Classical Planning

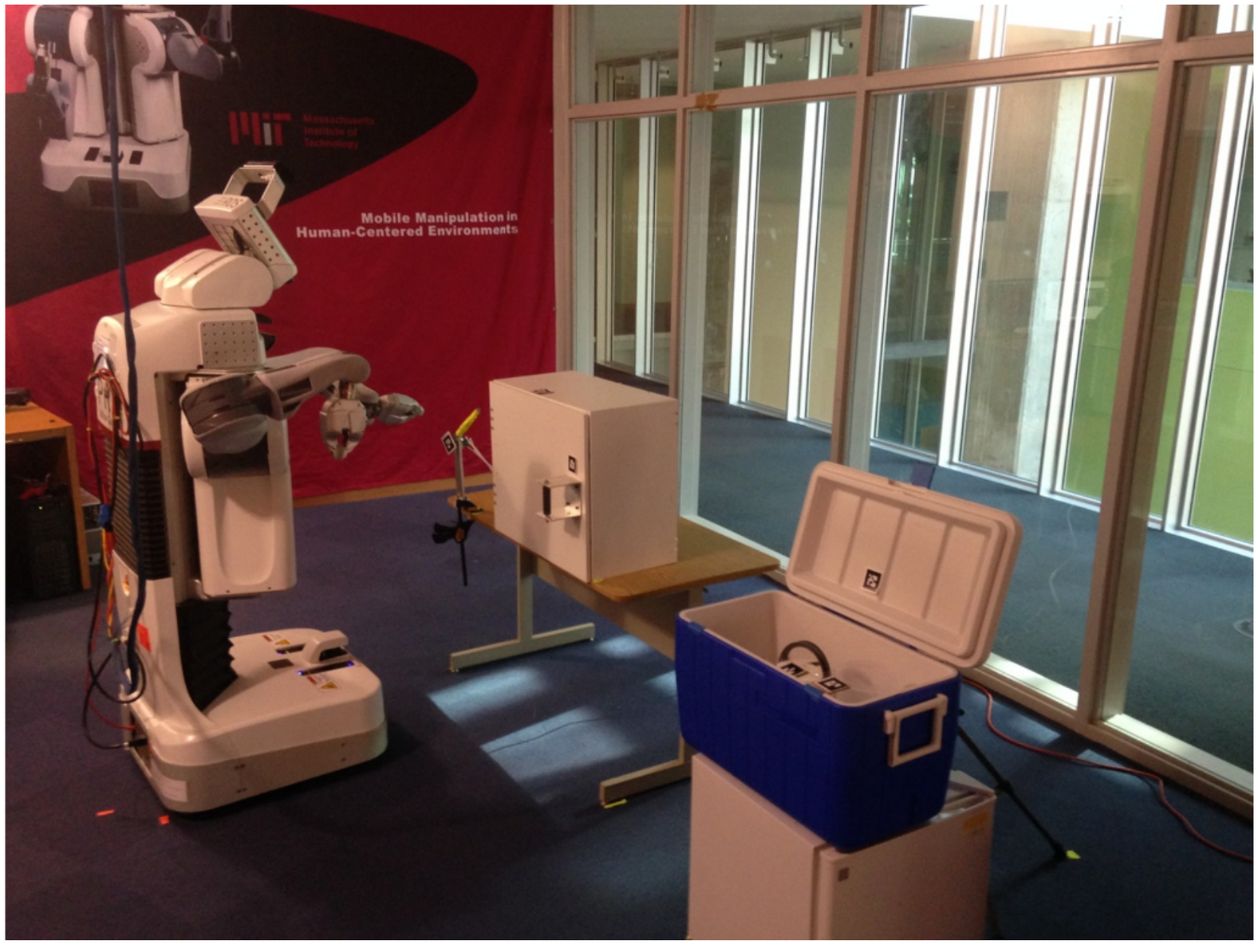
George Konidaris
gdk@cs.brown.edu

Fall 2021

Planning

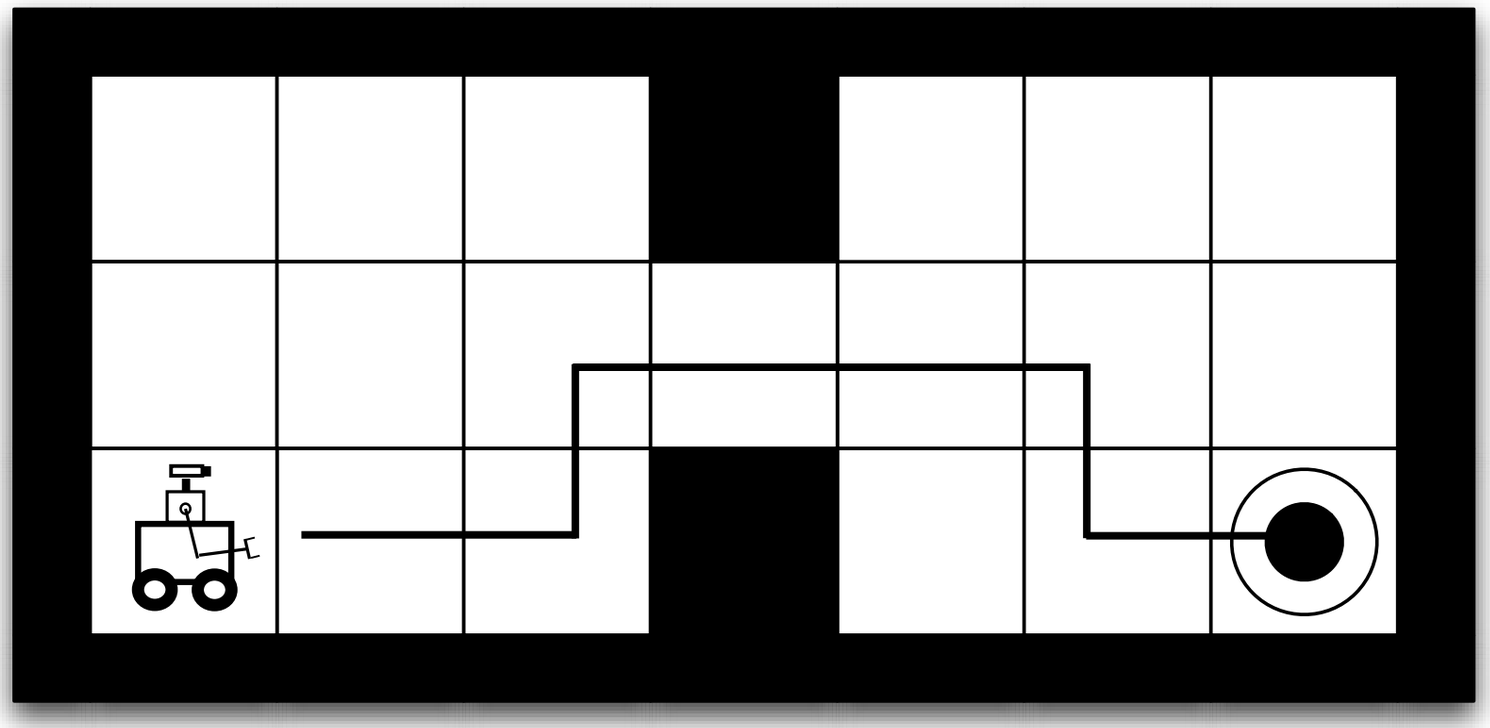
Fundamental to AI:

- Intelligence is about behavior.



The Planning Problem

Finding a sequence of actions to achieve some goal.

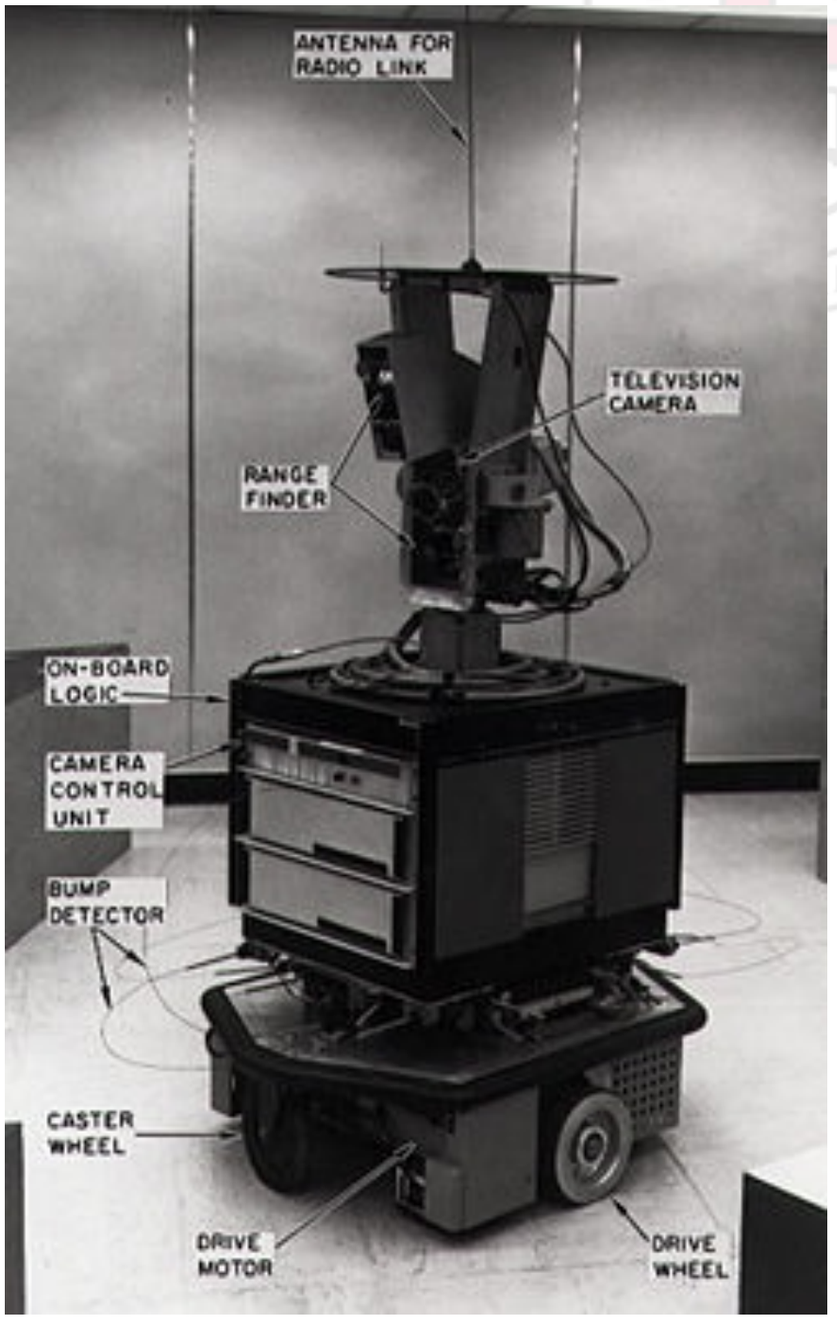


Shakey the Robot

Research project started in 1966.

Integrated:

- Computer vision.
- Planning.
- Control.
- Decision-Making.
- KRR



Classical Planning

Describe the world ***(domain)*** using logic.

Describe the ***actions*** available to the agent.
In terms of:

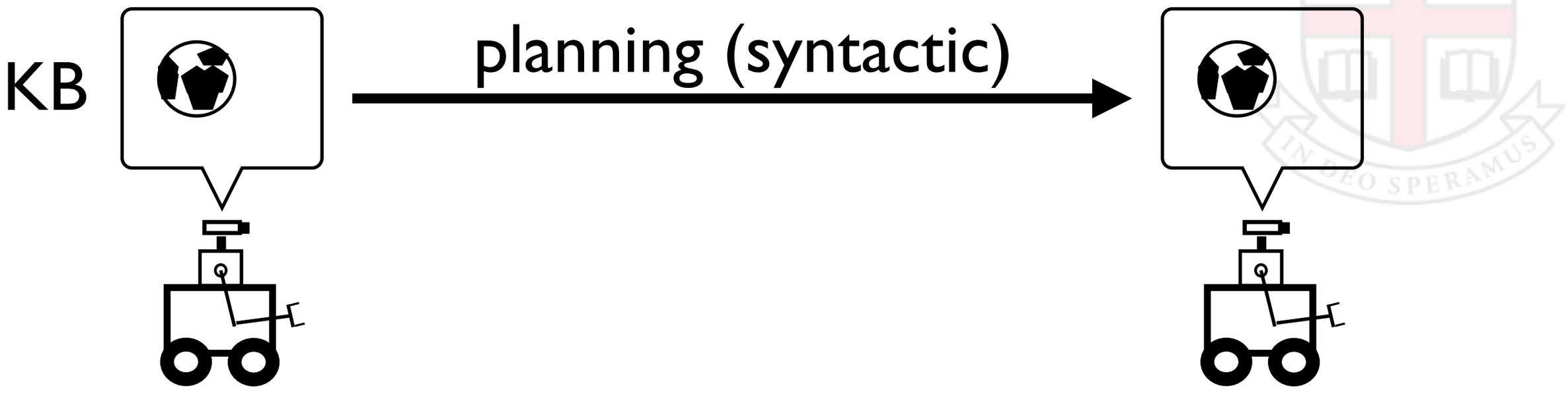
- When they can be executed.
- What happens if they are.

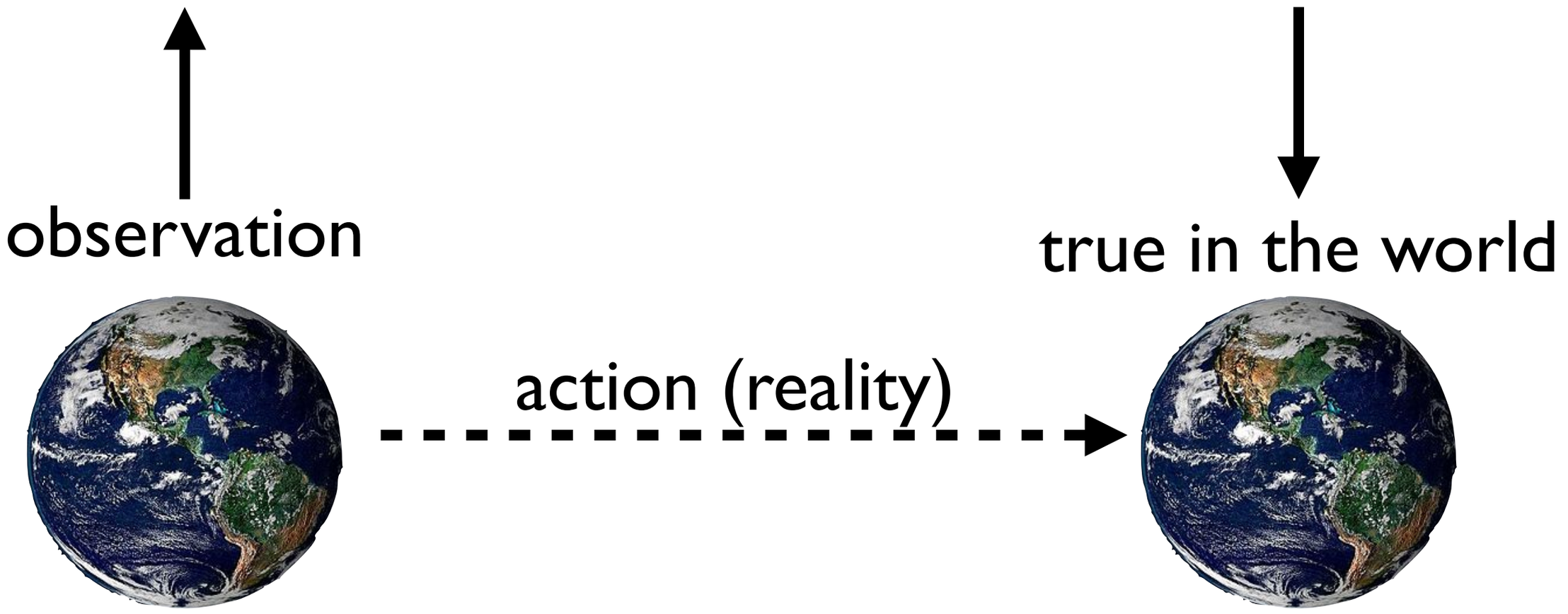
Describe the ***start state*** and ***goal***.

Task:

- Find a plan that moves the agent from start state to goal

The World and the Model





STRIPS Planning

Represent the world using a KB of **first-order logic**.

Actions can change **what is currently true**.

Describe the actions available:

- Preconditions
- Effects

must be true in KB (t)

change to KB after execution (t+1)

PDDL

Planning Domain Description Language

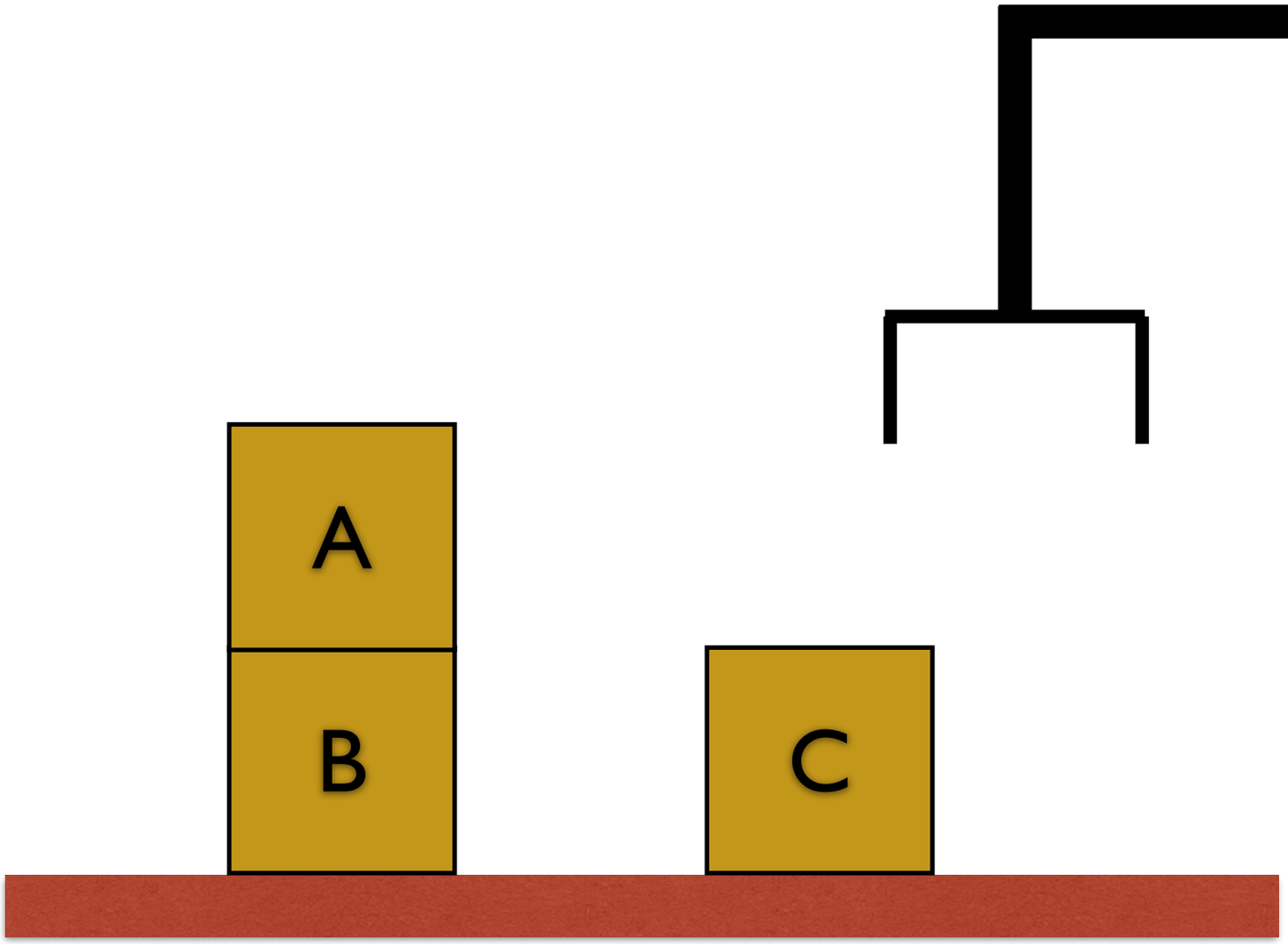
- Standard language for planning domains
- International programming competitions
- At version 3, quite complex.

Separate definitions of:

- A domain, which describes a class of tasks.
  - Predicates and operators.

- A task, which is an instance of domain.
  - Objects.
  - Start and goal states.

Examples: Blocks World



PDDL: Predicates

A predicate returns *True* or *False*, given a set of objects.

```
(define (domain blocksworld)
(:requirements :strips :equality)
(:predicates (clear ?x)
             (on-table ?x)
             (arm-empty)
             (holding ?x)
             (on ?x ?y))
```

cf. predicates in first-order logic

(example PDDL code from PDDL4J open source project)

PDDL: Operators

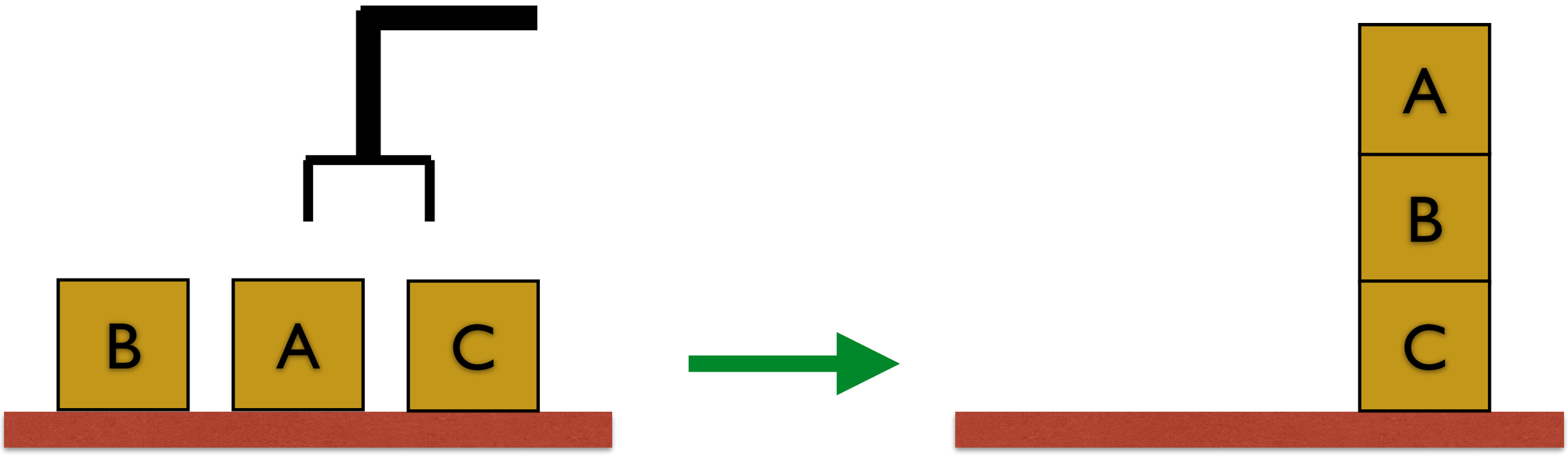
Operators:

- Name
- Parameters
- Preconditions
- Effects

```
(:action pickup
 :parameters (?ob)
 :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
 :effect (and (holding ?ob) (not (clear ?ob)) (not (on-table ?ob))
              (not (arm-empty))))
```

PDDL: A Problem

```
(define (problem pb3)
   (:domain blocksworld)
   (:objects a b c)
   (:init (on-table a) (on-table b)   (on-table c)
          (clear a)  (clear b) (clear c) (arm-empty))
   (:goal (and (on a b) (on b c))))
```



PDDL: States

As in HMMs, state describes the configuration of the world *at a moment in time*.

Conjunction of *positive literal predicates*.

- (on-table a)
- (on-table b)
- (on-table c)
- (clear a)
- (clear b)
- (clear c)
- (arm-empty)

Closed World Assumption

Those not mentioned ***assumed to be False***.
(closed world assumption)

c.f. Knowledge base concept of a *model*.

- *Set* of models consistent with KB.
- *Unknown things are unknown!*

Why?

- Avoid inference
- No uncertainty about which actions can be executed.
- No uncertainty about goal.
- Planning is hard enough.

PDDL: Operators

```
(:action putdown
  :parameters  (?ob)
  :precondition (and (holding ?ob))
  :effect (and (clear ?ob) (arm-empty) (on-table ?ob)
               (not (holding ?ob))))
```

Note! Implicit Markov assumption.

PDDL: Goals

Conjunction of literal predicates:

- (and (on a b) (on b c))

Predicates not listed are *don't-cares*.

Each goal is thus ***a partial state expression***.

Why?

- We want to refer to a set of goal states.

PPDL: Action Execution

Start state:

(on-table a) (on-table b) (on-table c)
(clear a) (clear b) (clear c) (arm-empty)

Action: pickup(a)

- Check preconditions
- Decide to execute.
- Delete negative effects.
- Add positive effects.

```
(:action pickup
 :parameters (?ob)
 :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
 :effect (and (holding ?ob) (not (clear ?ob)) (not (on-table ?ob))
          (not (arm-empty))))
```

Next state:

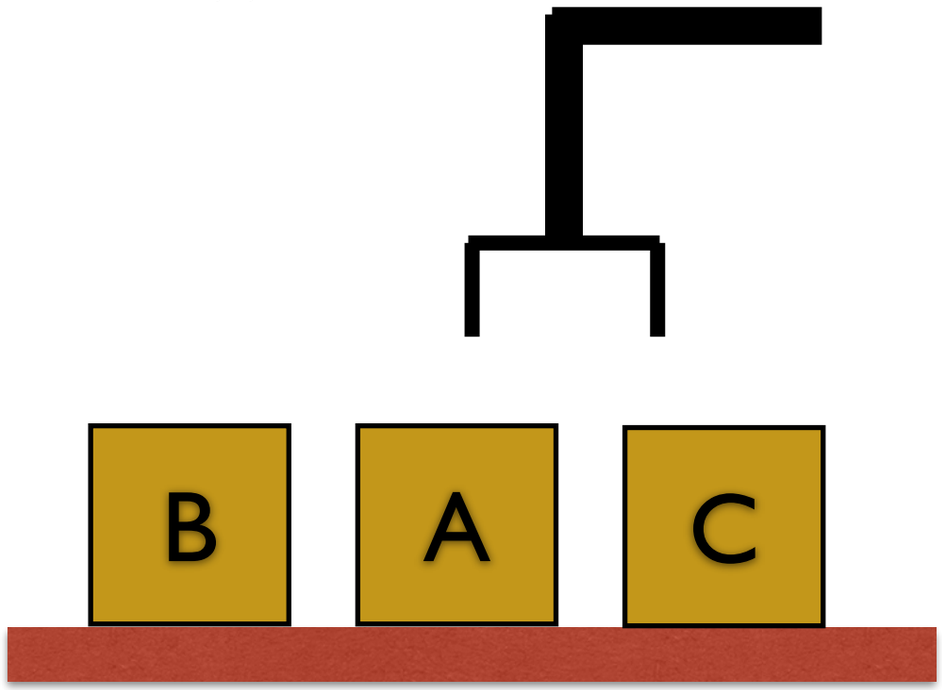
~~(on-table a)~~ (on-table b) (on-table c)
~~(clear a)~~ (clear b) (clear c) ~~(arm-empty)~~
(holding a)

Example

State: (on-table a) (on-table b) (on-table c)
(clear a) (clear b) (clear c) (arm-empty))
Goal: (and (on a b) (on b c))

```
(:action pickup
  :parameters (?ob)
  :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
  :effect (and (holding ?ob) (not (clear ?ob))
               (not (on-table ?ob))
               (not (arm-empty))))
```

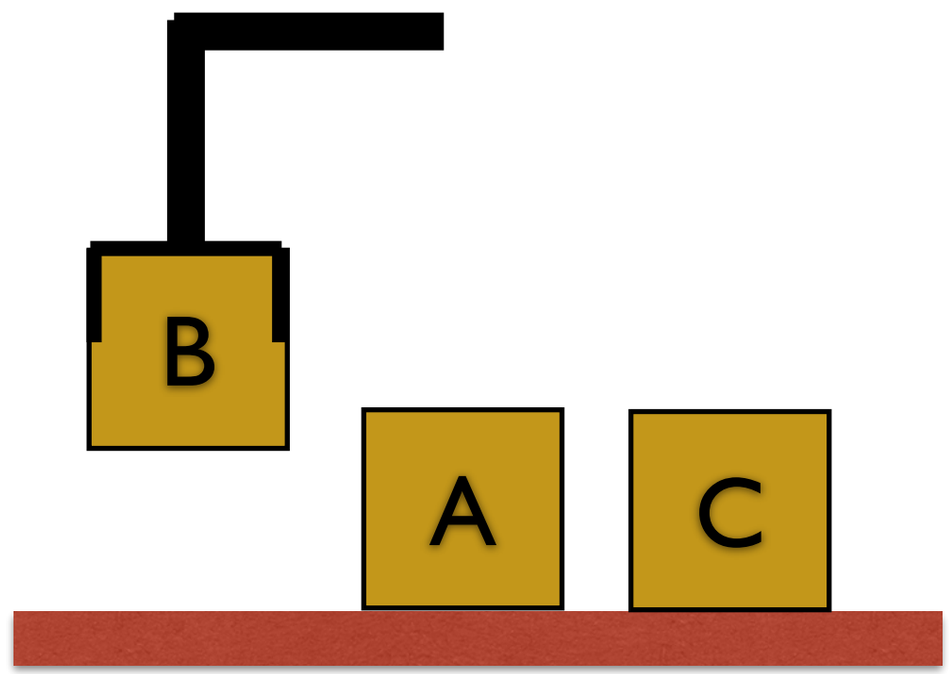
pickup(b)



Example

State: (on-table a) ~~(on-table b)~~ (on-table c)
(clear a) ~~(clear b)~~ (clear c) ~~(arm-empty)~~ (holding b))
Goal: (and (on a b) (on b c))

after pickup(b) …

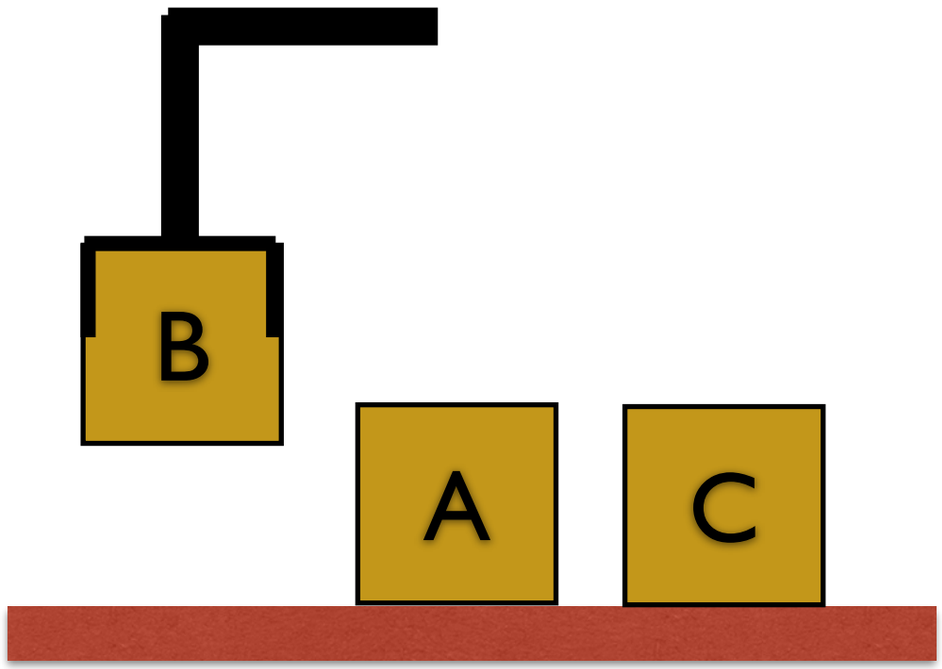


Example

State: (on-table a) (on-table c)
(clear a) (clear c) (holding b))
Goal: (and (on a b) (on b c))

```
(:action stack
  :parameters  (?ob ?underob)
  :precondition (and  (clear ?underob) (holding ?ob))
  :effect (and (arm-empty) (clear ?ob) (on ?ob ?underob)
               (not (clear ?underob)) (not (holding ?ob))))
```

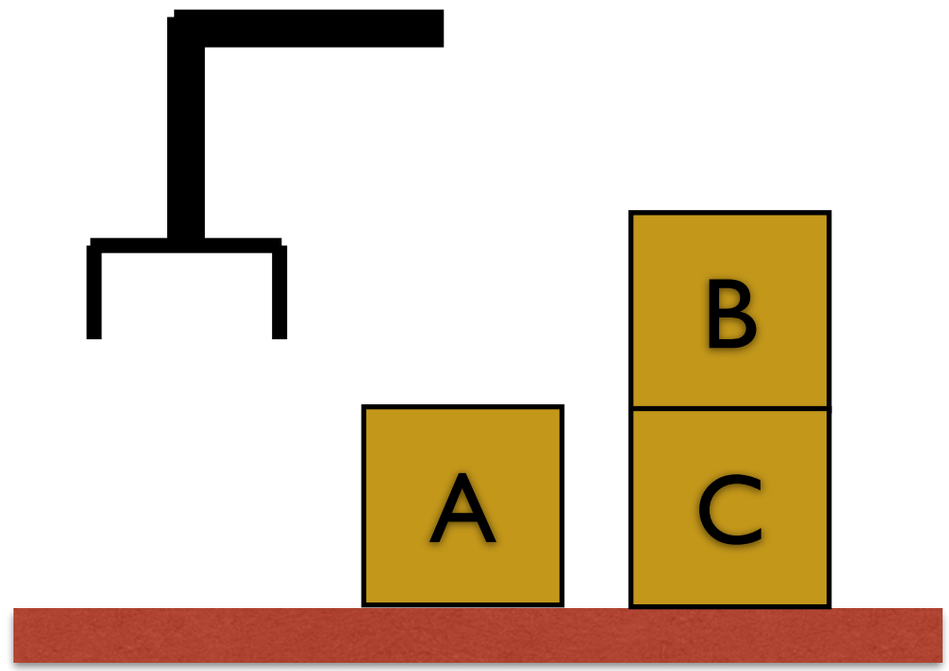
stack(b, c)



Example

State: (on-table a) (on-table c)
(clear a) ~~(clear c) (holding b)~~
(arm-empty) (clear b) (on b, c))
Goal: (and (on a b) (on b c))

after stack(b, c) …



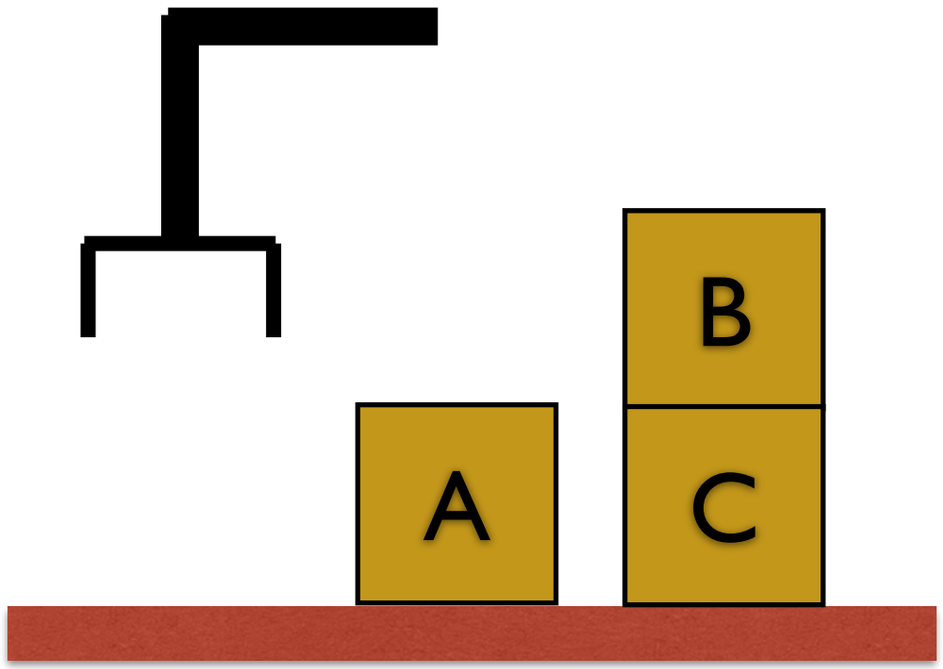
Example

State: (on-table a) (on-table c)
(clear a) (arm-empty) (clear b) (on b, c))

Goal: (and (on a b) (on b c))

```
(:action pickup
  :parameters (?ob)
  :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
  :effect (and (holding ?ob) (not (clear ?ob)) (not (on-table ?ob))
               (not (arm-empty))))
```

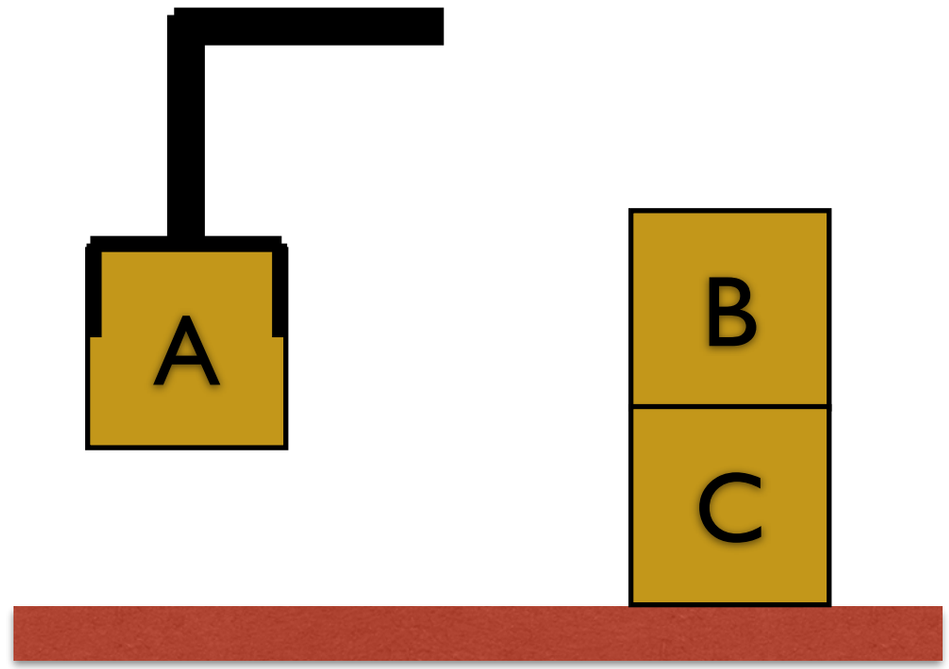
pickup(a)



Example

State: ~~(on-table a)~~ (on-table c)
~~(clear a) (arm-empty)~~ (clear b) (on b, c) (holding a))
Goal: (and (on a b) (on b c))

after pickup(a) …



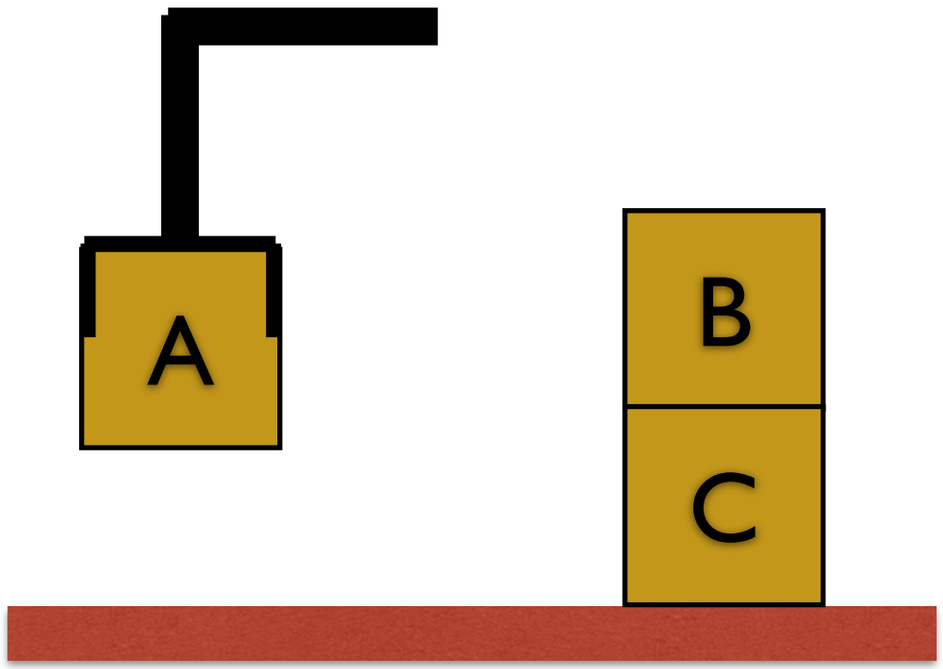
Example

State: (on-table c)
(on b, c) (clear b) (holding a))

Goal: (and (on a b) (on b c))

```
(:action stack
  :parameters  (?ob ?underob)
  :precondition (and  (clear ?underob) (holding ?ob))
  :effect (and (arm-empty) (clear ?ob) (on ?ob ?underob)
               (not (clear ?underob)) (not (holding ?ob))))
```

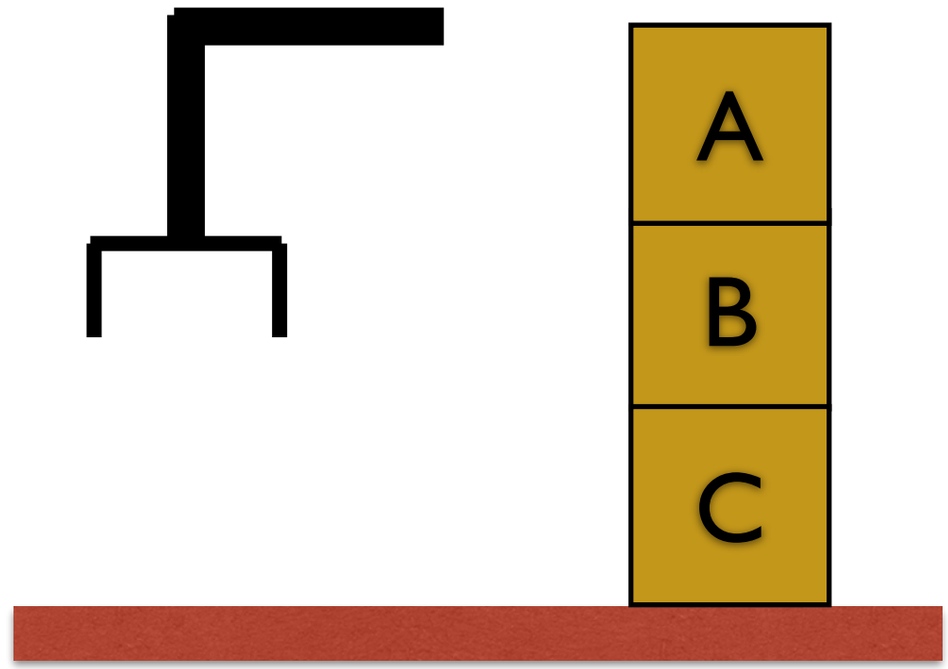
stack(a, b)



Example

State: (on-table c)
(on a b) (clear b) (on b, c) (holding a))

Goal: (and (on a b) (on b c))



Formal Definition

1. A set of predicates *P*, each with p_n parameters.
2. A set of objects *O*.

3. Literal predicates *L*: set of predicates from *P* with bound parameters from *O*.

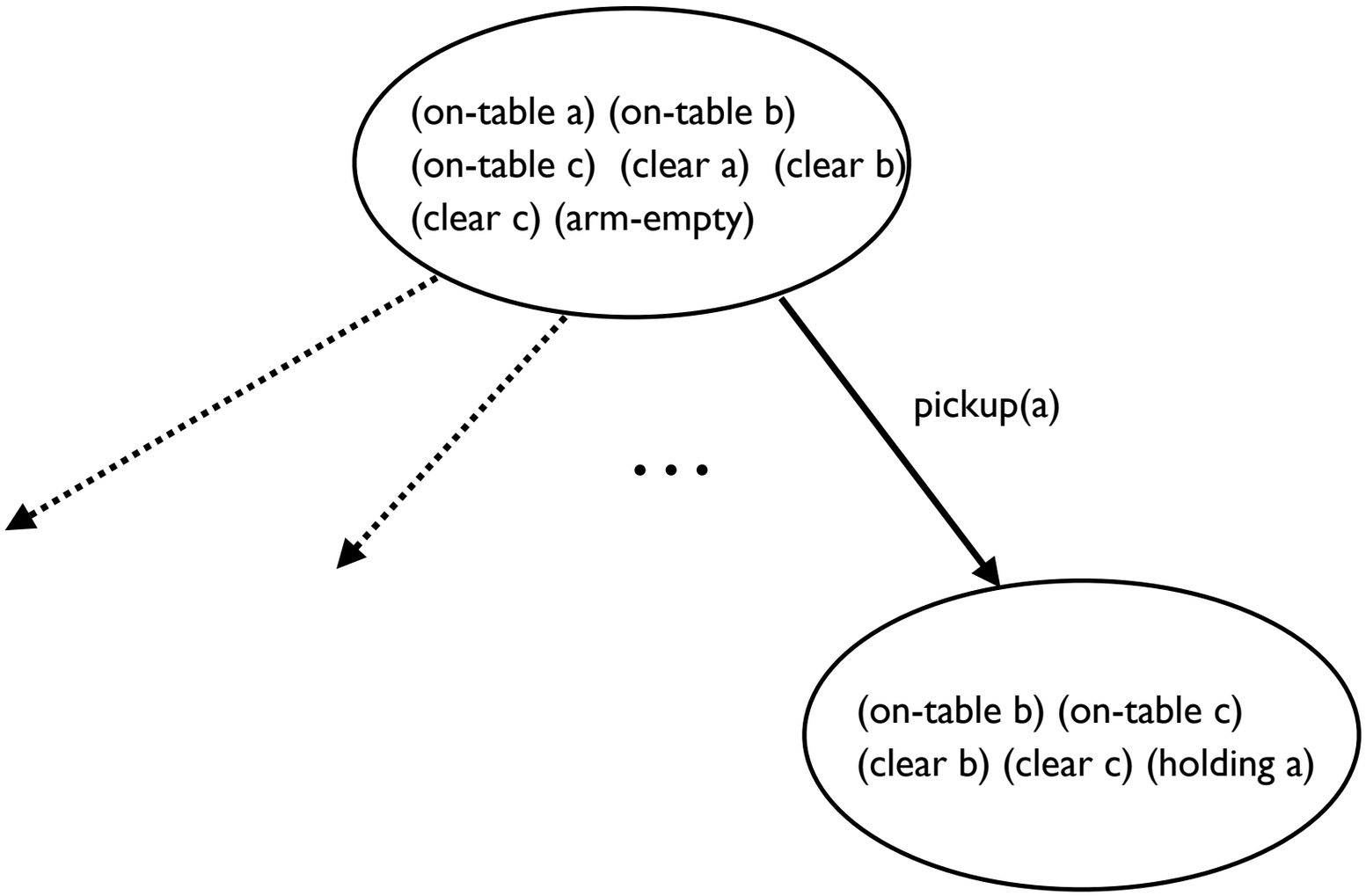
4. A state: a list of positive ground literals, $s \subseteq L$.
5. A goal test: a list of positive ground literals, $g \subseteq L$.

6. Operator List:
   - Name
   - Parameters
   - Preconditions
   - Effects

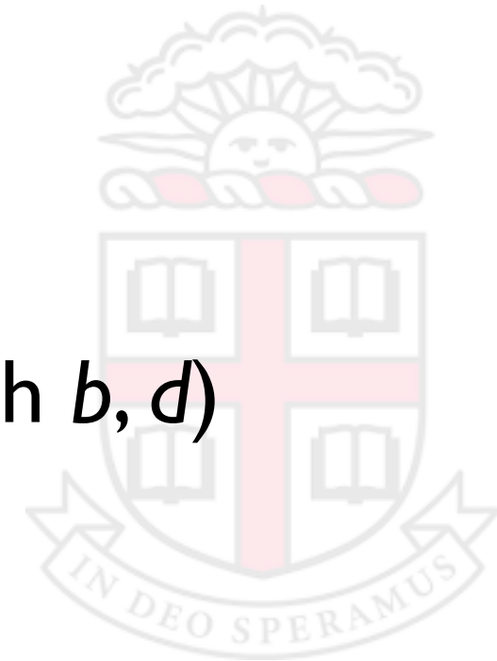
Planning

Search problem.

- Nodes are states.
- Actions are applicable operators.
- Goal expression is goal test.



Forward Search



Breadth- or depth-first search typically hopeless (high *b*, *d*)
We *must* use informed search.

The problem has a lot of known structure:

- States are conjunctions of predicates.
- We know the goal predicates.
- We know the predicates deleted and added by actions.

Major approach to solving planning problems:

- *Use this knowledge to automatically construct a domain-specific heuristic.*

General Strategy

Relaxation

- Make the problem easier
- Compute distances in easier problem
- Use distances as a heuristic to the hard problem.

FF planner (major breakthrough, circa 2000)

- Relax problem by deleting negative effects
- Actually solve relaxed problem using a planner

(:action pickup
  :parameters (?ob)
  :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
  :effect (and (holding ?ob) ~~(not (clear ?ob)) (not (on-table ?ob))~~
~~(not (arm-empty))~~)))

FFPlan

Why is the problem with deleted negative effects easier?

Recall!

Goal

- Conjunction of positive literals.

Actions

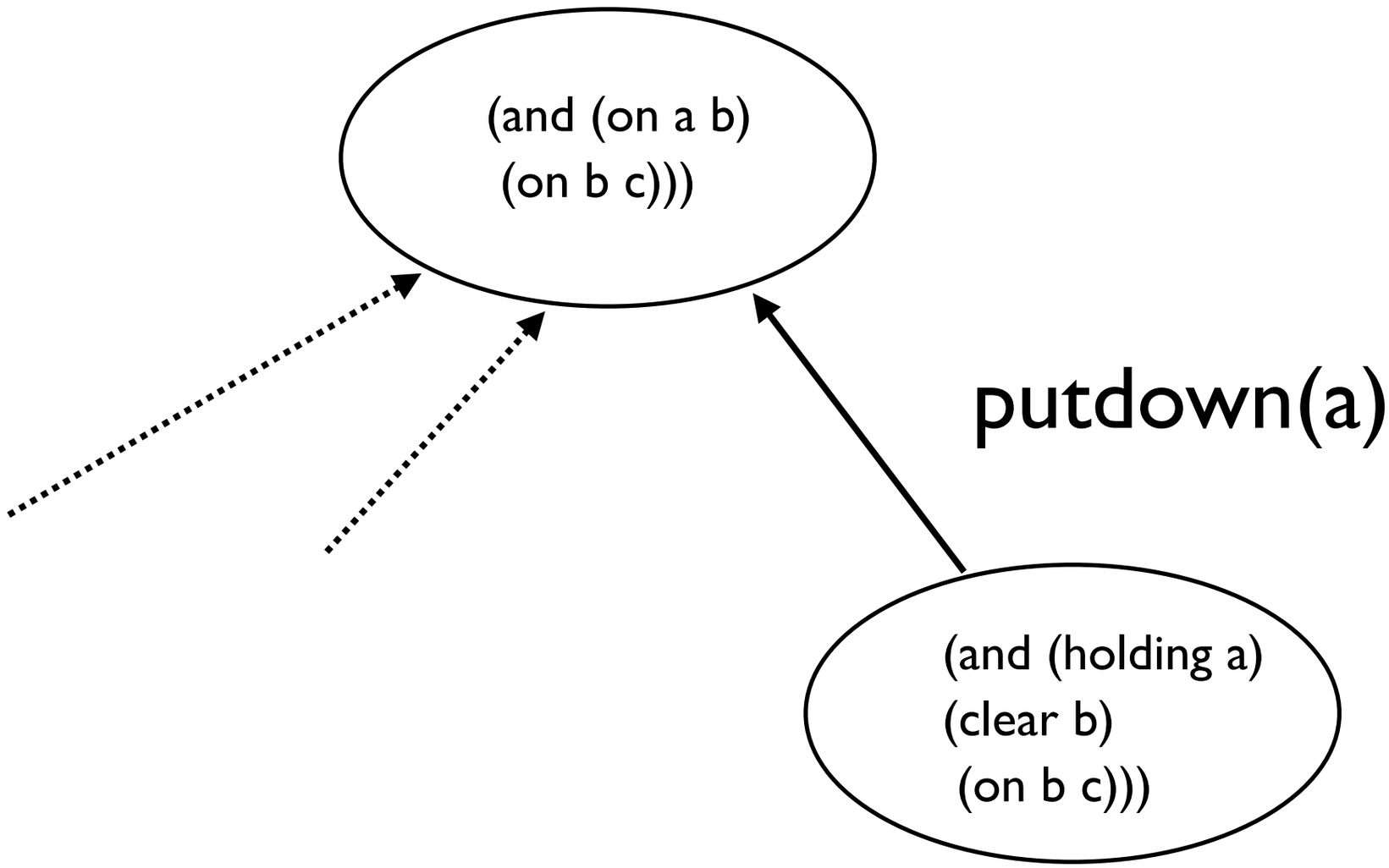
- Preconditions (conjunction of positive literals)
- Effects (adds and deletes)

- Each action execution monotonically adds applicable actions.
- Grounded actions need only be executed once.
- Progress towards goal expression monotonic.

Alternative Approach

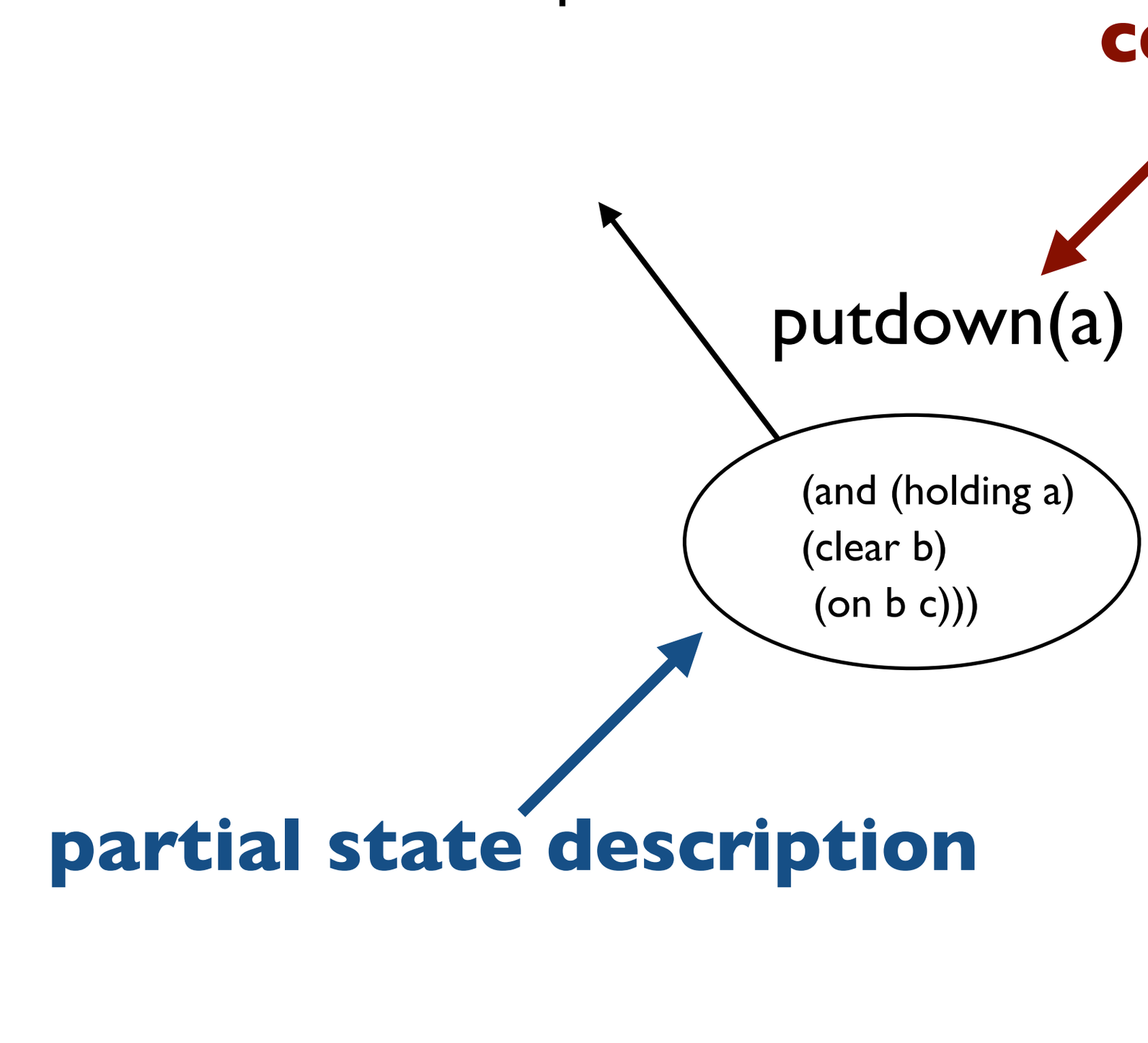
Regression Planning

- Start at the goal (partial state)
- Regress backwards



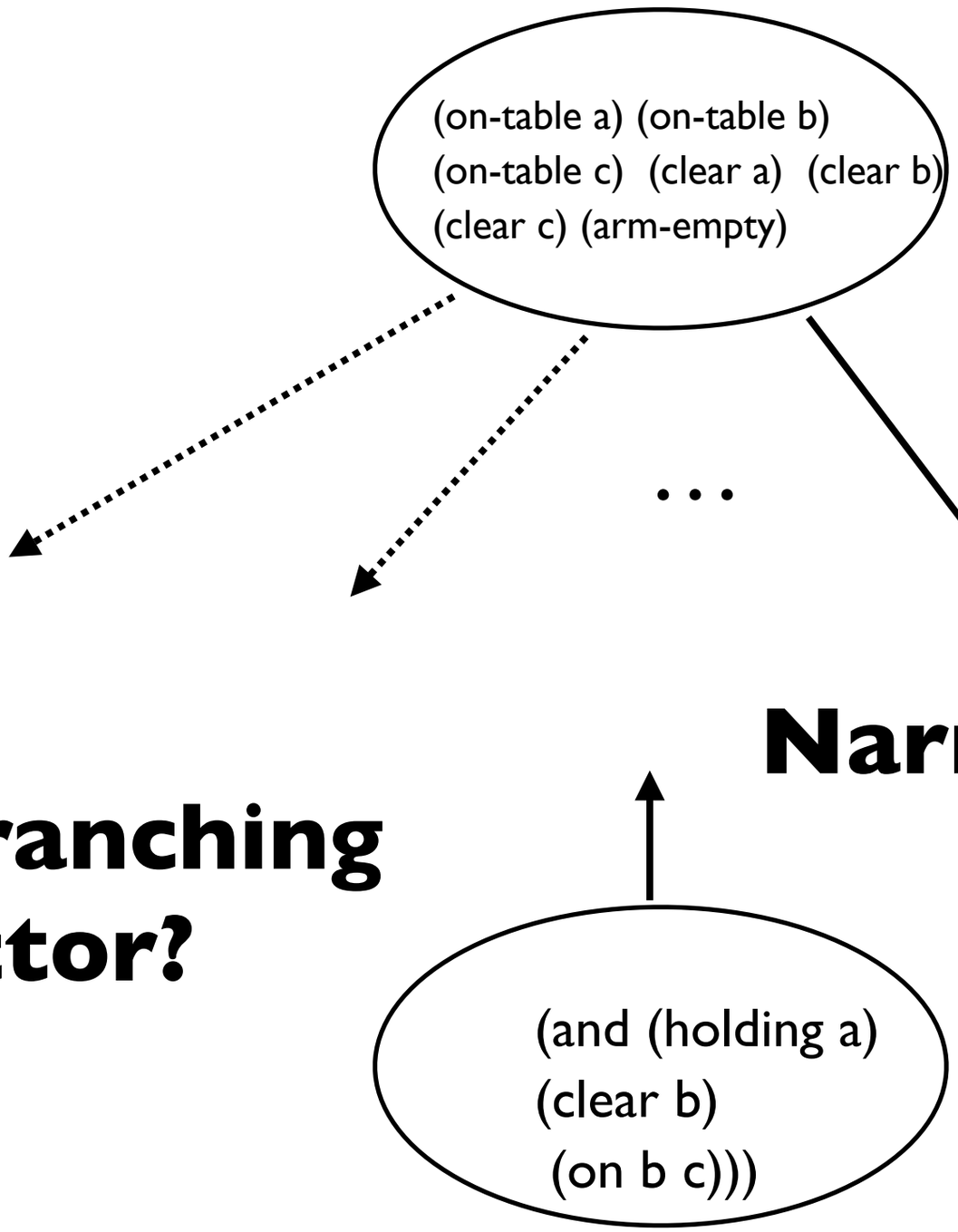
Regression Planning

What must we compute?



Regression Planning

Why do we expect this to work?



Specific

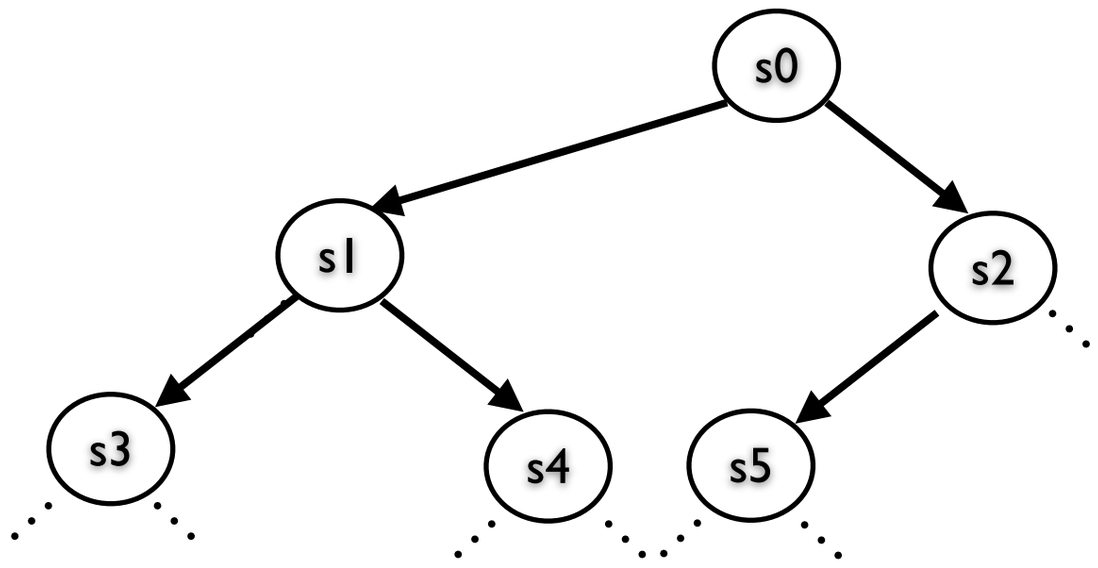
High branching factor

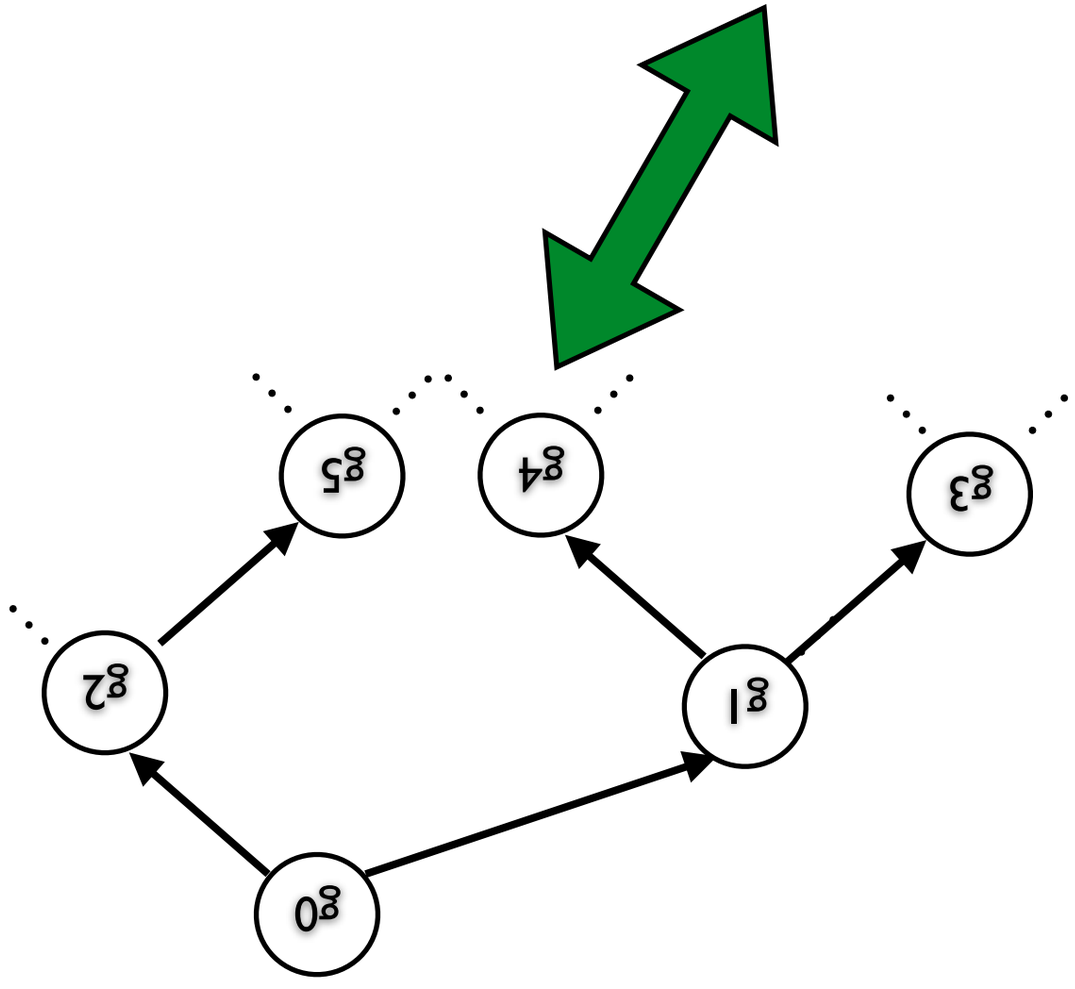
Narrow solution path

Low branching factor?

Generic

Bidirectional Search



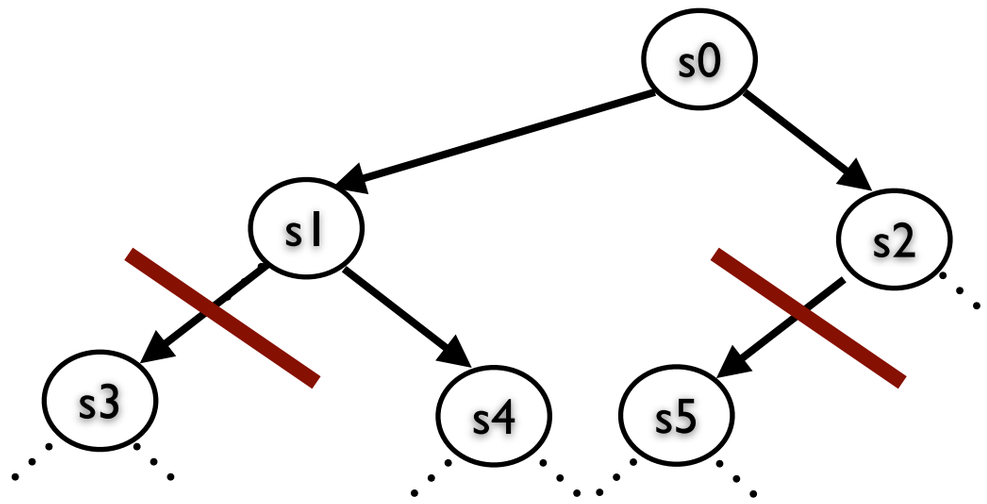


Exploiting Expert Knowledge

Often, domain expertise can be used to make planning more efficient.

One approach: *control rules*.

- Hand-written rules.
- Prune some node expansions.
- Effectively decrease branching factor.
- E.g., never move a goal block once placed.



Some progress on learning these automatically (e.g, PRODIGY)

Exploiting Domain Knowledge

Another approach: *specify partial plans*.

For example:

- Grasping a door handle always followed by turning it, then opening the door.

This can be written as a "macro-action".

- A new operator composed of old operators.
- Aim: reduce minimum solution depth.

Logical extreme: hierarchical task network.

- Specify the solution as a hierarchy of partly specified tasks.
- Planner's role is just to fill in the details.

Planning Competitions

Competitions held every few years

- Int. Conf. Automation and Planning
- Problems described in PDDL

IPC quality SCORE

	Tetris	Barman	Cave	Childsnack	Citycar	Floortile	Hiking	Maintenance	Openstacks	Parking	Thoughtful	Transport	Visitall	GED	Total
ibacop2	4,02	16,38	7,00	14,98	6,95	18,23	18,04	16,71	5,15	5,28	15,75	7,01	13,32	17,40	166,21
ibacop	6,28	16,27	7,00	15,29	7,32	15,18	18,65	16,81	3,58	1,74	13,76	9,94	14,18	16,73	162,73
mercury	14,38	13,94	3,00	0,00	3,99	2,00	16,46	5,06	19,69	15,64	0,00	20,00	19,88	19,01	153,04
miplan	7,46	16,54	7,00	18,22	4,69	4,10	18,14	16,62	9,07	11,08	11,18	0,00	8,18	17,72	150,00
jasper	9,52	19,78	8,00	0,00	8,89	2,00	17,19	9,28	17,28	12,91	0,00	7,59	15,18	17,27	144,89
uniform	11,52	17,83	7,00	1,23	12,74	1,53	17,01	9,36	10,83	9,74	0,00	9,25	19,56	15,65	143,25
cedalion	3,20	16,85	7,00	0,71	7,71	7,98	18,74	14,15	17,08	4,29	0,00	5,14	19,49	15,01	137,34
arvandherd	14,63	18,12	7,00	5,57	19,39	2,00	19,00	13,17	14,22	0,69	0,00	4,53	0,68	18,09	137,10
fdss-2014	9,87	11,64	7,00	1,36	5,00	2,00	17,44	16,63	17,63	10,86	0,00	4,06	8,78	15,61	127,89
dpmplan	1,80	16,57	7,00	18,46	5,82	1,97	16,28	15,07	10,15	0,00	13,18	0,00	3,41	15,79	125,50
use	3,76	15,12	0,00	0,00	1,65	9,17	8,79	15,03	15,76	3,65	0,00	6,27	14,80	13,15	107,14
nucelar	6,60	15,89	7,00	3,66	7,49	3,37	17,24	16,63	2,73	1,41	0,00	0,00	3,38	16,02	101,44
rpt	9,41	0,00	3,00	2,98	3,11	19,30	17,35	10,82	10,61	4,70	0,00	3,57	0,06	13,33	98,25
bfs-f	11,03	8,89	7,94	0,00	0,00	9,71	2,00	7,72	0,00	19,94	5,00	3,57	18,99	1,33	96,11
bifd	8,67	0,62	3,00	1,16	3,16	19,30	15,25	7,83	5,76	5,38	0,00	3,52	0,00	13,33	86,99
dae_yahsp	0,00	0,35	0,00	9,36	0,00	0,46	8,12	0,00	0,00	0,00	17,10	9,02	17,26	2,52	64,19
freelunch	3,74	0,00	3,00	17,43	0,00	7,52	1,91	15,26	9,82	0,00	0,00	0,02	2,51	0,00	61,21
yahsp3-mt	0,67	6,60	0,00	0,00	0,00	1,22	3,81	0,00	0,00	1,37	14,42	10,74	10,73	8,94	58,50
yahsp3	0,54	1,33	0,00	0,00	0,00	1,09	4,80	0,00	0,00	0,00	6,93	8,03	17,08	8,31	48,11
planets	0,00	0,00	5,97	0,00	3,96	1,00	0,77	0,00	0,00	0,00	13,25	0,00	0,00	0,00	24,95

2014 (deterministic)

Extensions - Time and Resources

What if there are temporal constraints in our problem?

- Actions take different amounts of time to execute.
- Preconditions depend on time.
- We can choose to *wait* for a period of time.

What if we are planning using resources?

- E.g., budget, raw material.
- Limited resources.
- Resources have levels, are exchangeable.

How to model, plan?

Often called *scheduling* - many real-life problems.

Time and Resources

Planning with time - notion of an interval.

- $[t_0, t_1]$
- Relationships between intervals
- Predicates hold during intervals
- Operators are parameterized by intervals
  - Durative action
  - Often allow for simultaneous actions

For resources:

- Fixed number and level of resources to start
- Actions require, and may produce, resource levels.
- Reusable vs. consumable
- Not typically an interval - usually a real number.

Time and Resources

Same principle for planning

- Search problem
- Nodes as state, operators

But much harder:

- Simultaneous actions
- Real-valued values
- Interval algebras

Solution is a schedule, not a plan.

DART

Planner used by the US military for logistics.

“Introduced in 1991, DART had by 1995 offset the monetary equivalent of all funds DARPA had channeled into AI research for the previous 30 years combined.”

“Directly following its launch, DART solved several logistical nightmares, saving the military millions of dollars. Military planners were aware of the tremendous obstacles facing moving military assets from bases in Europe to prepared bases in Saudi Arabia, in preparation for Desert Storm. DART quickly proved its value by improving upon existing plans of the U.S. military. What surprised many observers was DART's ability to adapt plans rapidly in a crisis environment.”

(wikipedia)