Planning and Partial-Order Planning

Sections 11.1-11.3

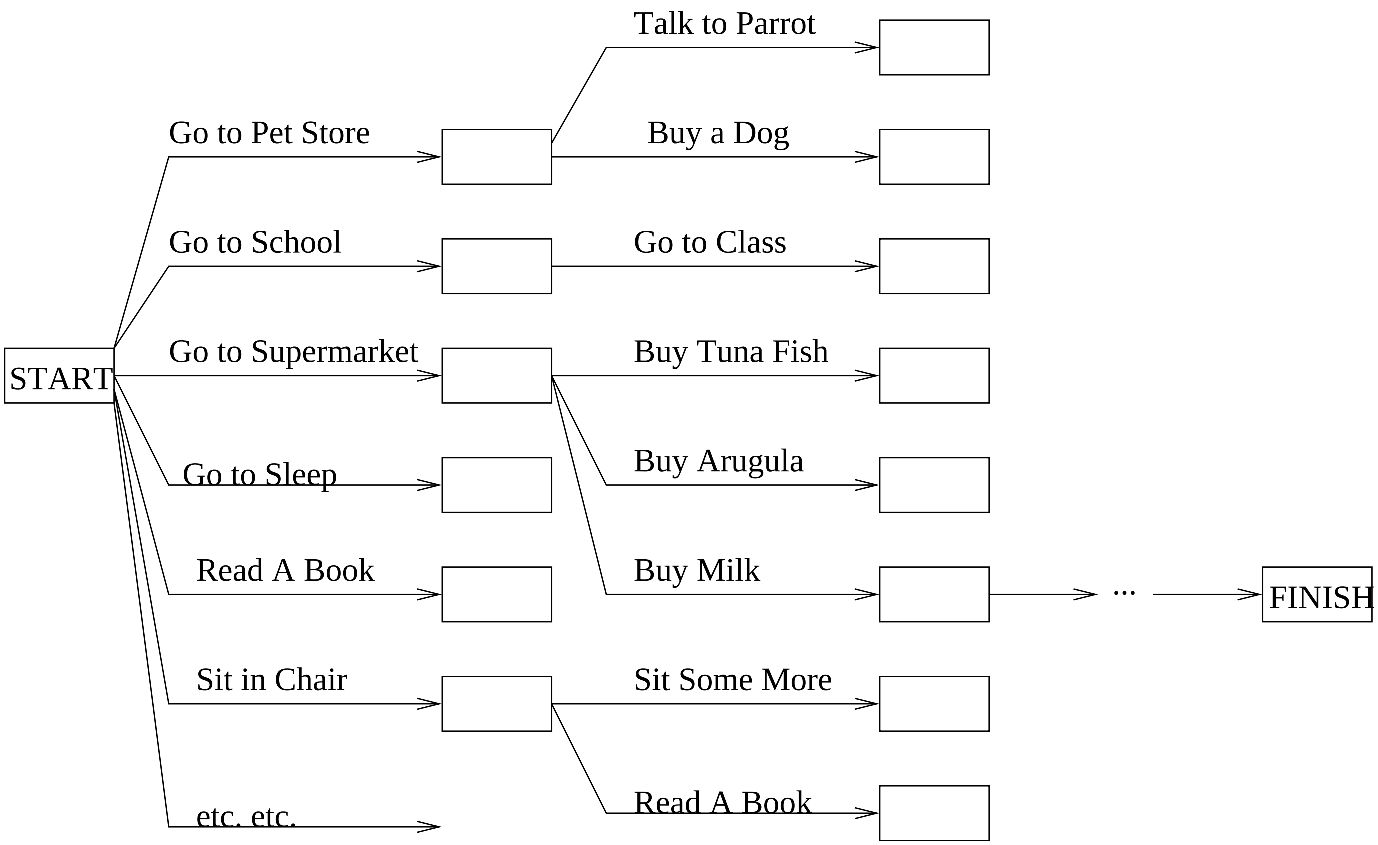
Outline

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

Additional reference used for the slides:
Weld, D.S. (1999). Recent advances in AI planning. *AI Magazine*, 20(2), 93-122.

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

Search vs. planning (cont'd)

	Search	**Planning**
States	Lisp data structures	Logical sentences
Actions	Lisp code	Preconditions/outcomes
Goal	Lisp code	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

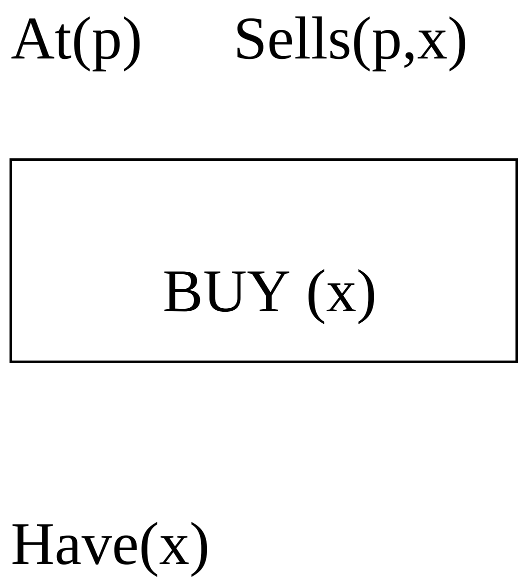
Search vs. planning (cont'd)

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

STRIPS operators

Tidily arranged actions descriptions, restricted language



ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p, x)$
EFFECT: $Have(x)$

STRIPS operators

ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p,x)$
EFFECT: $Have(x)$

[Note: this abstracts away many important details!]

Restricted language $\implies$ efficient algorithm
Precondition: conjunction of positive literals
Effect: conjunction of literals

(A complete set of STRIPS operators can be translated into a set of successor-state axioms)

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

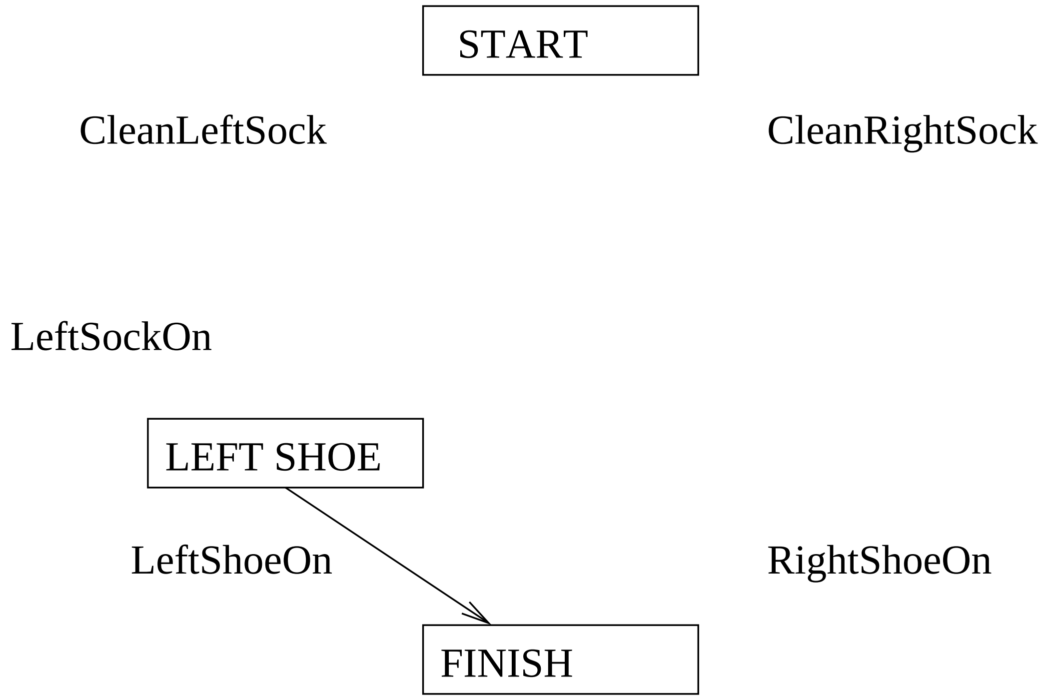
OC=
LeftShoeOn for FINISH
RightShoeOn for FINISH

LeftShoeOn

RightShoeOn

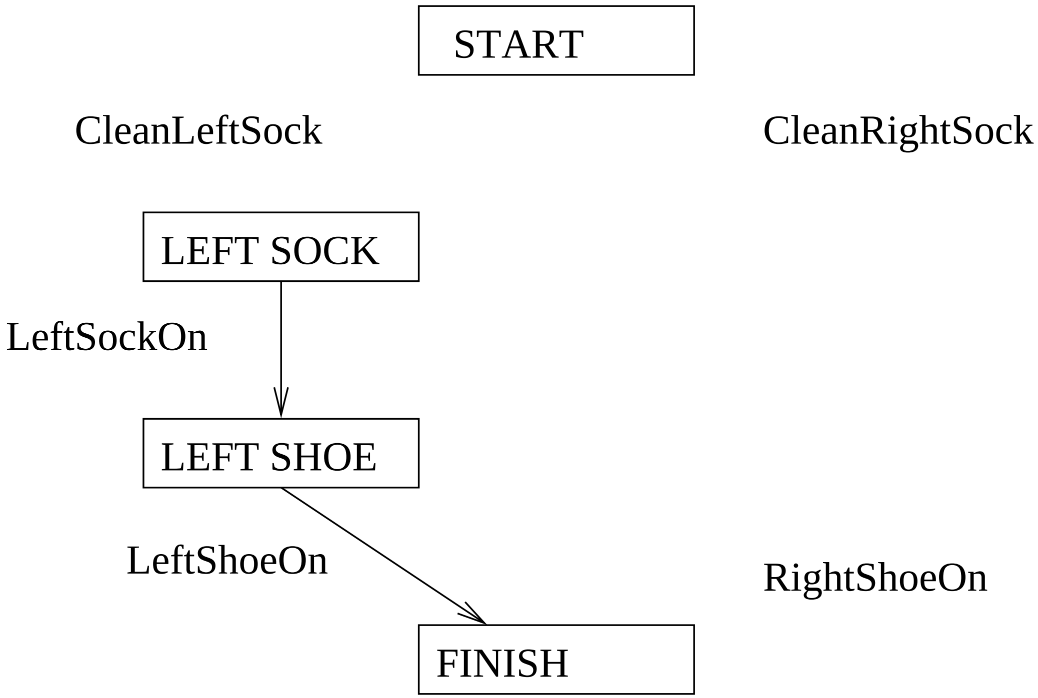
FINISH

Example (cont'd)



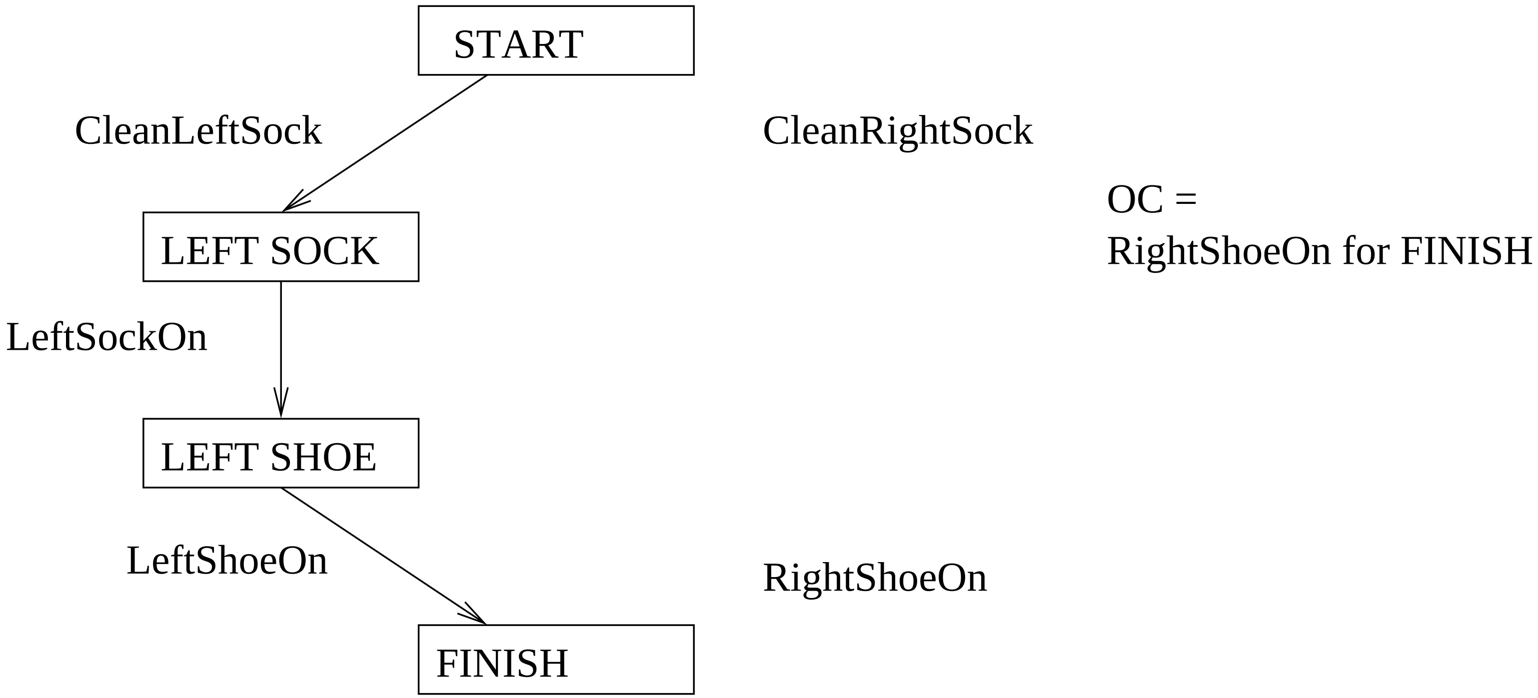
OC=
RightShoeOn for FINISH
LeftSockOn for LEFTSHOE

Example (cont'd)

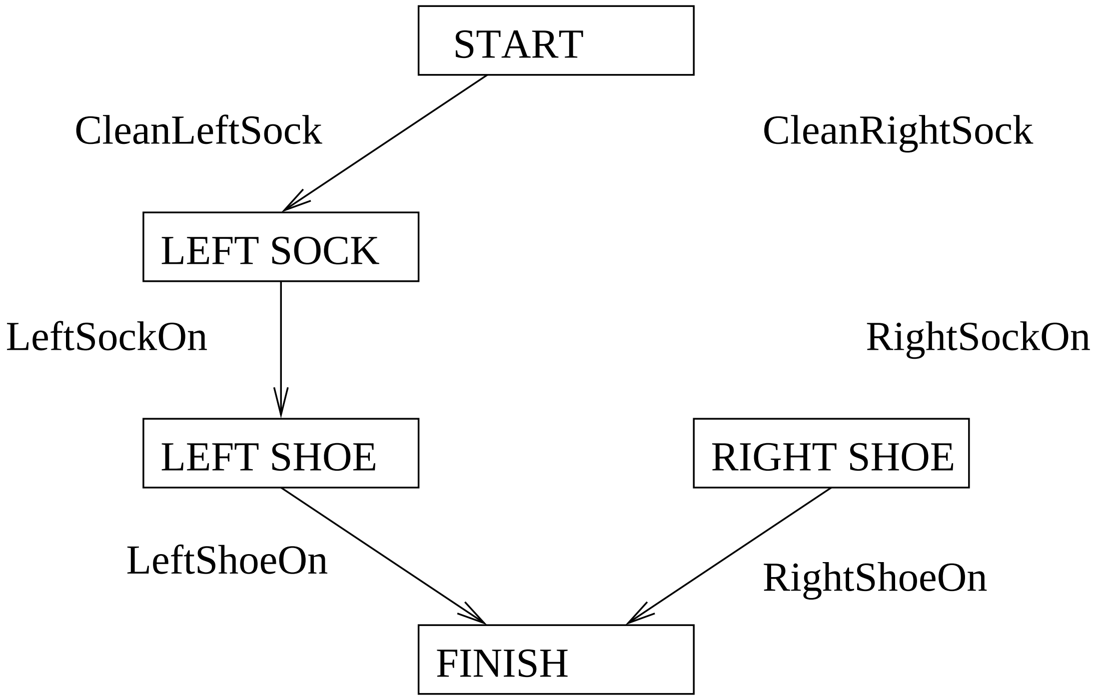


OC =
CleanLeftSock for LEFTSOCK
RightShoeOn for FINISH

Example (cont'd)

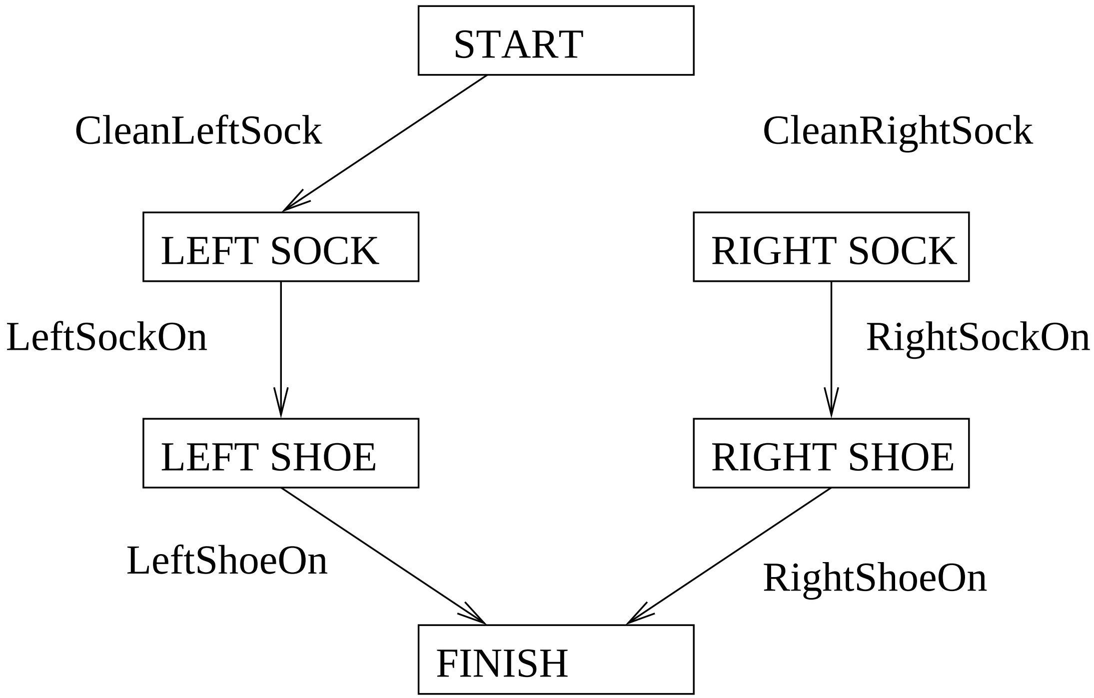


Example (cont'd)



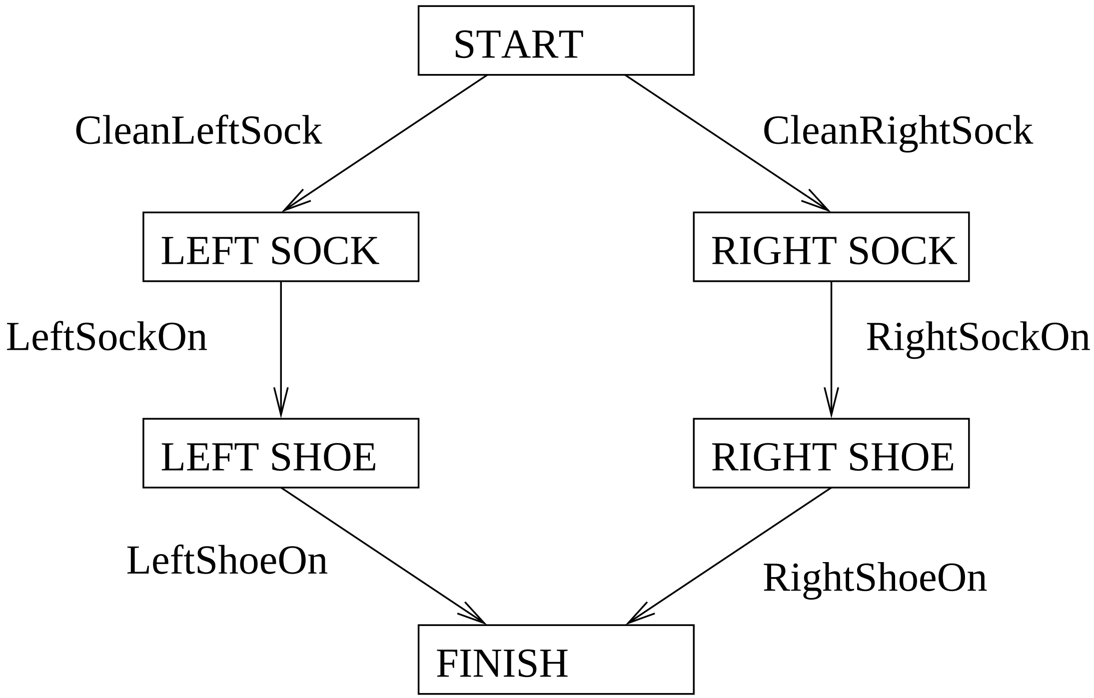
OC =
RightSockOn for RIGHTSHOE

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, fringe*)
returns a solution, or failure

fringe ← INSERT(MAKE-NODE(INITIAL-STATE [*problem*]),*fringe*)
loop do
  if EMPTY?(*fringe*) **then return** failure
  node ← REMOVE-FIRST(*fringe*)
  if GOAL-TEST[*problem*] applied to STATE[*node*] succeeds
    then return SOLUTION(*node*)
  fringe ← INSERT-ALL(EXPAND(*node, problem*), *fringe*)

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.

POP algorithm specifics (cont'd)

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

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*

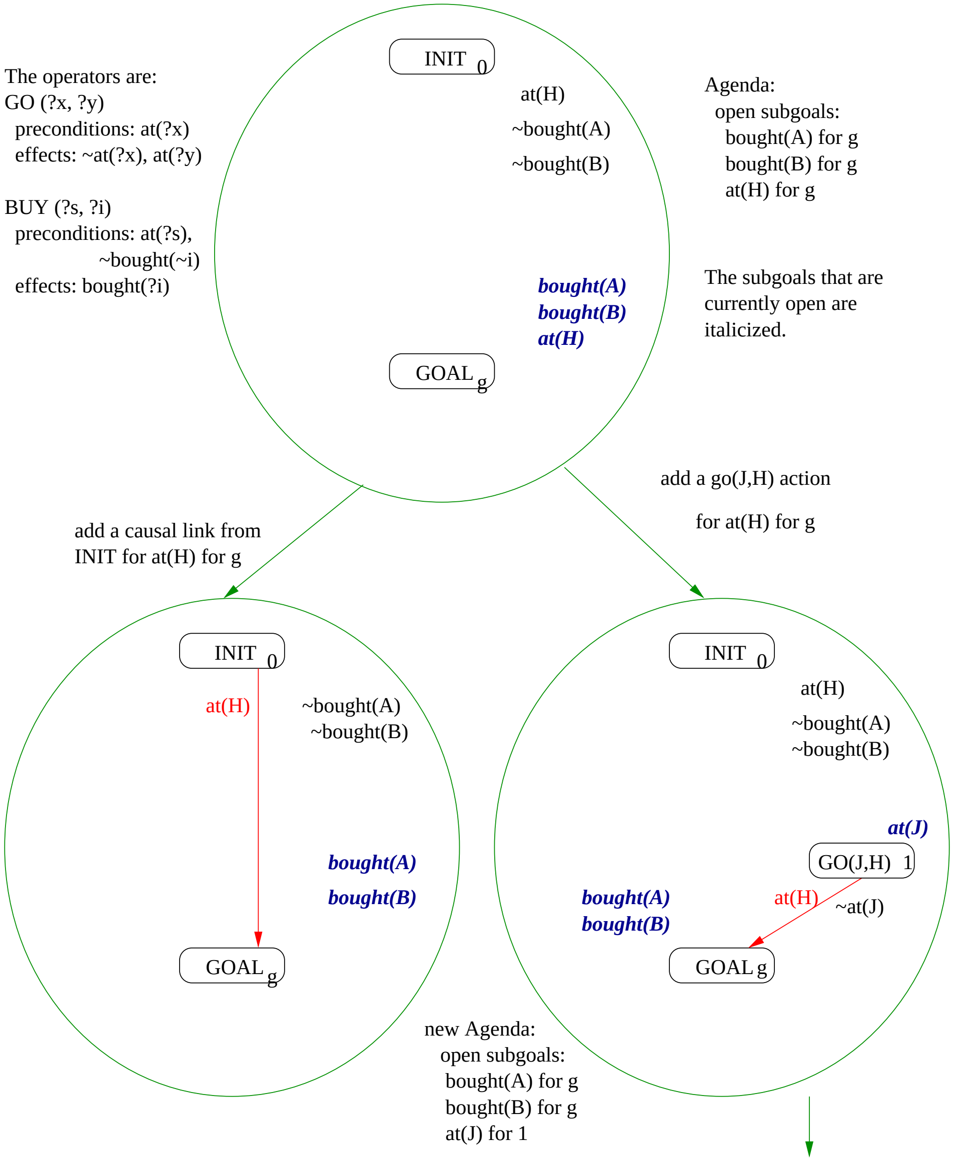
Shopping example

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)

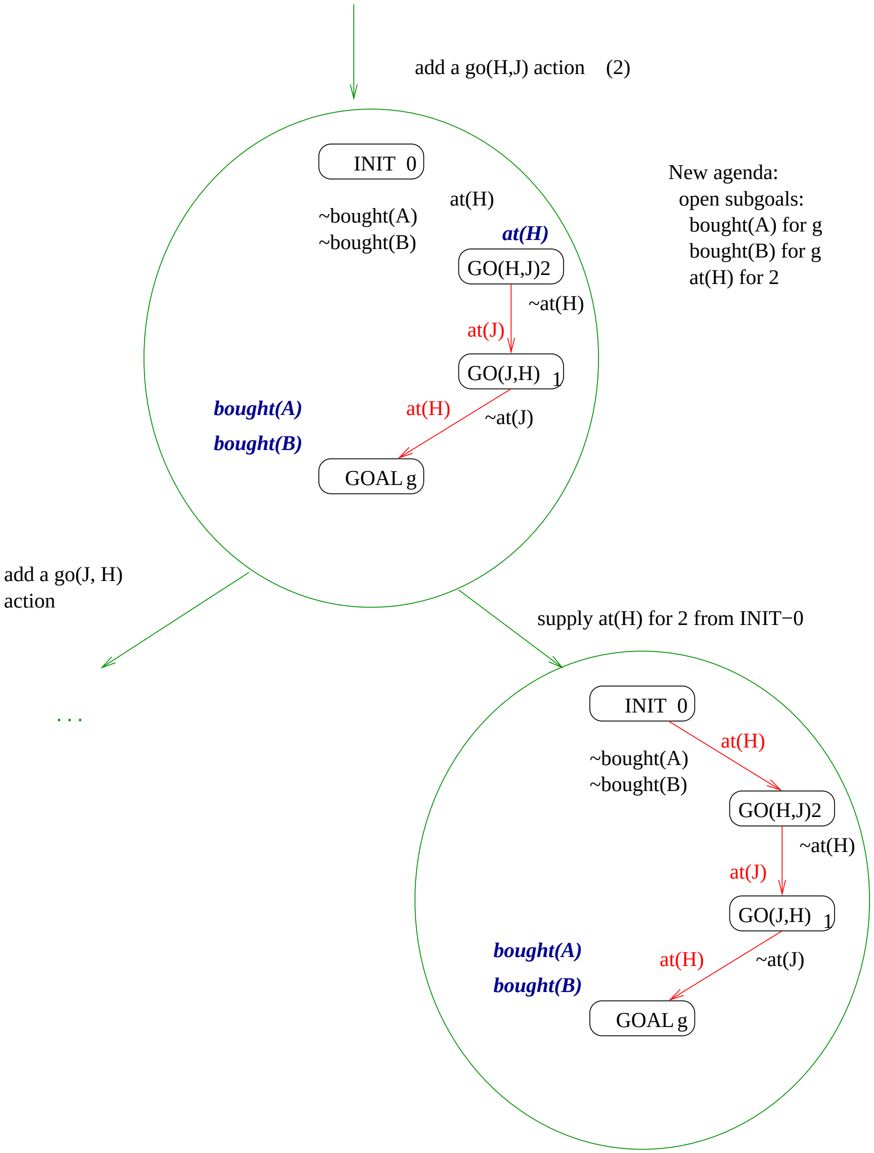
Agenda:
  open subgoals:
    bought(A) for g
    bought(B) for g
    at(H) for g

The subgoals that are currently open are italicized.

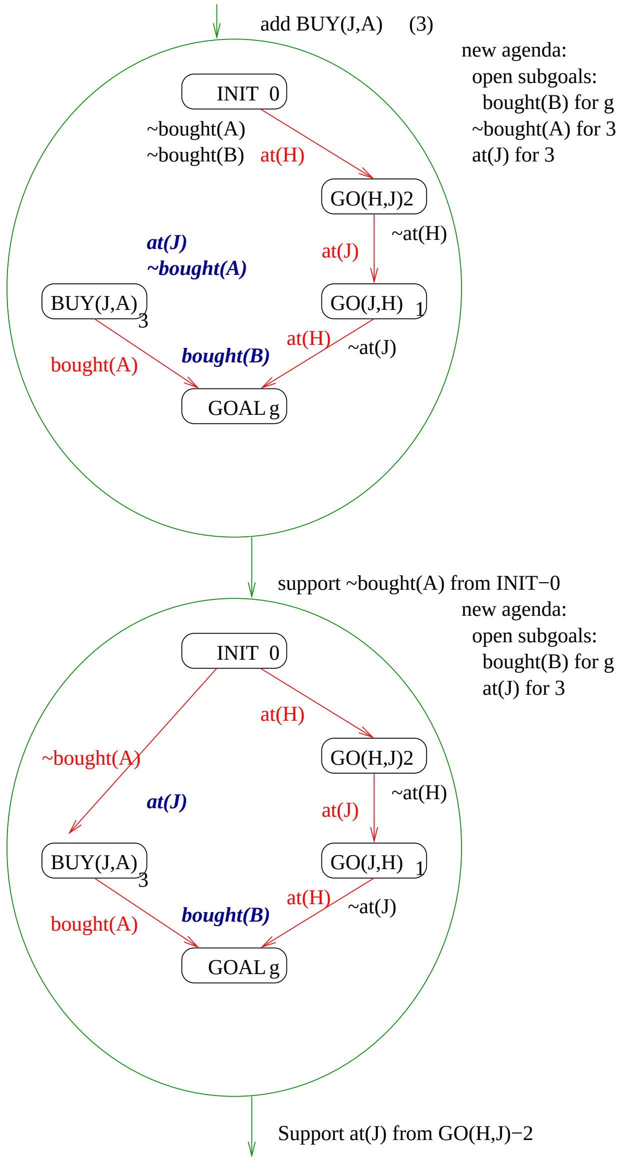


new Agenda:
  open subgoals:
    bought(A) for g
    bought(B) for g
    at(J) for 1

Shopping example (cont'd)

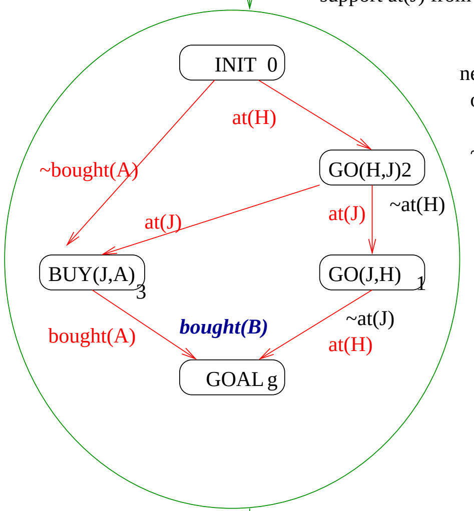


Shopping example (cont'd)



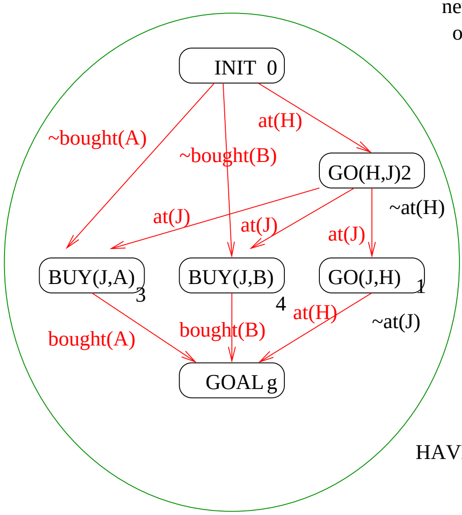
Shopping example (cont’d)

support at(J) from GO(H,J)−2



new agenda:
open subgoals:
bought(B) for g
~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 INIT−0

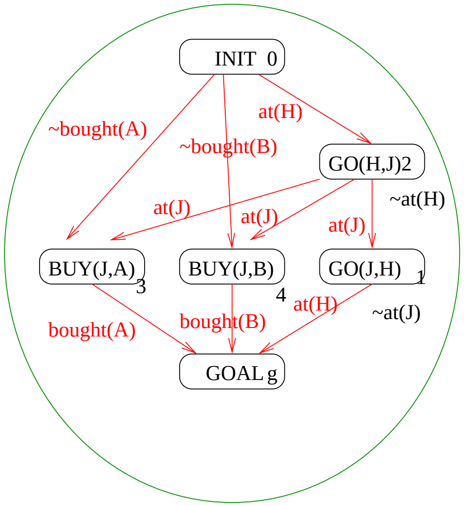
new agenda:
open subgoals:
at(J) for 4

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)

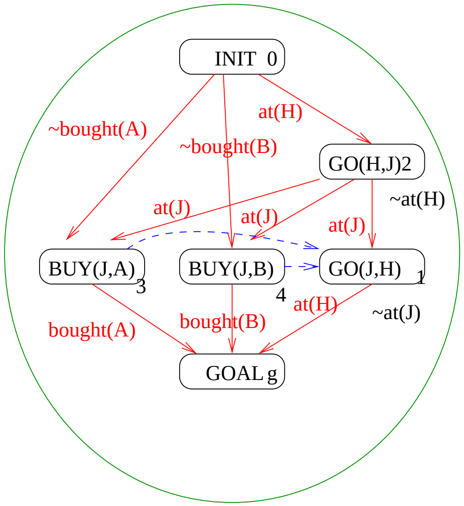


Now, the solution is a possible ordering of this plan. Those are:

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.

Another shopping example

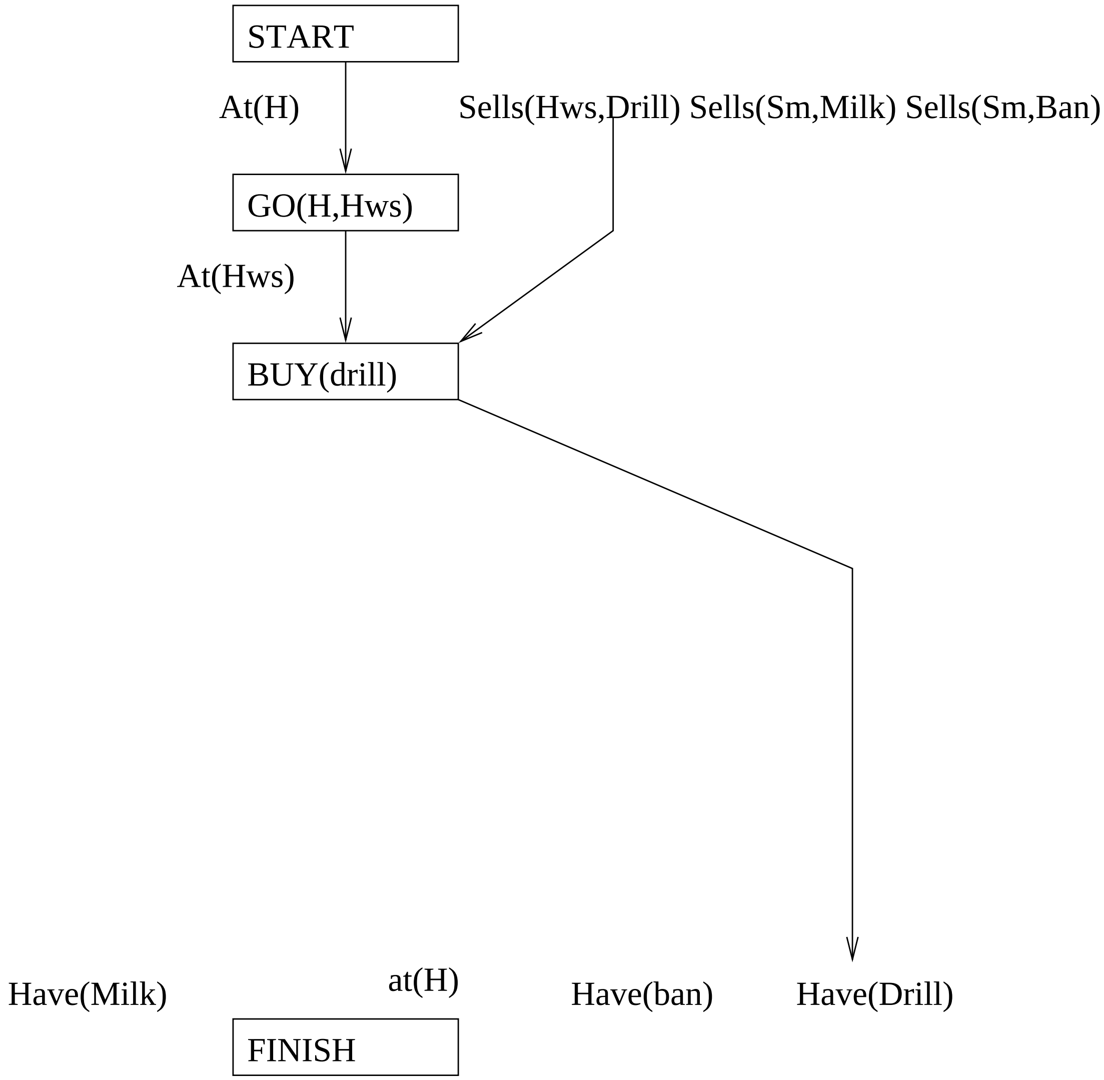
START

At(H) Sells(Hws,Drill) Sells(Sm,Milk) Sells(Sm,Ban)

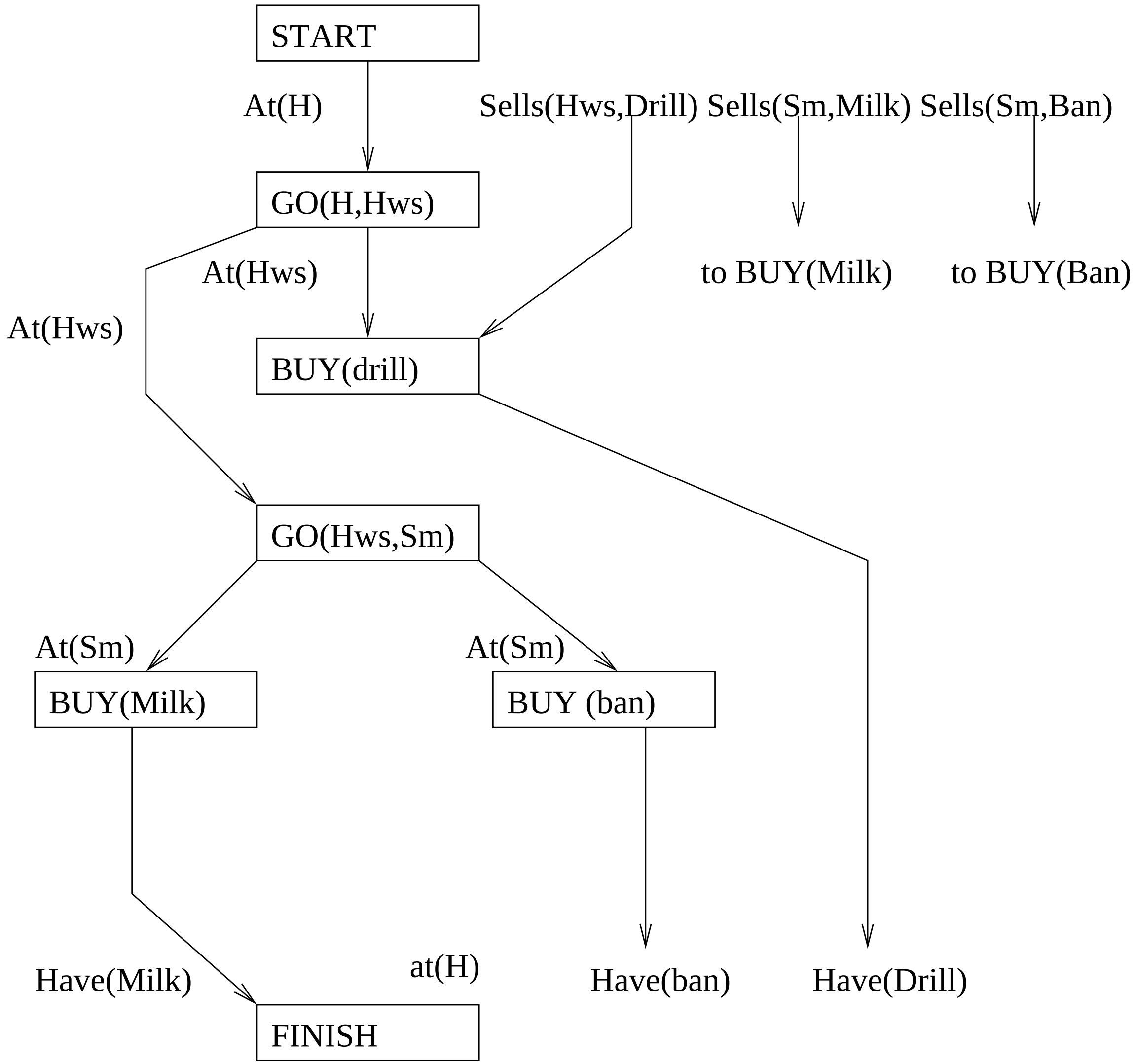
Have(Milk) at(H) Have(ban) Have(Drill)

FINISH

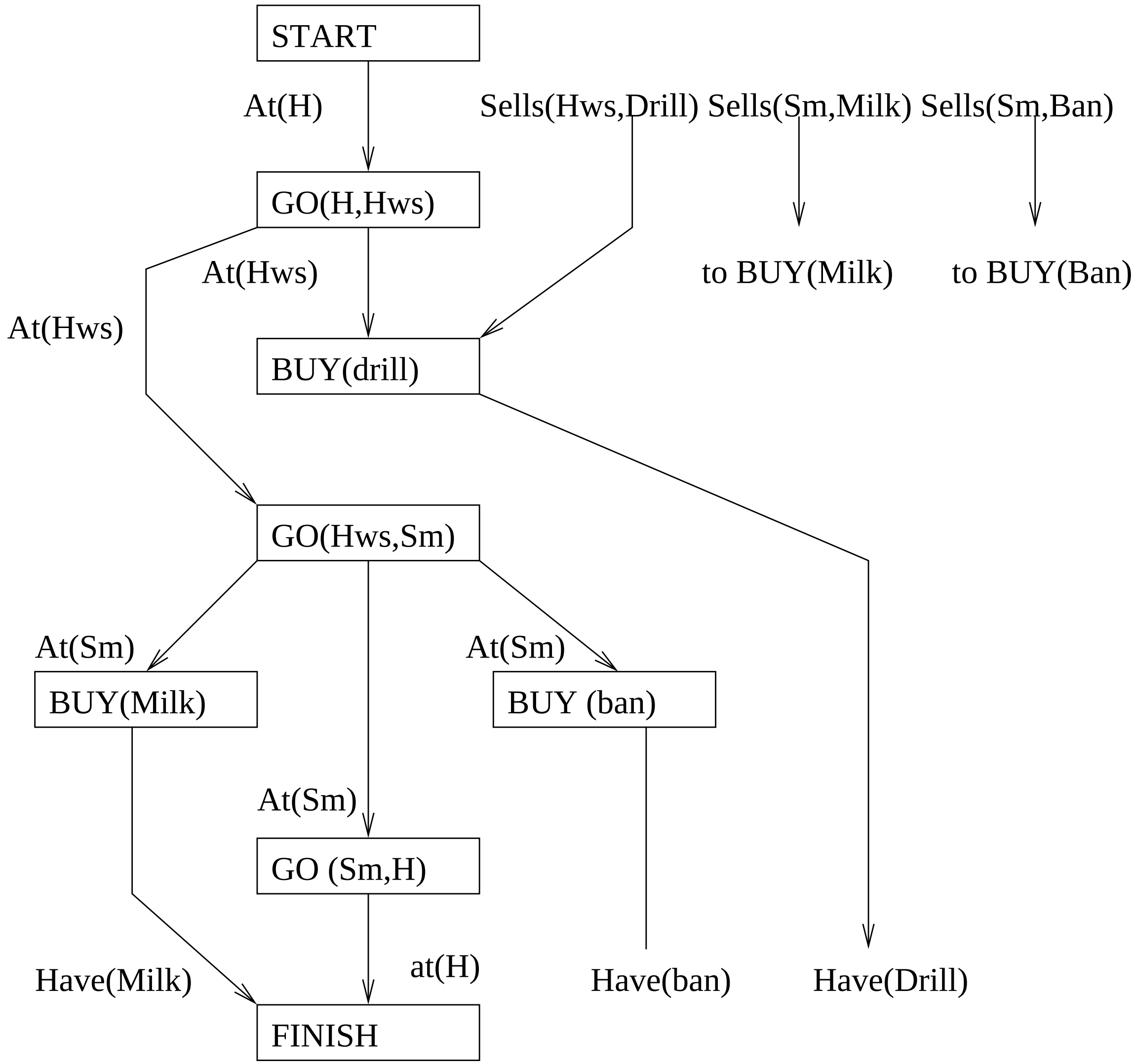
Another shopping example (cont'd)



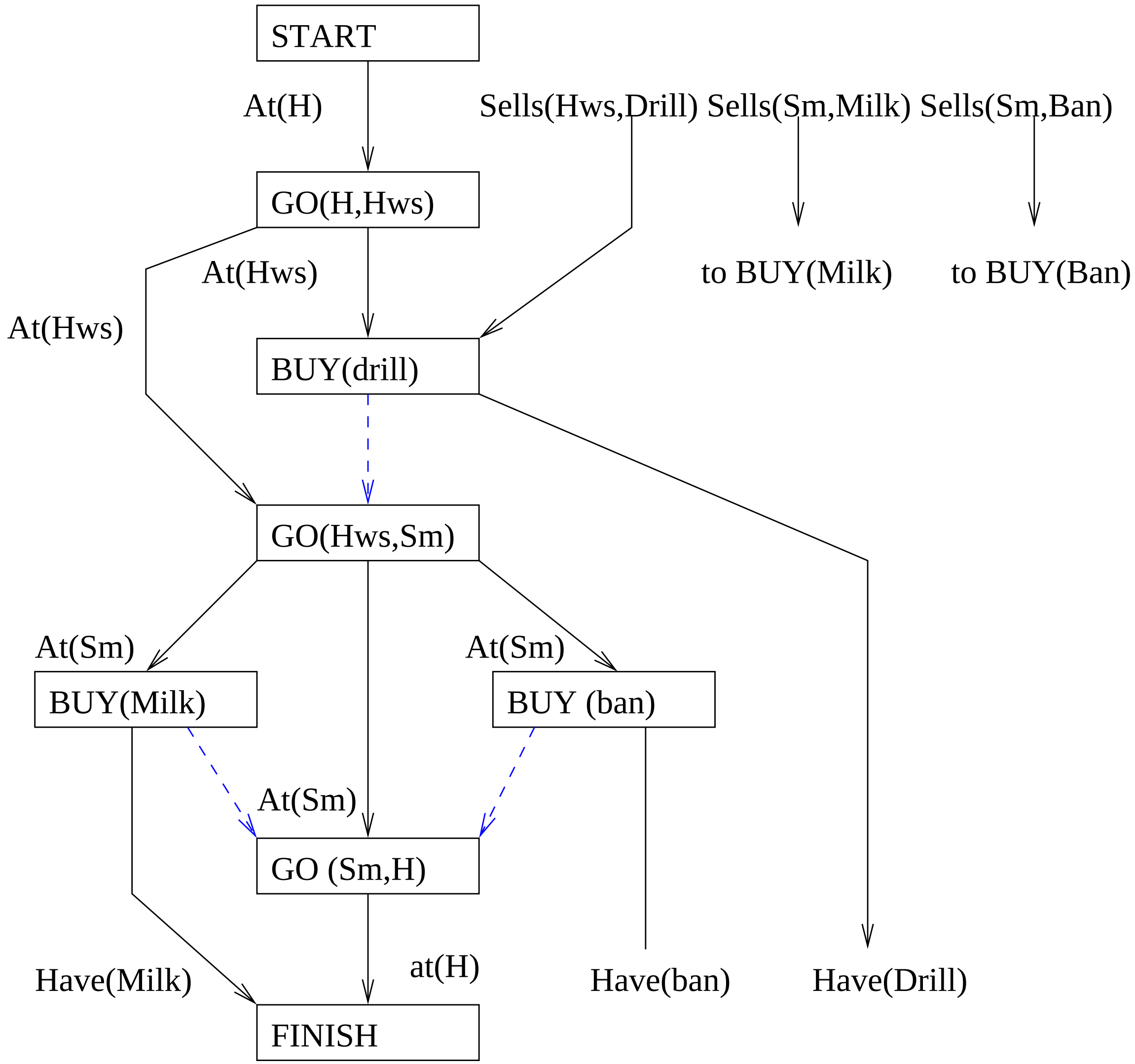
Another shopping example (cont'd)



Another shopping example (cont'd)

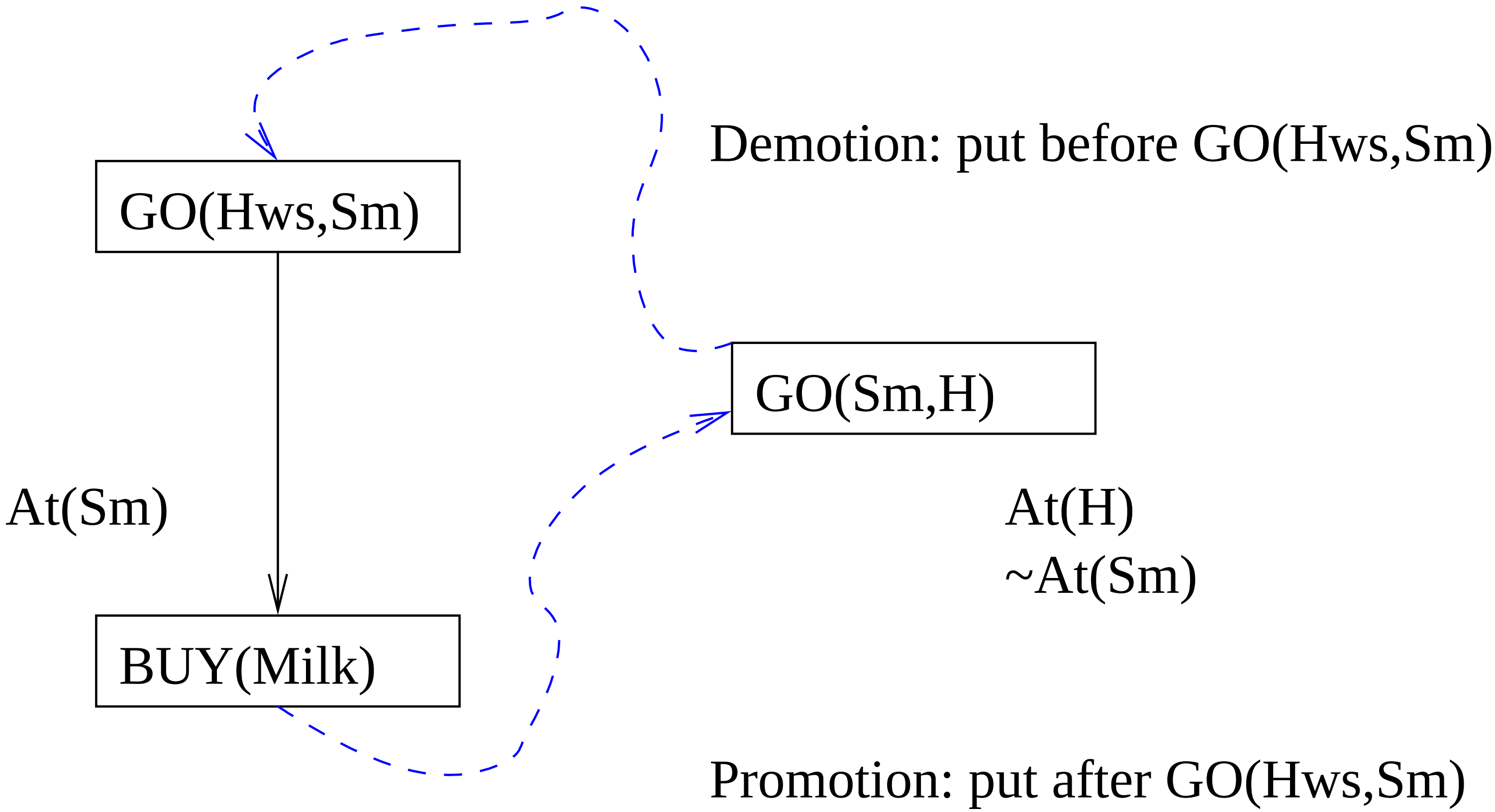


Another shopping example (cont'd)



Threats and promotion/demotion

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



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

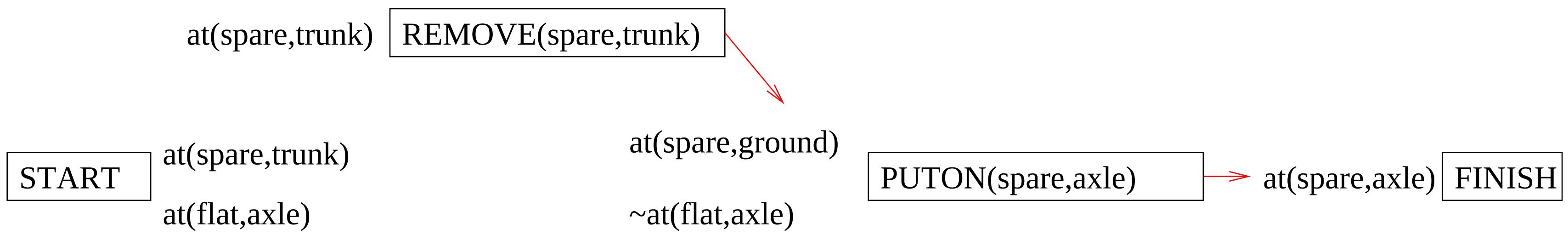
Heuristics for POP

- POP be made efficient with good heuristics derived from problem description
  - Which plan to select?
  - Which flaw to choose?
  - More after planning graphs
- Two additional POP examples follow. The flat tire example shows the effect of inserting an “impossible” action. The Sussman anomaly shows that “divide-and-conquer” is not always optimal.

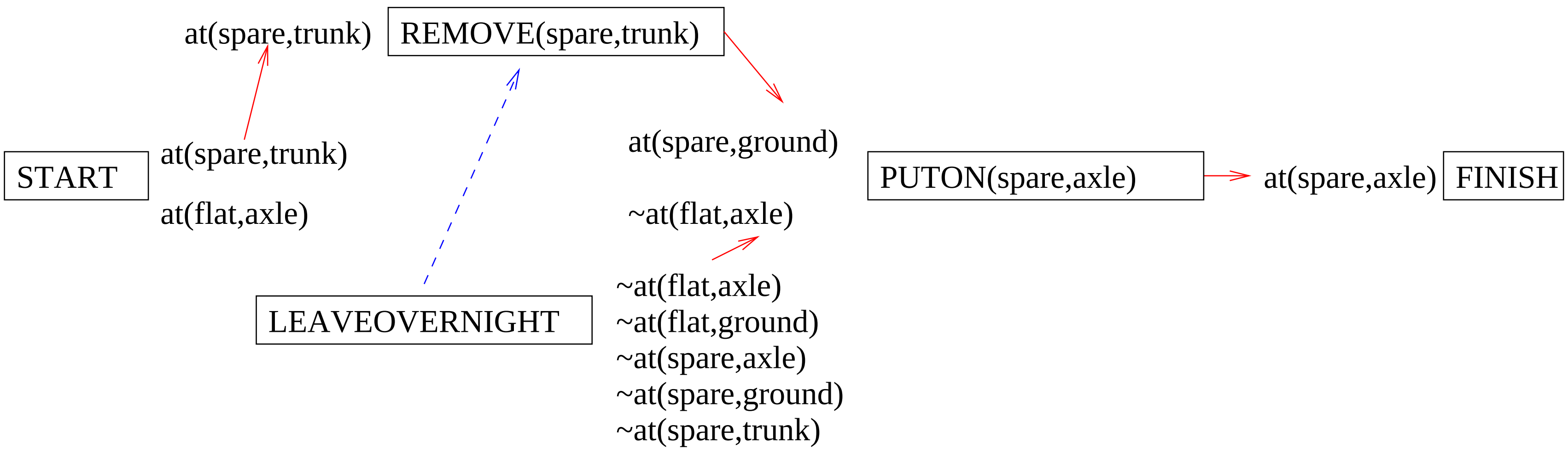
The flat tire domain

Init(At(Flat,Axle) $\wedge$ At(Spare,Trunk))
Goal(At(Spare,Axle))
Action(REMOVE(spare,trunk),
    Precond: At(spare,trunk)
    Effect: $\neg$At(spare,trunk) $\wedge$ At(spare,ground)
Action(REMOVE(flat,axle),
    Precond: At(flat,axle)
    Effect: $\neg$At(flat,axle) $\wedge$ At(flat,ground)
Action(PUTON(spare,axle),
    Precond: At(spare,ground) $\wedge$ $\neg$ at(flat,axle)
    Effect: $\neg$At(spare,ground) $\wedge$ At(spare,axle)
Action(LEAVEOVERNIGHT
    Precond:
    Effect: $\neg$At(spare,ground) $\wedge$ $\neg$ At(spare,axle)
        $\neg$At(spare,trunk) $\wedge$ $\neg$ At(flat,ground)
        $\neg$At(flat,axle)

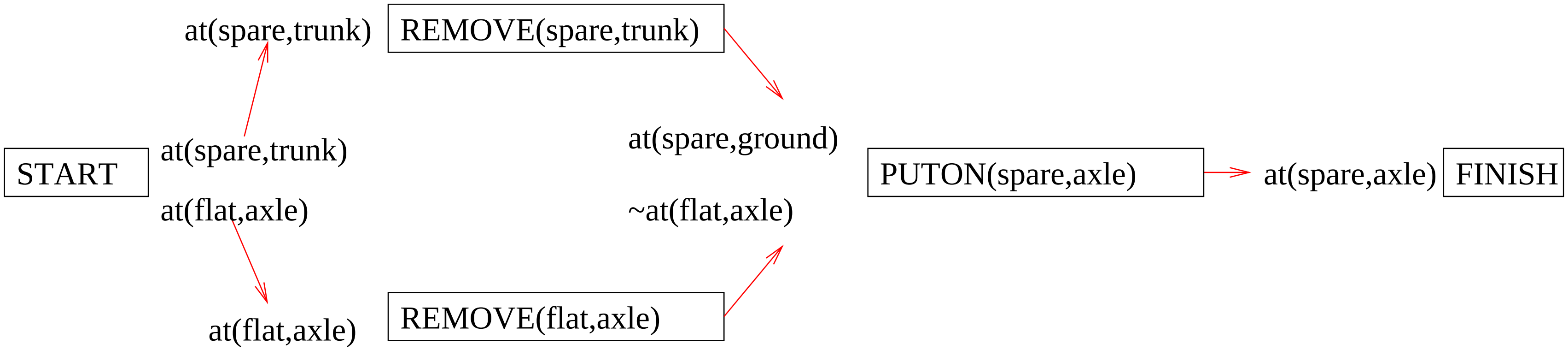
The flat tire plan



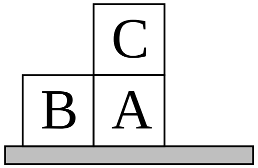
The flat tire plan (cont'd)



The flat tire plan (cont'd)



Sussman anomaly



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

PUTON(x,y)

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

Clear(z)   On(x,y)

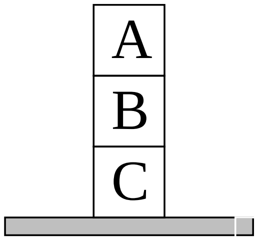
Clear(x)   On(x,z)

PUTONTABLE(x)

~On(x,z)

Clear(z)   On(x,Table)

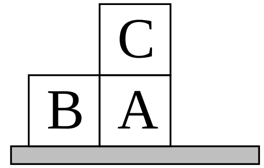
+ several inequality constraints



Sussman anomaly (cont'd)

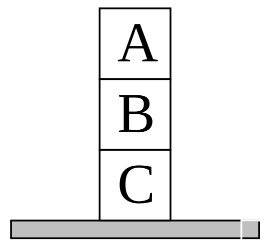
FINISH

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



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

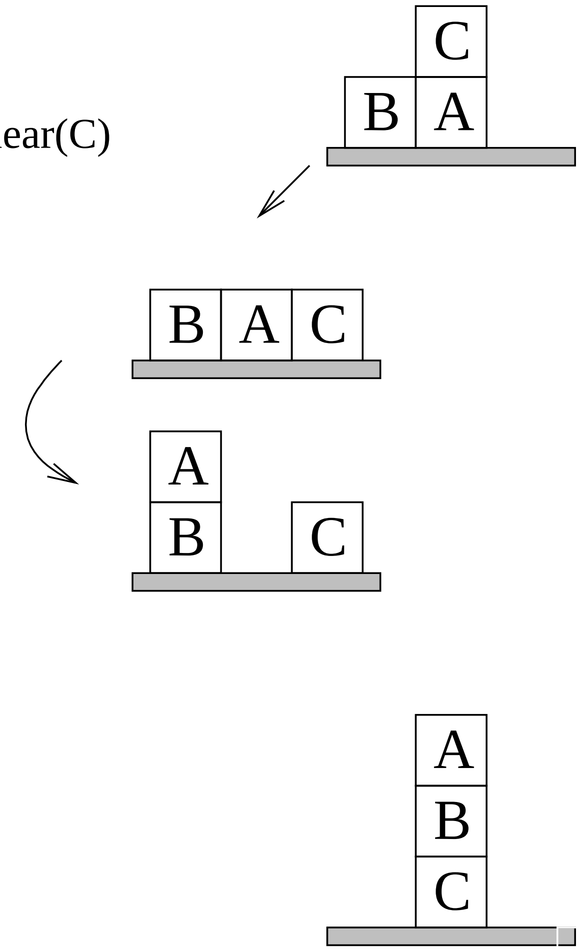
FINISH



Sussman anomaly (cont'd)

FINISH

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

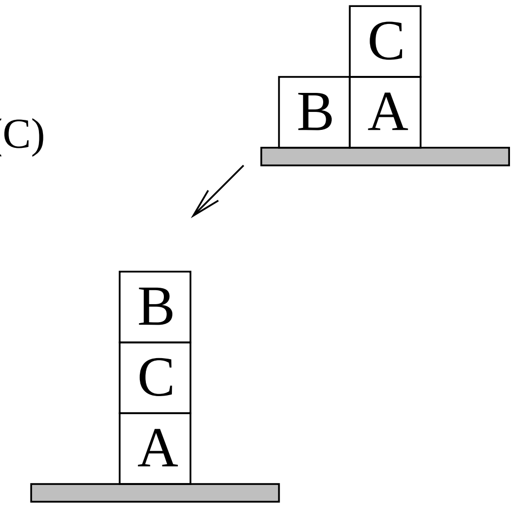
On(A,B)   On(B,C)

FINISH

Sussman anomaly (cont'd)

FINISH

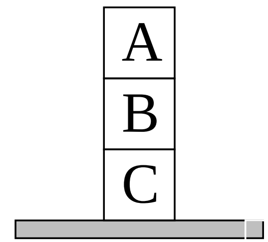
On(C,A)   On(A,Table)   Clear(B)   On(B,Table)   Clear(C)



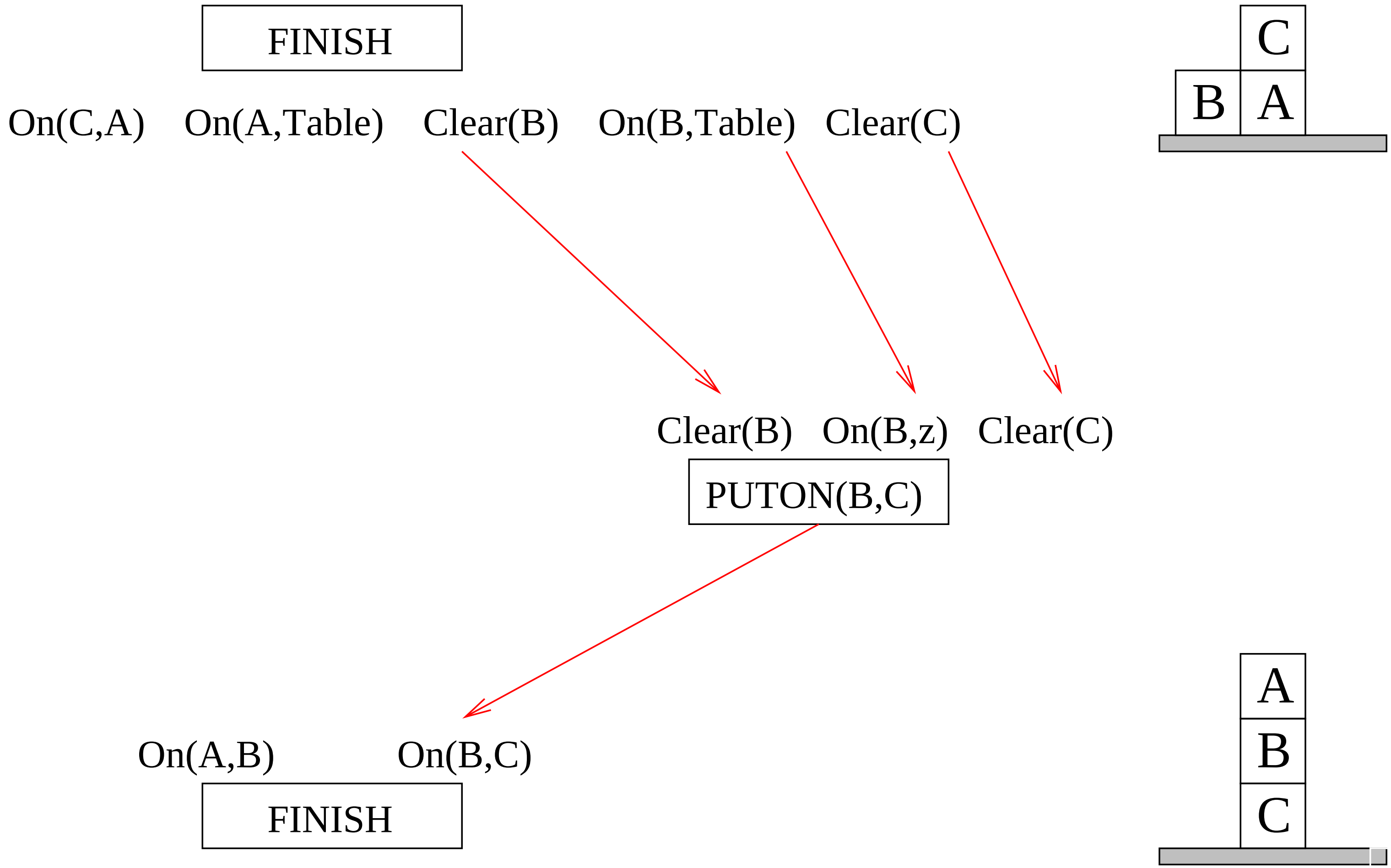
If we try the second goal (on (B,C)) first,
we can't proceed without undoing work.

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

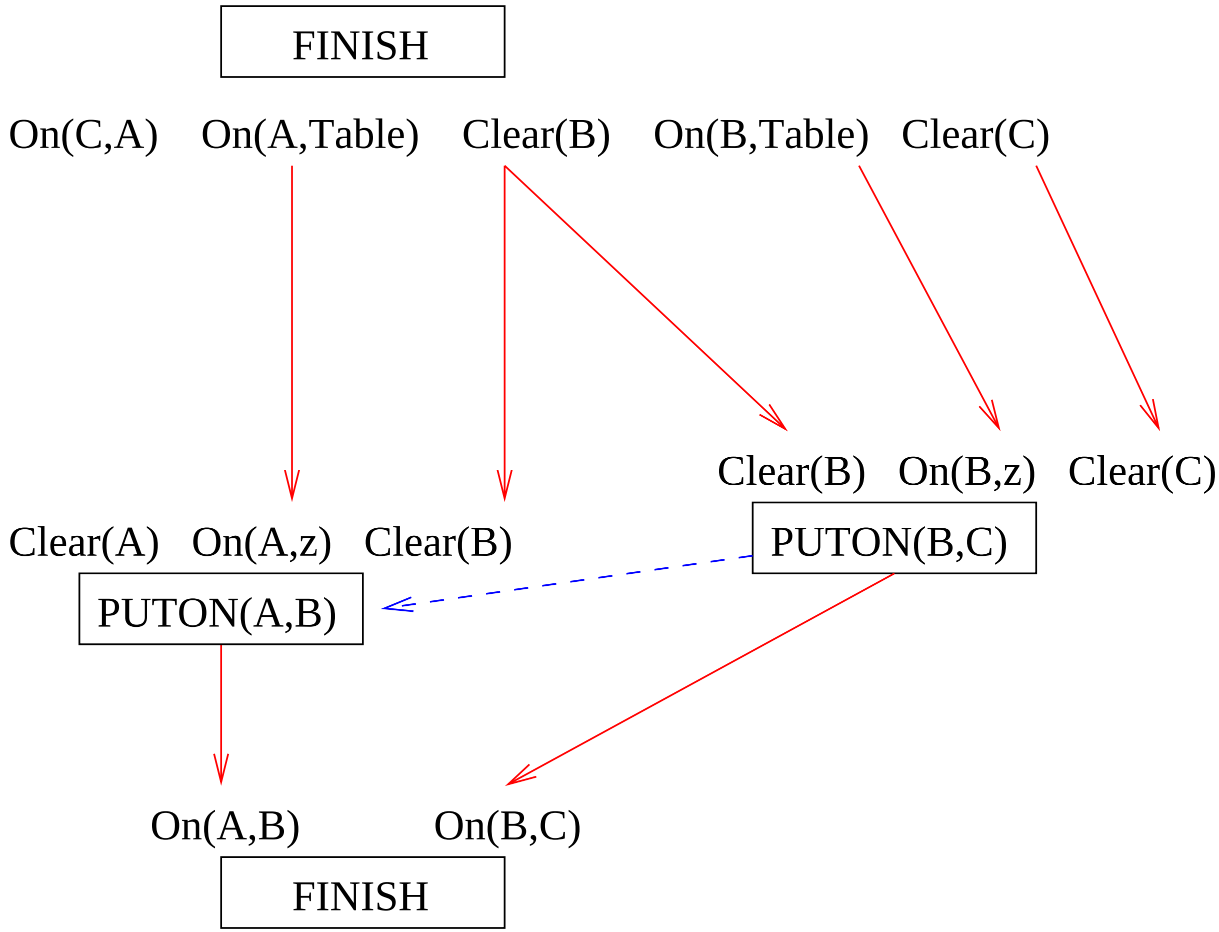
FINISH

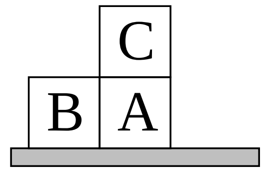


Sussman anomaly (cont'd)



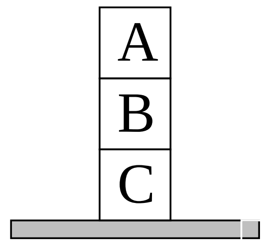
Sussman anomaly (cont’d)



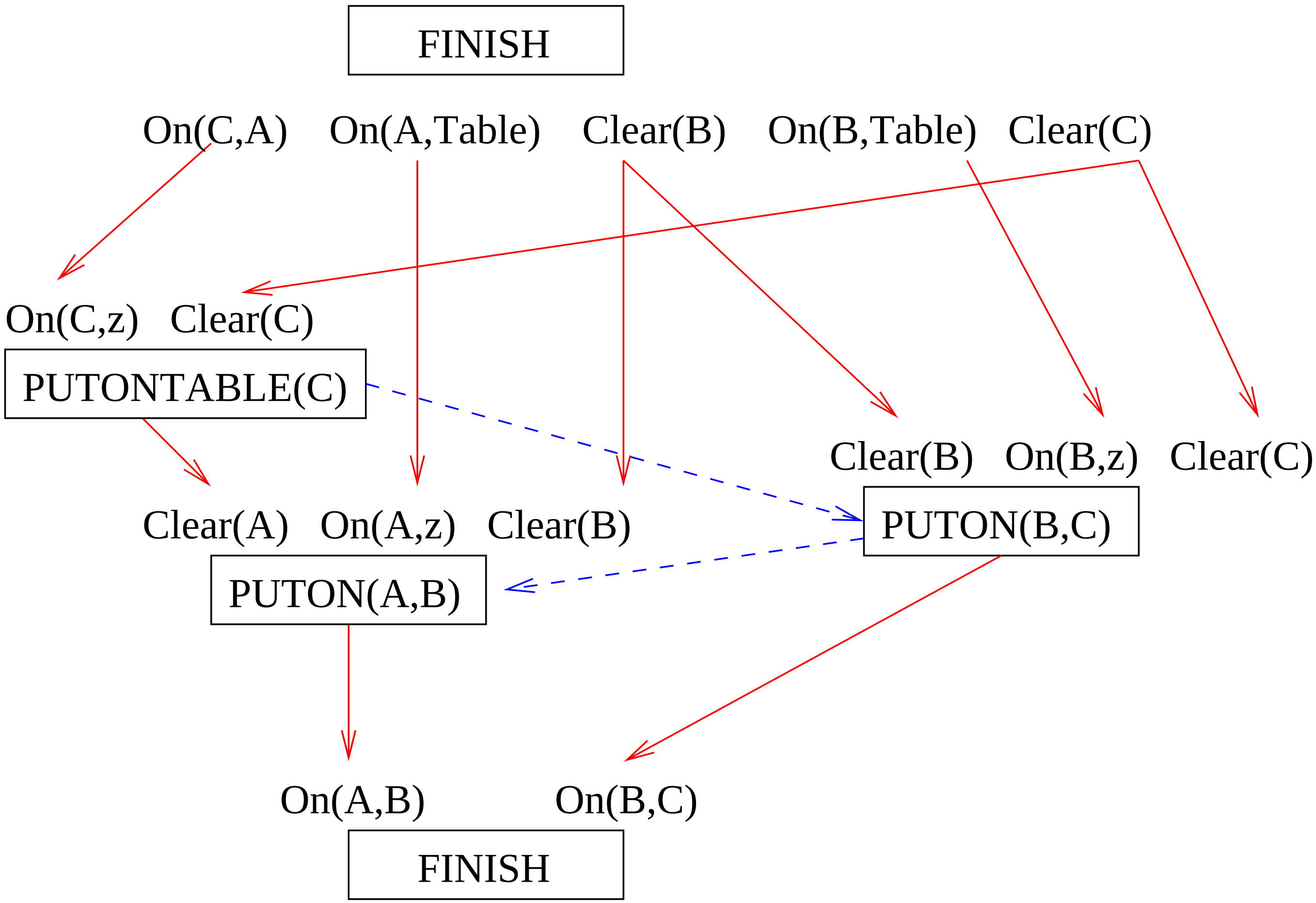


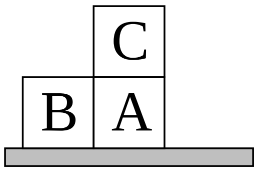
PUTON(A,B) threatens Clear(B)

Order after PUTON(B,C)



Sussman anomaly (cont'd)





PUTON(B,C)
threatens
Clear(C)

order after

PUTON(A,B)