Artificial Intelligence & Soft Computing

CSC 703

Subject In-charge

Safa Hamdare

Assistant Professor

Room No. 407

email: safahamdare@sfit.ac.in

Chapter 3
knowledge ,Reasoning & Planning

Based on CO3:

Analyze the strength and weakness of AI approaches to knowledge representation, reasoning and planning.

Outline of Planning

- **Planning Agent**
- **Types of Planning**
  - Partial Order
  - Hierarchical Order
  - Conditional Order

Planning Agent

Simple Planning Agents

Planning agent generates goals to achieve & constructs plan, Thus **executes this plan** until it is finished, and then begins again with a new goal.

Simple Planning Agents

What is the difference between problem-solving agent and planning agent? (Dec 2015, Dec 2016 5 Marks)

- Differences are in **representations** of
  - states
  - actions
  - goals
- Differences are in the **representation** and **construction** of action sequences (search for solutions)

Basic Elements of a Search-Based Problem-Solver

1. **Representation of states**
   - States are **simple data structures**.
   - All state descriptions are complete. (A complete description of initial state is given. Actions are represented by a program that generates complete state descriptions)
2. **Representation of actions**
   - Actions are described by programs that generate successor **state descriptions**
3. **Representation of goals**
   - Agent has information in the form of **goal test** and the **heuristic function**
4. **Representation of solutions (plans)**
   - A **solution is sequence of actions** beginning from the initial state, and ending at a goal state

Simple Planning Agent's Program

> **Percept**
>
> *TELL (KB, MAKE PERCEPT SENTENCE (percept))*
>
> *Current state description (kb)*

> **Goal generation**
>
> *Ask (kb, make goal query)* $\rightarrow$ *goal*

> **Plan construction**
>
> *Call planning algorithm (current state description, goal, kb)* $\rightarrow$ *plan*

> **Action**
>
> *First action from the PLAN*
>
> *TELL (KB, MAKE ACTION SENTENCE (action))*

> ***Return****: action*

Basic Elements of Planning Agent

- Planning algorithms use descriptions in some formal language, usually first-order logic

1. **Representation of states**
   - Set of **sentences** of first-order logic
2. **Representation of actions**
   - Logical descriptions of **preconditions** and **effects**
3. **Representation of goals**
   - Set of **sentences** of first-order logic
4. **Representation of plans**
   - **Direct connections** between states and actions

Representations for Planning

Representations for Planning

The **STRIPS** Representation. ... **STRIPS**, which stands for "STanford Research Institute Problem Solver," was the **planner** used in Shakey, one of the first robots built using **AI** technology.

1. Representations for states and goals
2. Representations for actions
   - Situation Space
   - Plan Space
3. Representations for Plans
   - Solutions

1.Representations for Planning: States and Goals

- In STRIPS language **states** are represented by **conjunctions** of predicates (possibly negated) with **constants** as their terms
  - **Example**:
    $at(shop) \wedge have(money) \wedge \neg have(computer)$
- In STRIPS language **goals** are represented by **conjunctions** of predicates with constants and variables used as their terms
  - **Examples:**
    $at(home) \wedge have(computer)$
    $at(x) \wedge sells(x, computer)$
  - **Variables** are assumed to be **existentially quantified.**
- Representations of initial states and goals are used as inputs of planning systems

2. Representations for Planning: Actions

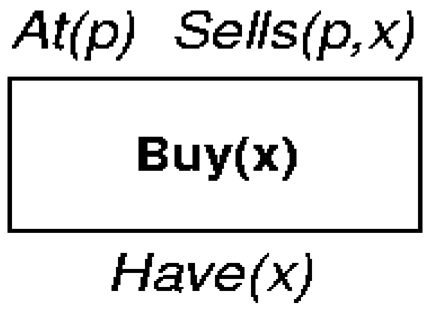
STRIPS Operator Syntax with Example

- **Syntax:**

  Operator (ACTION: action's name,
  PRECONDITION: $p(x) \wedge q(y) \wedge \ldots$
  EFFECT: $r(y) \wedge w(x) \wedge \ldots$)

  where *p* and *q* denotes any predicates; *p* and *q* may be changed to $\neg r$ and/or $\neg w$ in EFFECT
- An operator with variables is called **operator schema**
- All variables are assumed **universally quantified**

- Example:
  Action: Buy (*x*)
  Precondition: At (p), Sells (p, *x*)
  Effect: Have(*x*)



2.Representations for Planning: Actions

- STRIPS language is used in most planners
- STRIPS operators consist of **three components**:

1. The **precondition**
   - It is a conjunction of predicates (not negated) that must be true before the operator can be applied
2. The **action description**
   - Within the planner it is only the name of a possible action
   - Agent returns it to the environment in order to do something
3. The **effect of an operator**
   - It is a conjunction of predicates (possibly negated) that describes how the situation changes when the operator is applied

State space Vs. Plan space

State Space:

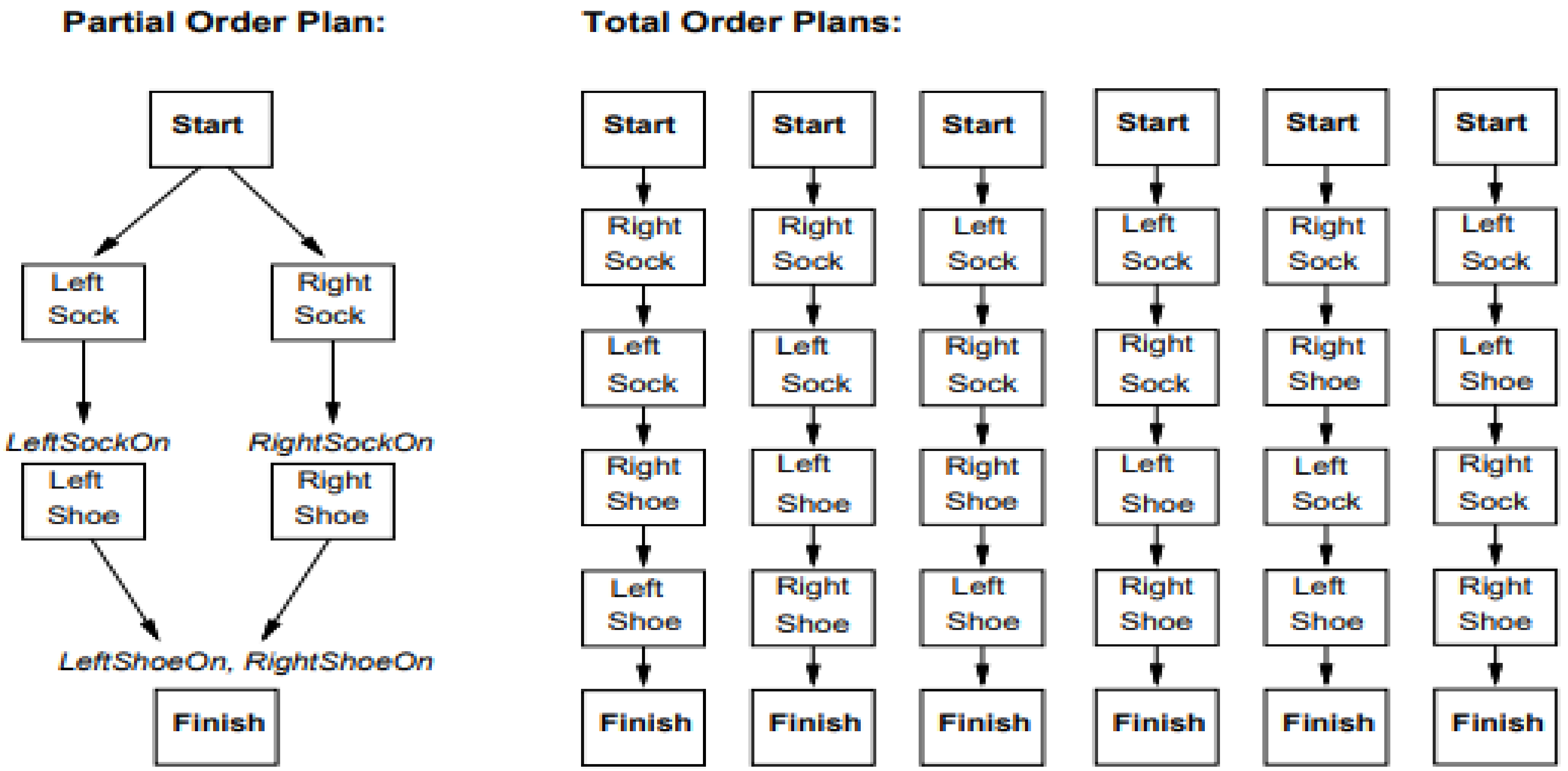
- Also called as **Situation Space.**
- Planning algorithms search through the space of possible states of the world searching for a plan that solves the problem.
- They can be based on **progression**, if it’s a forward search from the initial state looking for the goal state.
- They can be based on **Regression**, if it’s a backward search from the goals towards the initial state.

Plan Space:

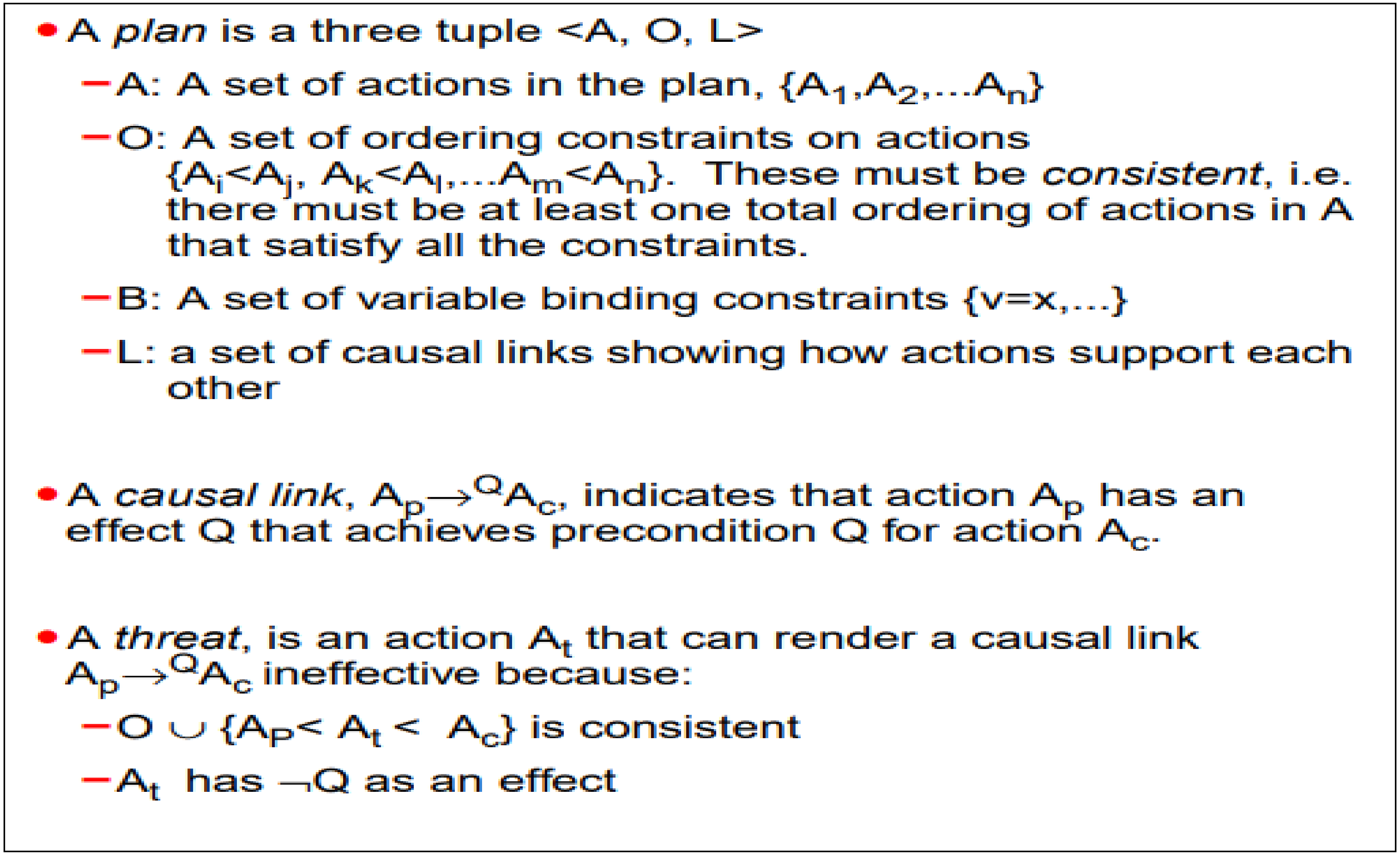
- **STRIPS** is an incomplete regression based algorithm
- **Plan Space planners** search through the space of partial plans, which are sets of actions that may not be totally ordered
- **Partial order planners** are plan based.
- It only introduce ordering constraints as necessary (least commitment)in order to avoid unnecessarily searching through the space of possible orderings.

Partial Order Plans

- Plan in which not all actions are ordered


Partial Order Plan:
Start
Left Sock
Right Sock
LeftSockOn
RightSockOn
Left Shoe
Right Shoe
LeftShoeOn, RightShoeOn
Finish
Total Order Plans:
Start → Right Sock → Left Sock → Right Shoe → Left Shoe → Finish
Start → Right Sock → Left Sock → Left Shoe → Right Shoe → Finish
Start → Left Sock → Right Sock → Right Shoe → Left Shoe → Finish
Start → Left Sock → Right Sock → Left Shoe → Right Shoe → Finish
Start → Right Sock → Right Shoe → Left Sock → Left Shoe → Finish
Start → Left Sock → Left Shoe → Right Sock → Right Shoe → Finish

3.Representations of Plans:



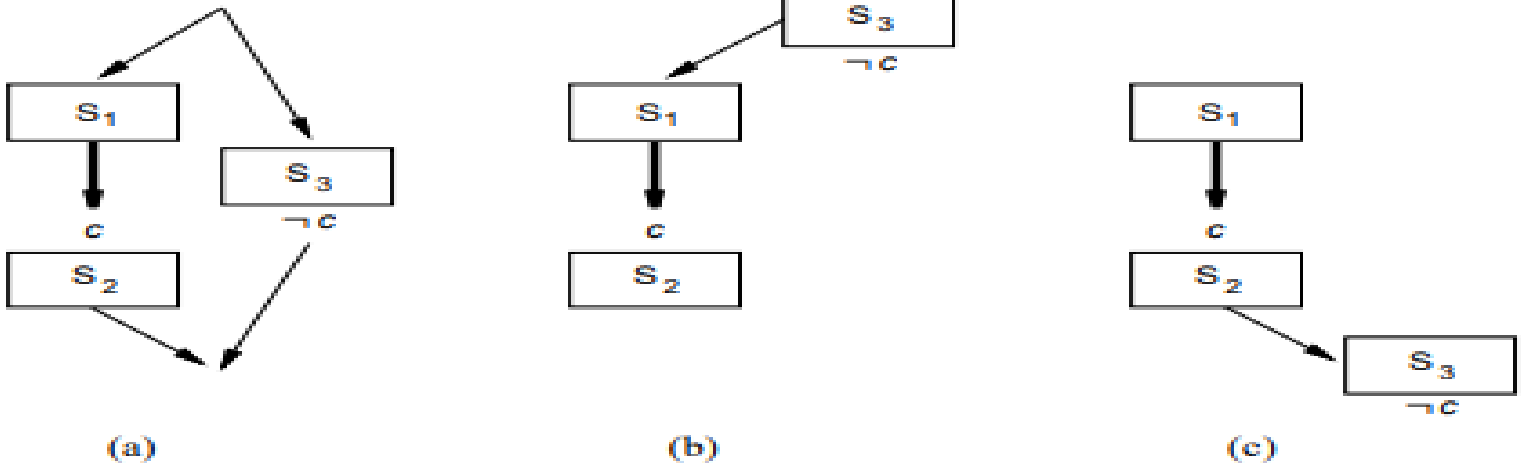
- A *plan* is a three tuple <A, O, L>
  - A: A set of actions in the plan, $\{A_1, A_2, \ldots A_n\}$
  - O: A set of ordering constraints on actions $\{A_i<A_j, A_k<A_l, \ldots A_m<A_n\}$. These must be *consistent*, i.e. there must be at least one total ordering of actions in A that satisfy all the constraints.
  - B: A set of variable binding constraints $\{v=x, \ldots\}$
  - L: a set of causal links showing how actions support each other

- A *causal link*, $A_p \rightarrow^{Q} A_c$, indicates that action A_p has an effect Q that achieves precondition Q for action A_c.

- A *threat*, is an action A_t that can render a causal link $A_p \rightarrow^{Q} A_c$ ineffective because:
  - $O \cup \{A_P < A_t < A_c\}$ is consistent
  - A_t has $\neg Q$ as an effect

Threat Removal

- Threats must be removed to prevent a plan from failing.
- **Demotion** adds the constraint $A_t < A_p$ to prevent clobbering, i.e. push the clobberer before the producer.
- **Promotion** adds the constraint $A_c < A_t$ to prevent clobbering, i.e. push the clobber after the consumer.



Threat Demotion Promotion

1. A Partial-Order Planning Algorithm

Explain Partial Order Planner with an example. Dec 2015 06 marks, Dec 2016 08 marks

- **POP algorithm**
  1. Starts with **minimal partial plan** (starts with goals that must be achieved)
  2. **Extends the plan** by achieving a precondition of a needed step (find operators) (Some operator that achieves the precondition is chosen either from the existing steps of the plan or from the set of operators)
  3. **Records the causal link** for the newly achieved precondition
  4. If the new step threatens an existing causal link or an existing step threatens the new causal link the **threat resolving** is applied
  5. If at any point the algorithm fails to find a relevant operator or resolve a threat, it **backtracks** to a previous choice point
  6. If all the preconditions of all the steps are achieved, it is a **solution**

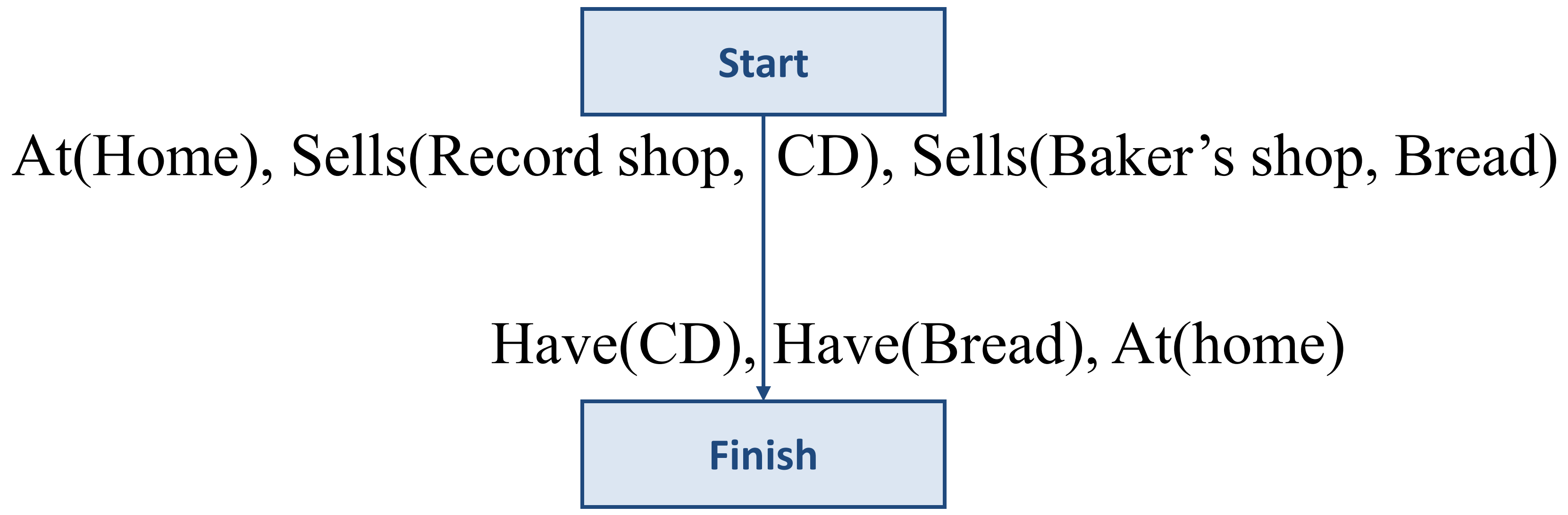
A Partial-Ordered Planning:

- Backtrack for trying another choice when dead end is reached
  - Causal links in a partial plan are **protected links**

Definition: A causal link is protected by ensuring that **threats**, that is, steps that might delete the protected condition, **are ordered** to come **before** or **after** the protected link

A Partial-Ordered Planning: Example 1

Example 1: Draw a Partial Order Plan for the following initial States and goal States:

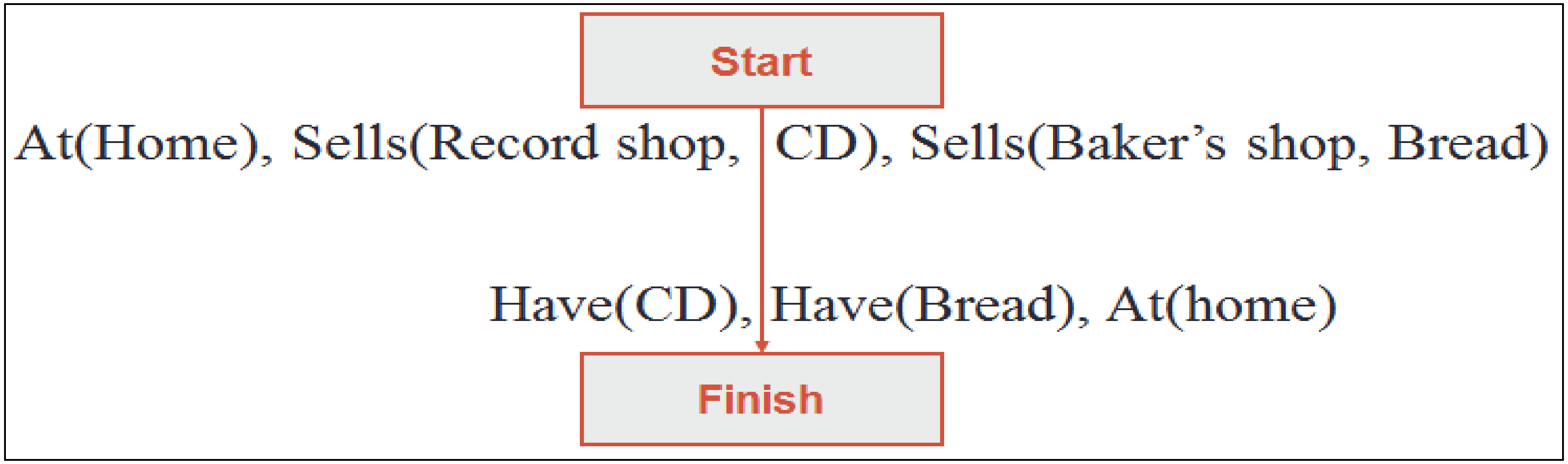

Start
At(Home), Sells(Record shop, CD), Sells(Baker's shop, Bread)
Have(CD), Have(Bread), At(home)
Finish

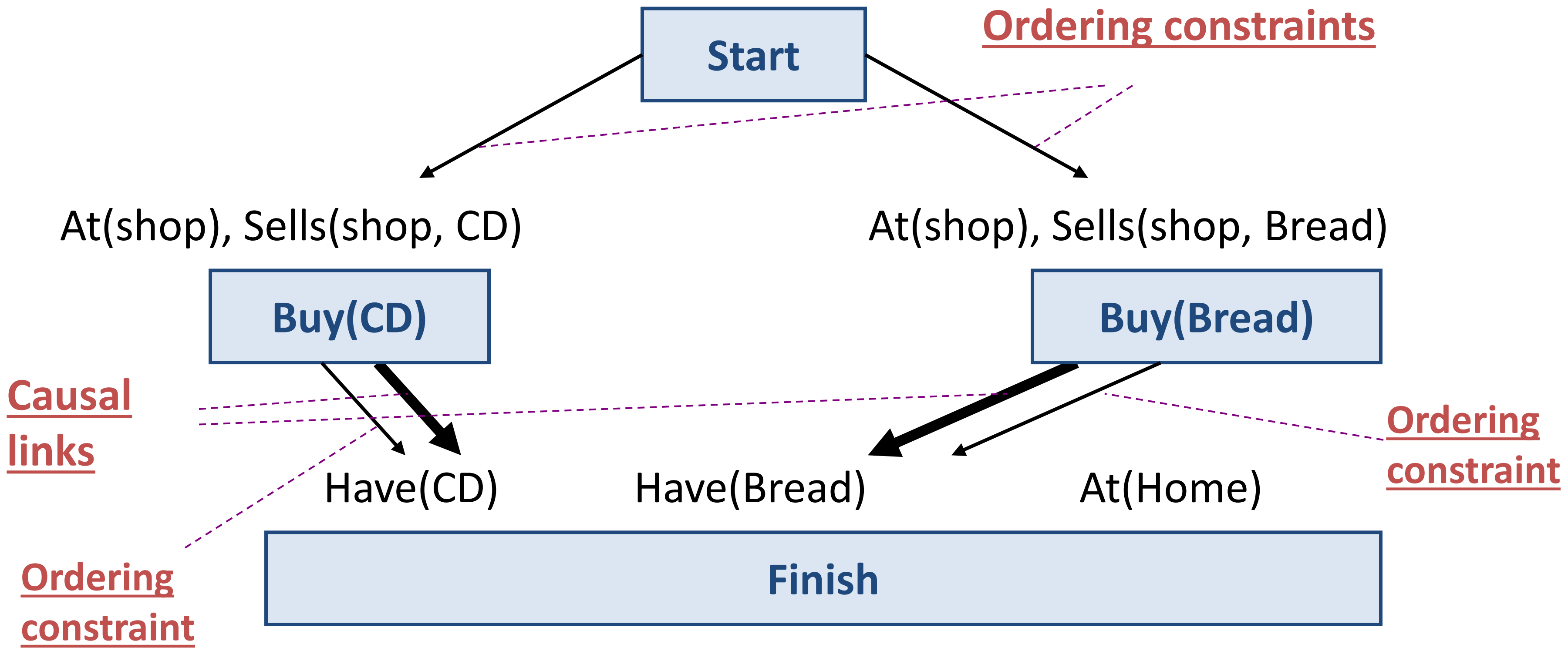
A Partial-Ordered Planning Example

- Definition of the **initial state**
  - Operator (ACTION: Start, EFFECT: At(Home), Sells(Record shop, CD), Sells(Baker's shop, Bread)) $\Big\}$ No Precondition
- Definition of the **final state**
  - Operator (ACTION: Finish, PRECONDITION: Have(CD), Have(Bread), At(Home)) $\Big\}$ No Effect

A Partial-Ordered Planning Example

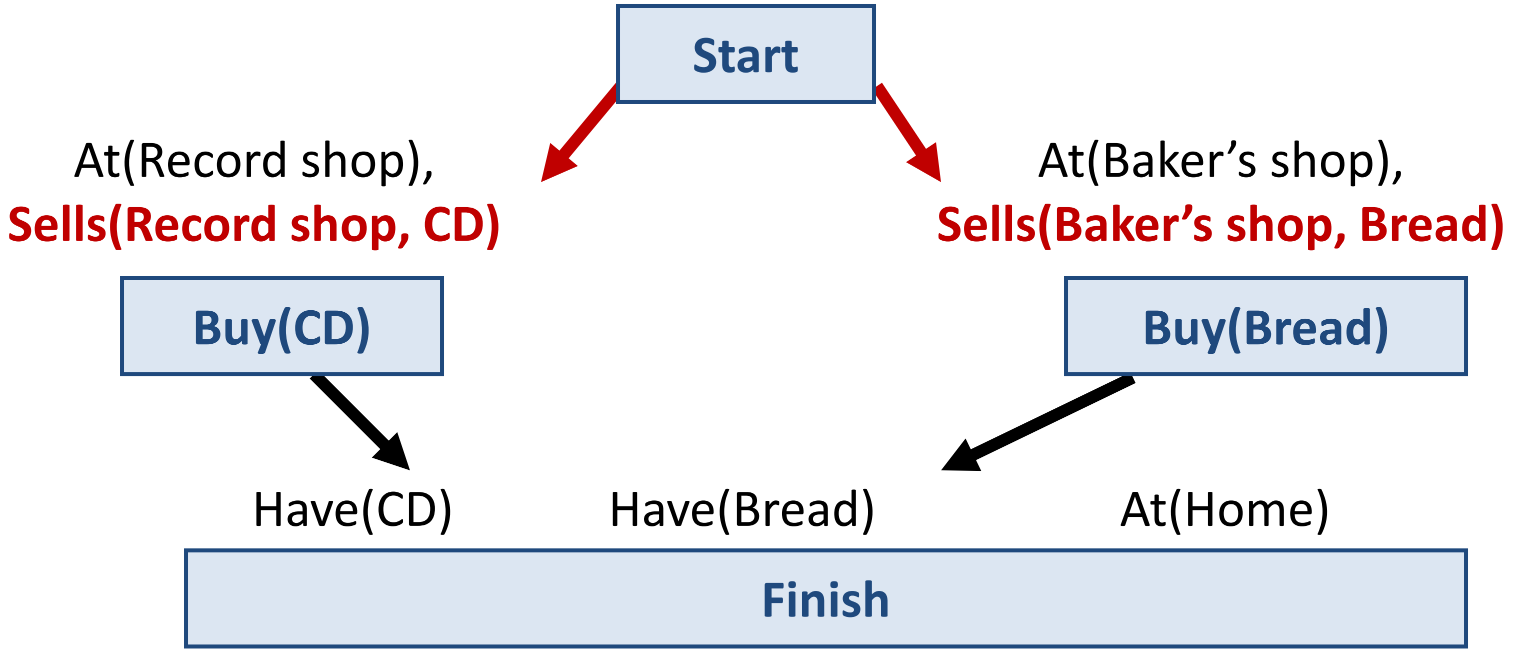
- Definition of the **actions**
  - Operator (ACTION: Go(shop), PRECONDITION: At(Home), EFFECT: (At(shop) $\wedge$ $\neg$At(Home))
  - Operator (ACTION: Buy(x), PRECONDITION: At(shop) $\wedge$ Sells(shop, x), EFFECT: Have(x))


Start
At(Home), Sells(Record shop, CD), Sells(Baker's shop, Bread)
Have(CD), Have(Bread), At(home)
Finish


Start
Ordering constraints
At(shop), Sells(shop, CD)
At(shop), Sells(shop, Bread)
Buy(CD)
Buy(Bread)
Causal links
Ordering constraint
Have(CD)
Have(Bread)
At(Home)
Finish
Ordering constraint

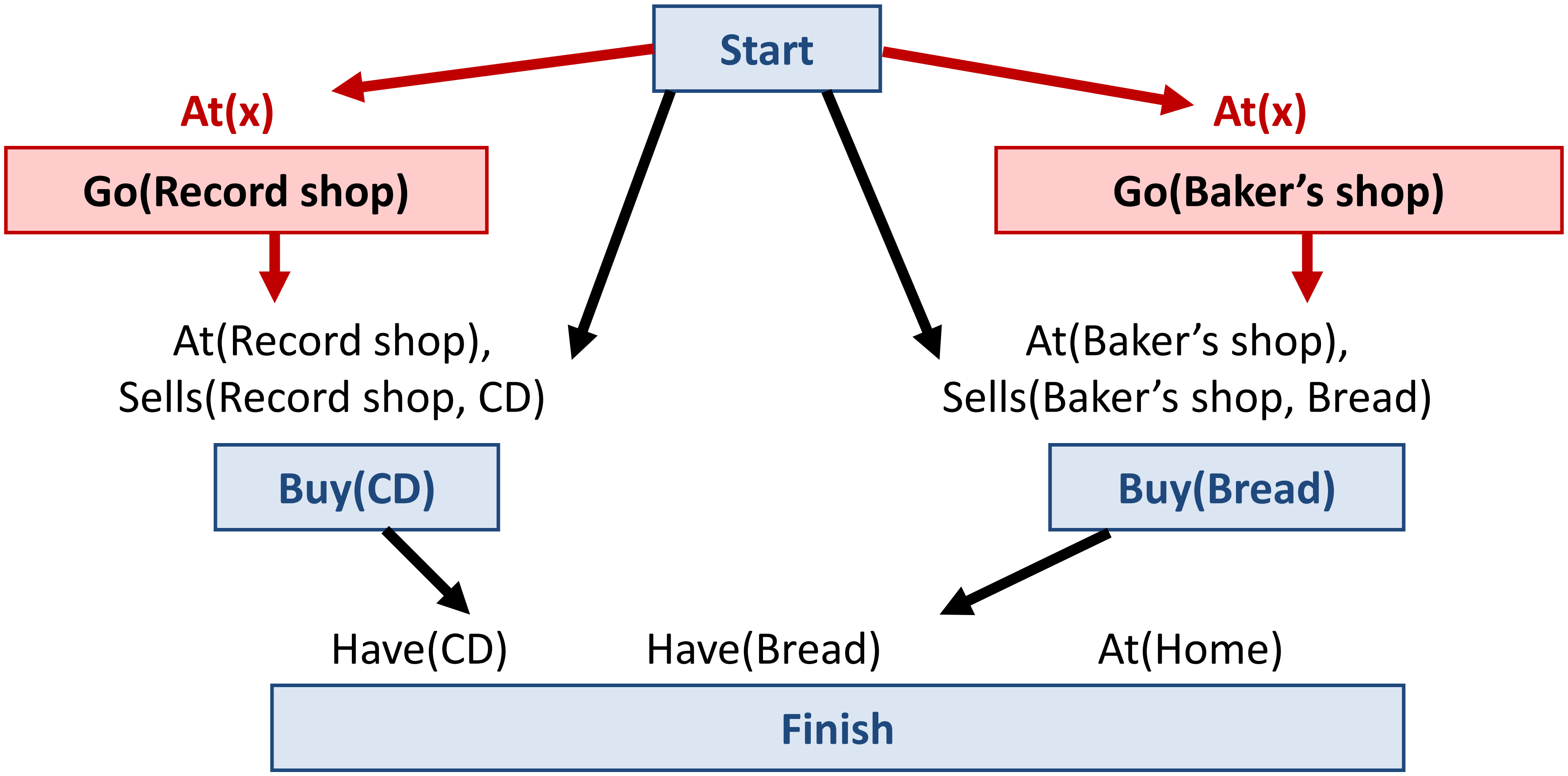
A Partial-Ordered Planning Example

- **The second step**: a partial plan that achieves two preconditions of *Finish* state



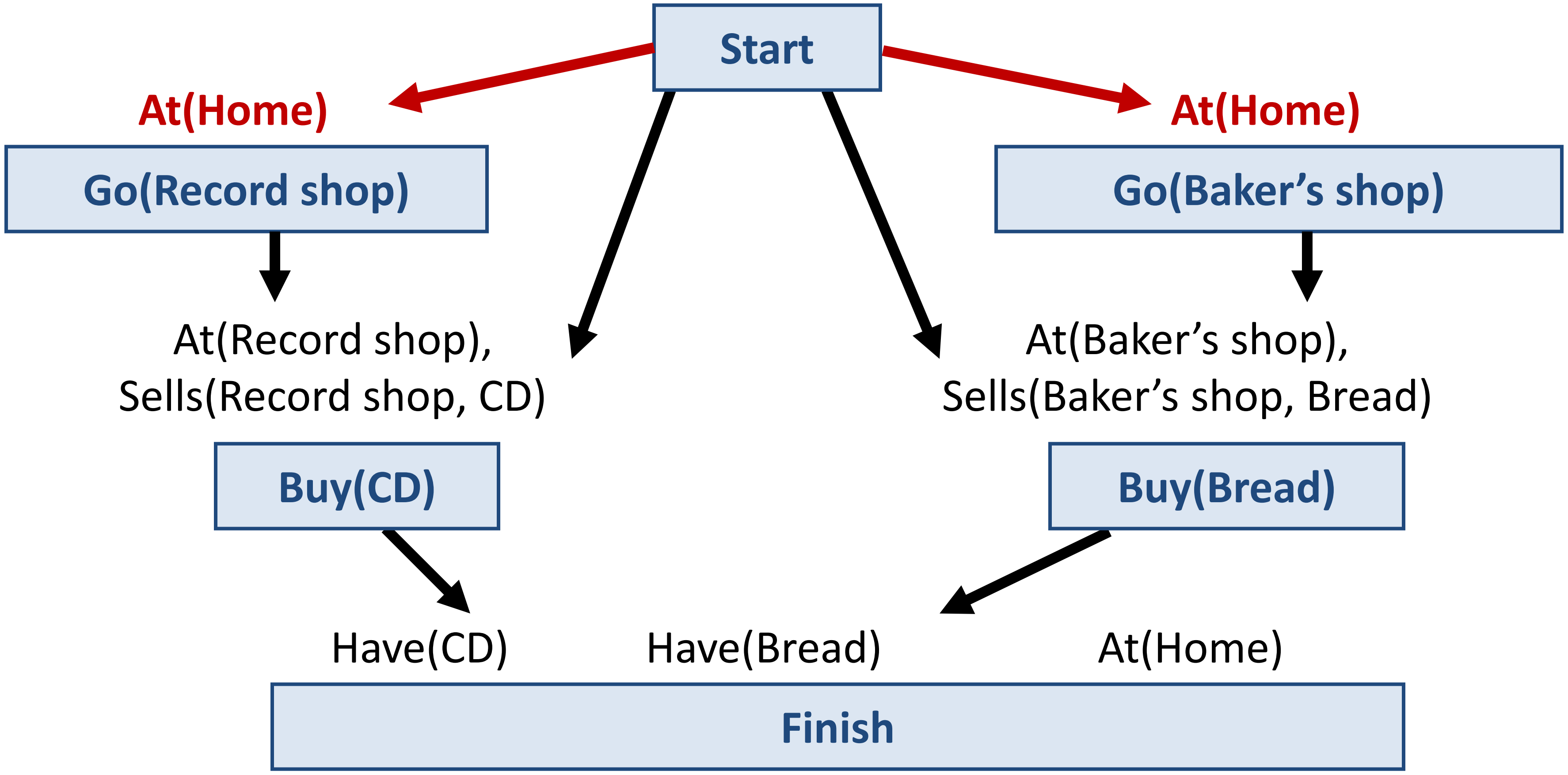
A Partial-Ordered Planning Example

- **The third step**: extension of the plan by choosing two *Go* actions



A Partial-Ordered Planning Example

- The fourth step: a dead end

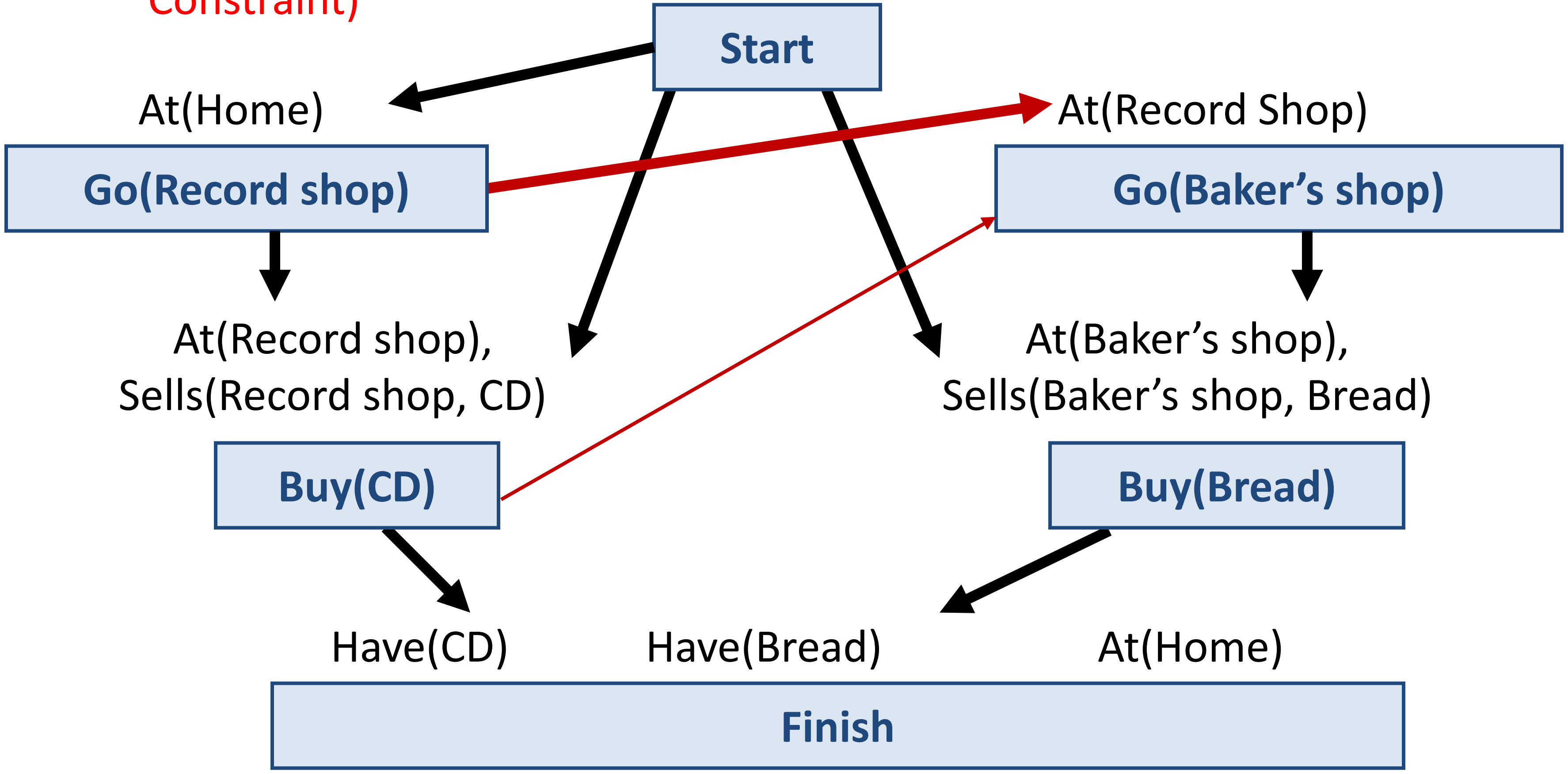


A Partial-Ordered Planning Example

- **Remark**: The step *Go(Record shop)* adds the condition *At(Record shop)*, but it also deletes the condition *At(Home)*, that is, now agent can not go from home to the baker's shop (**whichever *Go* step comes first will delete the *At(Home)* precondition on the other step**)

A Partial-Ordered Planning Example

- **The fifth step**: back up to try a different choice at some earlier point in the planning process. (Red line is Casual Link, black line is ordering Constraint)

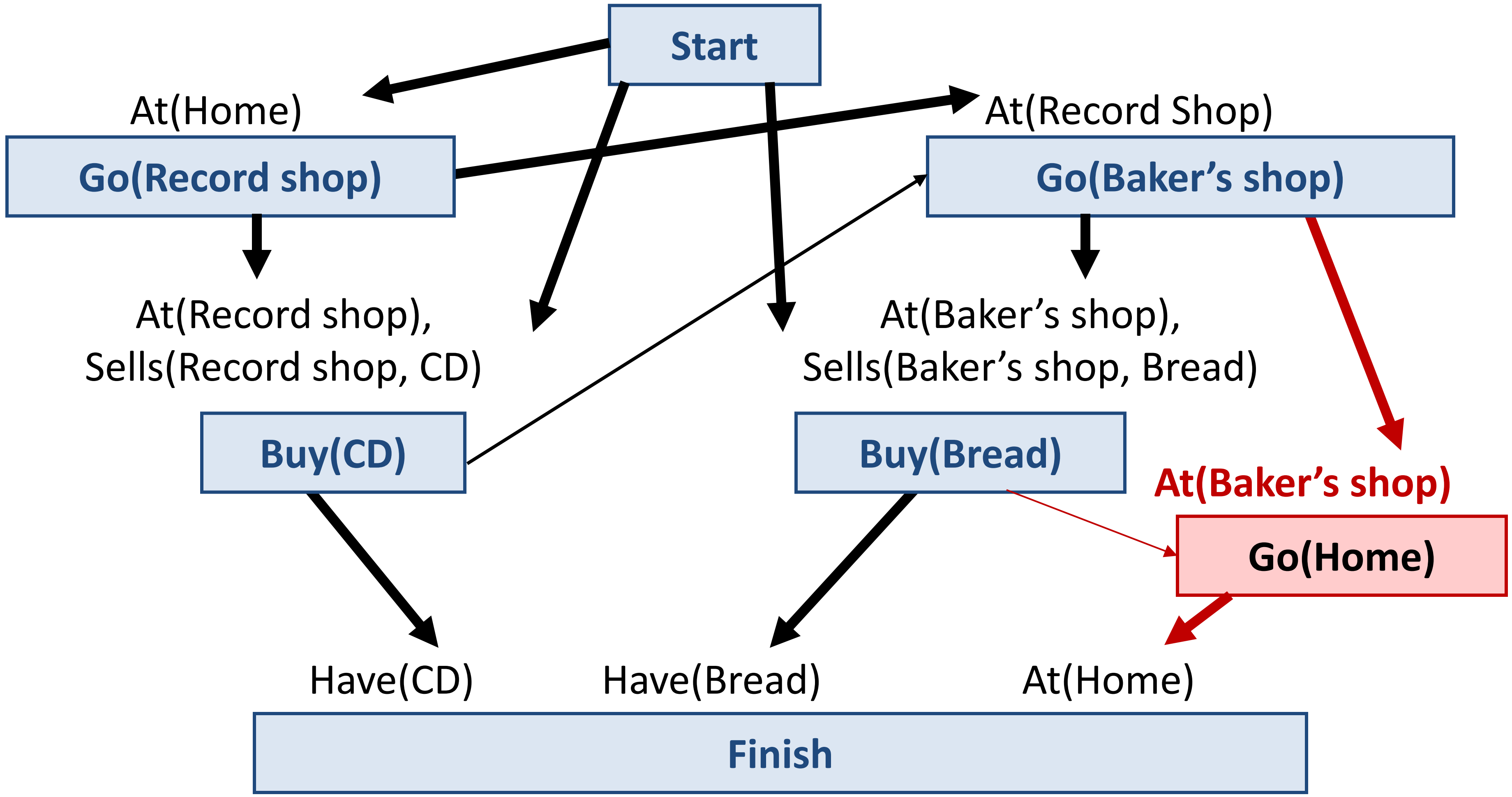


A Partial-Ordered Planning Example

- **Remark**: *Go(Baker's shop)* step threatens the *At(Record shop)* precondition of the *Buy(CD)* step. Constraining *Go(Baker's shop)* step to come after *Buy(CD)* step will not allow the agent to go from the record shop to the baker's shop without first buying the CD

A Partial-Ordered Planning Example

- **The sixth step**: adding a *Go(Home)* step

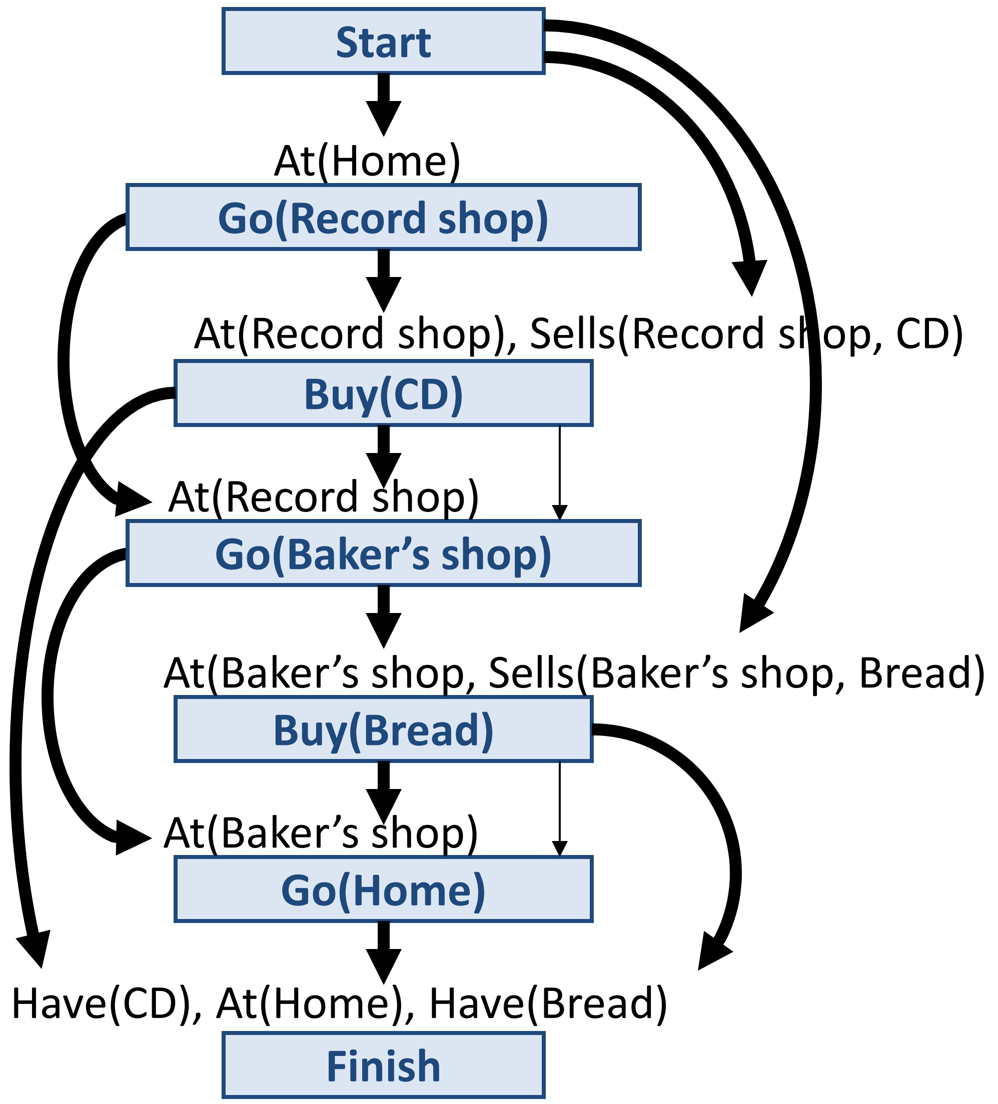


A Partial-Ordered Planning Example

- **Remark**: Notice that *Go(Home)* step is ordered to be after *Buy(Bread)* step

A Partial-Ordered Planning Example

- A complete solution plan (totally ordered)


Start
At(Home)
Go(Record shop)
At(Record shop), Sells(Record shop, CD)
Buy(CD)
At(Record shop)
Go(Baker's shop)
At(Baker's shop, Sells(Baker's shop, Bread)
Buy(Bread)
At(Baker's shop)
Go(Home)
Have(CD), At(Home), Have(Bread)
Finish

MCQ

1. The process by which the brain incrementally orders actions needed to complete a specific task is referred as _______________
   a) Planning problem
   b) Partial order planning
   c) Total order planning
   d) Both Planning problem & Partial order planning
2. What is the advantage of totally ordered plan in constructing the plan?
   A. Reliability
   B. Easy to use
   C. Flexibility
   D. All of the above
3. What are not present in start actions?
   A. Preconditions
   B. Effect
   C. Finish
   D. None of the Above

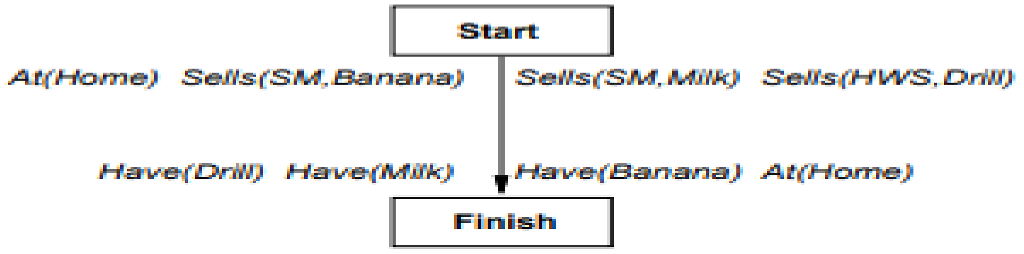
4. What are not present in finish actions?
A. Preconditions
B. Effect
C. Finish
D. None of the Above

Sample Example-2 to Solve on POP:

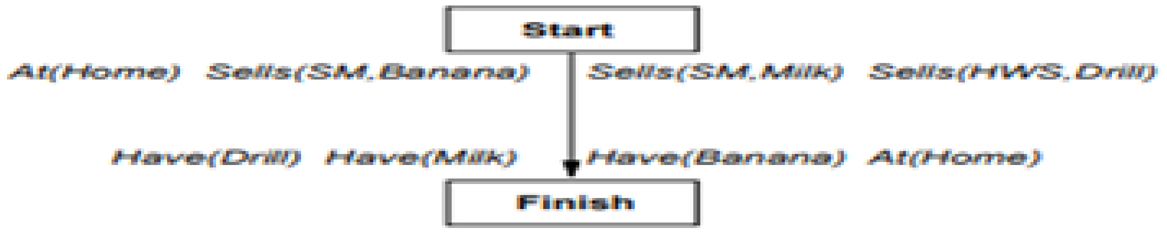
Initial (Null) Plan

- Initial plan has
  - $A=\{ A_0, \ A_\infty\}$
  - $O=\{A_0 < A_\infty\}$
  - $L = \{\}$
- A_0(Start) has no preconditions but all facts in the initial state as effects.
- A_∞ (Finish) has the goal conditions as preconditions and no effects.

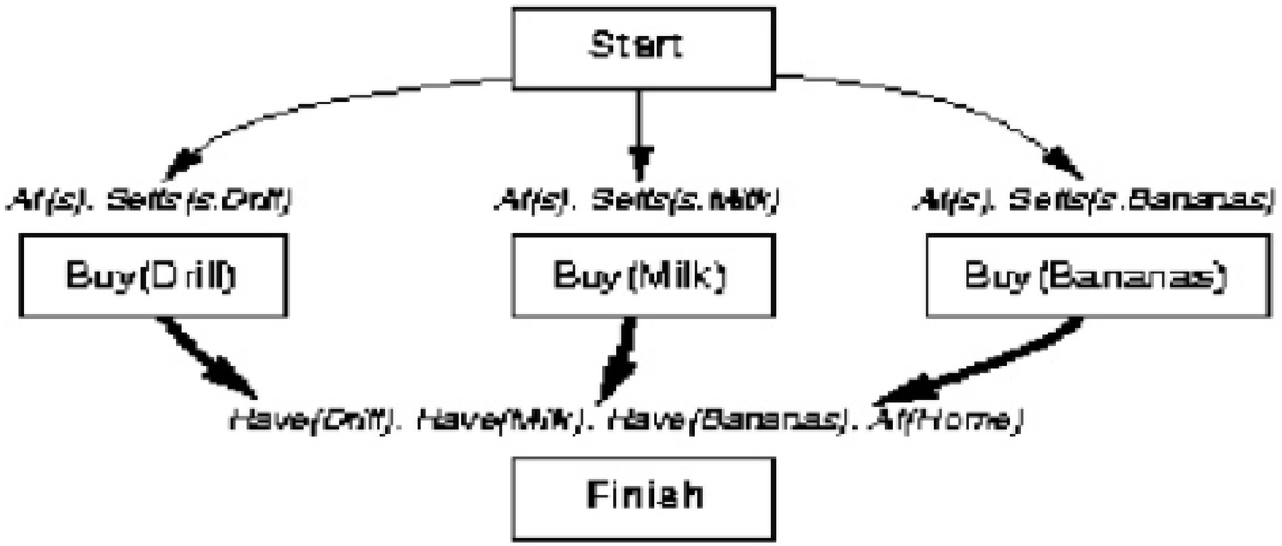
Op(Action: Go(there); Precond: At(here);
      Effects: At(there), $\neg$At(here))
Op(Action: Buy(x), Precond: At(store), Sells(store,x);
      Effects: Have(x))



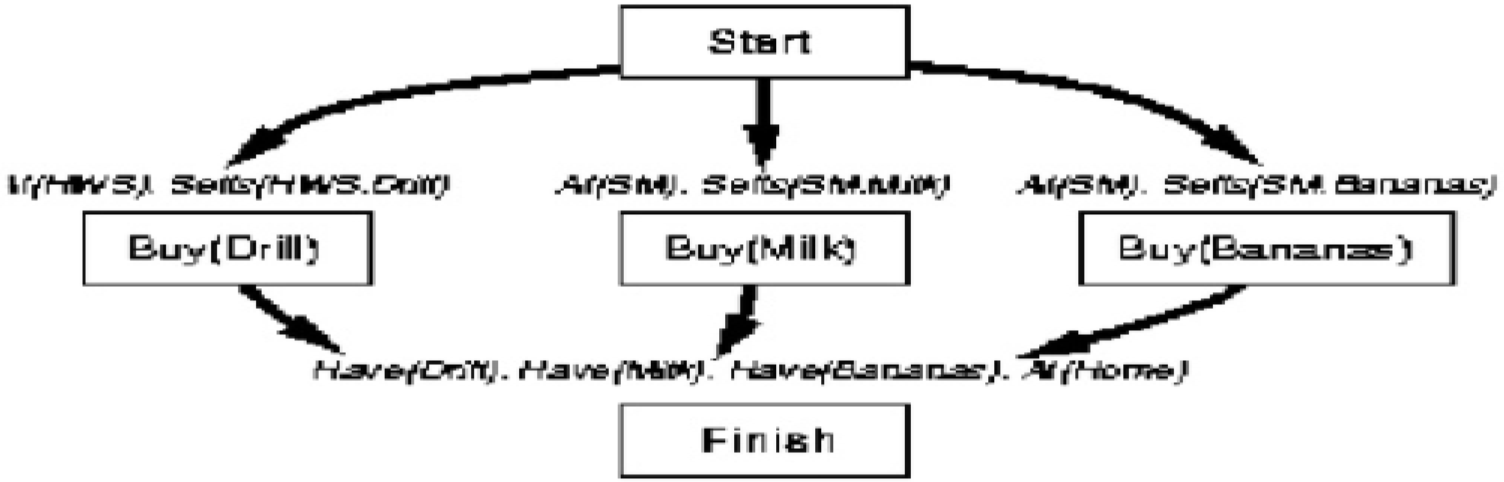
Op(Action: Go(there); Precond: At(here);
Effects: At(there), ¬At(here))
Op(Action: Buy(x), Precond: At(store), Sells(store,x);
Effects: Have(x))



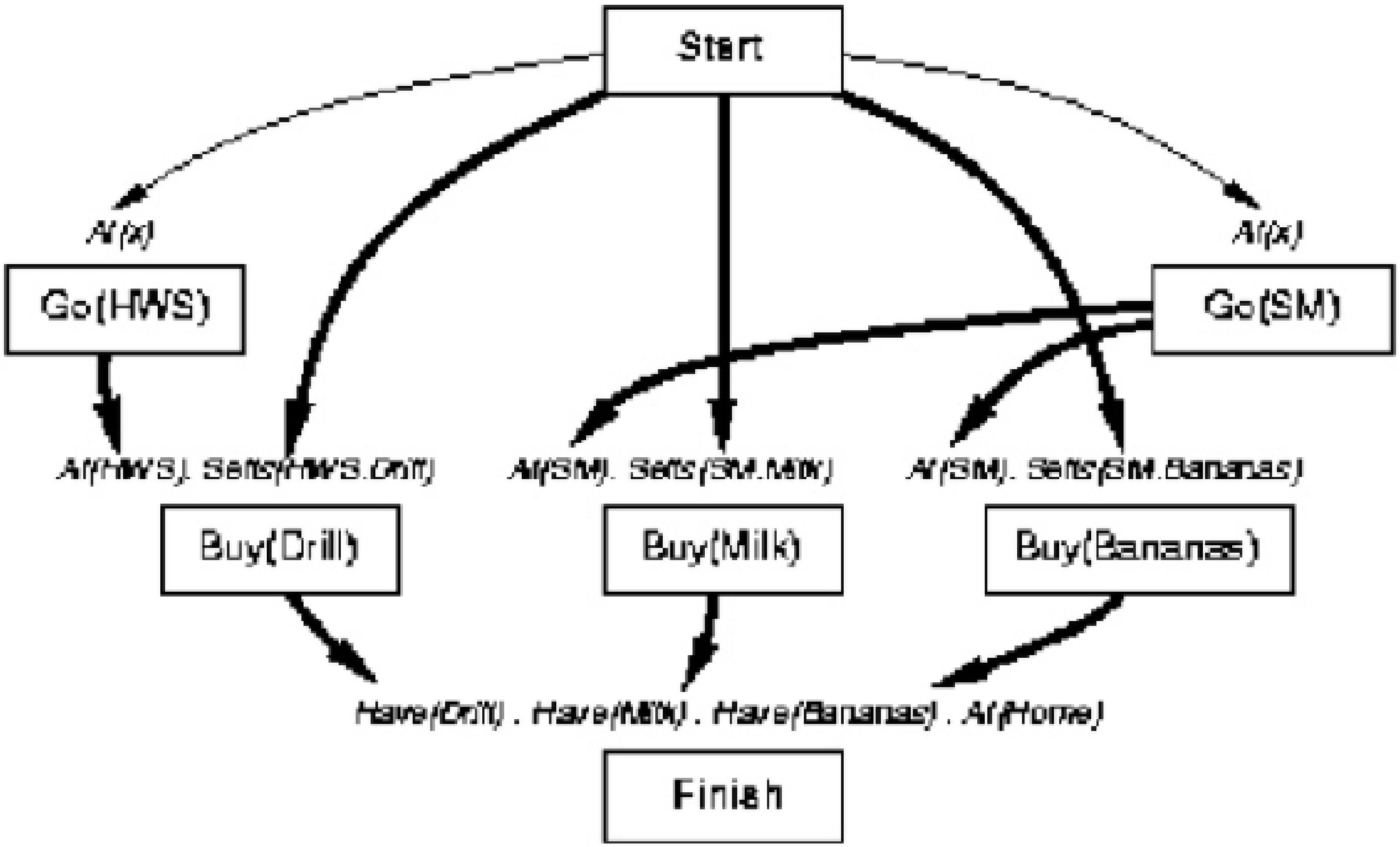
Step 1


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

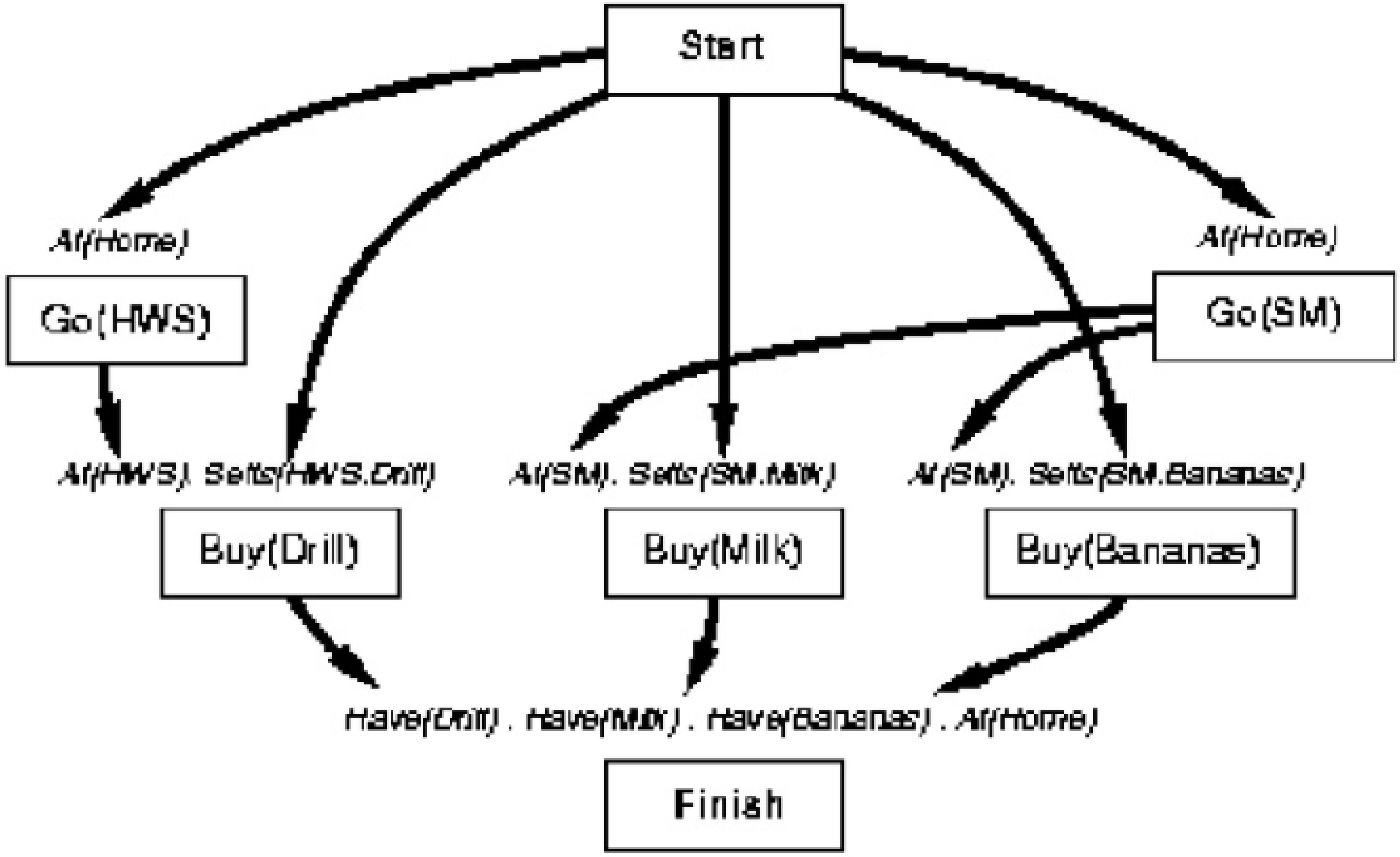
Step 2


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

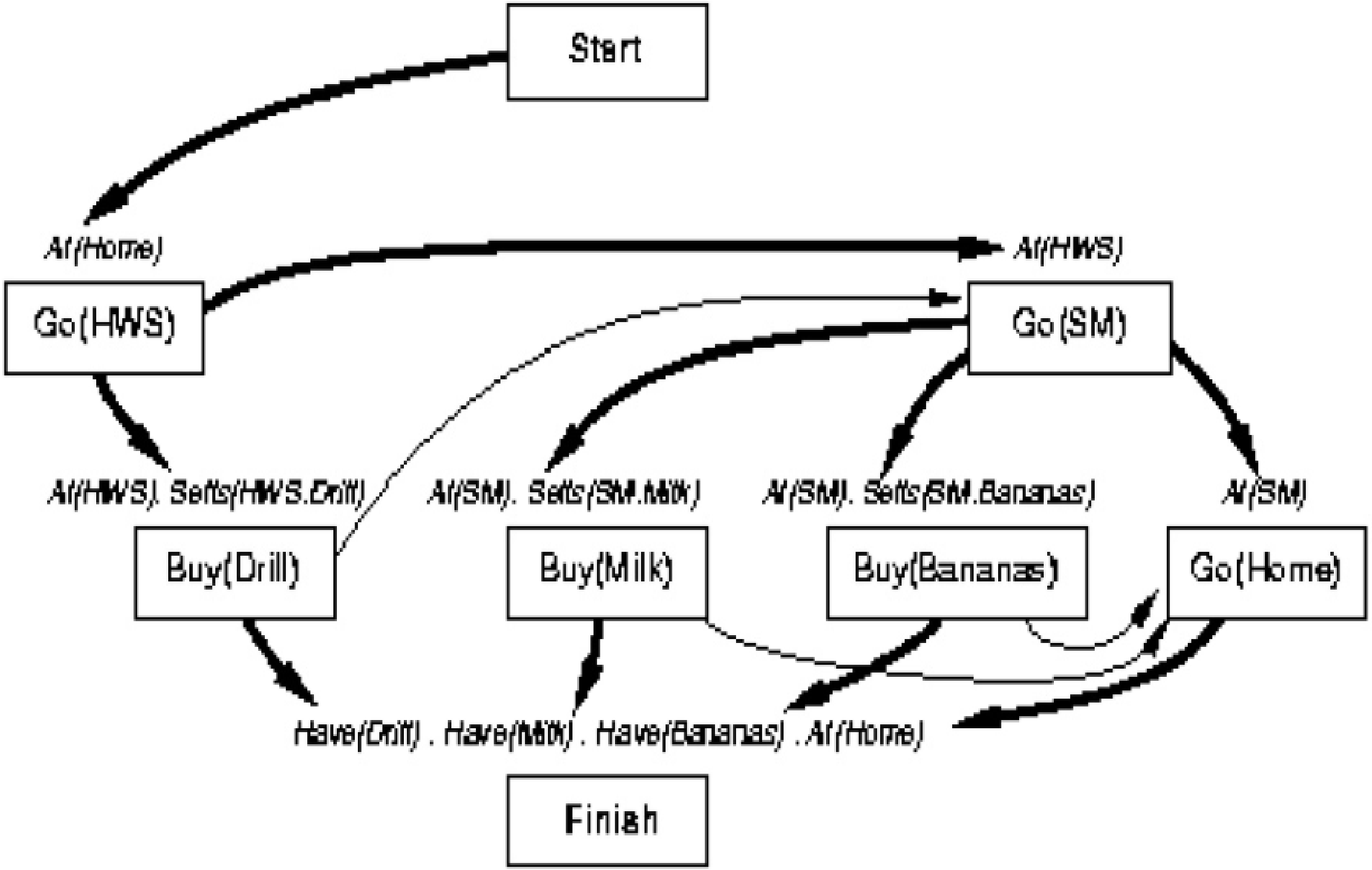
Step 3


Start
At(x)
At(x)
Go(HWS)
Go(SM)
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

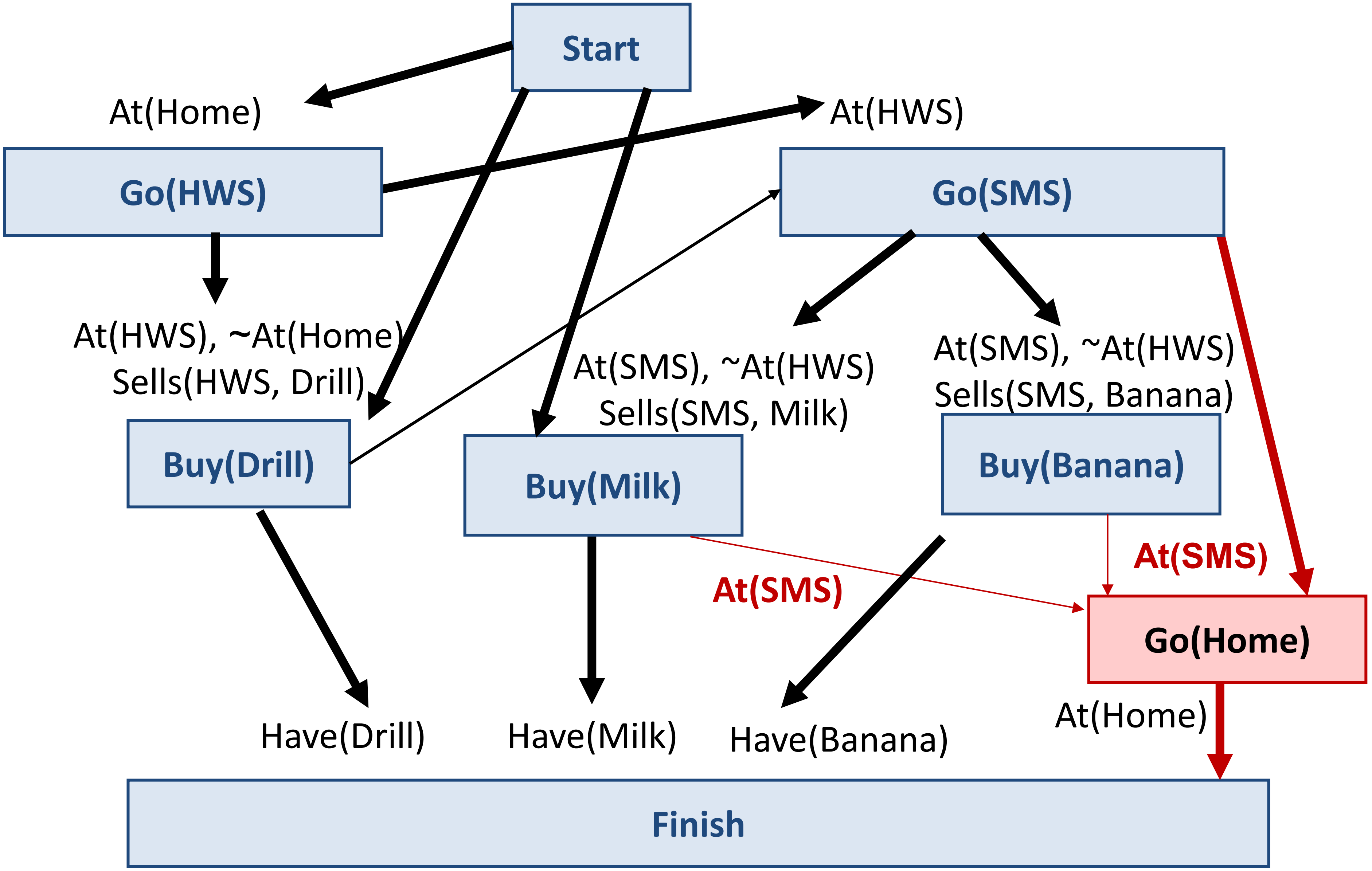
Step 4


Start
At(Home)
Go(HWS)
At(Home)
Go(SM)
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

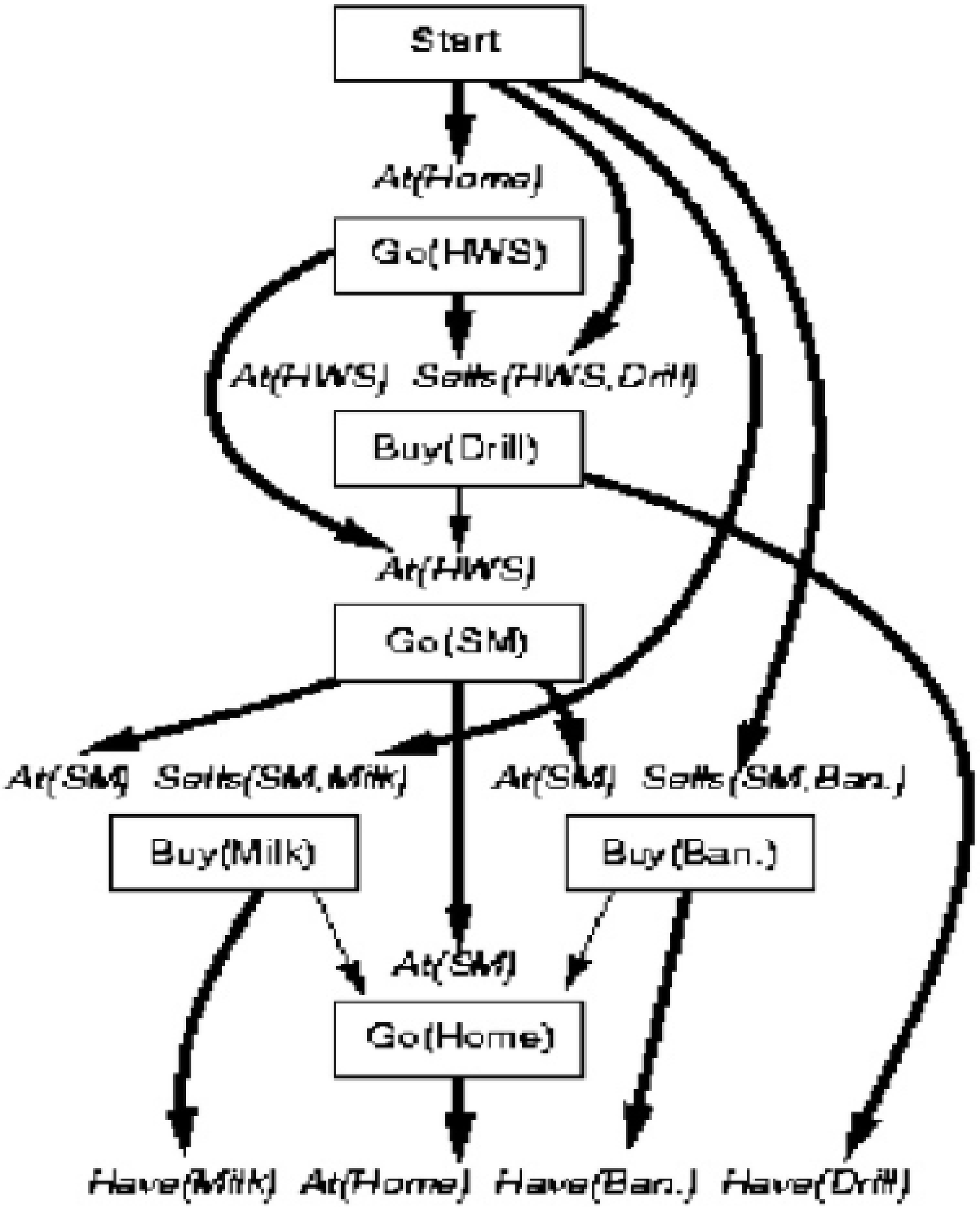
Step 5 (Resolving Threat)



Final Solution



Total order Plan

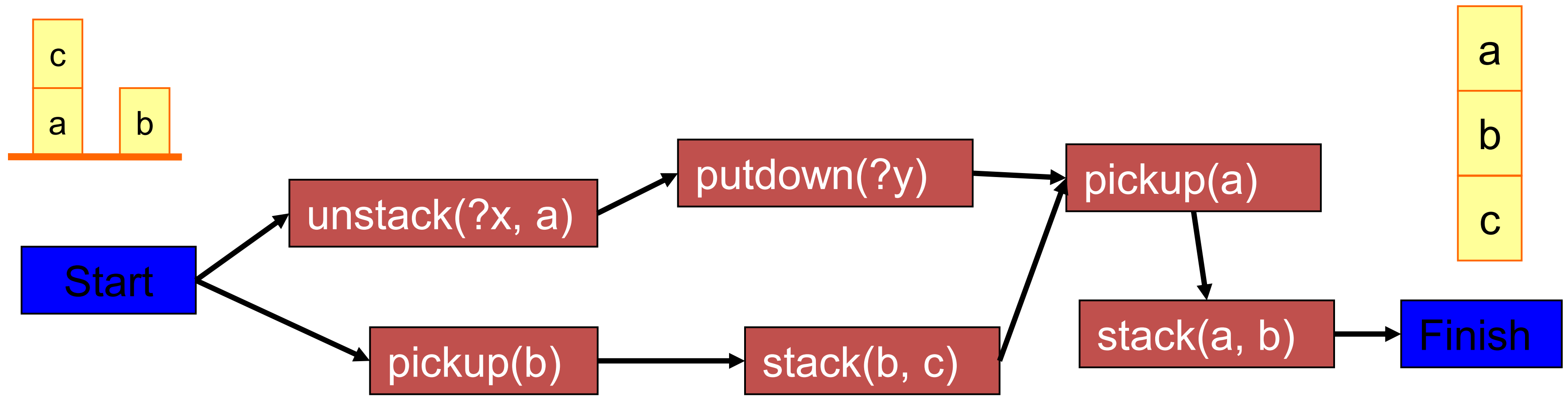


Partial Plans: Example 3

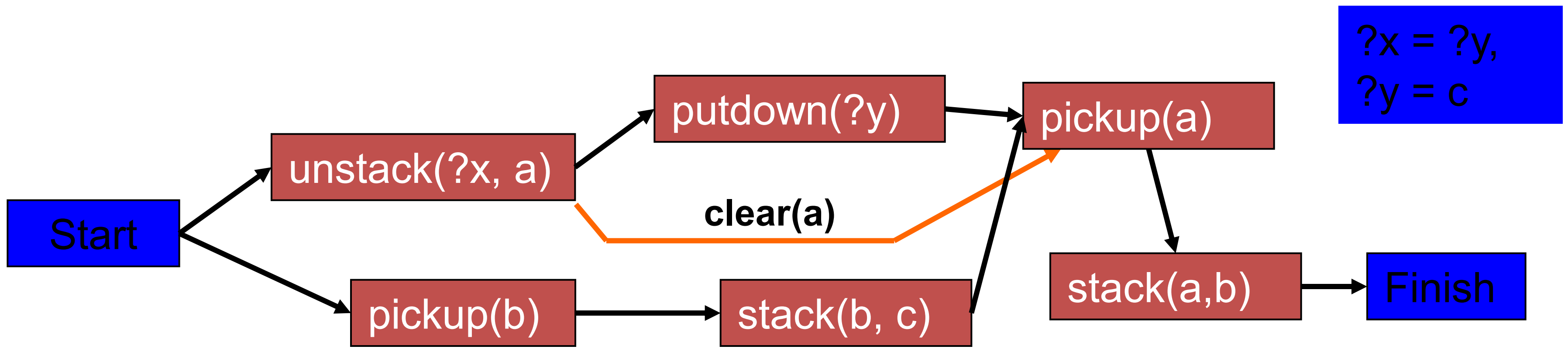
- Partial plan: a partially ordered set of operator instances

The partial order gives only some constraints on the order in which the operations have to be performed

- **Start** a dummy operator
- **Finish** another dummy operator

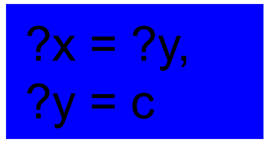


Partial Plans

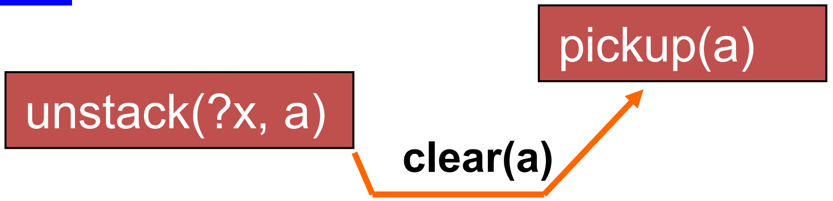


► Partial plans can also contain the following information:

♦ variable binding constraints



♦ causal links

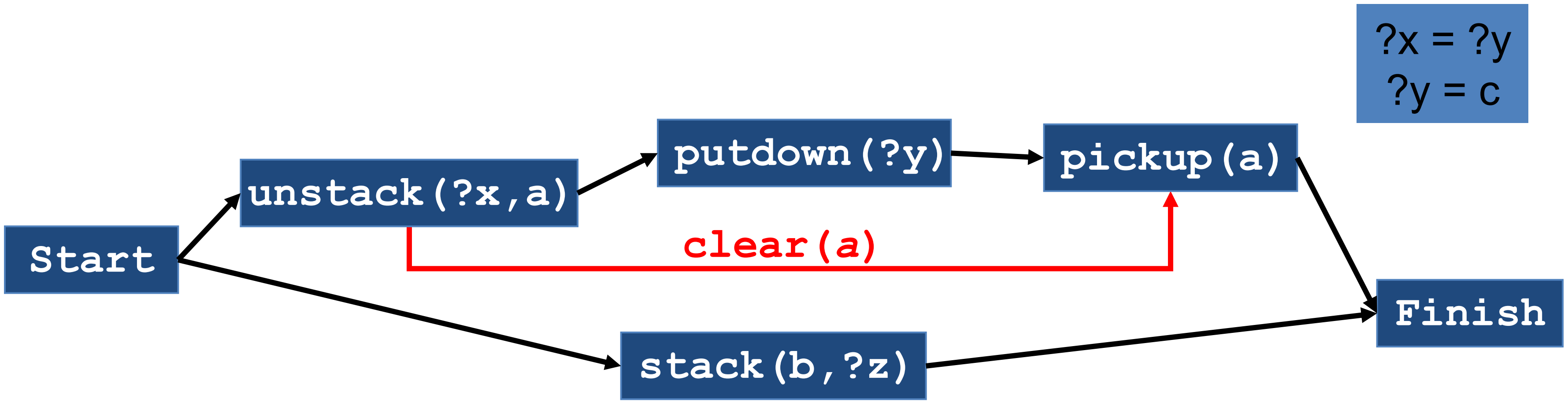


often called protected conditions

Threats: Example

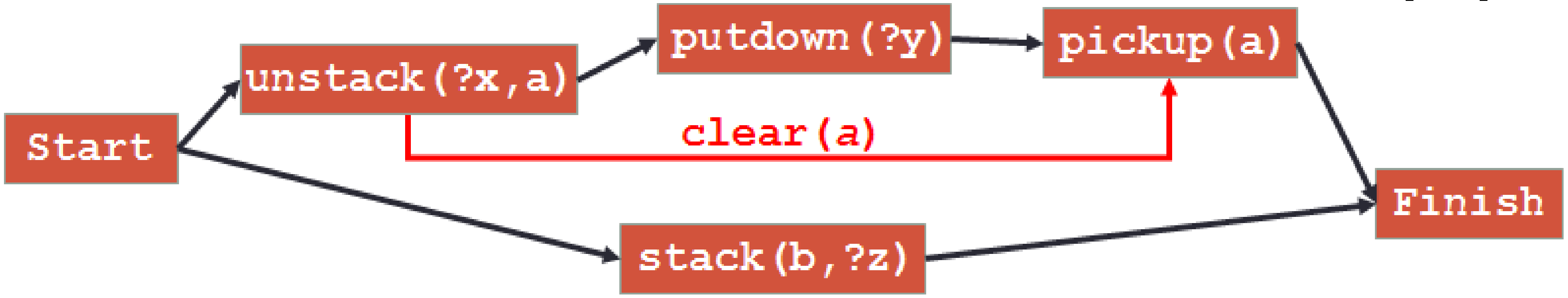
r = stack(*b*,?*z*)

below is a threat to the causal link clear (a)



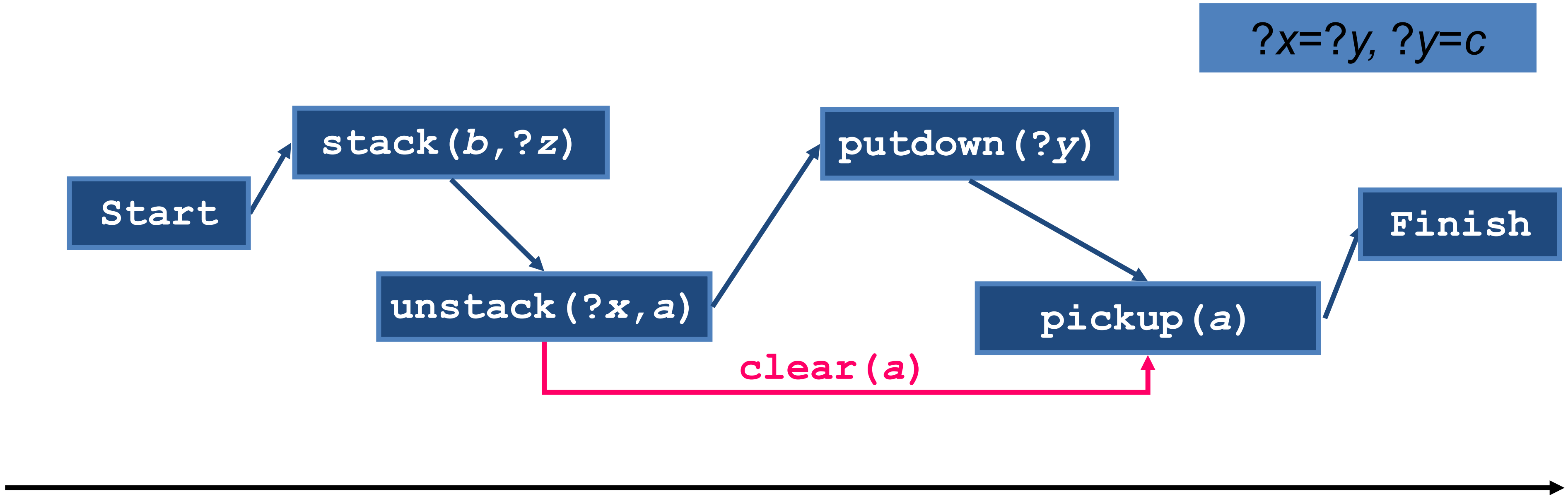
- Threats “threaten” correctness!

Threat Detection and Removal (1)



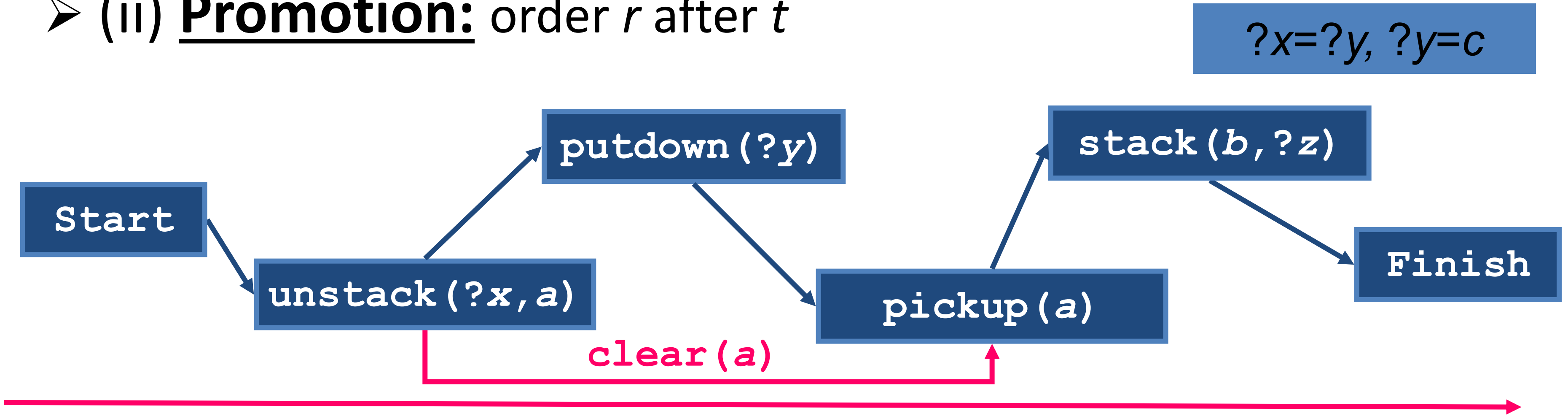
- (i) demotion, (ii) promotion

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

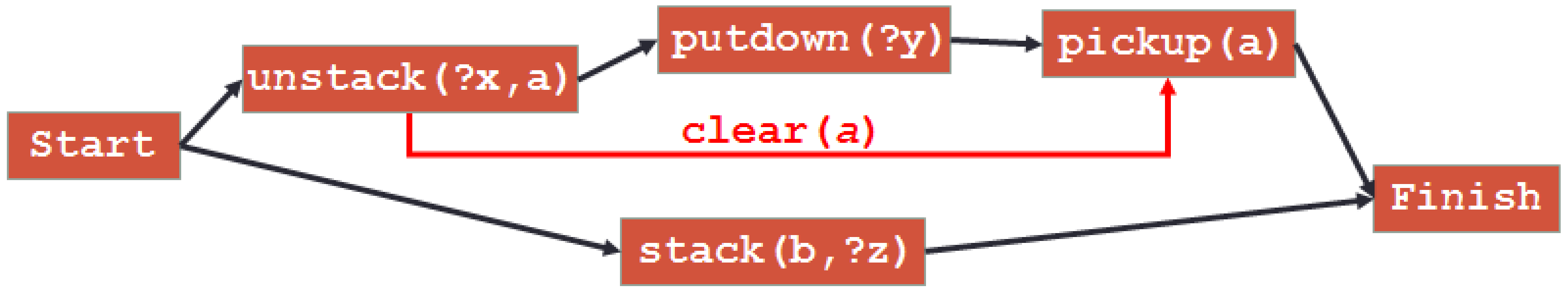


Threat Detection and Removal

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

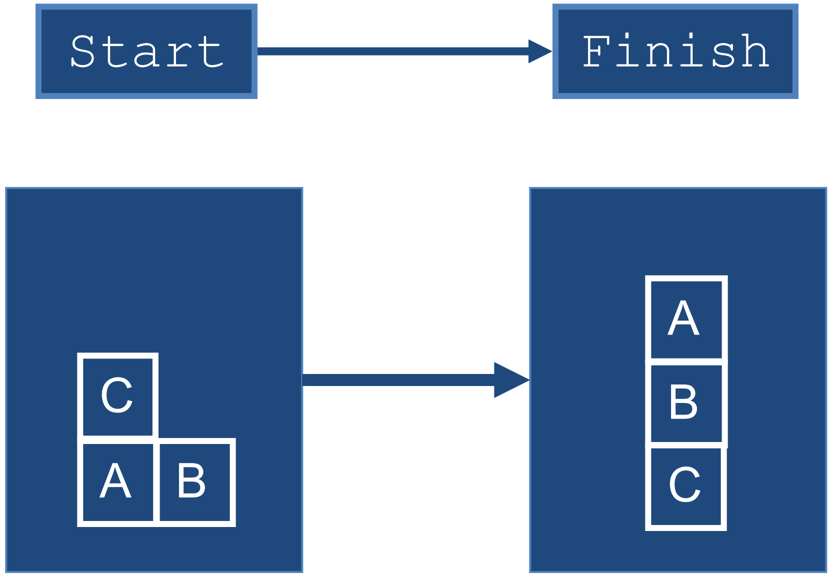


Actual Plan:



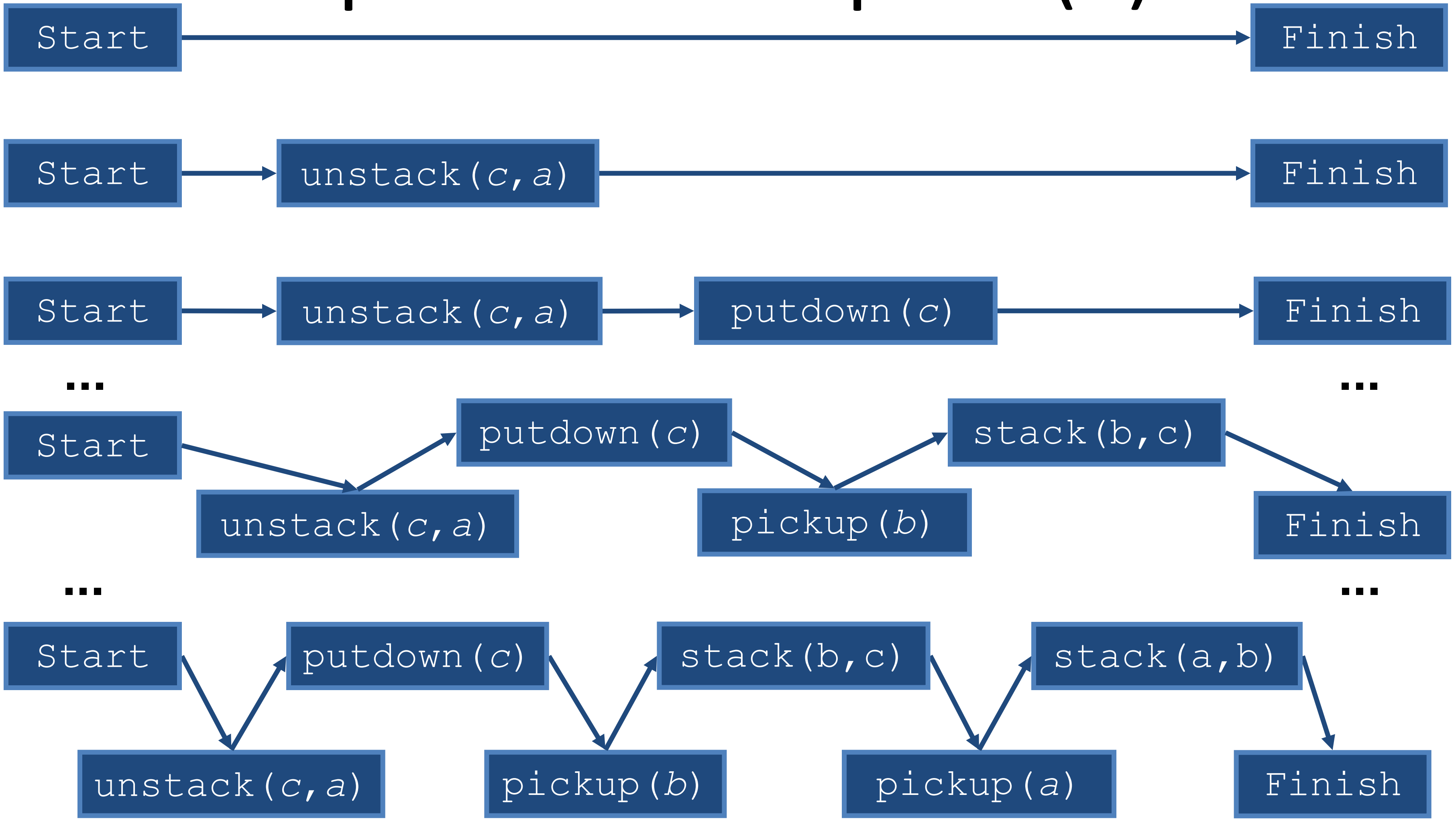
POPLAN Search Space (1)

```
on(c,a)
ontable(a)
clear(c)
ontable(b)
clear(b)
handempty
```



```
on(b,c)
on(a,b)
clear(a)
```

Poplan Search Space (2)



2. Hierarchical Planning

Hierarchical Planning / Plan Decomposition

Plans are organized in a hierarchy. Links between nodes at different levels in the hierarchy denote a decomposition of a “complex action” into more primitive actions (operator expansion).

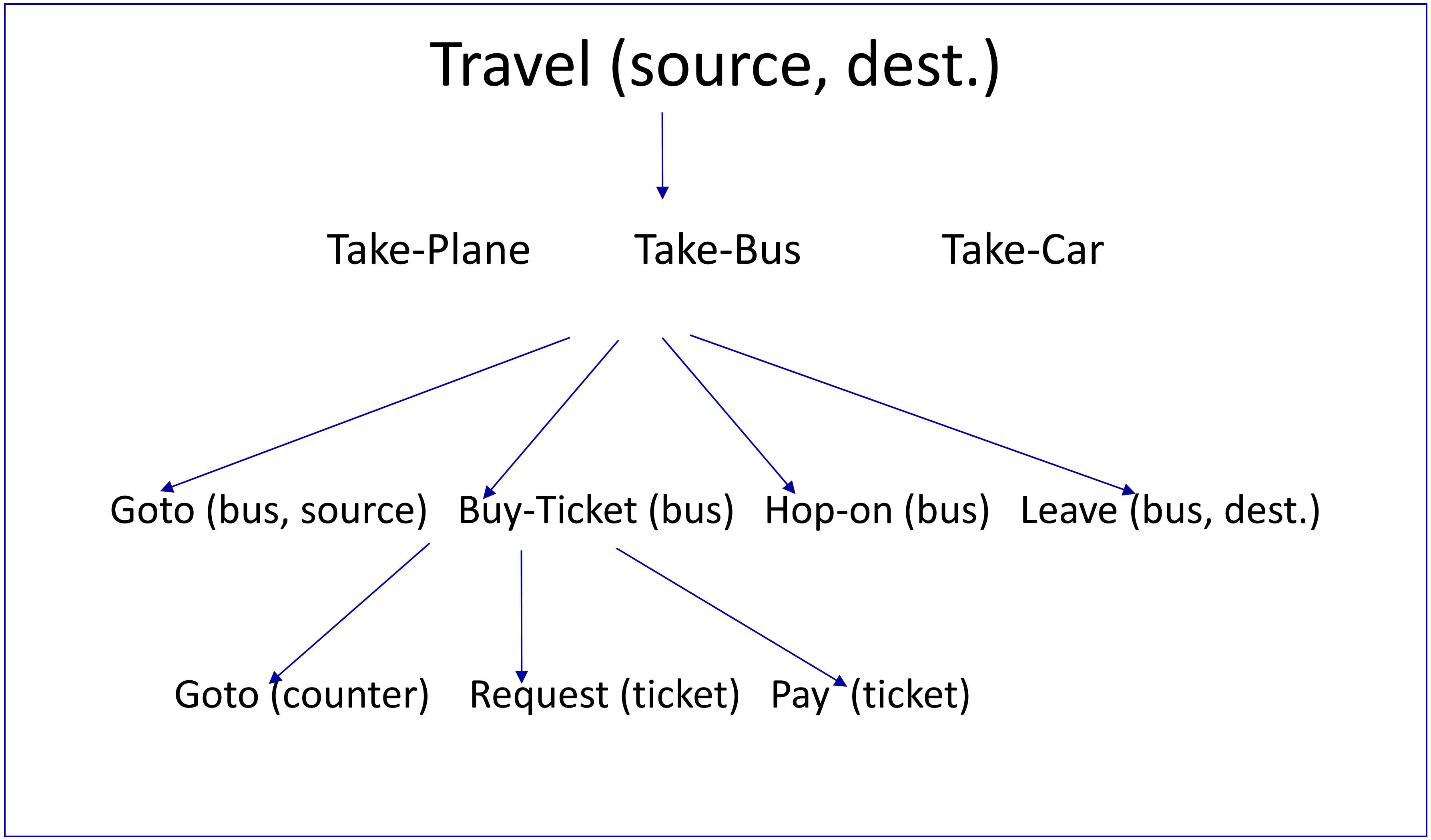
Example:

operator expansion

$$\begin{tikzcd}[column sep=small] & \text{move } (x, y, z) \arrow[dl, no head] \arrow[dr, no head] & \\ \text{pickup } (x, y) & & \text{putdown } (x, z) \end{tikzcd}$$

The lowest level corresponds to executable actions of the agent.

Hierarchical Plan - Example



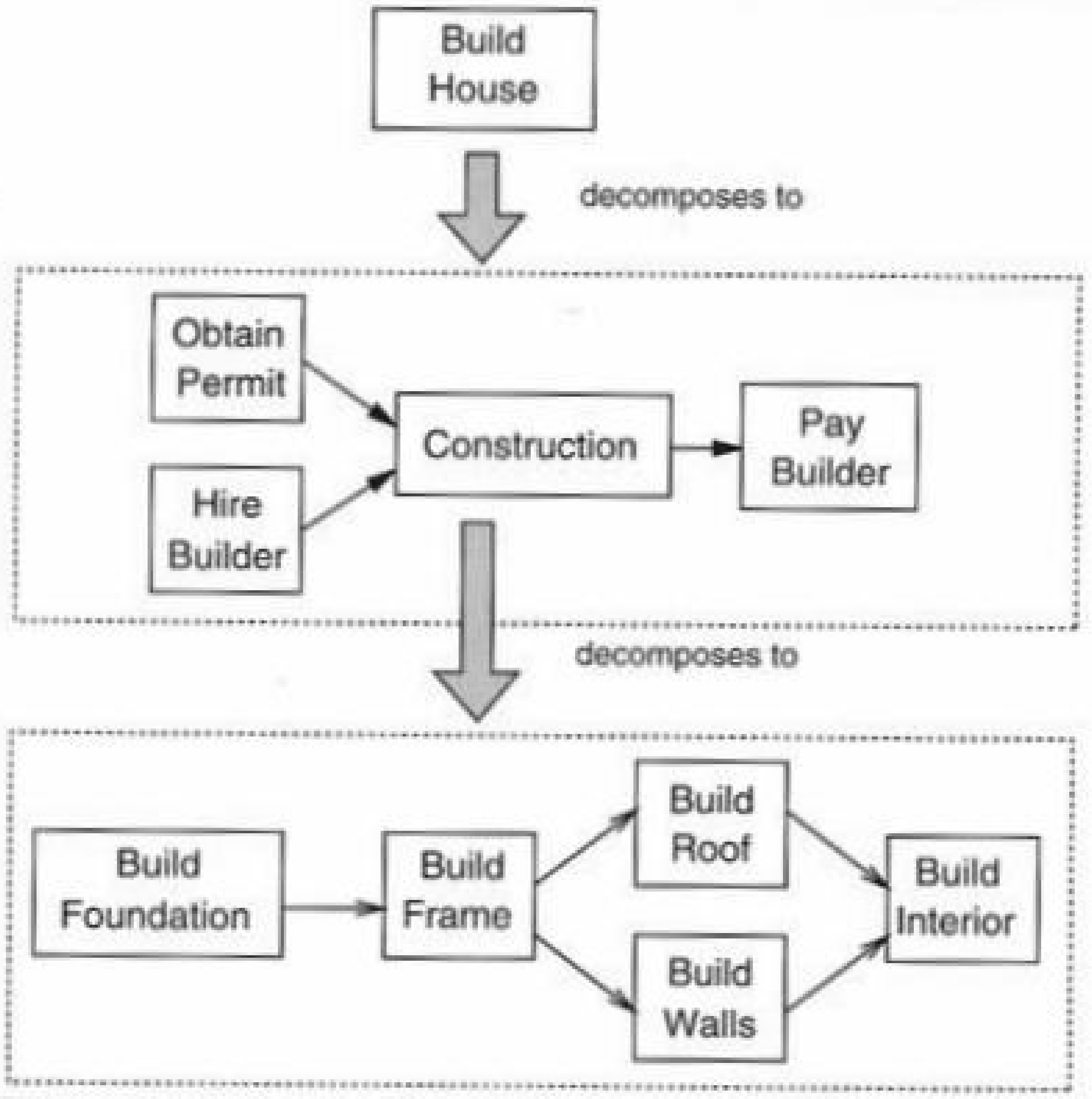


Figure 12.1 Hierarchical decomposition of the operator *Build House* into a partial-order plan.

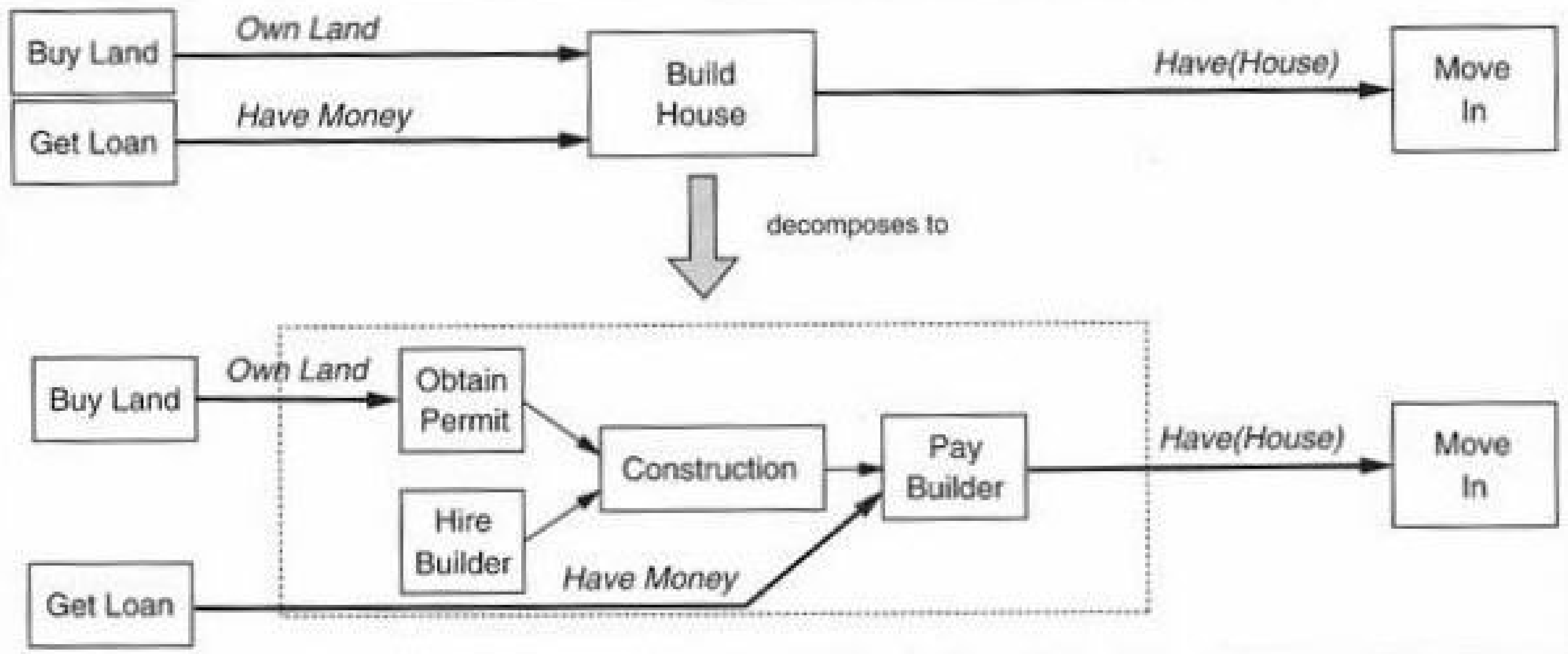


Figure 12.3 Detailed decomposition of a plan step.

3. Conditional Planning

- Conditional planning is a way to deal with uncertainty by checking what is actually happening in the environments at predetermined points in the plan.
- In this approach the agents plans first and then executes the plan that was produced.

```
function CONDITIONAL-PLANNING-AGENT(percept) returns an action
  static: KB, a knowledge base (includes action descriptions)
          p, a plan, initially NoPlan
          t, a counter, initially 0, indicating time
          G, a goal

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  current ← STATE-DESCRIPTION(KB, t)
  if p = NoPlan then p ← CPOP(current, G, KB)
  if p = NoPlan or p is empty then action ← NoOp
  else
     action ← FIRST(p)
     while CONDITIONAL?(action) do
        if ASK(KB, CONDITION-PART[action]) then p ← APPEND(THEN-PART[action], REST(p))
        else p ← APPEND(ELSE-PART[action], REST(p))
        action ← FIRST(p)
     end
     p ← REST(p)
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Figure 13.1 A conditional planning agent.

Conditional Planning in Fully Observable Domains:

- Fully Observablility means that the agent always knows the current state
- The agent handles non deterministic by building into the plan(at planning time)conditional steps that will check the state of the environment.

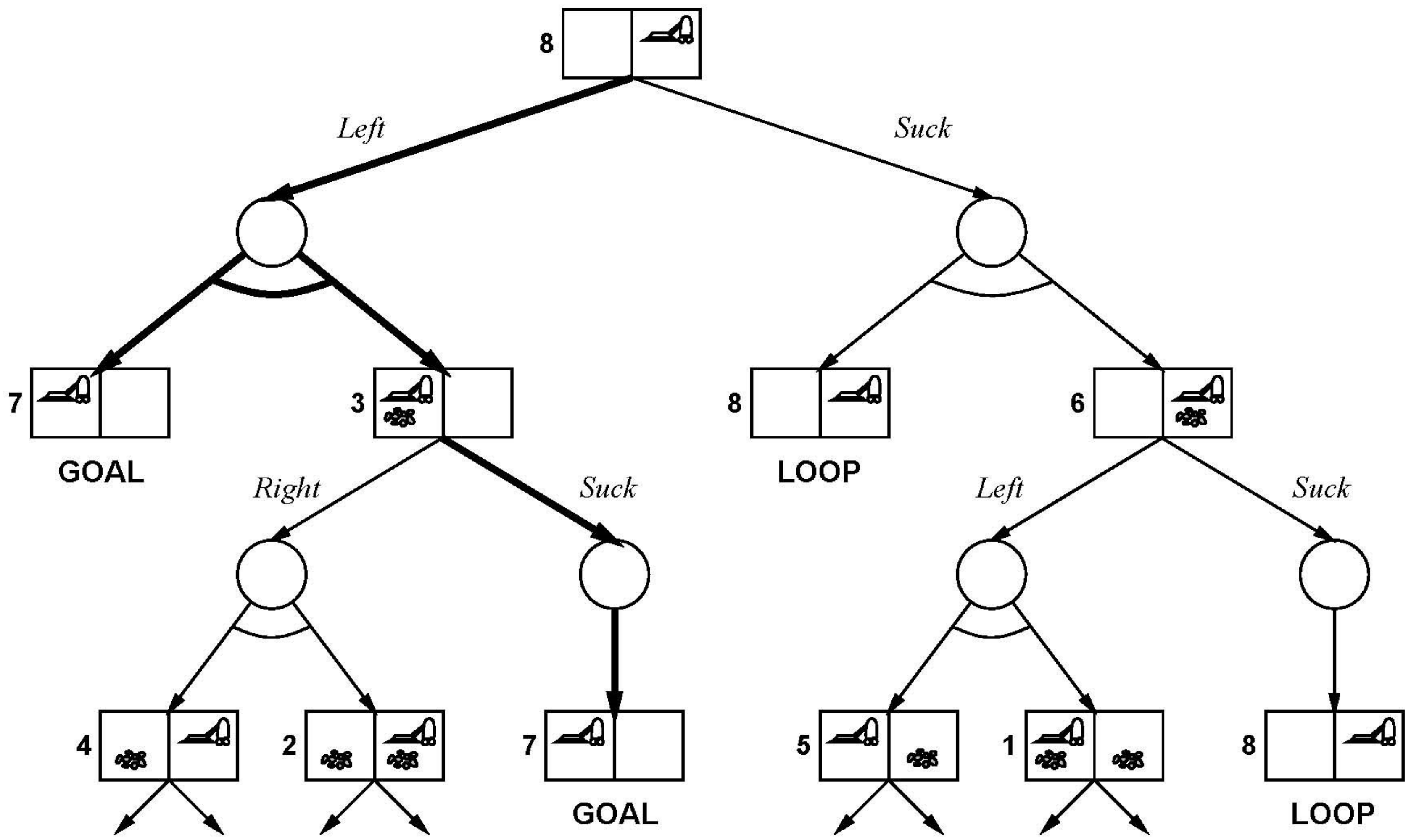
Conditional Planning in Fully Observable Domains:

- Murphy’s vacuum world with actions Left, Right, and Suck
- Plan needs to take some actions at every state and must handle every outcome for the action it takes
- State nodes (squares) and chance nodes (circles)
    - chance nodes: every outcome must be handled
    - state nodes: some action must be chosen

Notation Expantion:

- Expanding Plan Notation:
  - If (state) Then (…) else (…)
  - If (AtLeft$\wedge$CleanLeft $\wedge$CleanRight) Then {} else Suck.

State Space:

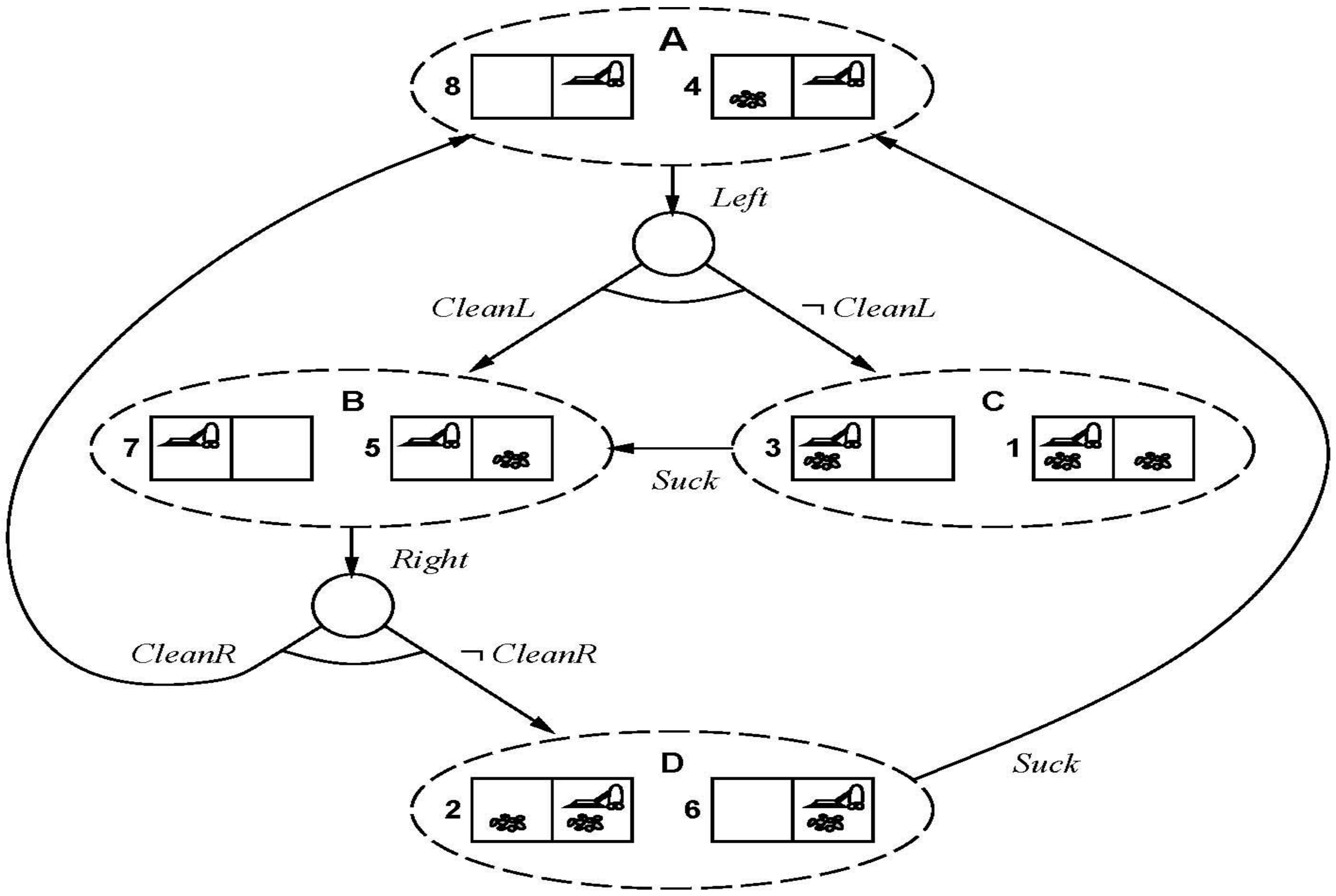

8
Left
Suck
7
GOAL
3
8
LOOP
6
Right
Suck
Left
Suck
4
2
7
GOAL
5
1
8
LOOP

Partially Observable Domains

- In fully observable environment ,we have the advantage that conditional tests can ask any question at all and be sure of getting an answer.

- In partially observable the agent knows only a certain amount about the actual state.

Partially Observable Domains



- Example-2 : The Spare Tire Problem
  - Initial State: (At(Flat, Axle) ∧ At(Spare, Trunk))
  - Goal State: (At(Spare, Axle) ∧ At(Flat, Trunk))

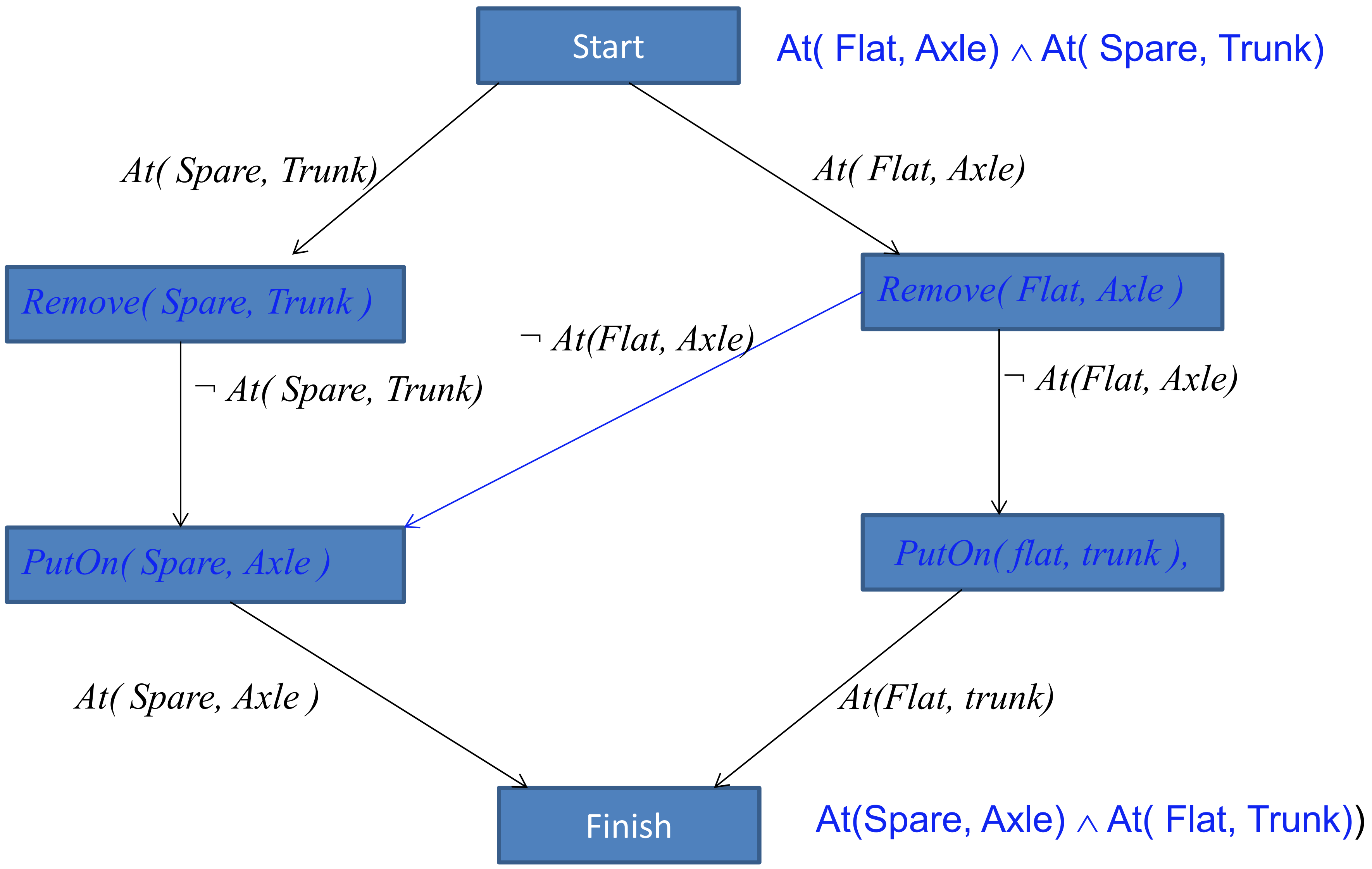
- *Action(Remove(Spare, Trunk)*
  Precond: At(Spare, Trunk)
  Effect: ¬ At(Spare, Trunk)

- *Action(Remove(Flat, Axle),*
  Precond: At(Flat, Axle)
  Effect: ¬ At(Flat, Axle)

- *Action(PutOn(Spare, Axle),*
  Precond: ¬ At(Spare, trunk)
  Effect: At(Spare, Axle))

- *Action(PutOn(flat, trunk),*
  Precond: ¬ At(Flat, Axle)
  Effect: At(Flat, trunk)

Example 2: Plan for Spare Tire Problem



Thank you