PLANNING AND SCHEDULING: PLAN-SPACE PLANNING (PSP)

Hochschule Bonn-Rhein-Sieg

b-it Bonn-Aachen International Center for Information Technology

Prof. Dr.-Ing. Gerhard K. Kraetzschmar

Acknowledgements

- These slides are based on those slides by Dana Nau and Manuela Veloso
- Some improvements have been added by Iman Awaad

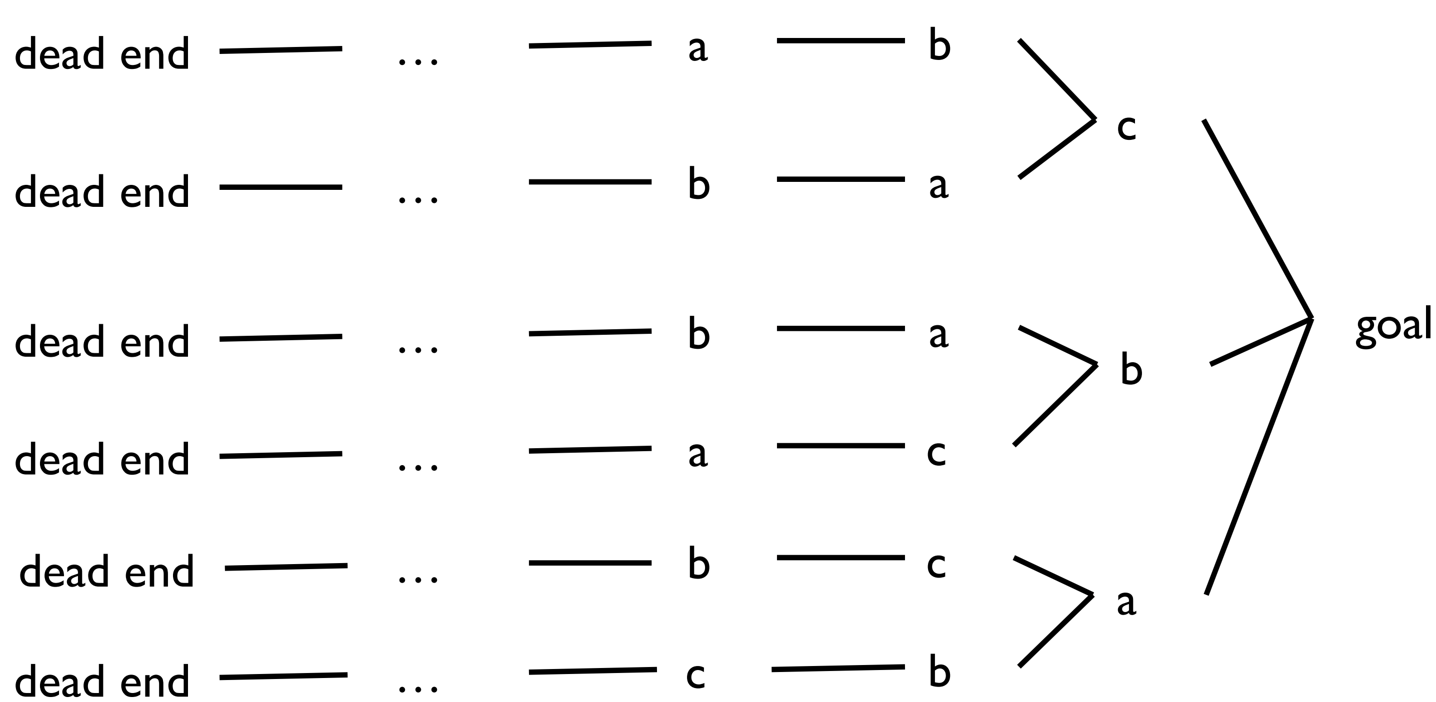
Remember?

- Planning as search...
- Which search space?

- State Space
  - Each node represents a state of the world
  - A plan is a path through the space
- Plan Space
  - Each state is a partially complete plan,
    i.e. a set of partially instantiated operators and some constraints
  - We impose more and more constraints until we get a plan

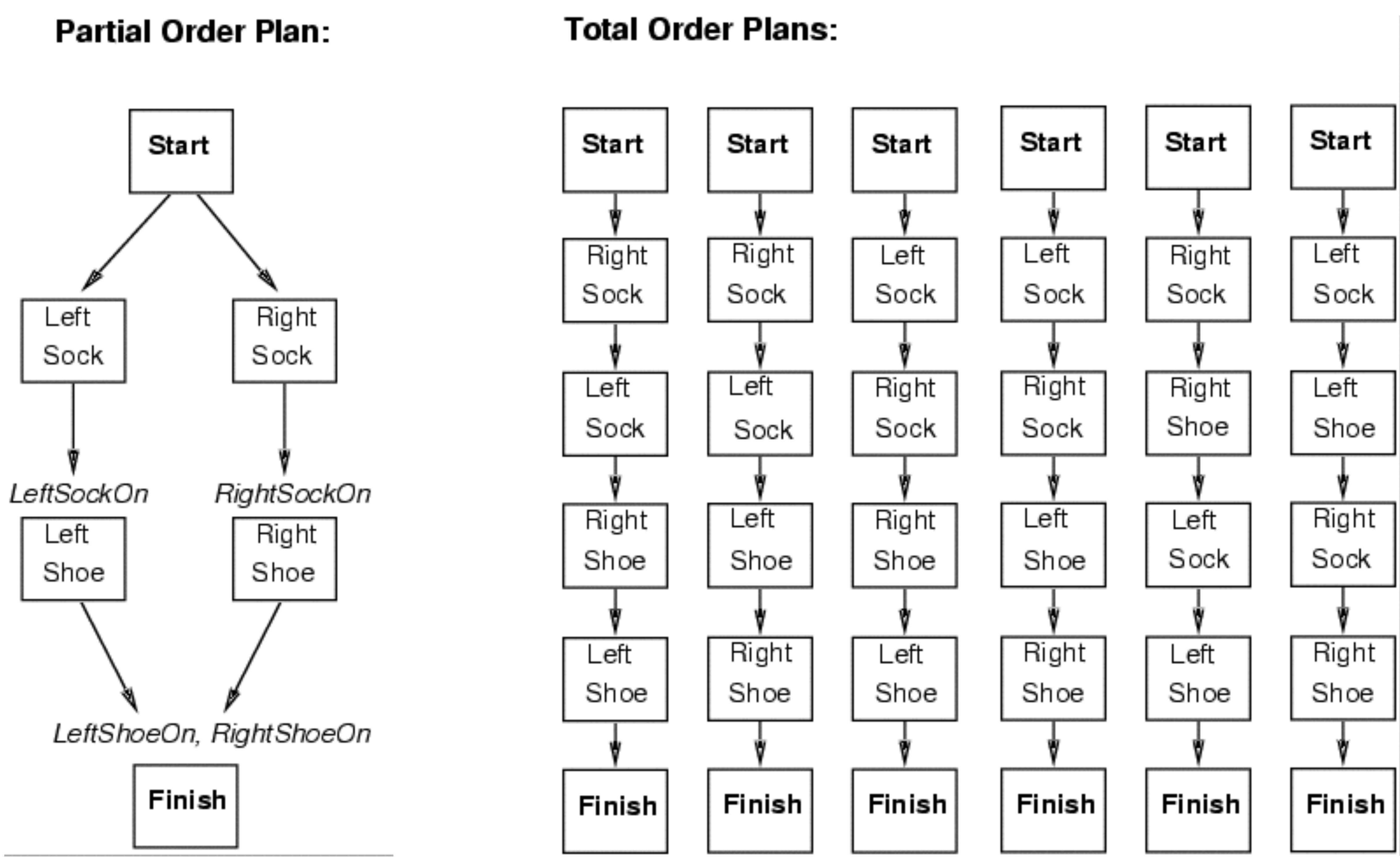
Motivation

- Problem with state-space search
  - In some cases we may try many different orderings of the same actions before realising there is no solution



- PSP: adopt a least-commitment strategy:
  Do not commit to orderings, instantiations, etc., until necessary.

Partial order vs total order plans

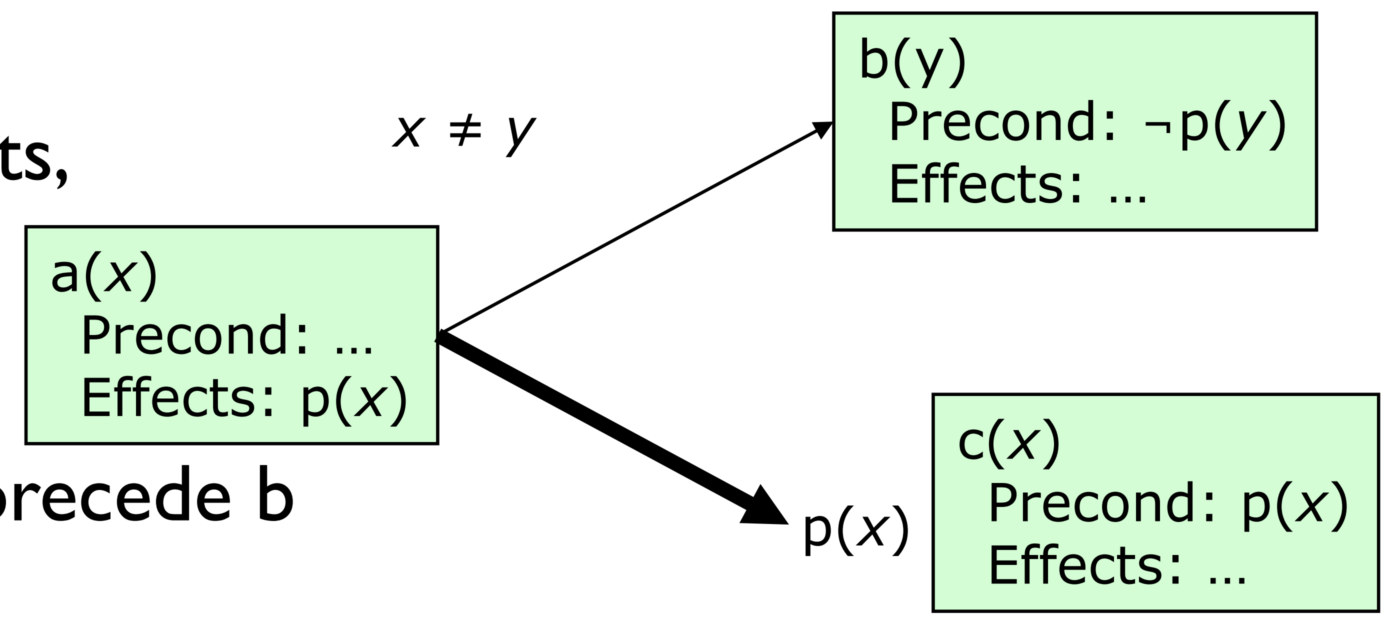


Outline

- Basic idea of plan space planning
- PSP plan representation
- What we mean by constraints
- Flaws & their resolutions
- The PSP algorithm
- An example
- Comments
- Partially-ordered vs totally-ordered plans
- The POP algorithm

Plan-Space Planning: Basic Idea

- Backward search from the goal
- Each node of the search space is a partial plan
  - A set of partially-instantiated operators, called steps
  - Several sets of constraints
- Make more and more refinements, until we have a solution

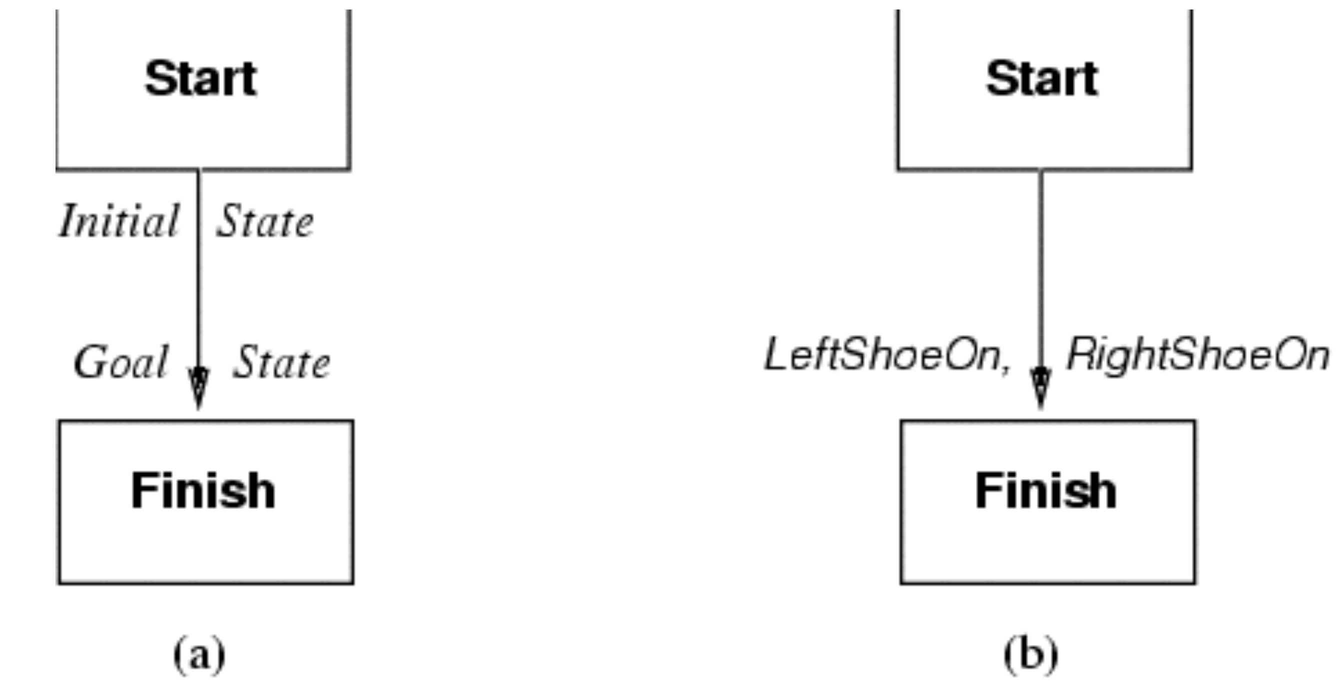


- Types of constraints:
  - Precedence constraints: a must precede b
  - Binding constraints:
    - Inequality constraints, e.g., v1 $\neq$ v2 or v $\neq$ c
    - Equality constraints (e.g., v1 = v2 or v = c) or substitutions
  - Causal links: use step a to establish the precondition p needed by step c
- How to tell we have a solution: no more flaws in the plan
  - Will discuss flaws and how to resolve them

PSP Representation (I)

- A plan is a quadruplet <Steps, Orderings, Bindings, CausalLinks>
- Example

$$
\begin{aligned}
&Plan(\textsc{Steps}:\{S_1 : Op(\textsc{Action}:Start),\\
&\qquad S_2 : Op(\textsc{Action}:Finish,\\
&\qquad\quad \textsc{Prec}:RightShowOn \wedge LeftShowOn)\}\\
&\quad \textsc{Orderings}:\{S_1 \prec S_2\}\\
&\quad \textsc{Bindings}:\{\}\\
&\quad \textsc{CausalLinks}:\{\} \quad)
\end{aligned}
$$



PSP representation (II)

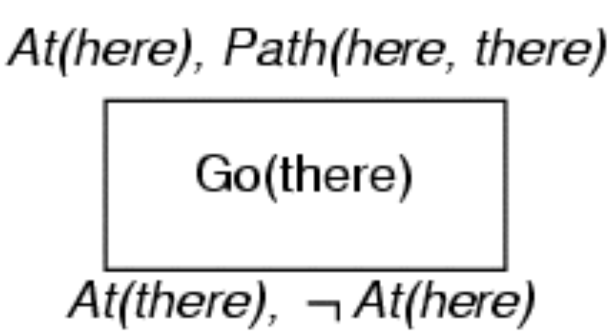
- Initial state: an arbitrary logical sentence

$$At(Home, S_0) \land \neg Have(Milk, S_0) \\ \land \neg Have(Bananas, S_0) \\ \land \neg Have(Drill, S_0)$$

- Goal state: a logical query asking for suitable situations

$$\exists s [At(Home, s) \land Have(Milk, s) \\ \land Have(Bananas, s) \\ \land Have(Drill, s)]$$

- Operators: triples of $\langle \textsc{Action}, \textsc{Precondition}, \textsc{Effects} \rangle$

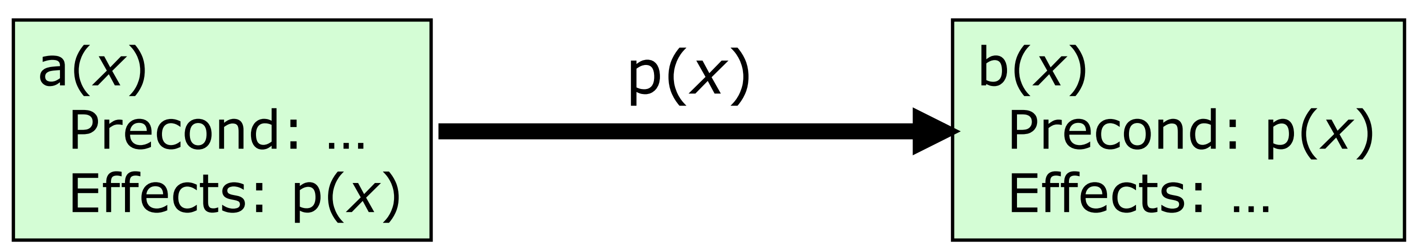


$$Op(\textsc{Action}{:}Go(there), \\ \textsc{Precondition}{:}At(here) \land Path(here, there) \\ \textsc{Effects}{:}At(there) \land \neg At(here) \quad)$$

Flaw 1: open goal

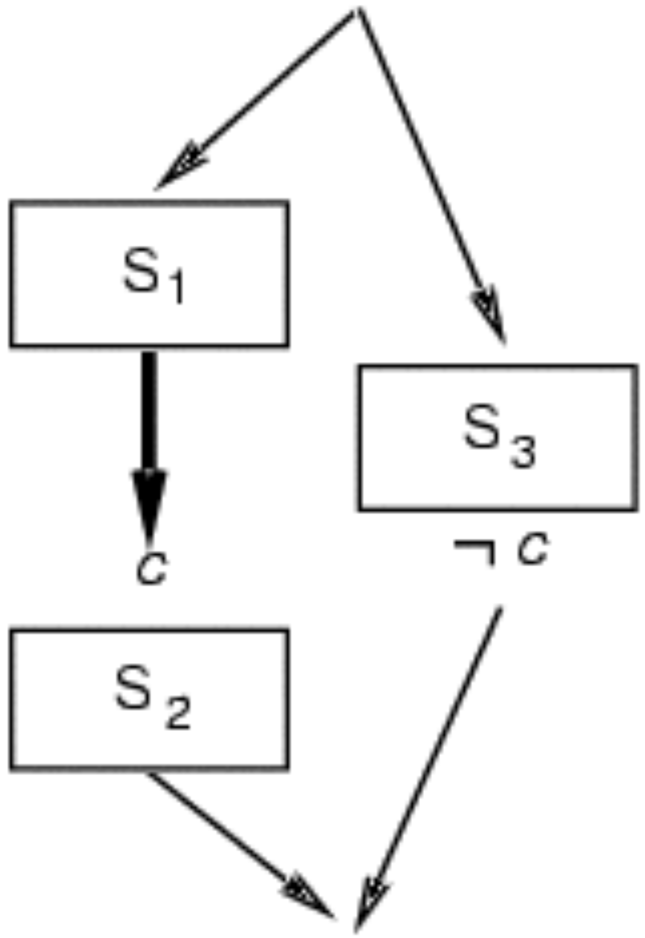
- Flaw:
  - A plan step b has a precondition p that we haven't decided how to establish

- Resolving the flaw:
  - Find a step a …
    (either one already in the plan, or a newly inserted one)
  - … that can be used to establish p (can precede b and produce p)
  - Instantiate variables
  - Create a causal link



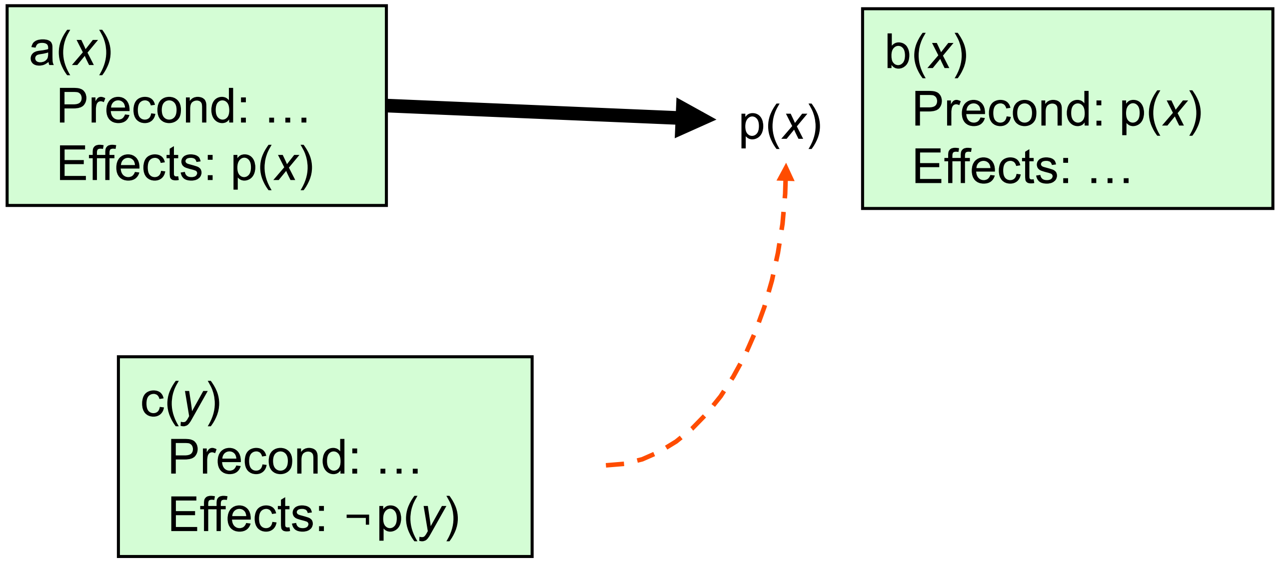
Causal link protection

- Consider the following situation
- Plan step S_3 threatens the execution of S_2 by potentially destroying the precondition c needed by S_2
- Step S_3 is called
  - a threat or
  - a clobberer



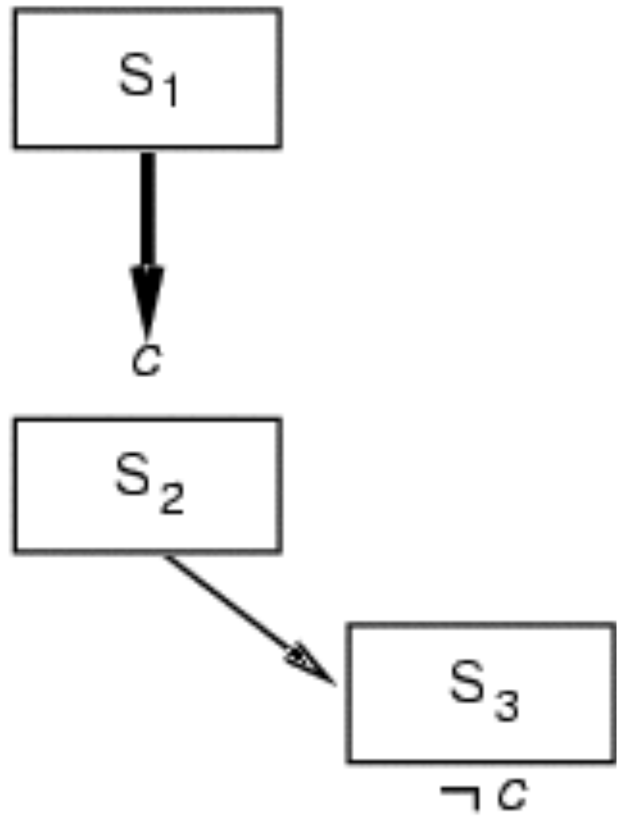
Flaw 2: threat

- Flaw: a precondition/effect interaction
  - A step a establishes an effect p(x) needed as precondition for a step b
  - Another plan step c, possibly executable between steps a and b, is capable of falsifying condition p(x)
- Resolving the flaw:
  - Impose a constraint to prevent c from being able to falsify p(x)
- Several possibilities:
  - Promotion (c after b)
  - Demotion (c before a)
  - Separation
  - White knight

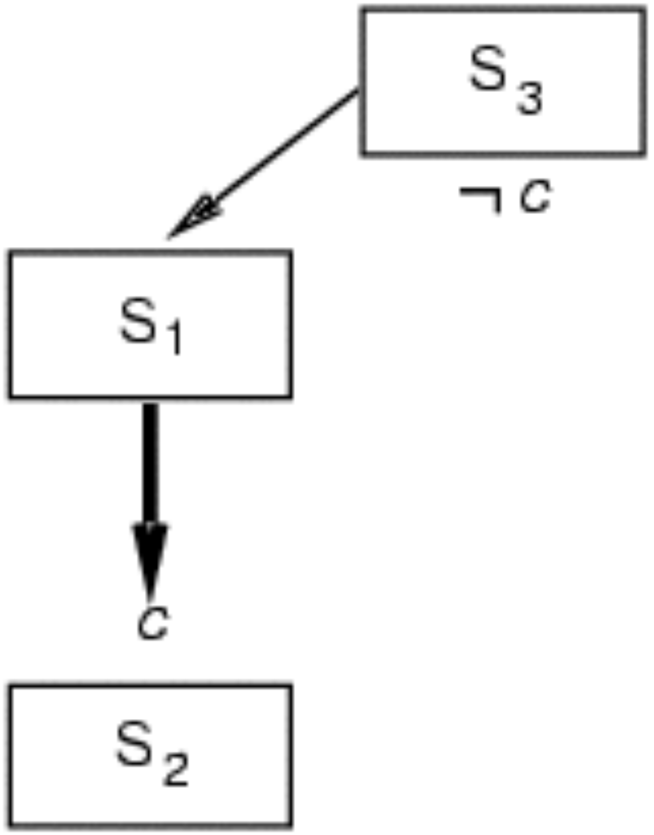


Causal link protection: promotion and demotion

- Promotion
  - Add ordering link to promote action S_3 after action S_2



- Demotion
  - Add ordering link to demote action S_3 before action S_1



Causal link protection

- Separation
  - Constrain the variables involved to prevent c from falsifying p(x)
  - For example, by an inequality constraint

- Sometimes,
  neither promotion nor demotion nor separation lead to a solution!
  - In this case, introducing an additional plan step (ensured to happen between c and b) could help. Such a plan step is called a white knight.
  - However, the planning algorithms do NOT explicitly use this for resolving threats.
  - Failure to resolve a threat will eventually lead, via backtracking, to undoing unfavorable choices -- here, of a causal link -- and generating an alternative solution

What do we know so far?

- Search space: plan space
- Node: a set of partially-instantiated plan steps + sets of constraints
- Backward search (start at the goal)
- First node: the two dummy steps start, and finish,
  - Start has effects = initial state
  - Finish has preconditions = goal statement
- Goal set (not a stack as in state space planning)
- Plan contains steps, bindings, ordering constraints, and causal links
- Flaws are resolved on each iteration until none remain:
  - Open goal
    - Solved by creating a causal link from an existing or newly added step (i.e. by finding an operator that establishes the goal)
  - Threat/clobberer
    - Solved by promoting or demoting the clobberer or adding a binding constraint

The PSP procedure

```
PSP(π)
    flaws ← OpenGoals(π) ∪ Threats(π)
    if flaws = ∅ then return(π)
    select any flaw φ ∈ flaws
    resolvers ← Resolve(φ, π)
    if resolvers = ∅ then return(failure)
    nondeterministically choose a resolver ρ ∈ resolvers
    π′ ← Refine(ρ, π)
    return(PSP(π′))
end
```

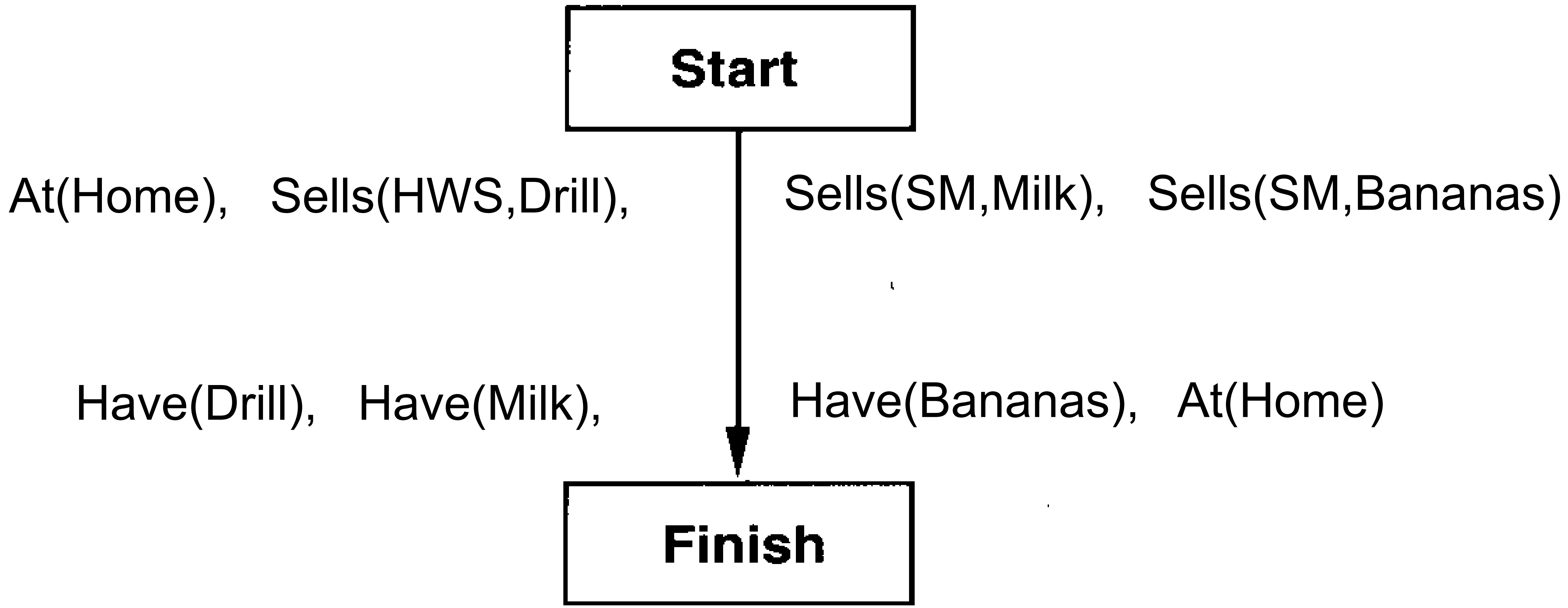
- PSP is both sound and complete

Example

- Similar (but not identical) to an example in Russell and Norvig's Artificial Intelligence: A Modern Approach
- Operators:
  - **Start**
    - Precond: none
    - Effects: At(Home), sells(HWS,Drill), Sells(SM,Milk), Sells(SM,Banana)
  - **Finish**
    - Precond: Have(Drill), Have(Milk), Have(Banana), At(Home)
  - **Go(l,m)**
    - Precond: At(l)
    - Effects: At(m), ¬At(l)
  - **Buy(p,s)**
    - Precond: At(s), Sells(s,p)
    - Effects: Have(p)

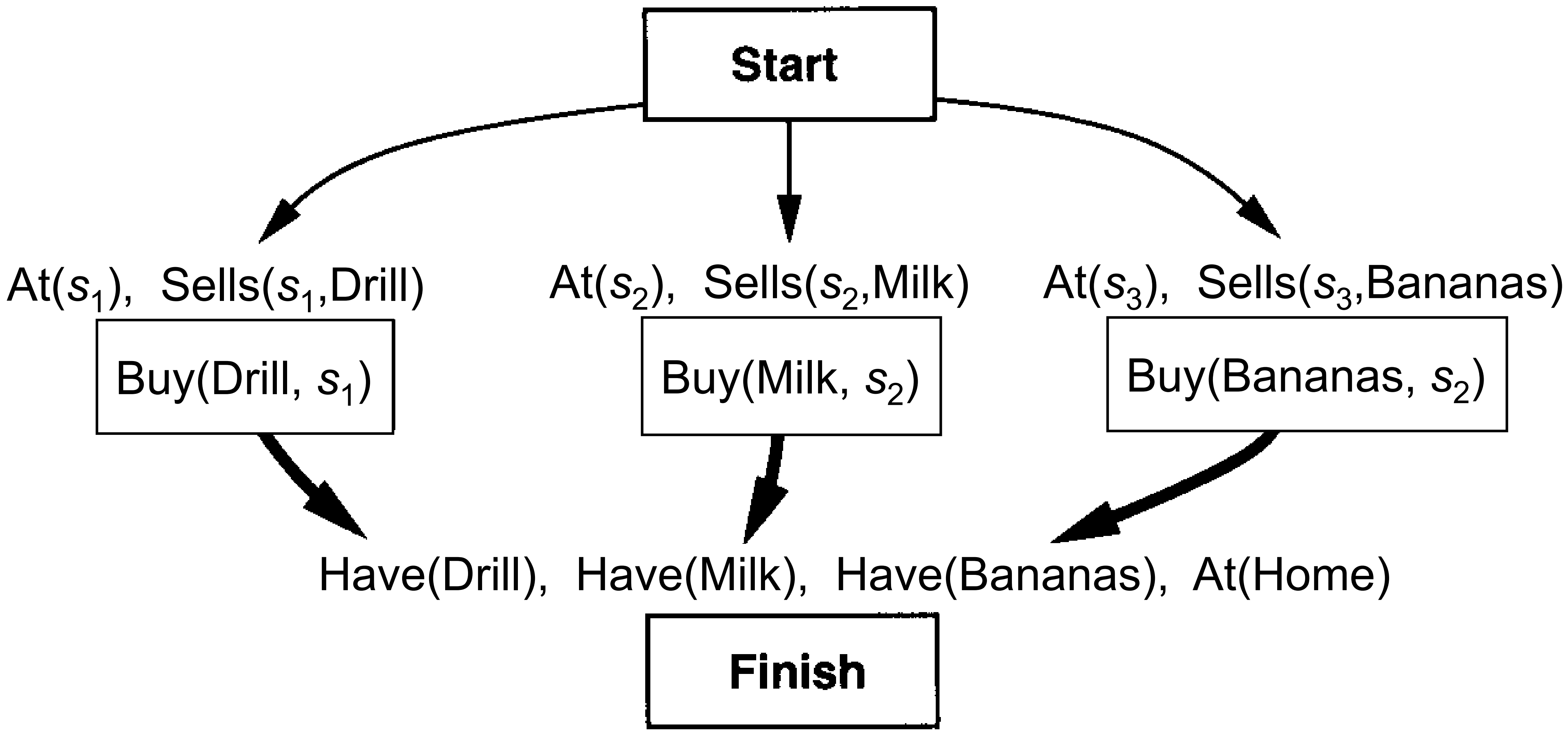
Example (continued)

- Initial plan



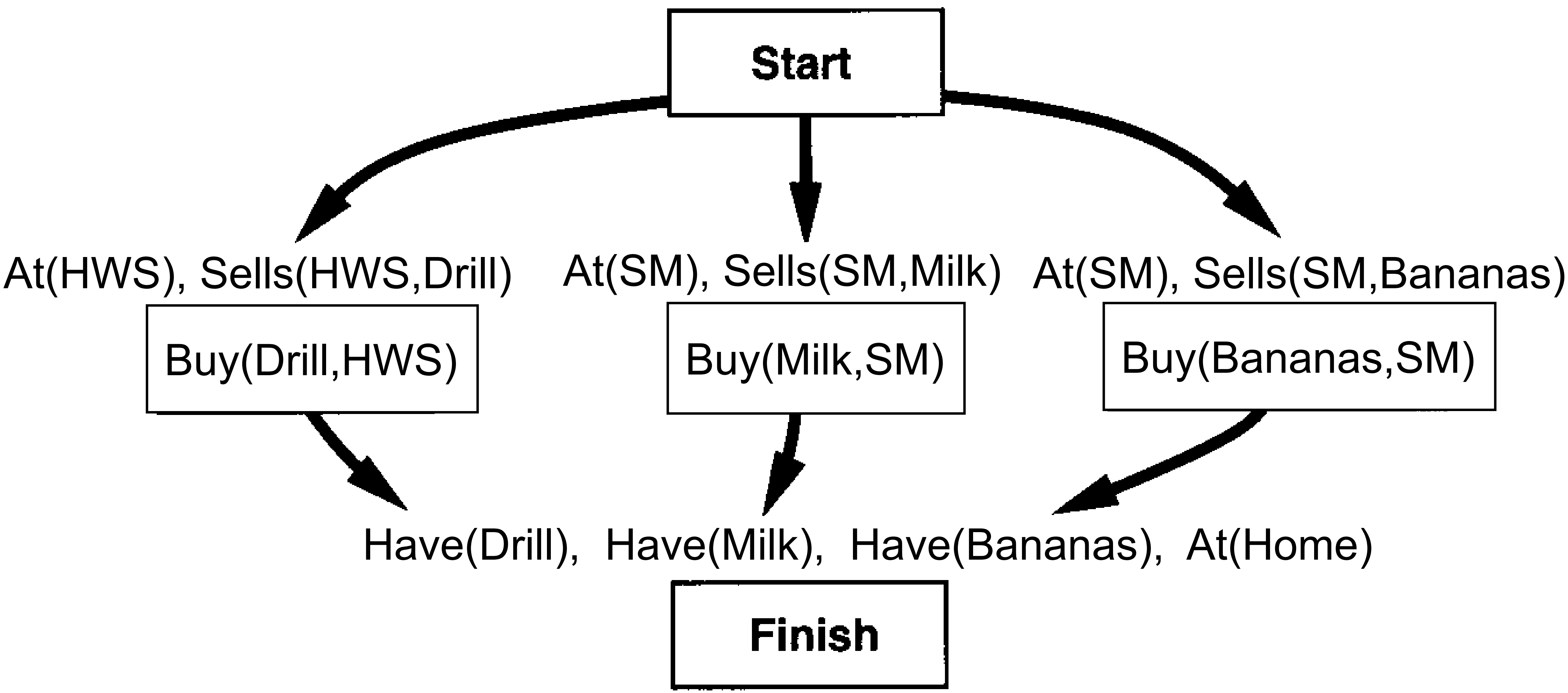
Example (continued)

- The only possible ways to establish the "Have" preconditions



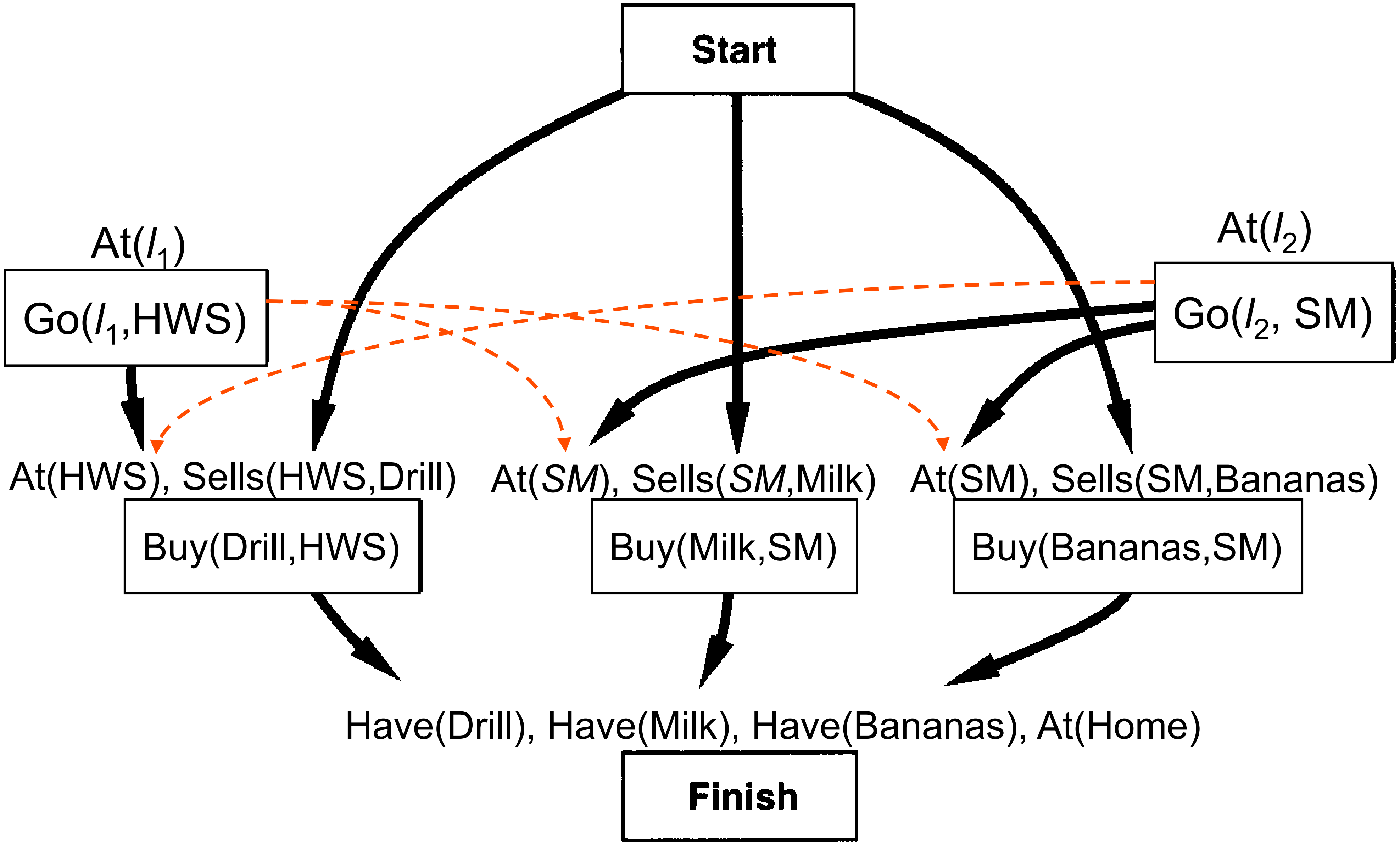
Example (continued)

- The only possible way to establish the "Sells" preconditions



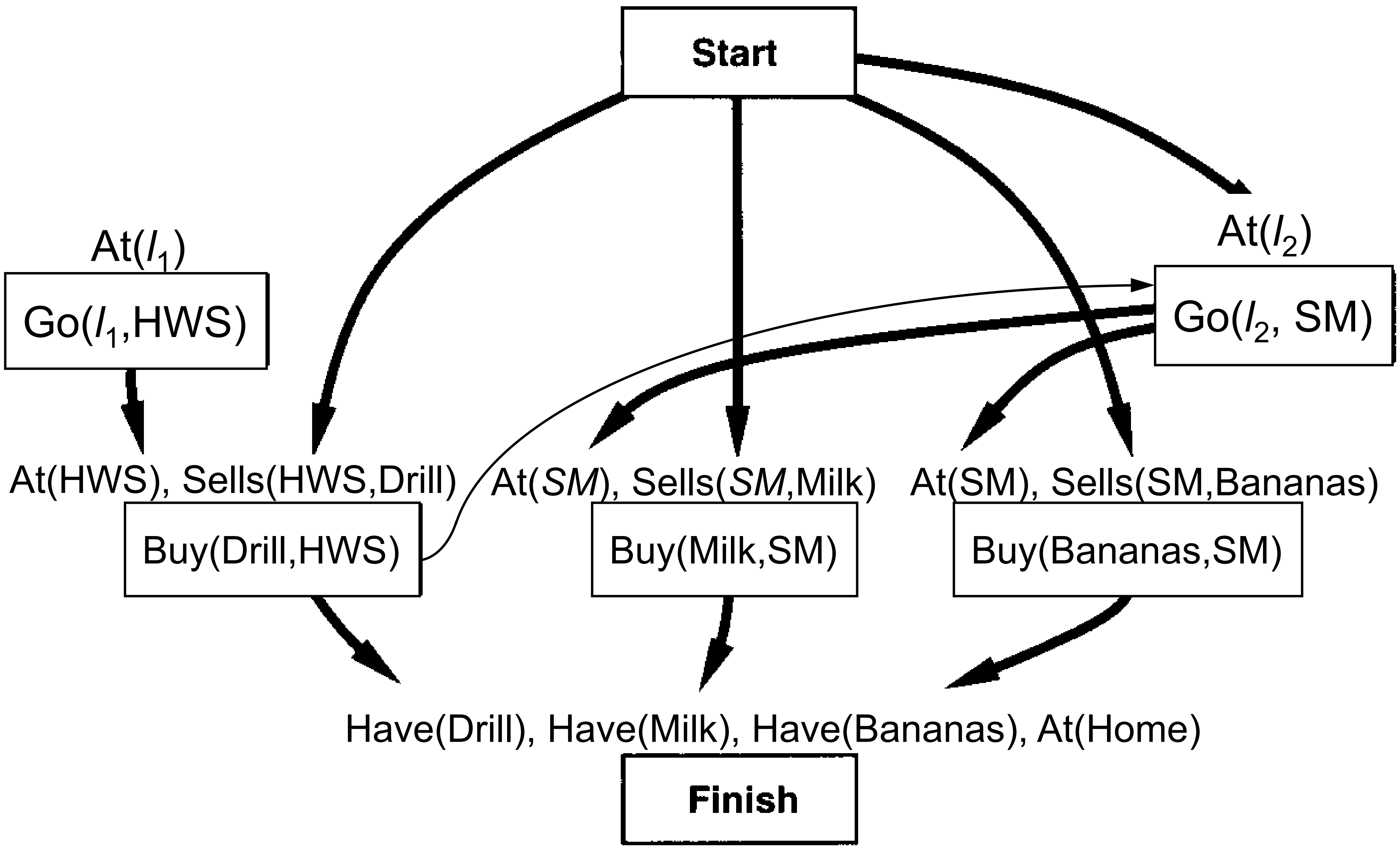
Example (continued)

- The only ways to establish At(HWS) and At(SM)
- Note the threats



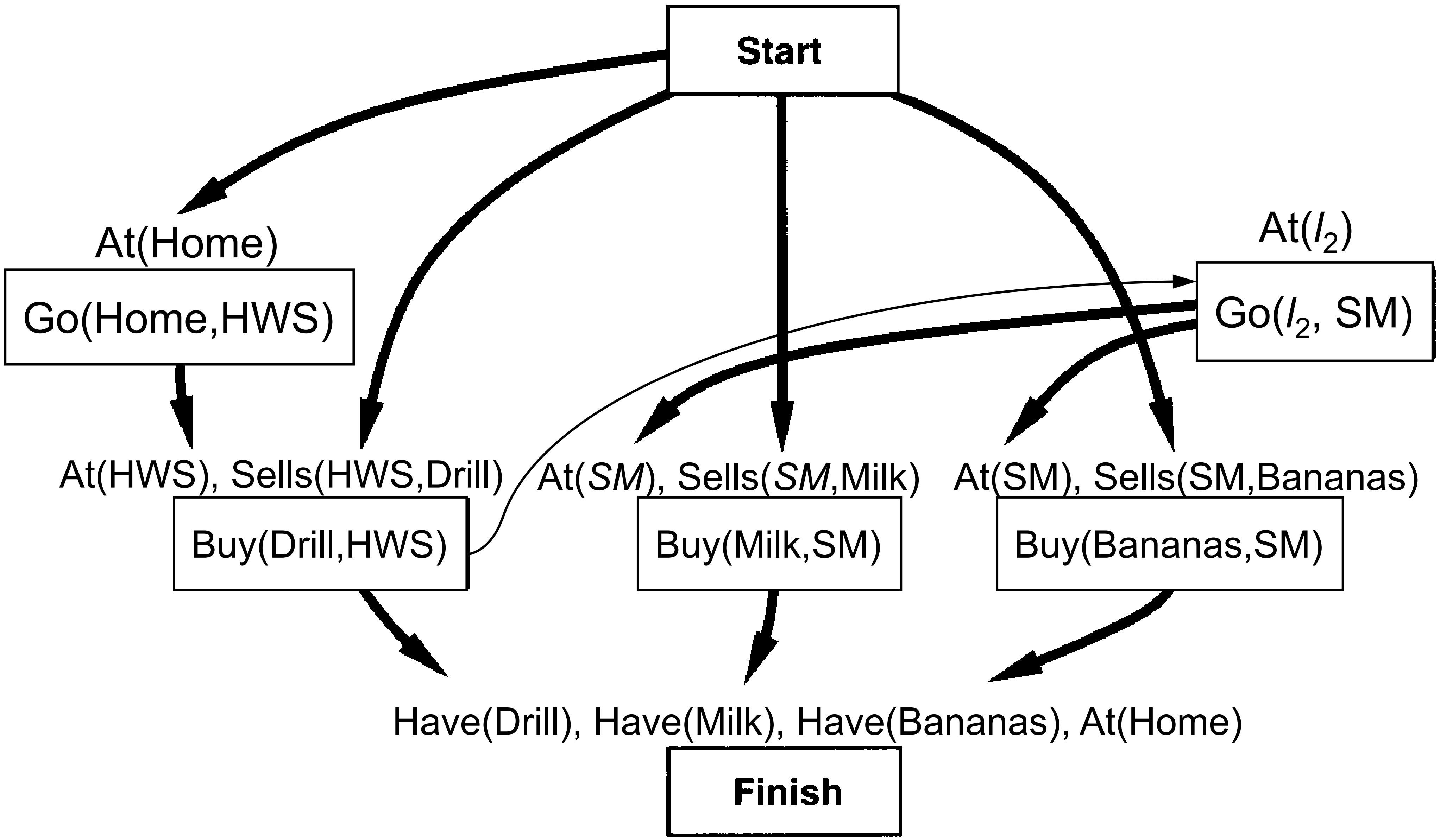
Example (continued)

- To resolve the third threat, make Buy(Drill) precede Go(SM)
  - This resolves all three threats



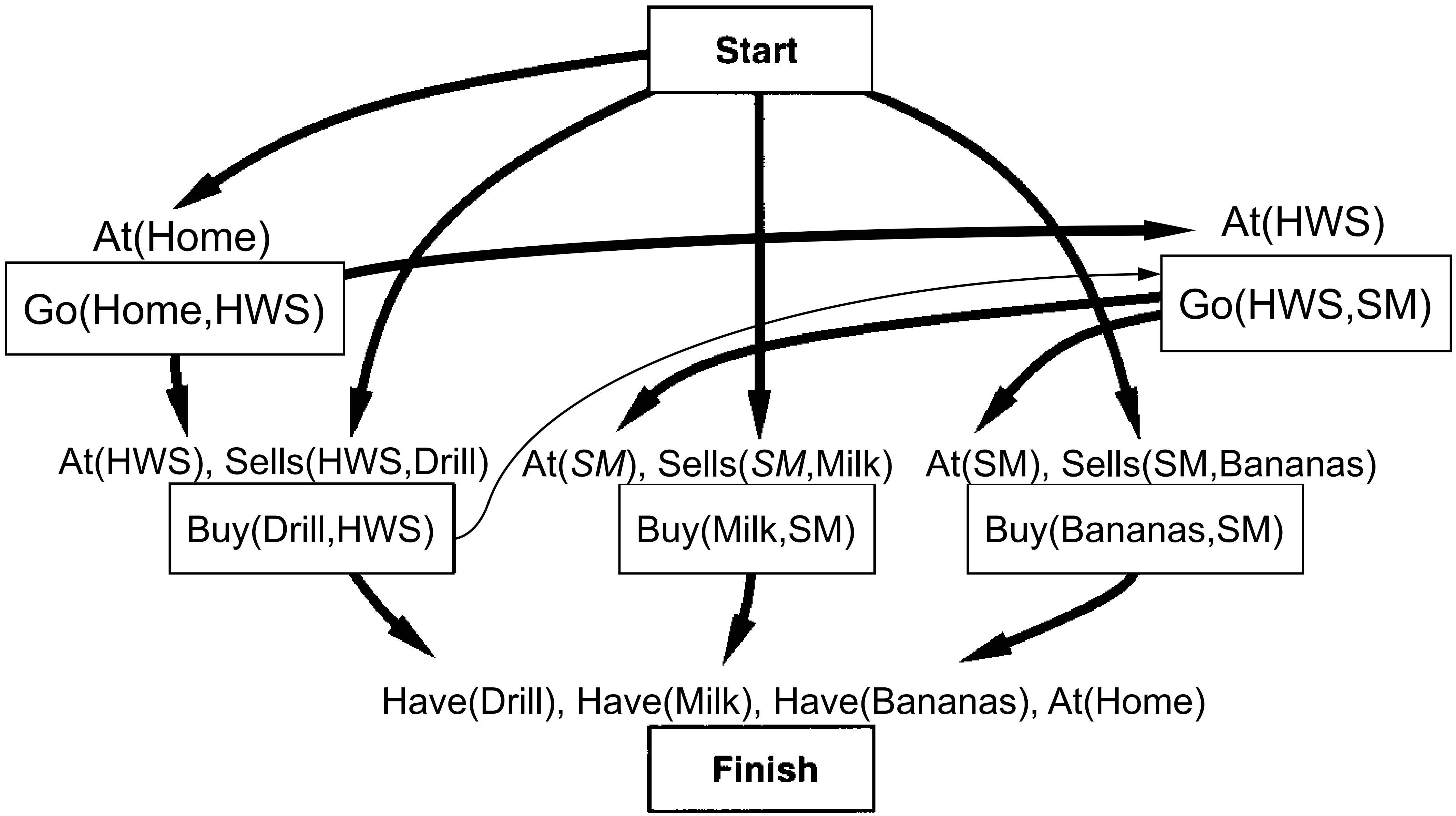
Example (continued)

- Establish $At(l_1)$ with l_1=Home



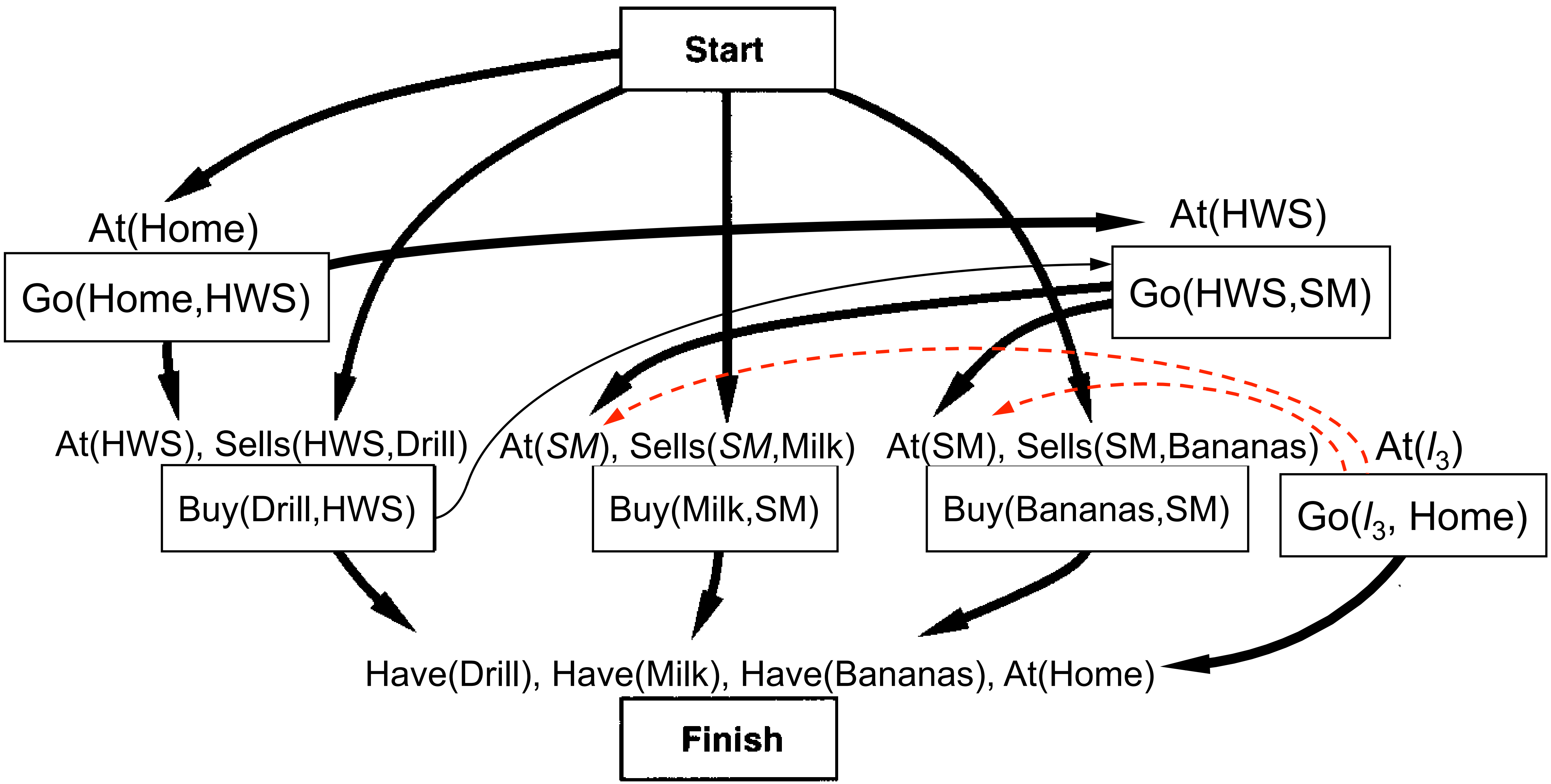
Example (continued)

- Establish At(l_2) with l_2=HWS



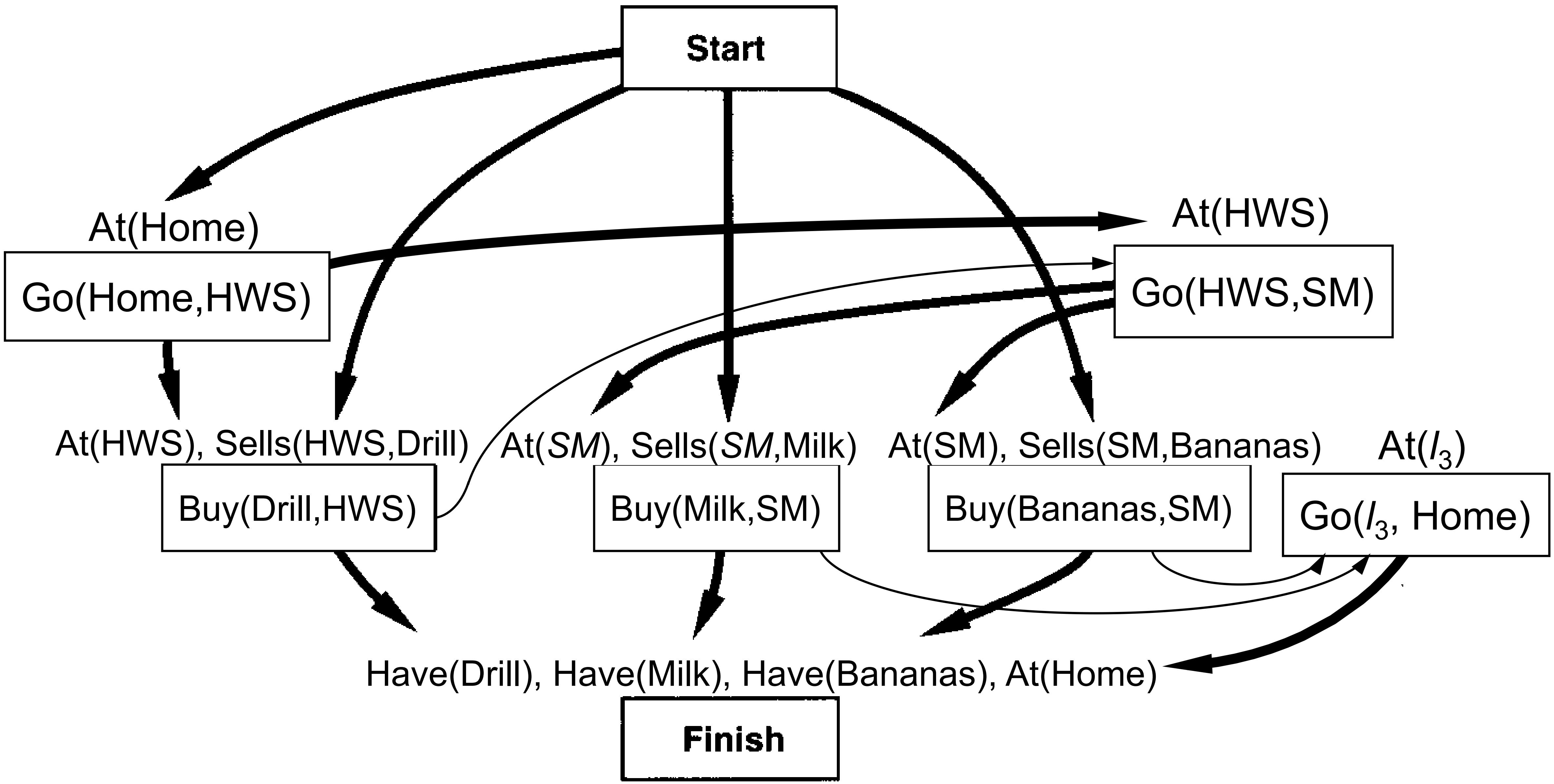
Example (continued)

- Establish At(Home) for Finish



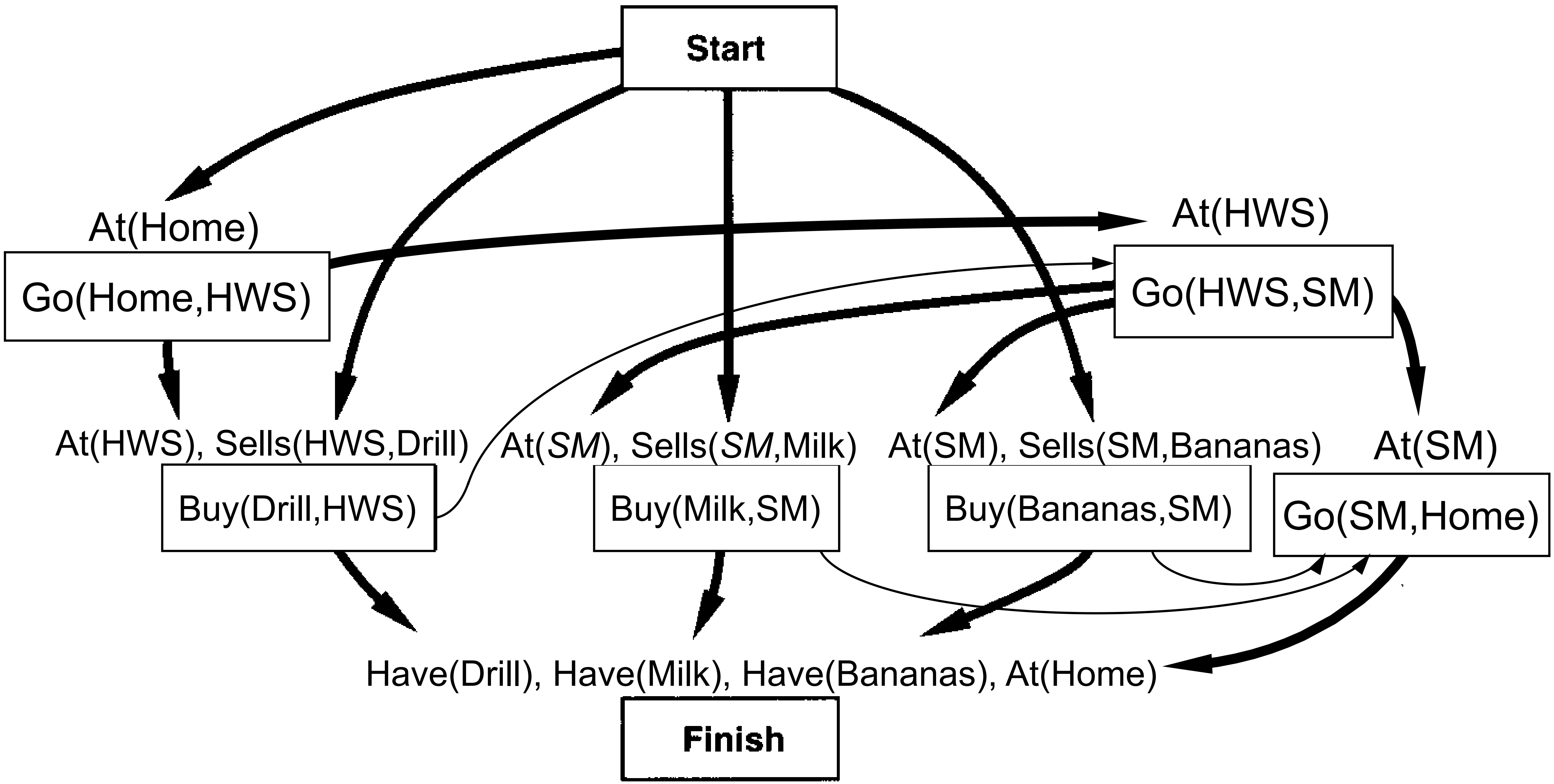
Example (continued)

- Constrain Go(Home) to remove threats to At(SM)



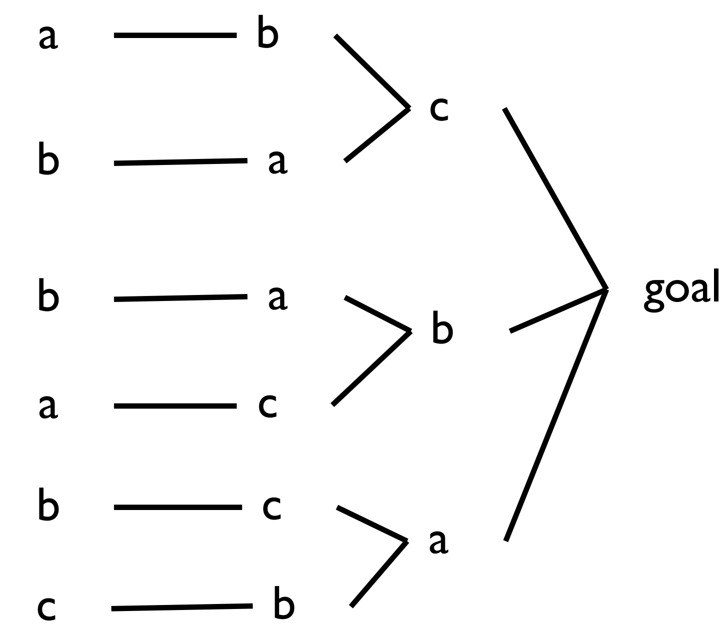
Final Plan

- Establish $At(l_3)$ with l_3=SM



Comments

- PSP doesn't commit to orderings and instantiations until necessary
  - Avoids generating search trees like this one:



- Problem: how to prune infinitely long paths?
  - Loop detection is based on recognizing states we've seen before
  - In a partially ordered plan, we don't know the states
- Can we prune if we see the same action more than once?

… — go(b,a) — go(a,b) – go(b,a) — at(a)

- No!
  Sometimes we might need the same action several times in different states of the world (see next slide)

Example

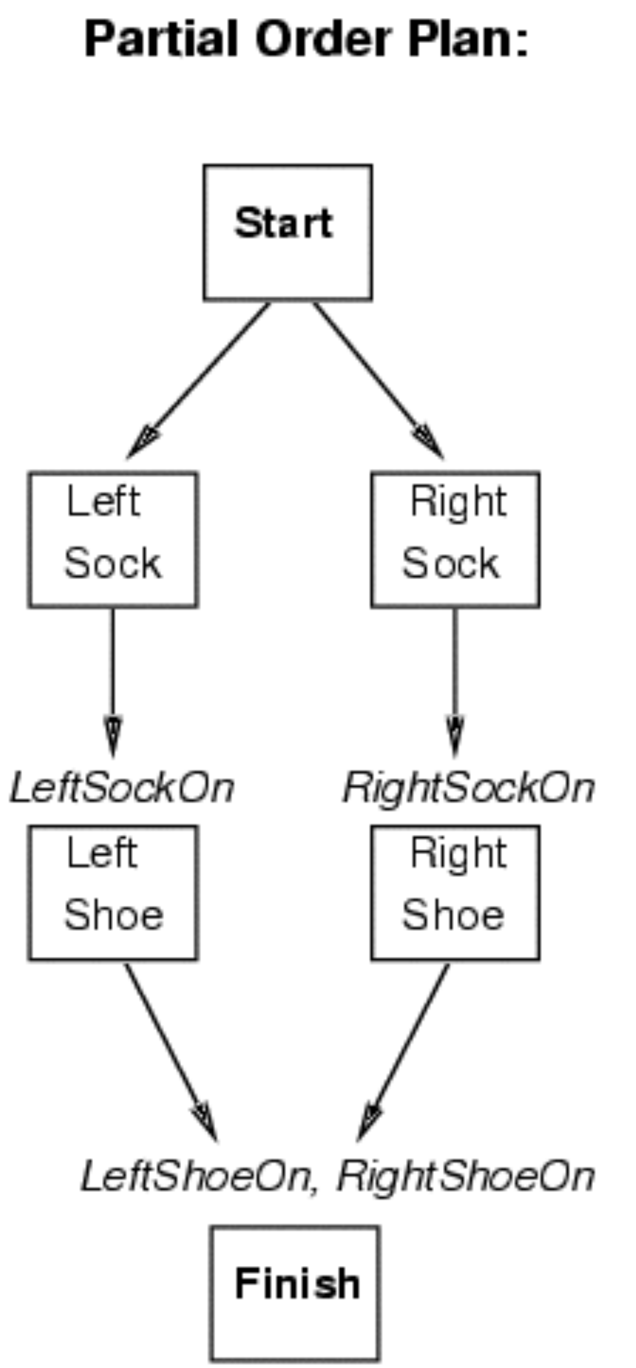
- 3-digit binary counter starts at 000, want to get to 110
  - $s_0 = \{d_3=0, d_2=0, d_1=0\}$
  - $g = \{d_3=1, d_2=1, d_1=0\}$
- Operators to increment the counter by 1:
  - incr0
    - Precond: $d_1=0$
    - Effects: $d_1=1$
  - incr01
    - Precond: $d_2=0, d_1=1$
    - Effects: $d_2=1, d_1=0$
  - incr011
    - Precond: $d_3=0, d_2=1, d_1=1$
    - Effects: $d_3=1, d_2=0, d_1=0$

A weak pruning technique

- We can prune all paths of length > n,
where n = ||{all possible states}||
    - This doesn't help very much, though

- It is not clear
whether there's a good pruning technique for plan-space planning

Partial order vs total order plans: The shoe example



POP algorithm: The basic idea

1) Terminate if the goal set is empty
2) Select a goal g from the goal set & identify the plan step that needs it, S_{need}.
3) Let S_{add} be a step (operator) that adds g,
   either a new step or a step that is already in the plan.
   Add the causal link $S_{add} \rightarrow_g S_{need}$,
   constrain S_{add} to come before S_{need},
   and enforce bindings that make S_{add} add g.
4) Update the goal set with all the preconditions of the step S_{add}, and delete g.
5) Identify threats and resolve the conflicts by adding ordering or bindings constraints.
   - A step S_k threatens a causal link $S_i \rightarrow_g S_j$
     when it occurs between S_i and S_j, and it adds or deletes p.
   - Resolve threats
     by using promotion, demotion, or separation.

The POP planning algorithm

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*)
end

POP planning algorithm: `Select-Subgoal`

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 planning algorithm: `Choose-Operator`

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*)

POP planning algorithm: `Resolve-Threats`

procedure RESOLVE-THREATS($plan$)

    for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS($plan$) **do**
        choose either
            Promotion: Add $S_{threat} \prec S_i$ to ORDERINGS($plan$)
            Demotion: Add $S_j \prec S_{threat}$ to ORDERINGS($plan$)
        if not CONSISTENT($plan$) **then fail**
    end

POP with partially-instantiated operators: `Choose-Operator`

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

  choose a step S_{add} from *operators* or STEPS(*plan*) that has c_{add} as an effect
    such that u = UNIFY(c, c_{add}, BINDINGS(*plan*))
  if there is no such step
    then fail
  add u to BINDINGS(*plan*)
  add $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
  add $S_{add} \prec S_{need}$ 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*)

POP with partially-instantiated operators: `Resolve-Threats`

```
procedure RESOLVE-THREATS(plan)

  for each S_i --c--> S_j in LINKS(plan) do
      for each S_threat in STEPS(plan) do
          for each c' in EFFECT(S_threat) do
              if SUBST(BINDINGS(plan), c) = SUBST(BINDINGS(plan), ¬ c') then
                  choose either
                      Promotion: Add S_threat ≺ S_i to ORDERINGS(plan)
                      Demotion: Add S_j ≺ S_threat to ORDERINGS(plan)
              if not CONSISTENT(plan)
                  then fail
          end
      end
  end
```