Planning

- Planning Examples
- State space search
  - Forward
  - Backward
  - Heuristic
- Partial order planning

Monkeys can plan

Why shouldn't computers?

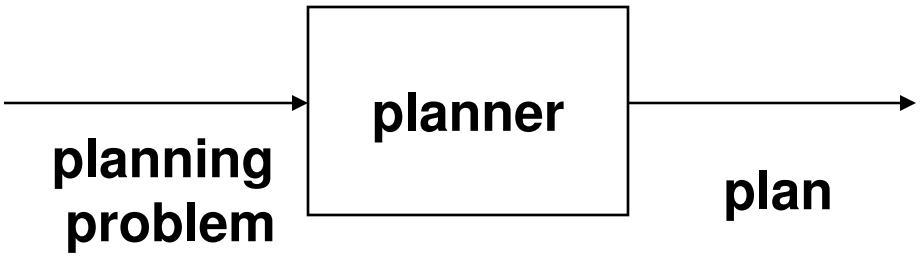
The problem statement: A monkey is in a laboratory room containing a box, a knife and a bunch of bananas. The bananas are hanging from the ceiling out of the reach of the monkey. How can the monkey obtain the bananas?

Motivating reasons

- **Planning is a component of intelligent behavior**
- **Lots of applications**

What is planning?

- A ***planner*** is a system that finds a sequence of actions to accomplish a specific task
- A planner synthesizes a plan



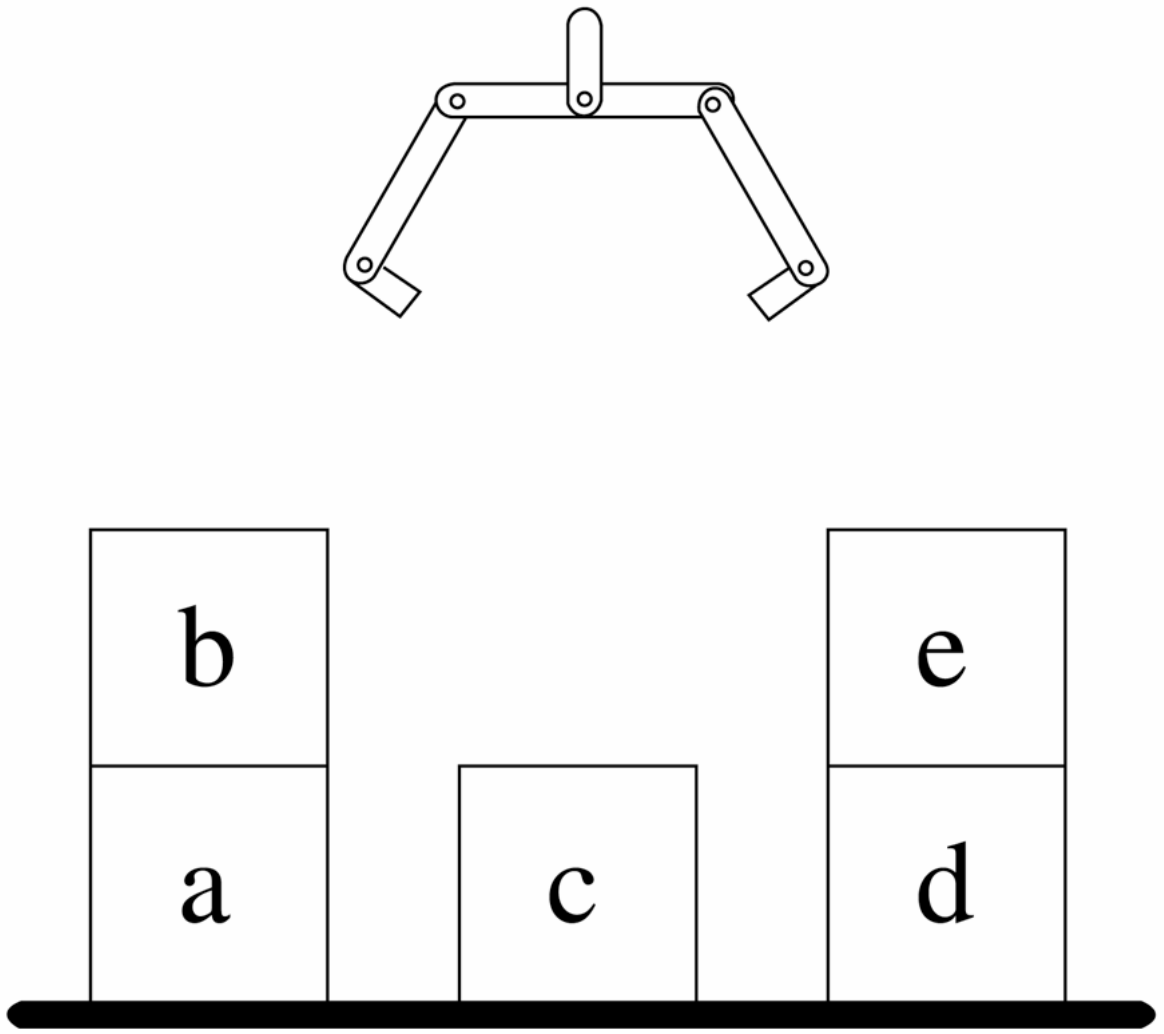
What is planning? (cont'd)

- **The main components of a *planning problem* are:**
  - **a description of the starting situation (*the initial state*), (the world now)**
  - **a description of the desired situation (*the goal state*), (how should the world be)**
  - **the actions available to the executing agent (*operator library*, a.k.a. *domain theory*) (possible actions to change the world)**

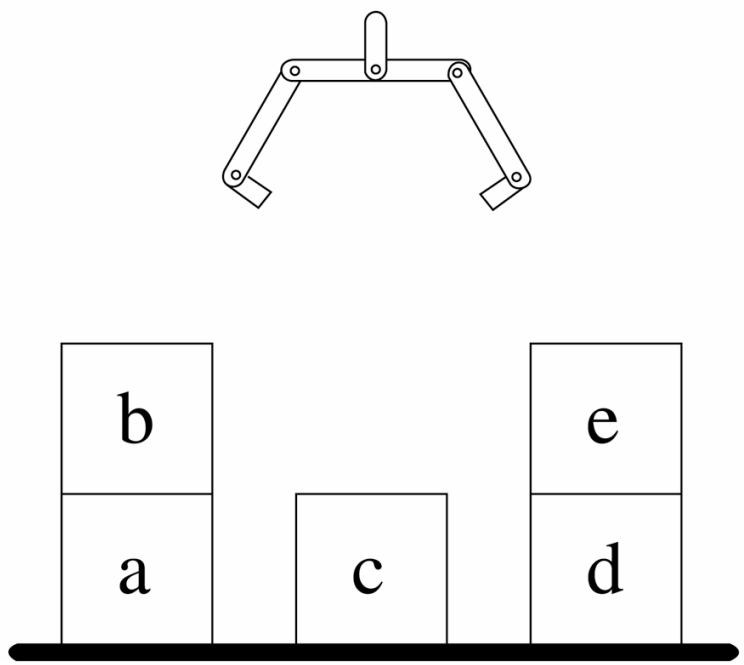
- **Formally, a (classical) planning problem is a triple: <I, G, D>**

where, I is the initial state,
G is the goal state, and
D is the domain theory.

The blocks world



Represent this world using predicates



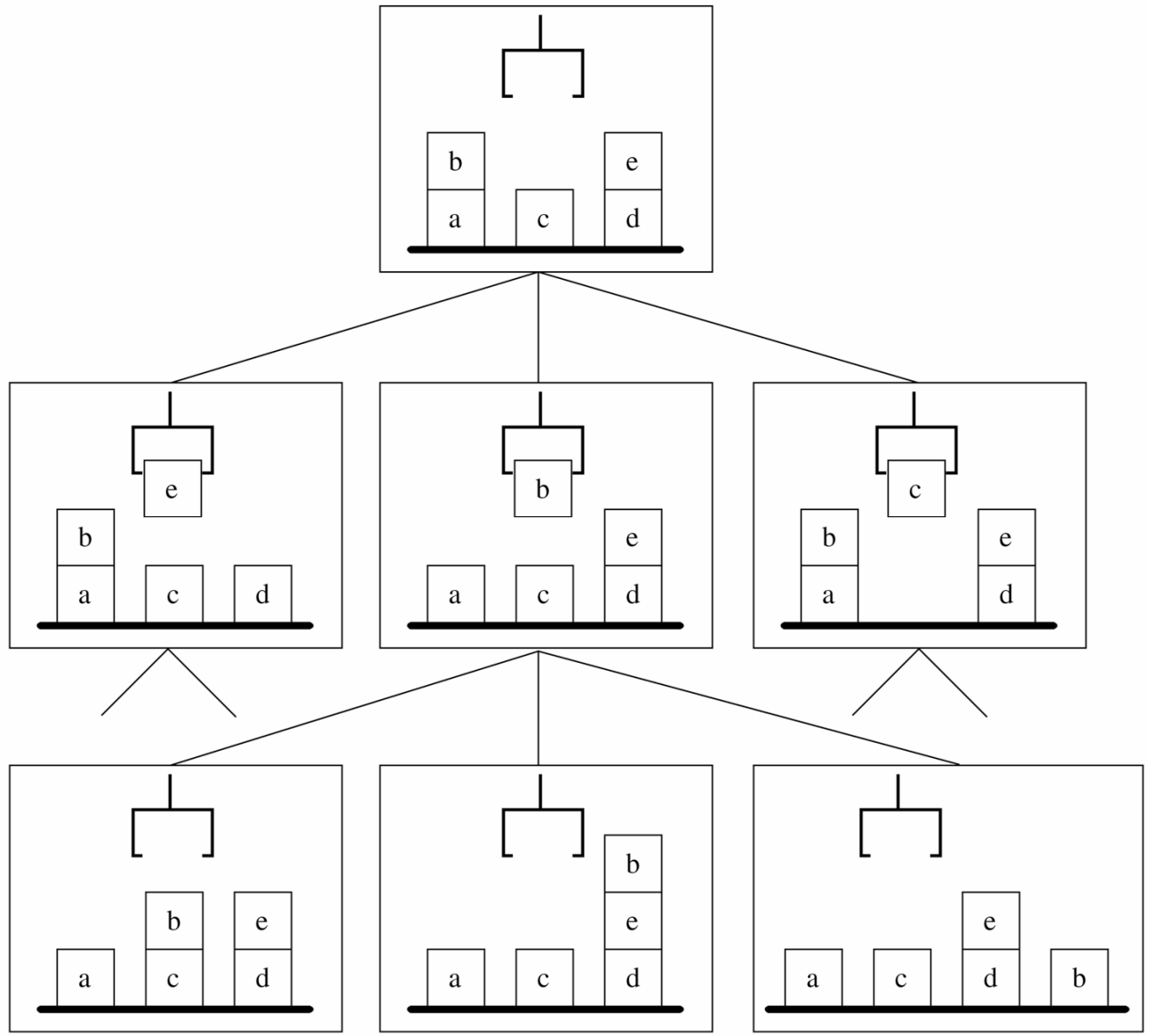
ontable(a)
ontable(c)
ontable(d)
on(b,a)
on(e,d)
clear(b)
clear(c)
clear(e)
gripping()

The robot arm can perform these tasks

- **pickup (W)**: pick up block W from its current location on the table and hold it
- **putdown (W)**: place block W on the table
- **stack (U, V)**: place block U on top of block V
- **unstack (U, V)**: remove block U from the top of block V and hold it

All assume that the robot arm can precisely reach the block.

Portion of the search space of the blocks world example



The STRIPS representation

Special purpose representation.

An operator is defined in terms of its:

- **name,**
- **parameters,**
- **preconditions, and**
- **results.**

A planner is a special purpose algorithm, i.e., it's not a general purpose logic theorem prover.

Four operators for the blocks world

pickup(X)	**P: gripping() ∧ clear(X) ∧ ontable(X)** **A: gripping(X)** **D: ontable(X) ∧ gripping()**
putdown(X)	**P: gripping(X)** **A: ontable(X) ∧ gripping() ∧ clear(X)** **D: gripping(X)**
stack(X,Y)	**P: gripping(X) ∧ clear(Y)** **A: on(X,Y) ∧ gripping() ∧ clear(X)** **D: gripping(X) ∧ clear(Y)**
unstack(X,Y)	**P: gripping() ∧ clear(X) ∧ on(X,Y)** **A: gripping(X) ∧ clear(Y)** **D: on(X,Y) ∧ gripping()**

Notice the simplification

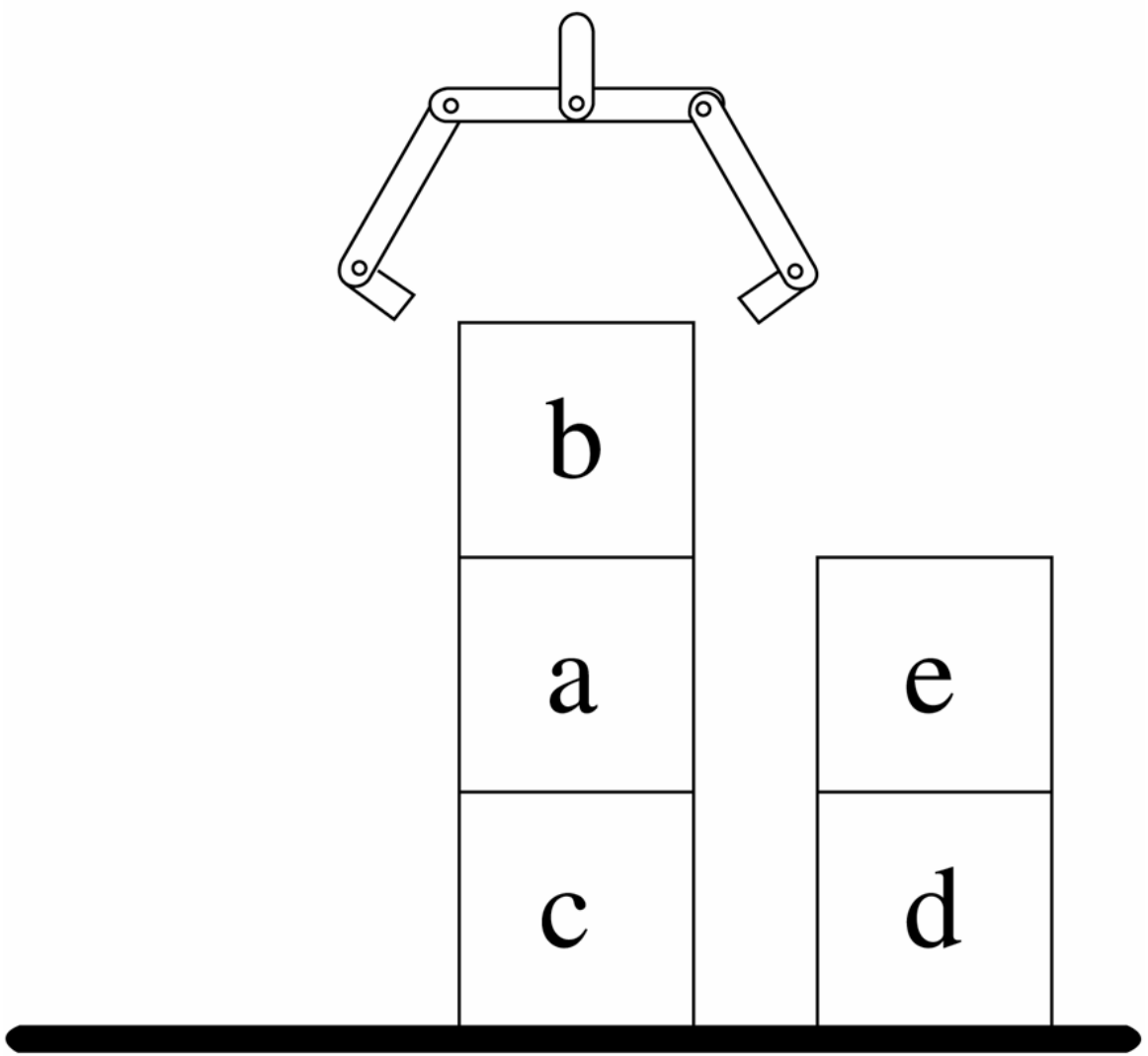
Preconditions, ***add lists***, and ***delete lists*** are all conjunctions. We don't have the full power of predicate logic.

The same applies to *goals*. Goals are conjunctions of predicates.

A detail:

Why do we have two operators for picking up (pickup and unstack), and two for putting down (putdown and stack)?

A goal state for the blocks world



A state space algorithm for STRIPS operators

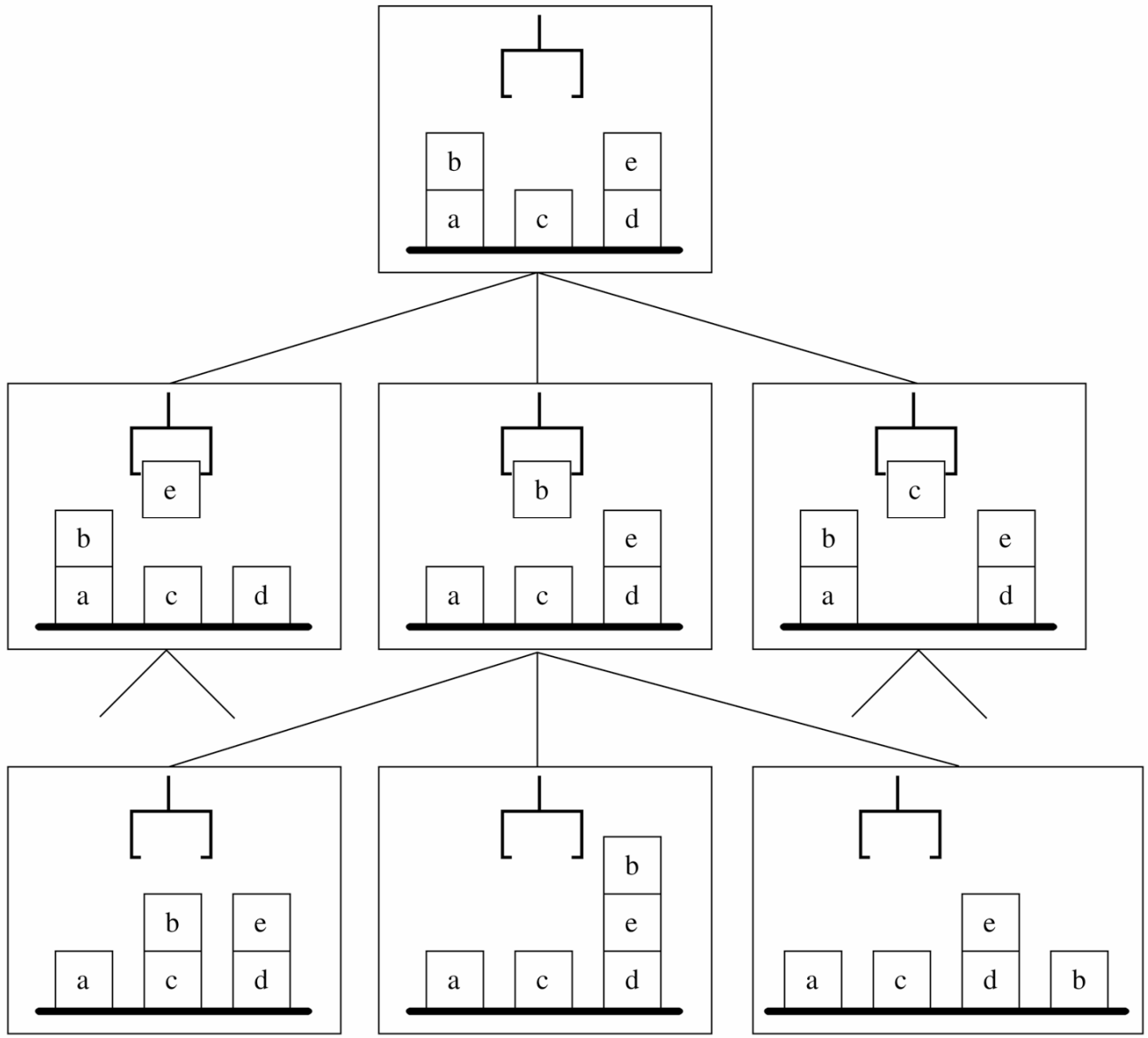
Search the space of situations (or states). This means each node in the search tree is a state.

The root of the tree is the start state.

Operators are the means of transition from each node to its children.

The goal test involves seeing if the set of goals is a subset of the current situation.

Now, the following graph makes much more sense



Planning algorithms

- **Forward chaining**
  - **Work in a state space**
  - **Start with the initial state, try to reach the goal state using forward progression**
- **Backward chaining**
  - **Work in a state space**
  - **Start with the goal state, try to reach the initial state using backward regression**
- **Partial order planning**
  - **Work in a plan space**
  - **Start with an empty plan, work from the goal to reach a complete plan**

State space search

Search the space of states

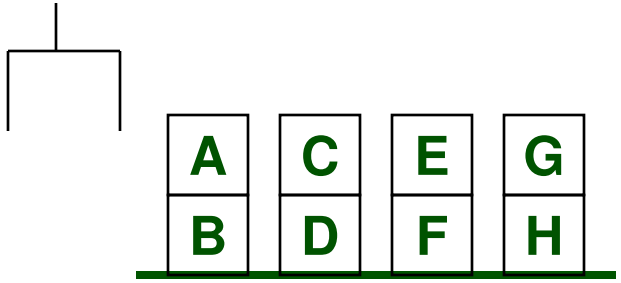
- Initial state, goal test, step cost, etc.
- Actions are the transitions between state

Actions are invertible (why?)

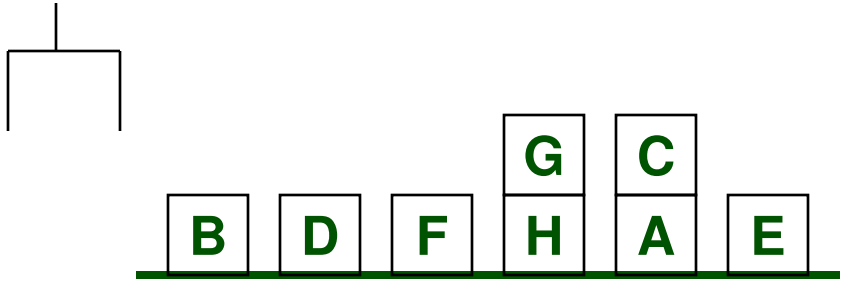
- Move forward from the initial state: Forward State-Space Search or Progression Planning
- Move backward from goal state: Backward State-Space Search or Regression Planning

Forward chaining

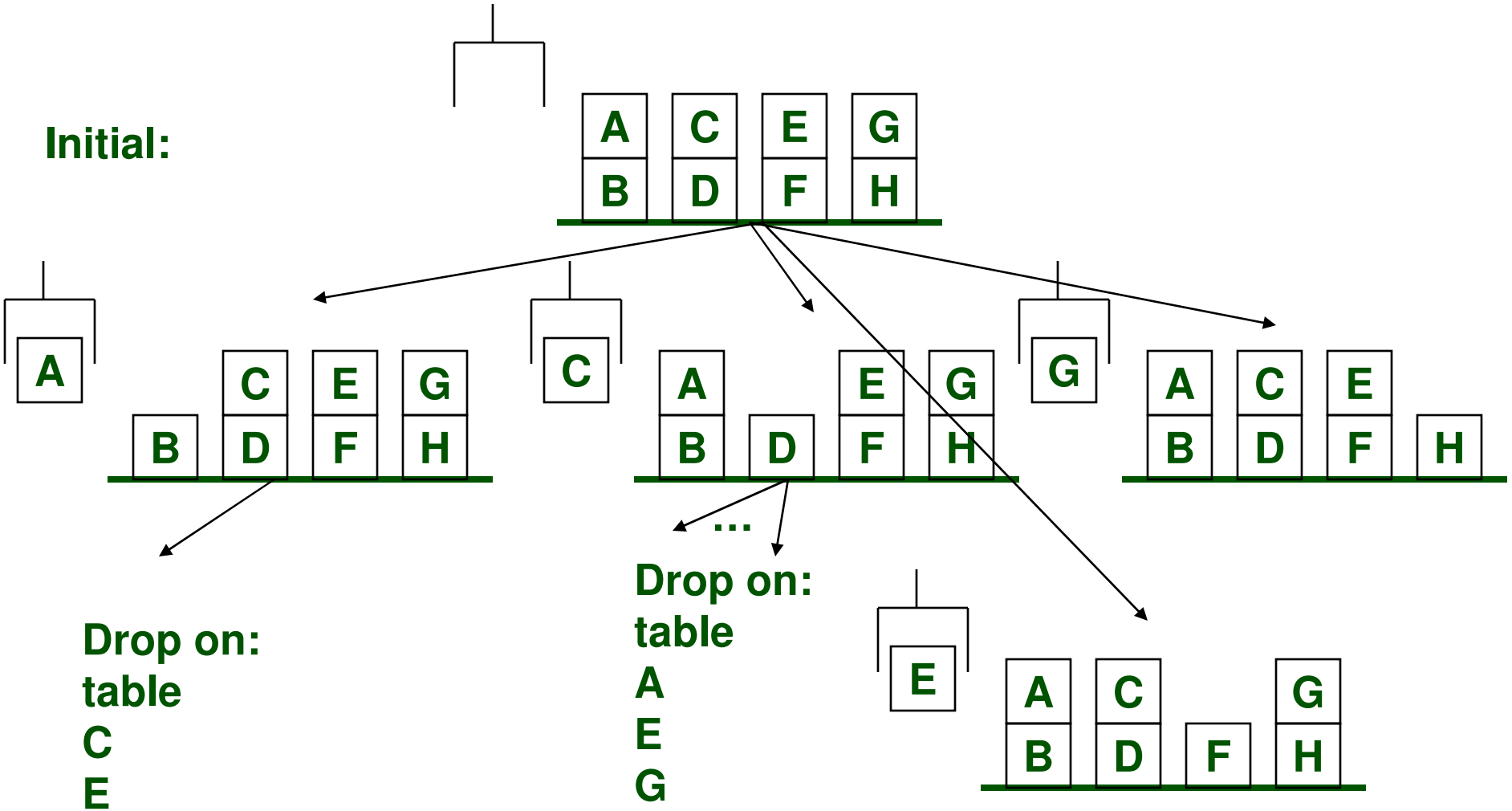
Initial:



Goal :



1st and 2nd levels of search



Results

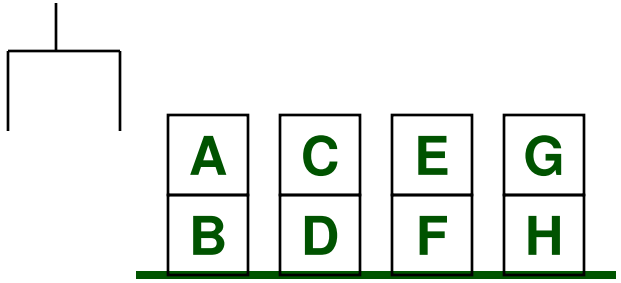
- **A plan is:**
  - **unstack (A, B)**
  - **putdown (A)**
  - **unstack (C, D)**
  - **stack (C, A)**
  - **unstack (E, F)**
  - **putdown (F)**
- **Notice that the final locations of D, F, G, and H need not be specified**
- **Also notice that D, F, G, and H will never need to be moved. But there are states in the search space which are a result of moving these. Working backwards from the goal might help.**

Contd….

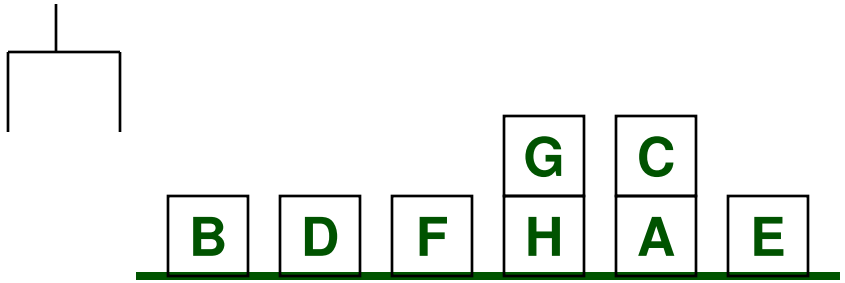
- Progression planning
- More than one goal states
- Infinite branching factor
- Inefficient search – heuristics are required

Backward chaining

Initial:



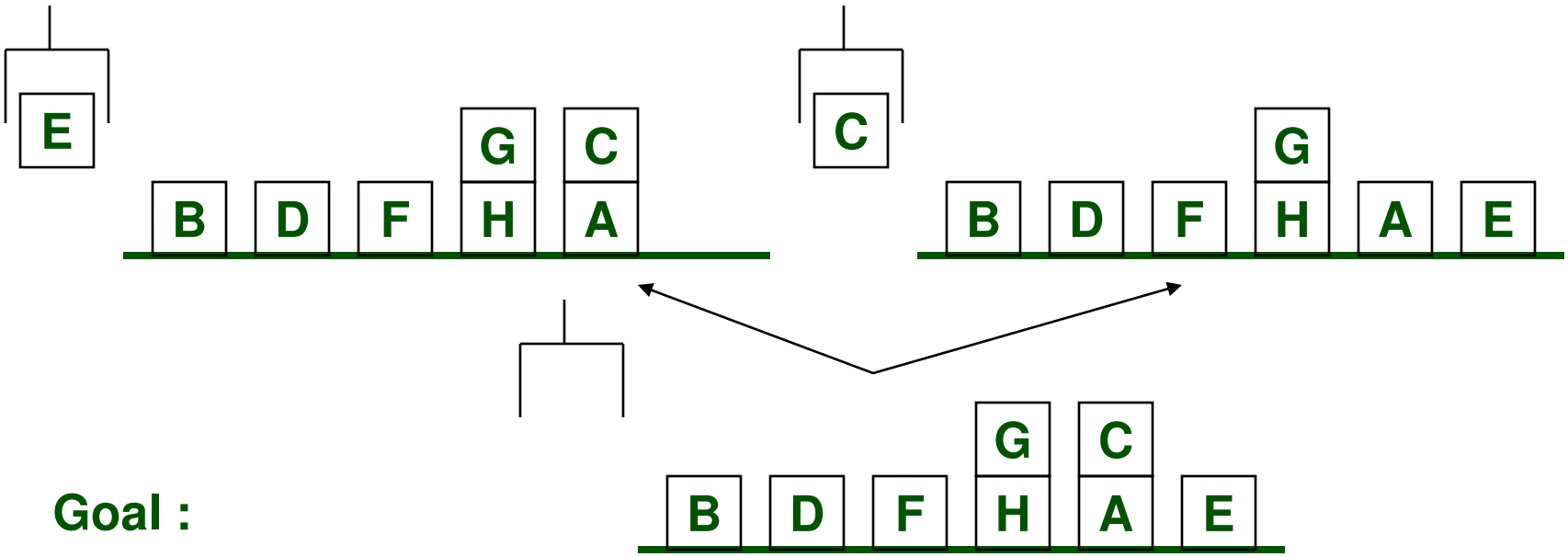
Goal :



1st level of search

For E to be on the table, the last action must be putdown(E)

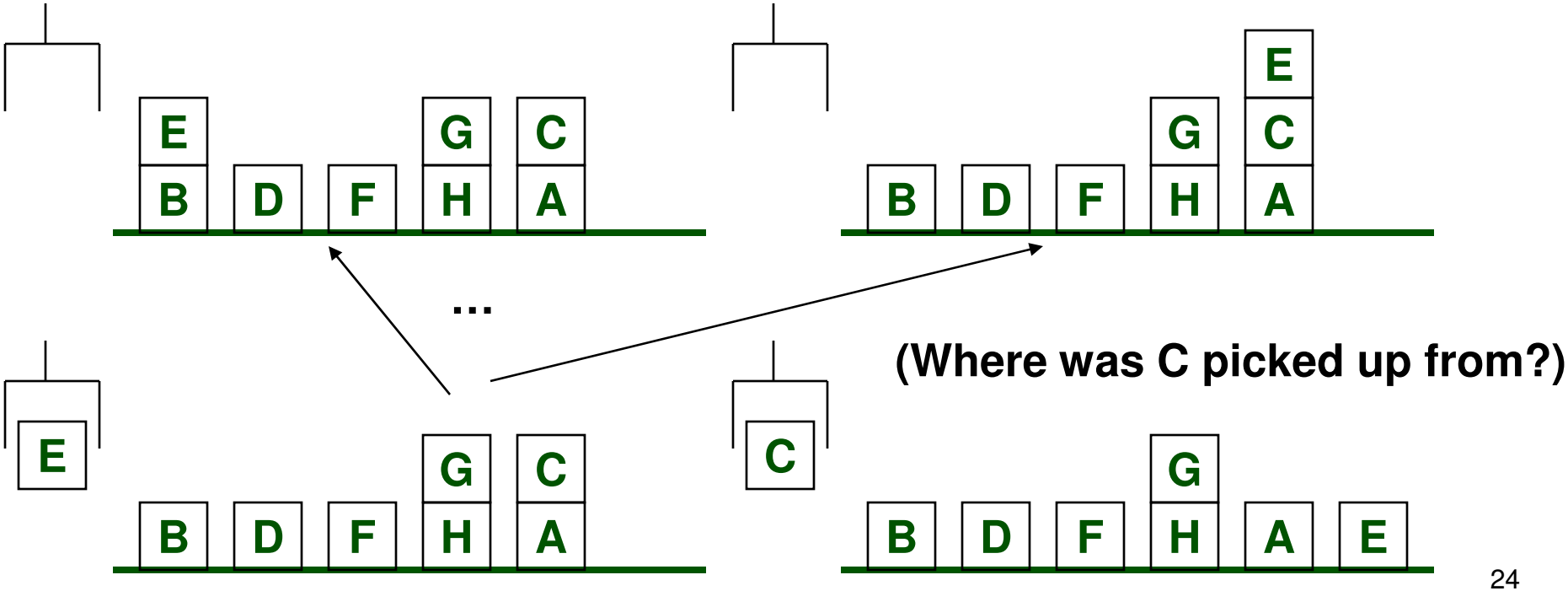
For C to be on A, the last action must be stack(C,A)



Goal :

2nd level of search

Where was E picked up from?



Results

- The same plan can be found
  - unstack (A, B)
  - putdown (A)
  - unstack (C, D)
  - stack (C, A)
  - unstack (E, F)
  - putdown (F)
- Now, the final locations of D, F, G, and H need to be specified
- Notice that D, F, G, and H will never need to be moved. But observe that from the second level on the branching factor is still high

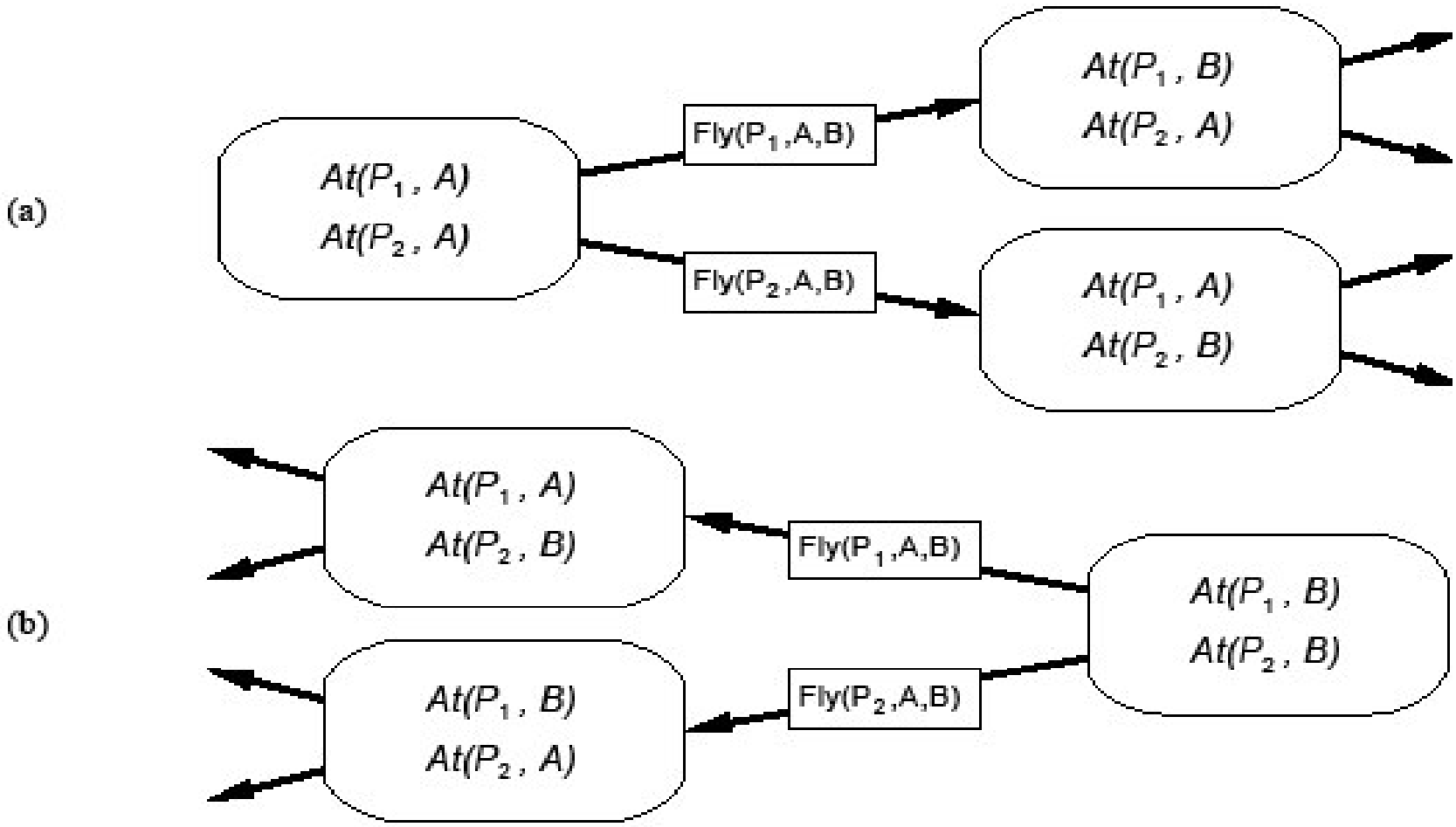
Contd….

- Regression planning
- Less branching factor
- Relevant actions are traced
- Consistency is maintained
- Predecessor is identified from G on action A:

➢ Each Precondition literal of A is added, unless it already appears.

➢ Any positive effect of A that appears in G are deleted

State space search - Example



Heuristic to Speed up Search

We can use A*, but we need an admissible heuristic

1. Divide-and-conquer: sub-goal independence assumption
   - Problem relaxation by removing
2. … all preconditions
3. … all preconditions and negative effects
4. … negative effects only: Empty-Delete-List

1. Subgoal Independence Assumption

The cost of solving a conjunction of subgoals is the sum of the costs of solving each subgoal independently

Optimistic

- **Where subplans interact negatively**
- **Example: one action in a subplan delete goal achieved by an action in another subplan**

Pessimistic (not admissible)

- **Redundant actions in subplans can be replaced by a single action in merged plan**

2. Problem Relaxation: Removing Preconditions

Remove preconditions from action descriptions

- **All actions are applicable**
- **Every literal in the goal is achievable in one step**

Number of steps to achieve the conjunction of literals in the goal is equal to the number of unsatisfied literals

Alert

- **Some actions may achieve several literals**
- **Some action may remove the effect of another action**

3. Remove Preconditions & Negative Effects

Considers only positive interactions among actions to achieve multiple subgoals

The minimum number of actions required is the sum of the union of the actions' positive effects that satisfy the goal

The problem is reduced to a set cover problem, which is NP-hard

- **Approximation by a greedy algorithm cannot guarantee an admissible heuristic**

4. Removing Negative Effects (Only)

Remove all negative effects of actions (no action may destroy the effects of another)

Known as the Empty-Delete-List heuristic

Requires running a simple planning algorithm

Quick & effective

Usable in progression or regression planning

Partial-order planning (POP) - Blocks world

- Notice that the resulting plan has two parallelizable threads:

unstack (A,B)
putdown (A)
unstack (C,D)
stack (C,A)

&

unstack (E, F)
putdown (F)

- These steps can be interleaved in 3 different ways:

unstack (E, F)
putdown (F)
unstack (A,B)
putdown (A)
unstack (C,D)
stack (C,A)

unstack (A,B)
putdown (A)
unstack (E, F)
putdown (F)
unstack (C,D)
stack (C,A)

unstack (A,B)
putdown (A)
unstack (C,D)
stack (C,A)
unstack (E, F)
putdown (F)

Partial Order Planning (POP)- Shoe problem

State-space search

- **Yields totally ordered plans (linear plans)**

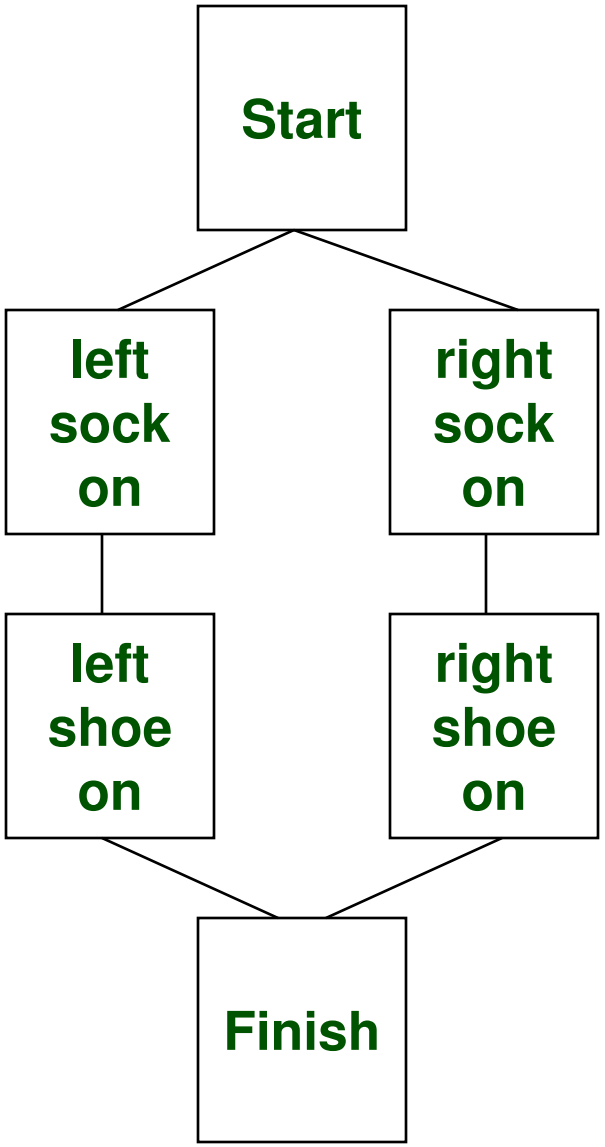
POP

- **Works on subproblems independently, then combines subplans**
- **Example**
  - Goal(RightShoeOn $\wedge$ LeftShoeOn)
  - Init()
  - *Action*(RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)
  - *Action*(RightSock, EFFECT: RightSockOn)
  - *Action*(LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)
  - *Action*(LeftSock, EFFECT: LeftSockOn)

Partial-order planning (cont'd)

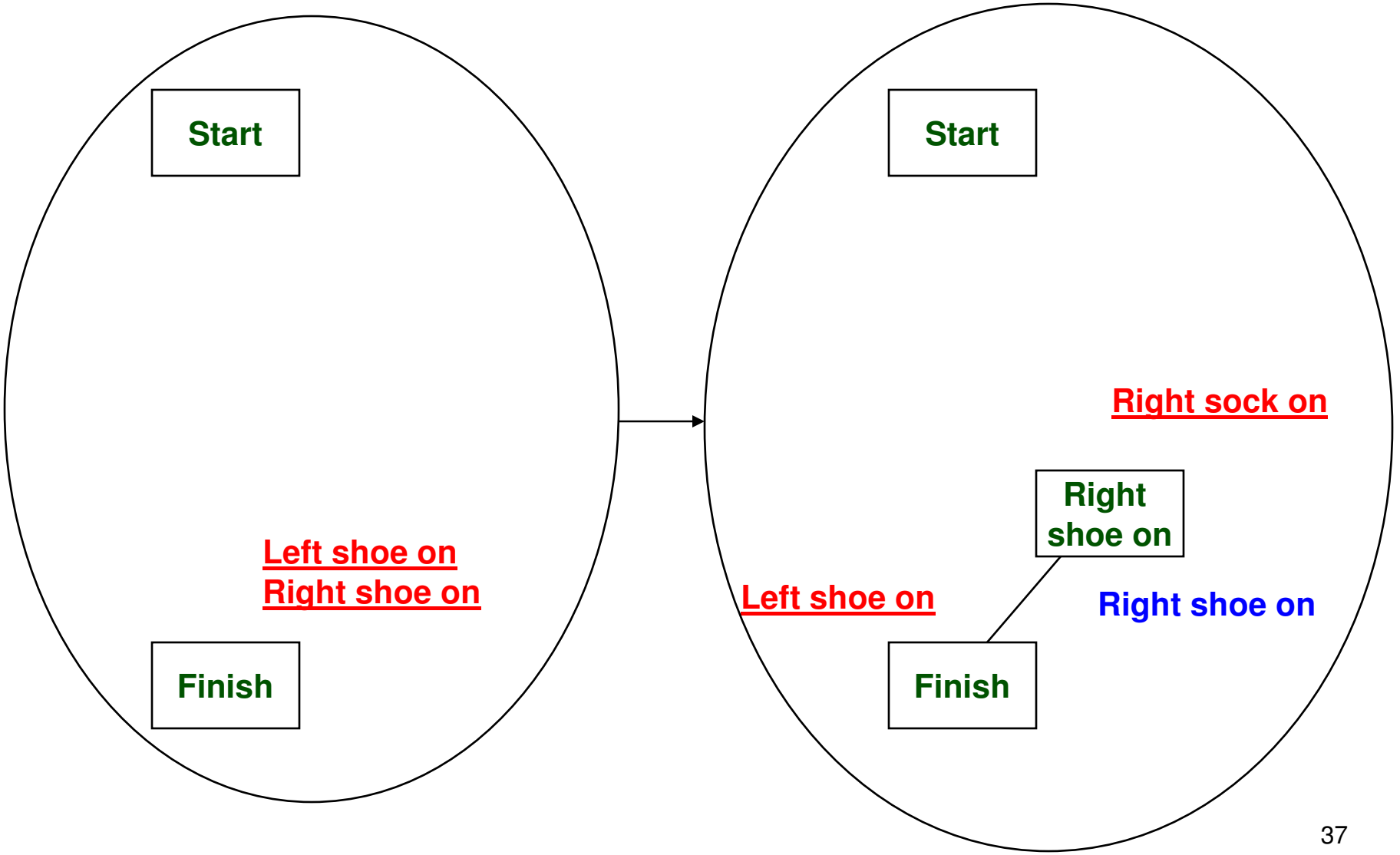
- **Idea: Do not order steps unless it is necessary**
- **Then a partially ordered plan represents several totally ordered plans**
- **That decreases the search space**
- **But still the planning problem is not solved, good heuristics are crucial**

Partial-order planning (cont'd)

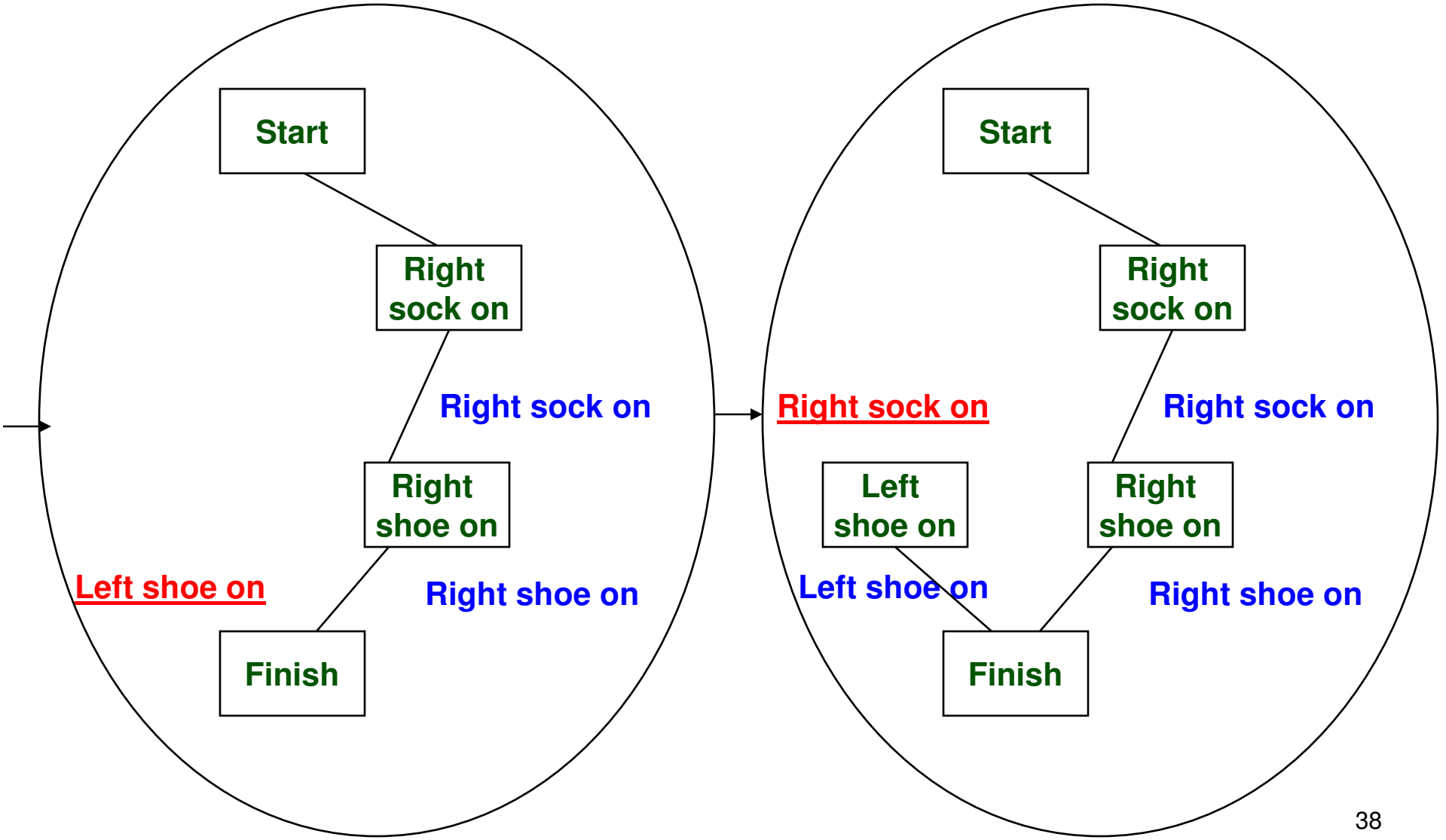


Start	Start	Start	Start	Start	Start
Left sock on	Right sock on	Left sock on	Right sock on	Left sock on	Right sock on
Left shoe on	Right shoe on	Right sock on	Left sock on	Right sock on	Left sock on
Right sock on	Left sock on	Left shoe on	Right shoe on	Right shoe on	Left shoe on
Right shoe on	Left shoe on	Right shoe on	Left shoe on	Left shoe on	Right shoe on
Finish	Finish	Finish	Finish	Finish	Finish

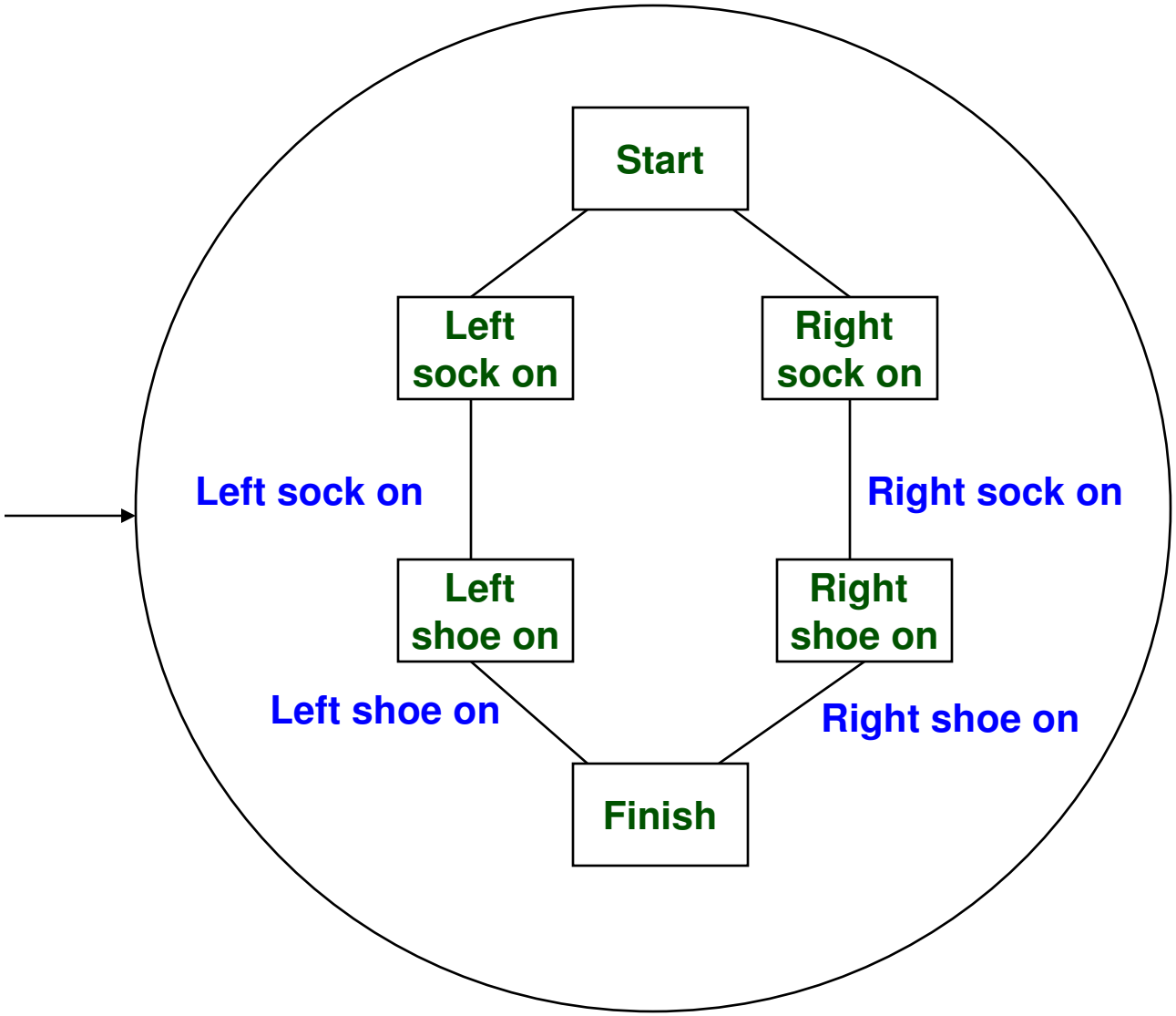
POP plan generation



POP plan generation (cont'd)



POP plan generation (cont'd)



DONE!

Components of a Plan (POP)

A set of **actions**

A set of **ordering constraints**

- A p B reads "A before B" but not necessarily immediately before B
- Alert: caution to cycles A p B and B p A

A set of **causal links** (protection intervals) between actions

- A $\xrightarrow{p}$ B reads "A achieves *p* for B" and p must remain true from the time A is applied to the time B is applied
- Example "RightSock $\xrightarrow{RightSockOn}$ RightShoe

A set of **open preconditions**

- Planners work to reduce the set of open preconditions to the empty set w/o introducing contradictions

Consistent Plan (POP)

Consistent plan is a plan that has

- **No cycle in the ordering constraints**
- **No conflicts with the causal links**

Solution

- **Is a consistent plan with no open preconditions**

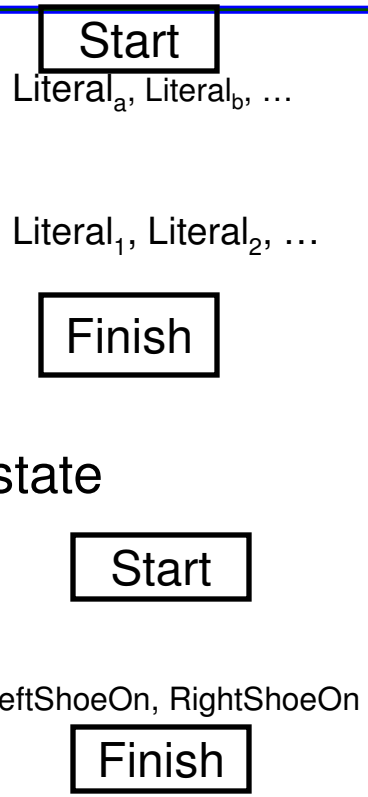
To solve a conflict between a causal link A $\xrightarrow{p}$ B and an action C (that clobbers, threatens the causal link), we force C to occur outside the "protection interval" by adding

- **the constraint C p A (demoting C) or**
- **the constraint B p C (promoting C)**

Setting up the PoP

Add dummy states

- **Start**
  - Has no preconditions
  - Its effects are the literals of the initial state
- **Finish**
  - Its preconditions are the literals of the goal state
  - Has no effects



Initial Plan:

- **Actions: {Start, Finish}**
- **Ordering constraints: {Start p Finish}**
- **Causal links: {}**
- **Open Preconditions: {LeftShoeOn,RightShoeOn}**

POP as a Search Problem

The successor function arbitrarily picks one open precondition *p* on an action B

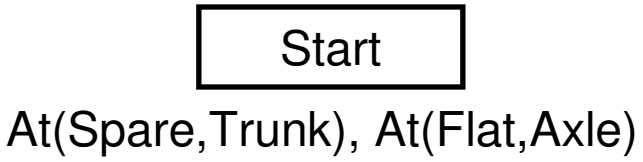
For every possible consistent action A that achieves *p*

- **It generates a successor plan adding the causal link A $\xrightarrow{p}$ B and the ordering constraint A p B**
- **If A was not in the plan, it adds Start p A and A p Finish**
- **It resolves all conflicts between**
  - the new causal link and all existing actions
  - between A and all existing causal links
- **Then it adds the successor states for combination of resolved conflicts**

It repeats until no open precondition exists

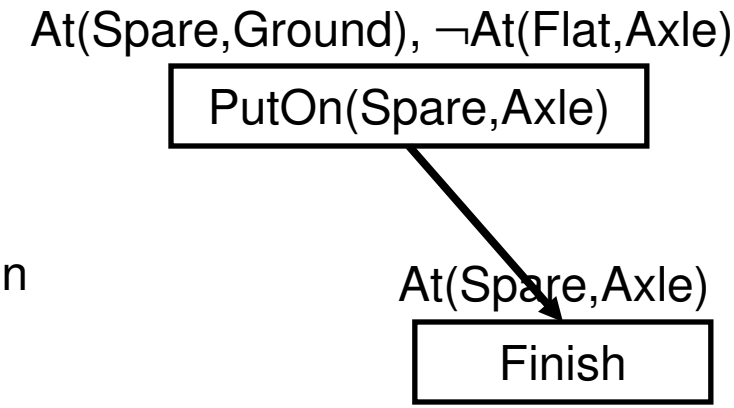
Example of POP: Flat tire problem

See problem description in Fig 11.7 page 391



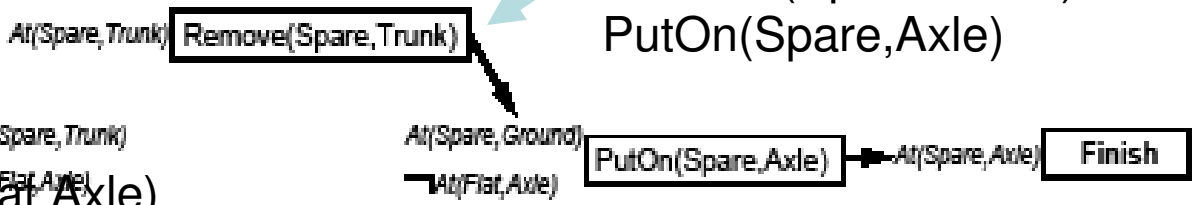
Only one open precondition

Only 1 applicable action

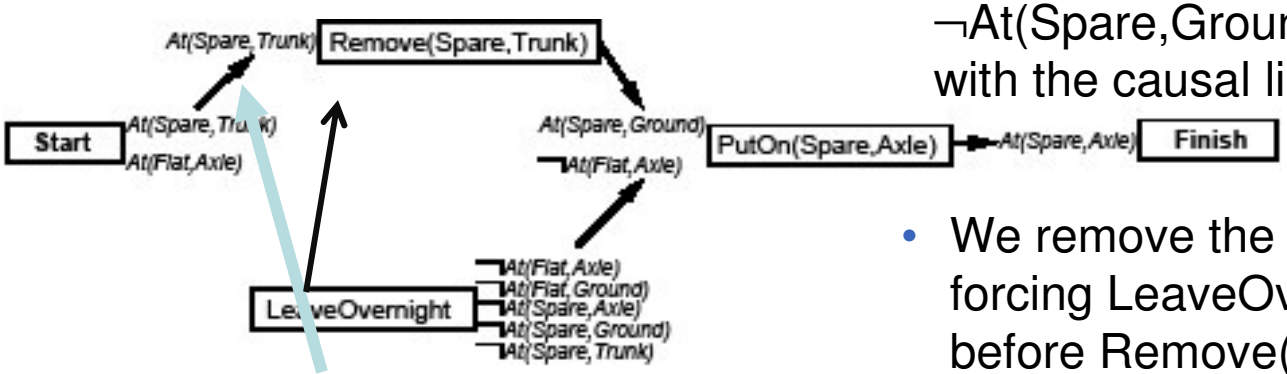


- Pick up At(Spare,Ground)
- Choose only applicable action Remove(Spare,Trunk)

Add causal link between Remove(Spare,Trunk) and PutOn(Spare,Axle)

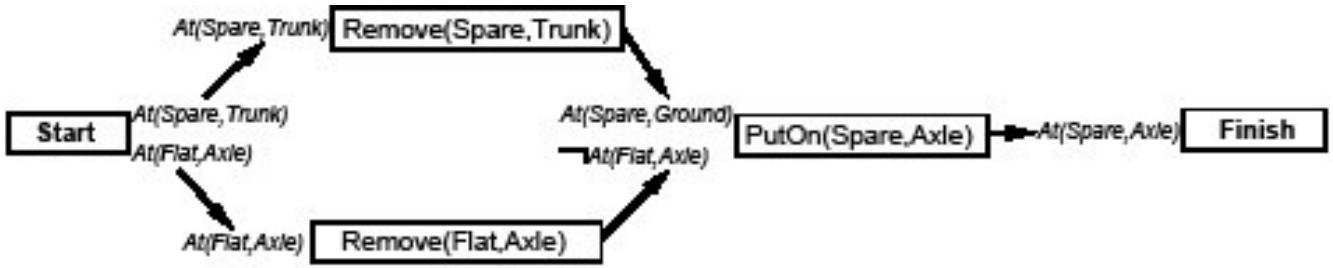


- Pick up ¬At(Flat,Axle)
- There are 2 applicable actions: LeaveOvernight and Remove(Flat,Axle)
- Choose LeaveOvernight



- LeaveOvernight has effect ¬At(Spare,Ground), which conflicts with the causal link
- We remove the conflict by forcing LeaveOvernight to occur before Remove(Spare,Trunk)

- Conflicts with effects of Remove(Spare,Trunk)
- The only way to resolve the conflict is to undo LeaveOvernightuse the action Remove(Flat,Axle)



- This time, we choose Remove(Flat,Axle)
- Pick up At(Spare,Trunk) and choose Start to achieve it
- Pick up At(Flat,Axle) and choose Start to achieve it.
- We now have a complete consistent partially ordered plan

POP Algorithm (1)

Backtrack when fails to resolve a threat or find an operator

Causal links

- **Recognize when to abandon a doomed plan without wasting time expanding irrelevant part of the plan**
- **allow early pruning of inconsistent combination of actions**

When actions include variables, we need to find appropriate substitutions

- **Typically we try to delay commitments to instantiating a variable until we have no other choice (least commitment)**

POP is sound, complete, and systematic (no repetition)

POP Algorithm (2)

Decomposes the problem (advantage)

But does not represent states explicitly: it is hard to design heuristic to estimate distance from goal

- **Example: Number of open preconditions – those that match the effects of the start node. Not perfect (same problems as before)**

A heuristic can be used to choose which plan to refine (which precondition to pick-up):

- **Choose the most-constrained precondition, the one satisfied by the least number of actions. Like in CSPs!**
- **When no action satisfies a precondition, backtrack!**
- **When only one action satisfies a precondition, pick up the precondiction.**

Comments on planning

- It is a synthesis task
- Classical planning is based on the assumptions of a deterministic and static environment
- Theorem proving and situation calculus are not widely used nowadays for planning
- Algorithms to solve planning problems include:
  - forward chaining: heuristic search in state space
  - Graphplan: mutual exclusion reasoning using plan graphs
  - Partial order planning (POP): goal directed search in plan space
  - Satifiability based planning: convert problem into logic

Comments on planning (cont'd)

- **Non-classical planners include:**
  - **probabilistic planners**
  - **contingency planners (a.k.a. conditional planners)**
  - **decision-theoretic planners**
  - **temporal planners**
  - **resource based planners**

Comments on planning (cont'd)

- **In addition to plan generation algorithms we also need algorithms for**
  - **Carrying out the plan**
  - **Monitoring the execution**
    (because the plan might not work as expected; or the world might change)
    (need to maintain the consistency between the world and the program's internal model of the world)
  - **Recovering from plan failures**
  - **Acting on new opportunities that arise during execution**
  - **Learning from experience**
    (save and generalize good plans)

Applications of planning

- **Robotics**
  - **Shakey, the robot at SRI was the initial motivator**
  - **However, several other techniques are used for path-planning etc.**
  - **Most robotic systems are *reactive***
- **Games**

The story is a plan and a different one can be constructed for each game

- **Web applications**

Formulating query plans, using web services

- **Crisis response**

Oil spill, forest fire, emergency evacuation

Applications of planning (cont'd)

- **Space**

Autonomous spacecraft, self-healing systems

- **Device control**

Elevator control, control software for modular devices

- **Military planning**
- **And many others …**