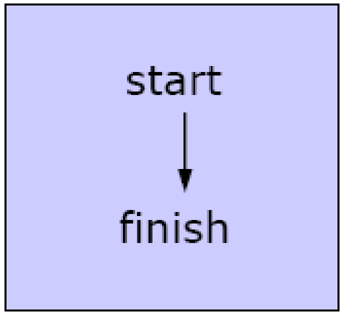
Partially Ordered Plan

- Plan
  - Steps
  - Ordering constraints
  - Variable binding constraints
  - Causal links
- POP Algorithm
  - Make initial plan



  - Loop until plan is a complete
    - Select a subgoal
    - Choose an operator
    - Resolve threats

Choose Operator

- Choose operator(c, S_{needs})
  - Choose a step S from the plan or a new step S by instantiating an operator that has c as an effect
  - If there's no such step, Fail
  - Add causal link $S \ \square_c \ S_{needs}$
  - Add ordering constraint $S < S_{needs}$
  - Add variable binding constraints if necessary
  - Add S to steps if necessary

Nondeterministic choice

- Choose – pick one of the options arbitrarily
- Fail – go back to most recent non-deterministic choice and try a different one that has not been tried before

Resolve Threats

- A step S threatens a causal link $S_i \square_c S_j$ iff $\neg c \in$ effects(S) and it's possible that $S_i < S < S_j$

- For each threat
  - Choose
    - Promote S : $S < S_i < S_j$
    - Demote S : $S_i < S_j < S$

- If resulting plan is inconsistent, then Fail

Threats with Variables

If c has variables in it, things are kind of tricky.

- S is a threat if there is any instantiation of the variables that makes $\neg$ c $\in$ effects(S)

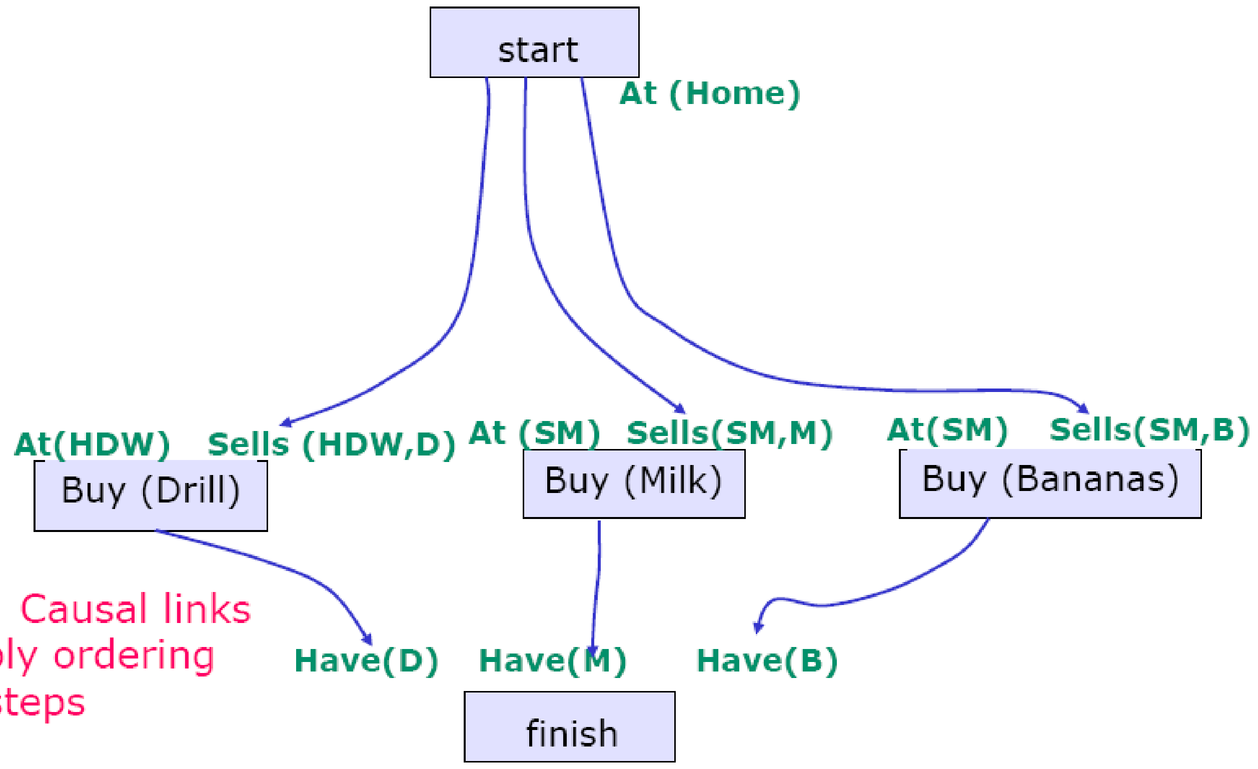
- We could possibly resolve the threat by adding a negative variable binding constraint, saying that two variables or a variable and a constant cannot be bound to one another

- Another strategy is to ignore such threats until the very end, hoping that the variables will become bound and make things easier to deal with

Shopping Domain

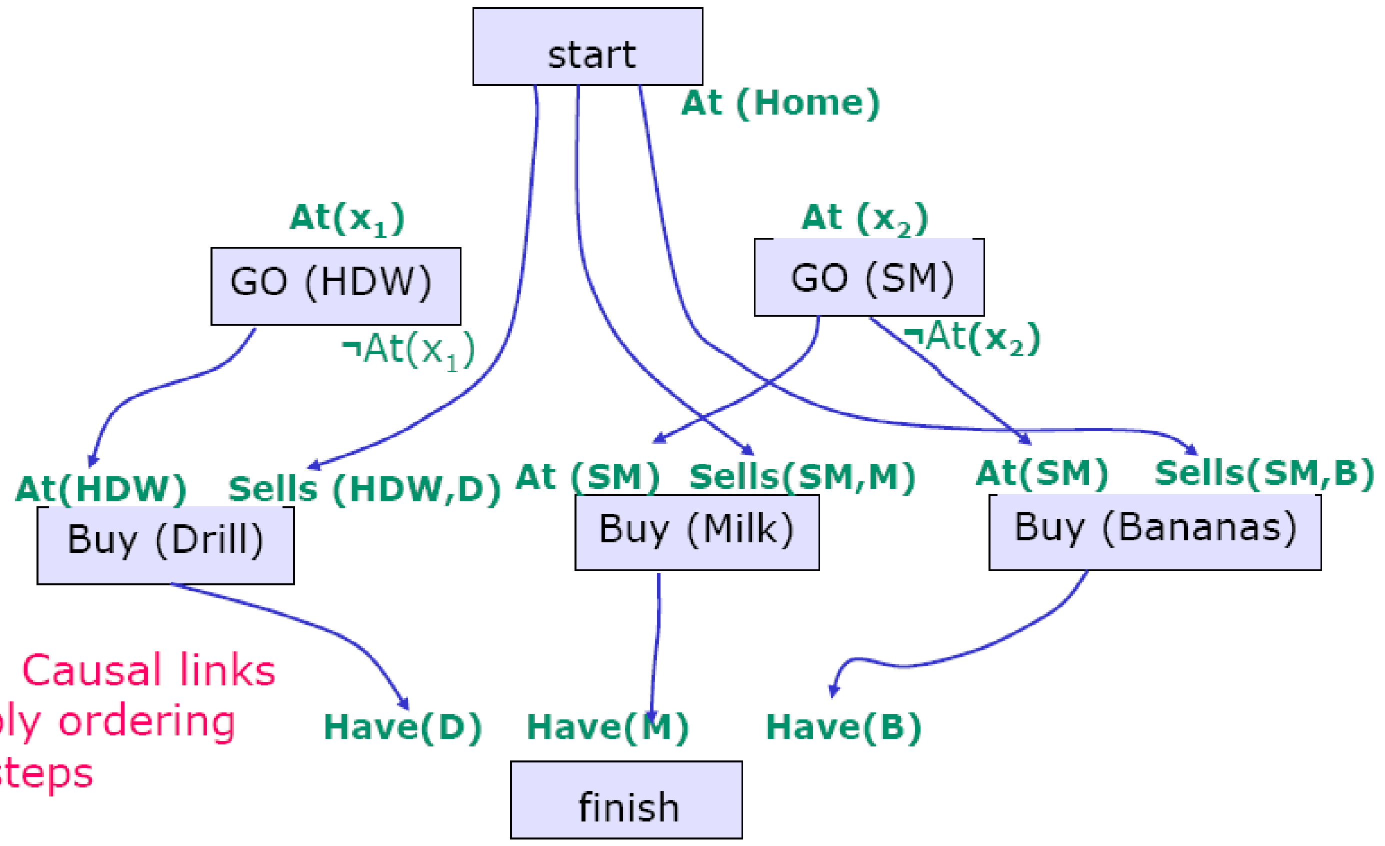
- Actions
  - Buy(x, store)
    - Pre: At(store), Sells(store, x)
    - Eff: Have(x)
  - Go(x, y)
    - Pre: At(x)
    - Eff: At(y), ¬At(x)
- Goal
  - Have(Milk) Æ Have(Banana) Æ Have(Drill)
- Start
  - At(Home) Æ Sells(SM, Milk) Æ Sells(SM, Banana) Æ Sells(HW, Drill)

Shop 'til You Drop!



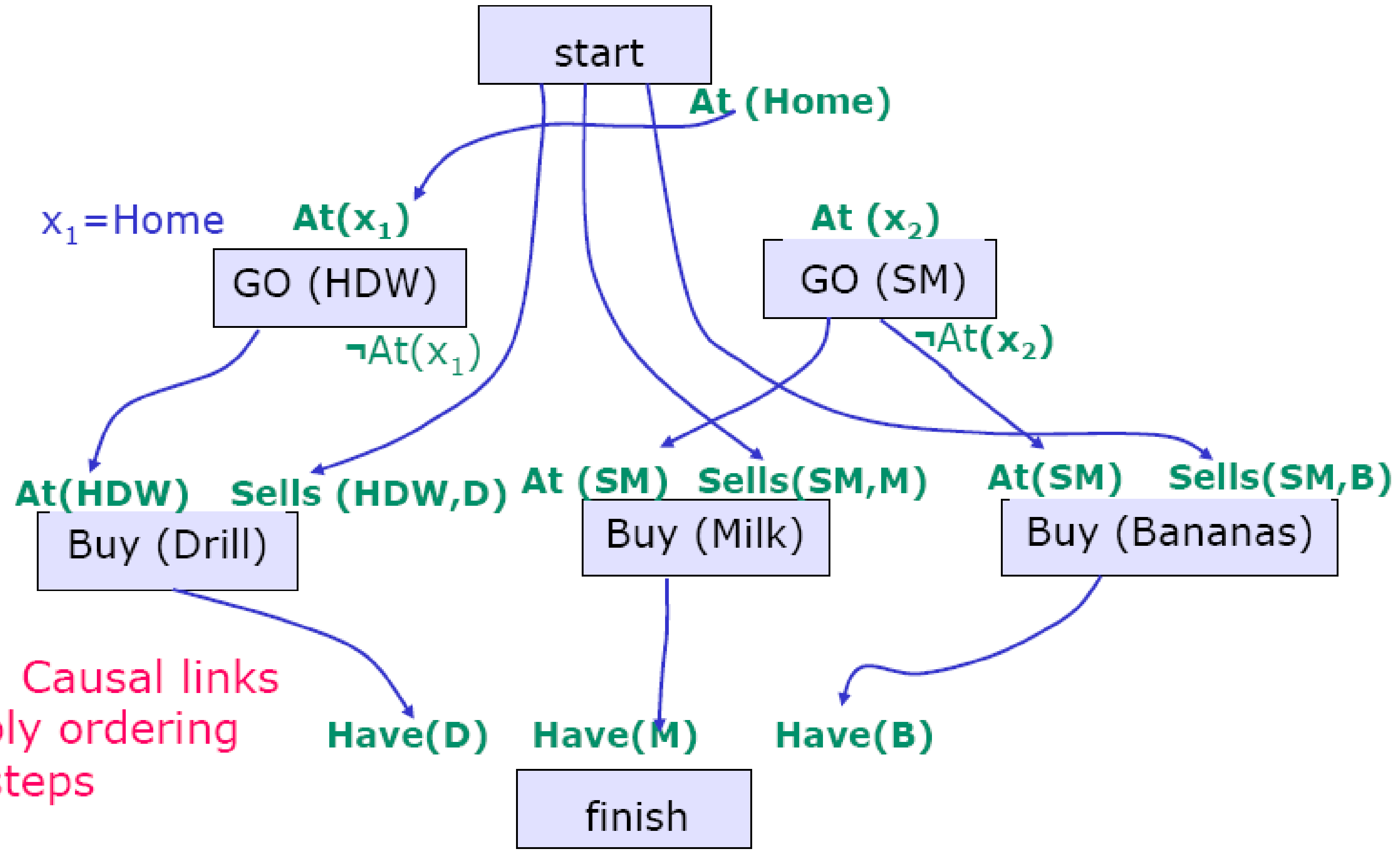
NB: Causal links imply ordering of steps

Shop 'til You Drop!



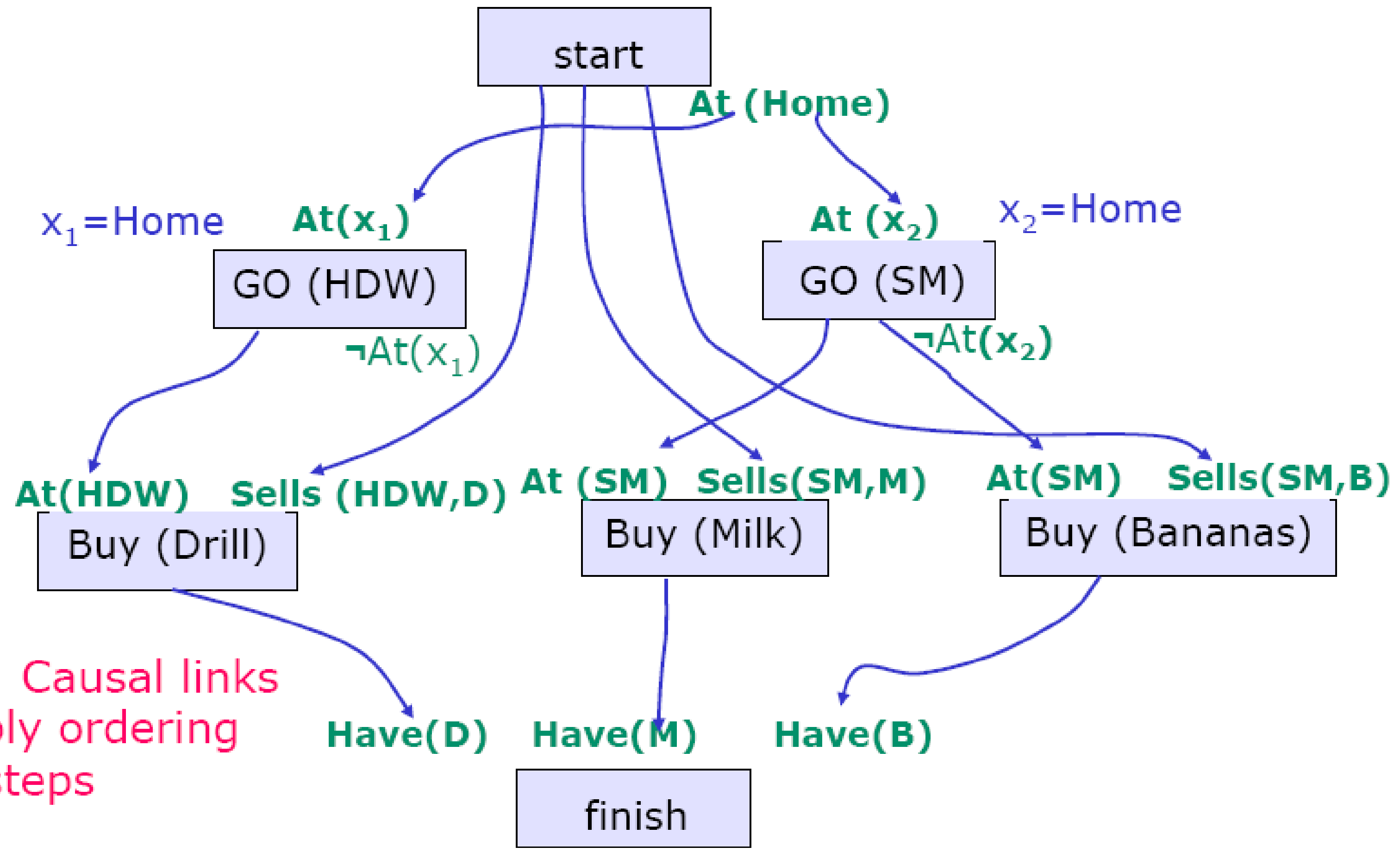
NB: Causal links imply ordering of steps

Shop 'til You Drop!



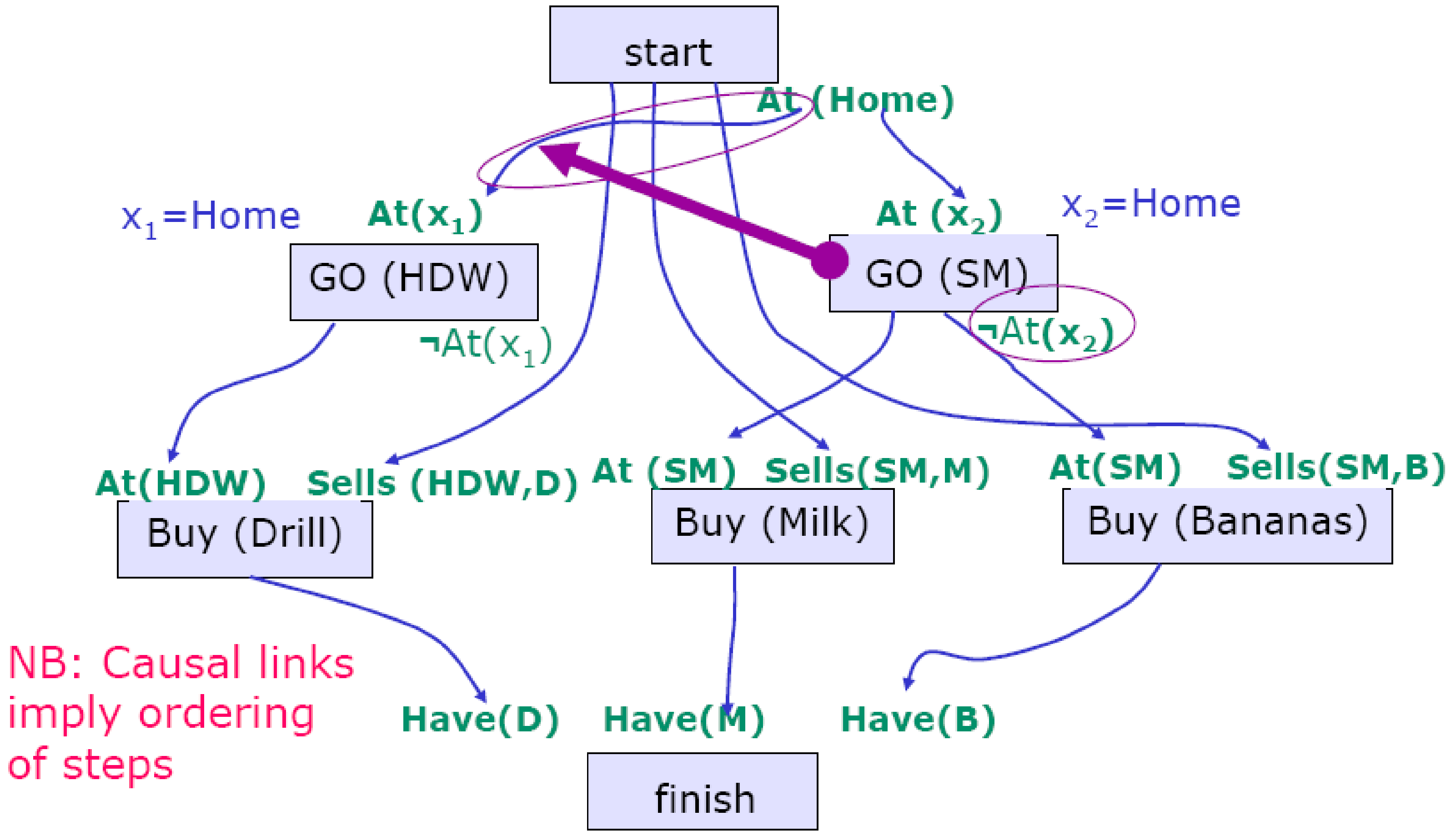
NB: Causal links imply ordering of steps

Shop 'til You Drop!



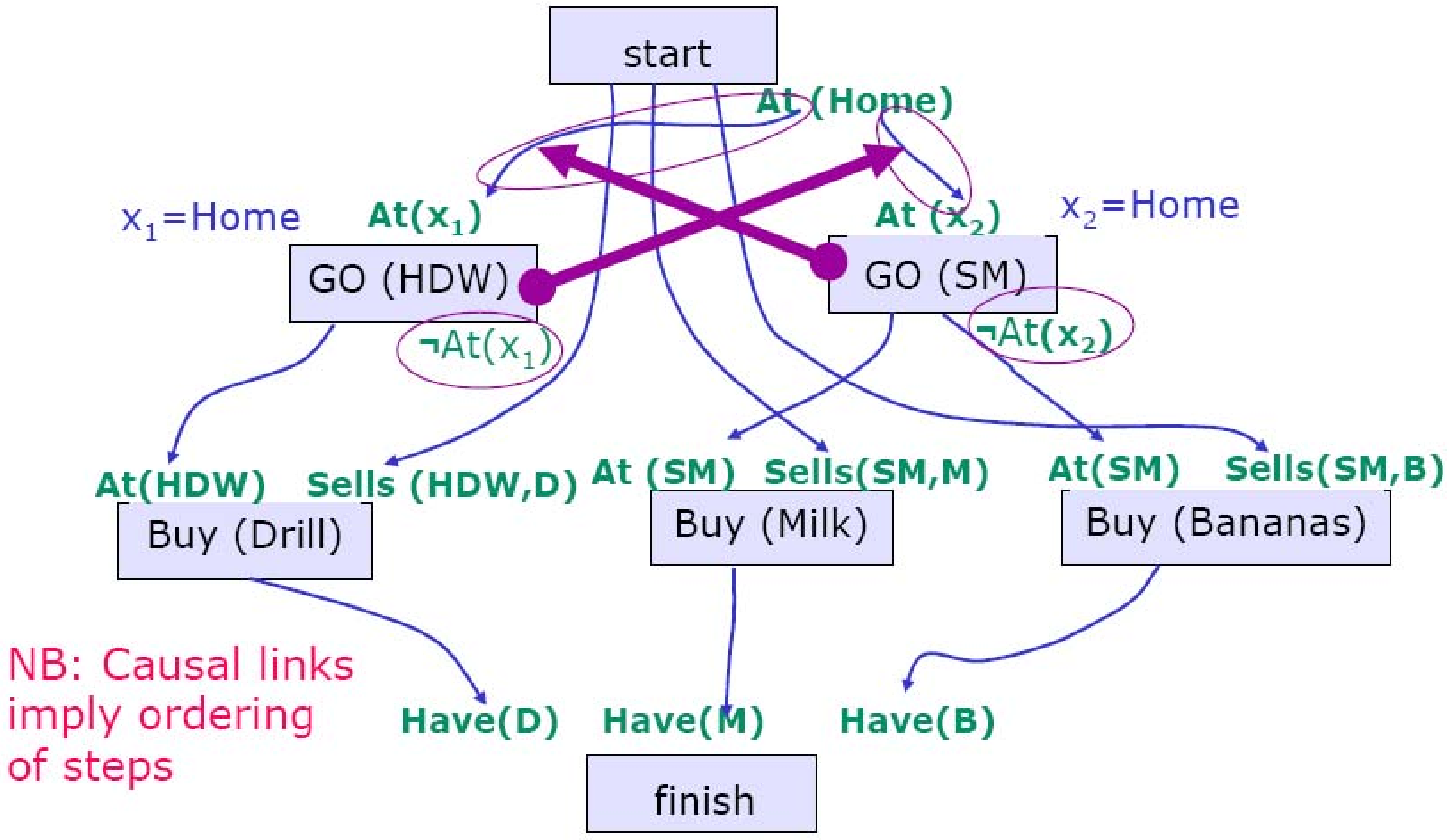
NB: Causal links imply ordering of steps

Shop 'til You Drop!



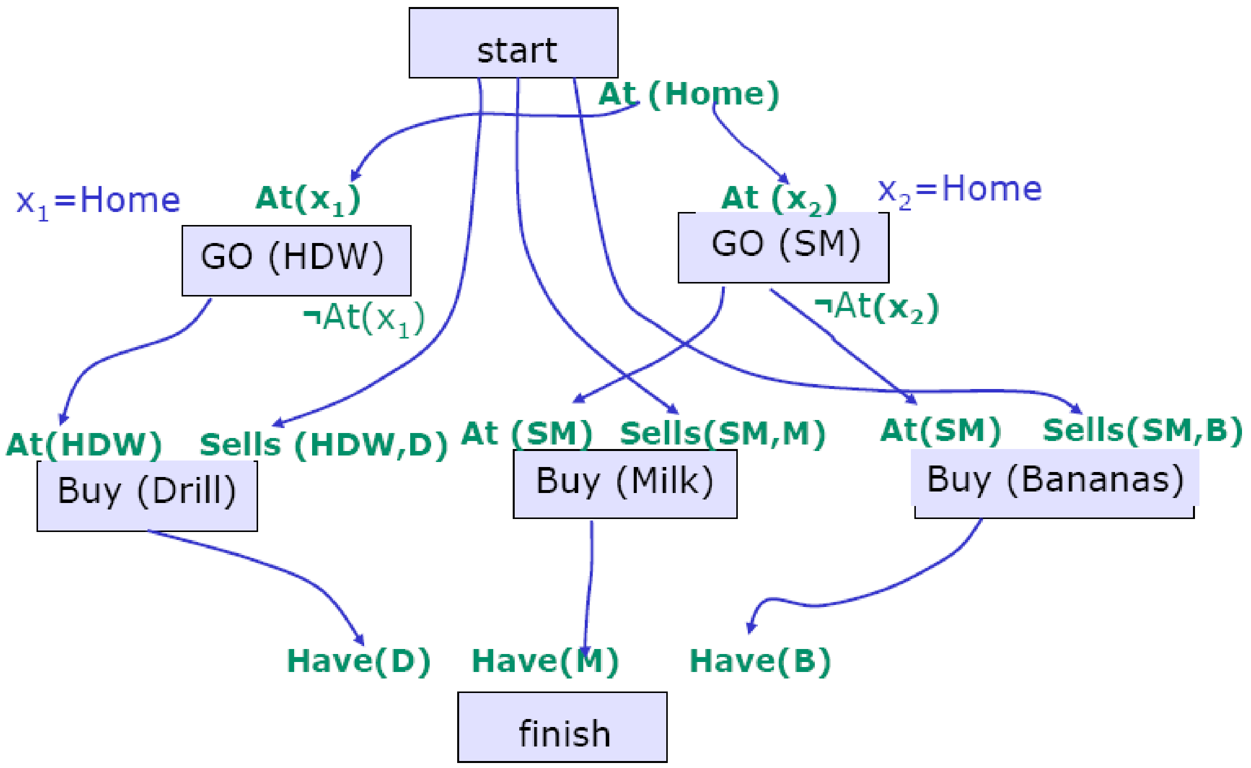
So, how can we resolve this threat? We're not allowed to put anything before the start step (there are implicit temporal constraints requiring everything to happen after start and before finish). So, maybe we could require Go(SM) to happen after Go(HDW).

Shop 'til You Drop!



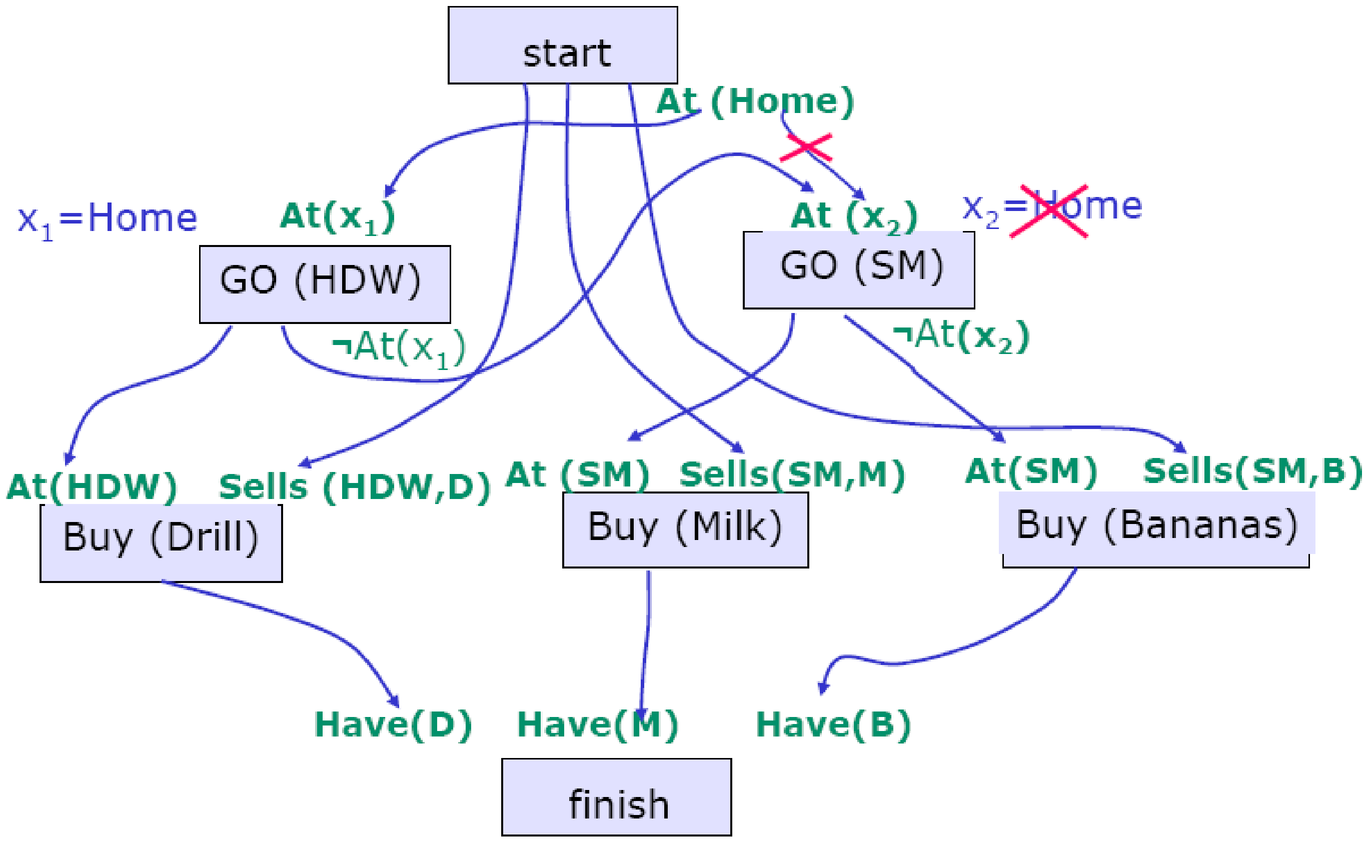
But we still have a symmetric problem. Look at the link from At(Home) to At(x2). It's threatened by Go(HDW). And the only way to fix it would be to make Go(HDW) happen after GO(SM).

Shop 'til You Drop!



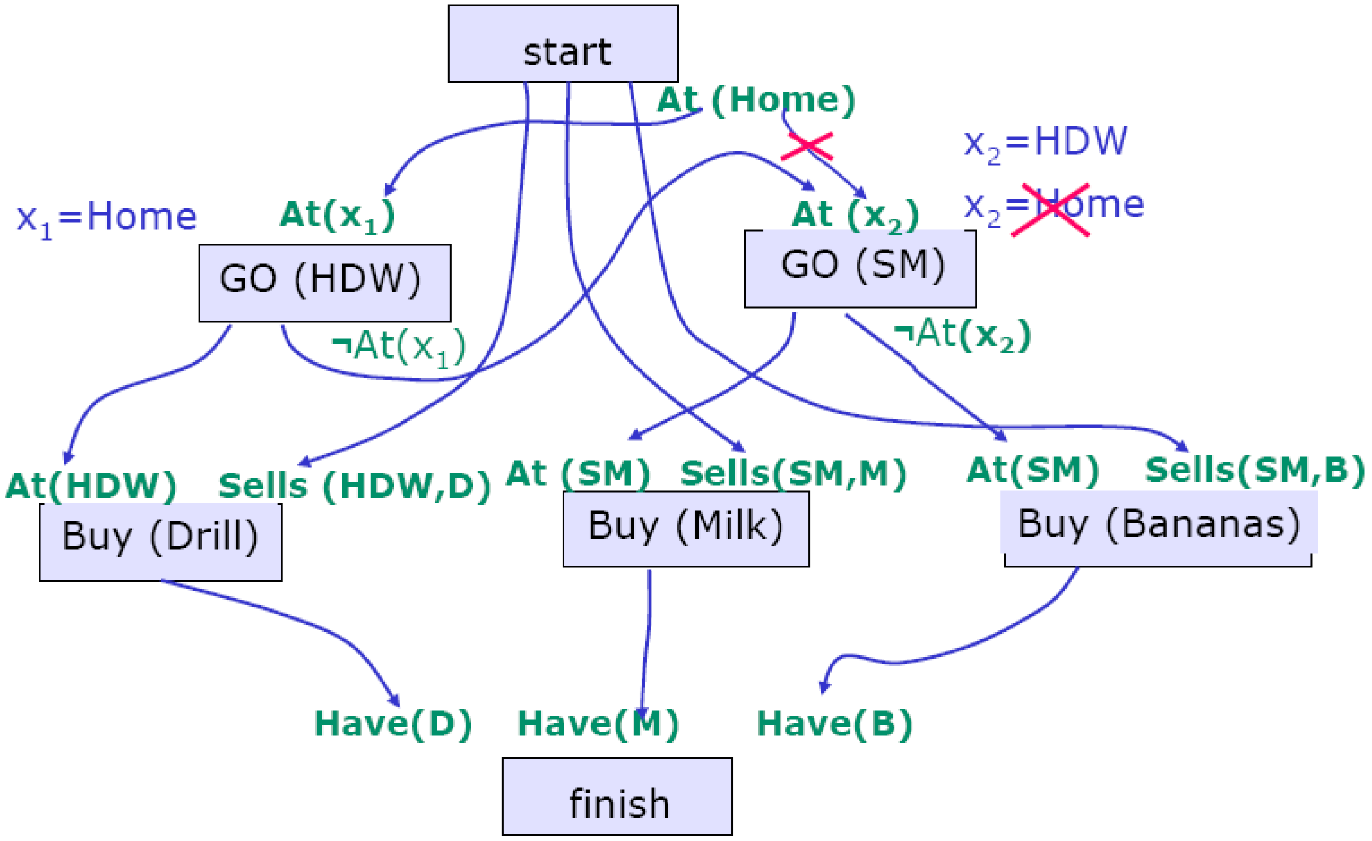
we can step back for a minute and see that we can't go from home to the hardware store and to the supermarket, without going back home in between. So, we need to do something else to make this plan work.

Shop 'til You Drop!



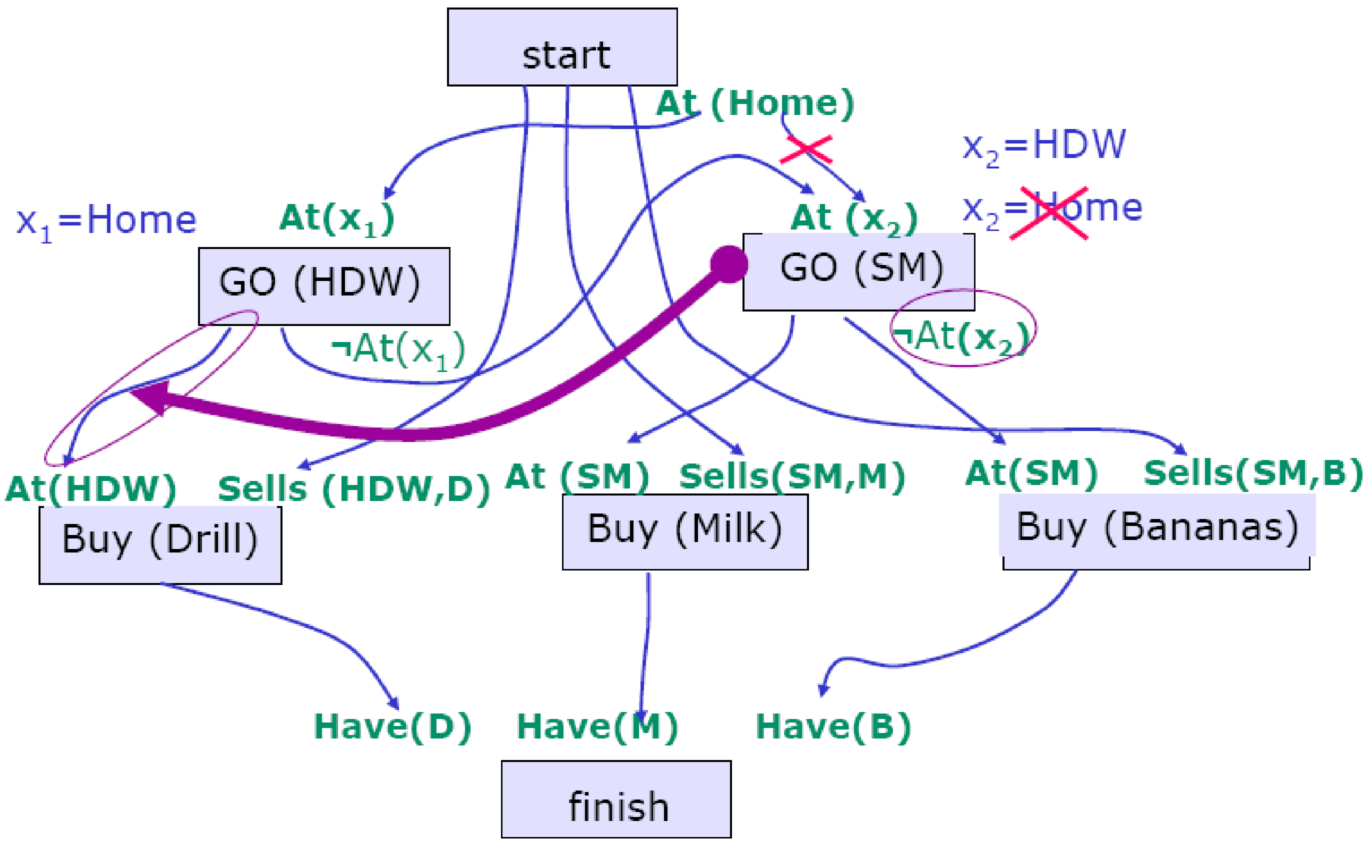
We could add a step to go home in between those two steps, but we can be more economical if we instead decide to satisfy At(x2) with the at(HDW) result of Go(HDW).

Shop 'til You Drop!



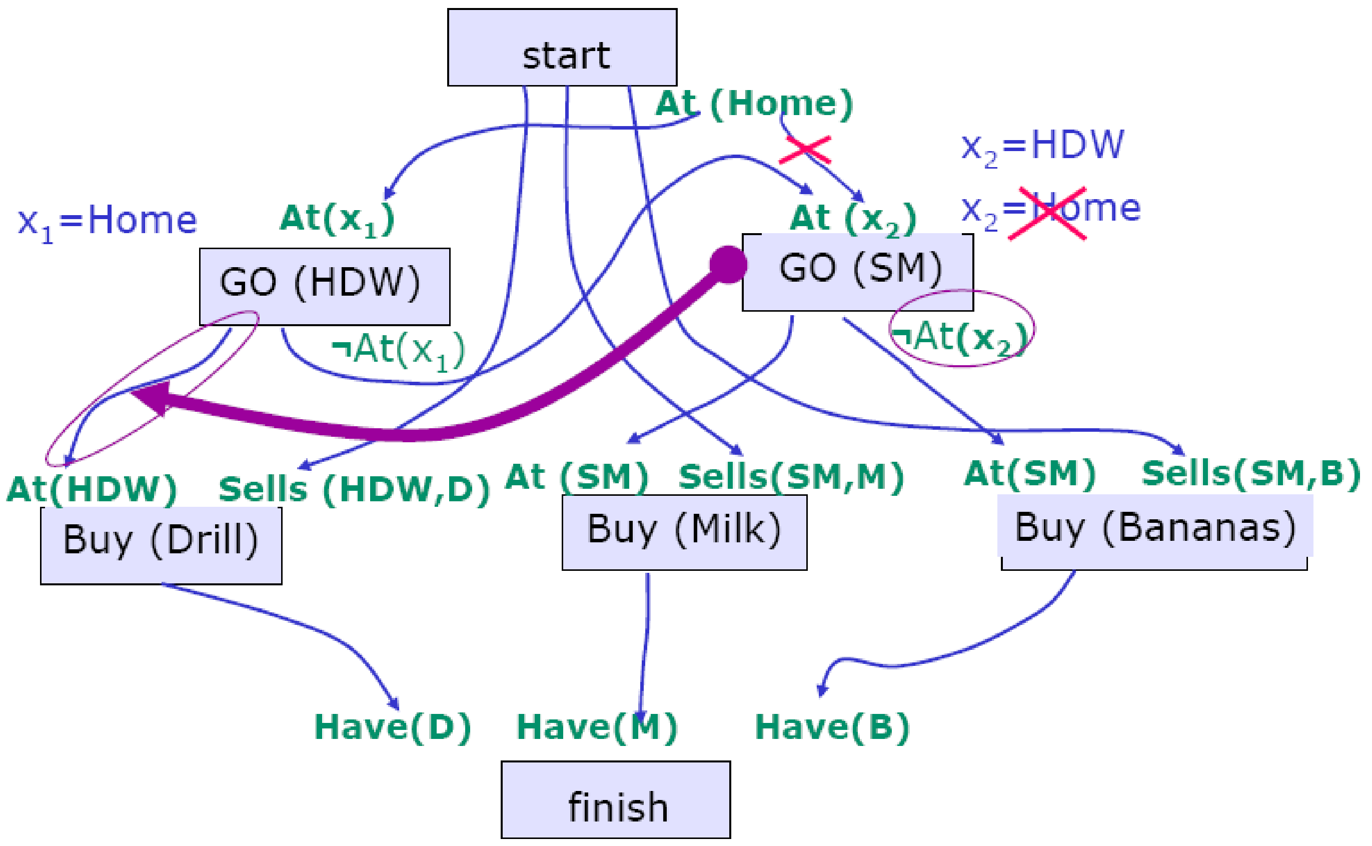
Now we constrain the variable x2 to be HDW rather than home. And this part of the plan seems to be in reasonably good shape.

Shop 'til You Drop!



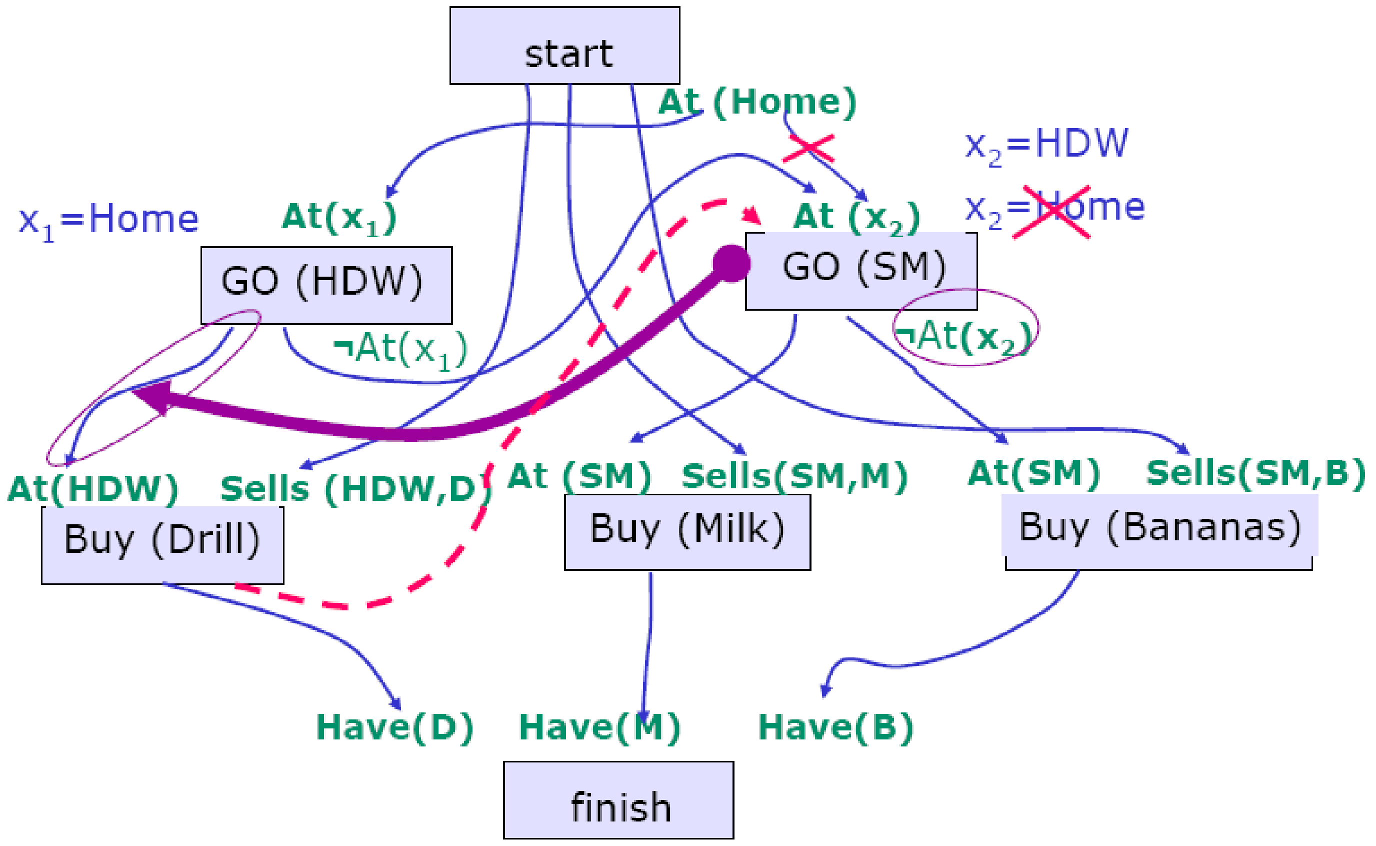
But we're not done yet! We still have some threats to take care of. There is currently nothing to keep us from going straight to the supermarket after we go to the hardware store, without buying our drill. More formally, the step Go(SM) threatens the causal link establishing AT(HDW), between Go(HDW) and Buy(Drill). This is because Go(SM) has not At(HDW) as an effect, and because there are no ordering constraints to keep it from coming between Go(HDW) and Buy(Drill).

Shop 'til You Drop!



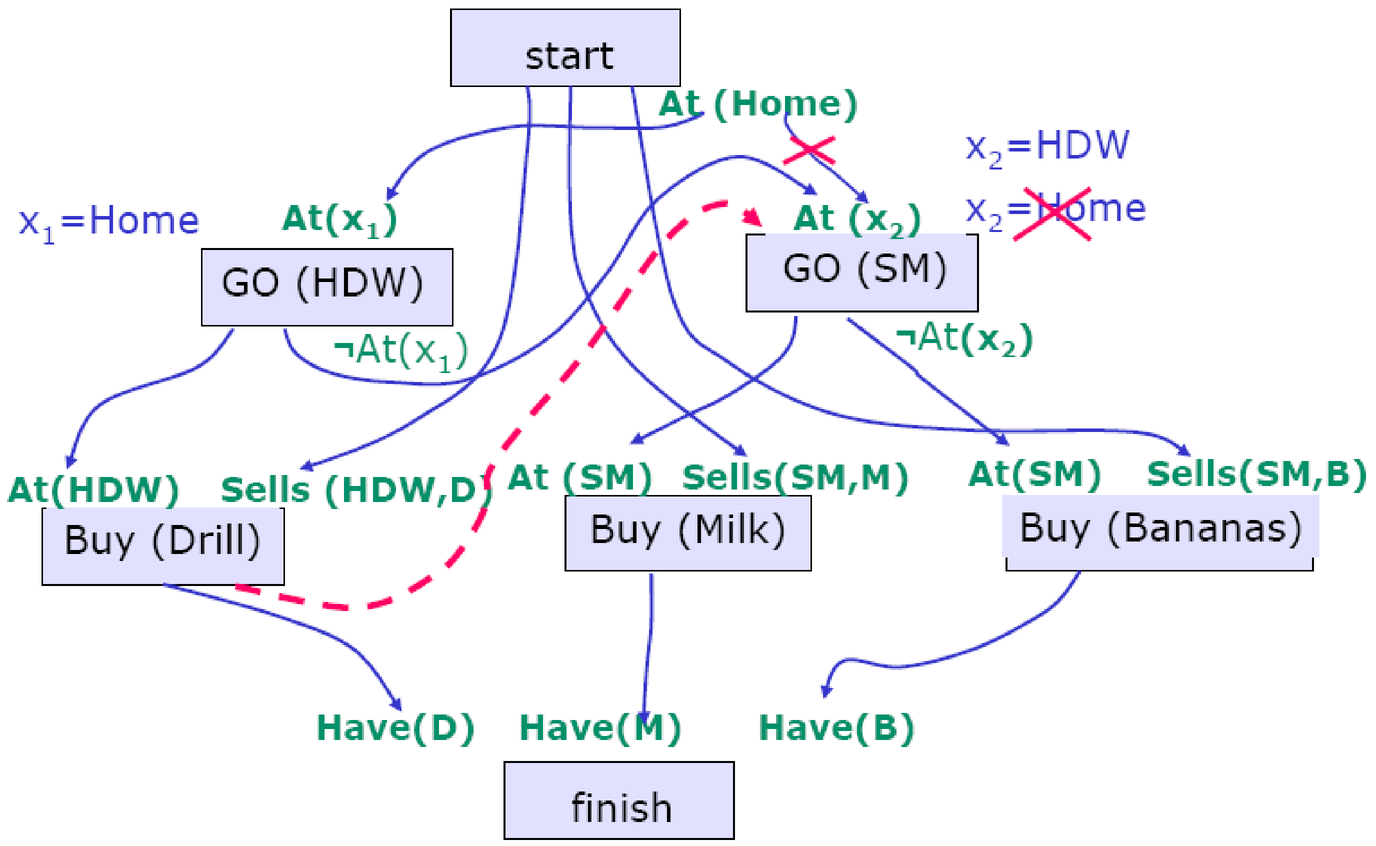
We could try to resolve the threat by putting Go(SM) before Go(HDW), but then we'd have an inconsistent set of temporal constraints (because there's already a causal link from Go(HDW) to Go(SM)).

Shop 'til You Drop!



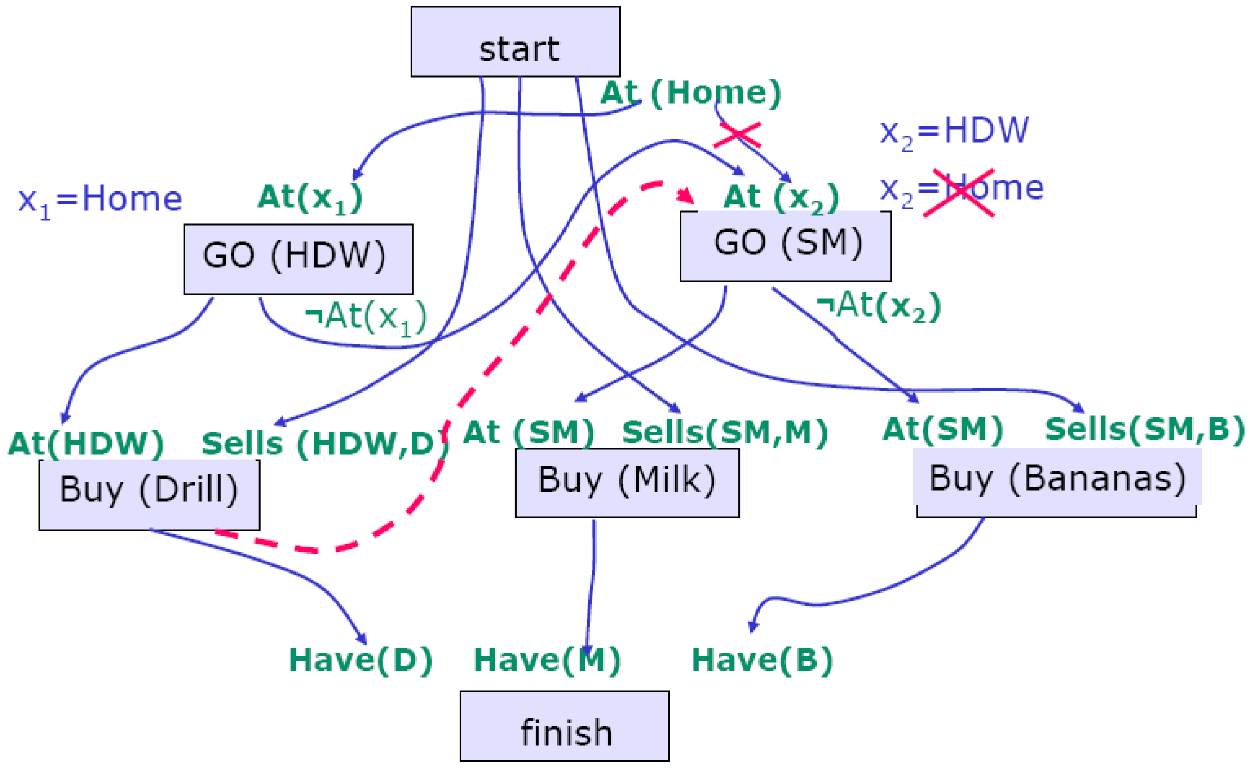
So, we're going to have to constrain Go(SM) to happen after Buy(Drill). We'll add a temporal constraint (the red dashed line) to make sure this happens.

Shop 'til You Drop!



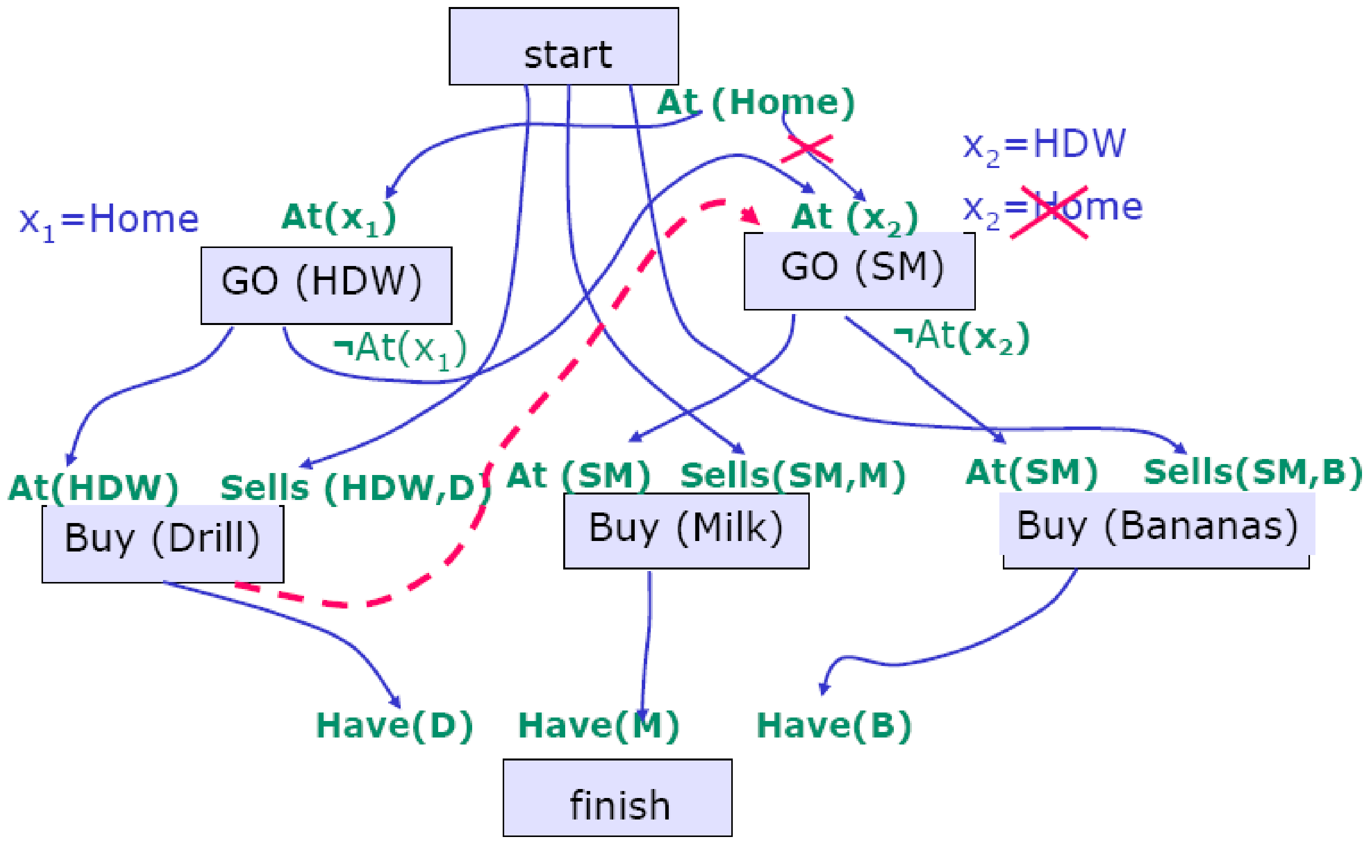
So, now we have a plan.

Shop 'til You Drop!



Let's look at this plan a little bit. It requires us to go to the hardware store first, and then to buy the drill. After that, we have to go to the supermarket. Once we're at the supermarket, though, we can buy milk and bananas in either order. So, we finished with a partially ordered plan. But the correctness theorem of the algorithm implies that if we finish with a partially ordered plan, then any ordering of the steps that's consistent with the ordering constraints of the plan will achieve the goal.

Shop 'til You Drop!



Note that this is not the only possible plan we could have come up with for this domain. If we had made some of our choices differently, we might have ended up with a plan that had us go to the supermarket before the hardware store. That would have been fine, too.