Planning vs. Search-based Problem Solving

Problem solving: actions generate successor states.

Planning: actions are represented as preconditions and effects.

Problem solving: state representations are complete.

Planning: complete state descriptions would be enormous.

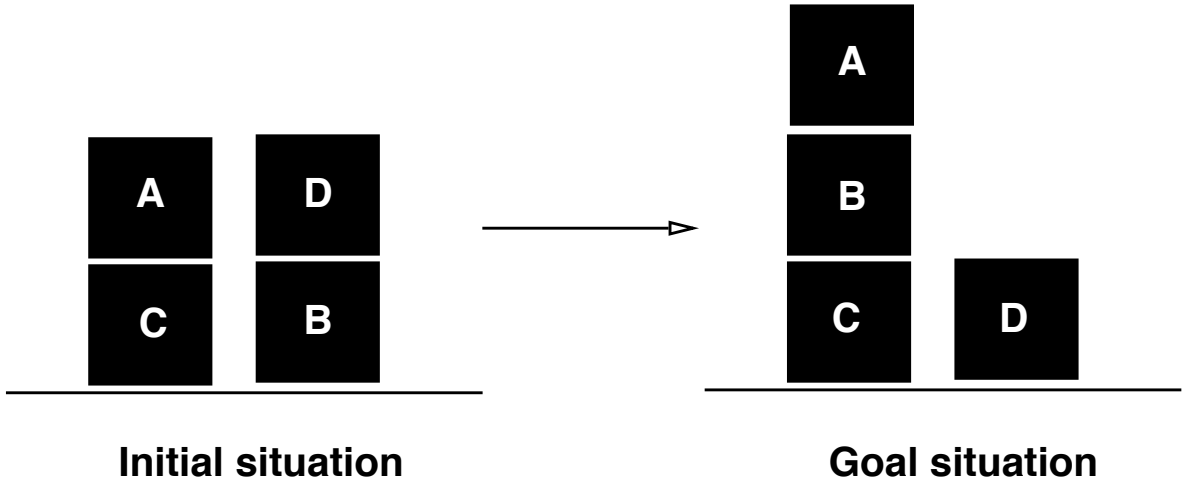
Problem solving: goal test and heuristic function used to evaluate states.

Planning: goals are usually represented as a conjunct of state variables.

Problem solving: incrementally generates solution as a sequence of actions.

Planning: can add actions to any part of the plan at any time.

Planning Example



Initial Situation:

on(A,C), on(C,Table), on(D,B), on(B,Table), clear(A), clear(D)

Goal Situation:

on(A,B), on(B,C)

Searching Plan Space

Alternative is to search through the space of *plans* rather than the space of *situations*.

Start with simple, incomplete **partial plan**; expand until complete.

Operators: add a step, impose an ordering on existing steps, instantiate a previously unbound variable.

Solution: final plan.

Plan Representation

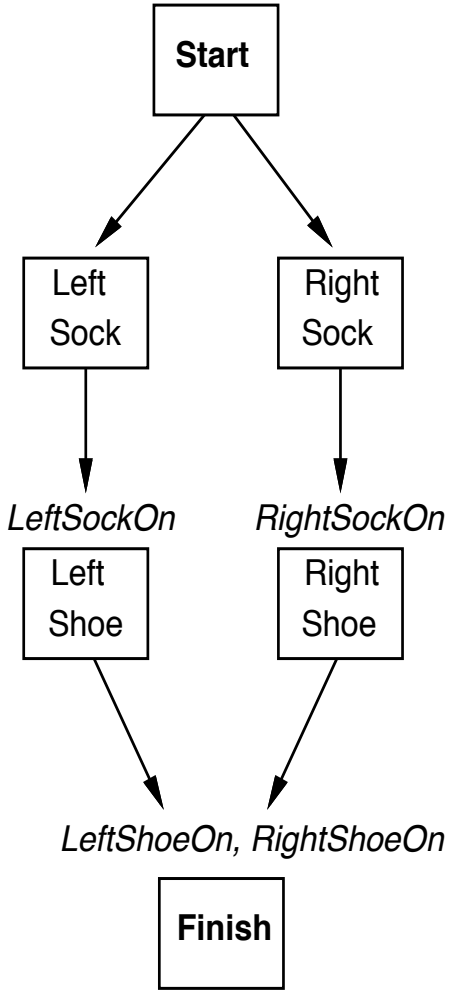
Most planners use the principle of **least commitment.**

Partial order planner: represents plans in which some steps are ordered and other steps are unordered.

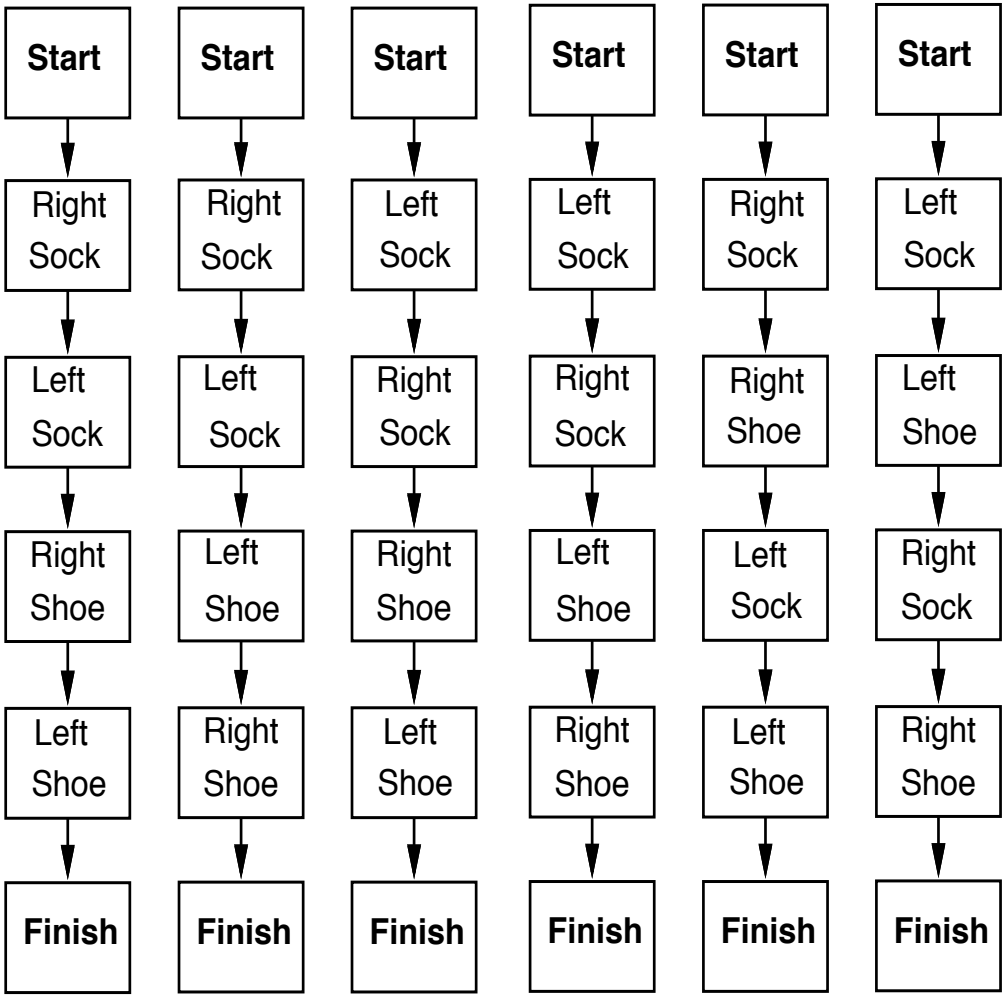
Total order planner: simple list of steps.

A totally ordered plan that is derived from a plan P by adding ordering constraints is called a **linearization** of P.

Partial Order Plan:



Total Order Plans:



Components of a Plan

1. A set of plan steps. Each step is one of the operators for the problem.
2. A set of step ordering constraints, $S_i \prec S_j$.
3. A set of variable binding constraints, $v = x$ where v is a variable and x is a constant or another variable.
4. A set of **causal links**, $S_i \xrightarrow{c} S_j$ (S_i achieves precondition c for S_j.)

What is a solution?

You might expect that only fully instantiated, totally ordered plans would be considered solutions. But this is not a good idea for several reasons:

- It makes more sense to have a planner return a partial order plan than to arbitrarily choose one linearization of it.
- Sometimes actions can be performed in parallel, so it is best to generate solutions that allow for actions to happen in parallel.
- A plan may be integrated with another plan later. Keeping the plan flexible can help with plan integration.

We consider a plan to be a solution if it is **complete** and **consistent**.

A complete plan is one in which every precondition is achieved by another step. A precondition is achieved if it is an effect of a step and no other step can cancel it out. Formally:

S_i *achieves* precondition c of S_j if
(1) $S_i \prec S_j$ and $c \in EFFECTS(S_i)$, and
(2) there is no step S_k such that $(\neg c) \in EFFECTS(S_k)$ where $S_i \prec S_k \prec S_j$ in some linearization of the plan.

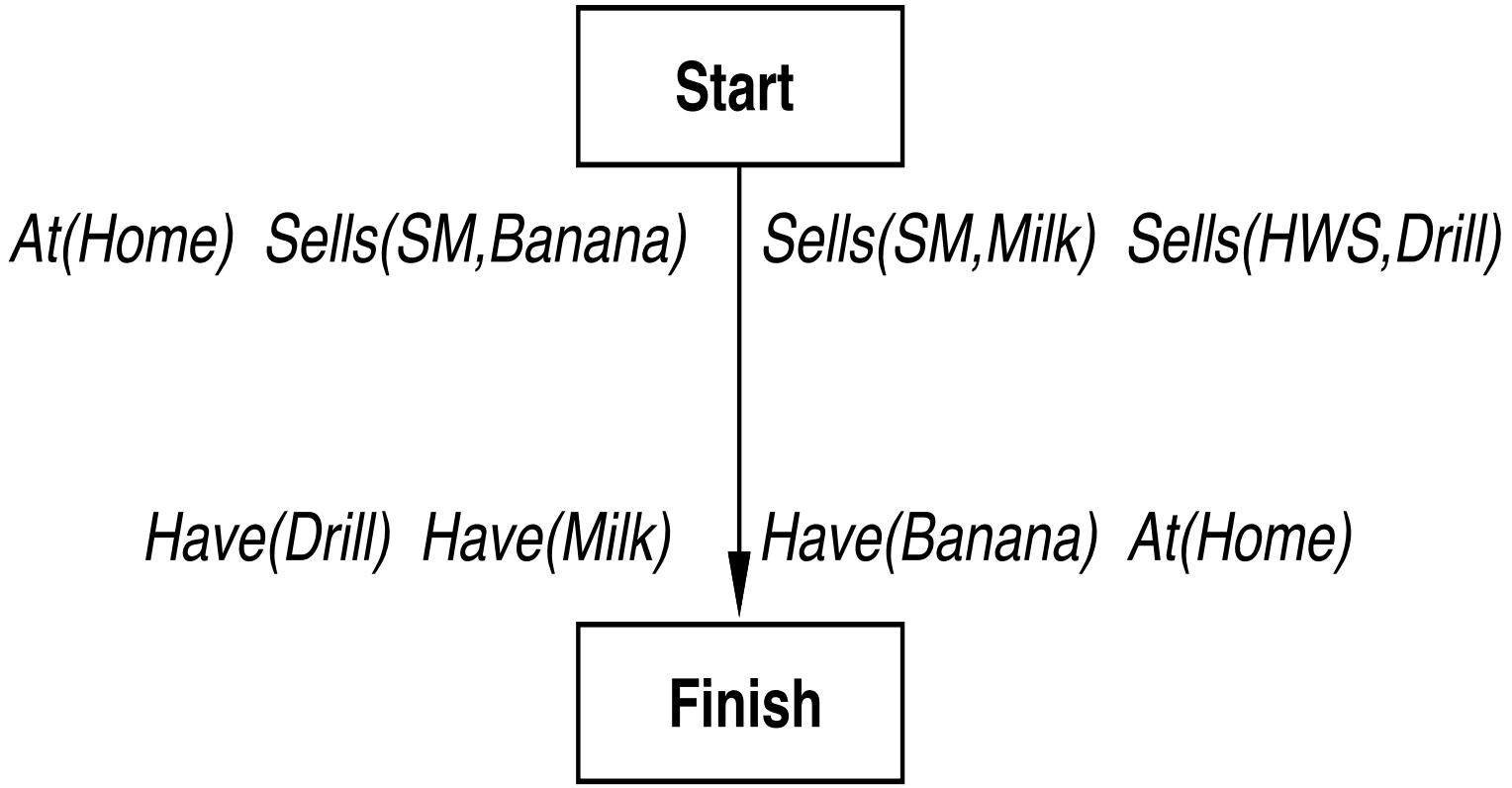
A consistent plan is a plan with no contradictions in the ordering or binding of constraints.

A contradiction occurs when $S_i \prec S_j$ and $S_j \prec S_i$ or if $v = A$ and $v = B$ for different constants A and B.

Under this definition, the partial plan for putting on socks is a solution!

Partial-Order Regression Planner Example

Suppose you want a plan to buy milk, bananas, and a drill.
The initial plan might be:

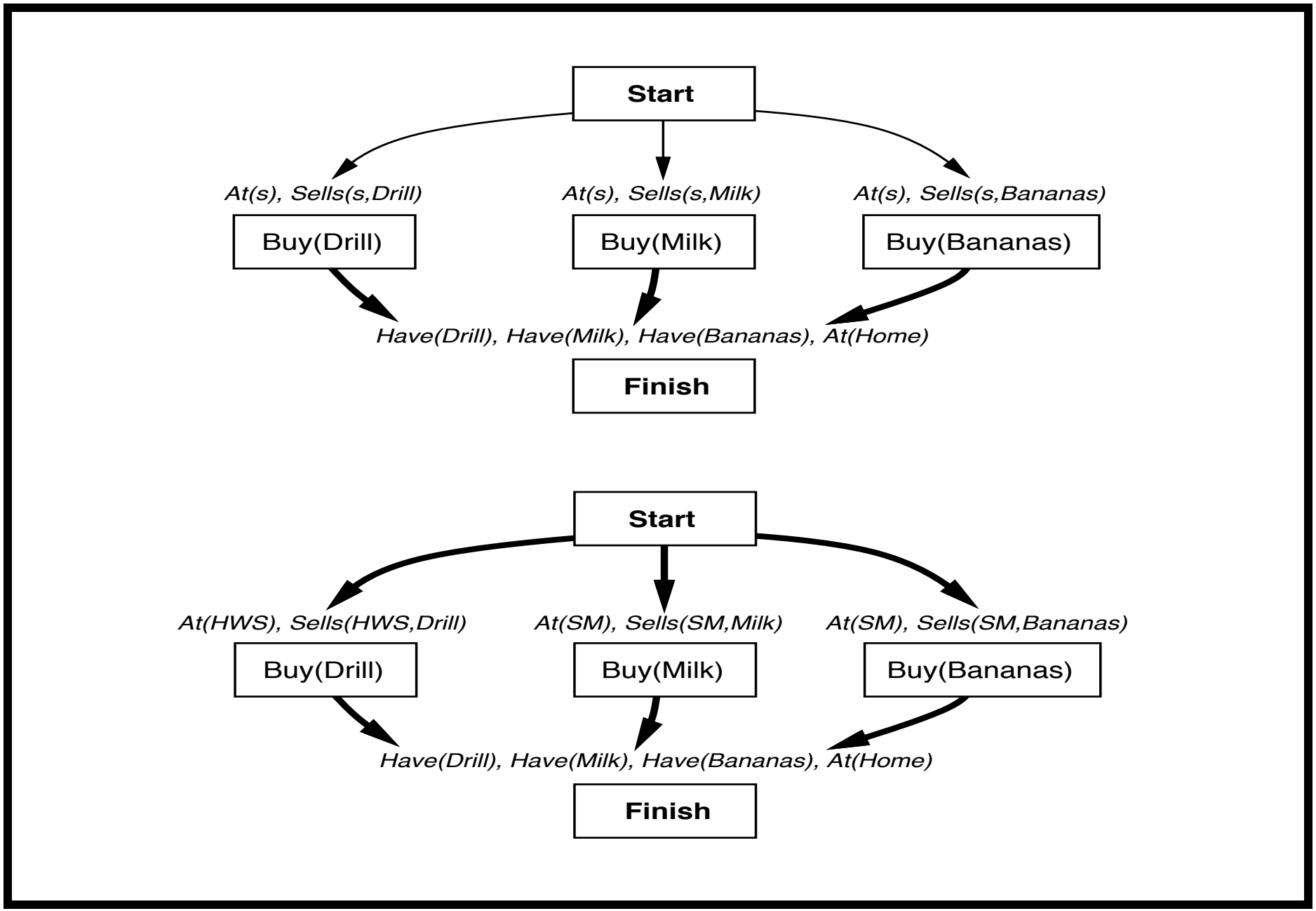


Operators

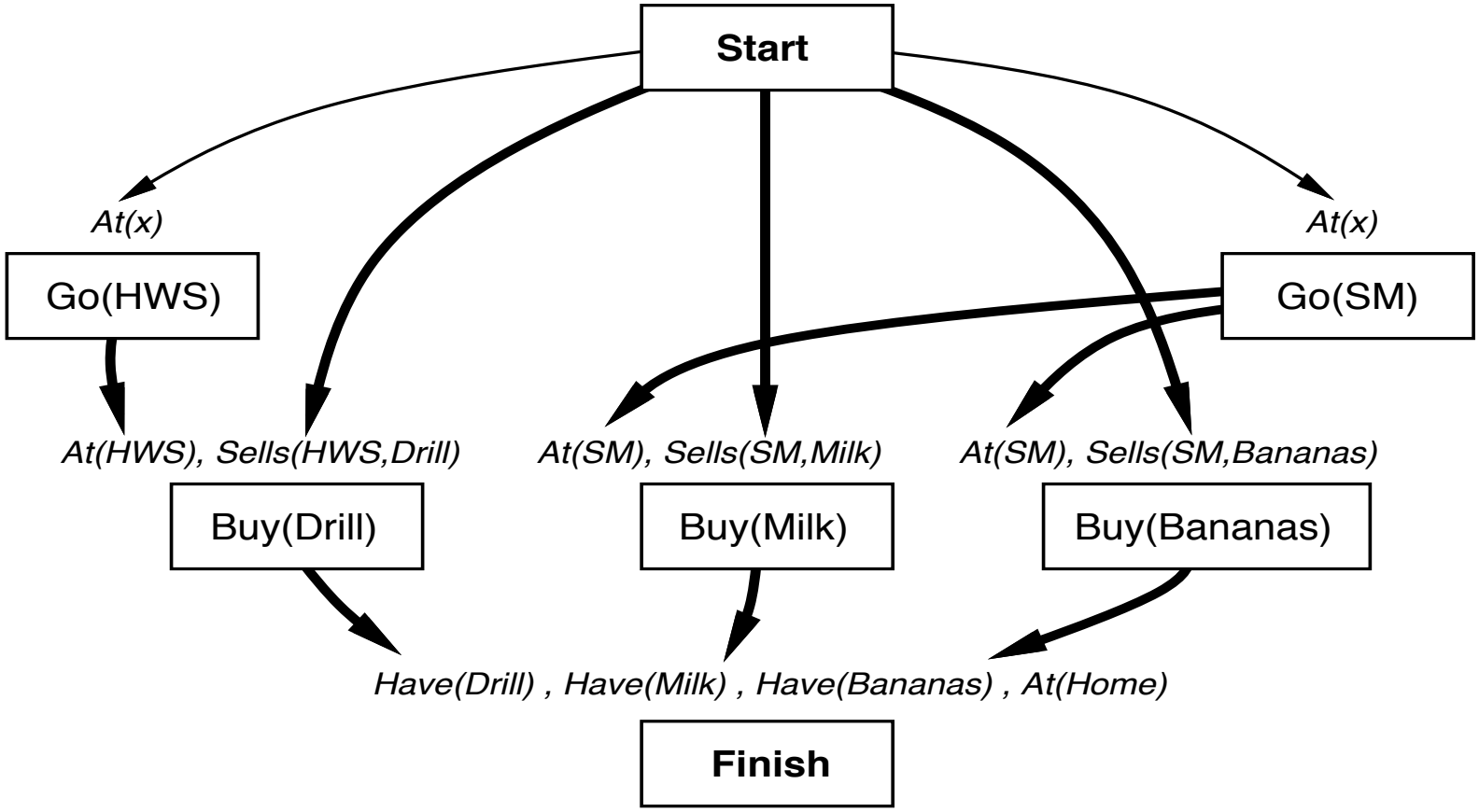
Operator:
ACTION = Go(there)
PRECONDITIONS = At(here)
EFFECTS = At(there) $\wedge$ $\neg$ At(here)

Operator:
ACTION = Buy(obj)
PRECONDITIONS = At(store) $\wedge$ Sells(store,obj)
EFFECT = have(obj)

To keep the search focused, the planner only considers adding steps that achieve a precondition that has not yet been achieved.

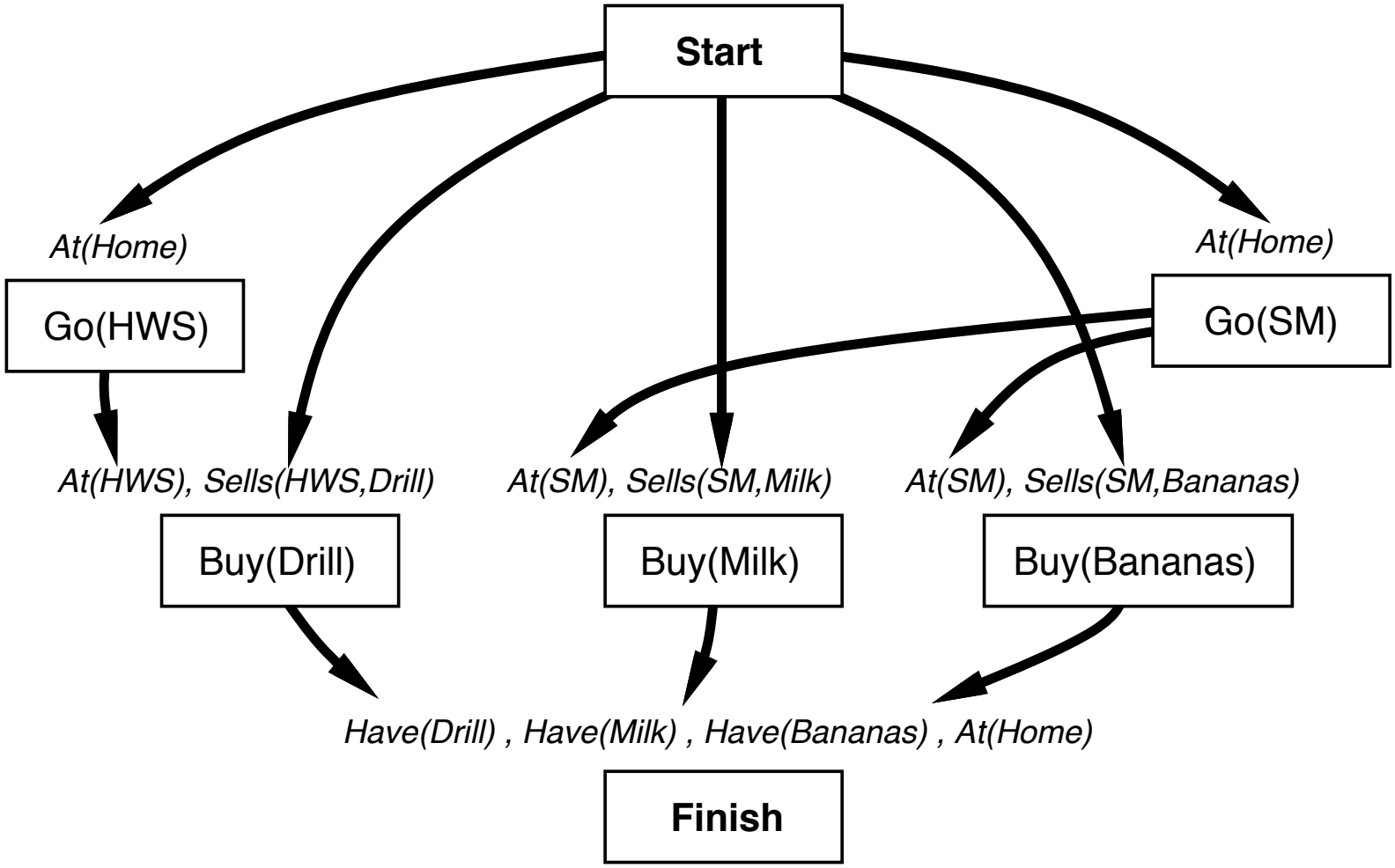

Start
At(s), Sells(s,Drill)
At(s), Sells(s,Milk)
At(s), Sells(s,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill), Have(Milk), Have(Bananas), At(Home)
Finish
Start
At(HWS), Sells(HWS,Drill)
At(SM), Sells(SM,Milk)
At(SM), Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill), Have(Milk), Have(Bananas), At(Home)
Finish

The plan is extended by choosing *Go* actions to achieve the *At* preconditions.



Note that the *Go* actions have unachieved preconditions that interact!

Look what happens if the planner tries to achieve the preconditions of *Go* with the *At(home)* condition in the initial state!

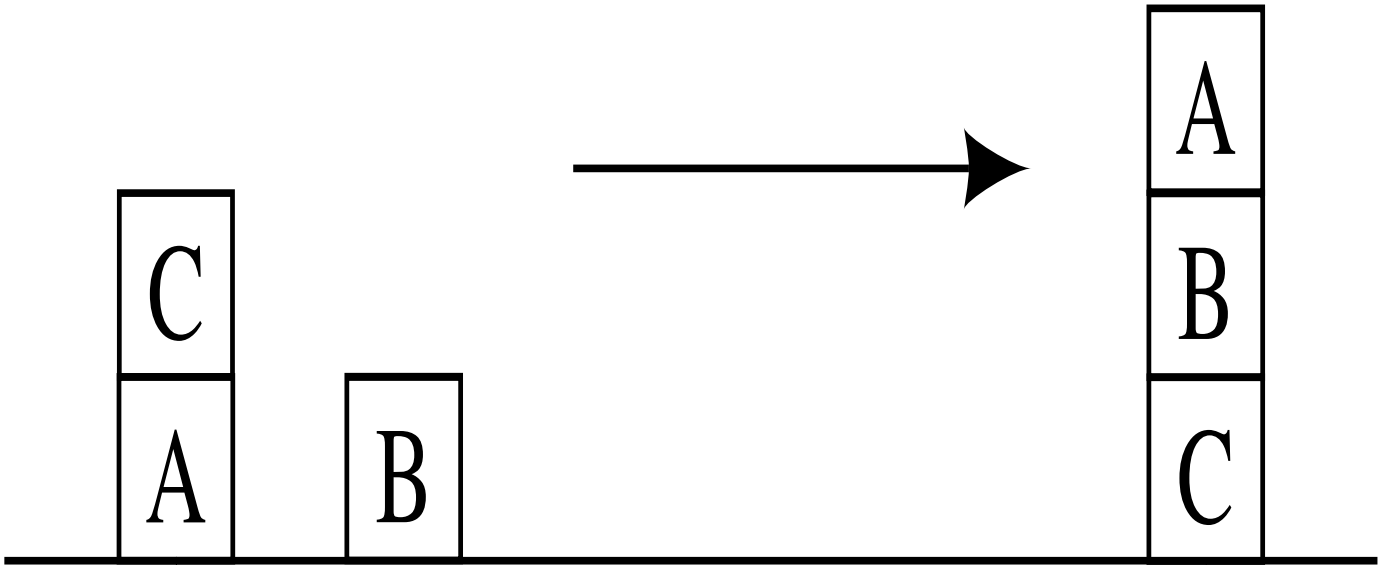


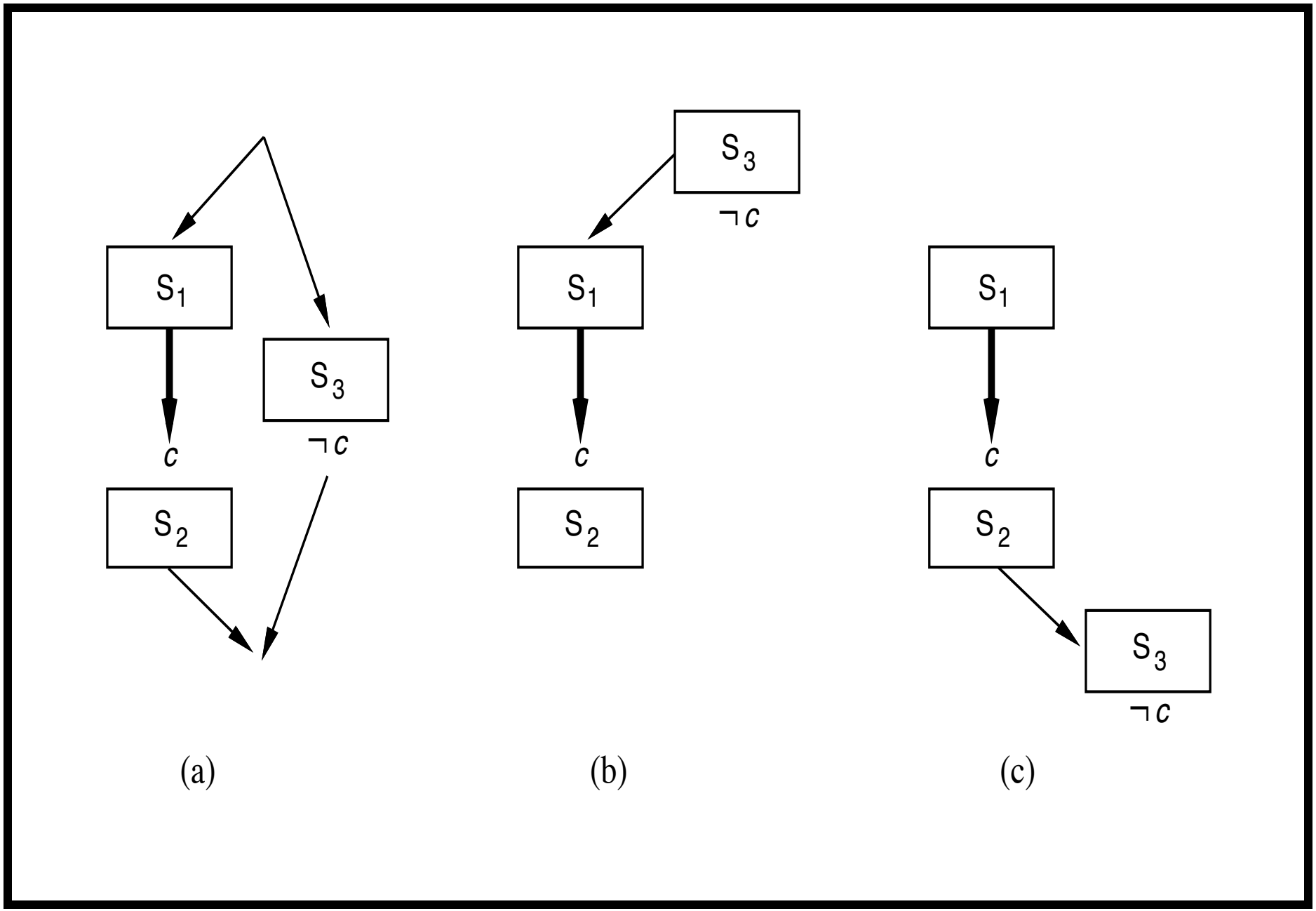
A planner can notice dead ends

- This partial plan is a dead end because one step *clobbers* a protected condition for another step.
- The planner must pay attention to the clausal links which are *protected*. The planner must ensure that *threats* (steps which can clobber protected preconditions) are order to come before or after the protected link.
- Threats can be added by adding ordering constraints to put the threat before the protected link (*demotion*) or after the protected link (*promotion*).

The Sussman Anomaly

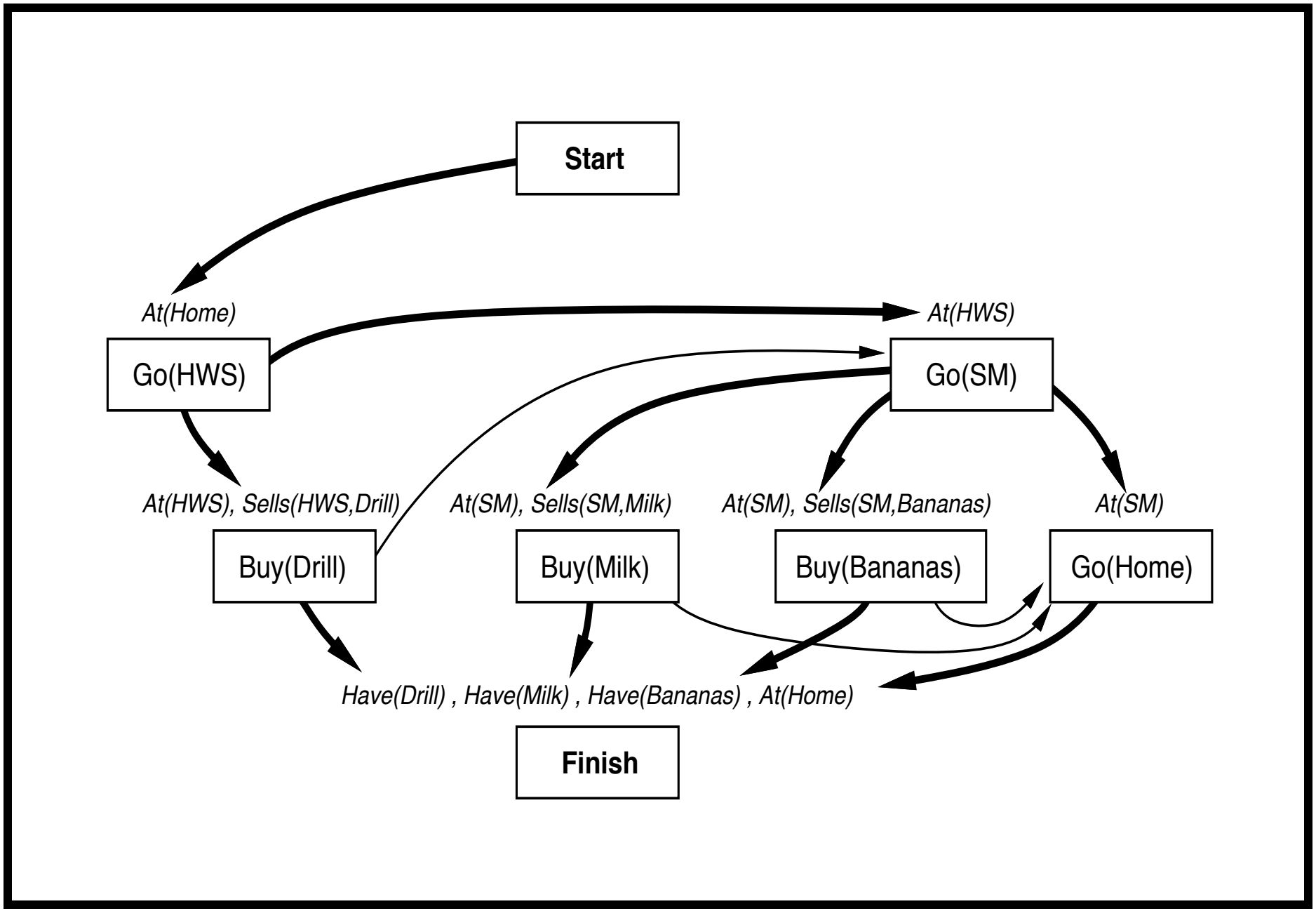
Focusing on one conjunct at a time can make clobbering unavoidable.

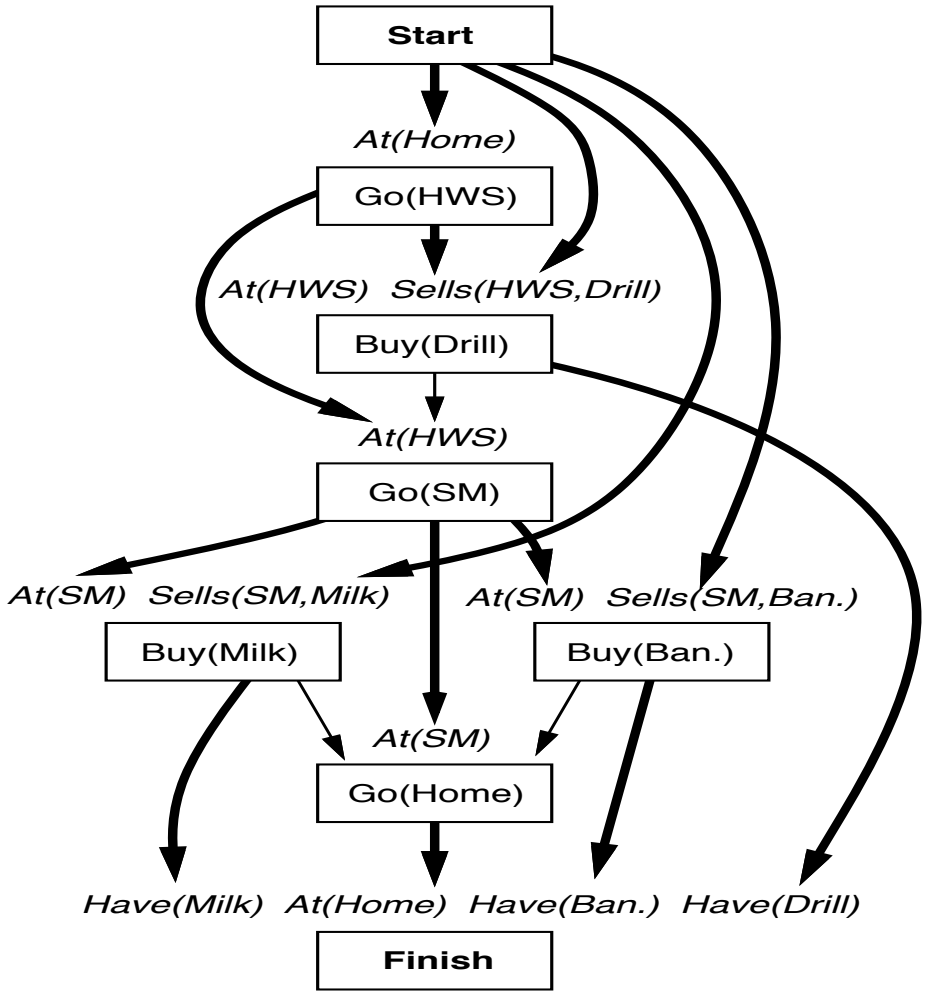



S1
S3
¬c
c
S2
(a)
S3
¬c
S1
c
S2
(b)
S1
c
S2
S3
¬c
(c)

Ordering constraints aren't always enough...

- Unfortunately, there is no way to reorder the *Go* threat because any order will delete the *At(home)* condition of the other step.
- When a planner can't resolve a threat, it has no choice but to backtrack.
- Suppose we trying adding a causal link from *Go(HWS)* to *Go(SM)*. Now *Go(SM)* threatens the *At(HWS)* precondition of *Buy(drill)*.
- We can resolve this threat by ordering *Go(SM)* to come after *Buy(drill)*.


Start
At(Home)
At(HWS)
Go(HWS)
Go(SM)
At(HWS), Sells(HWS,Drill)
At(SM), Sells(SM,Milk)
At(SM), Sells(SM,Bananas)
At(SM)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Go(Home)
Have(Drill) , Have(Milk) , Have(Bananas) , At(Home)
Finish


Start
At(Home)
Go(HWS)
At(HWS) Sells(HWS,Drill)
Buy(Drill)
At(HWS)
Go(SM)
At(SM) Sells(SM,Milk)
At(SM) Sells(SM,Ban.)
Buy(Milk)
Buy(Ban.)
At(SM)
Go(Home)
Have(Milk) At(Home) Have(Ban.) Have(Drill)
Finish

```
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
```

```
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
```

```
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 --c--> S_need to LINKS(plan)
  add the ordering constraint S_add ≺ S_need to ORDERINGS(plan)
  if S_add is a newly added step from operators then
    add S_add to STEPS(plan)
    add Start ≺ S_add ≺ Finish to ORDERINGS(plan)
```

```
procedure RESOLVE-THREATS(plan)

  for each S_threat that threatens a link S_i --c--> S_j in LINKS(plan) do
    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
```

Possible Threats

Variables can be left unbound in plans. If an operator produces an effect that could cause a threat if the variable takes a certain binding, then it is a *possible threat*.

There are three general apporaches to dealing with possible threats:

1. Resolve now by forcing a variable binding. The commitment may cause trouble later though.

2. Resolve now with an inequality constraint ($x \neq home$). Less commitment, but more complicated for unification algorithms.

3. Ignore possible threats and only deal with them when they become known threats. For example, if $x = home$ is added then resolve the threat. Low commitment, but can't say for sure that the plan is a solution.

Practical Applications for Planners

Job shop scheduling: assembling, manufacturing

Space missions: orchestrating observational equipment to maximize data acquisition while minimizing time and energy consumption.

Construction: Building facilities, airplanes, spacecraft, etc.

Event scheduling: Scheduling meetings, classes, and other events.

Limitations of the STRIPS language

Hierarchical planning: Generating complex plans often requires abstract planning over increasingly detailed search spaces.

Complex state conditions: STRIPS variables have limited in their complexity. For example, there is no quantification and no conditional statements.

Representing time: The STRIPS framework assumes that everything happens instantly. There is no way to represent duration, deadlines, time windows, etc.

Resource limitations: In the real world, resources are limited. You need to represent the fact that the number of available workers, equipment, money, etc. is constrained.