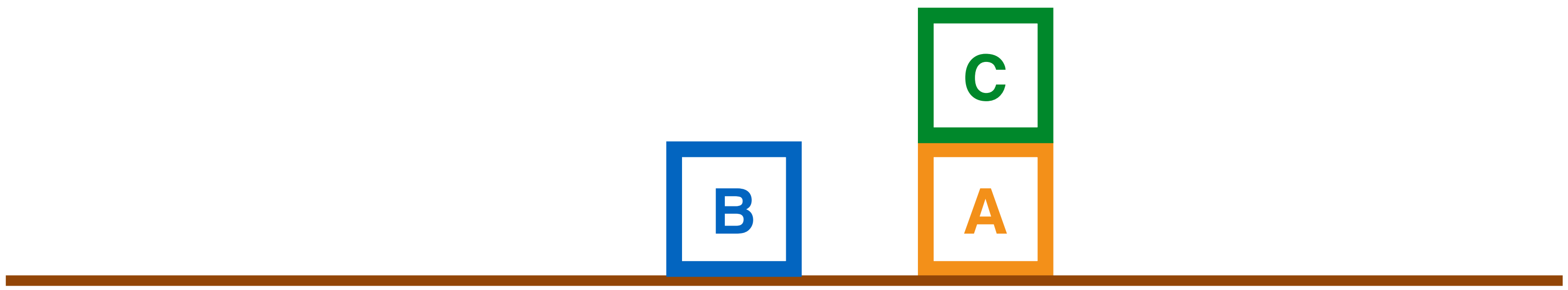
Representing States

- The state is represented by set of objects and predicates
- Constant symbols to represent objects in the domain
  - Table, LugNut1, LugNut2, Car, Tire, Axle, …
  - Obj1,Obj2, Obj3, Obj4, …
  - Assume unique names
- Predicate symbols to represent relations and properties
  - Bolt(a), On(a,b), Grasped(a,b), Metalic(a), Insertable(a,b,c)
  - Arity is the fixed number of arguments the predicate takes

Representing States

- The state is defined by a set of grounded predicates
  - Tire(Flat) ^ Tire(Spare) ^ At(Flat,Axle) ^ At(Spare,Trunk)

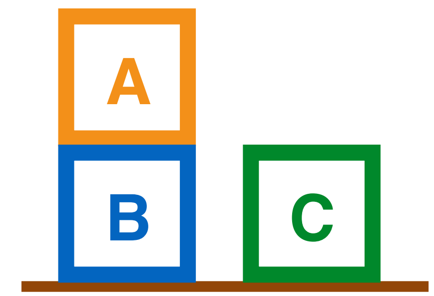
  - On(A,Table) ^ On(B,Table) ^ On(C,A) ^ Block(A) ^ Block(B) ^ Block(C) ^ Clear(B) ^ Clear(C) ^ Clear (Table)

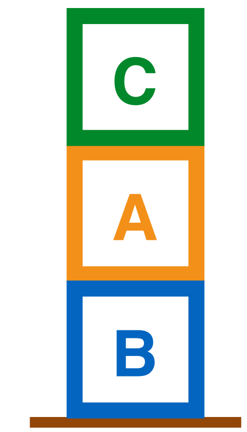


- Closed-world assumption:
  - Only include the positive predicates in the state list
  - All others are assumed to be negative

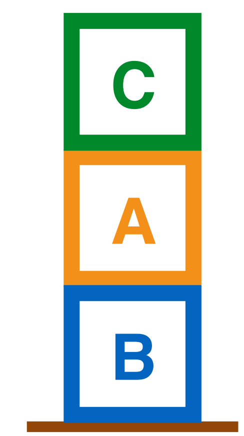
Goal Characterization

- Goals are characterised by a set of conditions
  - On(A,B) ^ Clear(C)
  - Goal states entail these conditions

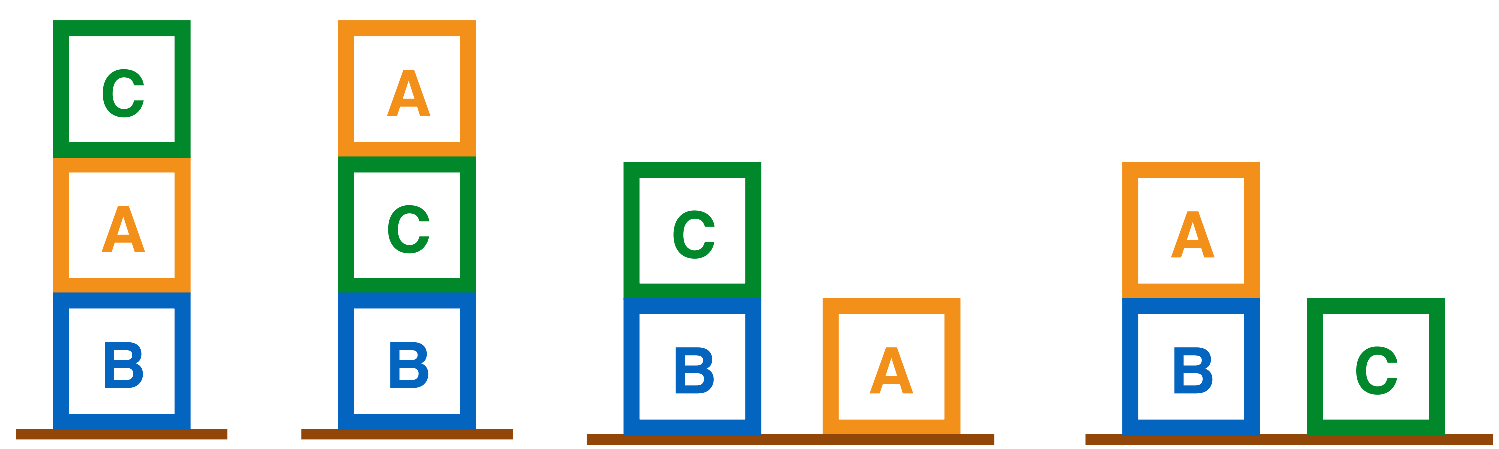




- Can include positive and negated conditions
  - On(A,B) ^ ~Clear(A)

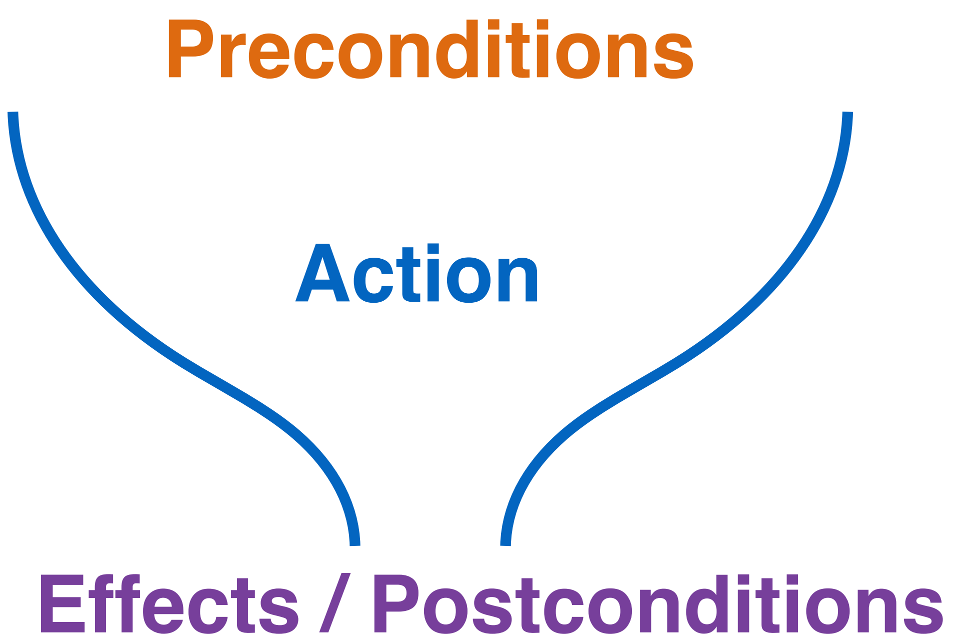


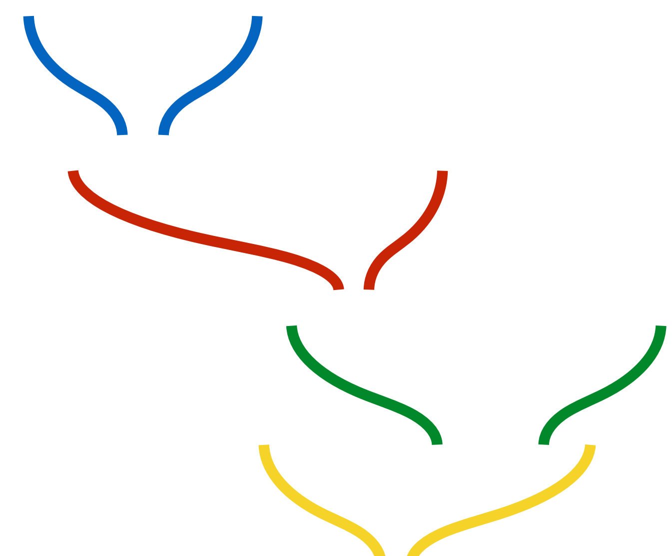
- Can include variables (true for any)
  - On(x,B) ^ Block(x)



Representing Actions

- Actions result in a change of state
  - Need to model the effects of the actions
- Actions are only applicable in certain states
  - Need to model the preconditions of the actions

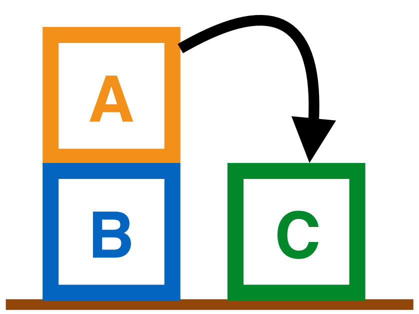


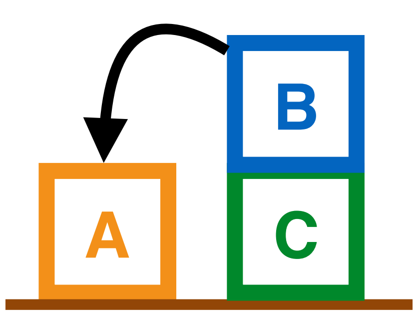


- Use pre- and post- conditions to sequence actions
- Actions are often interpreted as skills and planners

Action Schemas

- Defining every possible action is exhausting
- Define action schema, i.e. lifted actions with variables
    - Action Schema: PutOn(t, Axle)
      Action Examples: PutOn(Spare, Axle) or PutOn(Flat, Axle)
    - Action Schema: Move(b,x,y)
      Action Examples: Move(A,B,C) or Move(B,C,A)





- Why not Move(Table,A,B) or PutOn(Axle,Axle)?

Preconditions

- Precondition define states in which the action can be executed
- Preconditions have the same form as goal representation
  - Action Schema: PutOn(t, Axle)

    Preconditions:
    Tire(t)^At(t,Ground)^~At(Flat,Axle)^~At(Spare,Axle)

  - Action: Move(b,x,y)

    Preconditions:
    On(b,x) ^ Clear(b) ^ Clear (y) ^ Block(b) ^ Block(y) ^ Block(x)
    (b ~= y) ^ (b ~=x) ^ (x~=y)

Effects

- Performing an action has an effect on the state
  - Action Schema: PutOn(t, Axle)
    Effects:
    ~At(t,Ground)^At(t,Axle)
  - Action: Move(b,x,y)
    Effects:
    ~On(b,x) ^ ~Clear(y) ^ On(b,y) ^ Clear(x)
- Often model effects as delete and add lists
- Variables in the effects must also be in the preconditions

Planning Domain Definition Language

- In practice, define problem using planning language, e.g.,
  Stanford Research Institute Problem Solver (STRIPS) ’71
  Action Description Language (ADL) ‘87
  Planning Domain Definition Language (PDDL) ’98

- PDDL is the standard AI planning language for the International Planning Competition

- PDDL includes ADL and STRIPS

- PDDL problems are defined using two files:
  - Domain File: Specifies predicates and action (schemas)
  - Problem File: Specifies objects, initial state, goal characterisation

Domain File

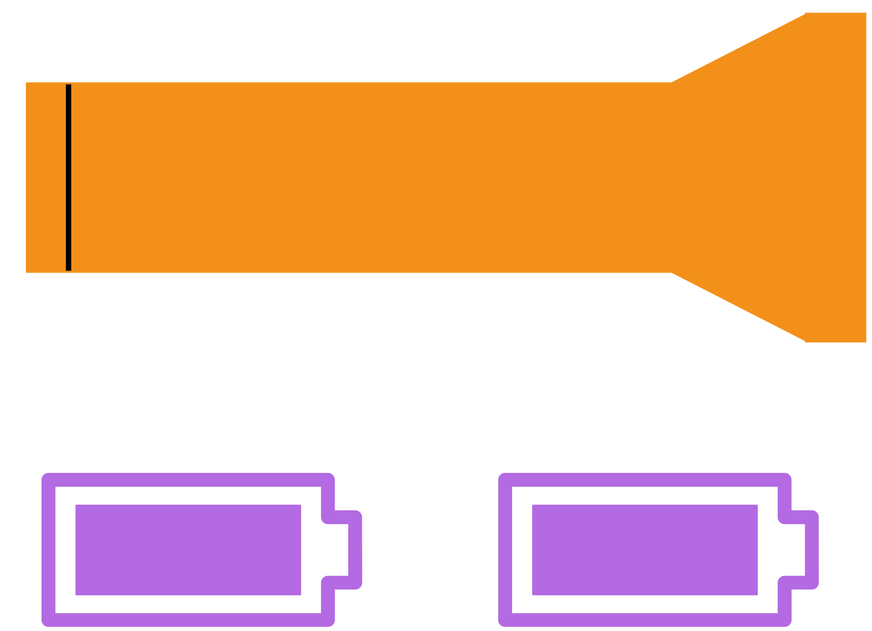
```
(define (domain domain_name)
(:requirements :strips)
(:predicates (p1 ?x)
                    ....
                    (pn ?x ?y))
(:action action_name
:parameters (?v1 ?v2 ... ?vm)
:precondition (and (p2 ?v2 ?v3)
                                ...
                                (not(pk ?v1)))
:effect (and (p1 ?v1)
                    ....))
)
(:action action_name2 ...)
```

Problem File

```
(define (problem problem_name)
(:domain domain_name)
(:objects i1,i2,...,il)
(:init  (p1 i2 ib)
            ...
        (pd i1))
(:goal  (and  (p2 i2 i6)
                   ...
              (pd i5)
)))
```

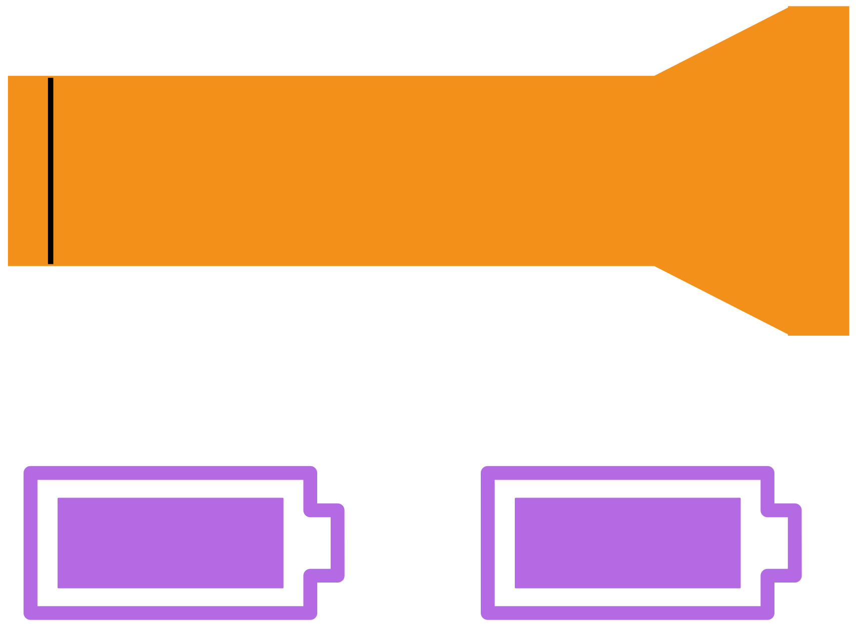
Flashlight Example

```
(define (domain flash_light)
(:requirements :strips)
(:predicates (on ?x ?y)
                   (in ?x ?y))
(:action remove_cap
:parameters (?cap ?flash)
:precondition ((on ?cap ?flash))
:effect (not (on ?cap ?flash))
))
(:action insert
:parameters (?bat  ?flash ?cap))
:precondition (and (not(in ?bat,?flash))
                             (not(on ?cap,?flash)))
:effect (in ?bat,?flash))
```



Flashlight Example

```
(define (problem flash_light_1)
(:domain flash_light)
(:objects cap, flash,bat1,bat2)
(:init  (on cap flash)
)
(:goal  (and  (in bat1 flash)
              (in bat2 flash)
              (on cap flash)
)))
```



Planning as a Search Problem

- Discrete state (defined by symbols)

$$x \in X$$

- Discrete actions (defined by preconditions)

$$u \in U(x)$$

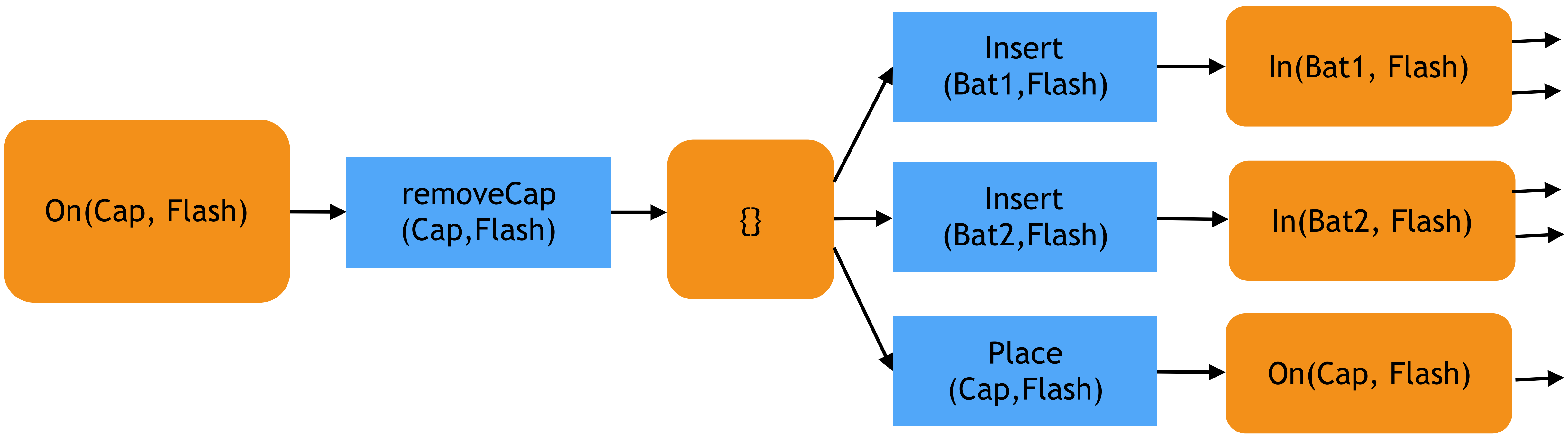
- State transition function (defined by effects)

$$f(x, u) = x'$$

- Treat the planning problem as a discrete search problem

Forward Search (Progression)

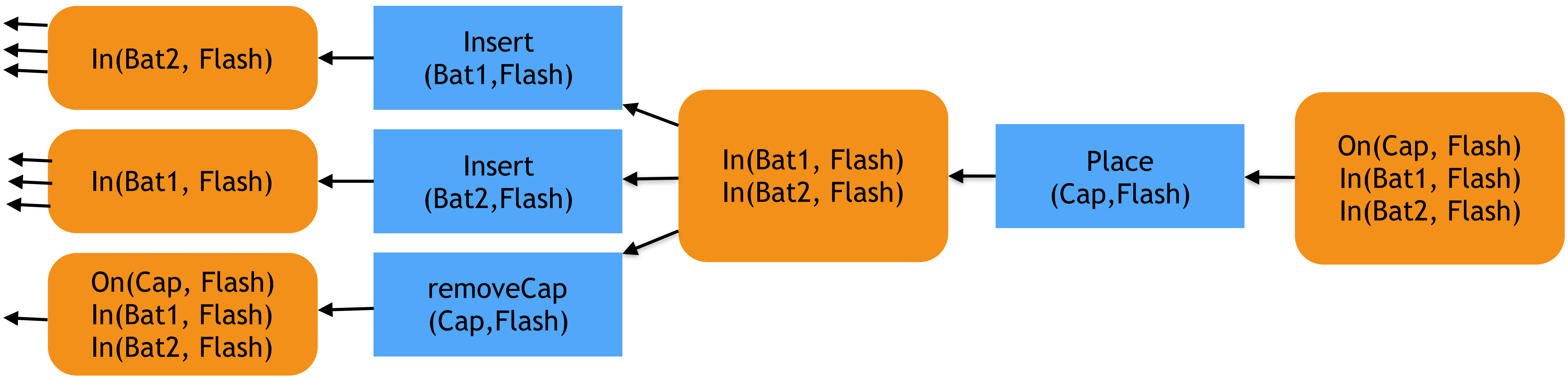
- Start with initial state and search actions forward



- Can have massive branching factor given many actions
- May try many irrelevant actions in search of goal
- Needs a good heuristic to guide action selection

Backward Search (Regression)

- Start with goal state and search actions backwards



- Reverse add/delete and ensure all preconditions met
- Focus on relevant actions - lower branching factor
- May need to consider set of multiple goal states
- Difficult to design heuristics

Heuristic Search

- Search can be very inefficient without some guidance
- Use heuristic to estimate distance from goal for A*
- Use relaxed problems to estimate cost-to-go
- Ignore preconditions
  - All actions applicable from any state
  - Quickly achieve goals directly
- Ignore delete list
  - Ensure that all goals and preconditions are defined as positive
  - Monotonic progress towards goal
- (Learn) heuristic for specific domains (e.g., TD-gammon)

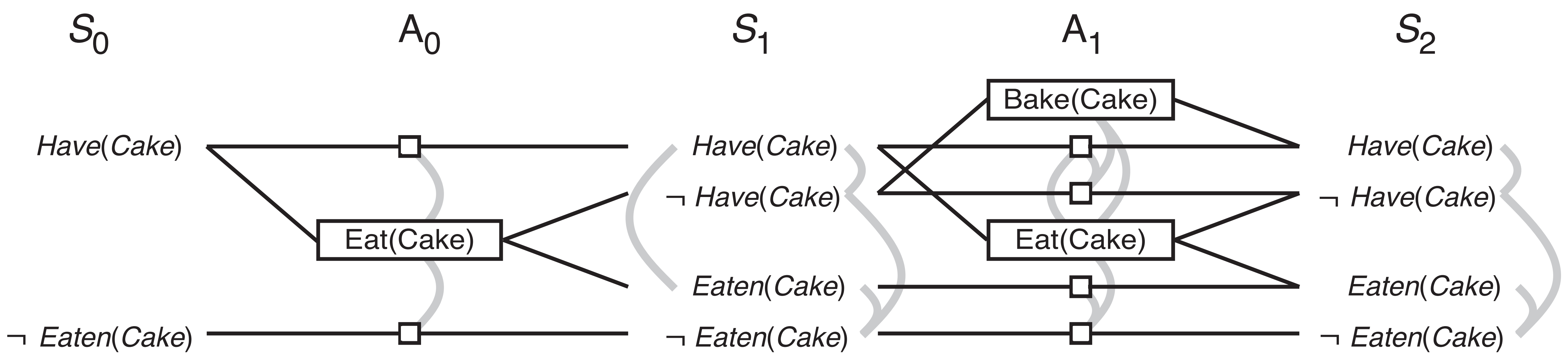
Planning Graph

- Planning graphs are powerful tools for planning
- Consider the cake problem

$$
\begin{aligned}
&Init(Have(Cake))\\
&Goal(Have(Cake) \;\wedge\; Eaten(Cake))\\
&Action(Eat(Cake)\\
&\quad \text{PRECOND: } Have(Cake)\\
&\quad \text{EFFECT: } \neg\, Have(Cake) \;\wedge\; Eaten(Cake))\\
&Action(Bake(Cake)\\
&\quad \text{PRECOND: } \neg\, Have(Cake)\\
&\quad \text{EFFECT: } Have(Cake))
\end{aligned}
$$

Planning Graph

- Planning graph for the cake problem



- Each layer indicates the achievable symbols/actions
- Curves indicate (some) mutual exclusions (mutex)
- Small squares are no-op/persistence actions
- n layers, a actions, l literals: O(n(a+l)^2) time and size

Russell and Norvig, Artificial Intelligence A Modern Approach 2010

Planning Graph

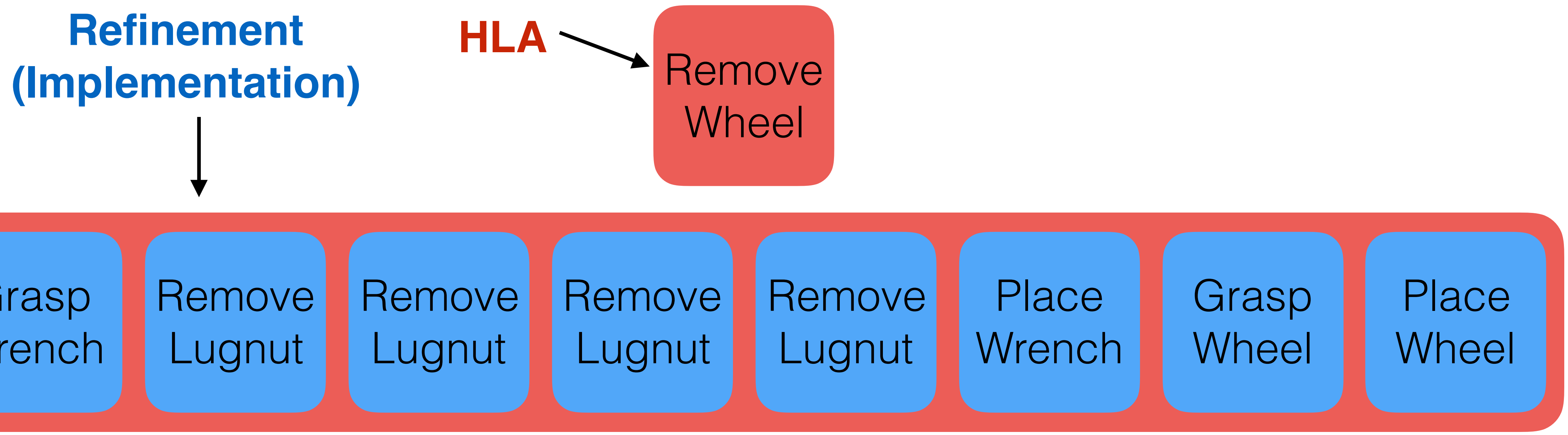
- Actions are mutex if:
  - An action negates the effect of the other (eat+persist cake)
  - An effect of one action negates precondition of the other
  - Preconditions of the actions are mutually exclusive
- State literals are mutex if:
  - One is the negation of the other
  - Each pair of actions that could achieve them are mutex
- Over time
  - Literals and actions increase monotonically
  - Mutexes decrease monotonically

Planning Graph

- Use planning graph to generate heuristics
  - Max-level
    - Max of level numbers for which each goal symbol first appears
  - Level sum
    - Sum of level numbers for which each goal symbol first appears
  - Set-level
    - Level at which all goal conditions appear without mutex
- Use planning graph to plan by incremental growth
  - Extract (regress) solution when goals reached and not mutex
  - Expand further if search does not find a valid plan

Hierarchical Task Networks (HTNs)

- Many tasks have an inherent hierarchical structure
- Can define high-level actions (HLA) for performing tasks



- HLA can contain primitive actions and other HLA
- Implementation only has primitive actions
- Allows defining domain knowledge for faster planning

Hierarchical Task Networks

- Treat HLA with one implementation as macro action
- An HLA can have multiple implementations
  - Goto airport: take car, taxi, or bus
  - Move block: push or pick-and-place
- Implementations may have different effects
  - Model the reachable set of states
  - Set of states reachable given at least one implementation
  - Reach goal when reachable set of states overlaps goal states
  - Determine exact implementation by refining plan

Semantic Attachment

- What if action not possible in some configurations?
  - e.g., obstacles, joint limits
- Use predicates with semantic attachment
  - CanGrasp(Obj, ObjPose, Grasp, ArmConfig)
  - Reachable(RobotConfig1, RobotConfig2)
- Use c-space planner to evaluate predictates during planning
  - Provides feedback from motion to symbolic task planner
- For continuous variables, e.g., Grasp, sample finite set

Grounding and Interpretation

- Ground Predicate
  - Ground predicate has constant syms assigned to it, no variables
  - e.g., At(Tire,Axle) is grounded, At(a,b) is not, At(a,b) is lifted
- Interpretation
  - Interpretation is what a symbol means in the real world
  - What does it mean to be block?
  - What does it mean for A to be on B?
- "Grounding" is often used to mean interpretation
  - Meaning may be obvious to us, but not to a robot
  - Define or learn groundings

Grounding and Interpretation

- Autonomous robot may need to interpret current state
- Grounding is often represented as a classifier
  - Object recognition
    Cat(Obj1), Screwdriver(Obj2)

  - Recognising relations
    On(A,B), Grasped(Cup,Hand)

Learning Symbols

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

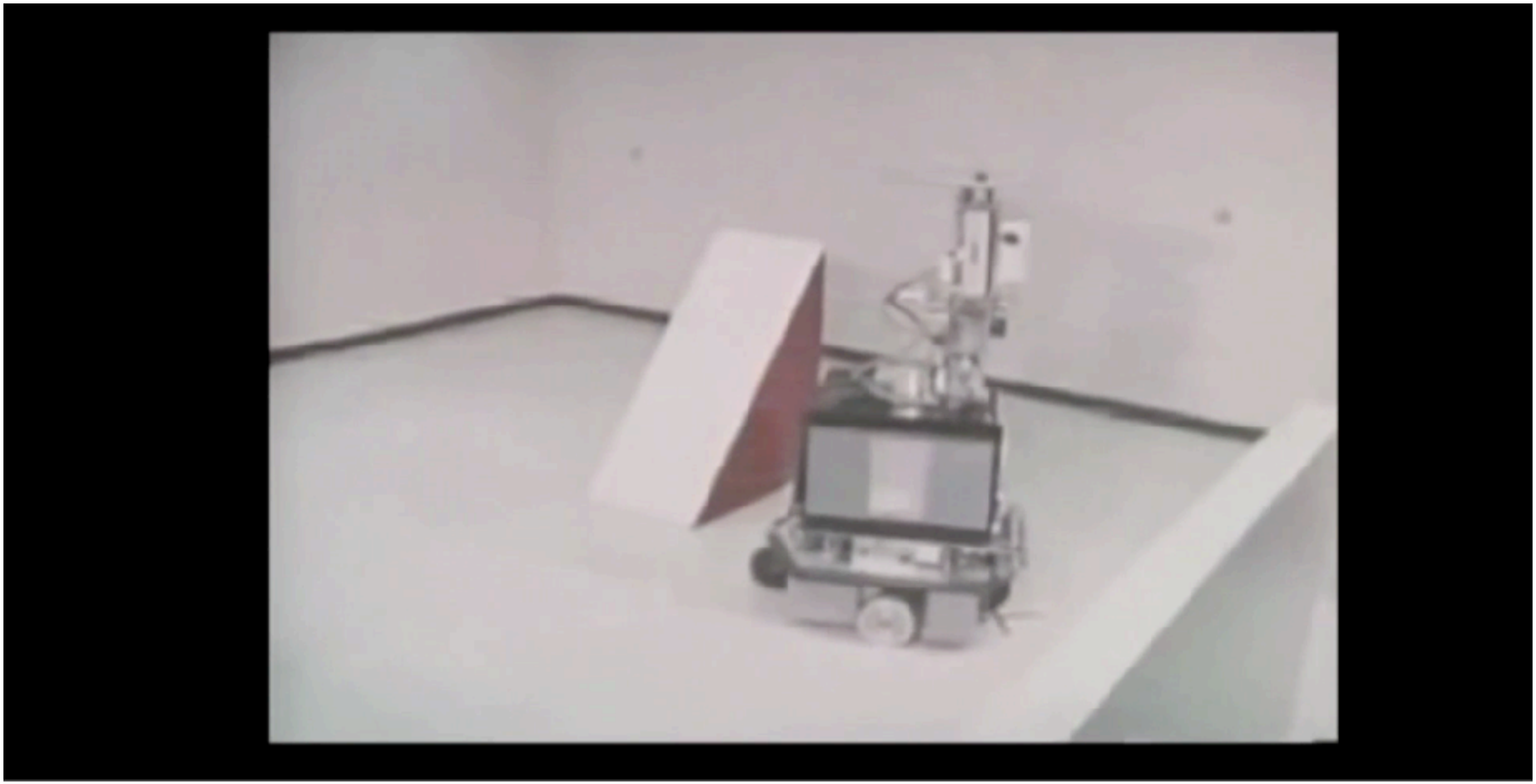
symbol1 symbol3 symbol4

Konidaris et al. Learning Symbolic Representations for High-level Planning, JAIR 2018

Monitoring and Replanning

- Robust execution requires execution monitoring
- Action monitoring:
  - Check preconditions before executing an action
- Plan monitoring:
  - Check remaining plan will still succeed
- Goal monitoring:
  - Check if there is a better set of goals to achieve
- If error occurs:
  - Replan from the current state
  - Repair plan - plan to get back onto original plan

Shakey



Shakey the Robot – SRI, youtube.com/watch?v=qXdn6ynwpiI

Questions?