



Planning

Introduction to Artificial Intelligence

Hadi Moradi

moradi@usc.edu



Planning

- Search vs. planning
- STRIPS operators
- Partial-order planning



What do we have so far?

- Can TELL KB about new percepts about the world
- KB maintains model of the current world state
- Can ASK KB about any fact that can be inferred from KB
- How can we use these components to build a **planning agent**,

i.e., an agent that constructs plans that can achieve its goals, and that then executes these plans?

Example: Robot Manipulators

- Example: (courtesy of Martin Rohrmeier)
 - Puma 560
 - Kr6



Remember: Problem-Solving Agent

```
function SIMPLE-PROBLEM-SOLVING-AGENT(p) returns an action
  inputs: p, a percept
  static: s, an action sequence, initially empty
           state, some description of the current world state
           g, a goal, initially null
           problem, a problem formulation

  state ← UPDATE-STATE(state, p)
  if s is empty then
    g ← FORMULATE-GOAL(state)
    problem ← FORMULATE-PROBLEM(state, g)
    s ← SEARCH(problem)
  action ← RECOMMENDATION(s, state)
  s ← REMAINDER(s, state)
  return action
```

Note: This is *offline* problem-solving. *Online* problem-solving involves acting w/o complete knowledge of the problem and environment

A Simple Planning Agent

function SIMPLE-PLANNING-AGENT(percept) **returns** an action

static: KB, a knowledge base (includes action descriptions)
p, a plan (initially, NoPlan)
t, a time counter (initially 0)

local variables: G, a goal
current, a current state description

TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))

current ← STATE-DESCRIPTION(KB, t)

given percept, return initial state description in format required by planner

if p = NoPlan **then**

G ← ASK(KB, MAKE-GOAL-QUERY(t))

used to ask KB what next goal should be

p ← IDEAL-PLANNER(current, G, KB)

Given a goal, generate plan of action

if p = NoPlan **or** p is empty **then**

action ← NoOp

else

action ← FIRST(p)

p ← REST(p)

Like popping from a stack

TELL(KB, MAKE-ACTION-SENTENCE(action, t))

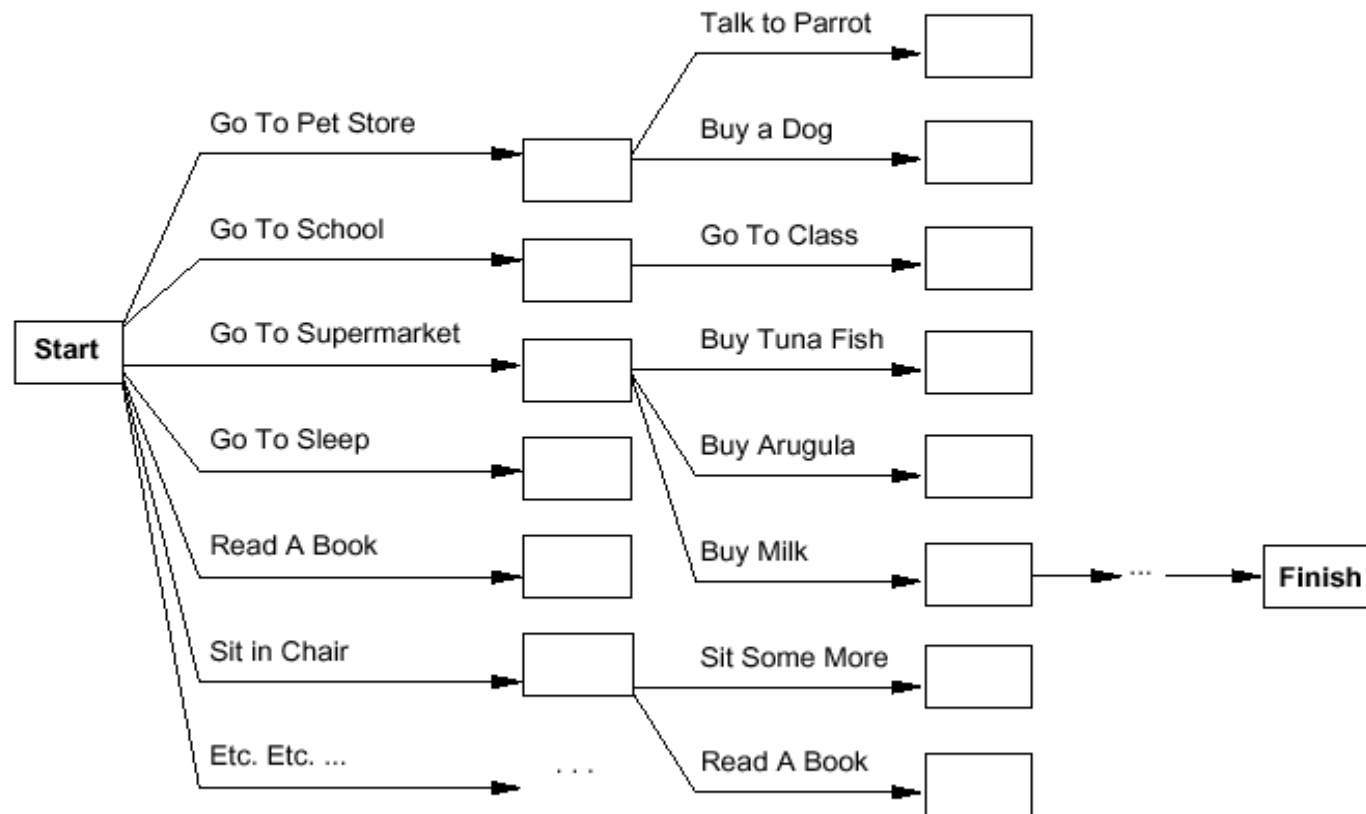
t ← t+1

return action

Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*

- Standard search algorithms seem to fail miserably:



After-the-fact heuristic/goal test inadequate



Review

- Situation Calculus

- Successor state axioms:

$$\text{Broken}(x, \text{do}(s,a)) \leftrightarrow [a = \text{drop}(x) \wedge \text{fragile}(x, s)] \vee \\ \exists b[a = \text{explode}(b) \wedge \text{nextTo}(b, x, s)] \vee \\ \text{broken}(x,s) \wedge \neg \exists a = \text{repair}(x)$$



Assumptions

- Fully observable and no uncertainty.
 - Classical planning
- Non-classical
 - Partially observable or
 - Stochastic

Planning in situation calculus

$PlanResult(p, s)$ is the situation resulting from executing p in s

$$PlanResult([], s) = s$$

$$PlanResult([a|p], s) = PlanResult(p, Result(a, s))$$

Initial state $At(Home, S_0) \wedge \neg Have(Milk, S_0) \wedge \dots$

Actions as Successor State axioms

$$Have(Milk, Result(a, s)) \Leftrightarrow$$

$$[(a = Buy(Milk) \wedge At(Supermarket, s)) \vee (Have(Milk, s) \wedge a \neq \dots)]$$

Query

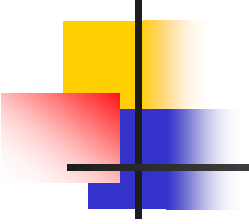
$$s = PlanResult(p, S_0) \wedge At(Home, s) \wedge Have(Milk, s) \wedge \dots$$

Solution

$$p = [Go(Supermarket), Buy(Milk), Buy(Bananas), Go(HWS), \dots]$$

Principal difficulty: unconstrained branching, hard to apply heuristics

Basic representation for planning

- 
- Most widely used approach: STRIPS
 - STanford Research Institute Problem Solver
 - **states**: conjunctions of function-free ground literals (i.e., predicates applied to constant symbols, possibly negated); e.g.,
$$\text{At(Home)} \wedge \neg\text{Have(Milk)} \wedge$$
$$\neg\text{Have(Bananas)} \wedge \neg\text{Have(Drill)} \dots$$

Basic representation for planning

- **goals**: also conjunctions of literals; e.g.,

$\text{At}(\text{Home}) \wedge \text{Have}(\text{Milk}) \wedge$
 $\text{Have}(\text{Bananas}) \wedge \text{Have}(\text{Drill})$

but can also contain variables (implicitly
universally quantified); e.g.,

$\text{At}(x) \wedge \text{Sells}(x, \text{Milk})$



Question

- Does STIPS use propositional literals or first order literals?
- Note: STRIPS assumes closed world
 - Any condition that is not mentioned in the state is assumed to be false.



Planner vs. theorem prover

- **Planner:** ask for sequence of actions that makes goal true if executed
- **Theorem prover:** ask whether query sentence is true given KB

STRIPS operators

Tidily arranged actions descriptions, restricted language

ACTION: $Buy(x)$

PRECONDITION: $At(p), Sells(p, x)$

EFFECT: $Have(x)$

[Note: this abstracts away many important details!]

Restricted language \Rightarrow efficient algorithm

Precondition: conjunction of positive literals

Effect: conjunction of literals

Graphical notation:





Types of planners

- Forward state space planner (Progression Planner): search through possible situations
 - Also called progression planner.
 - start with initial state, apply operators until goal is reached

Problem: high branching factor!



Types of planners

- **Regression planner**: start from goal state and apply operators until start state reached
 - Backward state-space search
 - **Why desirable?** Smaller branching factor.
 - **Difficulty**: when want to achieve a conjunction of goals

Initial STRIPS algorithm: situation-space regression planner

- **Situation state planner**: searches the space of possible solutions



What about Heuristics?

- Relaxed Problem: Admissible heuristic
 - For instance: remove negative effects of an action
- Divide-and-conquer
 - Assume sub-goal independence.
 - Can be
 - Optimistic: an action will undo result of another action in another subgoal
 - Pesimistic: there are duplicate actions in subgoals.



State space vs. plan space

Standard search: node = concrete world state

Planning search: node = partial plan

Search space of plans rather than of states.

Defn: open condition is a precondition of a step not yet fulfilled

Operators on partial plans:

add a link from an existing action to an open condition

add a step to fulfill an open condition

order one step wrt another

irradually move from incomplete/vague plans to complete, correct plans



Operations on plans

- **Refinement operators**: add constraints to partial plan (no further plan added)

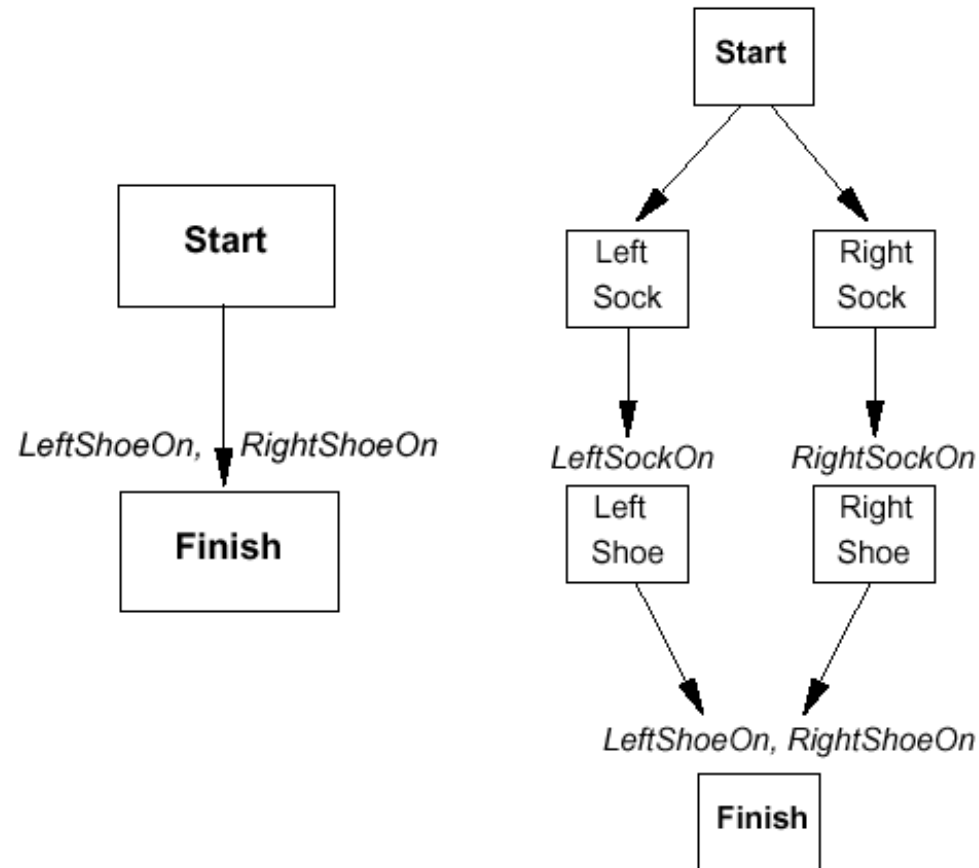
- **Modification operator**: every other operators (further plan may be added)



Types of planners

- **Partial order planner:** some steps are ordered, some are not
 - Example: getting banana and milk
- **Total order planner:** all steps ordered (thus, plan is a simple list of steps)
- **Linearization:** process of deriving a totally ordered plan from a partially ordered plan.

Partially ordered plans

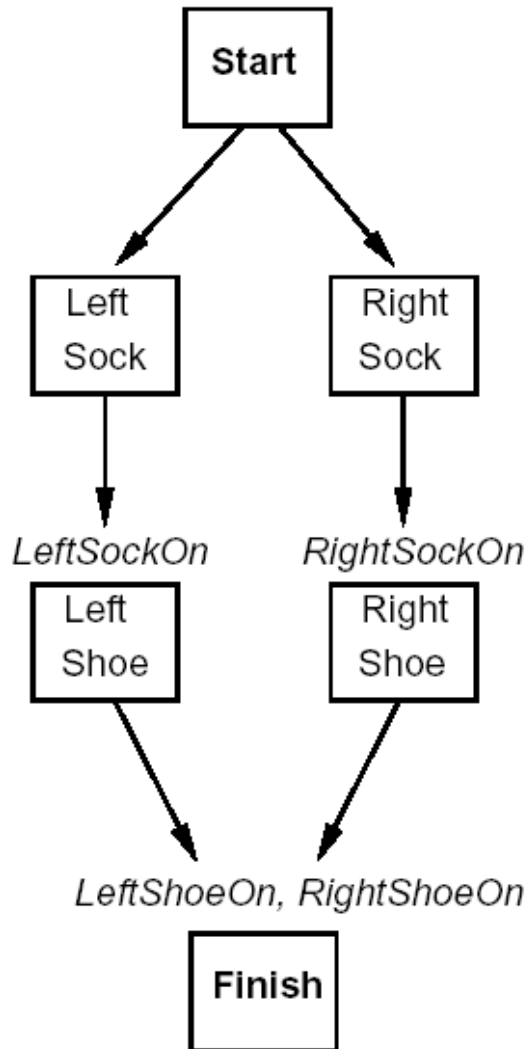


A plan is complete iff every precondition is achieved

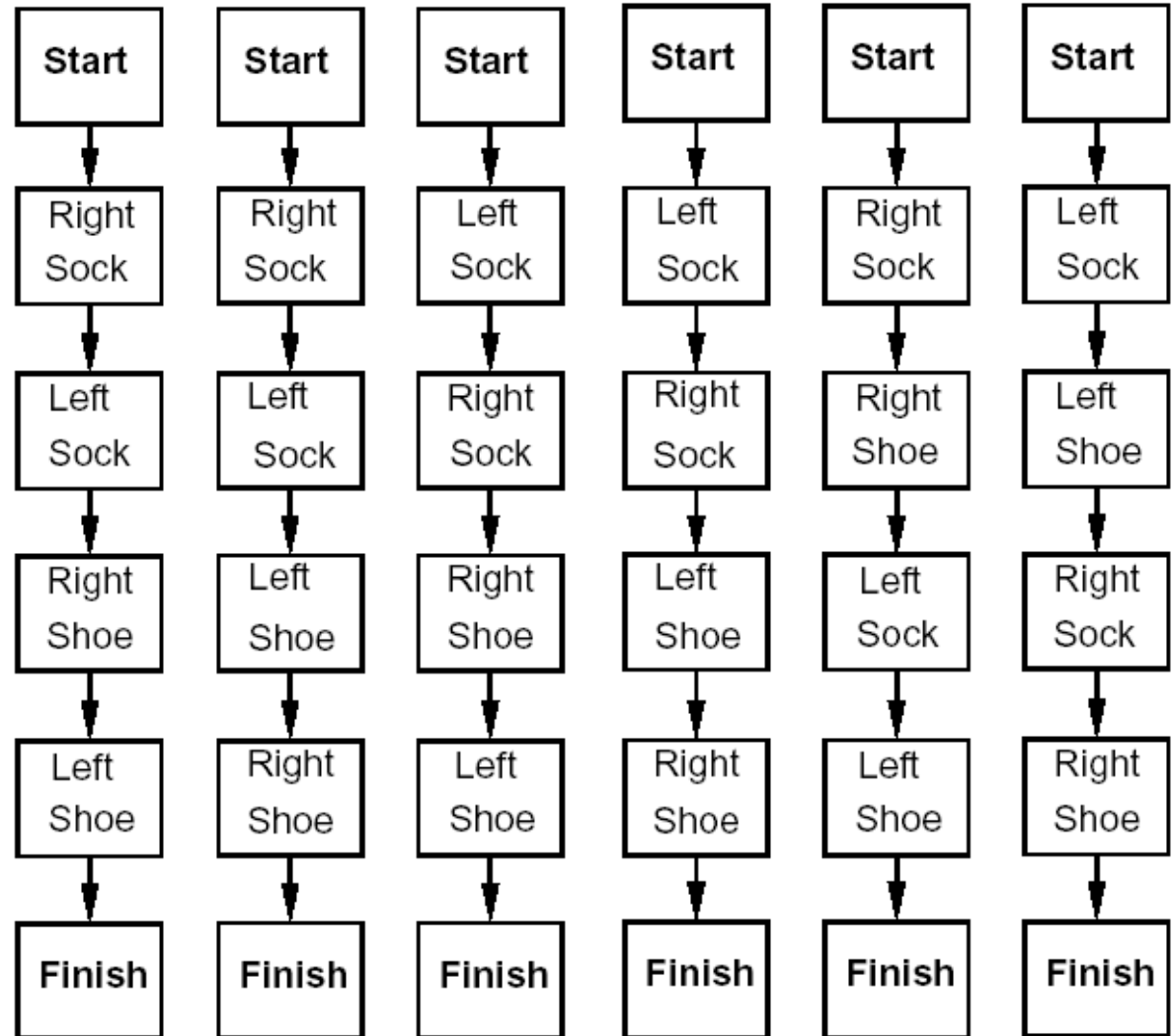
A precondition is achieved iff it is the effect of an earlier step and no possibly intervening step undoes it

Total-Order vs Partial-Order Plans

Partial Order Plan:



Total Order Plans:





Plan

We formally define a plan as a **data structure** consisting of:

- Set of **plan steps** (each is an operator for the problem)
- Set of **step ordering constraints**

e.g., $A < B$ means "A before B"



Plan

- Set of **variable binding constraints**

e.g., $v = x$ where v variable and x
constant or other variable

- Set of **causal links**

e.g., $A \xrightarrow{c} B$ means "A achieves c for B"

- C is a precondition for B

POP algorithm sketch

function POP(*initial, goal, operators*) **returns** *plan*

plan ← MAKE-MINIMAL-PLAN(*initial, goal*)

loop do

if SOLUTION?(*plan*) **then return** *plan*

S_{need}, c ← SELECT-SUBGOAL(*plan*)

 CHOOSE-OPERATOR(*plan, operators, S_{need}, c*)

 RESOLVE-THREATS(*plan*) //does adding the new step violates constraints?

end

function SELECT-SUBGOAL(*plan*) **returns** S_{need}, c

 pick a plan step S_{need} from STEPS(*plan*)

 with a precondition *c* that has not been achieved

return S_{need}, c

POP algorithm (cont.)

procedure CHOOSE-OPERATOR($plan, operators, S_{need}, c$)

choose a step S_{add} from $operators$ or $STEPS(plan)$ that has c as an effect

if there is no such step **then fail**

add the causal link $S_{add} \xrightarrow{c} S_{need}$ to $LINKS(plan)$

add the ordering constraint $S_{add} \prec S_{need}$ to $ORDERINGS(plan)$

if S_{add} is a newly added step from $operators$ **then**

add S_{add} to $STEPS(plan)$

add $Start \prec S_{add} \prec Finish$ to $ORDERINGS(plan)$

procedure RESOLVE-THREATS($plan$)

for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in $LINKS(plan)$ **do**

choose either

Demotion: Add $S_{threat} \prec S_i$ to $ORDERINGS(plan)$

Promotion: Add $S_j \prec S_{threat}$ to $ORDERINGS(plan)$

if not CONSISTENT($plan$) **then fail** For example, S_i is the initial step

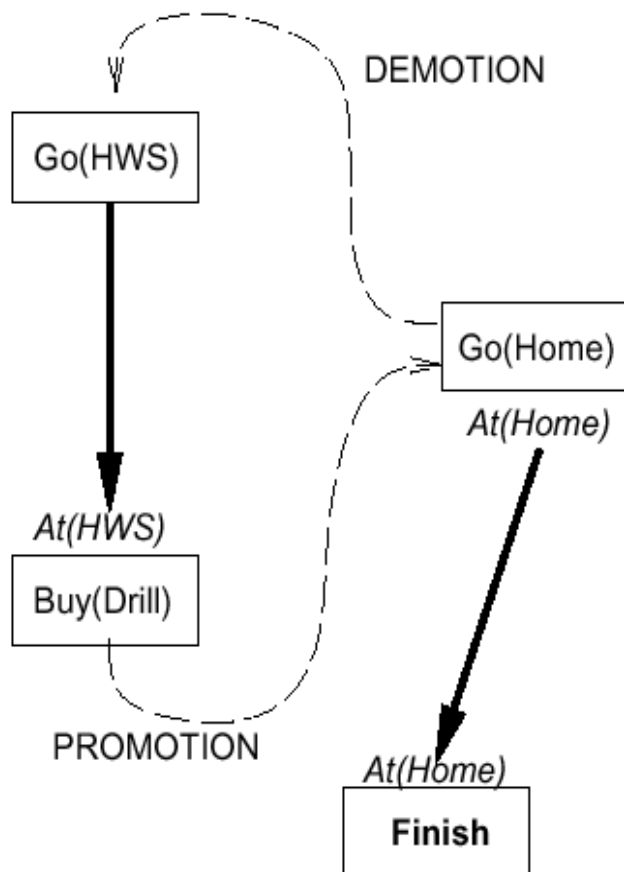
end

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

Extensions for disjunction, universals, negation, conditionals

Clobbering and promotion/demotion

A clobberer is a potentially intervening step that destroys the condition achieved by a causal link. E.g., $Go(Home)$ clobbers $At(HWS)$:

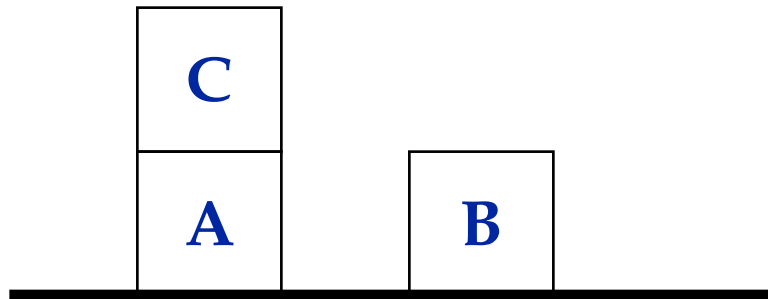


Demotion: put before $Go(HWS)$

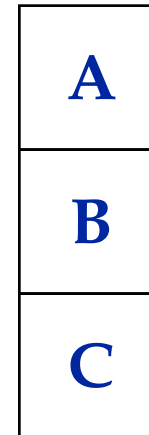
Promotion: put after $Buy(Drill)$

Example Problem Instance:

Initial State:



Goal:



Initial State: (and (on-table A) (on C A)
(on-table B) (clear B) (clear C))

Goal: (and (on A B) (on B C))



Action Representation: Propositional STRIPS

Move-C-from-A-to-Table:

preconditions: (on C A) (clear C)

effects:

add (on-table C)

delete (on C A)

add (clear A)

Solution to frame problem: explicit effects are the only changes to the state.



Action Representation: Propositional STRIPS

Move-C-from-A-to-Table:

preconditions: (and (on C A) (clear C))

effects:

(and (on-table C)

(not (on C A))

(clear A))

Solution to frame problem: explicit effects are the only changes to the state.

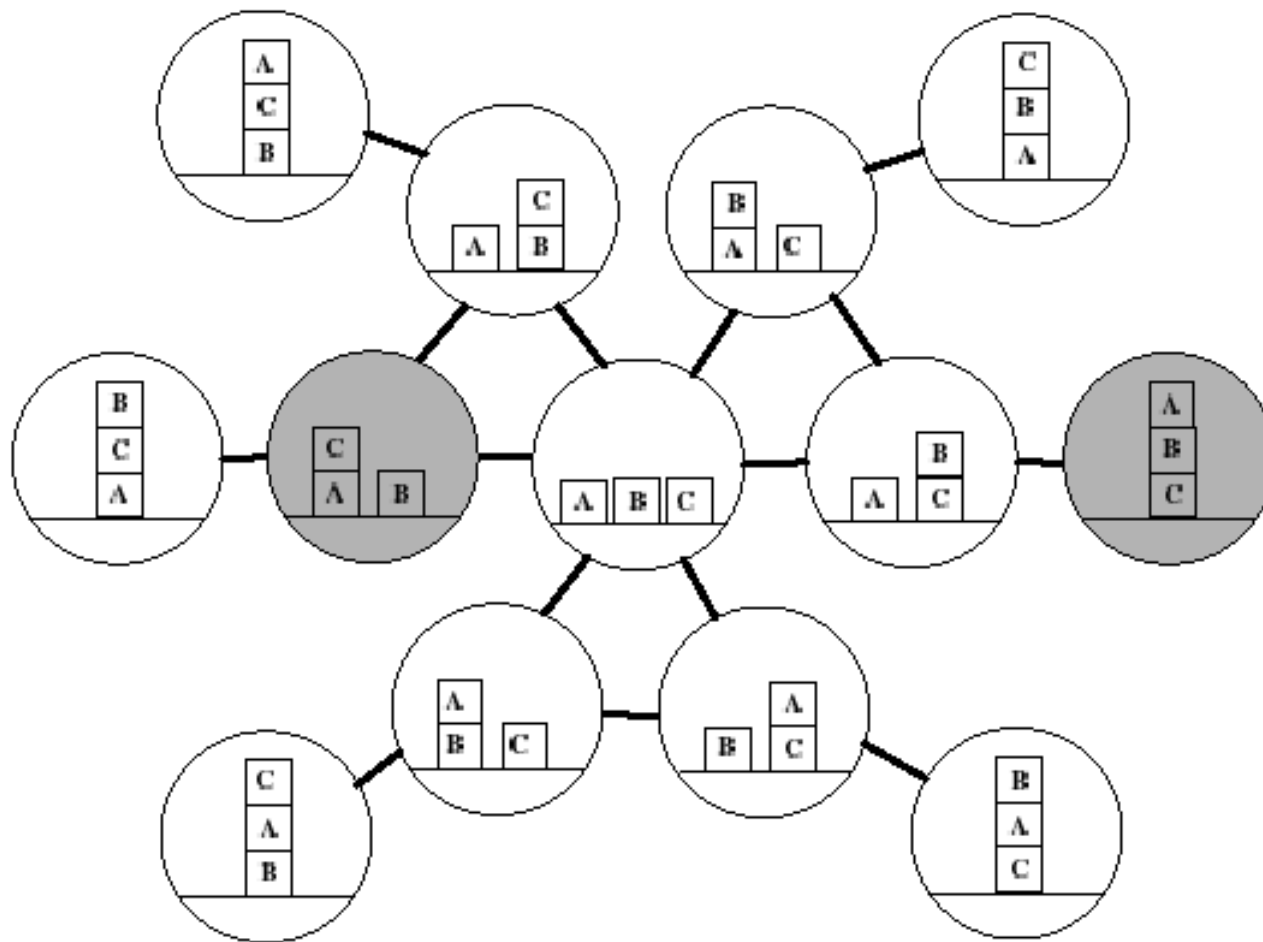


Plan Generation: Search space of world states

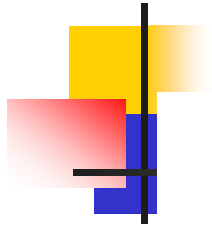
Planning as a (graph) search problem

- Nodes: world states
- Arcs: actions
- Solution: path from the initial state to one state that satisfies the goal
 - Initial state is fully specified
 - There are many goal states

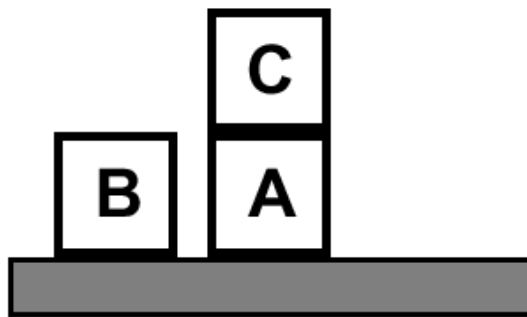
Search Space: Blocks World



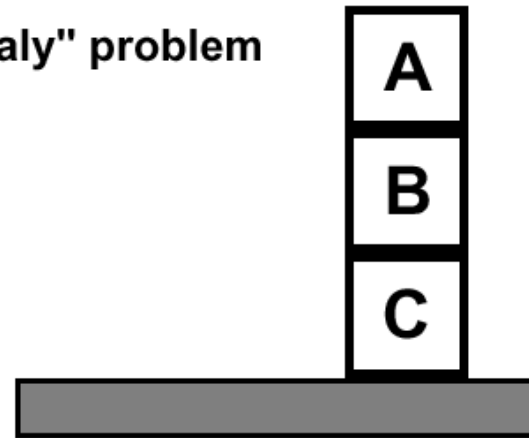
Example: block world



"Sussman anomaly" problem

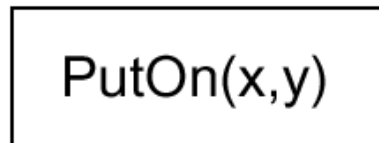


Start State



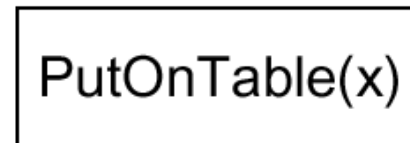
Goal State

$Clear(x) \ On(x,z) \ Clear(y)$



$\sim On(x,z) \ \sim Clear(y)$
 $Clear(z) \ On(x,y)$

$Clear(x) \ On(x,z)$



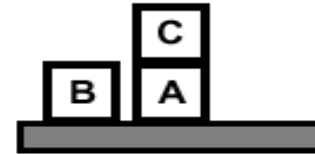
$\sim On(x,z) \ Clear(z) \ On(x, Table)$

+ several inequality constraints

Example (cont.)

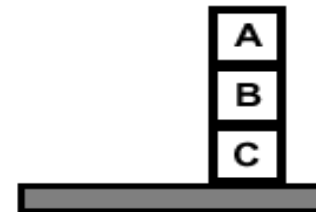
START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

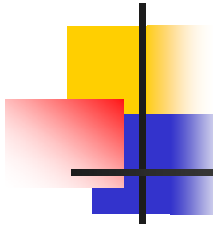


On(A,B) On(B,C)

FINISH



Example (cont.)



START

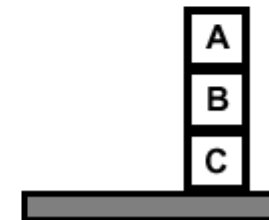
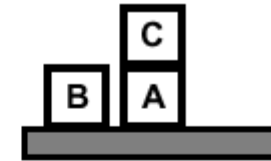
$On(C,A)$ $On(A,Table)$ $Cl(B)$ $On(B,Table)$ $Cl(C)$

$Cl(B)$ $On(B,z)$ $Cl(C)$

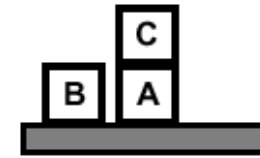
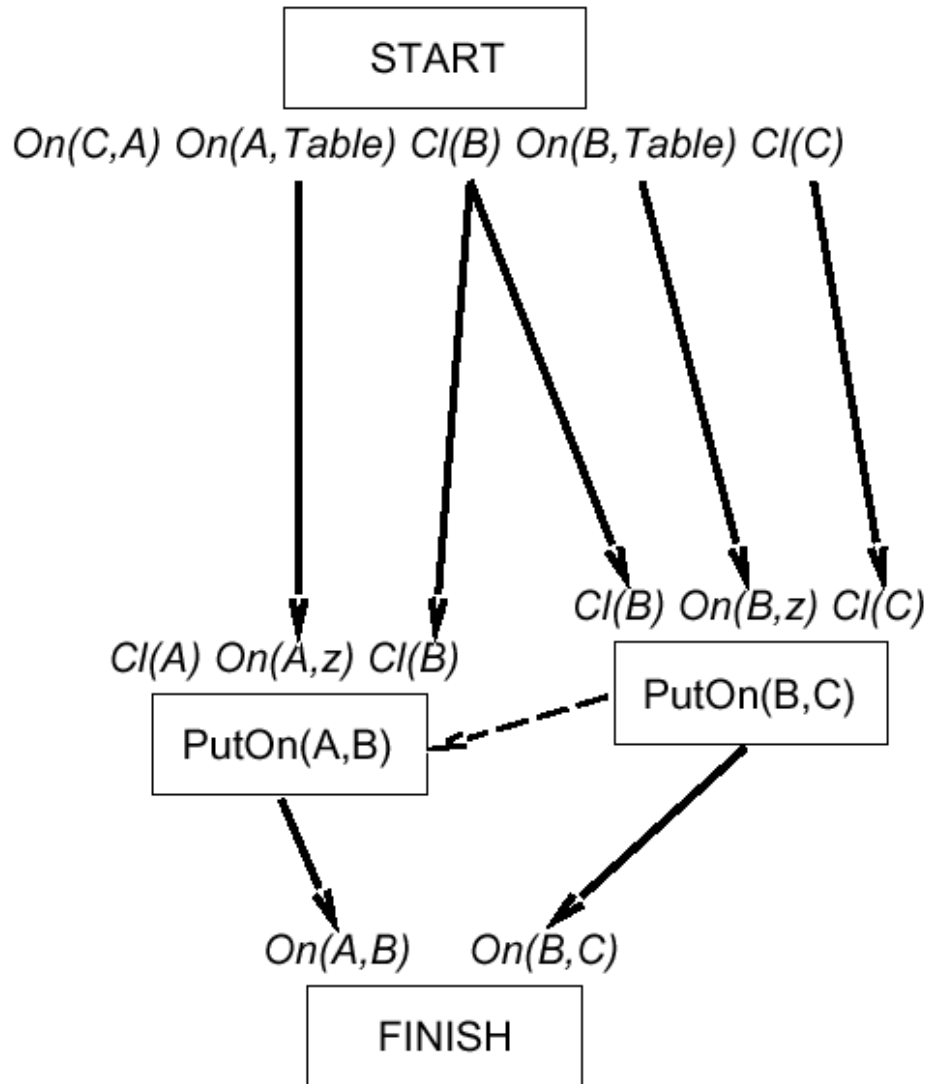
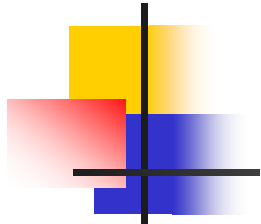
PutOn(B,C)

$On(A,B)$ $On(B,C)$

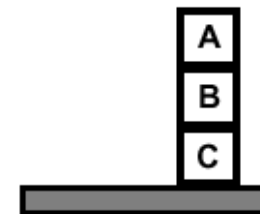
FINISH



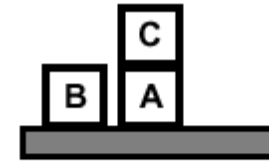
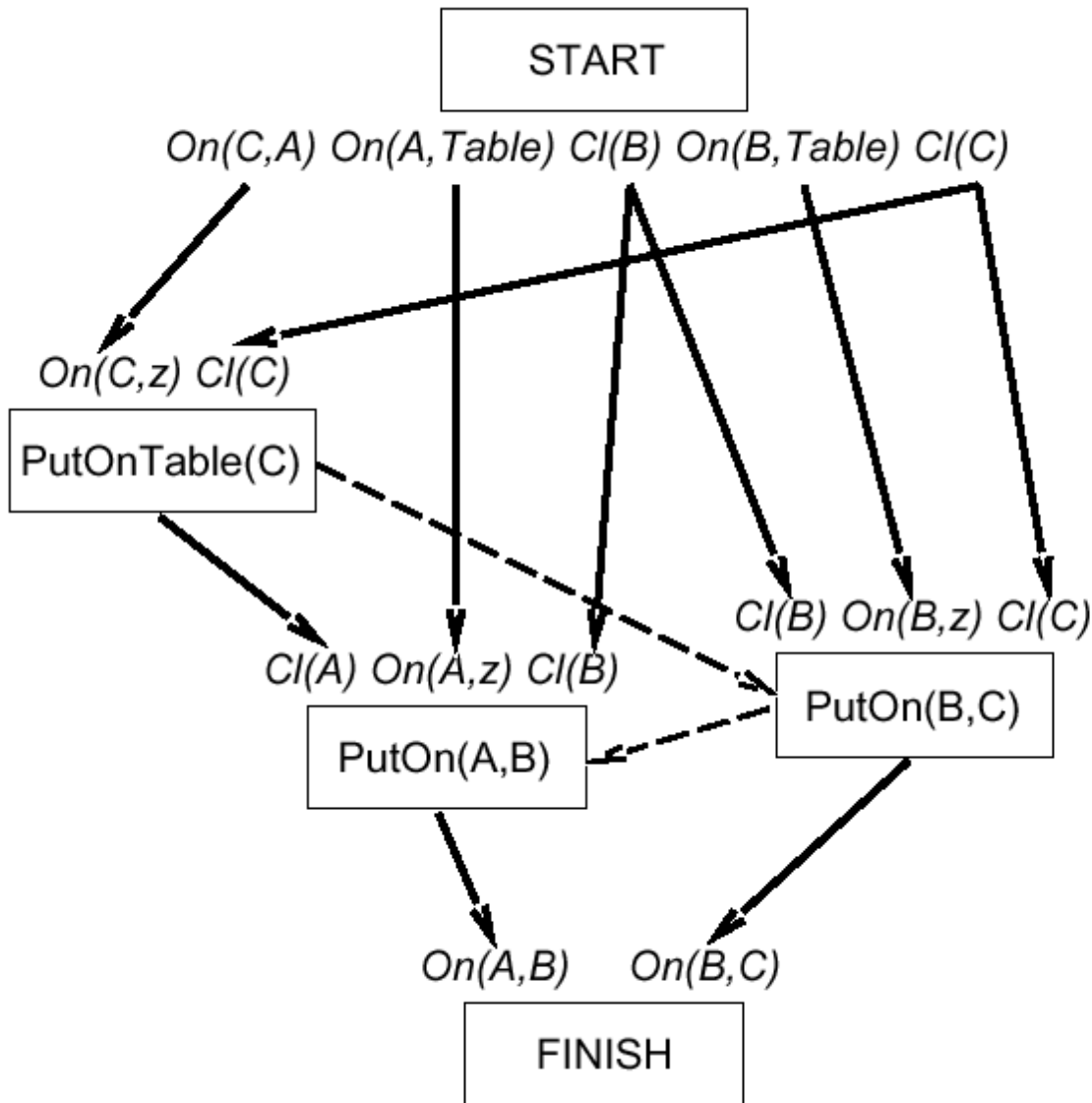
Example (cont.)



PutOn(A,B)
 clobbers Cl(B)
 => order after
 PutOn(B,C)

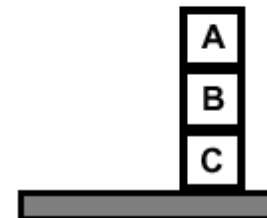


Example (cont.)



PutOn(A,B)
 clobbers $Cl(B)$
 \Rightarrow order after
 PutOn(B,C)

PutOn(B,C)
 clobbers $Cl(C)$
 \Rightarrow order after
 PutOnTable(C)





Heuristics for POP

- Predict how far a POP is from achieving the goal.
- Benefit?
- Some heuristics:
 - Number of open preconditions:
 - Problem?
 - The most-constrained-variable
 - Select an open precondition that can be satisfied in the fewest number of ways.
 - Detect impossible sub-goals
 - Detect more constraints.



Summary

- Search vs. planning
- STRIPS operators
- Partial-order planning
 - Promotion and demotion
- AIMA Chapter 11 up to 11.4