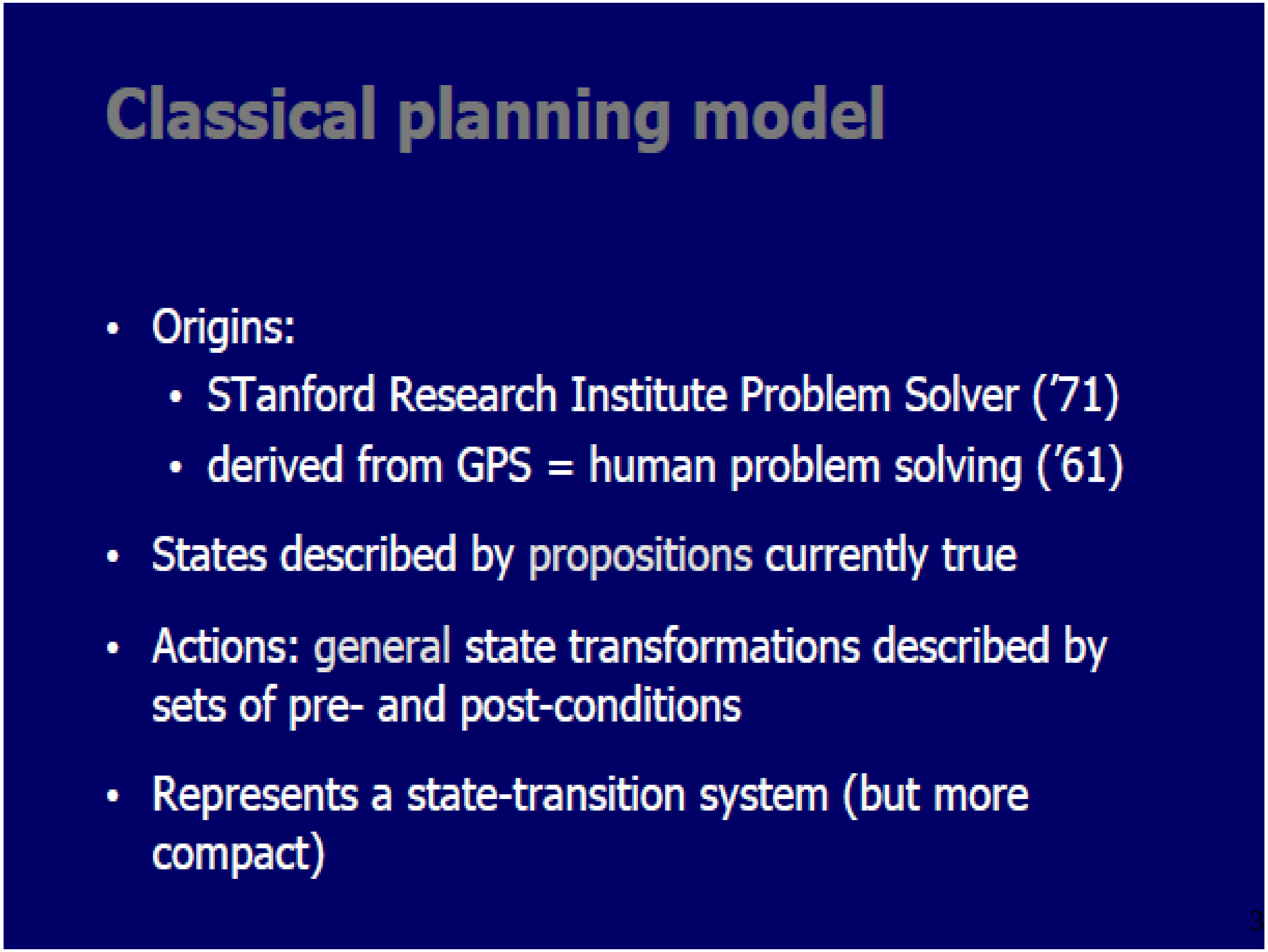
PLANNING

Planning problem

- How to get from the current state to your goal state?
- Planning involves
  - Action selection
  - Action sequencing
  - Resource handling
- Plans can be
  - Action sequences
  - Policies/strategies (action trees)



Classical planning model

- Origins:
  - STanford Research Institute Problem Solver ('71)
  - derived from GPS = human problem solving ('61)
- States described by propositions currently true
- Actions: general state transformations described by sets of pre- and post-conditions
- Represents a state-transition system (but more compact)

Classical Planning Representation

- Factored Representation
  - State : Collection of variables
  - Actions represented using PDDL (Planning Domain Definition Language)
  - PDDL
    - Initial State
    - Actions
    - Effect of action
    - Goal Test

Classical Planning Representation

- Factored Representation
  - State
    - Conjunction of fluents that are ground, functionless atoms
  - Actions
    - Represented using set of action schemas that defines Actions(s) and Result(s,a)
  - Action Schema
    - Set of ground actions
    - Lifted representation
    - Action(Fly(p, From,To))
    - Pre: At(p,from) Plane(P) Airport(From) Airport(To)
    - Effect: Not At(P, From) At(P, To)

Partial Order Planning

The Planning Problem

Formally a planning algorithm has three inputs:

1. A description of the world in some formal language,
2. A description of the agent's goal in some formal language, and
3. A description of the possible actions that can be performed.

   The planner's o/p is a sequence of actions which when executed in any world satisfying the initial state description will achieve the goal.

Representations for actions

Three components:

• the ***action description*** is what an agent actually returns to the environment in order to do something.

• the ***precondition*** is a conjunction of atoms (positive literals) that says what must be true before the operator can be applied.

• the ***effect*** of an operator is a conjunction of literals (positive or negative) that describes how the situation changes when the operator is applied.

Here's an example for the operator for going from one place to another:

Op(Action:Go(there), Precond:At(here)^Path(here, there), Effect:At(there)^ ¬At(here))

Search through the Space of Plans

An alternative is to search through space of plans rather than space of situations i.e plan-space node denote plans

- Start with a simple partial plan
- Expand the plan until the complete plan is developed
- Operators in this step:
  - Adding a step
  - Imposing an ordering that puts one step after another
  - Instantiating a previously unbound variable
- The solution → final plan   Path →

What is planning

1. We have some ***Operators***.
2. We have a ***Current state.***
3. We have a ***Goal State.***
4. We want to know:

    How to arrange the operators to reach the Goal State from Current State.

Classes of General-Purpose Planners

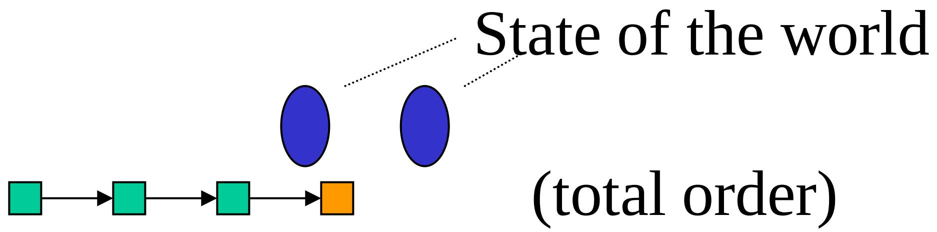
General purpose planners can be classified according to the space where the search is performed

State space

Plan space

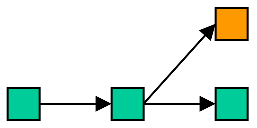
State- and Plan-Space Planning

- **State-space** planners transform the state of the world. These planners search for a sequence of transformations linking the starting state and a final state



(total order)

- **Plan-space** planners transform the plans. These planners search for a a plan satisfying certain conditions

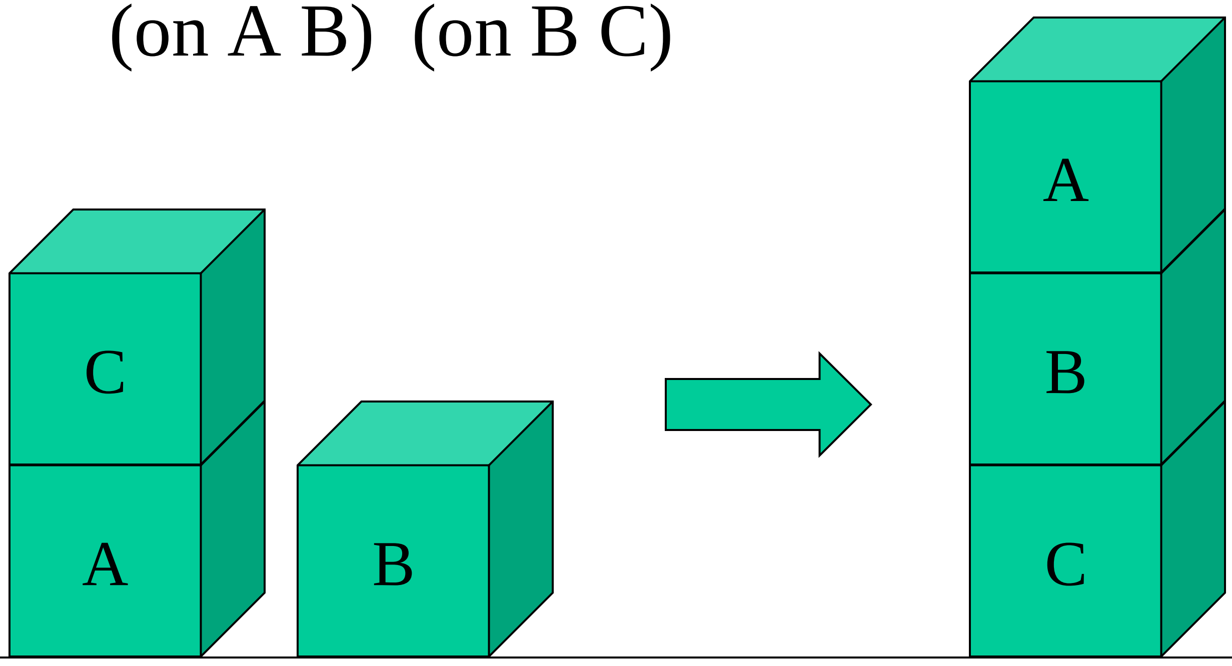


(partial-order, least-commitment)

Why Plan-Space Planning?

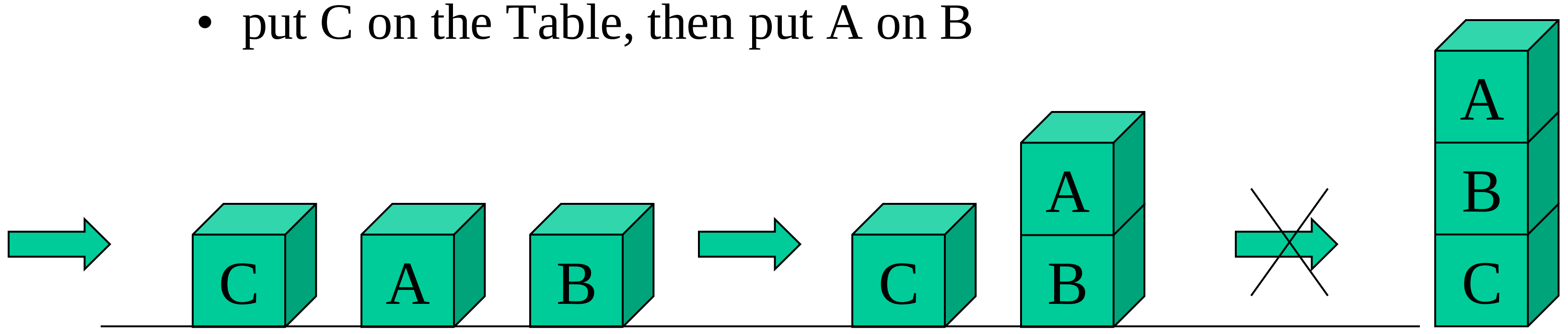
- Motivation: "Sussman Anomaly"
  - Two subgoals to achieve:

    (on A B) (on B C)



Why Plan-Space Planning?

- Problem of state-space search:
  - Try (on A B) first:
    - put C on the Table, then put A on B



    - Accidentally wind up with A on B when B is still on the Table
    - We can not get B on C without taking A off B
    - Try to solve the first subgoal first appears to be mistaken

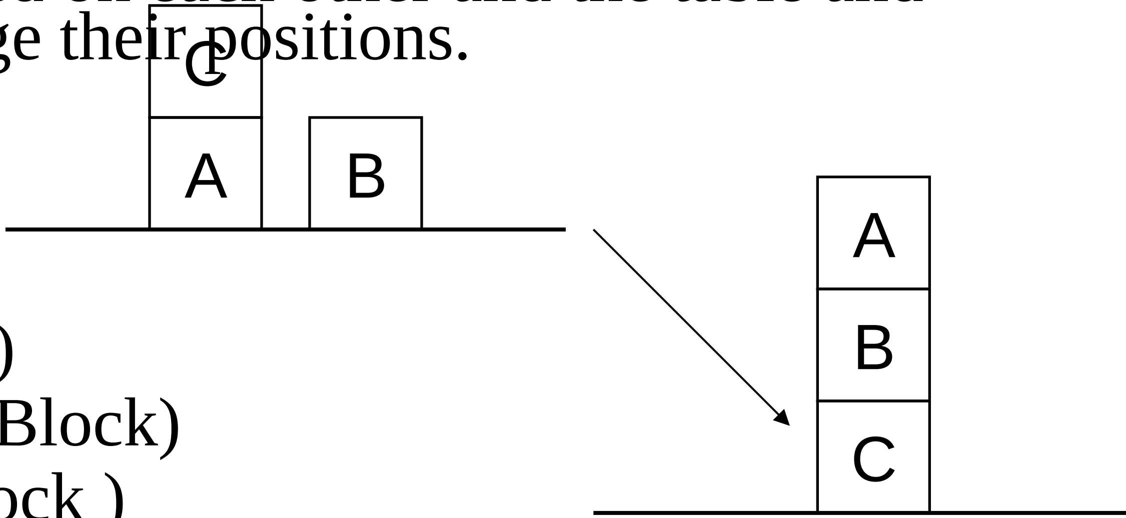
Air Cargo Transfer Example

- What's in Domain:
  - We have a set of airports as SFO,JFK, ...
  - We have a set of cargos as C1, C2, …
  - We have some airplanes as P1, P2, …
- State:
  - Plans and Cargos are at specific airports and we want to change the positions.

- Actions:
  - Load (*Cargo, Plane, Airport)*
  - Fly (*Plane, Airport1, Airport2)*
  - *Unload (Cargo, Plane, Airport)*

Blocks World Example

- Domain Objects:
  - A set of blocks and a table.
- States:
  - Blocks are stacked on each other and the table and we want to change their positions.
- Actions:
  - PickUp(Block),
  - PutDown(Block)
  - Unstack(Block, Block)
  - Stack(Block, Block)



Domain Definition Example 1

AIR CARGO TRANPORT DOMAIN:

- Objects:
  - SFO , JFK, C1 , C2 , P1 , P2.
- Predicates:
  - At(C1, SFO)
  - In(C1, P2)
  - Plane(P1)
  - Cargo(C1)
- Actions:
  - …

Domain Definition Ex.1(cont.)

- Actions:
  - Name,
  - Parameters,
  - Preconditions,
  - Effects.
  - **LOAD(c, p , a)**
    - Prec.:

      At(c,a), At(p,a), Cargo(c),Plane(p),Airport(a).
    - Effects:

      ~At(c,a), In(c,p)

Domain Definition Ex.1 (cont.)

- Actions:
  - **UNLOAD(c, p , a)**
    - Prec.:
      In(c,p), At(p,a), Cargo(c),Plane(p),Airport(a).
    - Effects:
      At(c,a), ~In(c,p)
  - **FLY(p , a1, a2)**
    - Prec.:
      At(p,a1) Plane(p),Airport(a1) ,Airport(a2).
    - Effects:
      ~At(p,a1), At(p,a2)

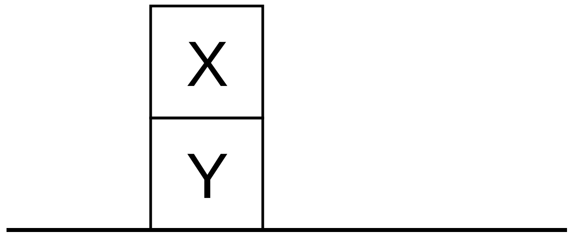
Domain Definition Example 2

BLOCKS WORLD DOMAIN

- Objects:
  - A, B, C, … (the blocks) & the ROBOT.

- Predicates:
  - On(x , y).
  - OnTable(x).
  - Holding(x).
  - HandEmpty.
  - Clear(x).
  - OnTable(x).
  - Block(x).

Domain Definition Ex.2(cont.)



- Actions:
  - **UnStack(x , y):**
    - Prec:
      On(x,y), HandEmpty, Clear(x).
    - Effects:
      ~On(x,y), Holding(x), ~Clear(x), Clear(y).
  - **Stack(x,y):**
    - Prec:
      Holding(x), Clear(y)
    - Effects:
      On(x,y), HandEmpty, ~Holding(x), Clear(x), ~Clear(y).

NOTE: Nothing about what is block and what's not.

Domain Definition Ex.2(cont.)

- Actions:
  - **PickUp(x):**
    - Prec:
      HandEmpty, Clear(x), OnTable(x).
    - Effects:
      Holding(x), ~Clear(x), ~OnTable(x).
  - **PutDown(x):**
    - Prec:
      Holding(x).
    - Effects:
      OnTable(x), HandEmpty, Clear(x).

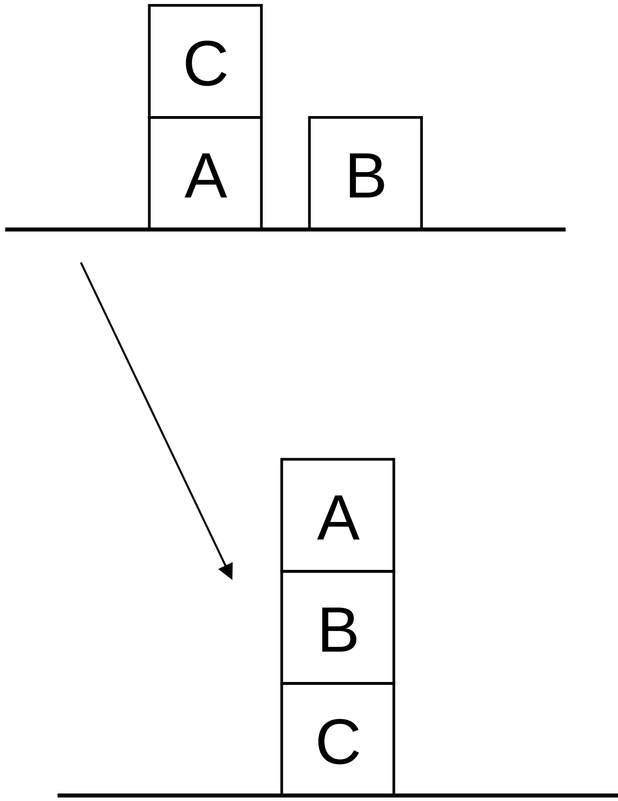
Propblem Definition Ex.2

- PROBLEM DEFINITION:
  - Initial State:

    On(C,A), Clear(C), ~Clear(A), OnTable(A), Clear(B), OnTable(B), HandEmpty.

  - Goal State:

    HandEmpty,
    Clear(A),
    On(A,B),
    On(B,C),
    OnTable(C).



Simplest Approach

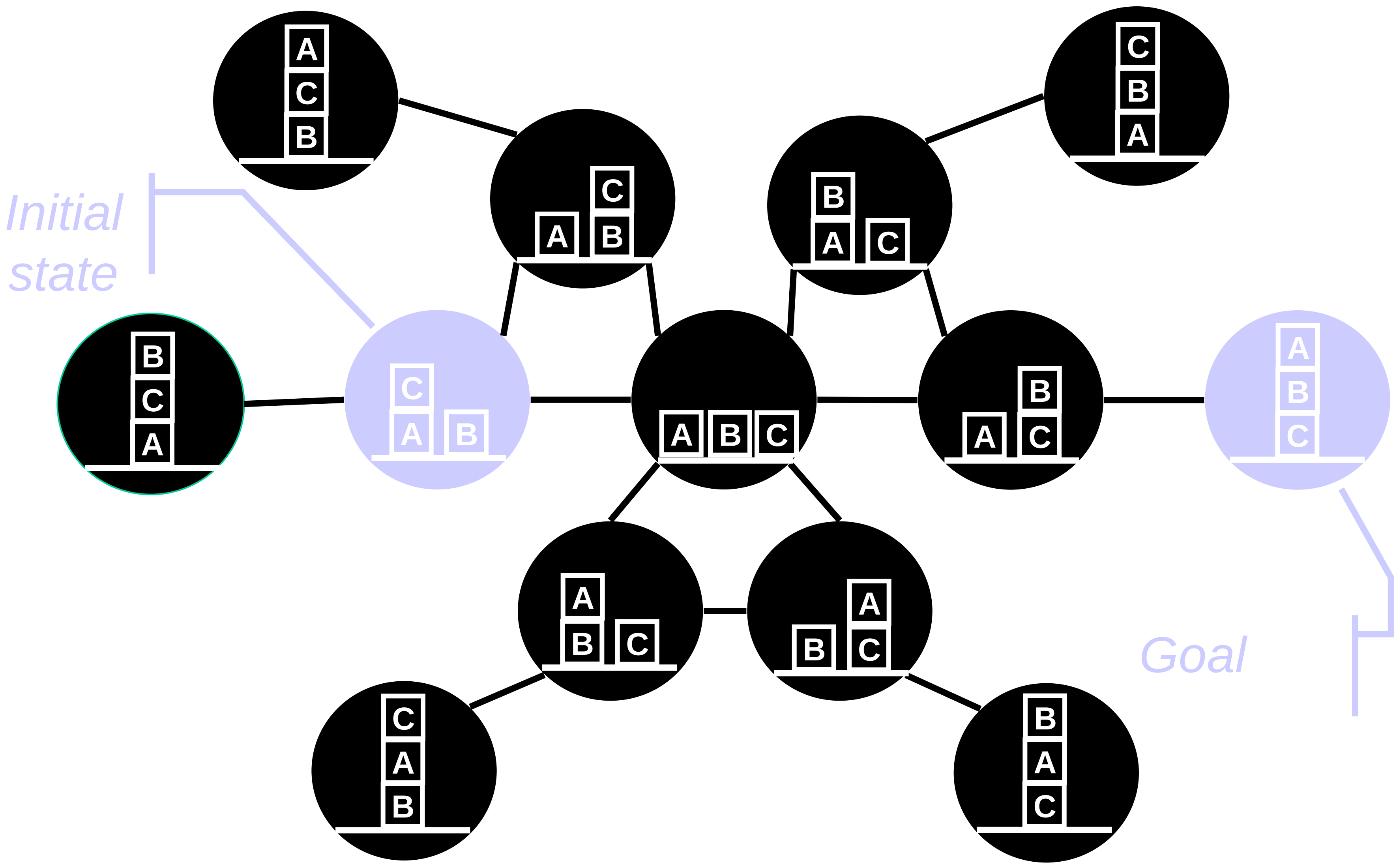
- It's all about SEARCH.
  - States:

    As described before.
  - Next State Generator:

    Which actions are applicable, apply every one of em.
  - Path Cost:

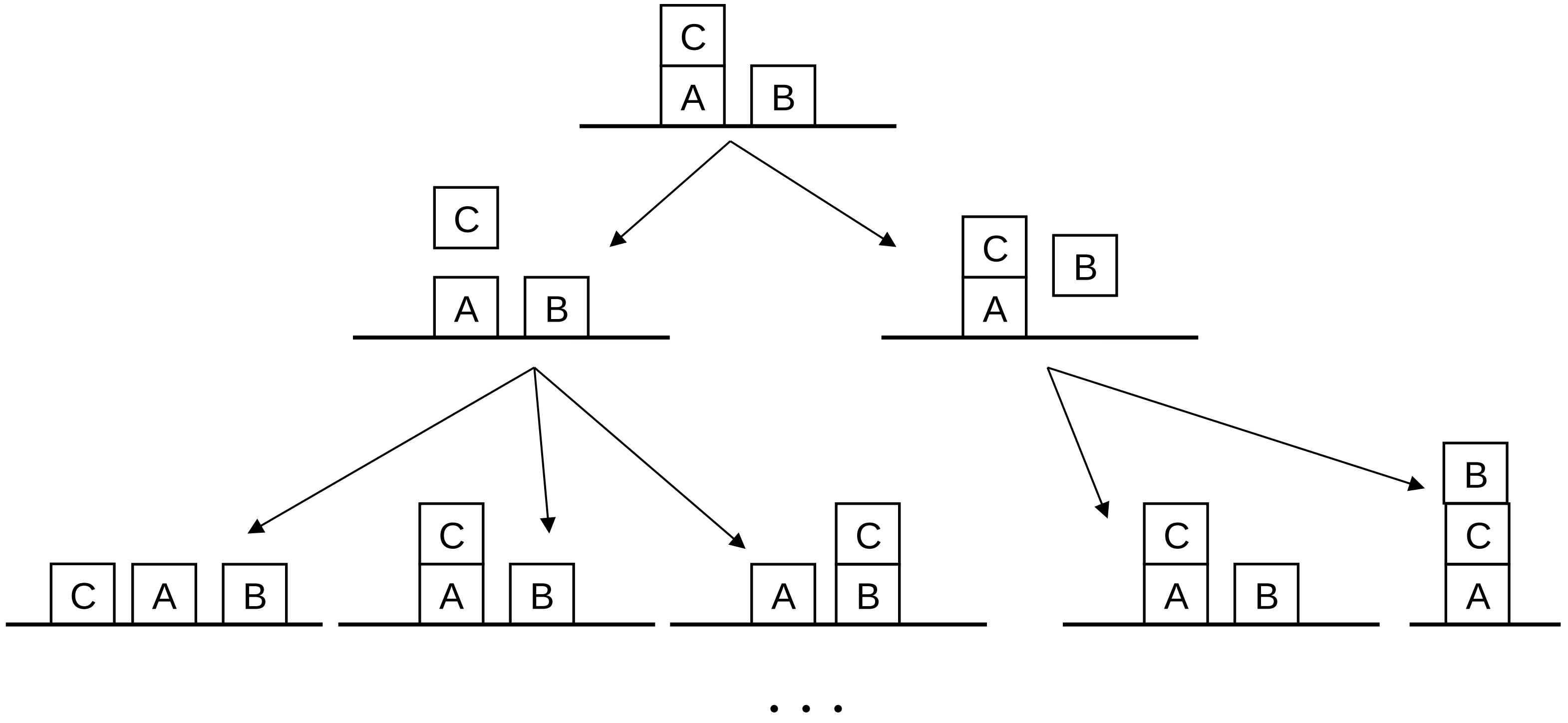
    One for each action.
  - Goal Test:

    Has goal state reached?


Initial state
Goal
A C B
C B A
A B C
B A C
B C A
C A B
A B C
B A C
C A B
B A C
A B C
B C A
A B C
B A C
C A B
B A C

Simplest Approach

- Progression (Forward Search):
  - Start from Initial State, move forward till you reach goal state.



NOTE: Backtracking is mandatory.

Progression

- Progression
  - Initial state:
    - Initial state of the planning problem
  - Actions:
    - Applicable to the current state (actions’ preconditions are satisfied)
  - Goal test:
    - Whether the state satisfies the goal of the planning
  - Step cost:
    - Each action is 1

Progression

- Progression
  - A plan is a sequence of STRIPS operators
  - From initial state, search forward by selecting operators whose preconditions can be unified with iterals in the state
  - New state includes positive literals of effect; the negated literals of effect are deleted
  - Search forward until goal unifies with resulting state
  -

Regression

- Regression
  - A plan is a sequence of STRIPS operators
  - The goal state must unify with at least one of the positive literals in the operator’s effect
  - Its preconditions must hold in the previous situation, and these become subgoals which might be satisfied by the initial conditions
  - Perform backward chaining from goal

Example- Changing Flat tire

- Consider the problem of changing a flat tire.

- More precisely, the goal is to have a good spare tire properly mounted onto the car’s axle, where the initial state has a flat tire on the axle and a good spare tire in the trunk. There are just four actions: removing the spare from the trunk, removing the flat tire from the axle, putting the spare on the axle, and leaving the car unattended overnight. We assume that the car is in a particularly bad neighborhood, so that the effect of leaving it overnight is that the tires disappear.

$Init(At(Flat, Axle) \land At(Spare, Trunk))$

$Goal(At(Spare, Axle))$

$Action(Remove(Spare, Trunk),$
  PRECOND: $At(Spare, Trunk)$
  EFFECT: $\neg At(Spare, Trunk) \land At(Spare, Ground))$

$Action(Remove(Flat, Axle),$
  PRECOND: $At(Flat, Axle)$
  EFFECT: $\neg At(Flat, Axle) \land At(Flat, Ground))$

$Action(PutOn(Spare, Axle),$
  PRECOND: $At(Spare, Ground) \land \neg At(Flat, Axle)$
  EFFECT: $\neg At(Spare, Ground) \land At(Spare, Axle))$

$Action(LeaveOvernight,$
  PRECOND:
  EFFECT: $\neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk)$
    $\land \neg At(Flat, Ground) \land \neg At(Flat, Axle))$

- The sequence of events is as follows:

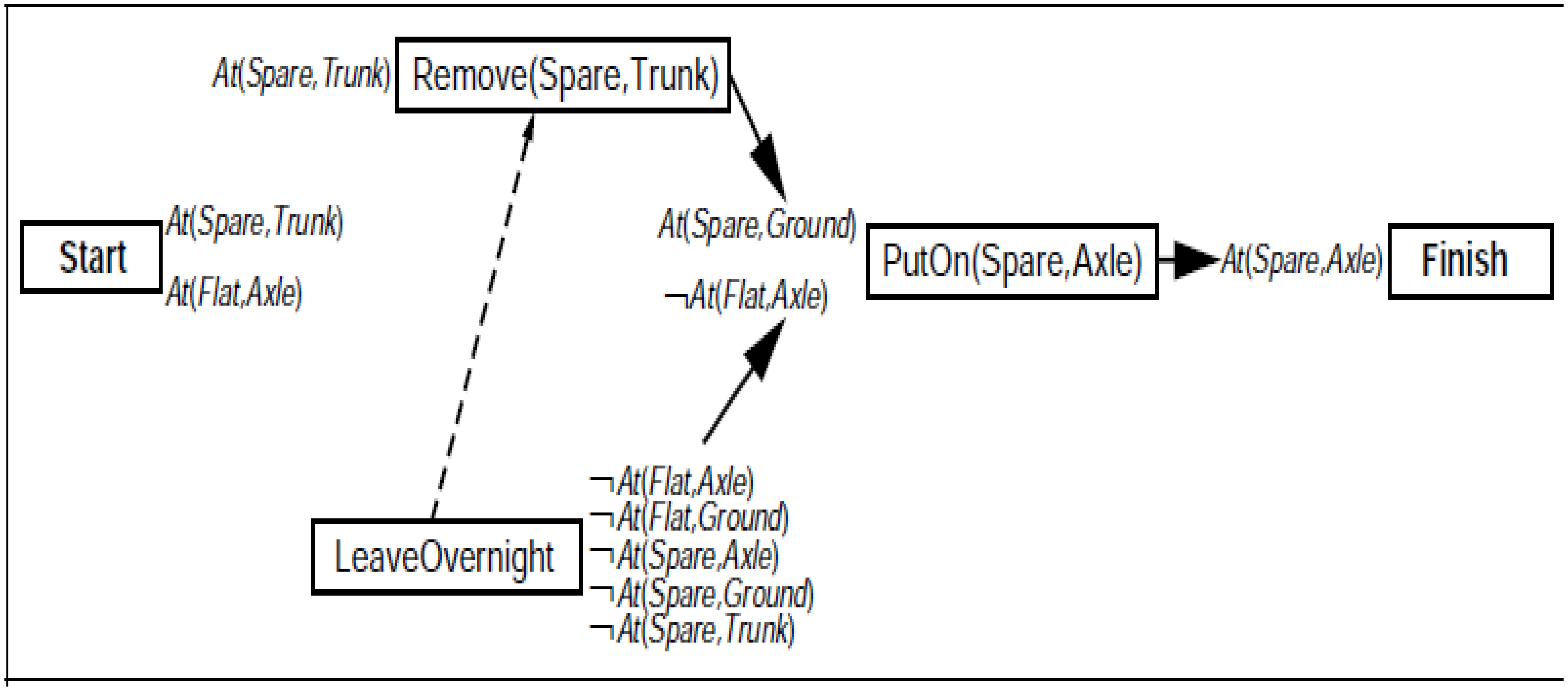
- 1. Pick the only open precondition, At (Spare; Axle) of Finish. Choose the only applicable action, PutOn(Spare; Axle).
- A precondition is open if it is not achieved by some action in the plan.

- 2. Pick the At (Spare; Ground) precondition of PutOn(Spare; Axle). Choose the only applicable action, Remove(Spare;Trunk) to achieve it.

3. Pick the ~At (Flat; Axle) precondition of PutOn(Spare; Axle). Just to be contrary, choose the LeaveOvernight action rather than the Remove(Flat; Axle) action.

LeaveOvernight also has the effect ~At (Spare; Ground), which means it conflicts with the causal link

To $Remove(Spare, Trunk) \xrightarrow{At(Spare, Ground)} PutOn(Spare, Axle)$ tting LeaveOvernight before Remove(Spare;Trunk).



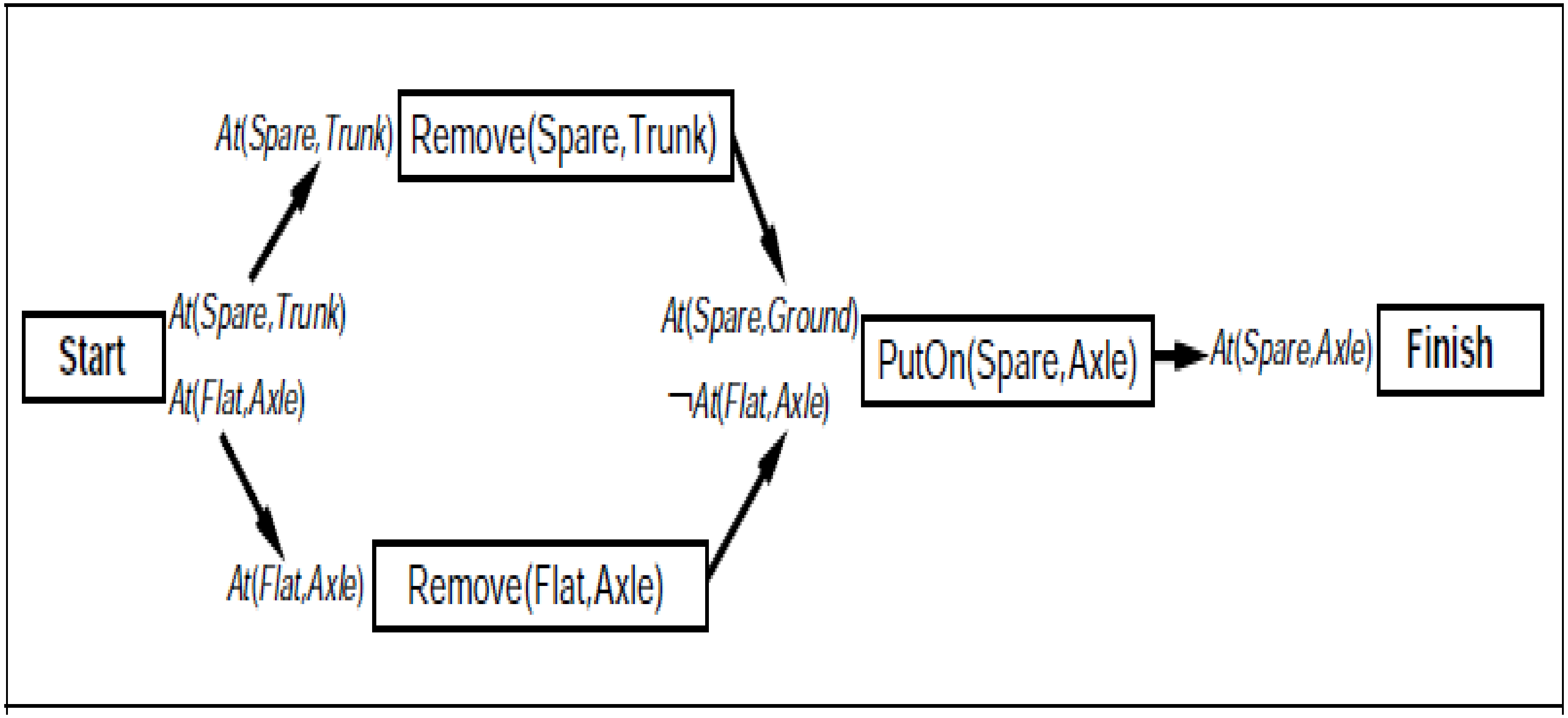
- The plan after choosing LeaveOvernight as the action for achieving ~At(Flat; Axle). To avoid a conflict with the causal link from Remove(Spare;Trunk) that protects At(Spare; Ground), LeaveOvernight is constrained to occur before Remove(Spare;Trunk), as shown by the dashed arrow.

- The only remaining open precondition at this point is the At (Spare;Trunk) precondition of the action Remove(Spare;Trunk).

- The only action that can achieve it is the existing Start action, but the causal link from Start to Remove(Spare;Trunk) is in conflict with the ~At (Spare;Trunk) effect of LeaveOvernight.

- In essence, the planner has proved that LeaveOvernight doesn’t work as a way to change a tire.

Final solution to the tire problem



- Remove(Spare;Trunk) and Remove(Flat; Axle) can be done in either order, as long as they are completed before the PutOn(Spare; Axle) action.

Representation of Plans

Partial Order Planner –
A planner that can represent plans in which some steps are ordered (before or after) w.r.t each other and other steps are unordered.

Any planning algorithm that can place two actions into a plan without specifying which PARTIALORDER comes first is called a partial-order planner.

Consider a simple problem:

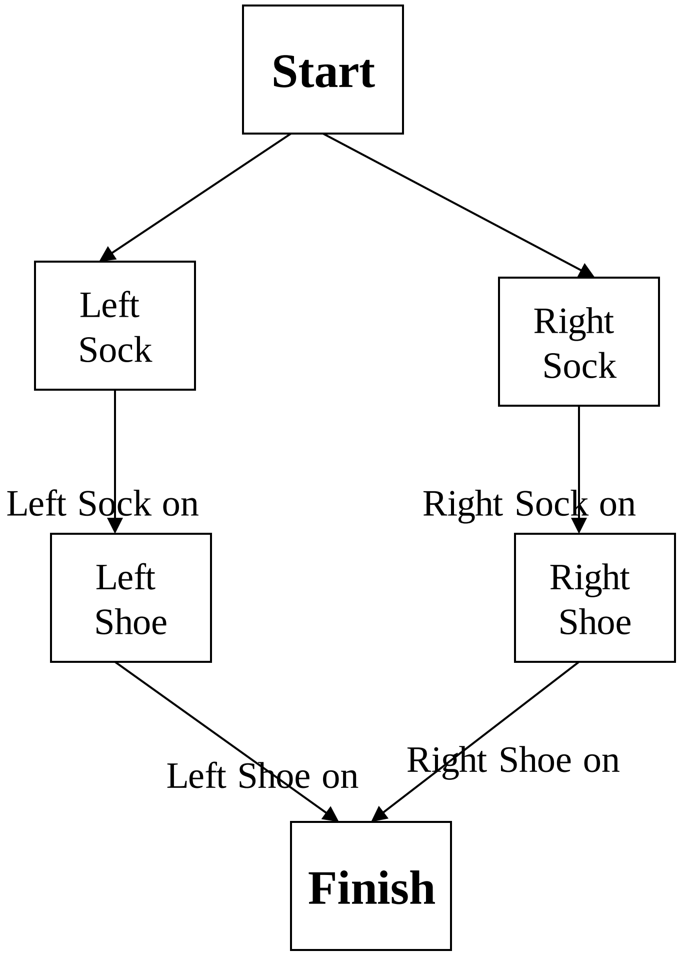
- •Putting on a pair of shoes

•Goal →RightShoeOn ^ LeftShoeOn

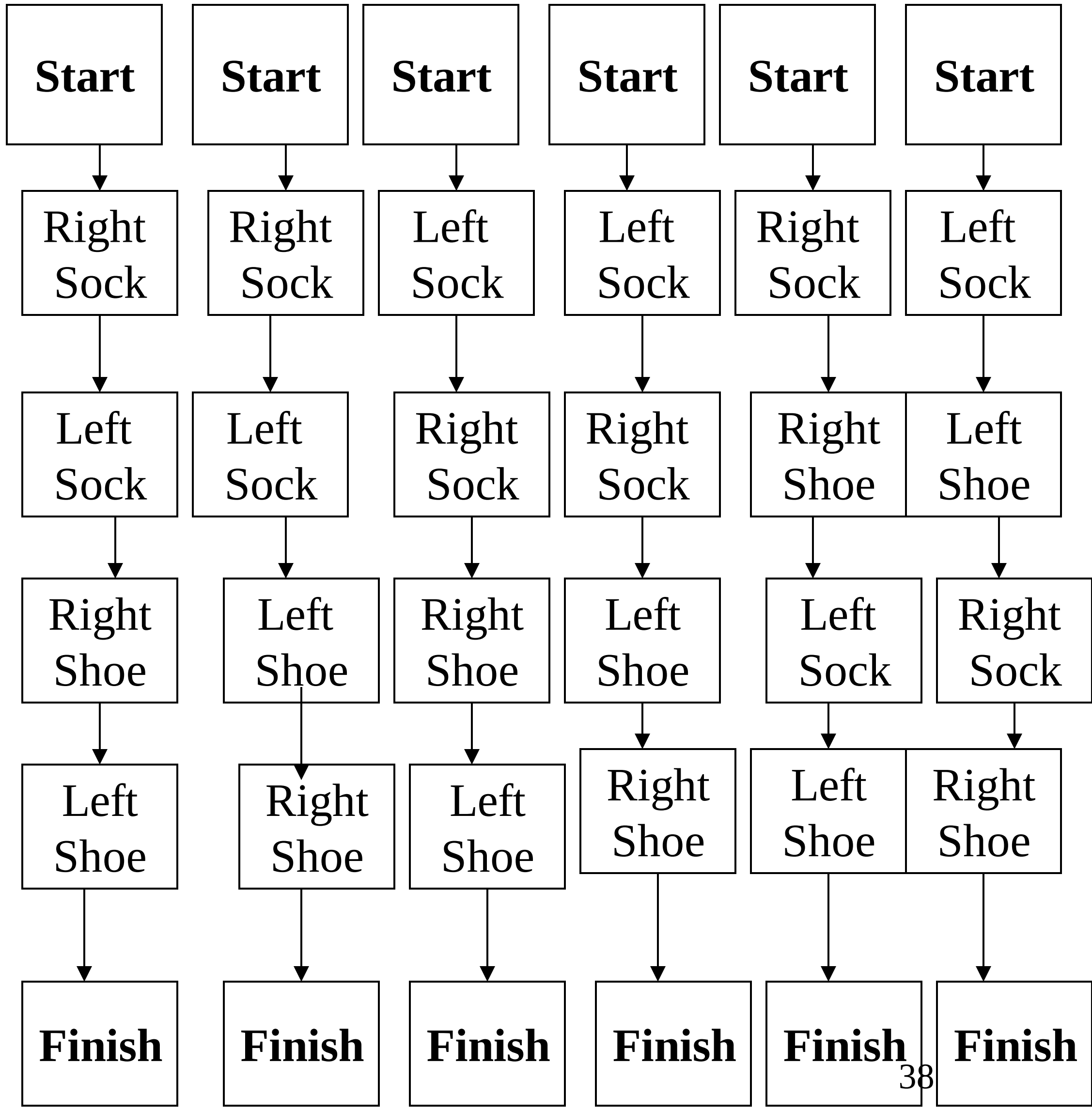
•Four operators:

- •**Op(Action:RightShoe,PreCond:RightSockOn,Effect:RightShoeON)**
- •**Op(Action:RightSock , Effect: RightSockOn)**
- •**Op(Action:LeftShoe, Precond:LeftSockOn, Effect:LeftShoeOn)**
- •**Op(Action:LeftSock,Effect:LeftSockOn)**

Partial Order Plans:



Total Order Plans:



Plans,Causal Links and Threats

Representing a Plan as a three tuple :-- <A,O,L>

- A → Set of Actions
- L → Set of Causal Links
- O → Set of Ordering constraints over A

If A ={A1,A2,A3} then O might be set{A1<A3,A2<A3}

These constraints specify a plan in which A3 is necessarily the last action but does not commit to a choice of which of the three actions comes first .

As least commitment planners refine their plans, they must do constraint satisfaction to ensure consistency of O

- The successor function arbitrarily picks one open precondition *p* on an action B
- *A causal link between two actions A and B in the plan is written as A B and is read $\xrightarrow{p}$ as “A achieves p for B.”*
  - It generates a successor plan adding the causal link A to B and the ordering constraint A $\prec$ B
  - If A was not in the plan, it adds Start $\prec$ A and A $\prec$ Finish
  - It resolves all conflicts between
    - the new causal link and all existing actions
    - between A and all existing causal links
  - Then it adds the successor states for combination of resolved conflicts
- It repeats until no open precondition exists

Causal Links—

A Causal Links is written as $S_i \xrightarrow{c} S_j$ and read as “ S_i achieves c for S_j “.Causal Links serve to record the purpose of steps in the plan:here a purpose of S_i is to achieve the precondition c of S_j

Threat –

Causal Links are used to detect when a newly introduced action interferes with past decision. Such an action is called a Threat.

Resolving Threats

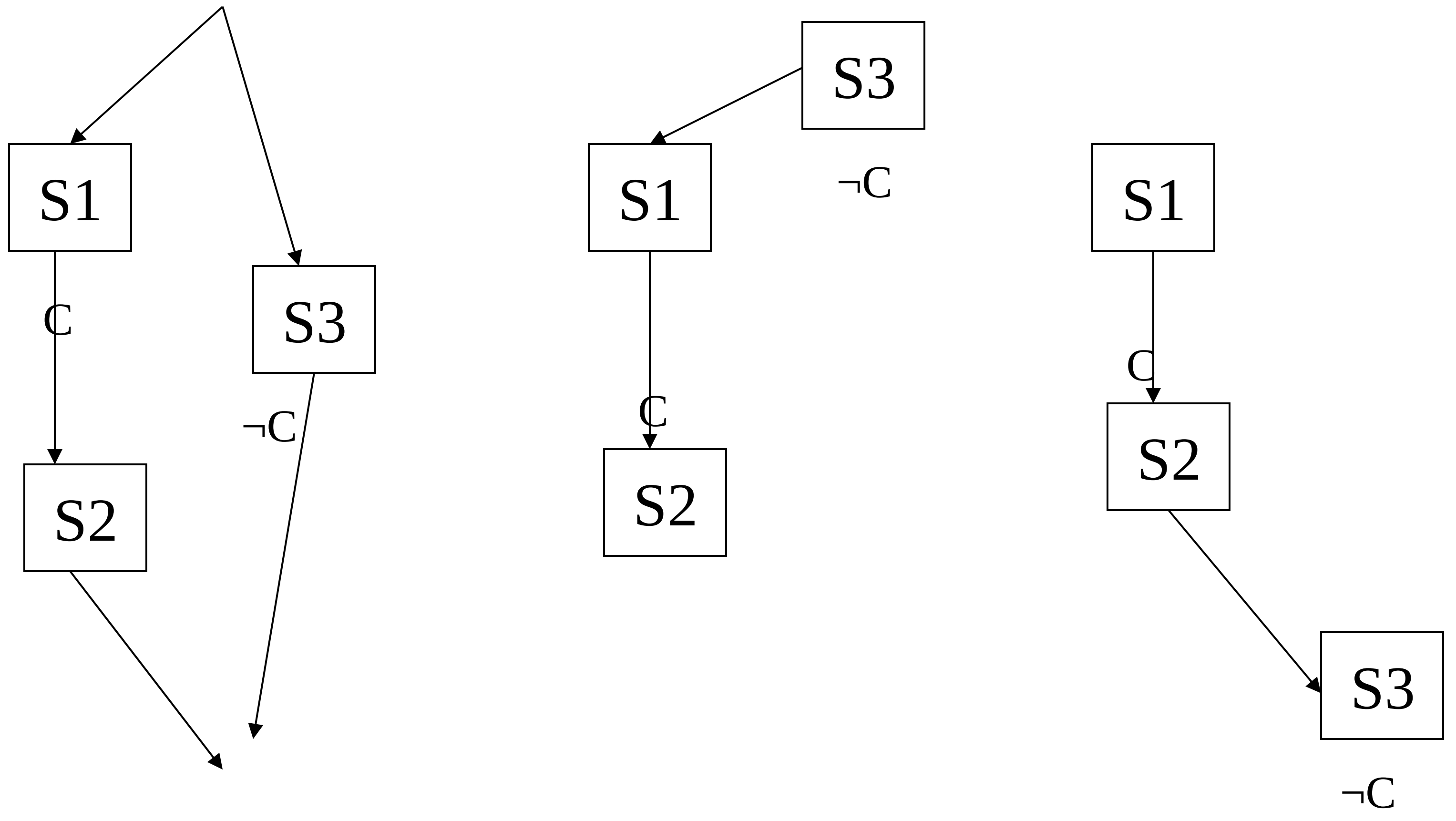
When a plan contains a threat , then there is a danger that the plan won't work as anticipated i.e. We have reached a dead-end in the search.

The causal link S1 $^{c}\rightarrow$ S2 is threatened by a new step S3 because one effect of S3 is to delete c.

The way to resolve the Threat is to add ordering constraints to make sure that S3 does not intervene between S1 and S2

Demotion -- S3 is placed before S1 .

Promotion – S3 is placed after S2.


S1
C
S2
S3
¬C
S3
¬C
S1
C
S2
S1
C
S2
S3
¬C

- **Detection:** for each causal link ⟨*s* -> *t* ⟩ and for each step *r*, check positive and negative threats
- **Threat removal**:
- (i) demotion, (ii) promotion and (iii) separation

(i) **Demotion**: order *r* before *s*

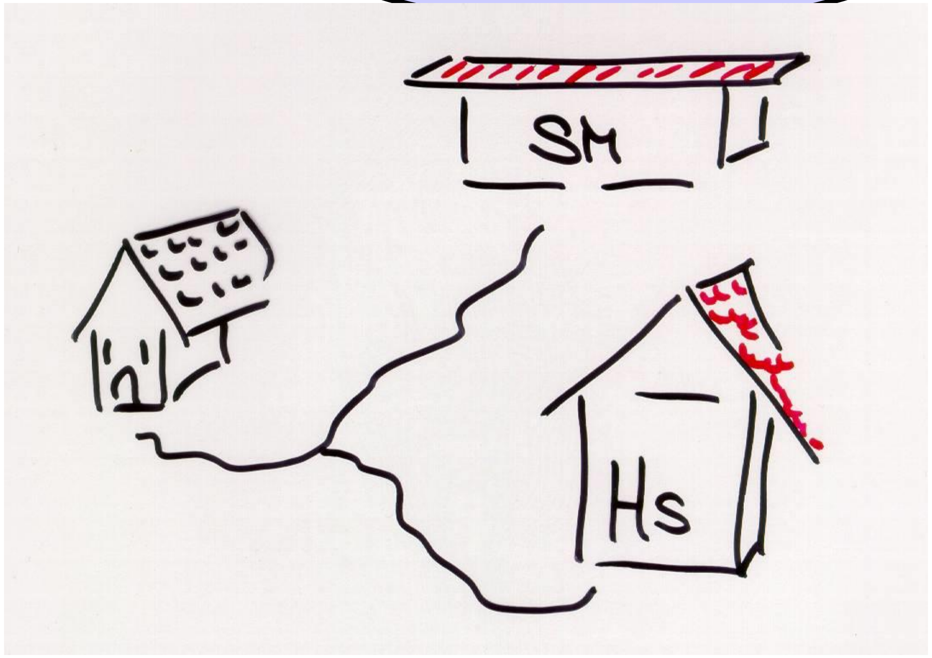
(ii) **Promotion:** order *r* after *t*

(iii) **Separation**: Introduce a variable-binding constraint. This is a process of binding variables in a way that they do not assume values which lead to disruption. For example On(B,C) should not happen when C is not resting on table (for our goal).

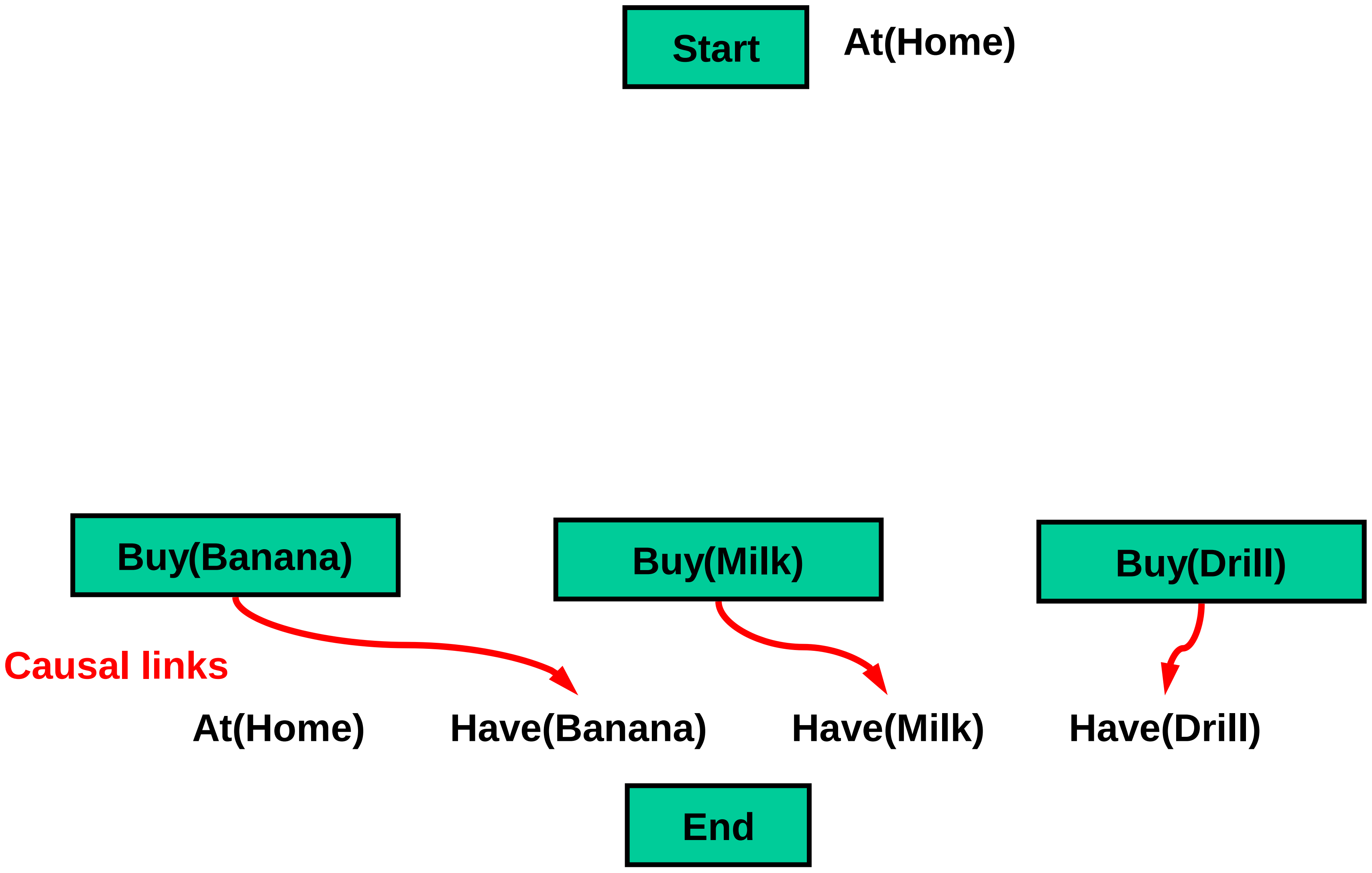
Example

SM: Super Market
HS: Hardware Store

- At(Home)
- Sells(SM, Banana)
- Sells(HS, Drill)
- Have(Drill)
- Have(Milk)
- Have(Banana)

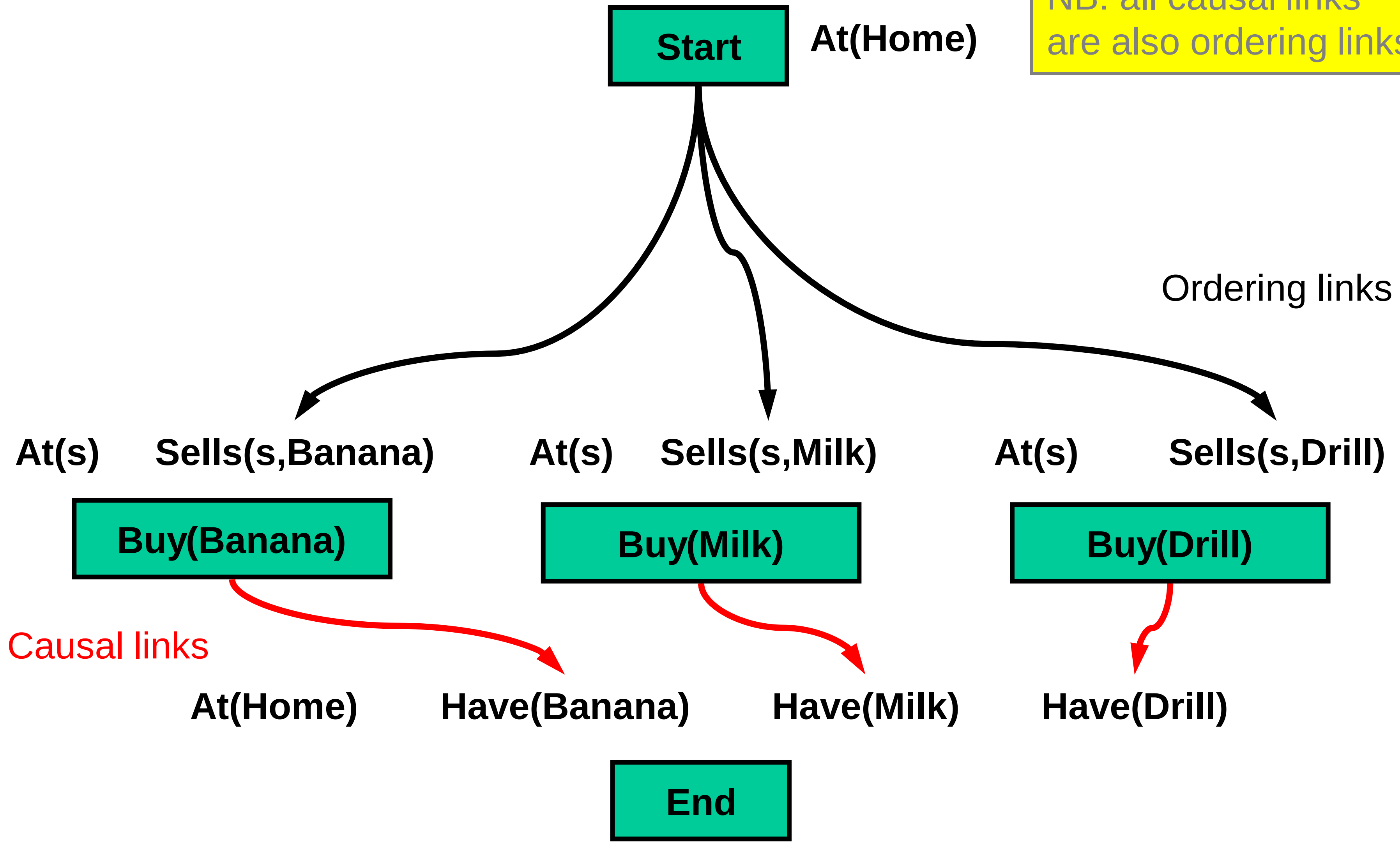


Partial Order Planning



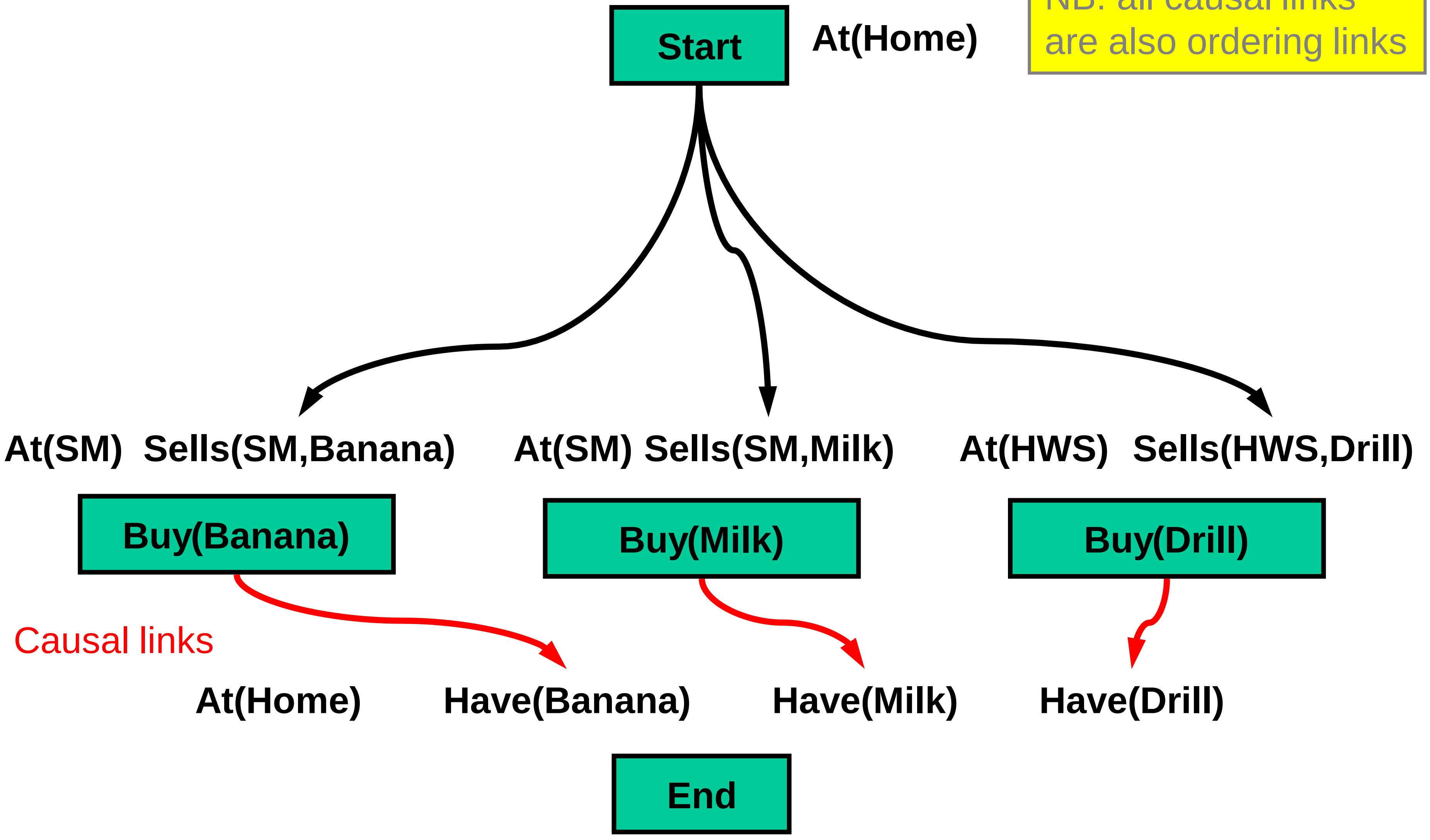
Refining the Plan (1)

NB: all causal links are also ordering links

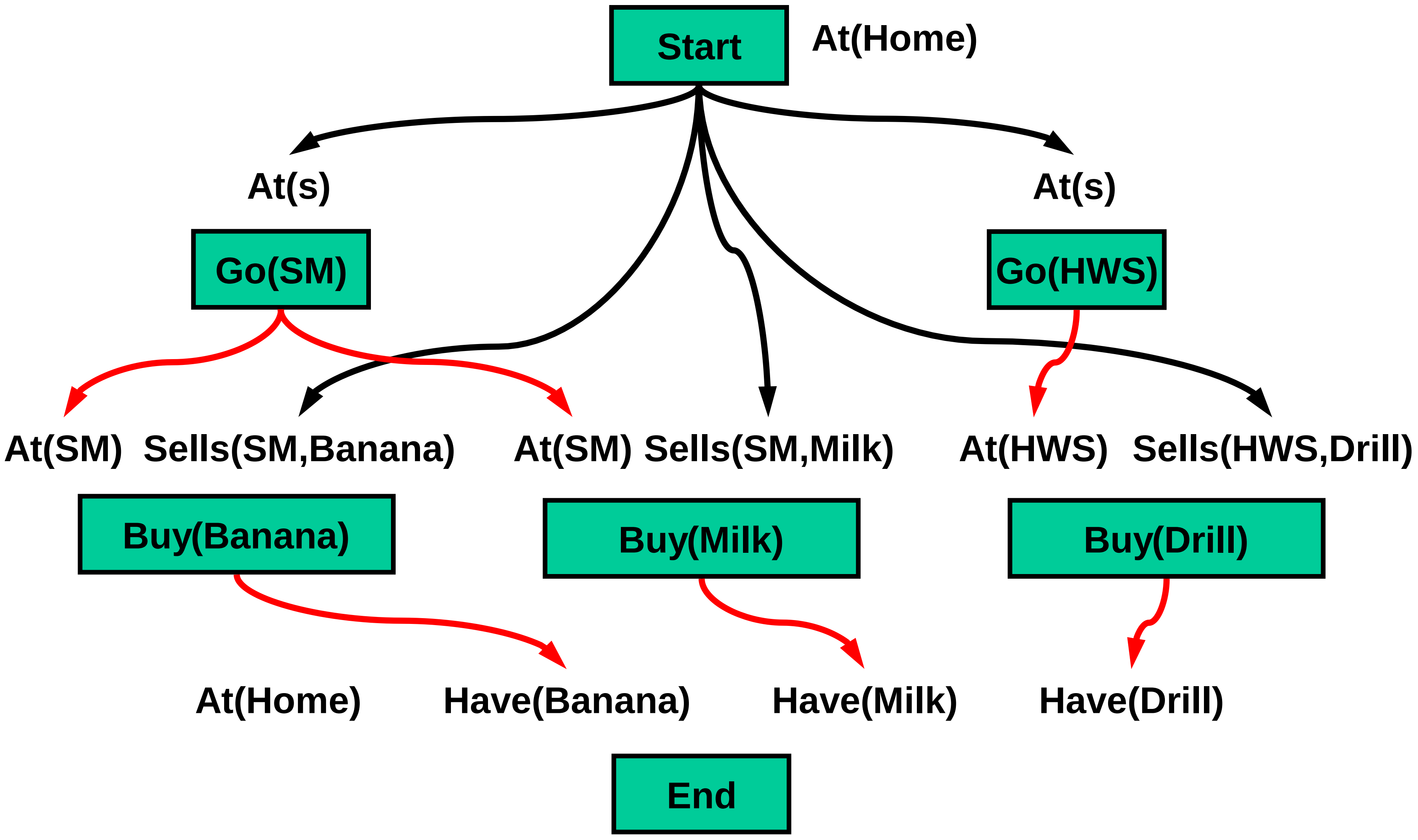


Refining the Plan (2)

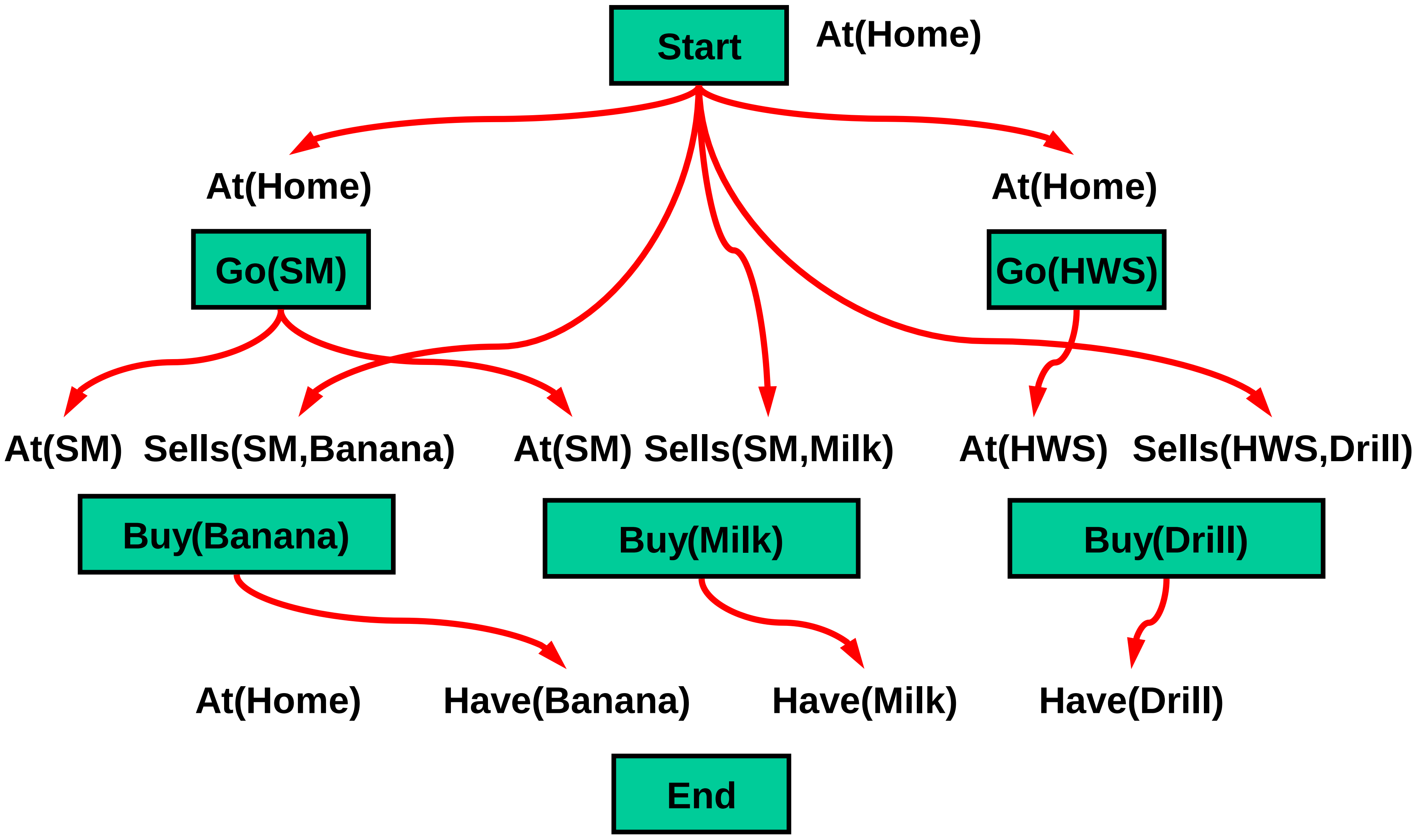
NB: all causal links are also ordering links



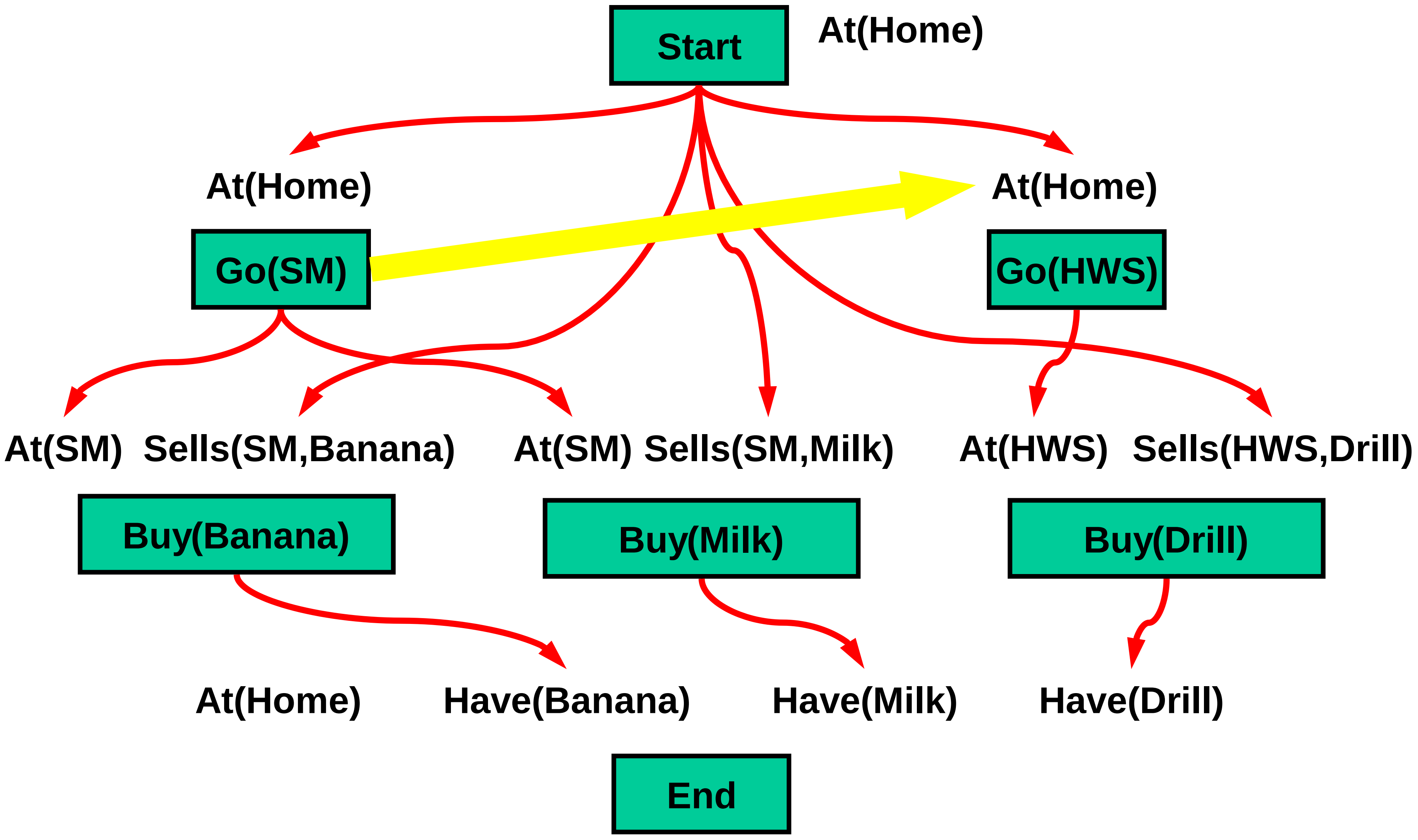
Refining the Plan (3)



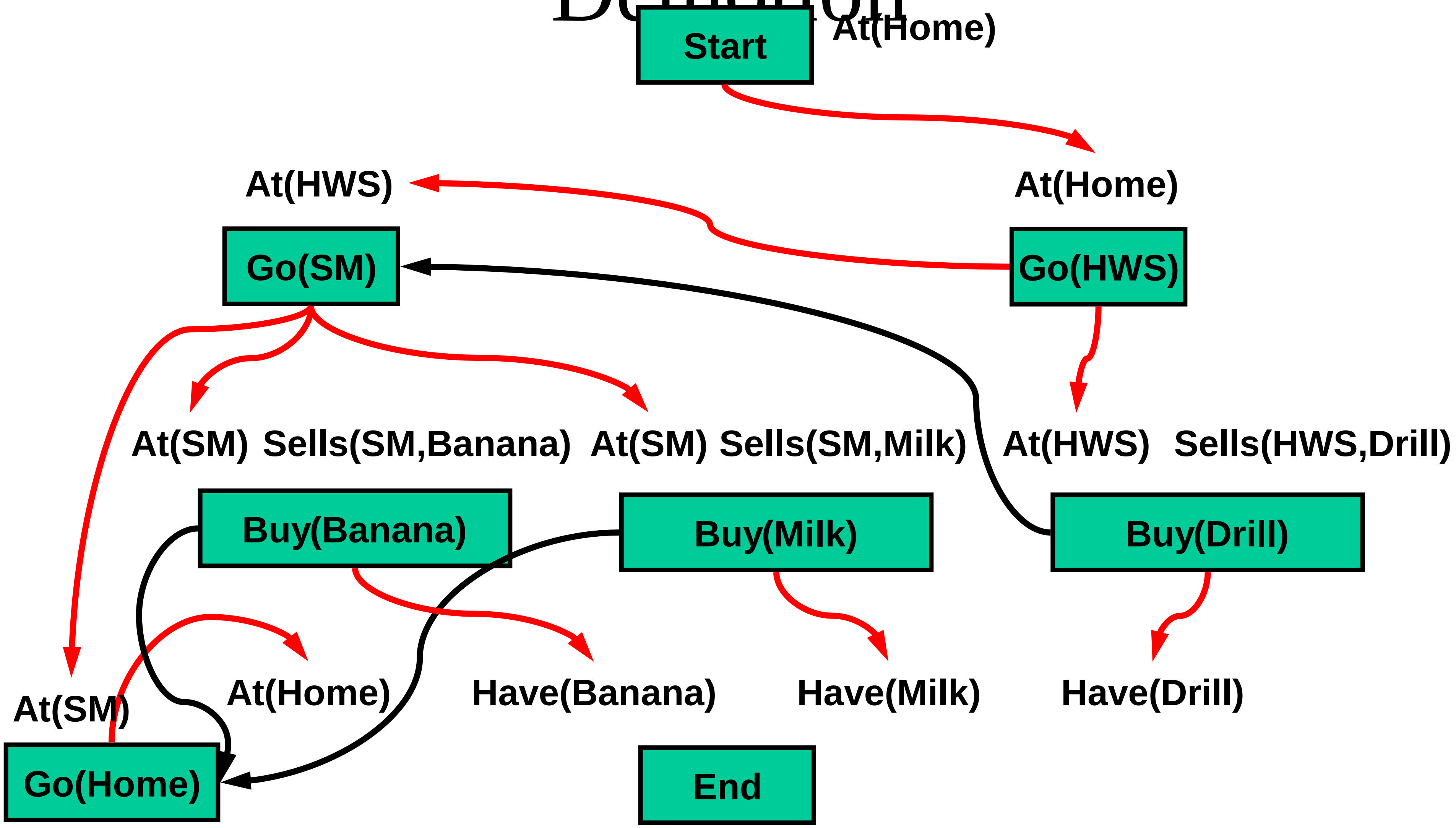
Refining the Plan (4)



Threats to Causal Links



Threats to Causal Links: Demotion



Start
At(Home)
At(HWS)
At(Home)
Go(SM)
Go(HWS)
At(SM)
Sells(SM,Banana)
At(SM)
Sells(SM,Milk)
At(HWS)
Sells(HWS,Drill)
Buy(Banana)
Buy(Milk)
Buy(Drill)
At(SM)
At(Home)
Have(Banana)
Have(Milk)
Have(Drill)
Go(Home)
End