

# Planning

# Motivation

- previously considered decision-making problems from the perspective of **search** and **game-playing**
  - Given current state, enumerate all possible future states
  - Pick best action to execute from current state
- ideal for problems where consequences of action are not well understood in advance, e.g. missionaries and cannibals, NIM, 8-puzzle

# Grocery Shopping as Search

“from home, get milk, some bananas, and a cordless drill”

- assume we will use search technique with heuristic: minimize the number of items we have not yet acquired
- how to choose best operator? may be thousands!
- before agent can purchase anything, it has to get to the store, but how does a search technique know this?

# A Better Way (sometimes)

- many actions are obviously useless or unproductive
- can be quite expensive to examine all of these
- what if we know outcome of actions?

**planning** = find a sequence of actions that achieves a goal when performed in a given state

# First-Order Logic (FOL)

- Propositional logic: propositions (sentences)
- FOL adds quantification ( $\forall$ ,  $\exists$ ) and predicates

# Predicates

- assume that Spot and Fido are dogs
- then the predicate,  $\text{Dog}(x)$ 
  - returns TRUE if  $x$  is Spot or Fido

# Unification

- Let  $p$  and  $q$  be sentences in FOL
- Let  $U$  be a **unifier**, i.e., some set of substitutions of values for variables
- $subst(U,x)$  is the result of applying the substitutions of  $U$  to sentence  $x$
- If  $subst(U,p) = subst(U,q)$  then  $UNIFY(p,q) = U$ 
  - The **unification** of  $p$  and  $q$  is the result of applying  $U$  to both of them.

# Language of Planning Problems

- States: conjunction of positive literals
  - e.g., *Poor*  $\wedge$  *Unknown* or  
*At(Plane<sub>1</sub>, Melbourne)*
- Goals: a partially specified state
  - e.g., *Rich*  $\wedge$  *Famous* or *At(P<sub>2</sub>, Tahiti)*

# Comparison of Actions

## search

- described by state transitions
- must be considered in-order

## planning

- described by preconditions & effects
- can be considered in **any** order

# STRIPS

- language used by most classical planners
- initial state: specifies everything that is true in the world (everything else assumed false)  
 $At(Home) \wedge Have(Money)$
- goals: represented by conjunctions of literals  
 $At(Home) \wedge Have(Milk) \wedge Have(Bananas) \wedge Have(Cookies)$
- goals can contain variables, e.g., “get to a store that sells milk”  
 $At(x) \wedge Sells(x, Milk)$

# Representation of Actions

- **preconditions**
  - what must be true before action can be performed
- **effects** or **post-conditions**
  - what must be true (what changed) after action is executed

e.g.,

Action (*Fly* ( $p$ ,  $from$ ,  $to$ ),

PRECOND:  $At(p,from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

EFFECT:  $\neg At(p,from) \wedge At(p,to)$

- effects may be divided into add list and delete list (for negative literals)
- assumption that every literal not mentioned in EFFECT remains unchanged

# Forward State-Space Search (Progression) Planners

- search forward from initial state
- determine which operators apply using preconditions
- use effects lists to compute new state

# Algorithm

procedure PROGRESSION-PLAN( $s$ ,  $plan$ )

  for some possible operator,  $\alpha_i$

$u \leftarrow$  result of UNIFY( $s$ , preconditions of  $\alpha_i$ )

    if unification step succeeds then

      add  $\alpha_i$  to  $plan$

      apply substitution list of  $u$  to DeleteList( $\alpha_i$ ) and AddList( $\alpha_i$ )

$t \leftarrow s$

      for each  $d_j$  in DeleteList( $\alpha_i$ ) delete  $d_j$  from  $t$

      for each  $a_j$  in AddList( $\alpha_i$ ) add  $a_j$  to  $t$

      if GOAL-TEST ( $t$ ) succeeds then return  $plan$

  return PROGRESSION-PLAN ( $t$ ,  $plan$ )

# Practical Considerations

- **main problem:** often have huge search space because of branching factor
- not practical for real-world problems

# Backward State-Space (Regression) Planners

- search backwards from goal state to initial state
- **advantage**: consider only relevant actions; i.e., those that achieve one of the conjuncts of the goal
- significantly decreases branching factor

# Regression Planner

procedure REGRESSION-PLAN ( $t$ ,  $plan$ )

for some possible operator,  $\alpha$

if current state contains  $\geq 1$  literal  $L$ , that unifies with a member  $a_i$  of  $AddList(\alpha)$

add  $\alpha$  to head of  $plan$

$u \leftarrow$  result of  $UNIFY(L, a_i)$

$p' \leftarrow$  apply substitution list of  $u$  to  $Preconditions(\alpha)$

$t' \leftarrow$  apply substitution list of  $u$  to terms of  $t$

regress each member  $m$  of  $t'$  through  $a$  as follows:

if  $m$  is in  $AddList(\alpha)$  then True

else if  $m$  is in  $DeleteList(\alpha)$  then False

else leave  $m$  as is

previous state  $s \leftarrow UNION(p', t')$

if  $s = INITIAL-STATE$  then return  $plan$

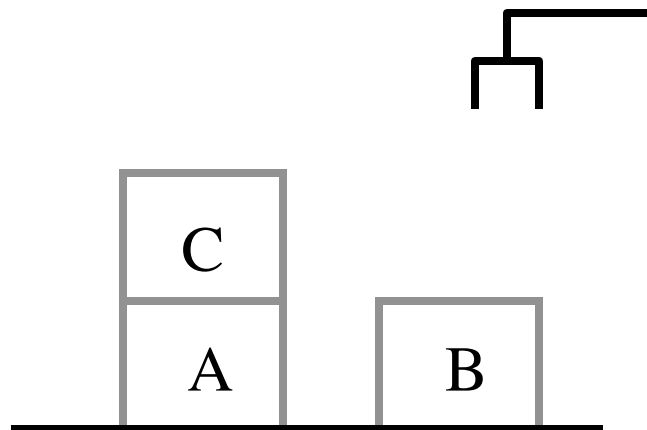
return REGRESSION-PLAN ( $s$ ,  $plan$ )

# Blocks World Example

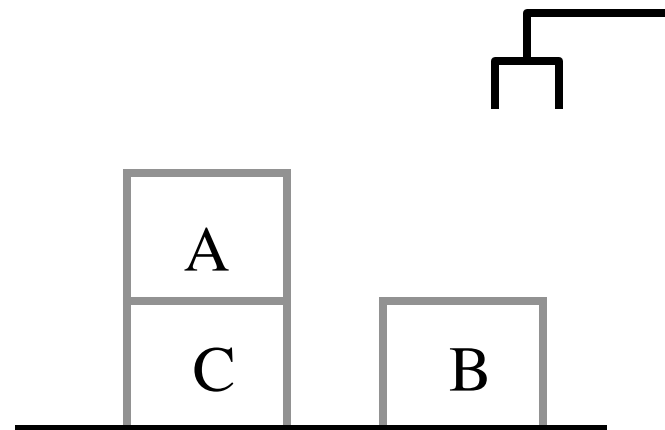
- *pickup(x)*
  - P, D: ontable(x), clear(x), HE (i.e., hand empty)
  - A: holding(x)
- *putdown(x)*
  - P, D: holding(x)
  - A: ontable(x), clear(x), HE
- *stack(x, y) // putdown(x) on another block(y)*
  - P,D: holding(x), clear(y)
  - A: HE, on(x,y), clear(x)
- *unstack(x, y) // pickup (x) sitting on block(y)*
  - P,D: HE, clear(x), on(x,y)
  - A: holding(x), clear(y)
- Note: in general, P and D are **not** equivalent

# Exercise

- write the initial state description
- apply progression planning to reach the goal state

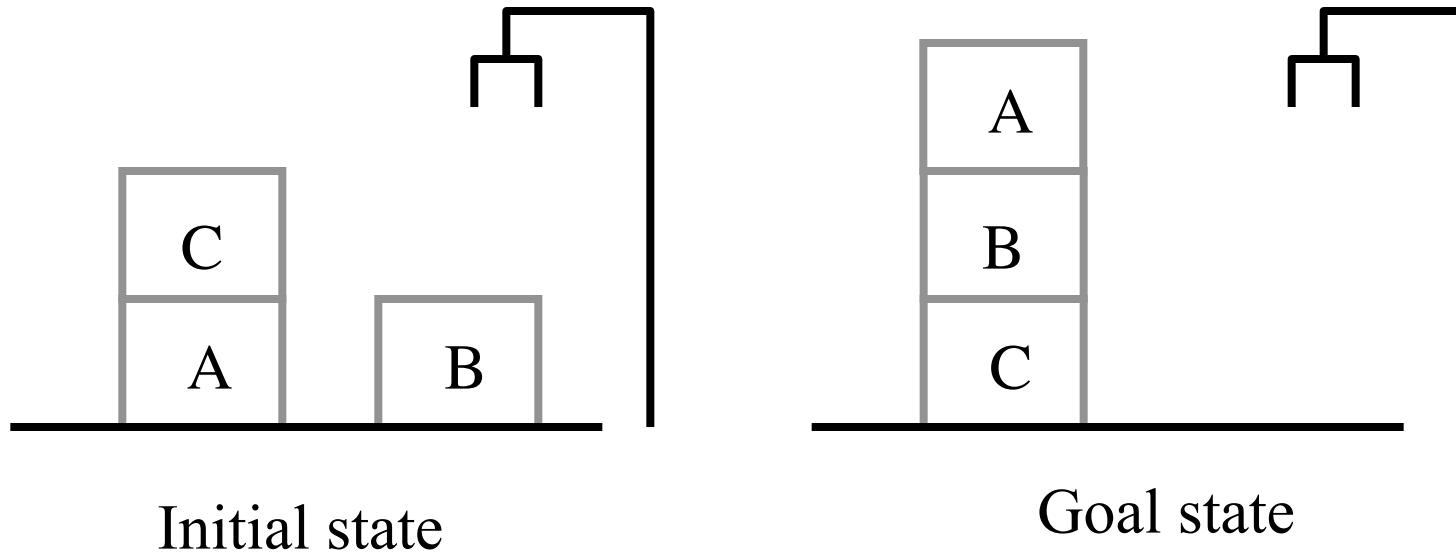


Initial state



Goal state

# Sussman Anomaly



- if we do  $\text{On}(A,B)$  first, likely to undo it as we try to achieve  $\text{On}(B,C)$
- similarly, if we do  $\text{On}(B,C)$  first, likely to undo it with  $\text{On}(A,B)$
- problem stems from interaction of goals in linear planners
- forward and backward state-space planners are capable of handling the Sussman anomaly, but inordinate effort required

# Grocery Shopping Problem

- initial state:  
*At(Home)*
- goal:  
*Have(Drill) ^ Have(Milk) ^ Have(Banana) ^ At(Home)*
- actions:  
Action (*Go (there)*),  
PRECOND: *At(here)*  
EFFECT: *At(there) ^ ¬ At(here)*)  
Action (*Buy (x)*),  
PRECOND: *At(store) ^ Sells(store, x)*  
EFFECT: *Have(x)*)

# Partial Order Planning (POP)

start with initial plan

- consists only of **Start** and **Finish** states
- at each iteration, add one more step
- if inconsistent, backtrack

only adds steps that achieve unachieved preconditions

# Start and Finish Steps

## ACTION: Start

- EFFECT: At(Home) ^ Sells (HWS, Drill) ^ Sells (SM, Milk) ^ Sells (SM, Banana)  
Notice: Start step adds, as its effect, all necessary facts to knowledge base.

## ACTION: Finish

- PRECOND: Have(Drill) ^ Have(Milk) ^ Have(Banana) ^ At(Home)

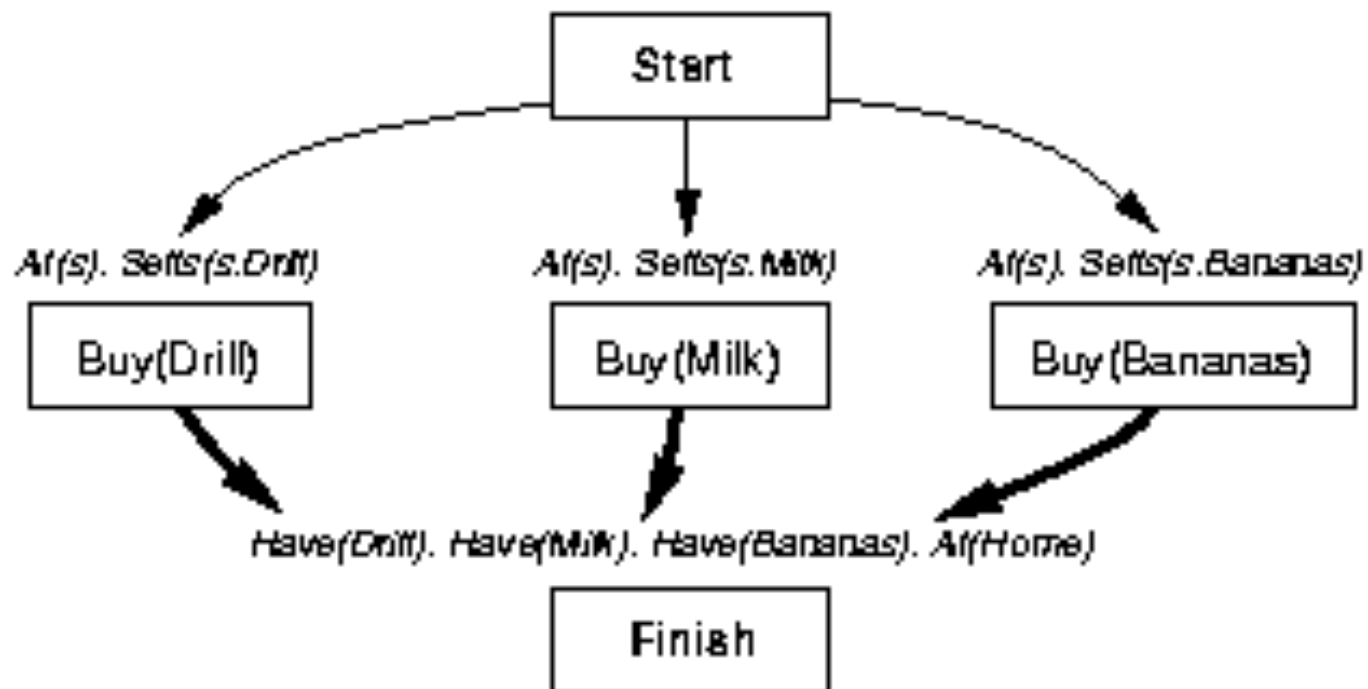
**Start**

*At(Home) Sells(HWS,Drill) Sells(SM,Milk) Sells(SM,Ban.)*

*Have(Milk) At(Home) Have(Ban.) Have(Drill)*

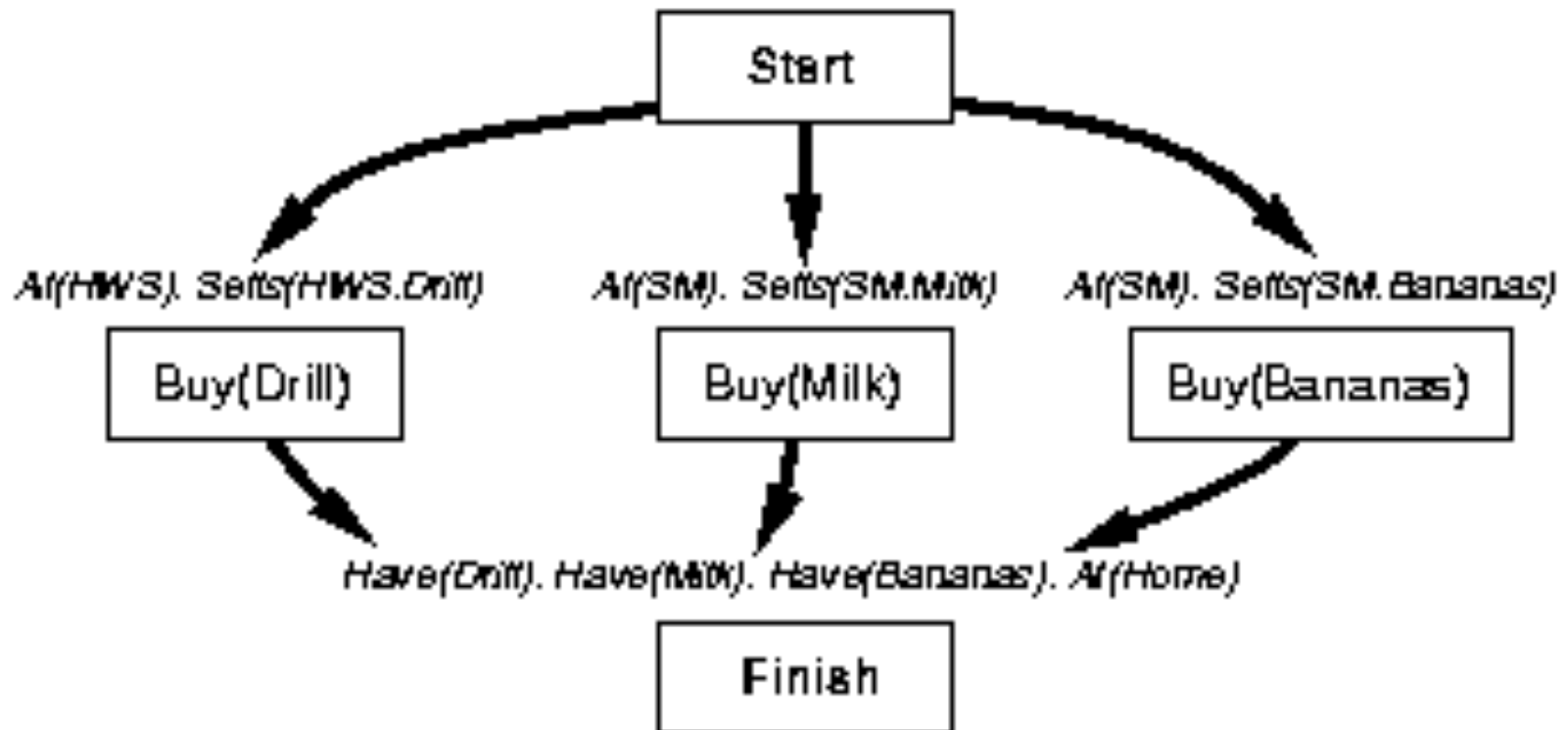
**Finish**

# Achieve preconditions of 'Finish'



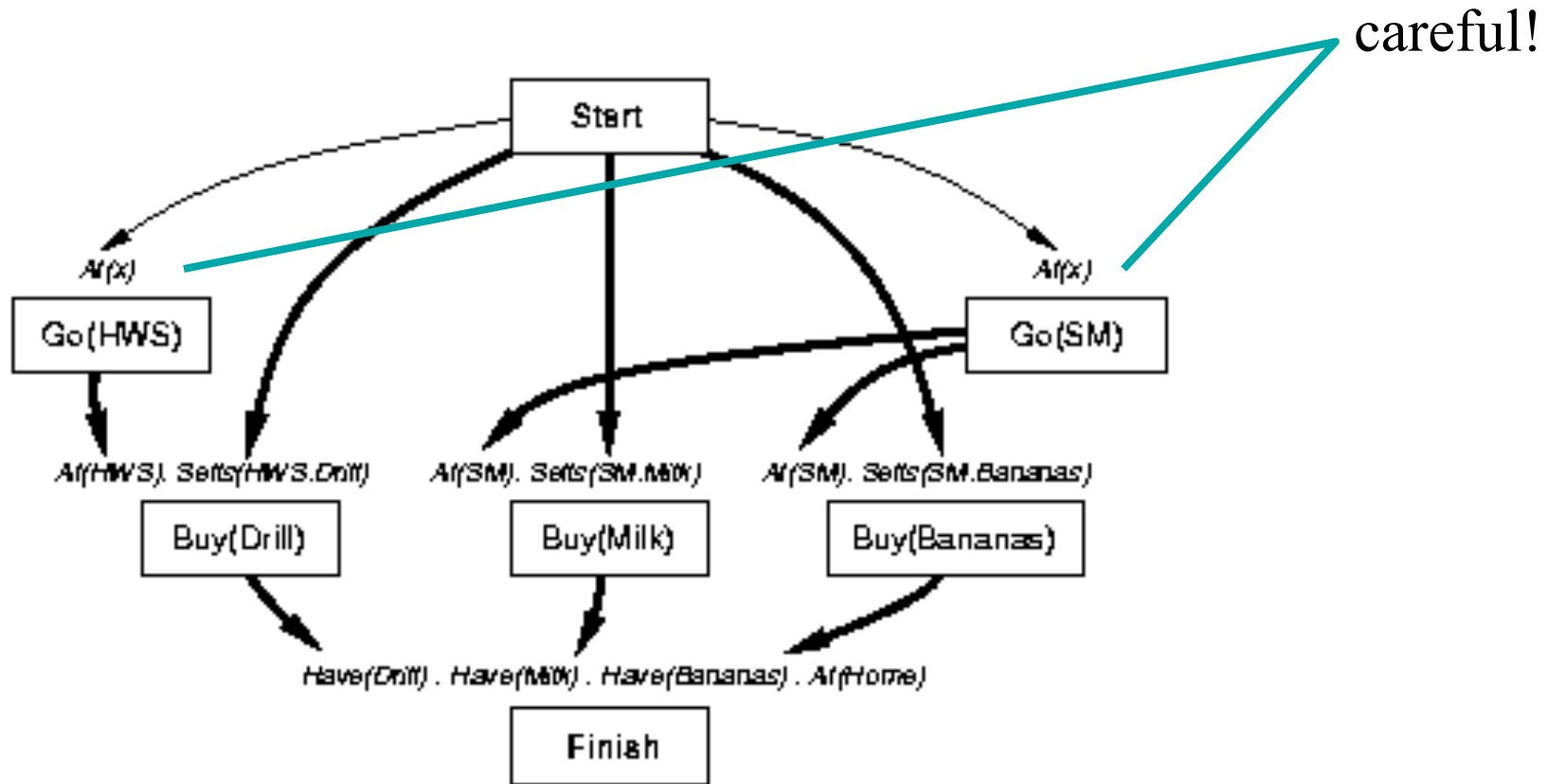
# Achieve preconditions of 'Buy'

- start with Sells()



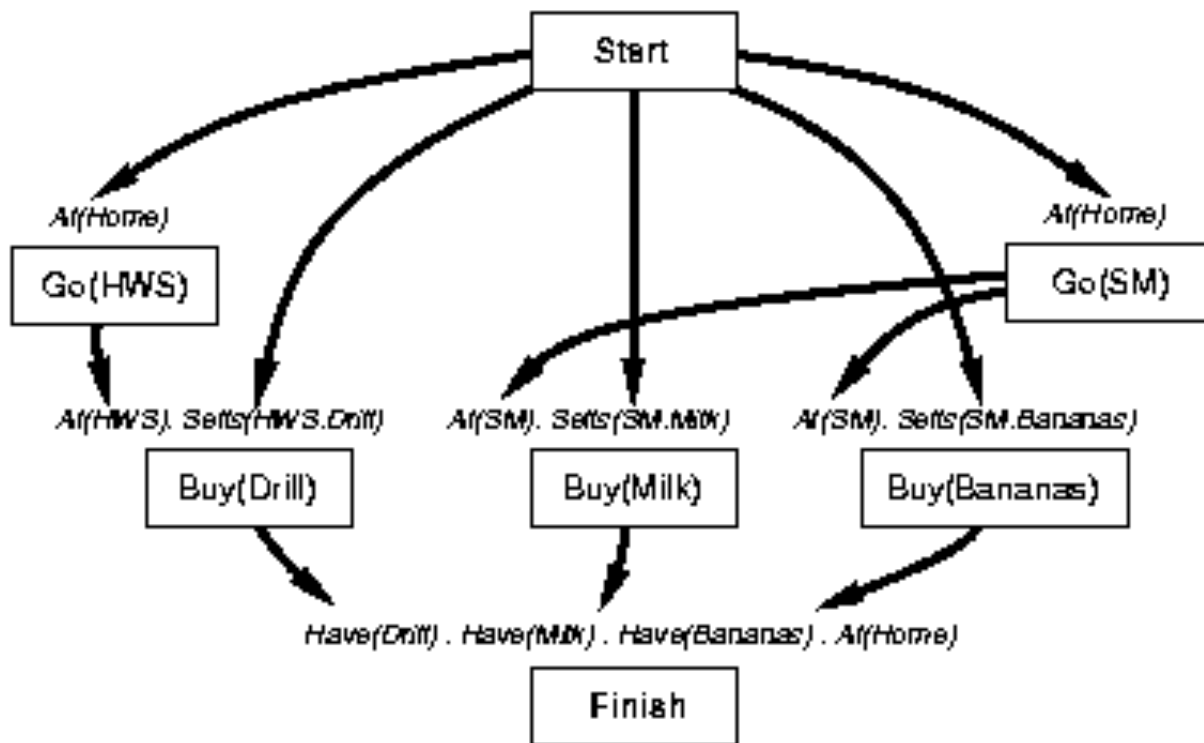
# Achieve preconditions of 'Buy'

- now satisfy the  $At()$  preconditions

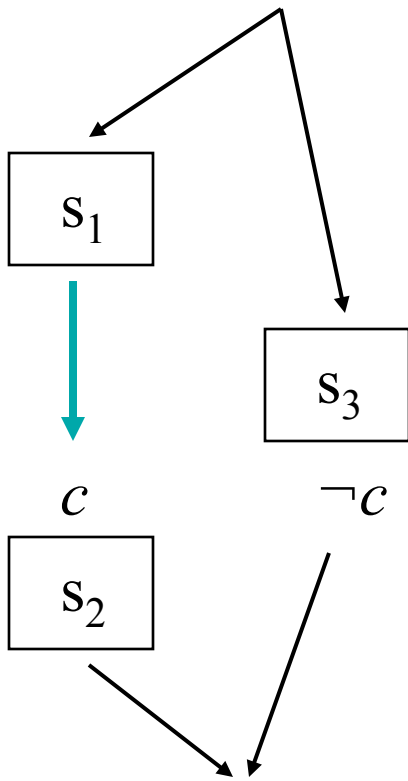


# First Attempt

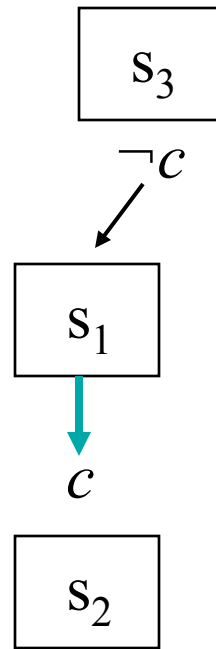
- Link to *At(Home)* from *Start*



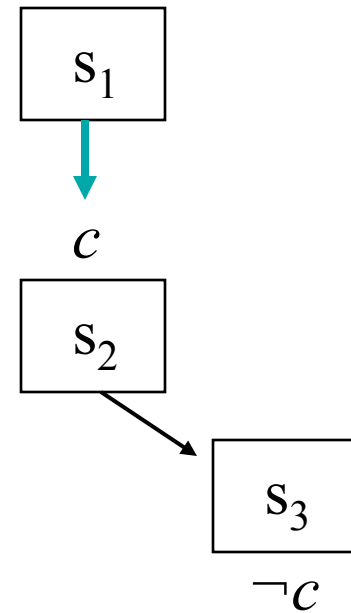
# Protecting Causal Links



Threat



$S_3$  demoted

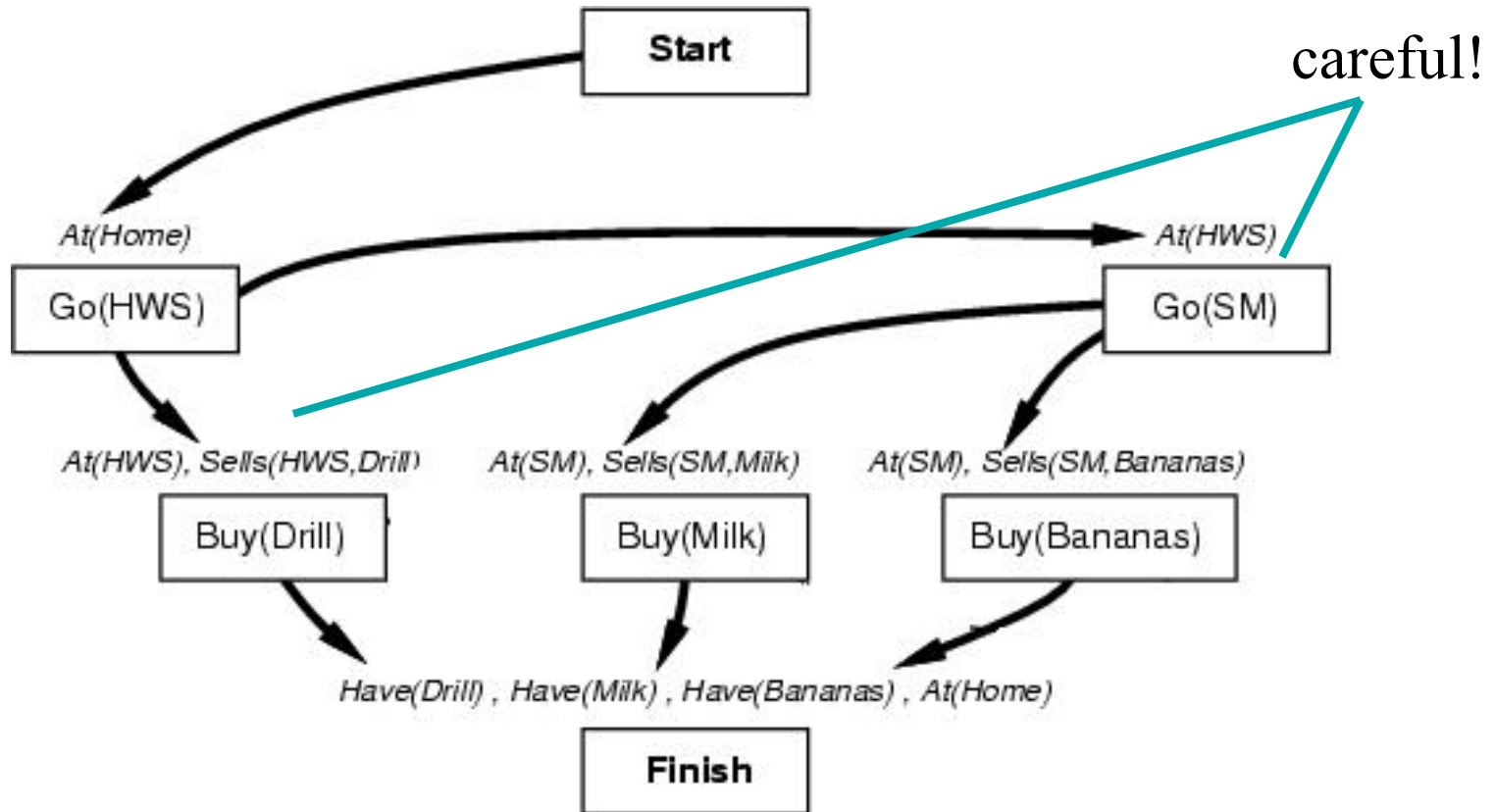


$S_3$  promoted



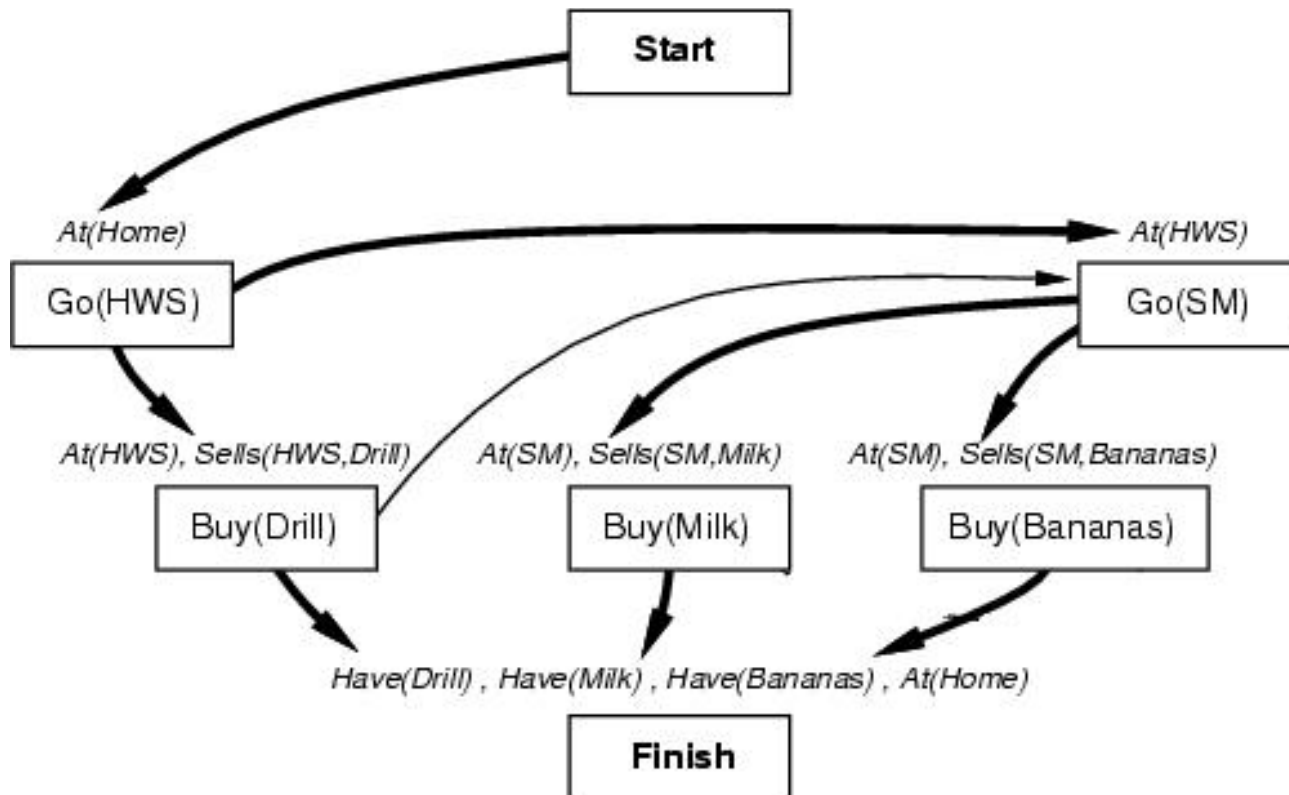
# Second Attempt

- Add causal link from **Go(HWS)** to **Go(SM)**



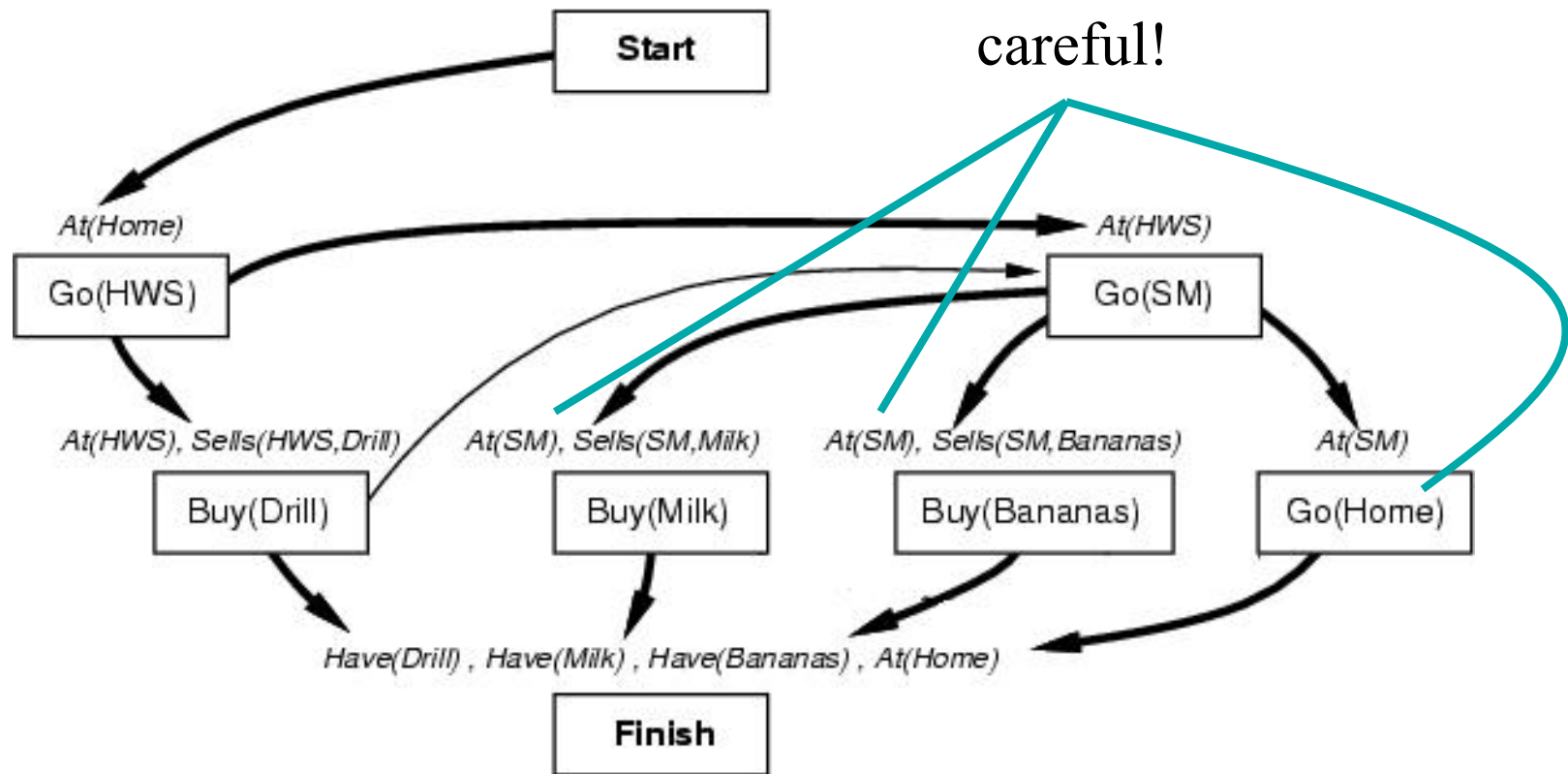
# Resolving the Threat

- Add temporal constraint for Buy(Drill) to precede Go(SM)



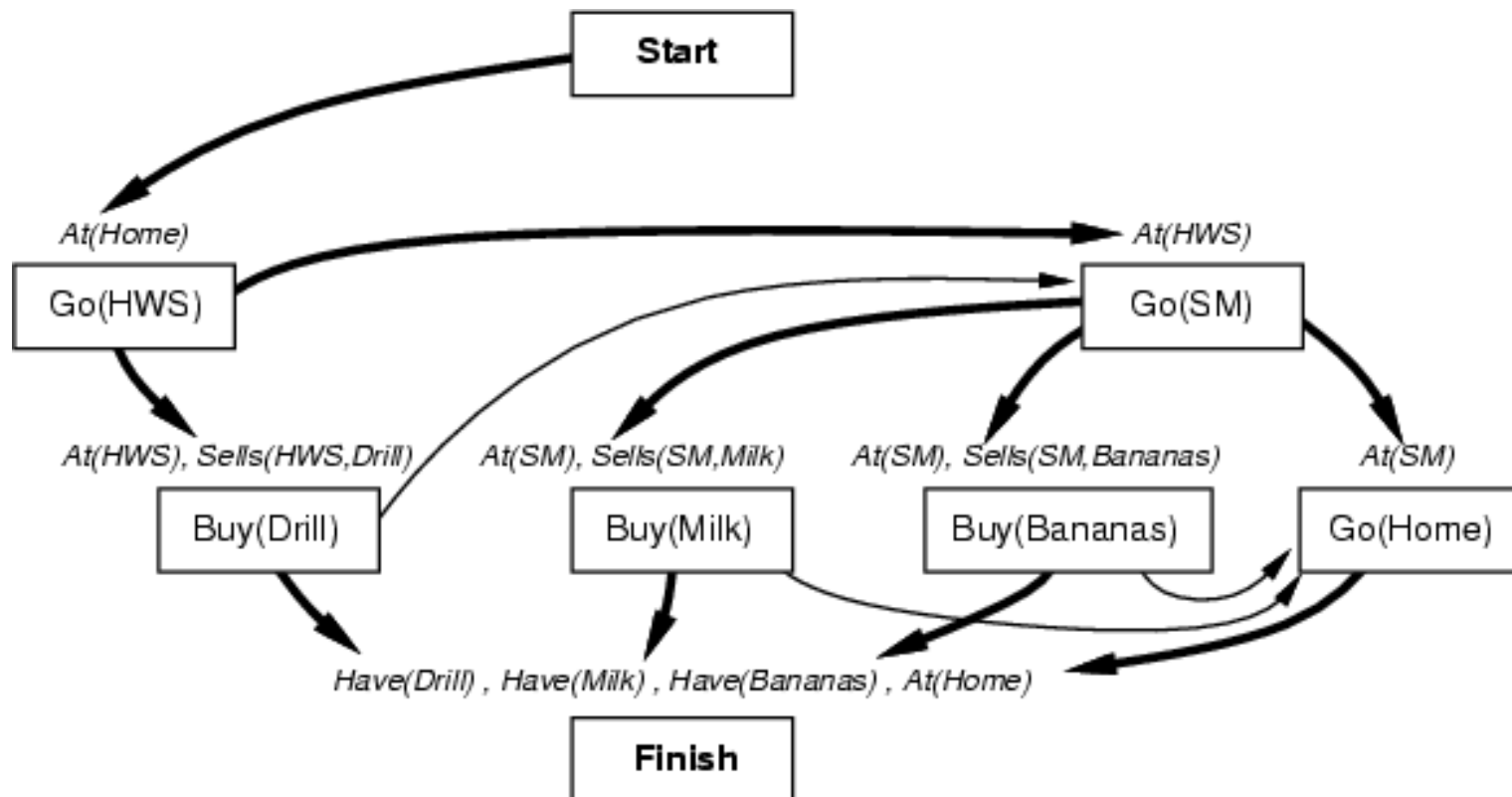
# And now to get home...

- But again... beware of threats!



# Resolving the Threat

- Add temporal constraint for Buy(Milk), Buy(Bananas) to precede Go(Home)



# Unrolled Solution

