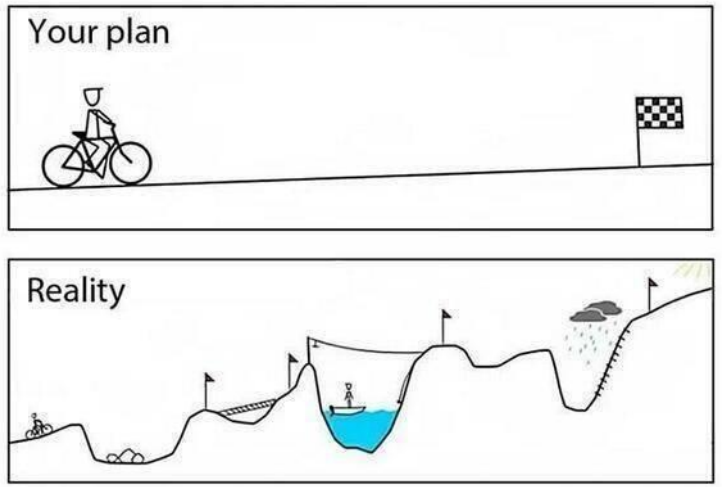
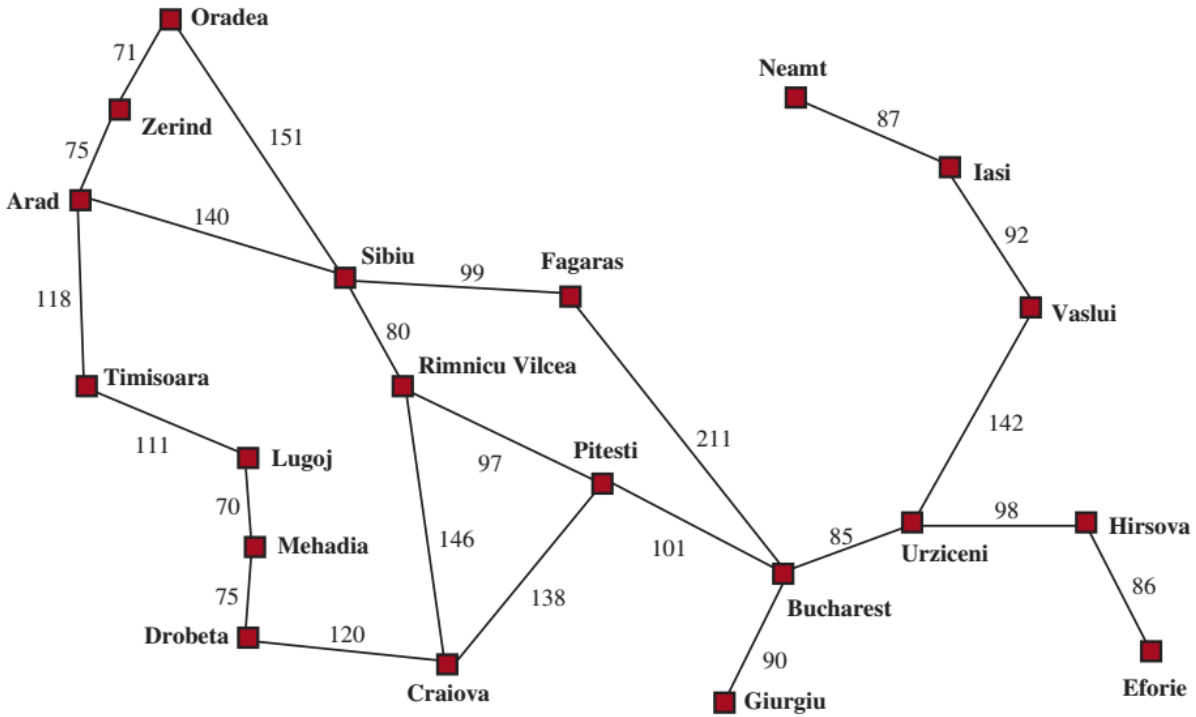
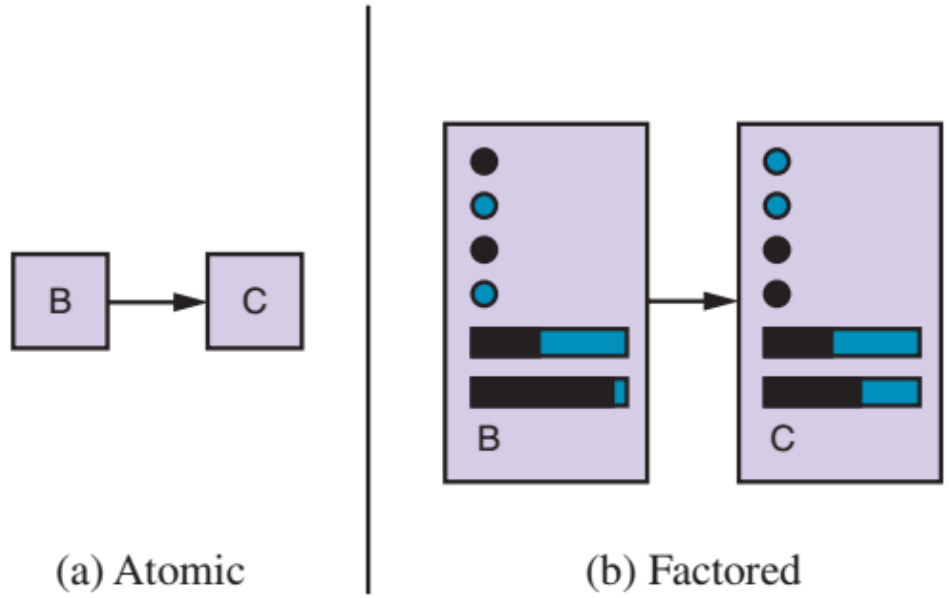
Lecture 11 - Automated Planning

Gleb Sizov
Norwegian University of Science and Technology




Oradea
71
Neamt
87
Zerind
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
Pitesti
211
Lugoj
97
98
Hirsova
70
Mehadia
146
85
101
Urziceni
75
86
138
Bucharest
Drobeta
120
90
Craiova
Giurgiu
Eforie


B
C
B
C
(a) Atomic
(b) Factored

Classical planning

Find sequence of actions to accomplish a goal
in a **discrete**, **deterministic**, **static**, **fully observable** environment.

Approaches so far:

1. Problem-solving by search
2. Propositional logic

Limitations:

1. Require domain-specific heuristic or code
2. Need explicitly represent exponentially large state state

Factored representaiton for planning
Planning Domain Definition Langiage (PDDL)

Prepositional logic:

$$Location_{11}^{0} \wedge FacingEast^{0} \wedge Forward^{0} \implies Location_{21}^{1} \wedge \neg Location_{11}^{1}$$

$$Location_{12}^{0} \wedge FacingEast^{0} \wedge Forward^{0} \implies Location_{22}^{1} \wedge \neg Location_{12}^{1}$$

...

4 directions $*$ T time steps $* n^2$ locattion

PDDL :

$Action(Move(who, from, to)),$

$Precond$: $At(who, from) \wedge Adj(from, to) \wedge \neg Pit(to)$

$Effect$: $\neg At(who, from) \wedge At(who, to)$

http://users.cecs.anu.edu.au/~patrik/pddlman/wumpus.html

PDDL - State

State:

$$At\left(P_1, SFO\right) \wedge Plane\left(P_1\right)$$

- Closed-world assumption - any fluents not mentioned are False
- Unique names assumption - different names refer to diffferent entities
- No variables or negations.

Initial and **goal** state

PDDL - Actions

Action:

$$Action(Fly\,(p, from, to)\,,$$
$$\textsc{Precond}{:}At\,(p, from) \wedge Plane\,(p) \wedge Airport\,(from) \wedge Airport\,(to)$$
$$\textsc{Effect}{:}\neg At\,(p, from) \wedge At\,(p, to))$$

Ground action:

$$Action(Fly\,(P_1, SFO, JFK)\,,$$
$$\textsc{Precond}{:}At\,(P_1, SFO) \wedge Plane\,(P_1) \wedge Airport\,(SFO) \wedge Airport\,(JFK)$$
$$\textsc{Effect}{:}\neg At\,(P_1, SFO) \wedge At\,(P_1, JFK))$$

> Any variable in the effect must also appear in the precondition

PDDL - Result of executing action

$$\textsc{Result}\,(s, a) = (s - \text{DEL}\,(a)) \cup \text{ADD}\,(a)\,.$$

Example:

$a = Fly(P_1, SFO, JFK)$

$Del(a) = At(P_1, SFO)$

$Add(a) = At(P_1, JFK)$

PDDL Example - Air cargo transport

PDDL description:

$$
\begin{aligned}
&Init(At(C_1,\ SFO)\ \wedge\ At(C_2,\ JFK)\ \wedge\ At(P_1,\ SFO)\ \wedge\ At(P_2,\ JFK)\\
&\quad \wedge\ Cargo(C_1)\ \wedge\ Cargo(C_2)\ \wedge\ Plane(P_1)\ \wedge\ Plane(P_2)\\
&\quad \wedge\ Airport(JFK)\ \wedge\ Airport(SFO))\\
&Goal(At(C_1,\ JFK)\ \wedge\ At(C_2,\ SFO))\\
&Action(Load(c,\ p,\ a),\\
&\quad \text{PRECOND: } At(c,\ a)\ \wedge\ At(p,\ a)\ \wedge\ Cargo(c)\ \wedge\ Plane(p)\ \wedge\ Airport(a)\\
&\quad \text{EFFECT: } \neg\ At(c,\ a)\ \wedge\ In(c,\ p))\\
&Action(Unload(c,\ p,\ a),\\
&\quad \text{PRECOND: } In(c,\ p)\ \wedge\ At(p,\ a)\ \wedge\ Cargo(c)\ \wedge\ Plane(p)\ \wedge\ Airport(a)\\
&\quad \text{EFFECT: } At(c,\ a)\ \wedge\ \neg\ In(c,\ p))\\
&Action(Fly(p,\ from,\ to),\\
&\quad \text{PRECOND: } At(p,\ from)\ \wedge\ Plane(p)\ \wedge\ Airport(from)\ \wedge\ Airport(to)\\
&\quad \text{EFFECT: } \neg\ At(p,\ from)\ \wedge\ At(p,\ to))
\end{aligned}
$$

Solution:

$$
\begin{aligned}
&[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK),\\
&\ Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)].
\end{aligned}
$$

PDDL Example - Spare tire

PDDL description:

$$
\begin{aligned}
&Init(Tire(Flat) \wedge Tire(Spare) \wedge At(Flat, Axle) \wedge At(Spare, Trunk)) \\
&Goal(At(Spare, Axle)) \\
&Action(Remove(obj, loc), \\
&\quad \text{PRECOND: } At(obj, loc) \\
&\quad \text{EFFECT: } \neg At(obj, loc) \wedge At(obj, Ground)) \\
&Action(PutOn(t, Axle), \\
&\quad \text{PRECOND: } Tire(t) \wedge At(t, Ground) \wedge \neg At(Flat, Axle) \wedge \neg At(Spare, Axle) \\
&\quad \text{EFFECT: } \neg At(t, Ground) \wedge At(t, Axle)) \\
&Action(LeaveOvernight, \\
&\quad \text{PRECOND:} \\
&\quad \text{EFFECT: } \neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk) \\
&\qquad \wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle) \wedge \neg At(Flat, Trunk))
\end{aligned}
$$

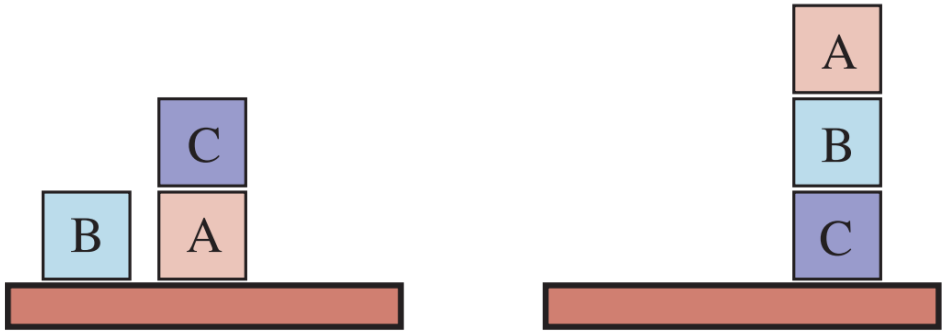
Solution:

$$[Remove(Flat, Axle), Remove(Spare, Trunk), PutOn(Spare, Axle)]$$

PDDL Example - Blocks world

PDDL description:

$Init(On(A, Table) \land On(B, Table) \land On(C, A)$
$\land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table))$
$Goal(On(A, B) \land On(B, C))$
$Action(Move(b, x, y),$
PRECOND: $On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land$
$(b \neq x) \land (b \neq y) \land (x \neq y),$
EFFECT: $On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$
$Action(MoveToTable(b, x),$
PRECOND: $On(b, x) \land Clear(b) \land Block(b) \land Block(x),$
EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x))$



Start State Goal State

Solution:

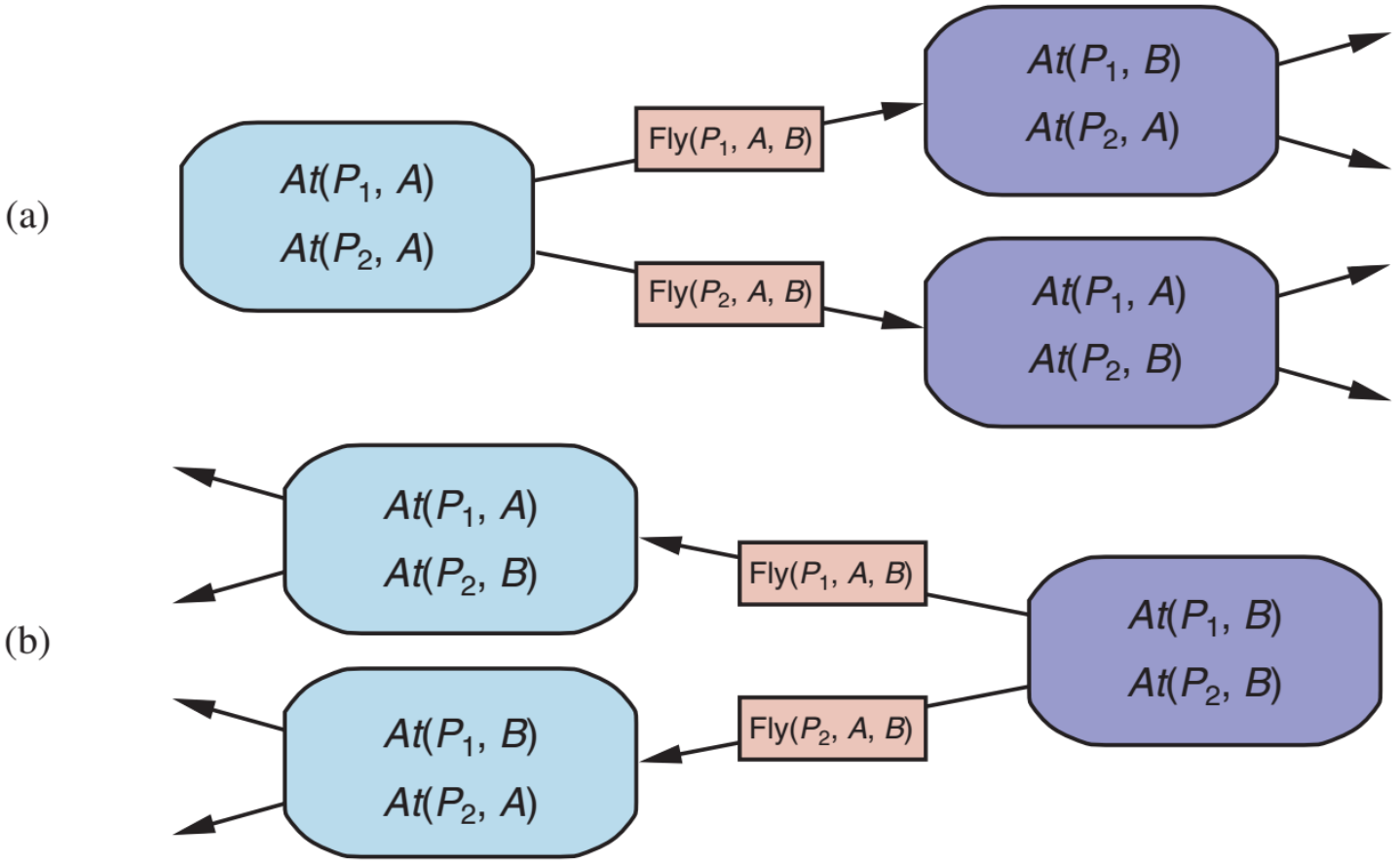
$[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]$

Algorithms for Classical Planning

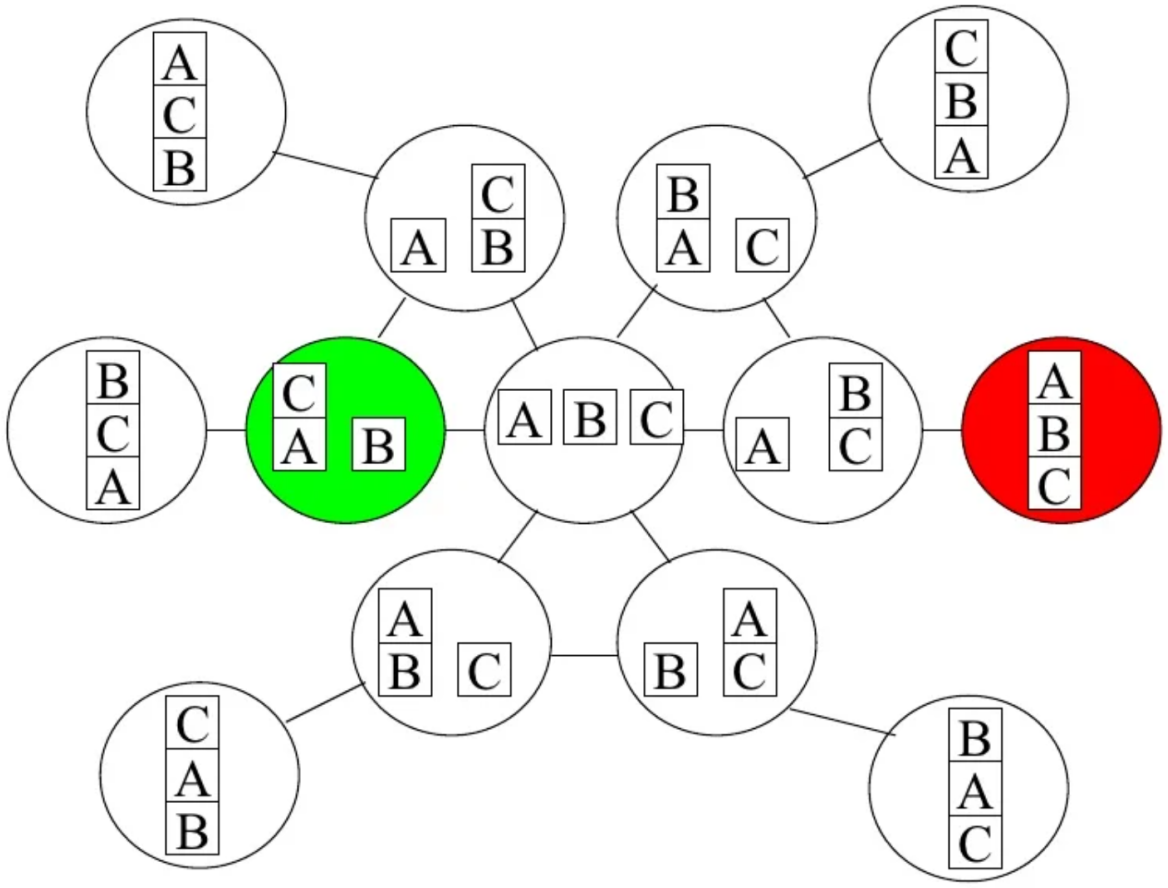
a. Forward (progression) search

b. Backward (regression) search

c. Logical inference



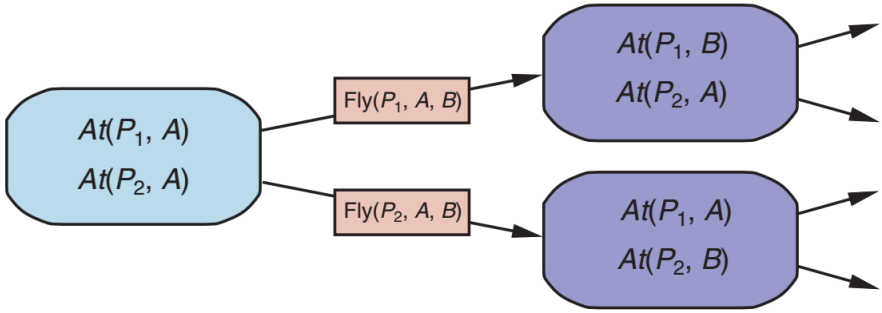
Planning search space



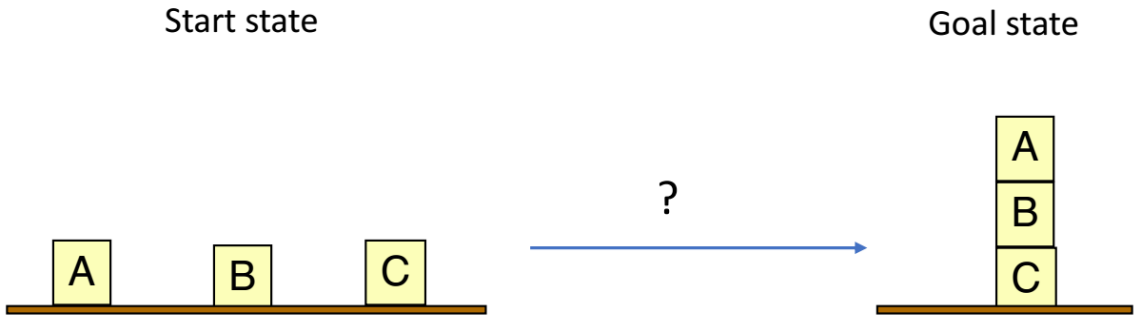
(Michael Moll)

Forward (progression) search algorithm

1. Ground the actions by substituting any variable with constants
2. Choose a ground action that is applicable
3. Determine the new content of the knowledge base
4. Repeat until goal state is reached



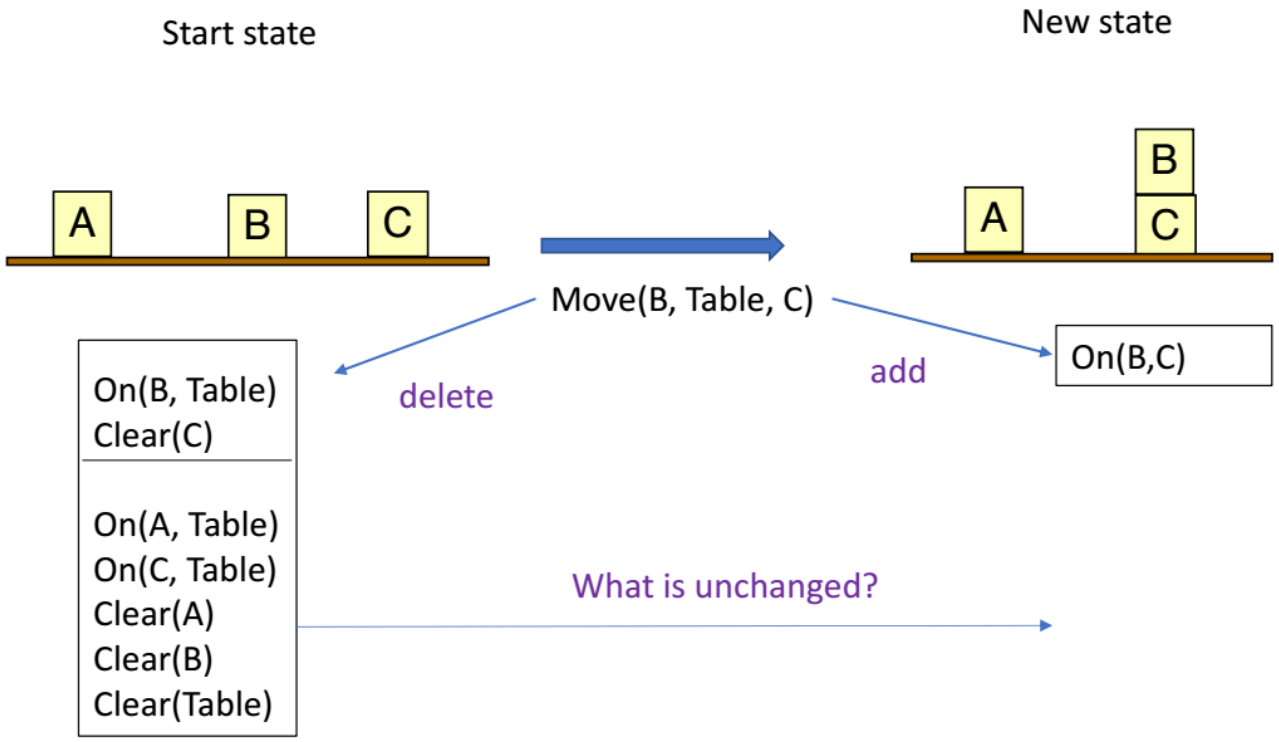
Forward search example - Step 0



What is your plan?

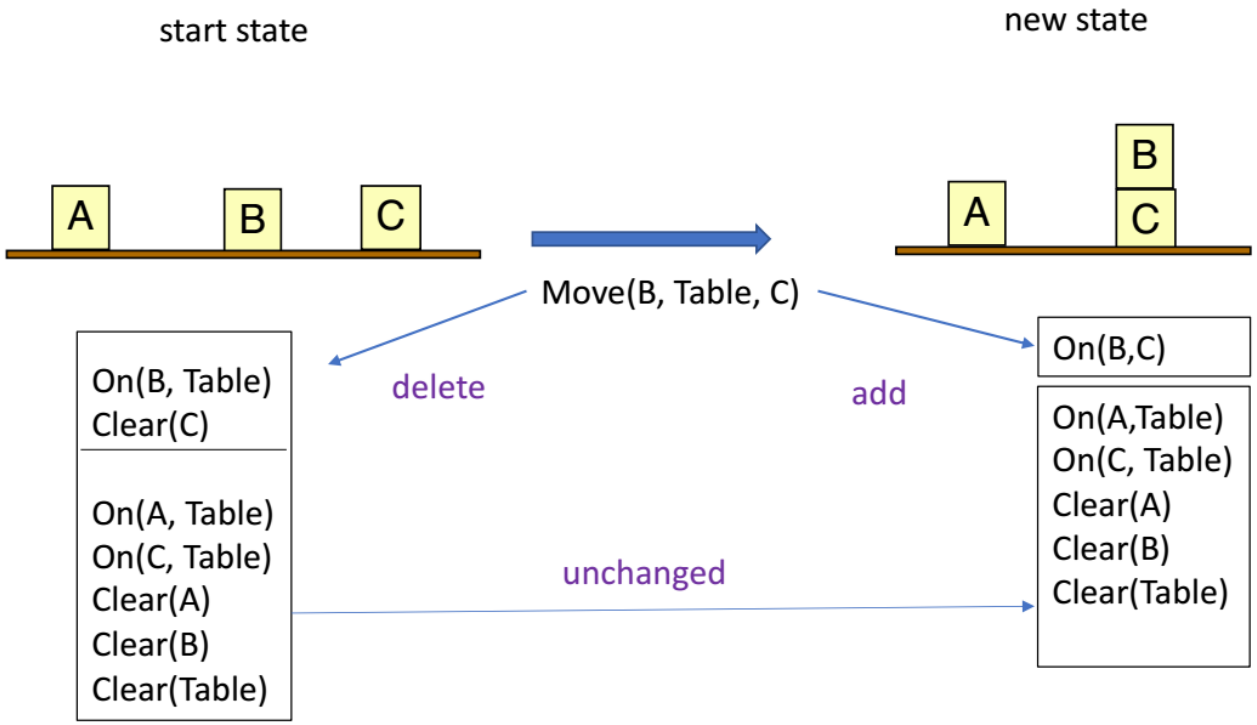
Forward search example - Step 1

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)



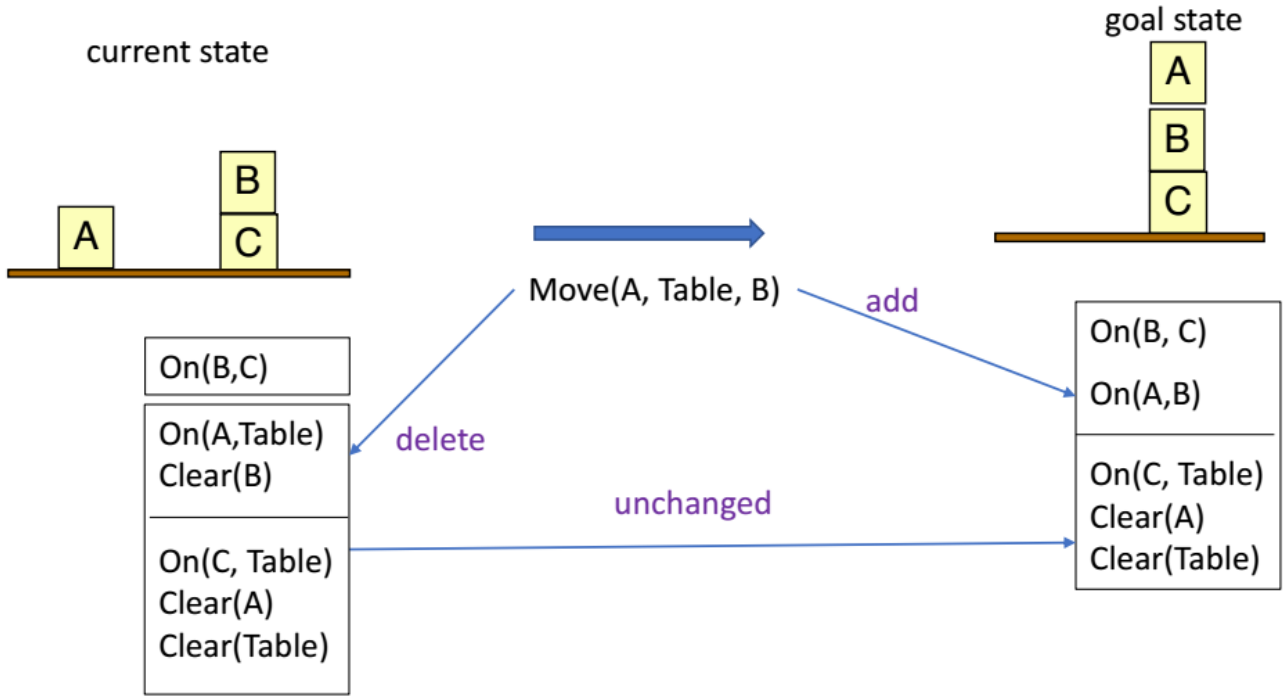
Forward search example - Step 2

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)

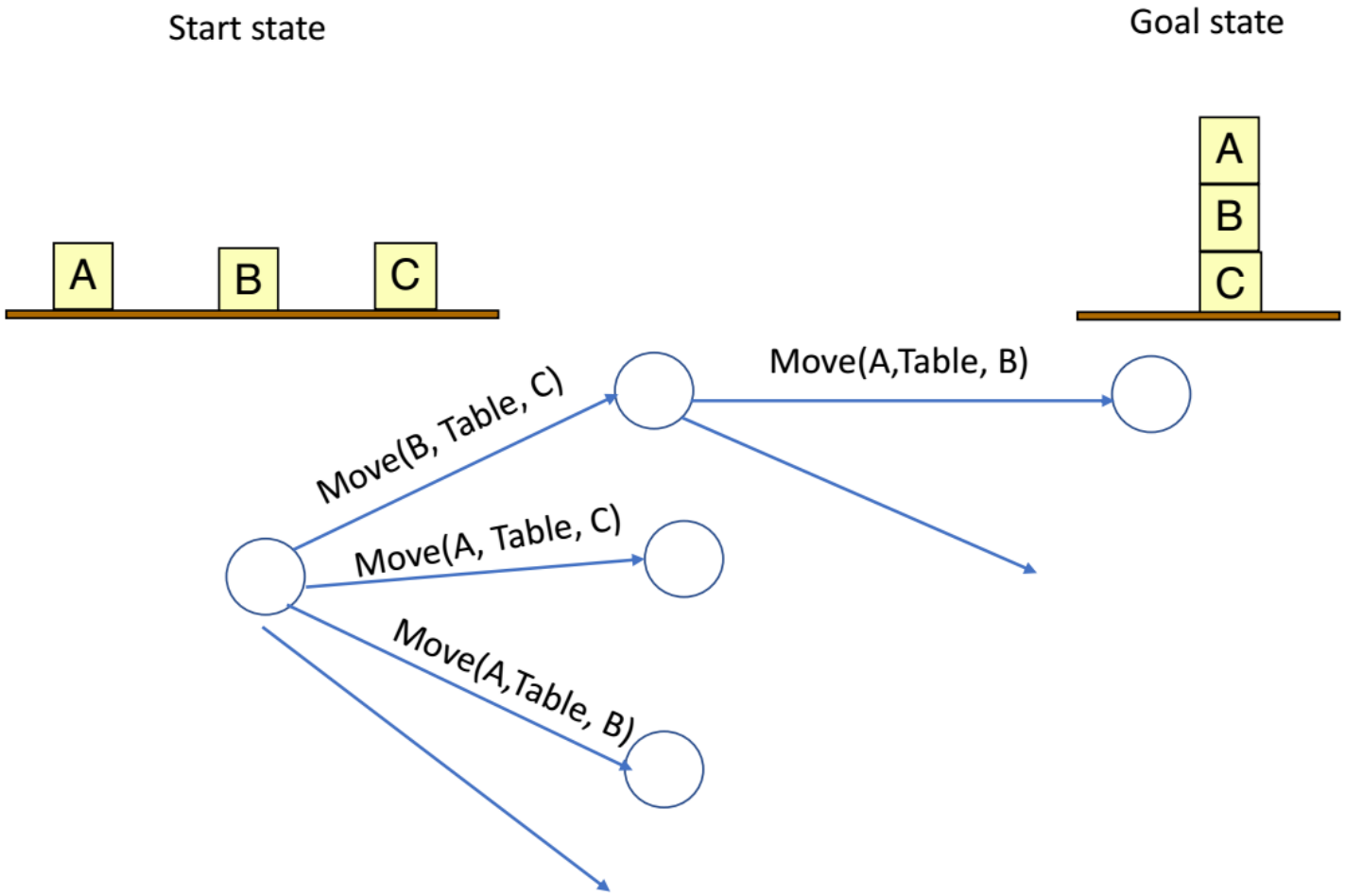


Forward search example - Step 3

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)



Forward search example - Branching



Efficiency of forward search

Forward search can have a very large branching factor.

Example:

Air cargo problem: 50 planes, 9 airports, 200 packages

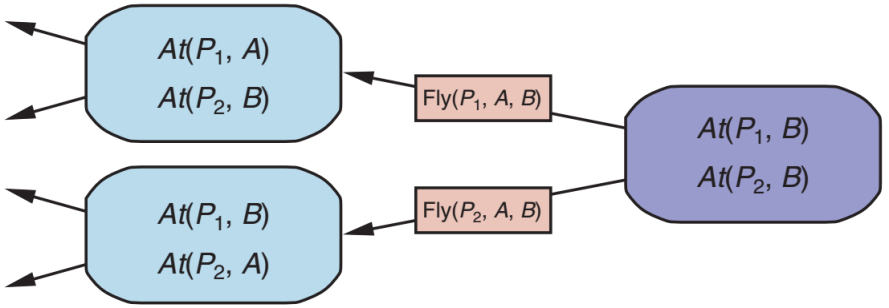
Avg branching: 2000

41-step solution: 2000^{41} nodes

> Need good heuristic or prunning.

Backward (regression) search algorithm

1. Pick a **relevant action** that satisfies (some of) goal propositions.
2. Make a new goal by applying an **action backwards**:
    - Removing the goal conditions satisfied by the action
    - Adding the preconditions of this action
    - Keeping any unsolved goal propositions
3. Repeat until the goal set is satisfied by the start state



Relevant action in backward search

> A relevant action is one with an effect that unifies with one of the goal literals, but with no effect that negates any part of the goal.

Example

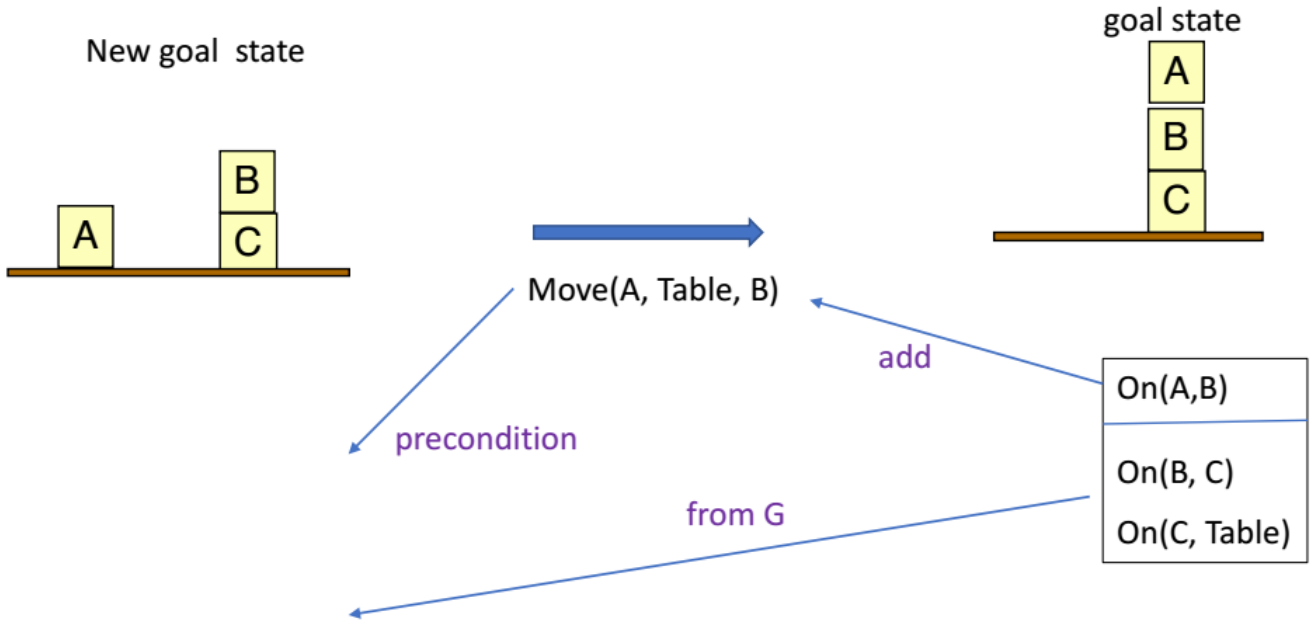
Goal: $\neg Poor \wedge Famous$

Relevant effect: $Famous$

Not relevant effect: $Poor \wedge Famous$

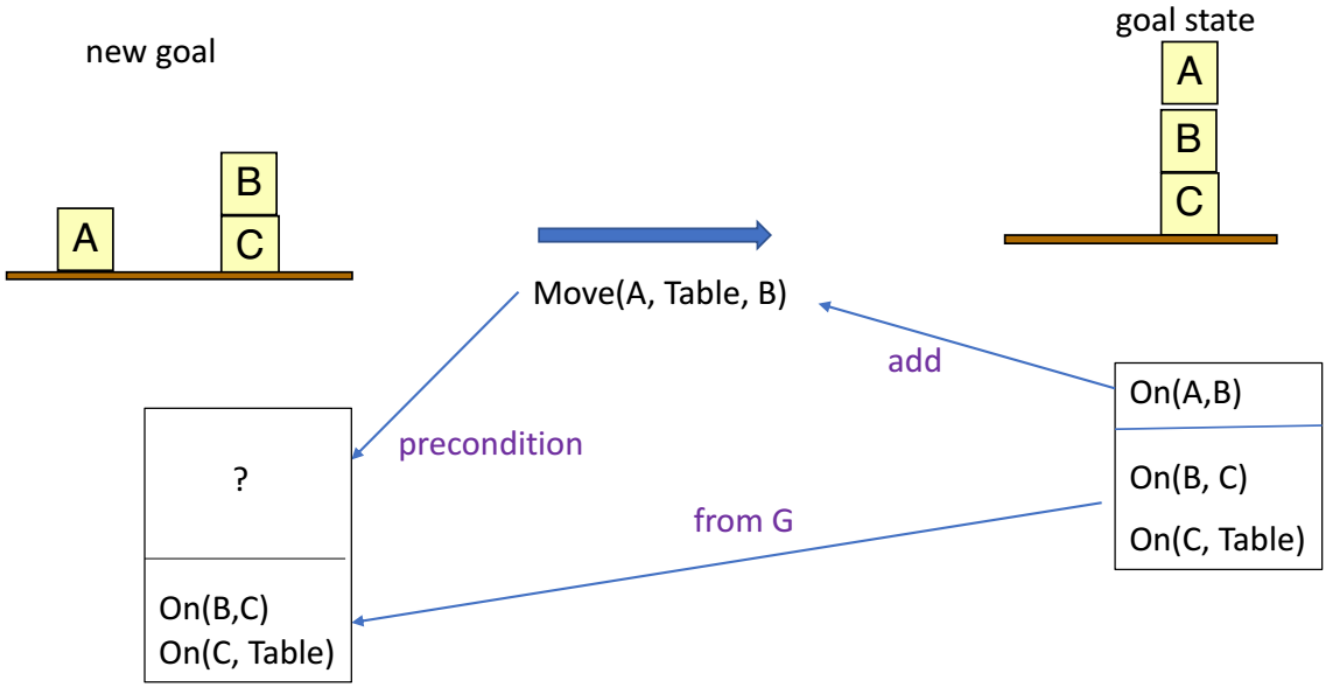
Backward search example - Step 1

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)



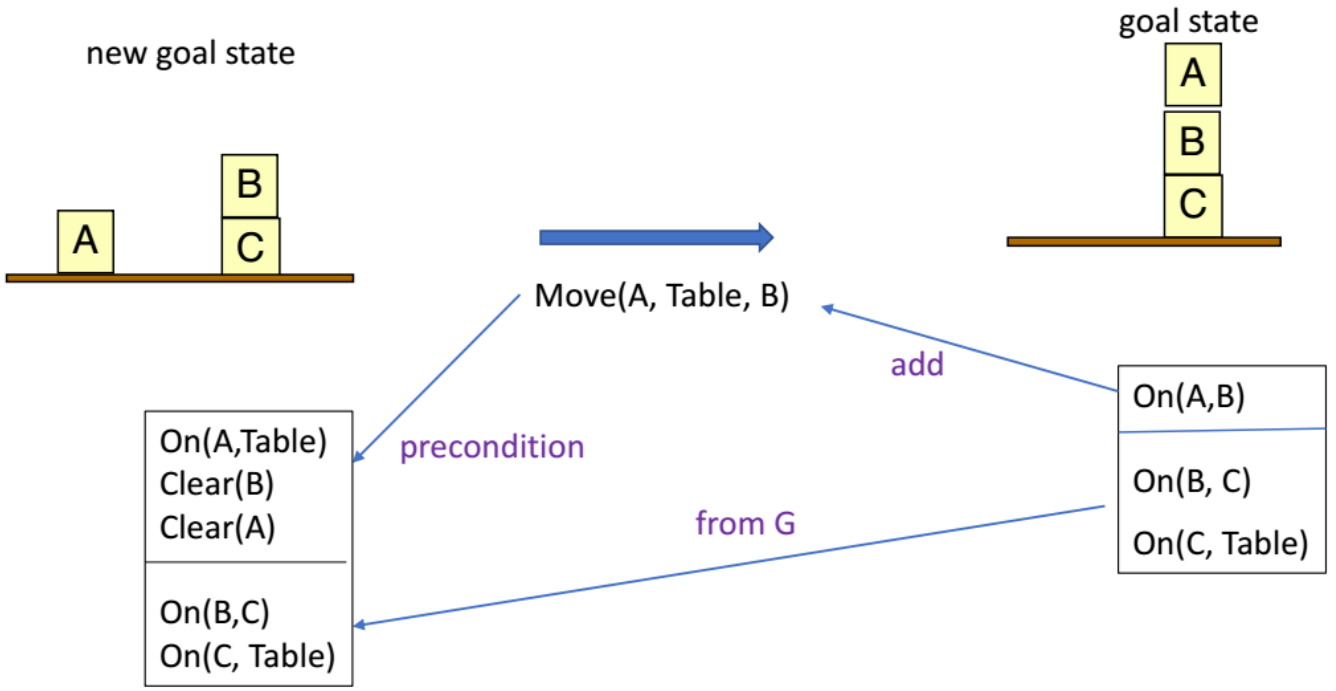
Backward search example - Step 2

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)

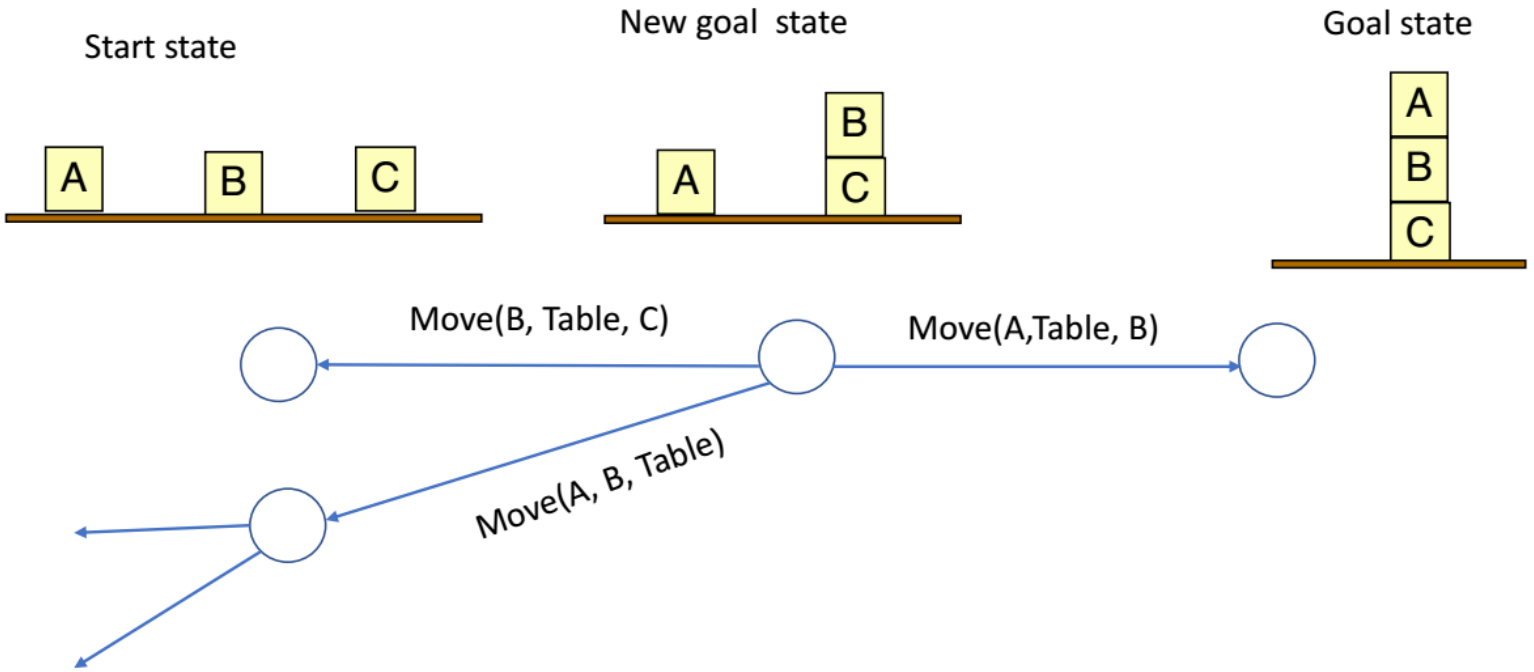


Backward search example - Step 3

(Move (b, x, y)
PRECOND}: On(b, x) ∧ Clear(b) ∧ Clear(y)
EFFECT : On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬Clear(y)



Backward search example - Branching



Efficiency of backward search - branching

Lower branching factor than forward search.

Example

Goal: $Own(9780134610993)$

Action: $A = Action(Buy(i), Precond{:}ISBN(i), Effect{:}Own(i))$

Efficiency of backward search - variables in states

Harder to come up with good heuristics because of variables in states

Example

Goal:

$$At(C_2, SFO)$$

Unified action:

$$Action(Unload(C_2, p', SFO),$$
$$Precond{:} In(C_2, p') \wedge At(p', SFO) \wedge Cargo(C_2) \wedge Plane(p') \wedge$$
$$Airport(SFO)$$
$$Effect{:} At(C_2, SFO) \wedge \neg In(C_2, p'))$$

Regressed goal:

$$g' = In(C_2, p') \wedge At(p', SFO) \wedge Cargo(C_2) \wedge Plane(p') \wedge Airport(SFO)$$

Planning as Boolean satisfiability (SAT)

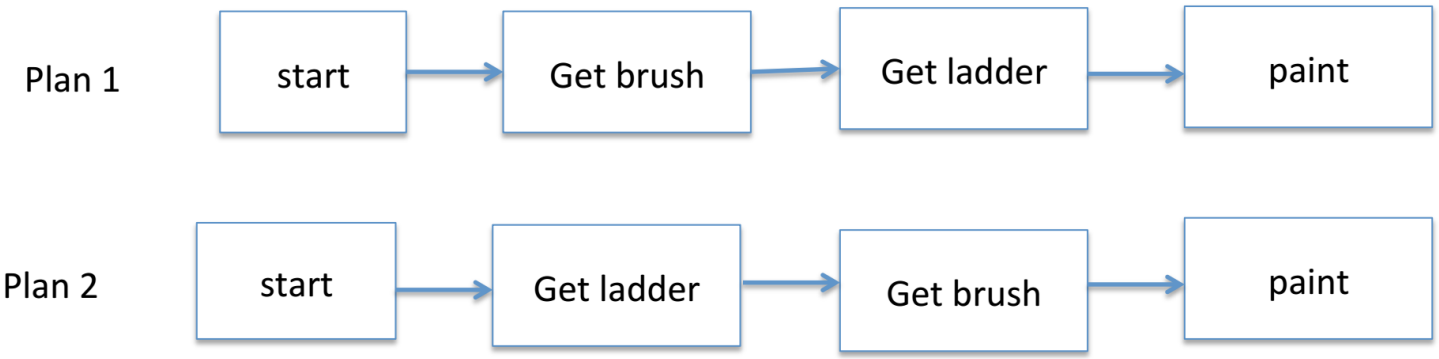
1. For each action substitute variables with constants, e.g.
   $Unload(c/C_2, p/P_3, a/SFO)$
2. Add action exclusion axioms to avoid two actions at the same time, e.g.
   $\neg(FlyP_1SFOJFK^1 \wedge FlyP_1SFOBUH^1)$
3. Add precondition axioms $A^t \implies Precond(A)^t$, e.g.
   $FlyP_1SFOJFK^1 \implies At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$
4. Define initial state, F^0 for mentioned and $\neg F^0$ for not mentioned fluents.
5. Propositionalize the goal, e.g. from $On(A, x) \wedge Block(x)$
   to $(On(A, A) \wedge Block(A)) \vee (On(A, B) \wedge Block(B))$
6. Add successor-state axioms, e.g.
   $F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)$

Other classical planning approaches

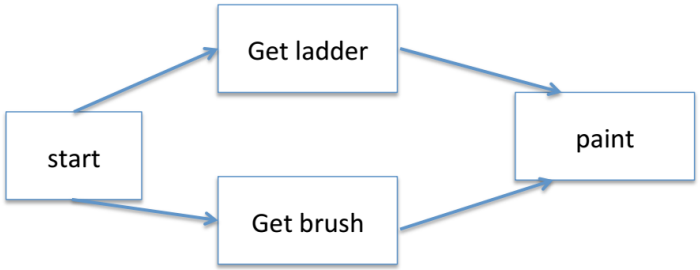
1. Graphplan - encodes precondition and effect constraints in a **planning graph**
2. Situation calculus - uses first-order logic and first-order solvers
3. Constrain sastifaction problem - similar to SAT but more compact

Total vs partial order plan

Total order: Plan is always a strict sequence of actions

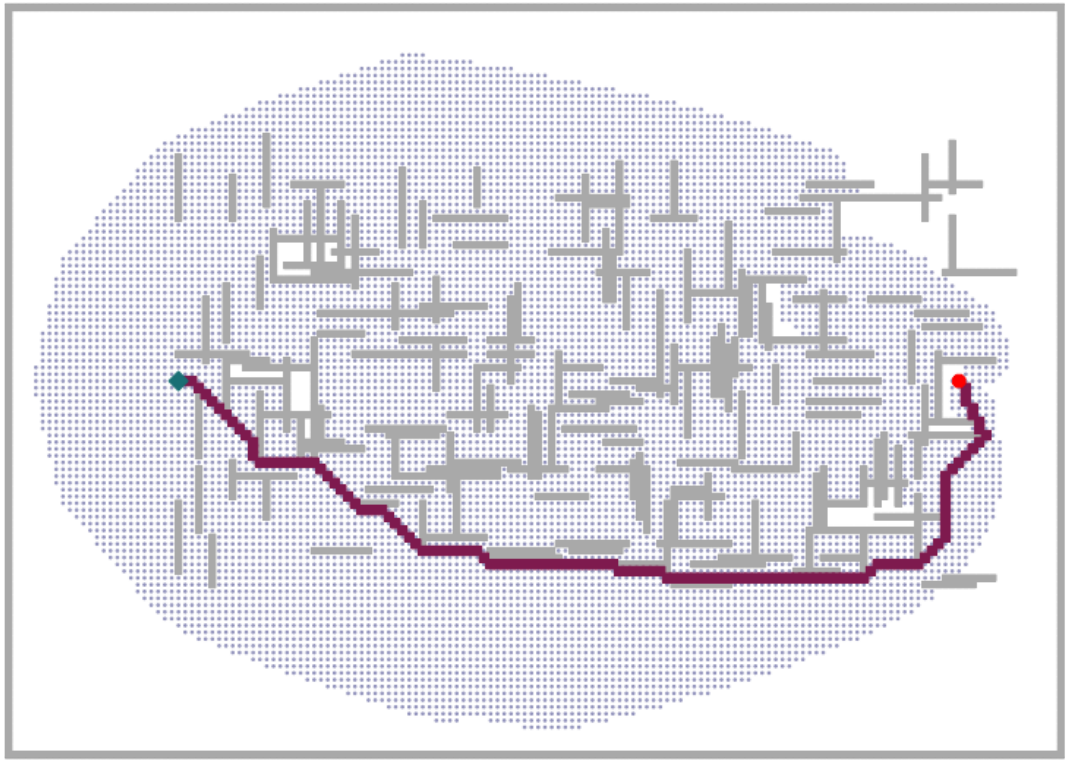


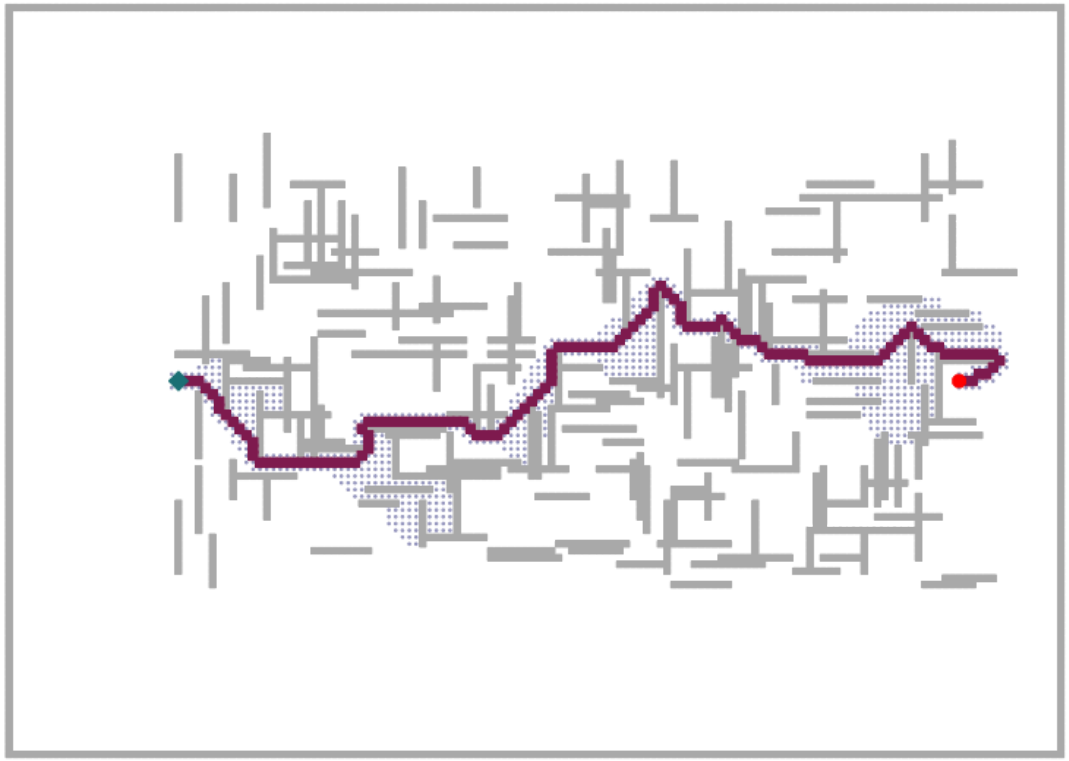
Partial order: Plan steps may be unordered



Heuristics for Planning

Forward and backward search are inefficient without a good heuristic $h(s)$.



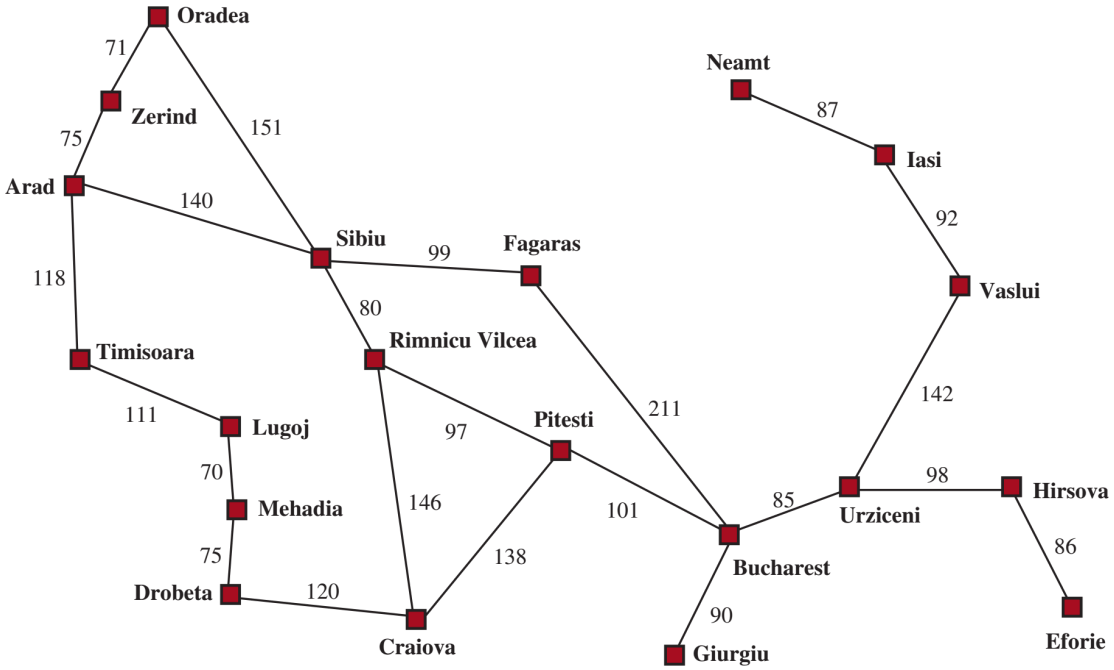


Heuristics for Planning

How many actions are needed?

We need **admissible** heuristic - don't overestimate.

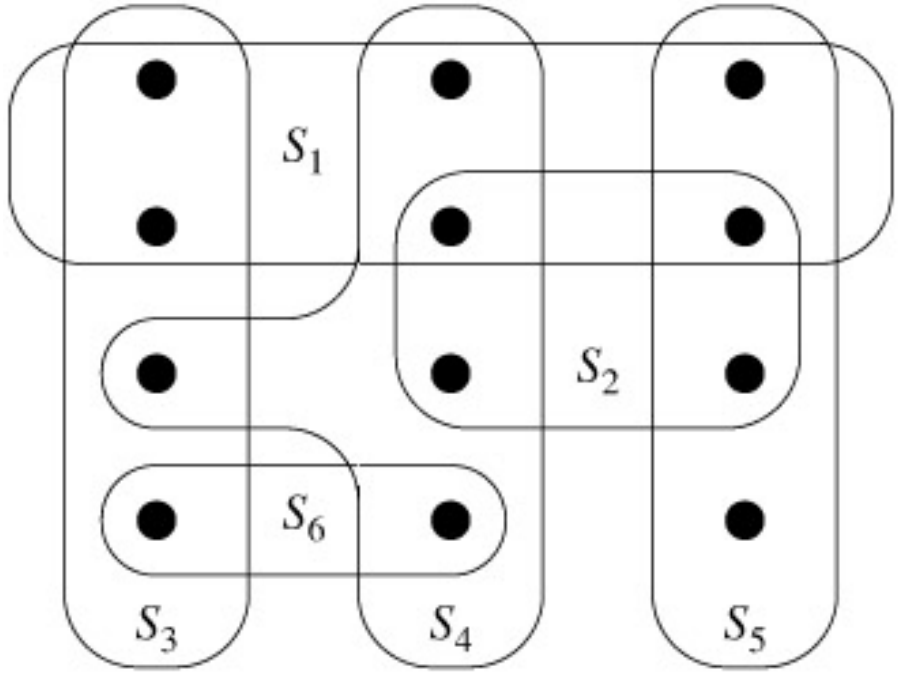
How to get it? Relax!



Ignore-precoditions heuristic

Ignore all preconditions:

- Every actions becomes applicable in every state.
- Set-cover problem - NP-hard
- Greedy solution not admissible



Ignore selected preconditions

$Action(Slide(t, s1, s2)$
$Precond : On(t, s1) \wedge Tile(t) \wedge Blank(s2) \wedge Adjacent(s1, s2)$
$Effect : On(t, s2) \wedge Tile(t) \wedge Blank(s1) \wedge \neg On(t, s1) \wedge \neg Blank(s2)$

Nr-of-misplaced tiles - remove $Blank(s2) \wedge Adjacent(s1, s2)$

Manhattan distance - remove $Blank(s2)$

Ignore-delete list heuristic

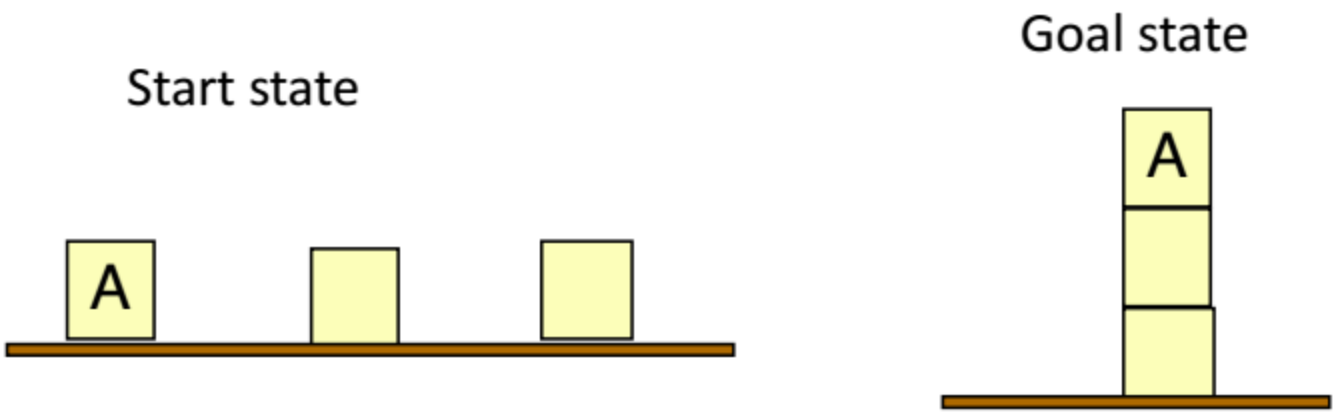
Relaxed version:

- Remove negative literals from preconditions and goals.
- Remove delete lists from all actions.

Properties:

- No undos by other actions - monotonic progress.
- NP-hard but can be approximated with hill-climbing

Domain-independent pruning - Symmetry reduction



Domain-independent pruning - Preferred action

Focus on promising branches. How?

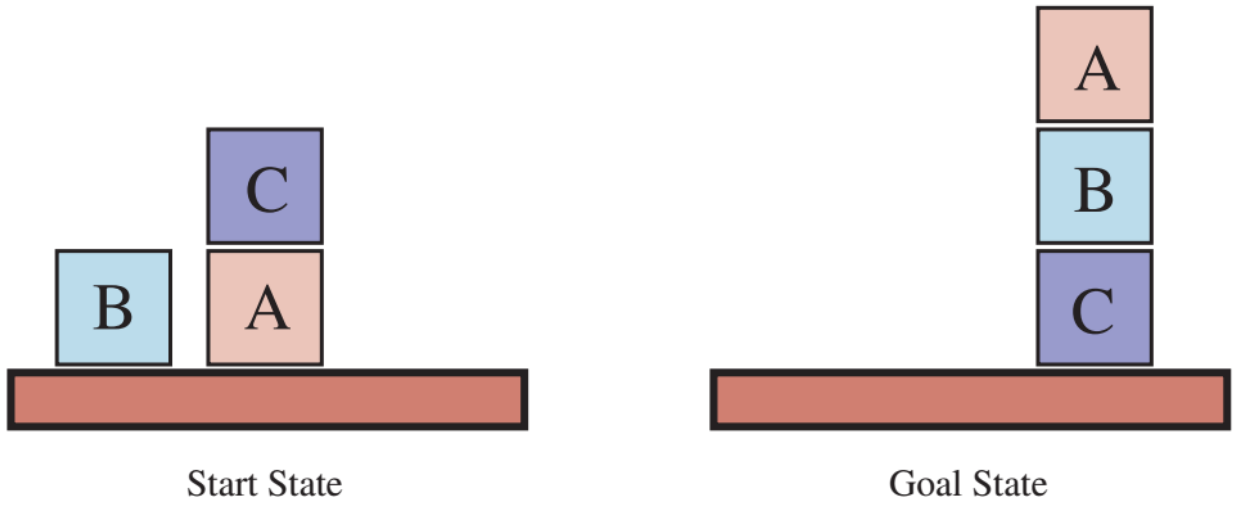
1. Define and solve a relaxed version to get **relaxed plan**
2. Choose preferred actions - steps of the relaxed plan or achieve a precondition of the relaxed plan

Domain-independent pruning - Serializable subgoals

Order of subgoals that can be achieved without undoing previously achived subgoals.

No backtracking!

Example 1: Bottom-to-top



Example 2: No dependency - Arbitrary switching on lights.

State abstraction

Many-to-one mapping of states to abstract representation.

Methods:

1. Identifying and achieving similar subgoals as one,
   e.g. many packages with the same src and destination.
2. Creating heuristic from decomposed subgoals,
   e.g. $Cost(P_i) + Cost(P_j)$
3. Pattern databases for subgoals

Planning and acting in classical environment

"Classical" environment:

- Deterministic
- Agent knows effects of each action
- Fully observable
- Static

The agent does not need perception:

- Can calculate which state results from any sequence of actions
- Always knows which state it is in

Planning and acting in non-classical environment

"Non-classical" environment:

- Partially observable and/or nondeterministic environment
- Incorrect information (differences between world and model)
- The agent’s future actions will depend on future percepts
- The future percepts cannot be determined in advance

Use percepts:

- Perceive the changes in the world
- Act accordingly
- Adapt plan when necessary

Types of non-classical planning

Sensorless planning for environments with no observations.

Contingency planning for partially observable and nondeterministic environments.

Online planning and **replanning** for unknown environments.

Percept schema - Part 1

Example problem: Paint chair and table the same color.

$$Init\left(Object\left(Table\right) \wedge Object\left(Chair\right) \wedge Can\left(C_1\right) \wedge Can\left(C_2\right) \wedge InView\left(Table\right)\right)$$
$$Goal\left(Color\left(Chair, c\right) \wedge Color\left(Table, c\right)\right)$$

$$Action(RemoveLid\left(can\right),$$
$$\textsc{Precond}{:}Can\left(can\right)$$
$$\textsc{Effect}{:}Open\left(can\right))$$
$$Action(Paint\left(x, can\right),$$
$$\textsc{Precond}{:}Object\left(x\right) \wedge Can\left(can\right) \wedge Color\left(can, c\right) \wedge Open\left(can\right)$$
$$\textsc{Effect}{:}Color\left(x, c\right))$$

Percept schema - Part 2

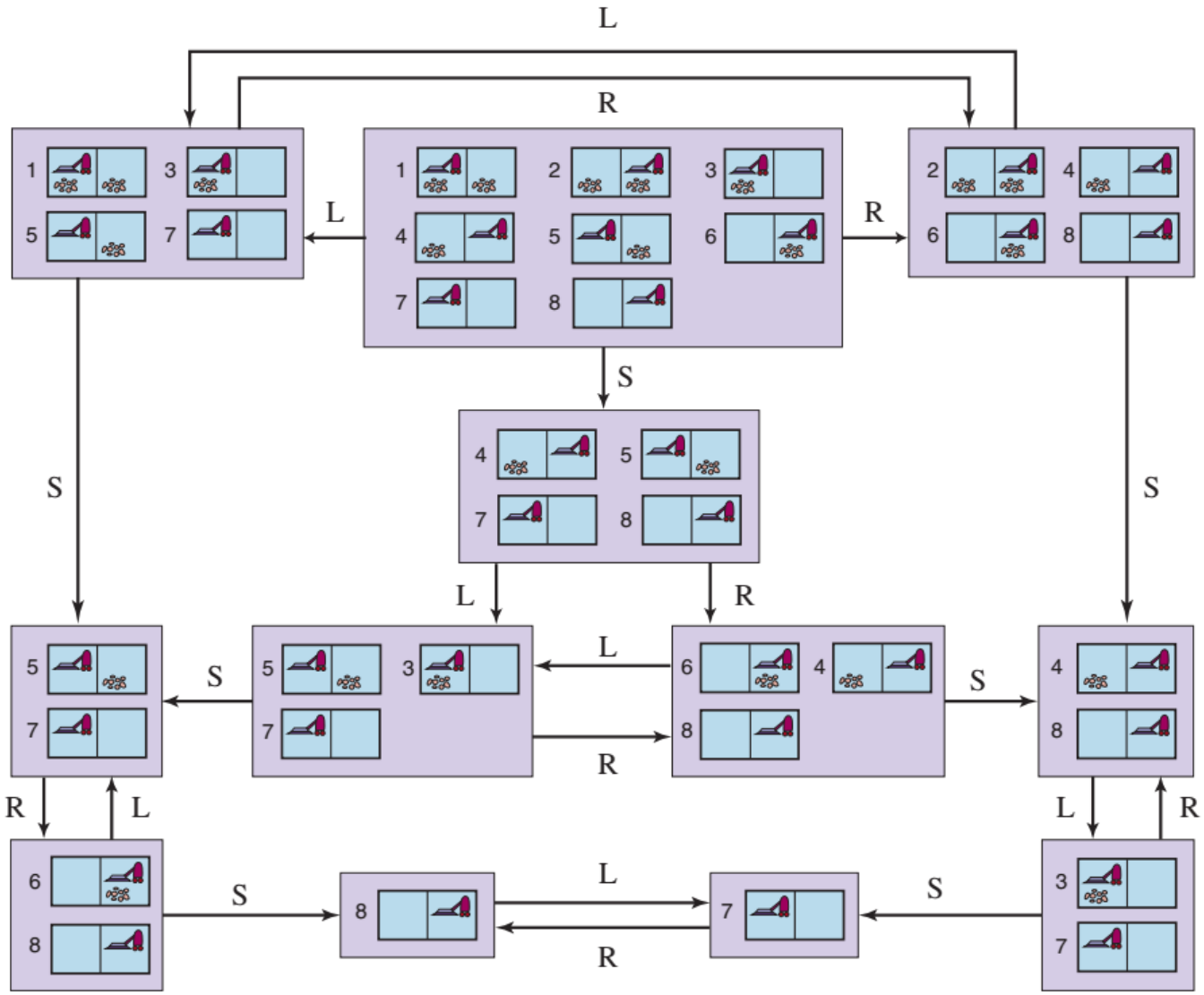
$$
\begin{aligned}
&Percept(Color\,(x, c)\,,\\
&\quad \textsc{Precond}{:}Object\,(x) \wedge InView\,(x)\\
&Percept(Color\,(can, c)\,,\\
&\quad \textsc{Precond}{:}Can\,(can) \wedge InView\,(can) \wedge Open\,(can)
\end{aligned}
$$

$$
\begin{aligned}
&Action(LookAt\,(x)\,,\\
&\quad \textsc{Precond}{:}InView\,(y) \wedge (x \neq y)\\
&\quad \textsc{Effect}{:}InView\,(x) \wedge \neg InView\,(y))
\end{aligned}
$$

Who will make a better plan?

Sensorless agent, contingency planning agent, online planning agent

Recall: Belief states in state-space search



Sensorless planning

- No percepts - plans without seeing colors of paint, cans, chair, table, e.g. $[RemoveLid(Can_1), Paint(Chair, Can_1), Paint(Table, Can_1)]$
- Open-world assumption $\Longrightarrow$ a belief state corresponds to the set of possible worlds that satisfy the formula representing it.
- All belief states (implicitly) include unchanging facts (invariants), e.g.: $Object(Table) \wedge Object(Chair) \wedge Can(C_1) \wedge Can(C_2)$
- Initial belief state includes facts that are part of the agent's domain knowledge, e.g.
  Objects and cans have colors: $b_0 = Color(x, C(x))$

Sensorless planning

1. Apply actions where $b \models Precond(a)$
2. Compute new belief state:

$$b' = \textsc{Result}(b, a) = \{s' : s' = \textsc{Result}_P(s, a) \text{ and } s \in b\}$$

   2.1 Set to false atoms that appears in Del(a)

   2.2 Set to true atoms that appears in Add(a)

Sensorless planning - Example

$b_0 : Color(x, C(x))$

Apply $RemoveLid(Can_1)$ in b_0 and obtain:
$b_1 : Color(x, C(x)) \wedge Open(Can1)$

Apply $Paint(Chair, Can_1)$ in b_1 using $\{x = Can_1, c = C(Can_1)\}$:
$b_2 : Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1))$

Apply $Paint(Table, Can_1)$ in b_2 :
$b_3 : Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1)) \wedge$
$Color(Table, C(Can_1))$

b_3 Satisfies the goal: $b_3 \models Color(Table, c) \wedge Color(Chair, c)$
The plan: $[RemoveLid(Can_1); Paint(Chair, Can_1); Paint(Table, Can_1)]$

Sensorless planning - Conditional effects

Actions where effect depends on the state?

- Fluent depend on other fluents.
- Not 1-CNF!

Example:

$$Action(Suck,$$
$$\textsc{Effect}: \textbf{when}\ AtL : CleanL \wedge \textbf{when}\ AtR : CleanR).$$

Result:

$$(AtL \wedge CleanL) \vee (AtR \wedge CleanR)$$

Wiggly belief state!

How to determine if we achived the goal?

Sensorless planning - Keep belief state simple

$$[Right, Suck, Left, Suck]$$

$$b_0 = True$$
$$b_1 = AtR$$
$$b_2 = AtR \wedge CleanR$$
$$b_3 = AtL \wedge CleanR$$
$$b_4 = AtL \wedge CleanR \wedge CleanL$$

Real-life: We always try to eliminate uncertainty.

Sensorless planning - Don't update belief state

$$\text{“}b_0 \text{ then } [a_1, \ldots, a_m]\text{.”}$$

Use SAT solver to determine:

$$b_0 \wedge A_m \models G_m$$

Sensorless planning
Use subset of belief state as heuristic

Admissible, solving subset of a belief state is easier:

$$\text{if } b_1 \subseteq b_2 \text{ then } h^*(b_1) \leq h^*(b_2).$$

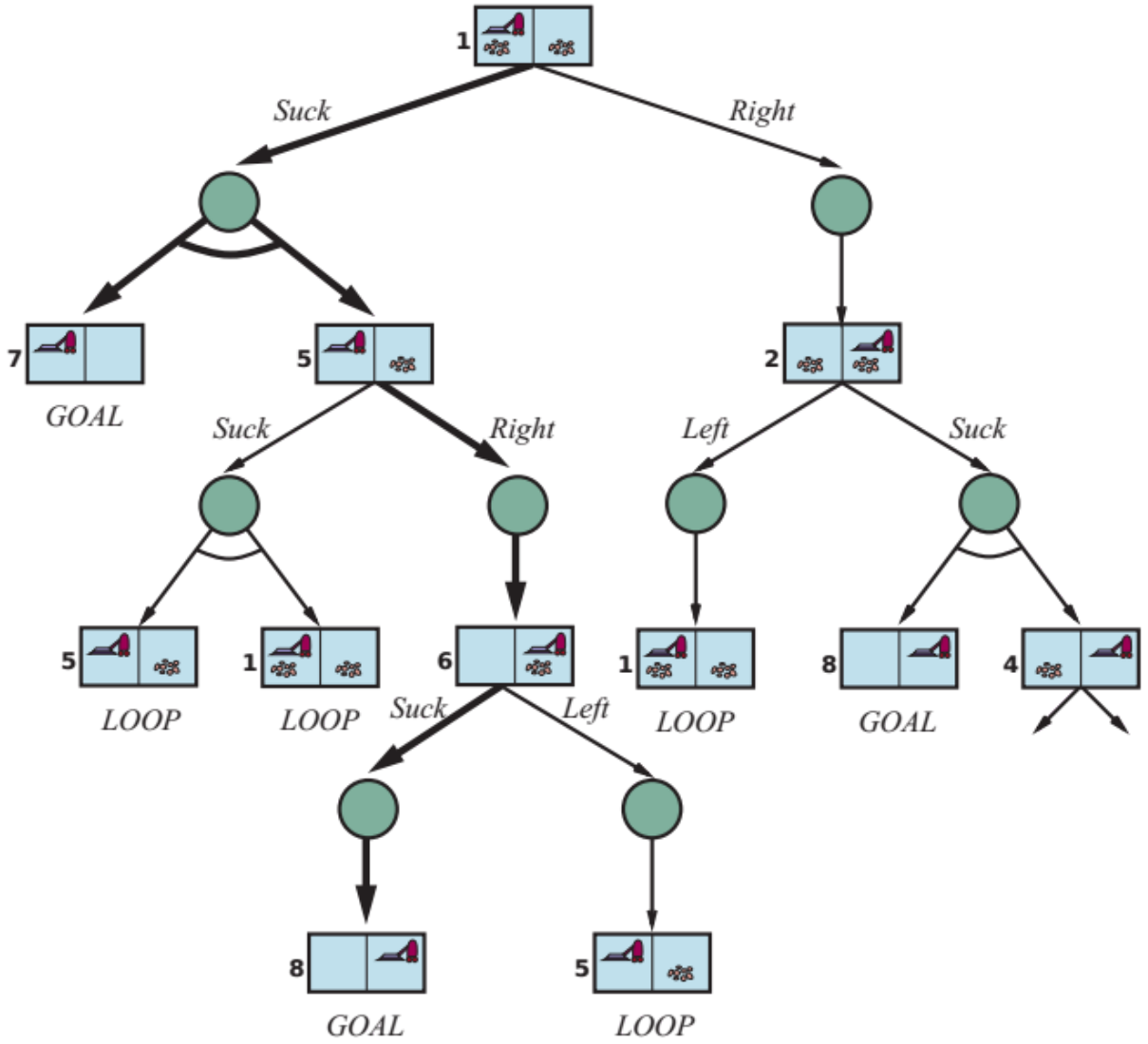
Combine heuristic values of individual states in belif state:

$$H(b) = \max\{h(s_1), \ldots, h(s_N)\}$$

Contingent planning

For partial observable and/or non-deterministic environments.

AND-OR search trees.



Contingent planning - Example

Given a chair and a table, the goal is to have them of the same colour. In the initial state we have two cans of paint, but the colours of the paint and the chair are unknown.

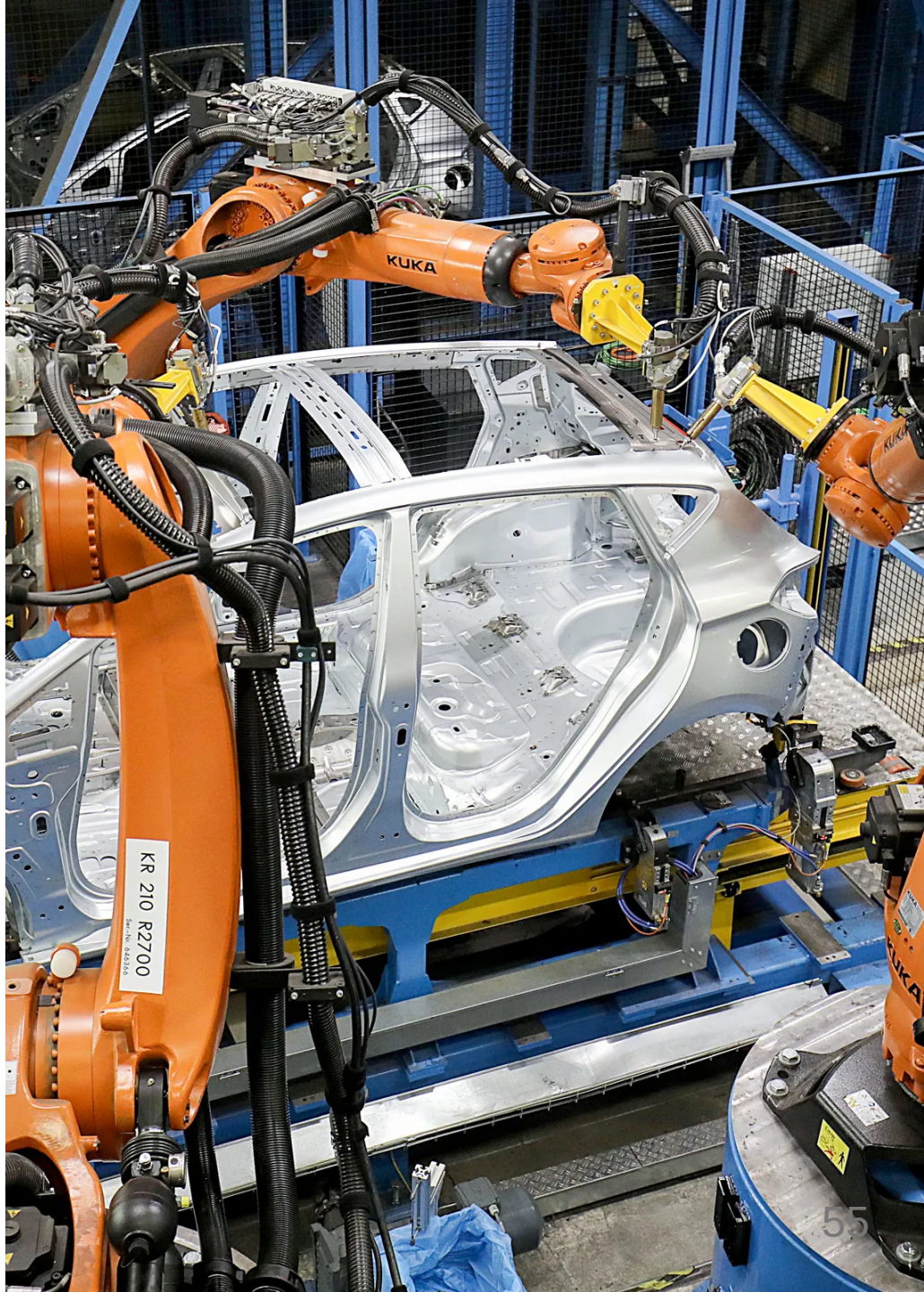
$$
\begin{aligned}
&[LookAt\,(Table)\,, LookAt\,(Chair)\,,\\
&\ \textbf{if}\ Color\,(Table, c) \wedge Color\,(Chair, c)\ \textbf{then}\ NoOp\\
&\quad \textbf{else}\,[RemovedLid\,(Can_1)\,, LookAt\,(Can_1)\,, RemovedLid\,(Can_2)\,, LookAt\,(Can_2)\,,\\
&\qquad \textbf{if}\ Color\,(Table, c) \wedge Color\,(can, c)\ \textbf{then}\ Paint\,(Chair, can)\\
&\qquad \textbf{else if}\ Color\,(Chair, c) \wedge Color\,(can, c)\ \textbf{then}\ Paint\,(Table, can)\\
&\qquad \textbf{else}\ [Paint\,(Chair, Can_1)\,, Paint\,(Table, Can_1)]]]
\end{aligned}
$$

Online planning

Execution monitoring to determine the need fo a new plan - **replanning**.

Incorrect model of the world:

- Misisng precondition
- Missing effect
- Missing fluent
- Exogenous events

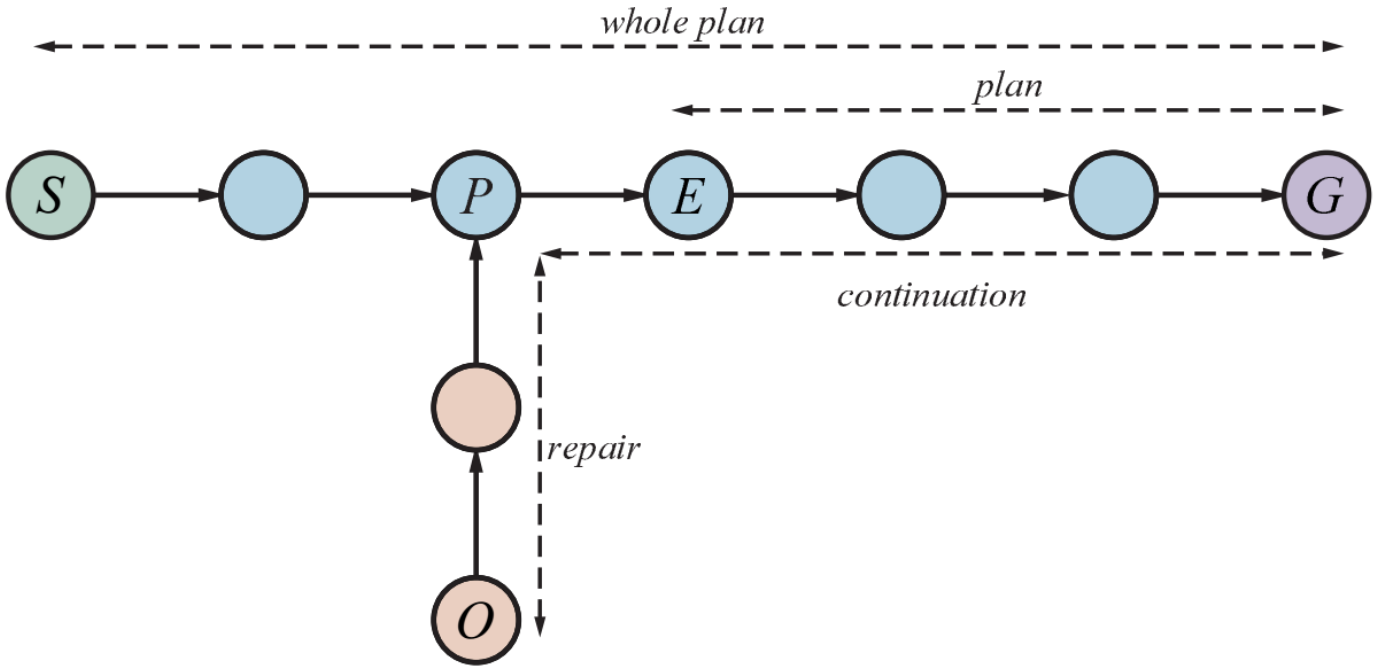


Online planning - monitoring approaches

Action monitorin - check that all preconditions hold

Plan monitoring - check the plan holds

Goal monitoring - check if there is a better goal



Questions?