Classical Planning
Partial-Order Planning

Sections 10.1,10.4.4

—

Nilufer Onder
Department of Computer Science
Michigan Technological University

Outline

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

What is AI planning?

Planning is the task of finding a set of actions that will achieve a goal. A *planner* is a program that searches for a plan. It inputs a description of the world and the goals. The output is a plan. The simplest plan is a sequence of actions:

``‘do action1, do action2 ...’'`

More complex plans may include branching actions: “if (condition) do action1 else do action2”

Planning Domain Definition Language (PDDL)

Tidily arranged actions descriptions, restricted language

At(s) ~Bought(x) Sells(s,x)

BUY (s,x)

Bought(x)

ACTION: $Buy(s, x)$
PRECONDITION: $At(s), \neg Bought(x), Sells(s, x)$
EFFECT: $Bought(x)$

PDDL operators (cont'd)

ACTION: $Buy(s, x)$
PRECONDITION: $At(s), \neg Bought(x), Sells(s, x)$
EFFECT: $Bought(x)$

- Restricted language $\Longrightarrow$ efficient algorithm (but many important details will have to be abstracted away)
- Action schema: name, parameters, preconditions, effects
- Precondition: conjunction of positive literals
  Effect: conjunction of literals
- STRIPS is the earliest planning representation

Search vs. planning (cont'd)

	Search	**Planning**
States	Data structure	Logical sentences
Actions	Program	Preconditions/outcomes
Goal	Program	Logical sentence (conjunction)
Plan	Path from S_0	(Sequence of) actions

Search vs. planning (cont'd)

Planning systems do the following:

1. open up action and goal representation to allow selection
2. divide-and-conquer by subgoaling
3. relax requirement for sequential construction of solutions

States

- The state of the world is represented by a collection of variables
(*factored representation*)
- Each *state* is represented as a conjunction of fluents that are ground, functionless atoms.
- A state is a set (*set semantics*)
- Use database semantics, *closed world assumption*:
If a fluent is not mentioned, assume it is false.
- Fluents that are non-ground, negated, or using functions are not allowed.

Partially ordered plans

Partially ordered collection of steps with

- START step has the initial state description as its effect
- FINISH step has the goal description as its precondition
- causal links from outcome of one step to precondition of another
- temporal ordering between pairs of steps

Partially ordered plans (cont'd)

A partially ordered plan is a 5-tuple (A, O, C, OC, UL)

- A is the set of actions that make up the plan. They are partially ordered.
- O is a set of ordering constraints of the form $A \prec B$. It means A comes before B.
- C is the set of causal links in the form (A, p, B) where A is the *supplier action*, where B is the *consumer action*, and p is the condition supplied. It is read as "*A achieves p for B*."

Partially ordered plans (cont'd)

A partially ordered plan is a 5-tuple (A, O, C, OC, UL)

- OC is a set of open conditions, i.e., conditions that are not yet supported by causal links. It is of the form p for A where p is a condition and A is an action.
- UL is a set of unsafe links, i.e., causal links whose conditions might be undone by other actions.

Partially ordered plans (cont'd)

A plan is *complete* iff every precondition is achieved, and there are no unsafe links. A precondition is *achieved* iff it is the effect of an earlier step and no *possibly intervening* step undoes it

In other words, a plan is complete when $OC \cup UL = \emptyset$.

$OC \cup UL$ is referred to as the *flaws* in a plan.

When a causal link is established, the corresponding condition is said to be *closed*.

Example

START

CleanLeftSock

CleanRightSock

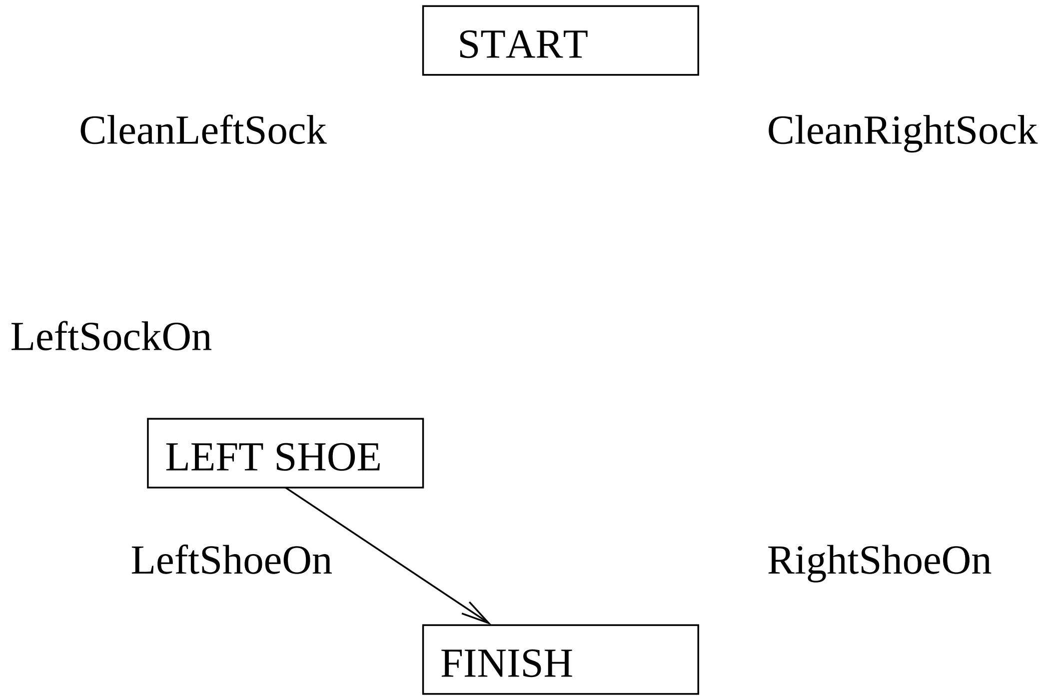
OC=
LeftShoeOn for FINISH
RightShoeOn for FINISH

LeftShoeOn

RightShoeOn

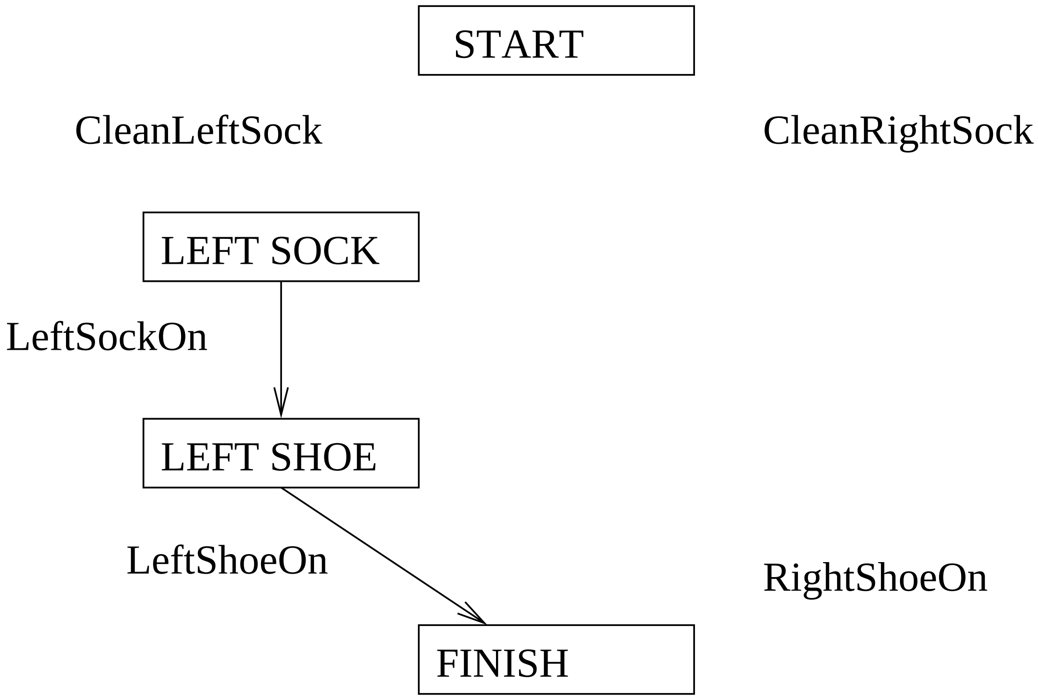
FINISH

Example (cont'd)



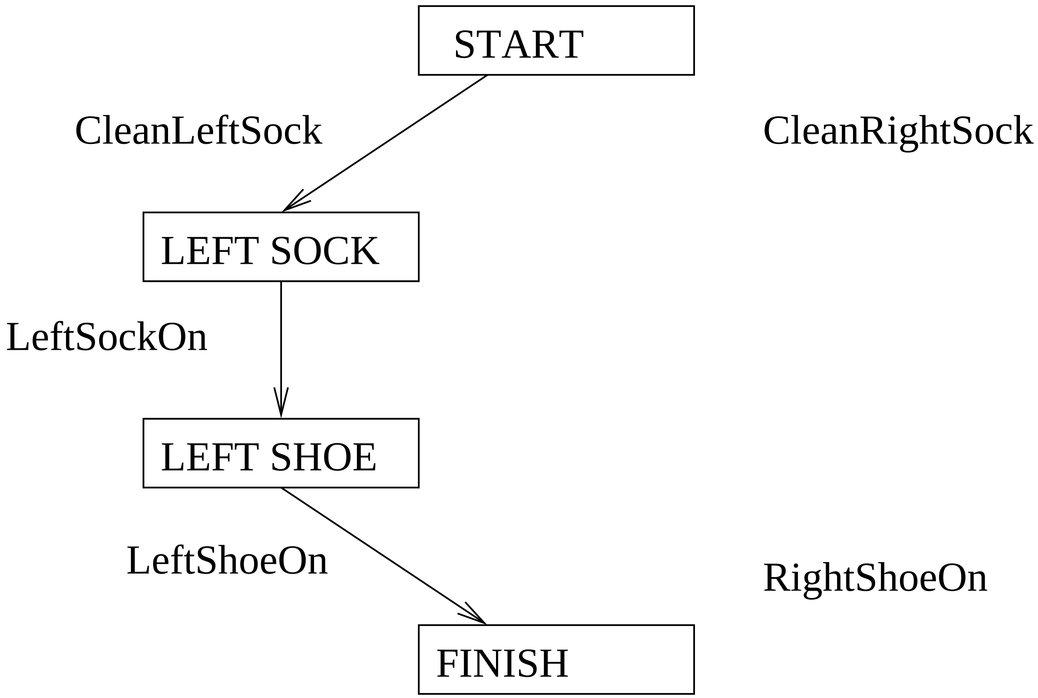
OC=
RightShoeOn for FINISH
LeftSockOn for LEFTSHOE

Example (cont’d)



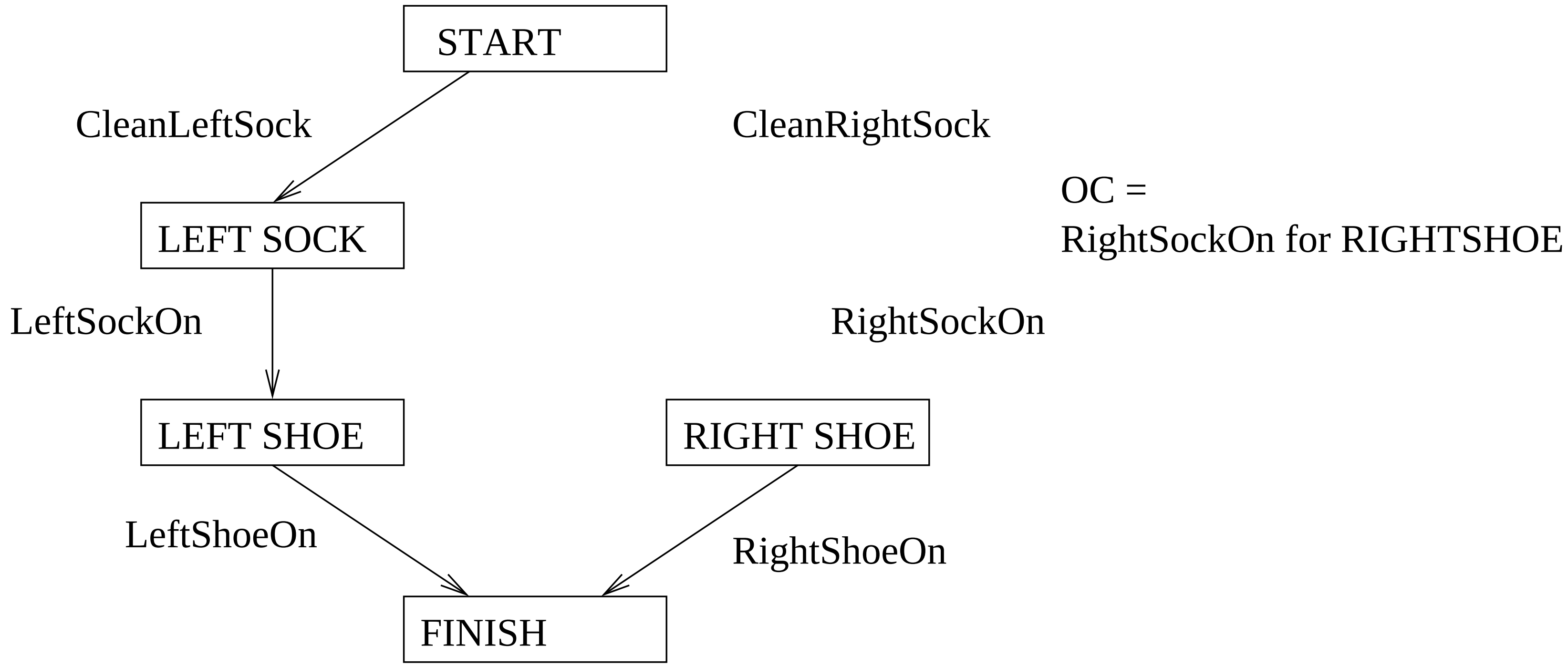
OC =
CleanLeftSock for LEFTSOCK
RightShoeOn for FINISH

Example (cont'd)

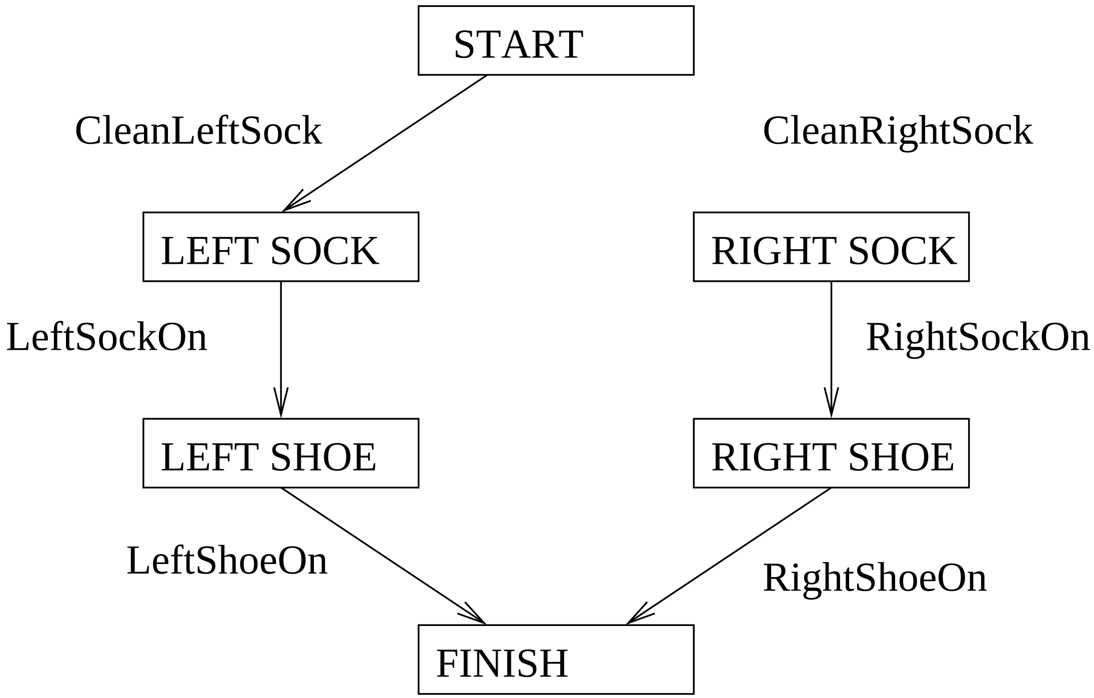


OC =
RightShoeOn for FINISH

Example (cont'd)

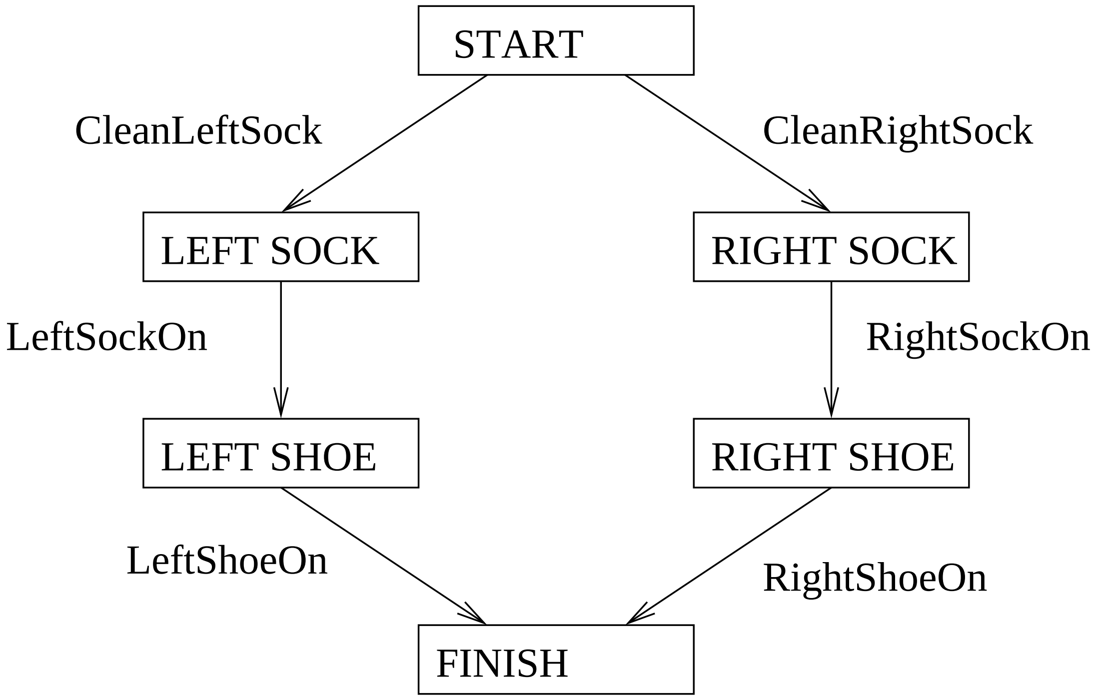


Example (cont'd)



OC =
CleanRightSock for RIGHTSOCK

Example (cont'd)



OC=
{ }

Planning process

Operators on partial plans:

- close open conditions:
  - add a link from an existing action to an open condition
  - add a step to fulfill an open condition
- resolve threats:
  - order one step wrt another to remove possible conflicts

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

Backtrack if an open condition is unachievable or if a conflict is unresolvable

POP is a search in the plan space

function TREE-SEARCH (*problem*)
returns a solution, or failure

initialize the frontier using the initial state of *problem*
loop do
  if the frontier is empty **then return** failure
  choose a leaf node and remove it from the frontier
  if the node contains a goal state
    then return the corresponding solution
  else expand the chosen node and add the resulting
    nodes to the frontier
end

POP algorithm specifics

The initial state, goal state and the operators are given. The planner converts them to required structures.

Initial state:
MAKE-MINIMAL-PLAN (*initial*,*goal*)

Goal-Test:
SOLUTION?(*plan*)

SOLUTION? returns true iff OC and UL are both empty.

Successor function:
The successors function could either close an open condition or resolve a threat.

POP algorithm specifics (cont'd)

function SUCCESSORS (*plan*)
returns *a set of partially ordered plans*

flaw-type $\leftarrow$ SELECT-FLAW-TYPE (*plan*)
if *flaw-type* is an open condition **then**
$S_{need}, c \leftarrow$ SELECT-SUBGOAL (*plan*)
return CLOSE-CONDITION (*plan, operators,* S_{need},*c*)
if *flaw-type* is a threat **then**
$S_{threat}, S_i, c, S_j \leftarrow$ SELECT-THREAT(*plan*)
return RESOLVE-THREAT (plan, S_{threat}, S_i, c, S_j)

POP algorithm specifics (cont'd)

function CLOSE-CONDITION (*plan, operators,* S_{need},*c*)
returns *a set of partially ordered plans*

plans $\leftarrow \emptyset$
for each S_{add} from *operators* or STEPS(*plan*)
that has c has an effect **do**
new-plan $\leftarrow$ plan
if S_{add} is a newly added step from *operators* **then**
add S_{add} to STEPS (*new-plan*)
add START $\prec S_{add} \prec$ FINISH to ORDERINGS (*new-plan*)
add the causal link (S_{add}, c, S_{need}) to LINKS (*new-plan*)
add the ordering constraint $(S_{add} \prec S_{need})$ to
ORDERINGS (*new-plan*)
add *new-plan* to *plans*
end

return *new-plans*

POP algorithm specifics (cont'd)

function RESOLVE-THREAT (*plan*, S_{threat}, S_i, c, S_j)
returns *a set of partially ordered plans*

plans $\leftarrow \emptyset$
//Demotion:
new-plan $\leftarrow$ plan
add the ordering constraint $(S_{threat} \prec S_i)$ to ORDERINGS (*new-plan*)
if *new-plan* is consistent **then**
add *new-plan* to *plans*
//Promotion:
new-plan $\leftarrow$ plan
add the ordering constraint $(S_j \prec S_{threat})$ to ORDERINGS (*new-plan*)
if *new-plan* is consistent **then**
add *new-plan* to *plans*

return *new-plans*

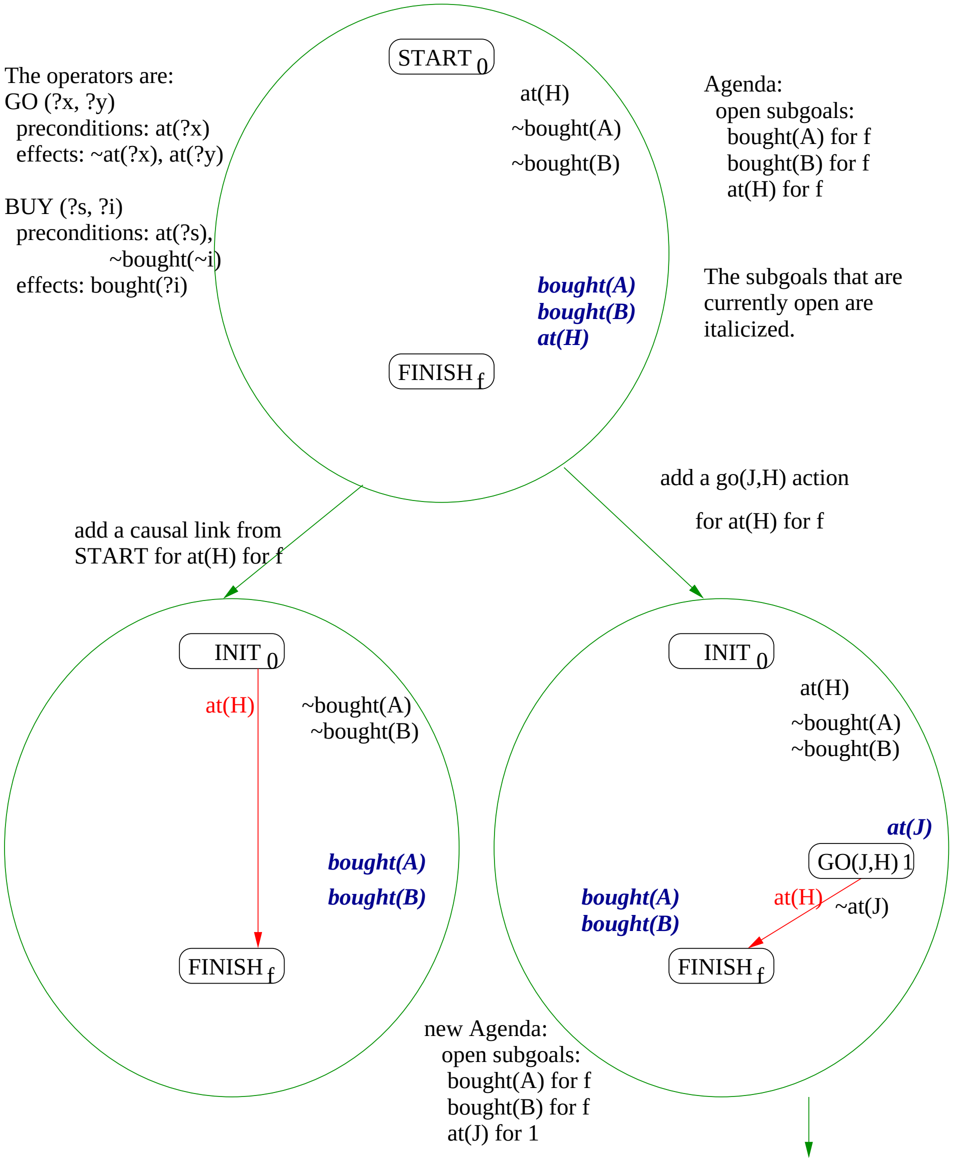
Shopping example

The operators are:
GO (?x, ?y)
  preconditions: at(?x)
  effects: ~at(?x), at(?y)

BUY (?s, ?i)
  preconditions: at(?s),
    ~bought(~i)
  effects: bought(?i)

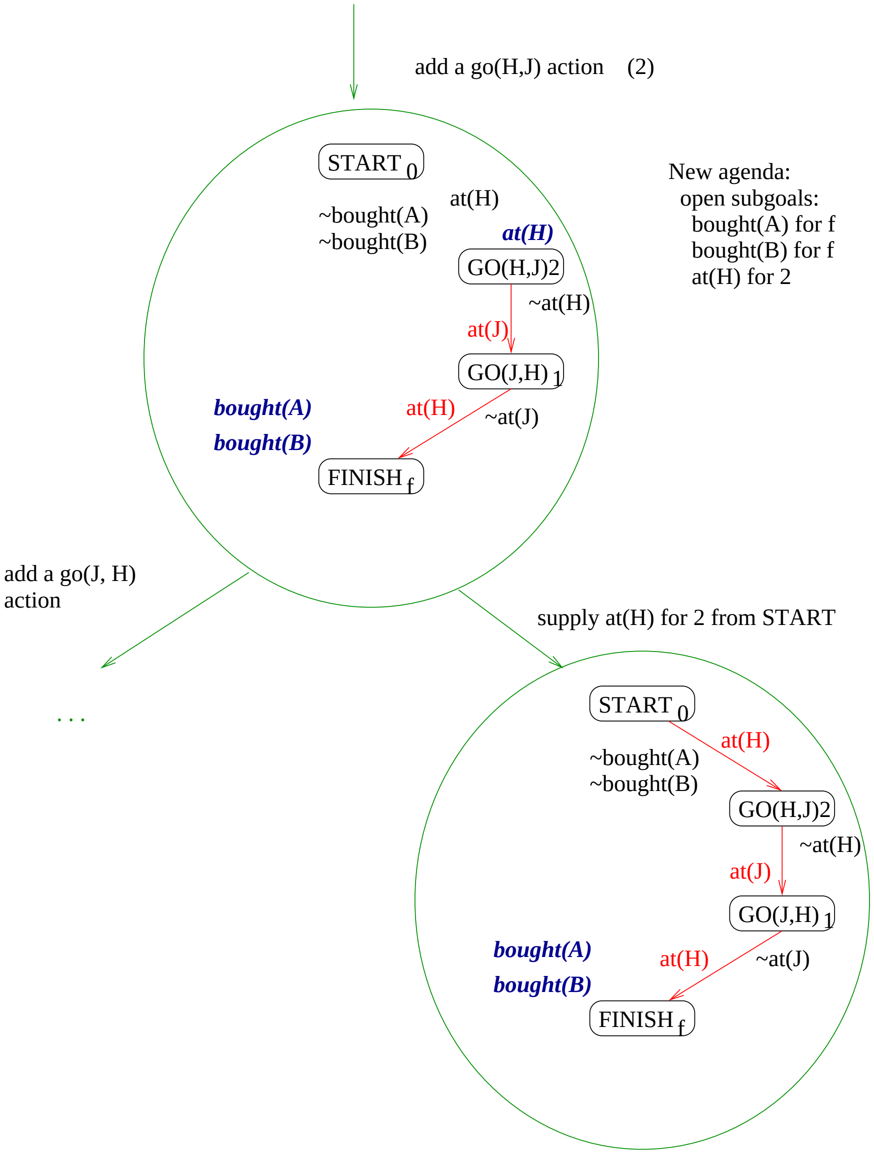
Agenda:
  open subgoals:
    bought(A) for f
    bought(B) for f
    at(H) for f

The subgoals that are currently open are italicized.

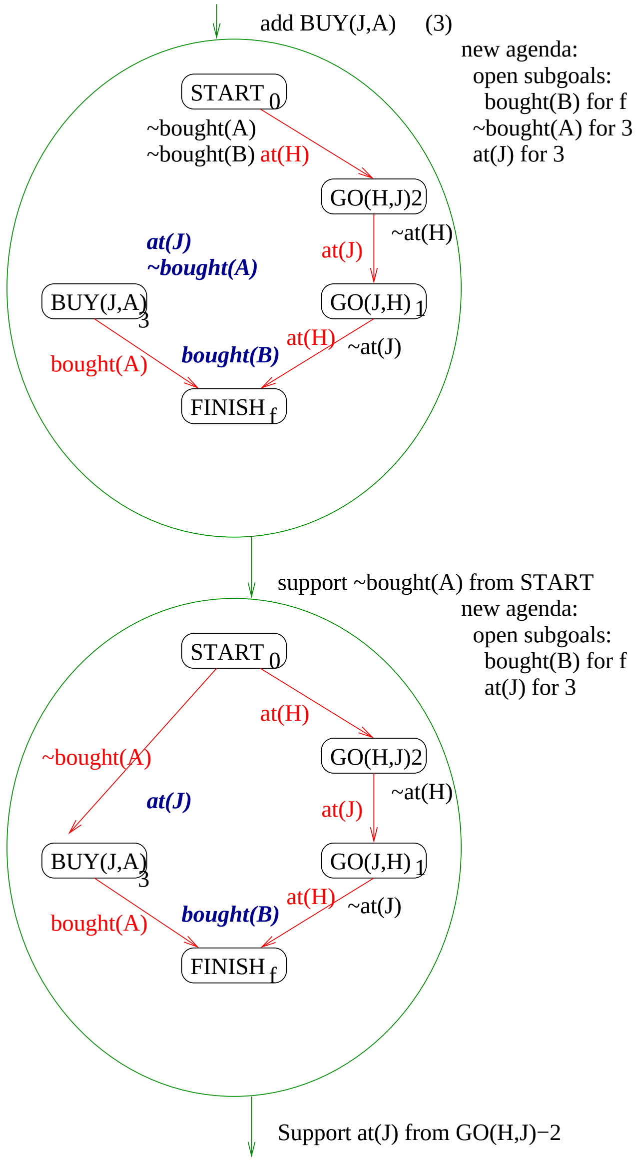


new Agenda:
  open subgoals:
    bought(A) for f
    bought(B) for f
    at(J) for 1

Shopping example (cont'd)

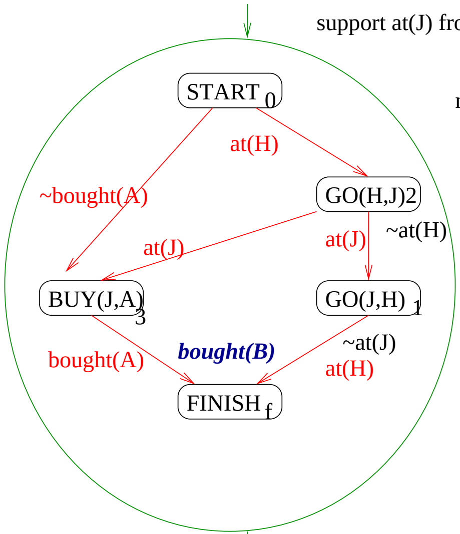


Shopping example (cont'd)



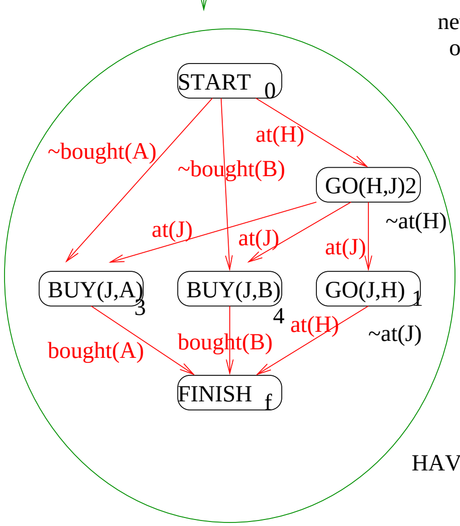
Shopping example (cont'd)

support at(J) from GO(H,J)−2



new agenda:
open subgoals:
bought(B) for f
~bought(A) for 3

add BUY(J,B) (4)



new agenda:
open subgoals:
~bought(B) for 4
at(J) for 4

support ~bought(B) for 4 from START

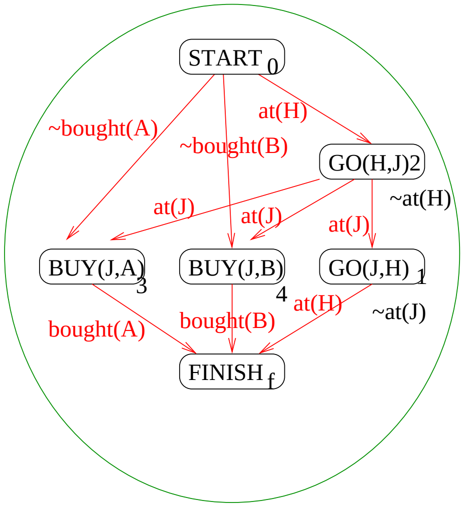
new agenda:
open subgoals:
at(J) for 4

support ~at(J) for 4
from GO(H,J)−2

new agenda:
open subgoals:
none

HAVEN'T CONSIDERED THE THREATS YET

Shopping example (cont'd)

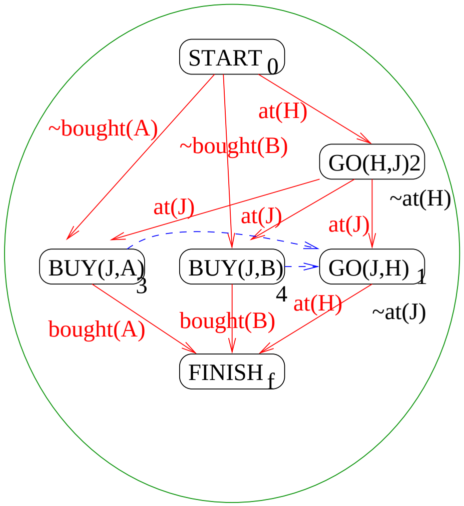


Now, the solution is a possible ordering of this plan. Those are:

2 3 4 1
2 3 1 4
2 4 3 1
2 4 1 3
2 1 3 4
2 1 4 3

It should not be possible to order GO(J,H) before any of the BUY actions.

Shopping example (cont'd)

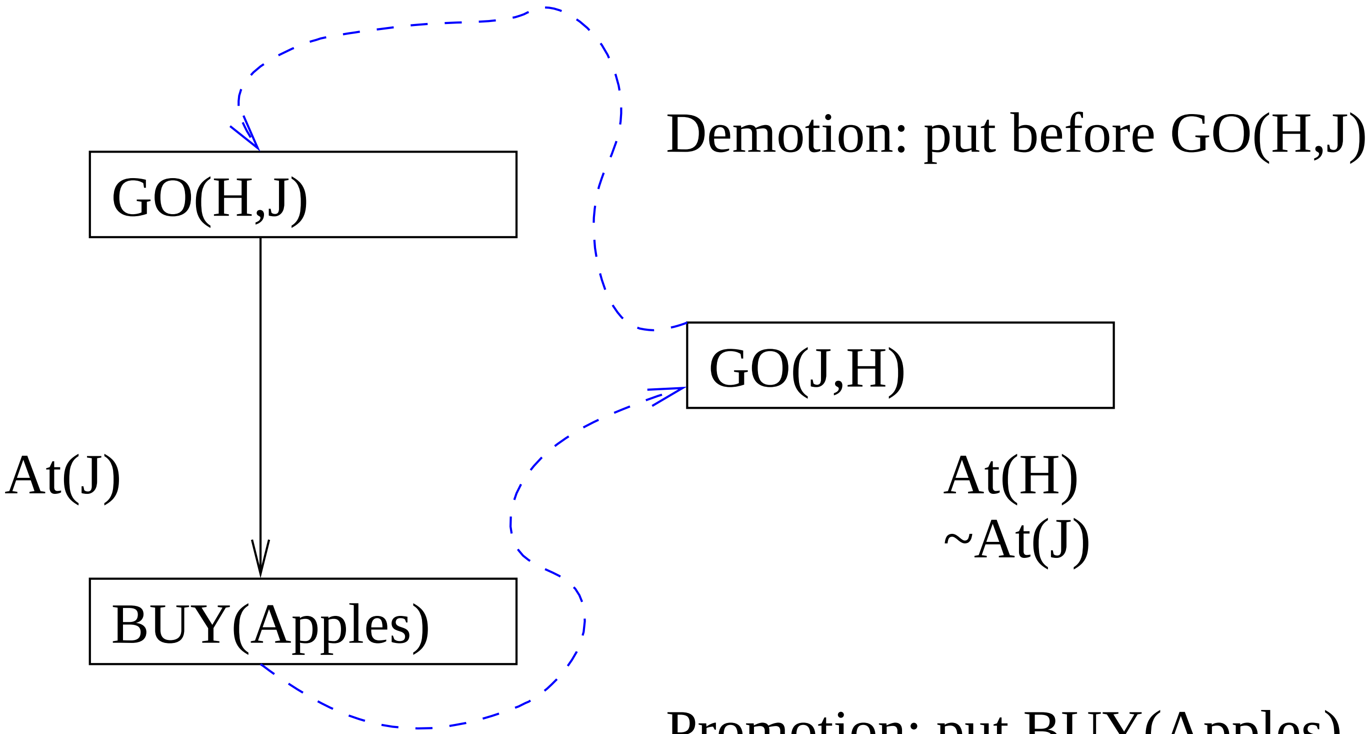


This is a correct partially ordered plan.
It is complete.
The possible total orders are:
2 3 4 1
2 4 3 1

The agent has to go to Jim's first.
It order of getting the items does not matter.
Then it has to go back home.

Threats and promotion/demotion

A threatening step is a potentially intervening step that destroys the condition achieved by a causal link.
E.g., GO(J,H) threatens At(J)



Properties of POP

- Nondeterministic algorithm: backtracks at choice points on failure:
  - choice of S_{add} to achieve S_{need}
  - choice of demotion or promotion for threat resolution
  - selection of S_{need} is irrevocable
- POP is sound, complete, and systematic (no repetition)
- Extensions for disjunction, universals, negation, conditionals
- Particularly good for problems with many loosely related subgoals

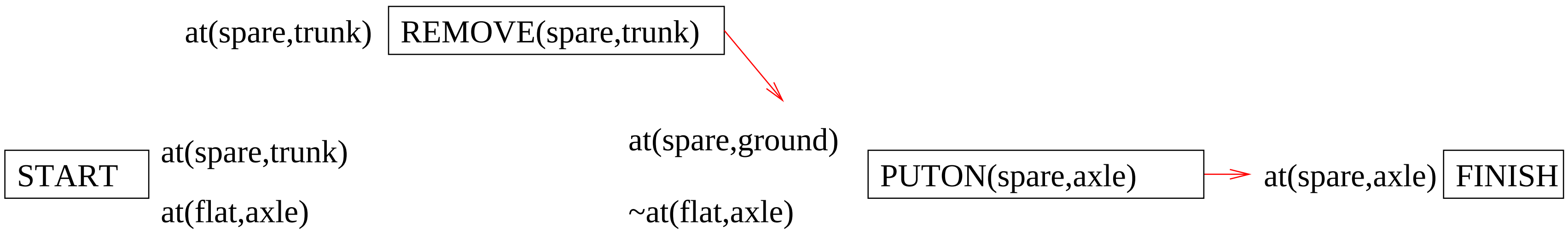
Additional POP examples

- The flat tire example shows the effect of inserting an "impossible" action.
- The Sussman anomaly shows that "divide-and-conquer" is not always optimal. POP can find the optimal plan.

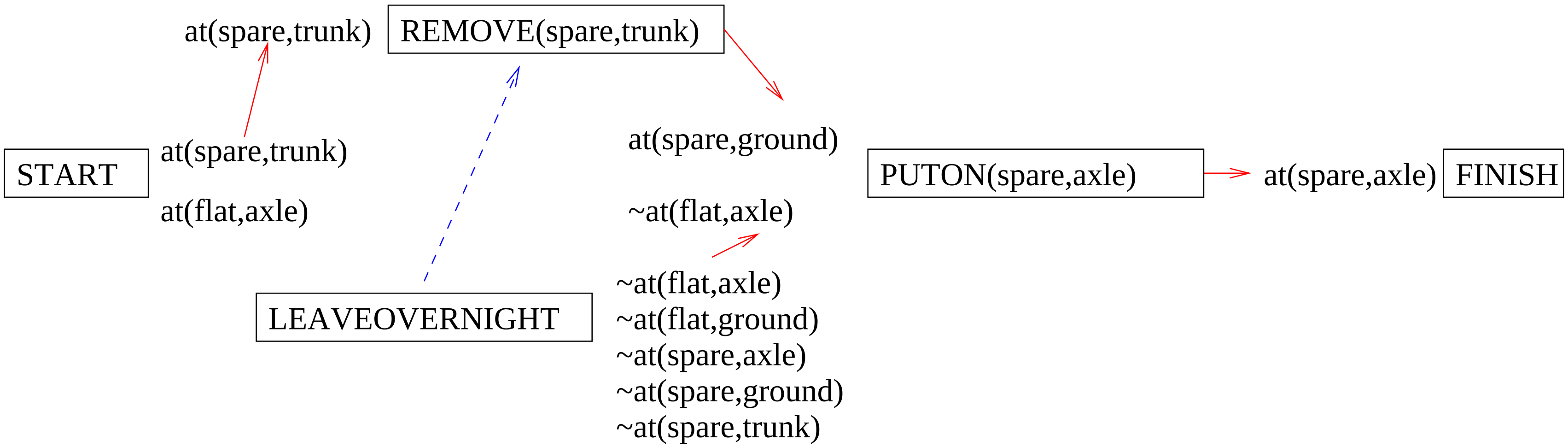
The flat tire domain

Init(At(Flat,Axle) ∧ At(Spare,Trunk))
Goal(At(Spare,Axle))
Action(REMOVE(spare,trunk),
  Precond: At(spare,trunk)
  Effect: ¬At(spare,trunk) ∧ At(spare,ground)
Action(REMOVE(flat,axle),
  Precond: At(flat,axle)
  Effect: ¬At(flat,axle) ∧ At(flat,ground)
Action(PUTON(spare,axle),
  Precond: At(spare,ground) ∧ ¬ at(flat,axle)
  Effect: ¬At(spare,ground) ∧ At(spare,axle)
Action(LEAVEOVERNIGHT
  Precond:
  Effect: ¬At(spare,ground) ∧ ¬ At(spare,axle)
    ¬At(spare,trunk) ∧ ¬ At(flat,ground)
    ¬At(flat,axle)

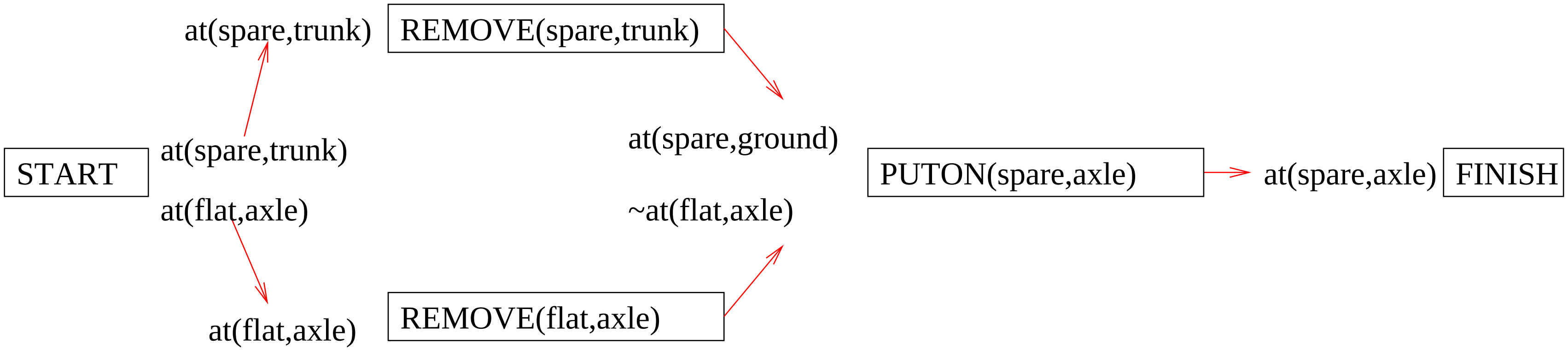
The flat tire plan



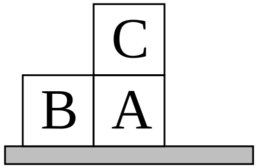
The flat tire plan (cont’d)



The flat tire plan (cont'd)



Sussman anomaly



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

$\boxed{\text{PUTON(x,y)}}$

~On(x,z)   ~Clear(y)

Clear(z)   On(x,y)

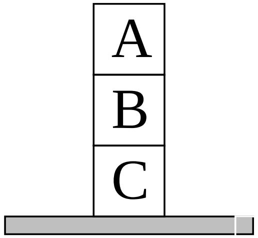
Clear(x)   On(x,z)

$\boxed{\text{PUTONTABLE(x)}}$

~On(x,z)

Clear(z)   On(x,Table)

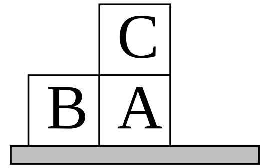
+ several inequality constraints



Sussman anomaly (cont'd)

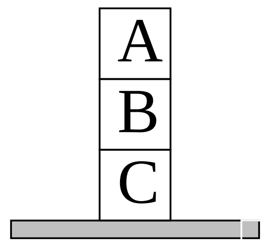
START

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



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

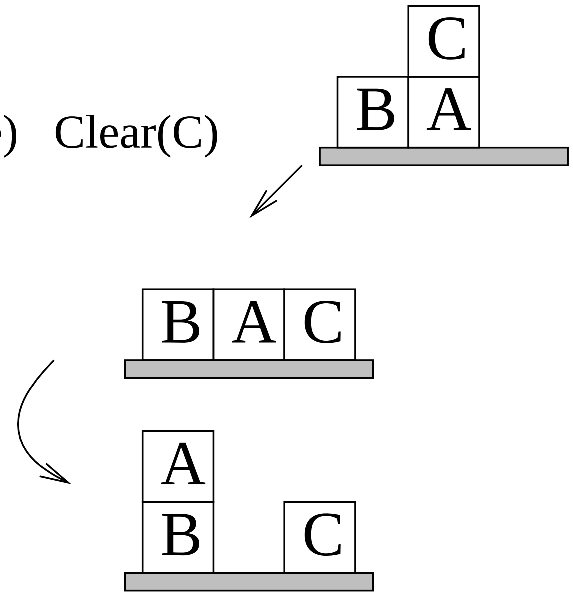
FINISH



Sussman anomaly (cont'd)

START

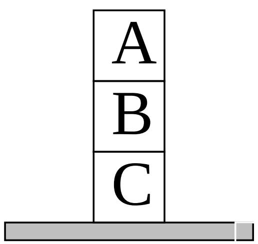
On(C,A) On(A,Table) Clear(B) On(B,Table) Clear(C)



If we try the first goal (on(A,B)) first,
we can't proceed without undoing work

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

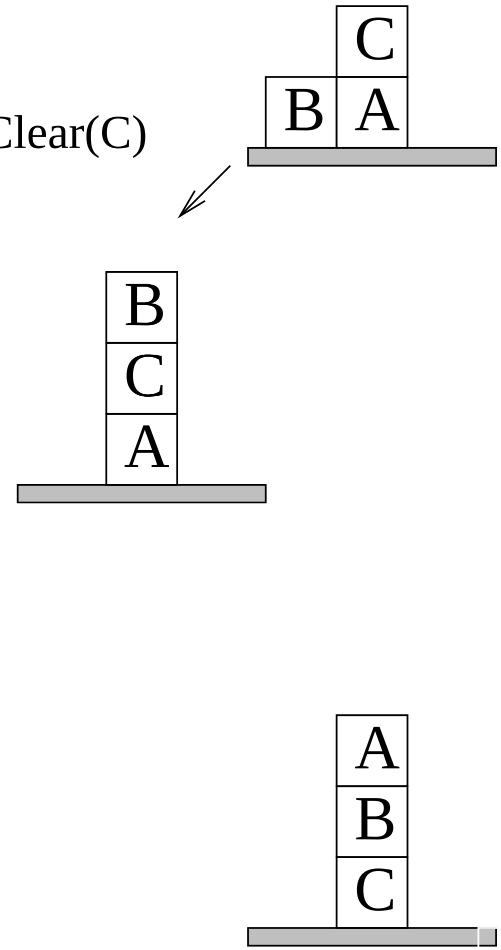
FINISH



Sussman anomaly (cont'd)

START

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



If we try the second goal (on (B,C)) first,
we can't proceed without undoing work.

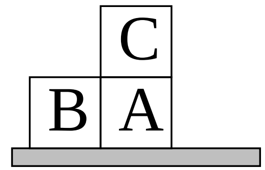
On(A,B) On(B,C)

FINISH

Sussman anomaly (cont'd)

START

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

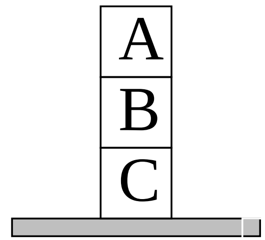


Clear(B) On(B,z) Clear(C)

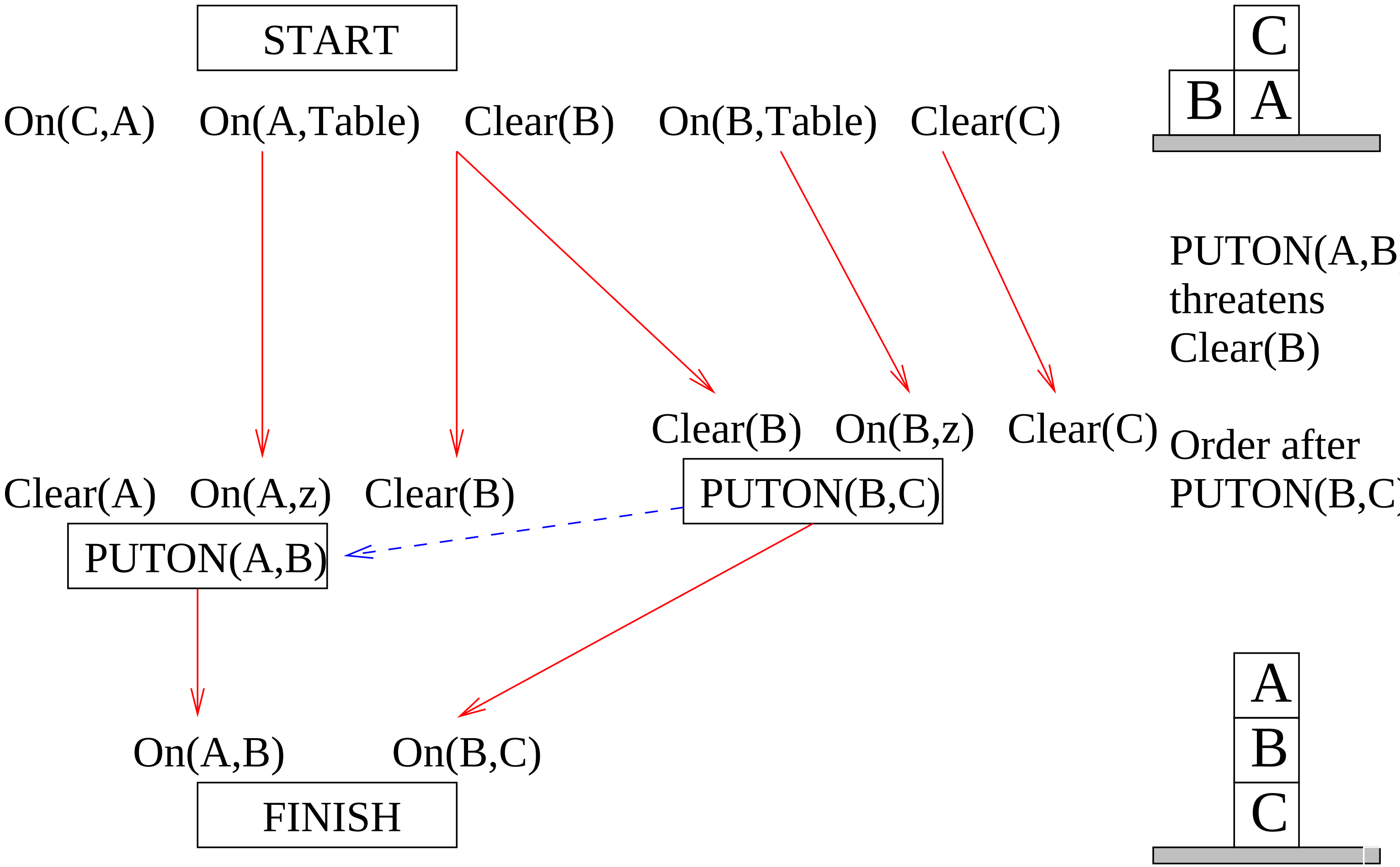
PUTON(B,C)

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

FINISH



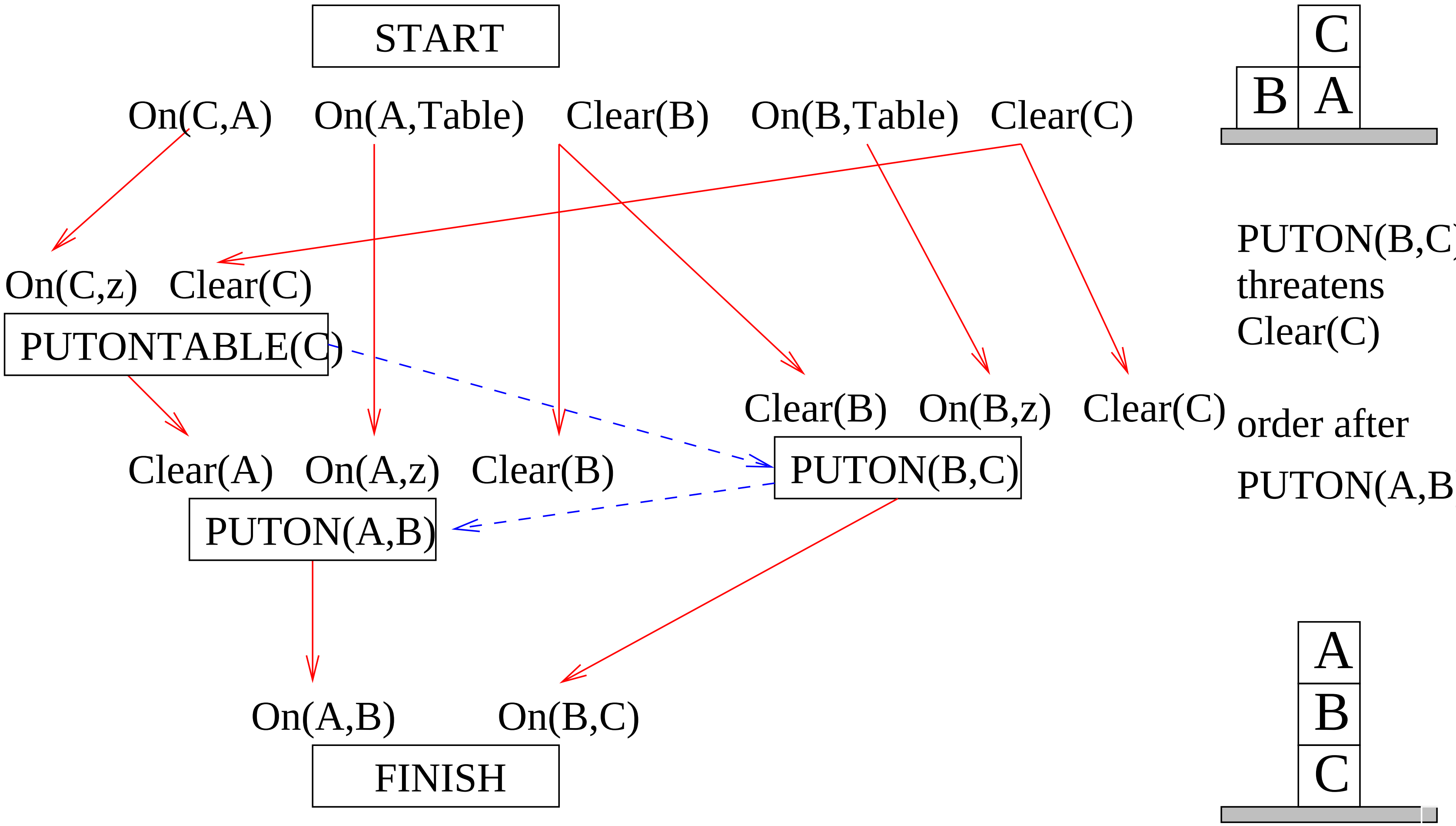
Sussman anomaly (cont'd)



PUTON(A,B) threatens Clear(B)

Order after PUTON(B,C)

Sussman anomaly (cont'd)



PUTON(B,C)
threatens
Clear(C)

order after

PUTON(A,B)

Heuristics for POP

POP be made efficient with good heuristics derived from problem description

- Which plan to select?
- Which flaw to choose?
- (We will see more after planning graphs)

Sources for the slides

- AIMA textbook (3^{rd} edition)
- AIMA slides (http://aima.cs.berkeley.edu/)
- Writing Planning Domains and Problems in PDDL, by Patrik Haslum (http://users.cecs.anu.edu.au/ patrik/pddlman/writing.html)
- Weld, D.S. (1999). Recent advances in AI planning. *AI Magazine*, 20(2), 93-122.