Artificial Intelligence

Steve James

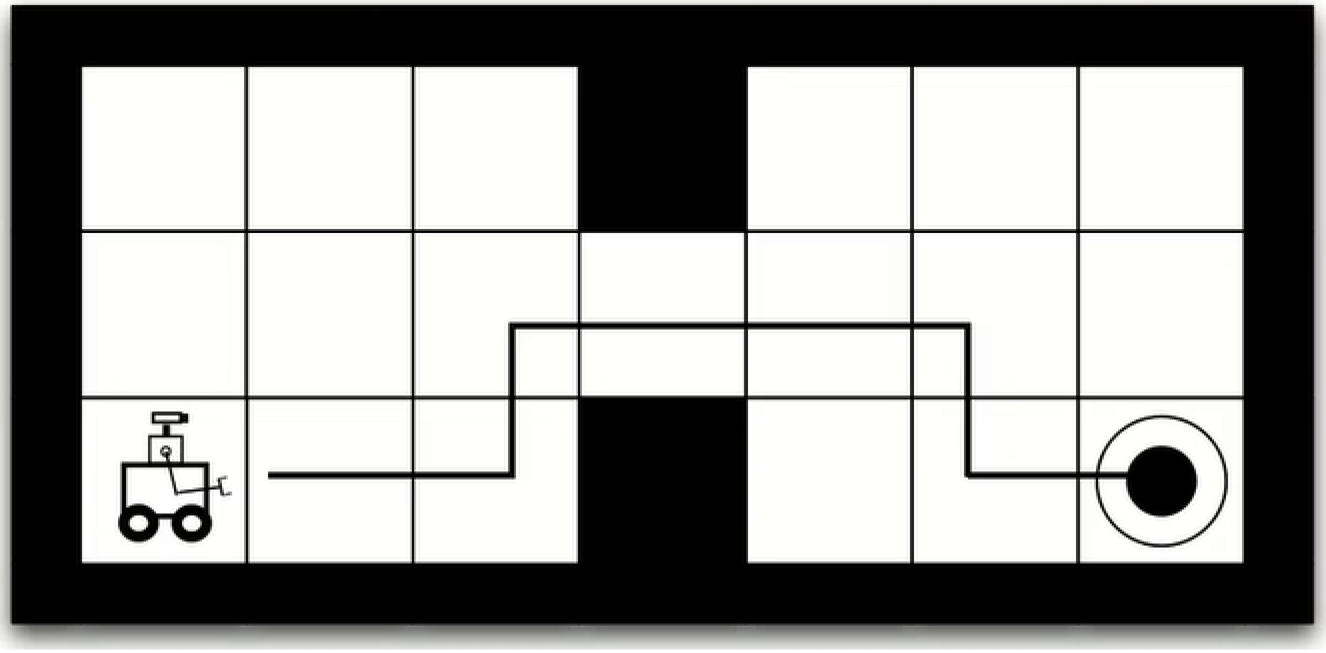
Classical Planning

Planning

- Fundamental to AI
  - Intelligence is about behaviour

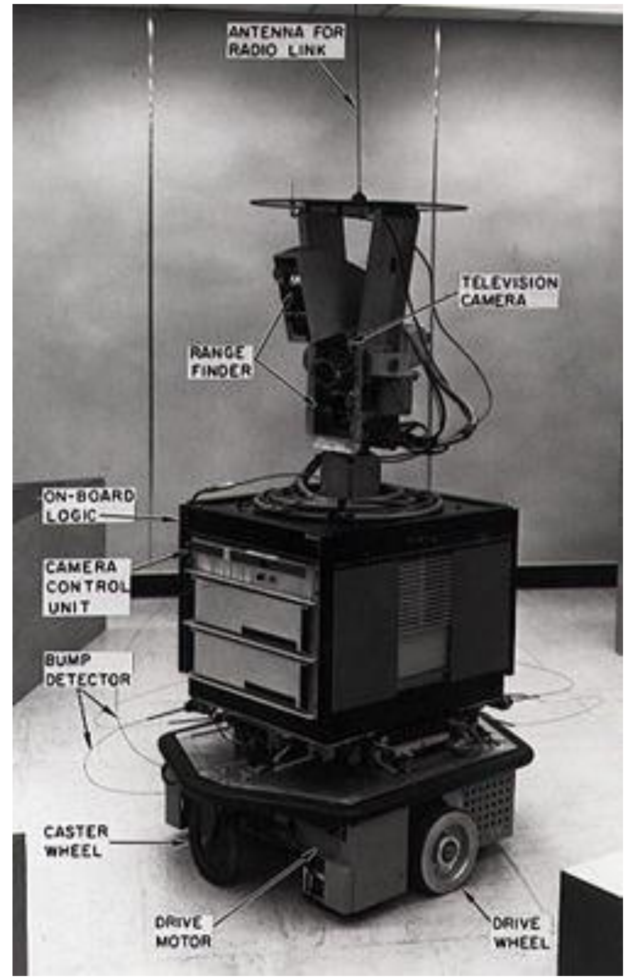
The planning problem

- Find a sequence of actions to achieve some goal



Shakey the robot

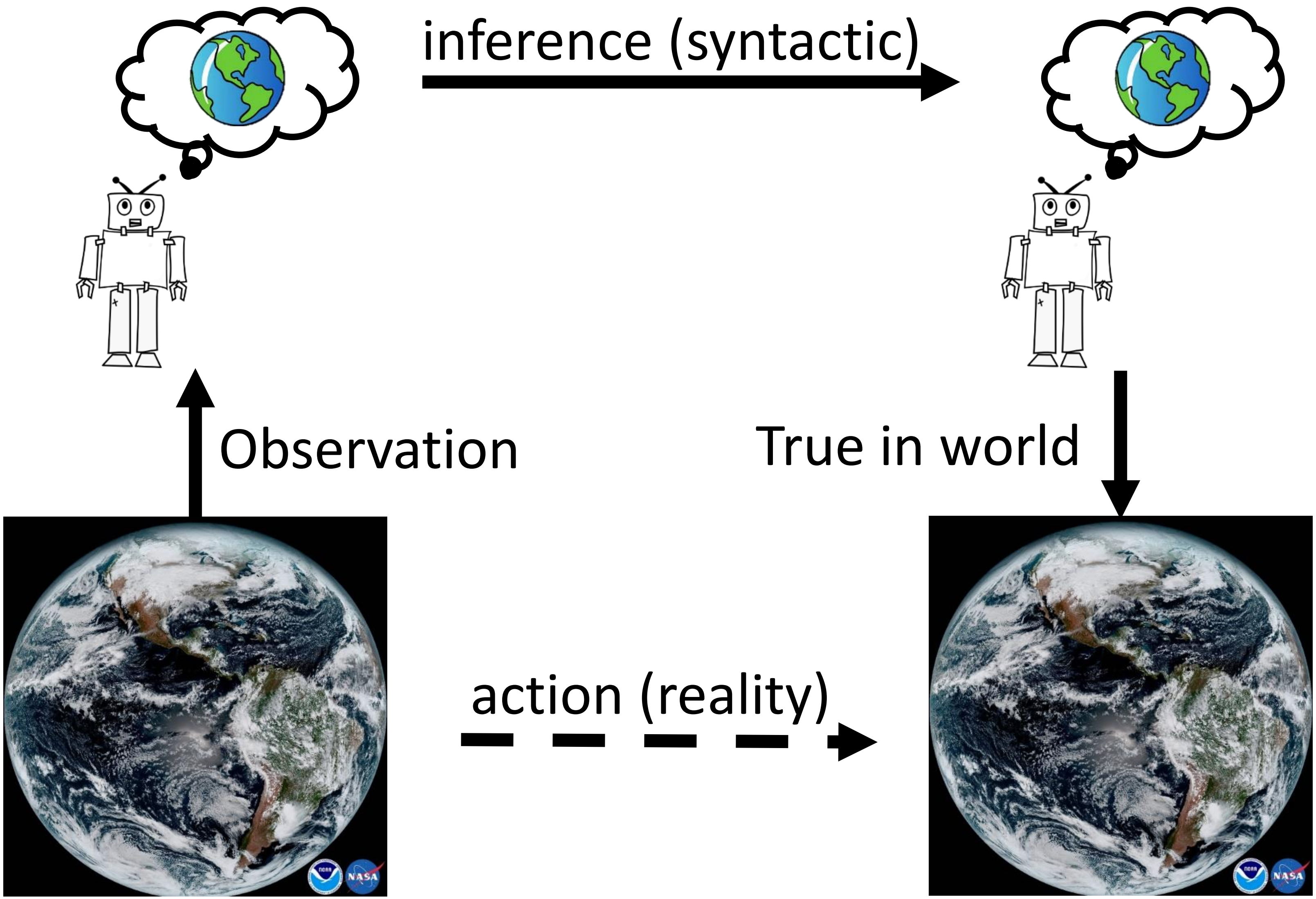
- Research project started in 1966
- Integrated:
  - Computer vision
  - Planning
  - Control
  - Decision-making
  - KRR



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 start state and goal
- Task:
  - Find a plan that moves agent from start state to goal

The world and the model



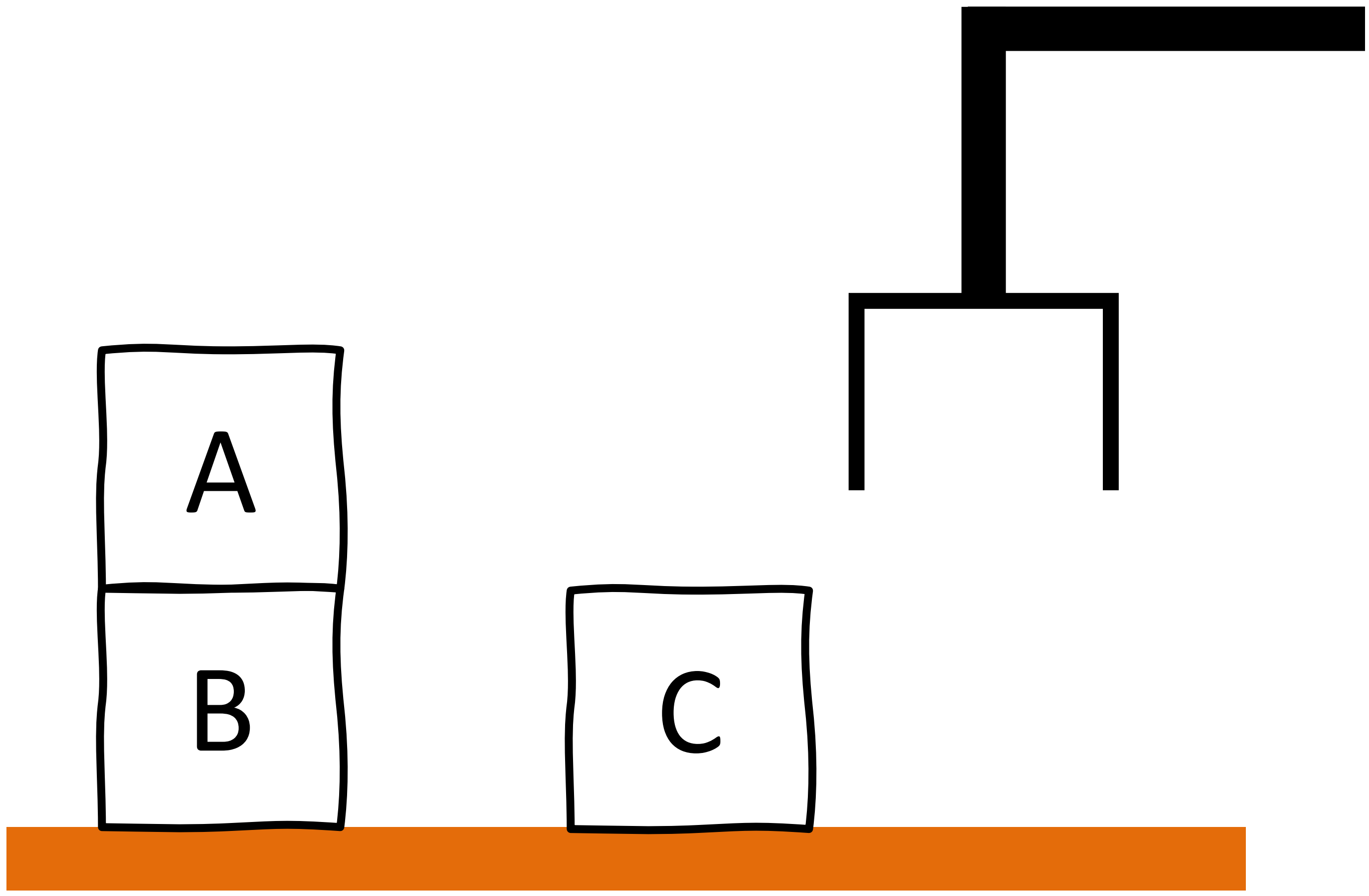
STRIPS planning

- Represent world using a KB of first-order logic
- Actions can change what is currently true
- Describe actions available:
  - Preconditions Must be true in KB (t)
  - Effects 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 that describes a class of tasks
    - Predicates and operators
  - A task that is an instance of a domain
    - Objects
    - Starts and goal states

Example: 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))
```

Predicates in first order logic

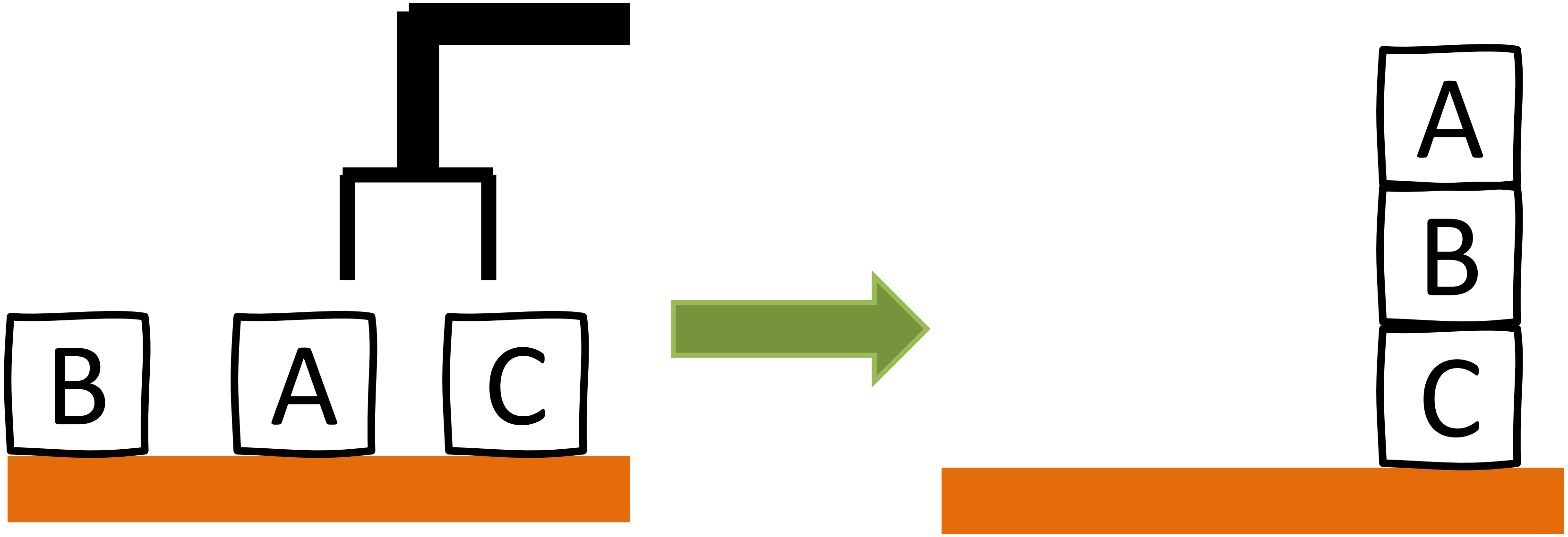
PDDL: operators

- 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: 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 configuration of 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

- No other changes to state variables
- Those not mentioned assumed to be false (closed world assumption)
- Knowledge base concept of model
  - Set of models consistent with KB
  - Unknown things are unknown!

- Why?
  - Avoid inference (because 1966)
  - 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))))
```

- 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 a partial state expression
  - Want to refer to set of goal states

PDDL: action execution

- Start state:
  - (on-table a) (on-table b)   (on-table c) (clear a) (clear b) (clear c) (arm-empty)

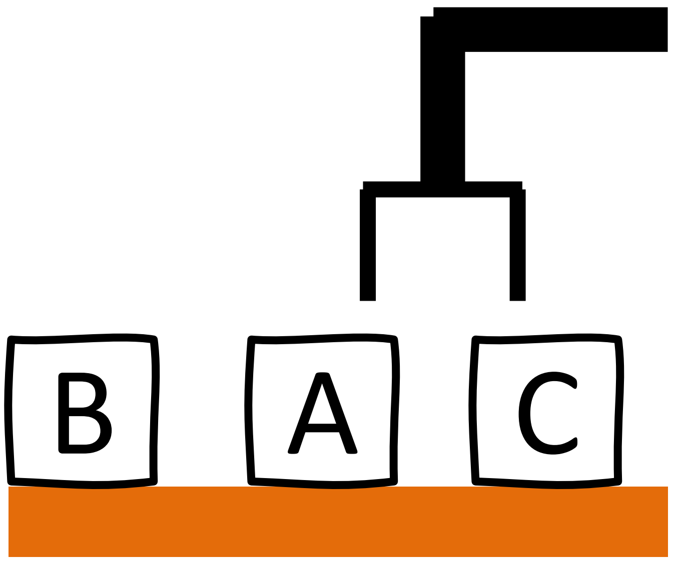
- Action: pickup(a)
  - Check preconditions
  - Decide to execute
  - Delete negative effects
  - Add positive effects

```
(: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)

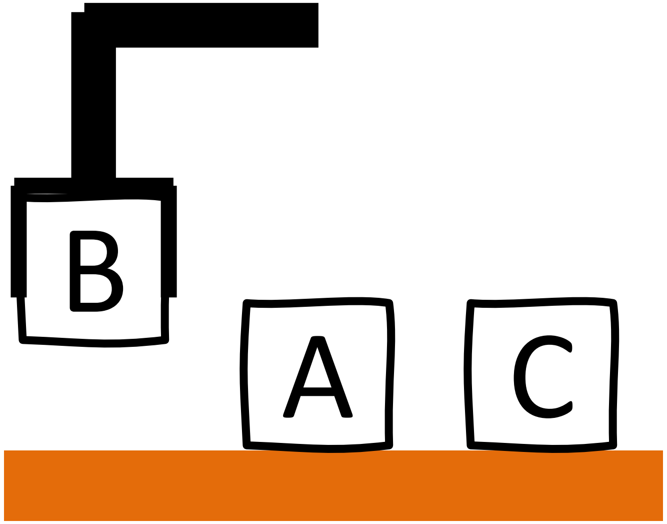
Example

- 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))`
- pickup(b)

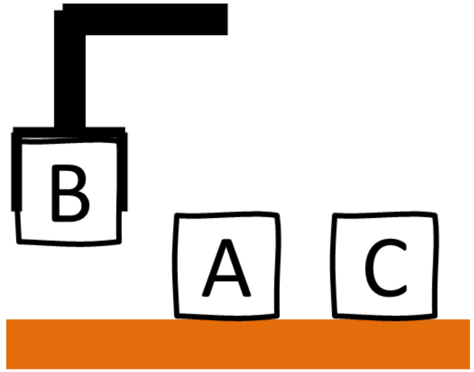


Example

- 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



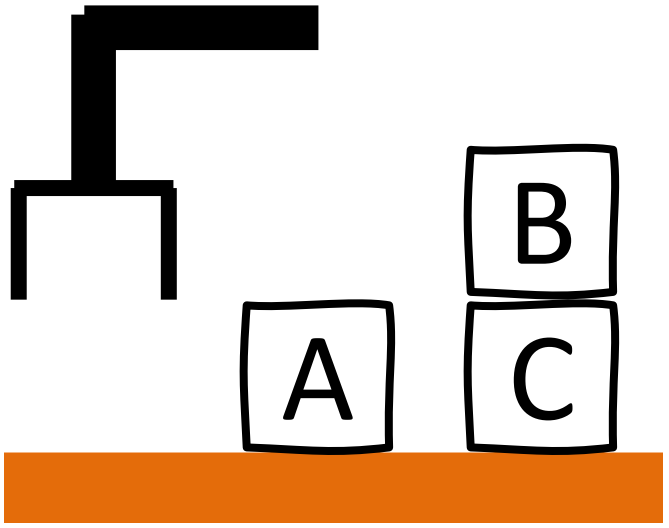
- State:
  - (on-table a) (on-table c) (clear a) (clear c) (holding b)
- Goal:
  - (and (on a b) (on b c))

```
(: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))))
```

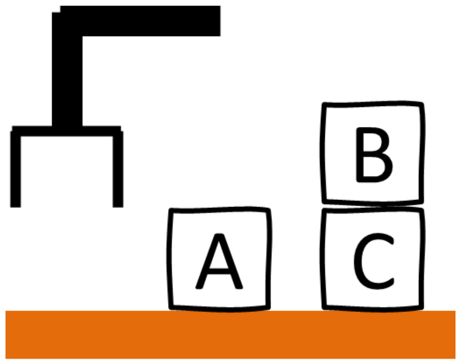
- stack(b, c)

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))

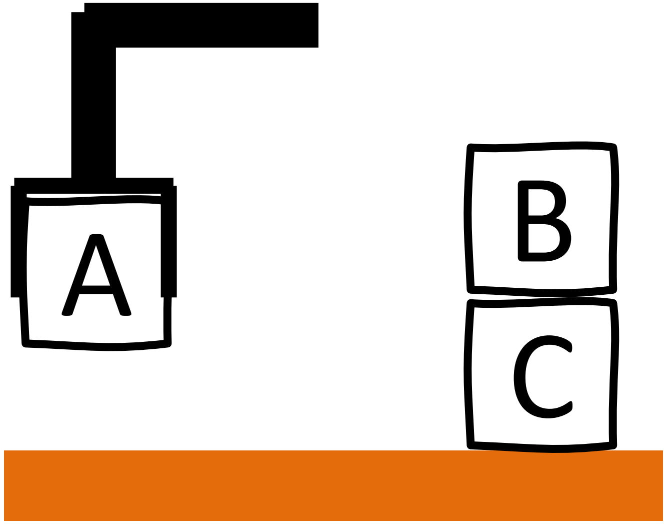
```
(:action pickup
  :parameters (?ob)
  :precondition (and (clear ?ob) (on-table ?ob) (arm-
empty))
  :effect (and (holding ?ob) (not (clear ?ob)) (not
(on-table ?ob)) (not (arm-empty))))
```

- pickup(a)

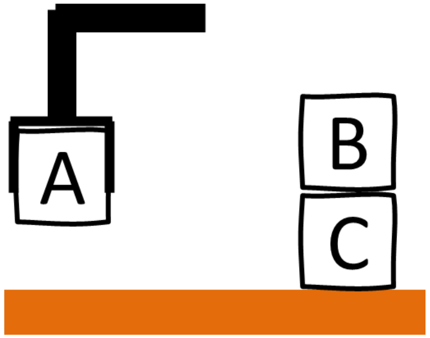
Example

- 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



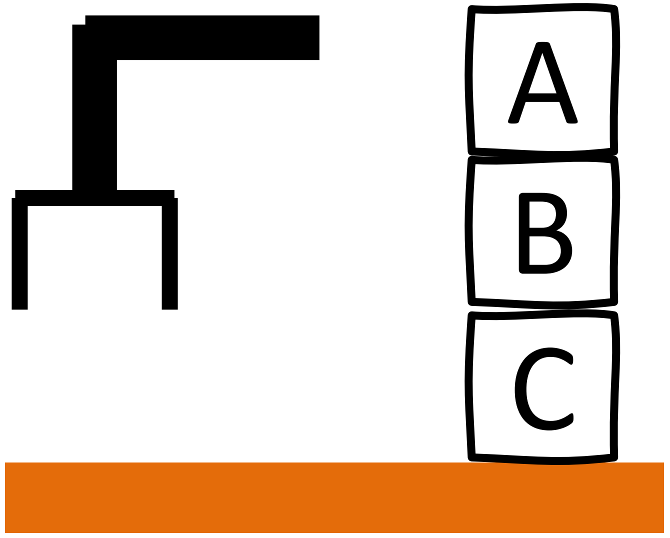
- State:
  - (on-table c) (clear b) (on b c) (holding a)
- Goal:
  - (and (on a b) (on b c))

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



Formal definition

- Set of predicates P with parameters p_n
- Set of objects O

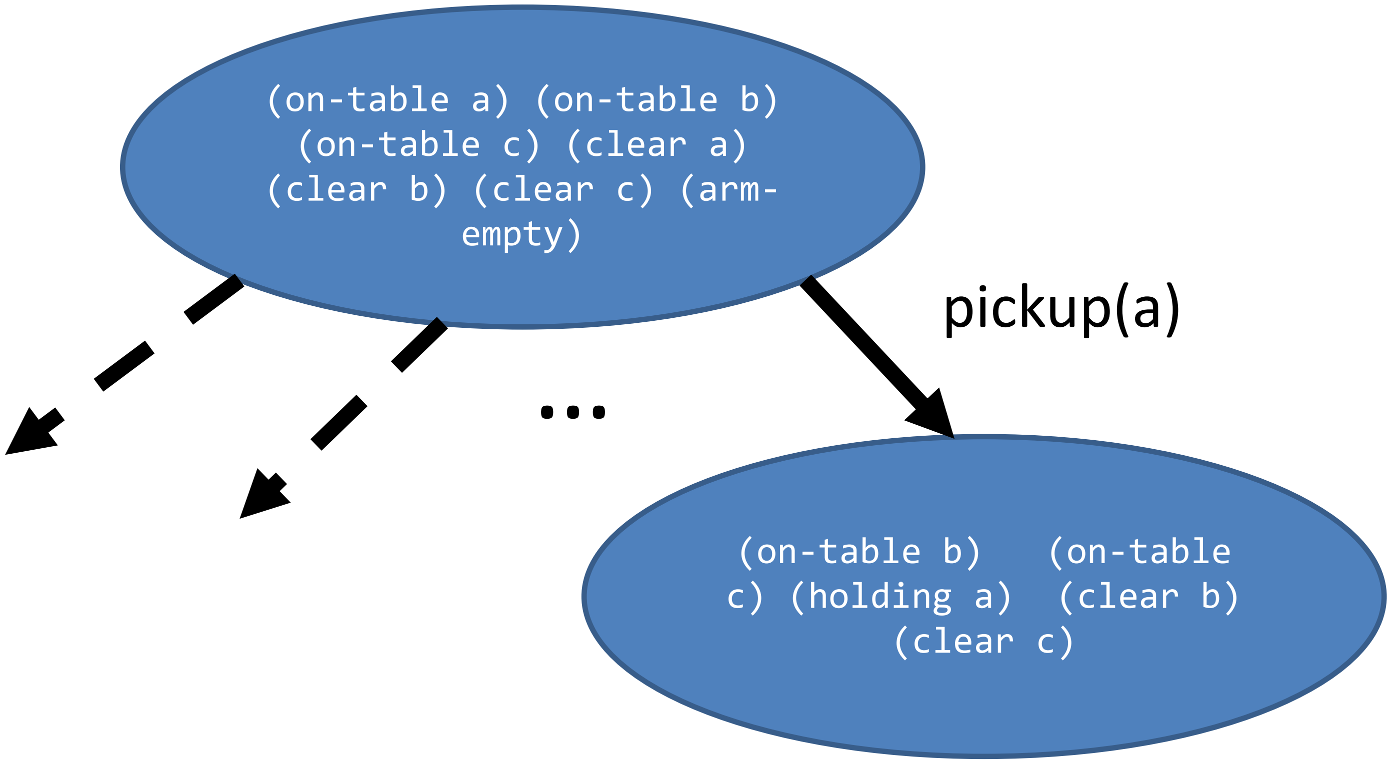
- Literal predicates L: set of predicates from P with bound parameters from O

- A state: list of positive ground literals $s \subseteq L$
- Goal test: list of positive ground literals $\mathrm{g} \subseteq L$

- Operator list:
  - Name
  - Parameters
  - Preconditions
  - Effects

Planning

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



Forward search

- BFS or DFS typically hopeless (high *b, d*)
  - Must use informed search

- Problem has a lot of known structure
  - States are conjunctions
  - Know goal predicates
  - Know predicates deleted and added by actions

- Major approach to solving planning problems:
  - Use knowledge to automatically construct domain-specific heuristic

General strategy

- Relaxation
  - Make problem easier
  - Compute distances in easier problem
  - Use distances as heuristic to hard problem
- FF Planner (major breakthrough circa 2000)
  - Relax problem by deleting negative effects
  - Solve relaxed problem using 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

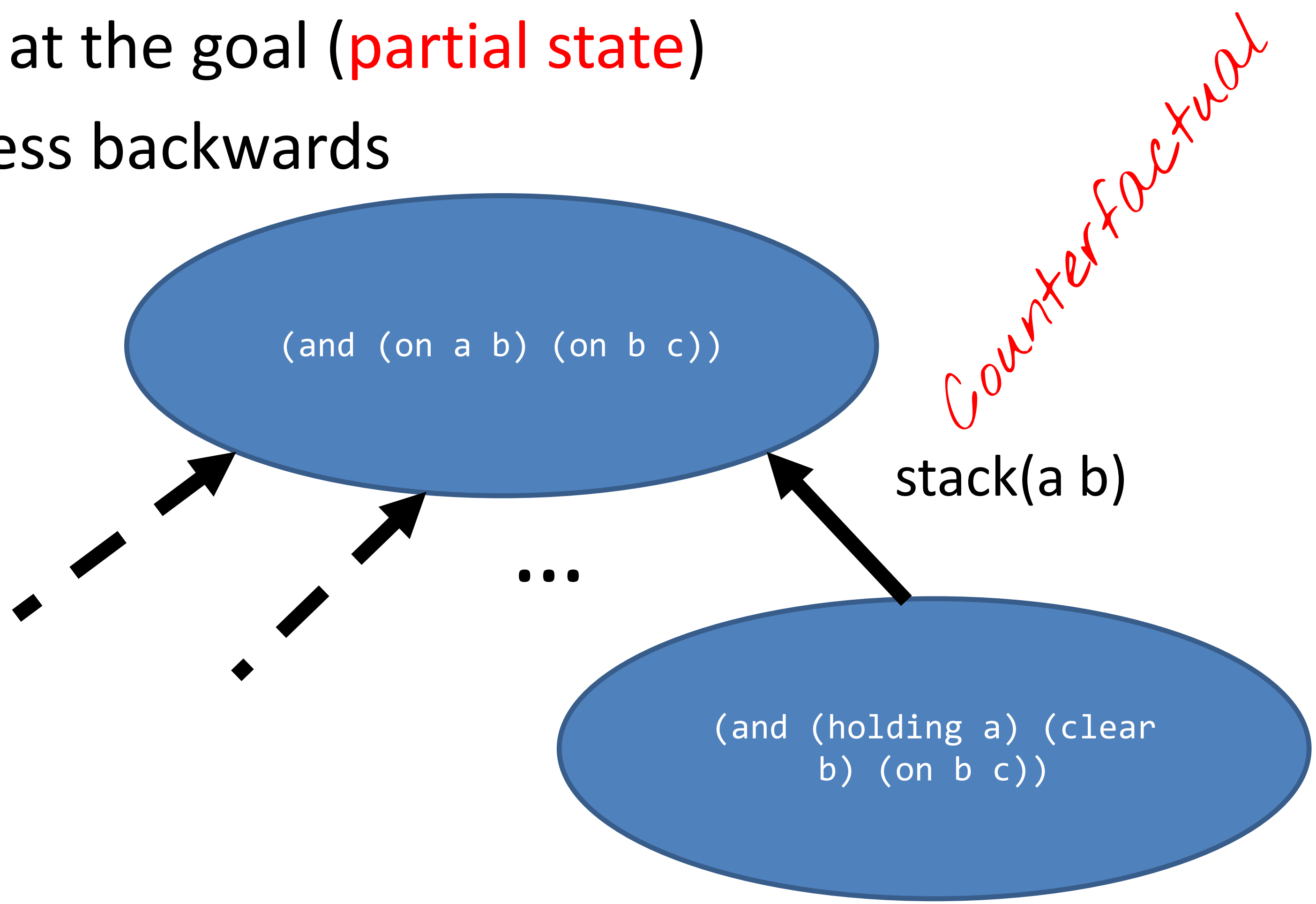
- Why is problem with deleted negative effects easier?

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

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

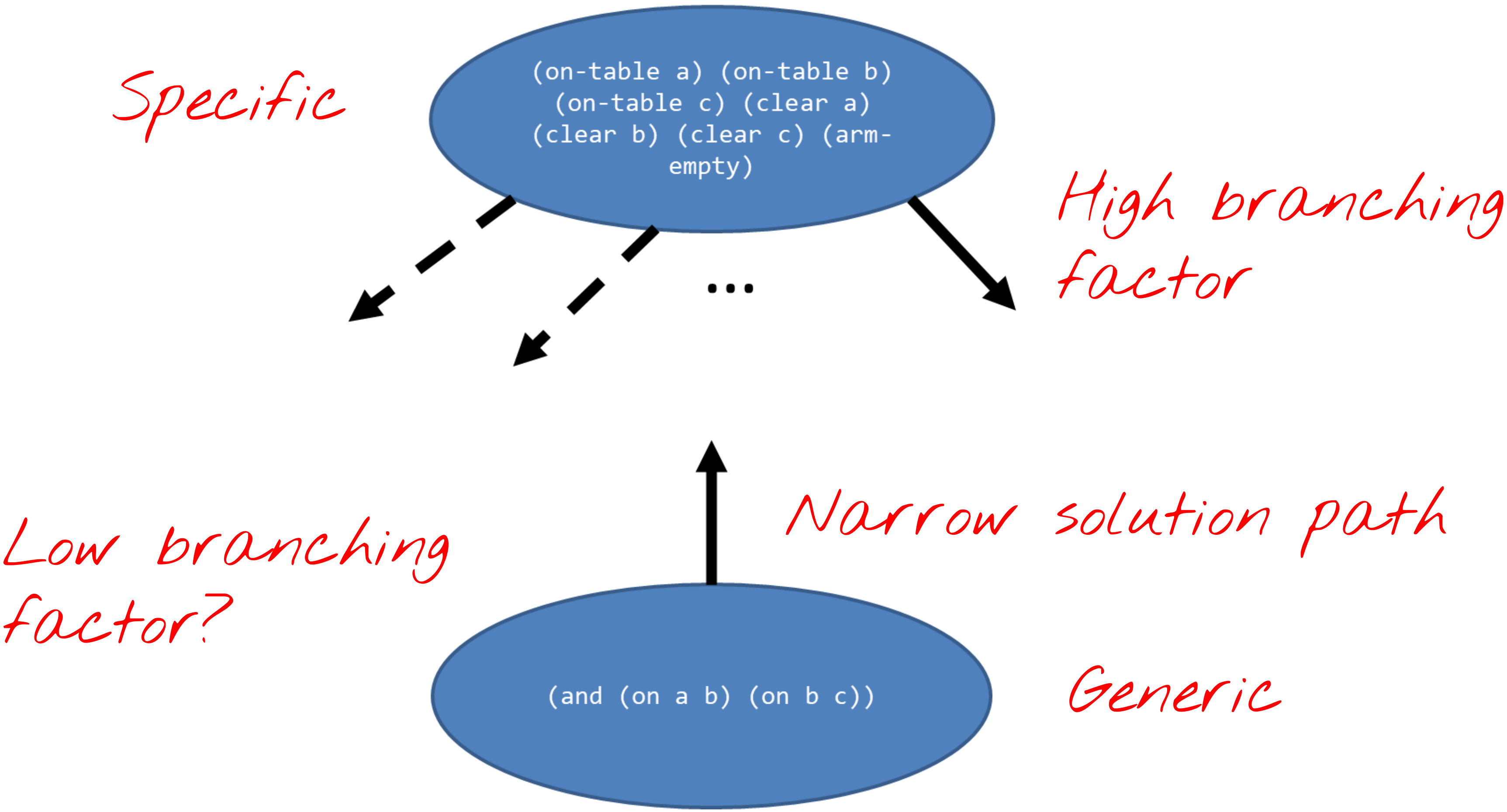
Alternative approach

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

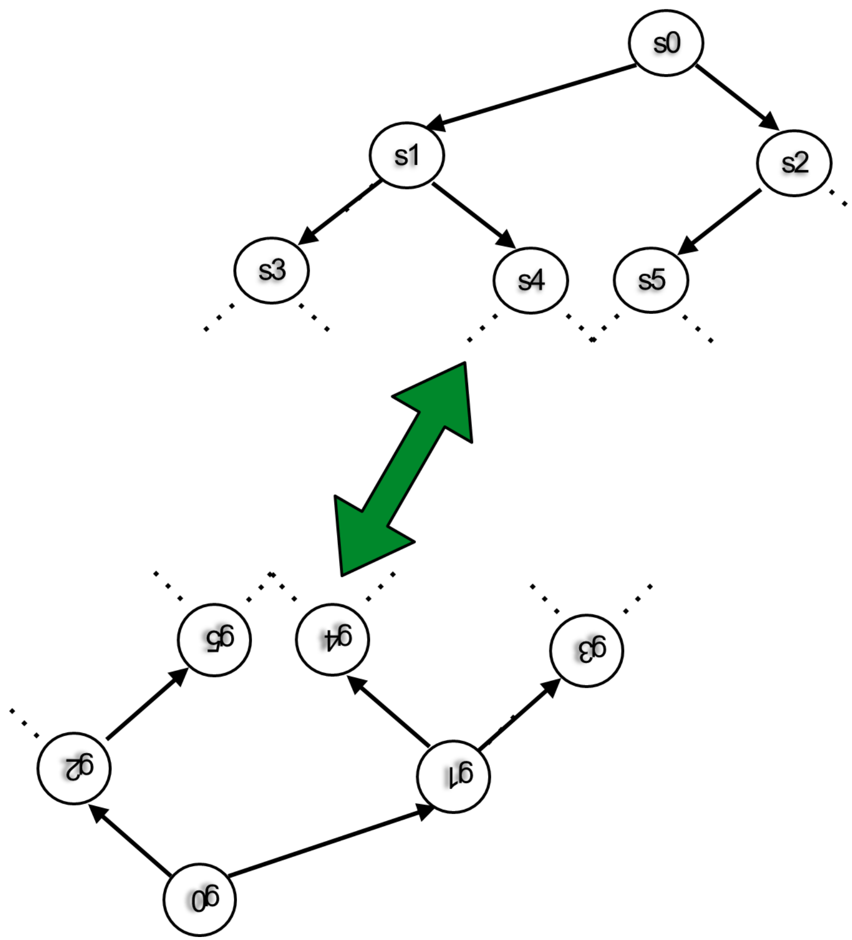


Regression planning

- Why do we expect this to work?



Bidirectional Search



Exploiting expert knowledge

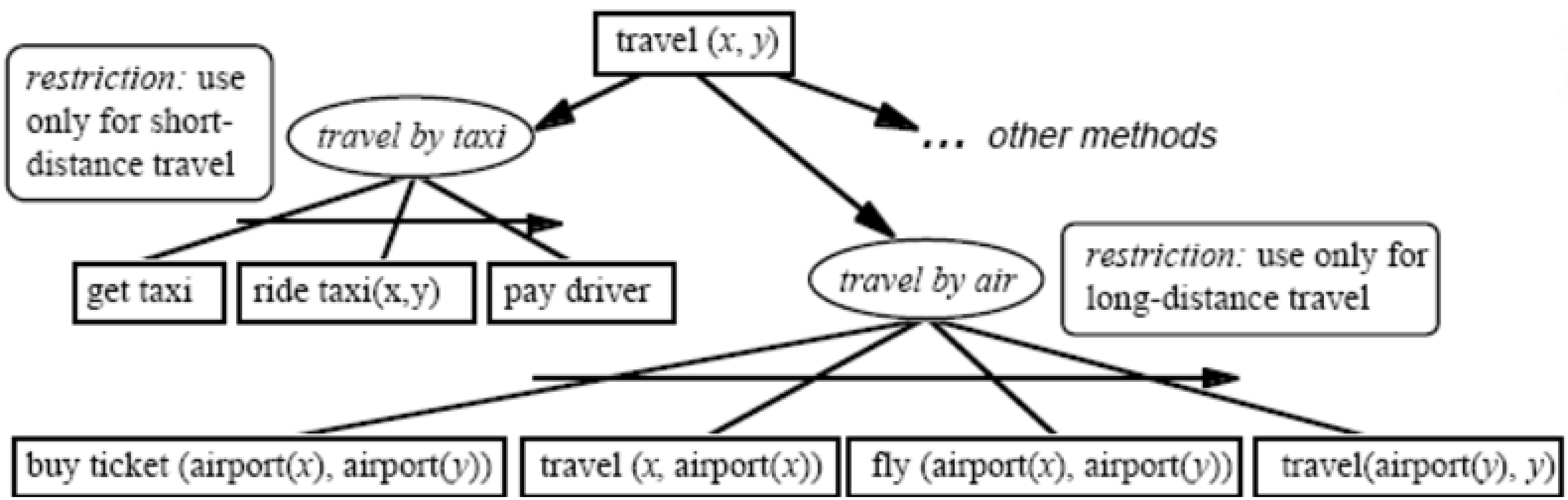
- Often domain expertise can be used to plan more efficiently

- One approach: control rules
  - Hand-written rules
  - Prune some node expansions
  - Effectively decrease branching factor
  - e.g. never move puzzle piece once placed

- Some progress on learning automatically (PRODIGY)

Exploiting domain knowledge

- Another approach: specify partial plans
- "Grasping a door handle always followed by turning it, then opening door"
- Can be written as macro-action
  - A new operator composed of old operators
  - Aim: reduce min solution depth
- Logical extreme: hierarchical task network
  - Specify solution as hierarchy of partly specified plans
  - Planner's role is to fill in details


travel (x, y)
restriction: use only for short-distance travel
travel by taxi
... other methods
get taxi
ride taxi(x,y)
pay driver
travel by air
restriction: use only for long-distance travel
buy ticket (airport(x), airport(y))
travel (x, airport(x))
fly (airport(x), airport(y))
travel(airport(y), y)

Planning competitions

- Competitions held every few years
  - International conference on automation and planning
  - Problems described in PDDL

Coverage	folding	labyrinth	quantum.	recharg.	ricochet.	rubiks	slither.	SUM
ragnarok	8	**8**	13	**14**	**17**	10	**7**	77
scorpion-2023	8	5	**14**	**14**	**17**	10	6	74
odin	8	5	13	**14**	**17**	10	6	73
dofri	8	5	13	13	**17**	10	4	70
cegarplusplus	**9**	5	13	**14**	**17**	0	**7**	65
hapori-stonesoup-opt	7	2	13	**14**	9	**11**	6	62
fdss-2023-opt	7	3	13	13	12	9	4	61
hapori-mip2-opt	7	1	13	**14**	9	10	6	60
hapori-ibacop2-opt	6	1	13	12	15	7	4	58
hapori-greedy-opt	5	1	13	11	9	10	**7**	56
baseline-blind	7	1	7	12	11	8	4	50
decstar-opt	6	1	12	11	8	8	4	50
hapori-delfi-opt	5	2	12	12	8	0	2	41
complementary	5	1	12	13	3	0	3	37
decabstar	2	1	12	10	7	0	5	37
symk	3	1	9	13	4	0	**7**	37
fts-ms-opt	1	1	12	13	2	0	**7**	36
baseline-lmcut	2	1	12	8	5	0	6	34
hapori-epslr-opt	2	1	9	6	4	10	2	34
SymBD-2023-opt	2	1	9	13	1	0	6	32
dom-opt	2	1	12	6	4	0	6	31
hapori-epsdt-opt	1	0	9	6	4	7	4	31
dalai-opt	2	1	11	7	4	0	4	29
fts-sbd-opt	1	0	4	13	0	0	4	22

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 (exchangeable)
- 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 parameterised by intervals (durative action)
  - Allow for simultaneous actions
- For resources:
  - Fixed number and level of resources to start
  - Actions require (and may produce) resource levels
  - Reusable vs consumable
  - Typically a real-valued number

Time and resources

- Same principle for planning
  - Search problem
  - Nodes are state, operators
- But much harder:
  - Simultaneous actions
  - Real-valued values
  - Interval algebras
- Solution is a schedule, not a plan

DART

- Planner used by US military for logistics
- Introduced in 1991, DART had by 1995 offset the monetary equivalent of all funds DARPA had channelled 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.

(https://en.wikipedia.org/wiki/Dynamic_Analysis_and_Replanning_Tool)