Planning

- Generate sequences of actions to perform tasks and achieve objectives.
  - States, actions and goals
- Search for solution over abstract space of plans.
- Classical planning environment: fully observable, deterministic, finite, static and discrete.
- Assists humans in practical applications
  - design and manufacturing
  - military operations
  - games
  - space exploration

Planning language

- What is a good language?
  - Expressive enough to describe a wide variety of problems.
  - Restrictive enough to allow efficient algorithms to operate on it.
  - Planning algorithm should be able to take advantage of the logical structure of the problem.
- STRIPS and ADL

General language features

- Representation of states
  - Decompose the world in logical conditions and represent a state as a *conjunction of positive literals.*
    - Propositional literals: *Poor* $\land$ *Unknown*
    - FO-literals (grounded and function-free): *At(Plane1, Melbourne)* $\land$ *At(Plane2, Sydney)*
  - Closed world assumption
- Representation of goals
  - Partially specified state and represented as a *conjunction of literals*
  - A goal is *satisfied* if the state contains all literals in goal.

General language features

- Representations of actions
  - Action = PRECOND + EFFECT

    Action(Fly(p,from, to),

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

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

  = action schema (p, from, to need to be instantiated)
    - Action name and parameter list
    - Precondition (conj. of function-free literals)
    - Effect (conj of function-free literals and P is True and not P is false)
  - Add-list vs delete-list in Effect

Language semantics?

- How do actions affect states?
  - An action is applicable in any state that satisfies the precondition.
  - For FO action schema applicability involves a substitution θ for the variables in the PRECOND.

    At(P1,JFK) $\wedge$ *At(P2,SFO)* $\wedge$ *Plane(P1)* $\wedge$ *Plane(P2)* $\wedge$ *Airport(JFK)* $\wedge$ *Airport(SFO)*

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

    With θ =\{*p/P1,from/JFK,to/SFO*\}

    Thus the action is applicable.

Language semantics?

- The result of executing action a in state s is the state s'
  - s' is same as s except
    - Any positive literal *P* in the effect of *a* is added to *s'*
    - Any negative literal ¬*P* is removed from *s'*

    EFFECT: ¬AT(p,from) ∧ At(p,to):

    At(P1,SFO) ∧ At(P2,SFO) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO)

  - STRIPS assumption: *every literal NOT in the effect remains unchanged*

Expressiveness and extensions

- STRIPS is simplified
  - Important limit: function-free literals
    - Allows for propositional representation
    - Function symbols lead to infinitely many states and actions
- Recent extension:Action Description language (ADL)

    Action(Fly(p:Plane, from: Airport, to: Airport),
    PRECOND: At(p,from) $\wedge$ *(from* $\neq$ *to)*
    EFFECT: ¬At(p,from) $\wedge$ *At(p,to))*

Standardization *: Planning domain definition language (PDDL)*

STRIPS representations

- States: conjunctions of ground literals
  - In(robot, r_3) Æ Closed($door_6$) Æ …
- Goals: conjunctions of literals
  - (implicit ∃ r) In(Robot, r) Æ In(Charger, r)
- Actions (operators)
  - Name (implicit ∀):   Go(here, there)
  - Preconditions: conjunction of literals
    - At(here) Æ path(here, there)
  - Effects: conjunctions of literals [also known as post-conditions, add-list, delete-list]
    - At(there) Æ ¬ At(here)
  - Assumes no inference in relating predicates (only equality)

Strips Example

- Action
  - 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)

Example: air cargo transport

Init(At(C1, SFO) ∧ At(C2,JFK) ∧ At(P1,SFO) ∧ At(P2,JFK) ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO))

Goal(At(C1,JFK) ∧ At(C2,SFO))

Action(Load(c,p,a)

PRECOND: *At(c,a) ∧At(p,a) ∧Cargo(c) ∧Plane(p) ∧Airport(a)*

EFFECT: ¬*At(c,a) ∧In(c,p))*

Action(Unload(c,p,a)

PRECOND: *In(c,p) ∧At(p,a) ∧Cargo(c) ∧Plane(p) ∧Airport(a)*

EFFECT: *At(c,a) ∧ ¬In(c,p))*

Action(Fly(p,from,to)

PRECOND: *At(p,from) ∧Plane(p) ∧Airport(from) ∧Airport(to)*

EFFECT: ¬ *At(p,from) ∧ At(p,to))*

[Load(C1,P1,SFO), Fly(P1,SFO,JFK), Load(C2,P2,JFK), Fly(P2,JFK,SFO)]

Example: Spare tire problem

Init(At(Flat, Axle) $\wedge$ *At(Spare,trunk))*

Goal(At(Spare,Axle))

Action(Remove(Spare,Trunk)

PRECOND: *At(Spare,Trunk)*

EFFECT: ¬*At(Spare,Trunk)* $\wedge$ *At(Spare,Ground))*

Action(Remove(Flat,Axle)

PRECOND: *At(Flat,Axle)*

EFFECT: ¬*At(Flat,Axle)* $\wedge$ *At(Flat,Ground))*

Action(PutOn(Spare,Axle)

PRECOND: *At(Spare,Groundp)* $\wedge$¬*At(Flat,Axle)*

EFFECT: *At(Spare,Axle)* $\wedge$ ¬*At(Spare,Ground))*

Action(LeaveOvernight

PRECOND:

EFFECT: ¬ *At(Spare,Ground)* $\wedge$ ¬ *At(Spare,Axle)* $\wedge$ ¬ *At(Spare,trunk)* $\wedge$ ¬ *At(Flat,Ground)* $\wedge$ ¬ *At(Flat,Axle))*

Example: Blocks world

Init(On(A, Table) $\wedge$ *On(B,Table)* $\wedge$ *On(C,Table)* $\wedge$ *Block(A)* $\wedge$ *Block(B)* $\wedge$ *Block(C)* $\wedge$ *Clear(A)* $\wedge$ *Clear(B)* $\wedge$ *Clear(C))*

Goal(On(A,B) $\wedge$ *On(B,C))*

Action(Move(b,x,y)

PRECOND: *On(b,x)* $\wedge$ *Clear(b)* $\wedge$ *Clear(y)* $\wedge$ *Block(b)* $\wedge$ *(b*$\neq$ *x)* $\wedge$ *(b*$\neq$ *y)* $\wedge$ *(x*$\neq$ *y)*

EFFECT: *On(b,y)* $\wedge$ *Clear(x)* $\wedge$ $\neg$ *On(b,x)* $\wedge$ $\neg$ *Clear(y))*

Action(MoveToTable(b,x)

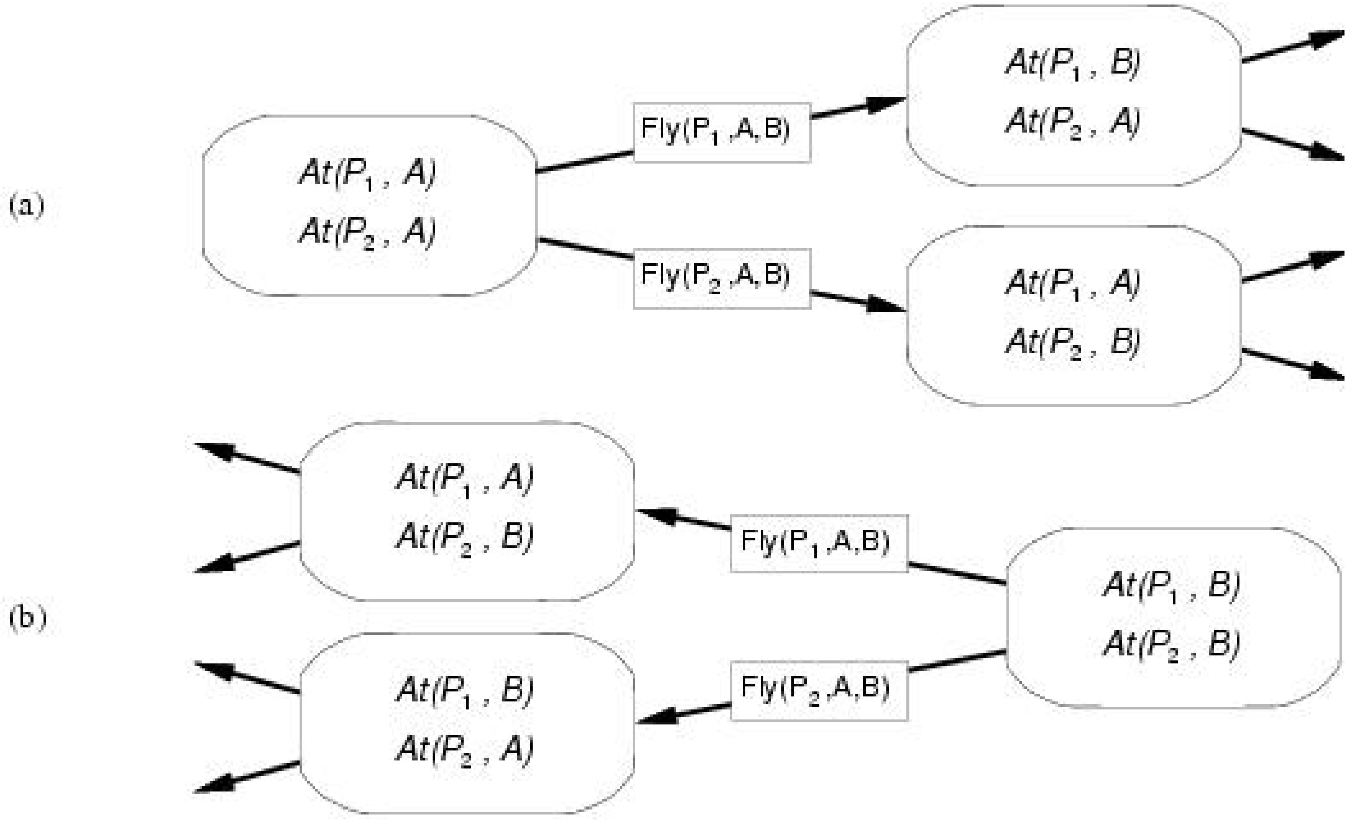
PRECOND: *On(b,x)* $\wedge$ *Clear(b)* $\wedge$ *Block(b)* $\wedge$ *(b*$\neq$ *x)*

EFFECT: *On(b,Table)* $\wedge$ *Clear(x)* $\wedge$ $\neg$ *On(b,x))*

Planning with state-space search

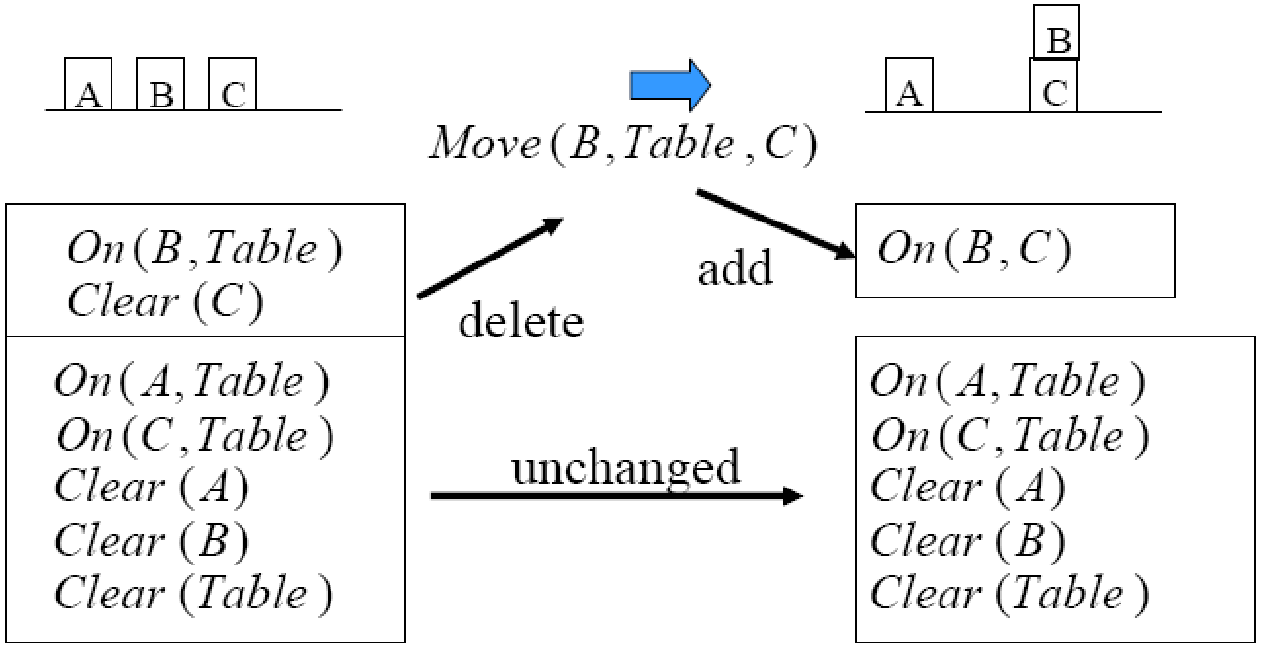
- Both forward and backward search possible
- Progression planners
  - forward state-space search
  - Consider the effect of all possible actions in a given state
- Regression planners
  - backward state-space search
  - To achieve a goal, what must have been true in the previous state.

Progression and regression



Operator: *Move (x,y,z)*

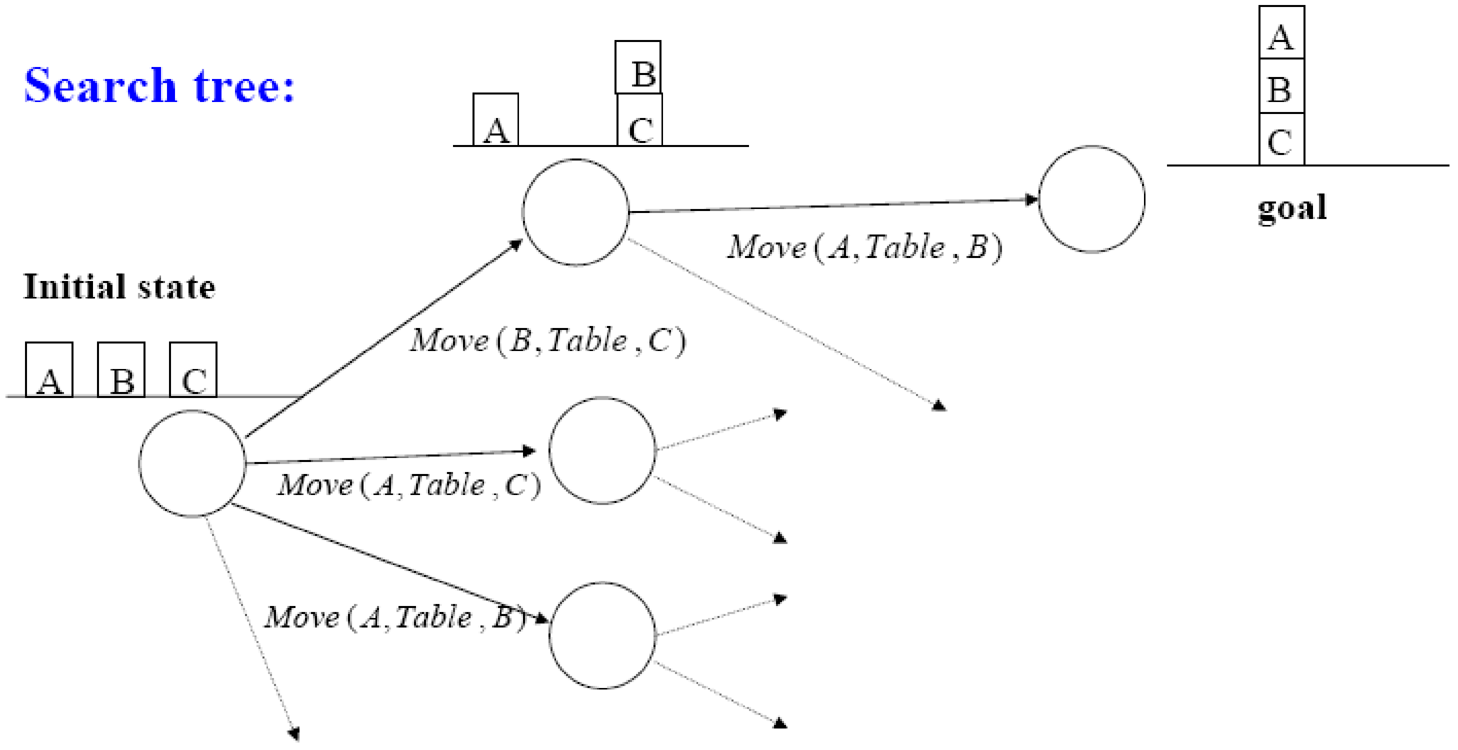
- **Preconditions:** *On(x,y), Clear(x), Clear(z)*
- **Add list:** *On(x,z), Clear(y)*
- **Delete list:** *On(x,y), Clear(z)*



Forward search (goal progression)

- Use operators to generate new states to search
- Check new states whether they satisfy the goal

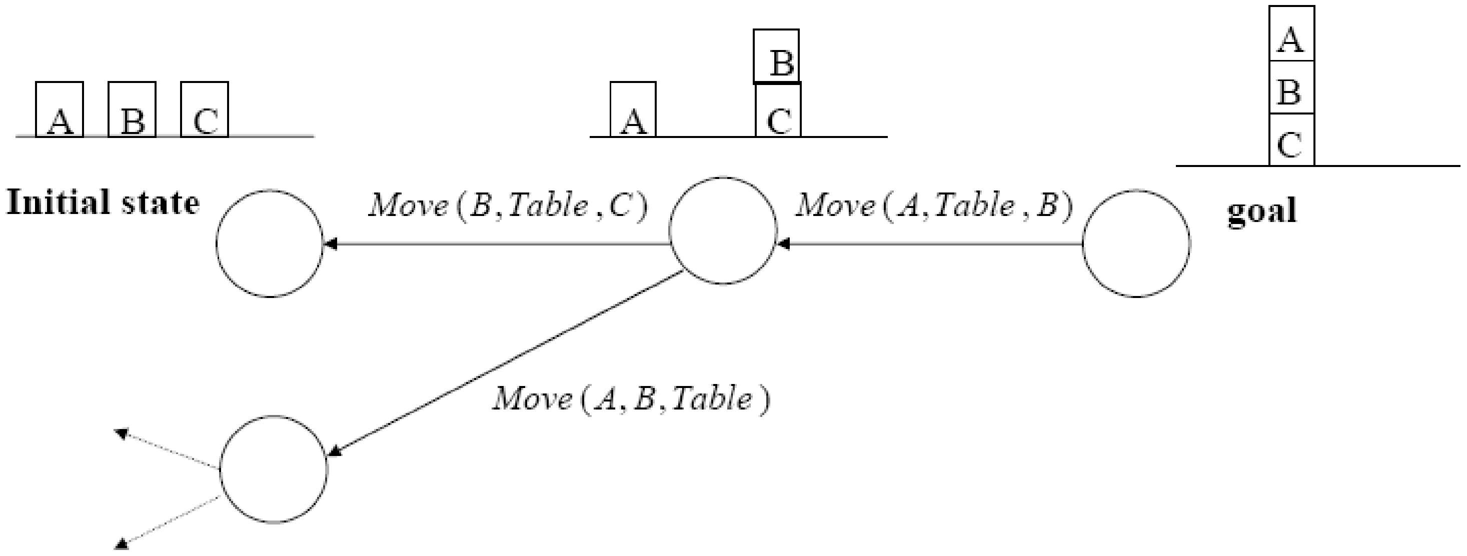
Search tree:



Backward search (goal regression)

- Use operators to generate new goals
- Check whether the initial state satisfies the goal

Search tree:



Progression algorithm

- Formulation as state-space search problem:
  - Initial state = initial state of the planning problem
    - Literals not appearing are false
  - Actions = those whose preconditions are satisfied
    - Add positive effects, delete negative
  - Goal test = does the state satisfy the goal
  - Step cost = each action costs 1
- No functions … any graph search that is complete is a complete planning algorithm.
  - E.g. A*
- Inefficient:
  - (1) irrelevant action problem
  - (2) good heuristic required for efficient search

Regression algorithm

- How to determine predecessors?
  - What are the states from which applying a given action leads to the goal?
    - Goal state = *At(C1, B)* $\wedge$ *At(C2, B)* $\wedge$ … $\wedge$ *At(C20, B)*
    - Relevant action for first conjunct*: Unload(C1,p,B)*
    - Works only if pre-conditions are satisfied.
    - Previous state= *In(C1, p)* $\wedge$ *At(p, B)* $\wedge$ *At(C2, B)* $\wedge$ … $\wedge$ *At(C20, B)*
    - Subgoal At(C1,B) should not be present in this state.
- Actions must not undo desired literals (consistent)
- Main advantage: only relevant actions are considered.
  - Often much lower branching factor than forward search.

Regression algorithm

- General process for predecessor construction
  - Give a goal description G
  - Let A be an action that is relevant and consistent
  - The predecessors is as follows:
    - Any positive effects of A that appear in G are deleted.
    - Each precondition literal of A is added , unless it already appears.
- Any standard search algorithm can be added to perform the search.
- Termination when predecessor satisfied by initial state.
  - In FO case, satisfaction might require a substitution.

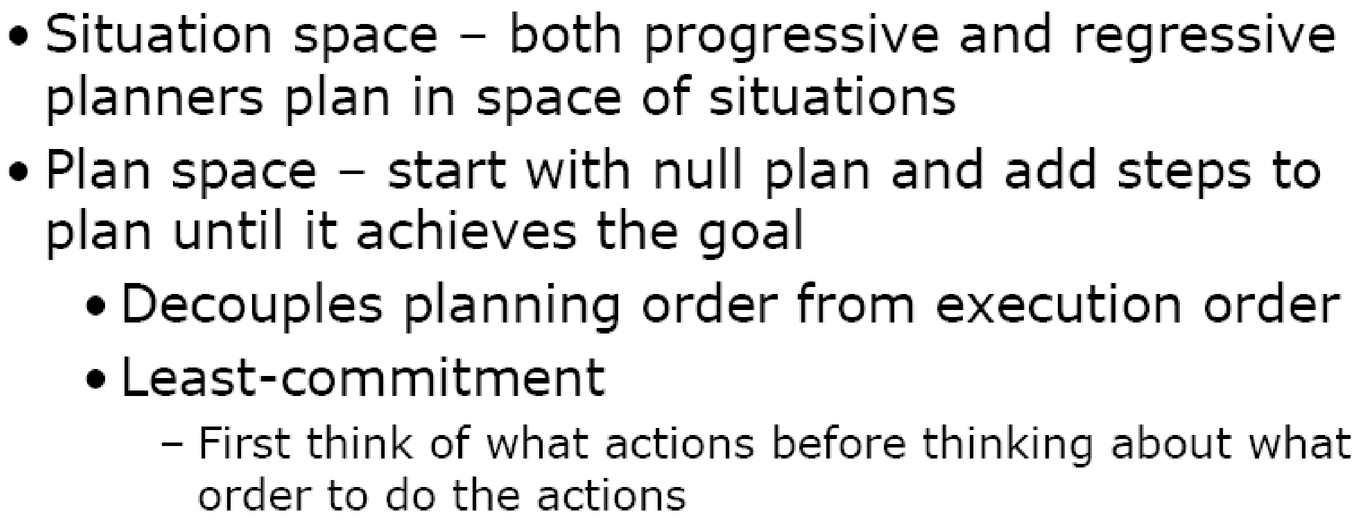
Heuristics for state-space search

- Neither progression or regression are very efficient without a good heuristic.
  - How many actions are needed to achieve the goal?
  - Exact solution is NP hard, find a good estimate
- Two approaches to find admissible heuristic:
  - The optimal solution to the relaxed problem.
    - Remove all preconditions from actions
  - The subgoal independence assumption:

    The cost of solving a conjunction of subgoals is approximated by the sum of the costs of solving the subproblems independently.

Partial-order Planning

- Progression and regression planning are *totally ordered plan search* forms.
  - They cannot take advantage of problem decomposition.
  - Decisions must be made on how to sequence actions on all the subproblems



- Situation space – both progressive and regressive planners plan in space of situations
- Plan space – start with null plan and add steps to plan until it achieves the goal
  - Decouples planning order from execution order
  - Least-commitment
    - First think of what actions before thinking about what order to do the actions

Shoe example

Goal(RightShoeOn $\wedge$ LeftShoeOn)

Init()

Action(RightShoe, PRECOND: RightSockOn
  EFFECT: RightShoeOn)

Action(RightSock, PRECOND:
  EFFECT: RightSockOn)

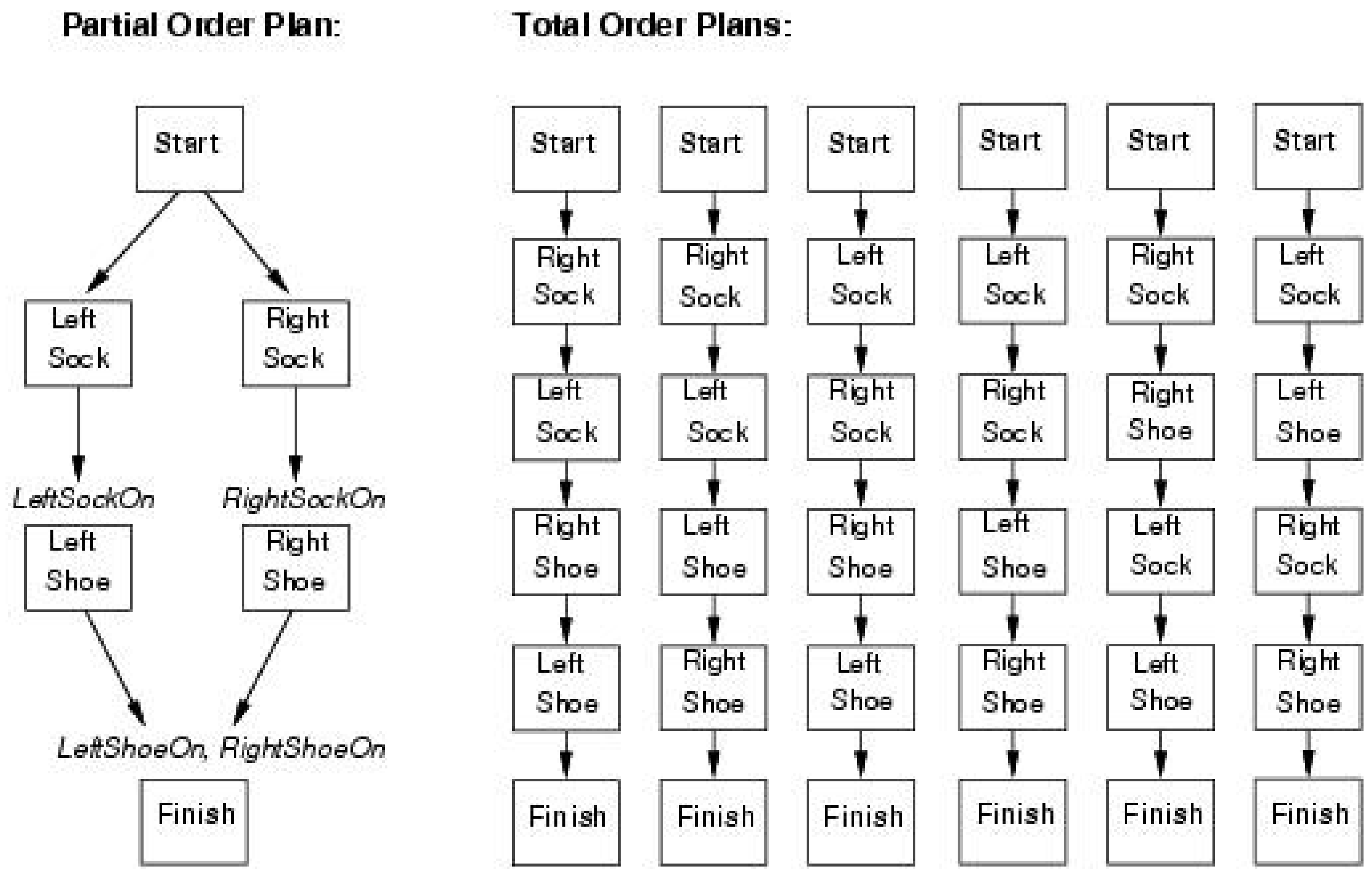
Action(LeftShoe, PRECOND: LeftSockOn
  EFFECT: LeftShoeOn)

Action(LeftSock, PRECOND:
  EFFECT: LeftSockOn)

Planner: combine two action sequences (1)leftsock, leftshoe (2)rightsock, rightshoe

Partial-order planning(POP)

- Any planning algorithm that can place two actions into a plan without which comes first is a PO plan.



Partially Ordered Plan

- Set of steps (instance of an operator)
- Set of ordering constraints $S_i < S_j$
- Set of variable binding constraints $v=x$
    - v is a variable in a step; x is a constant or another variable
- Set of causal links $S_i \square_c S_j$
    - Step i achieves precondition c for step j

Initial Plan

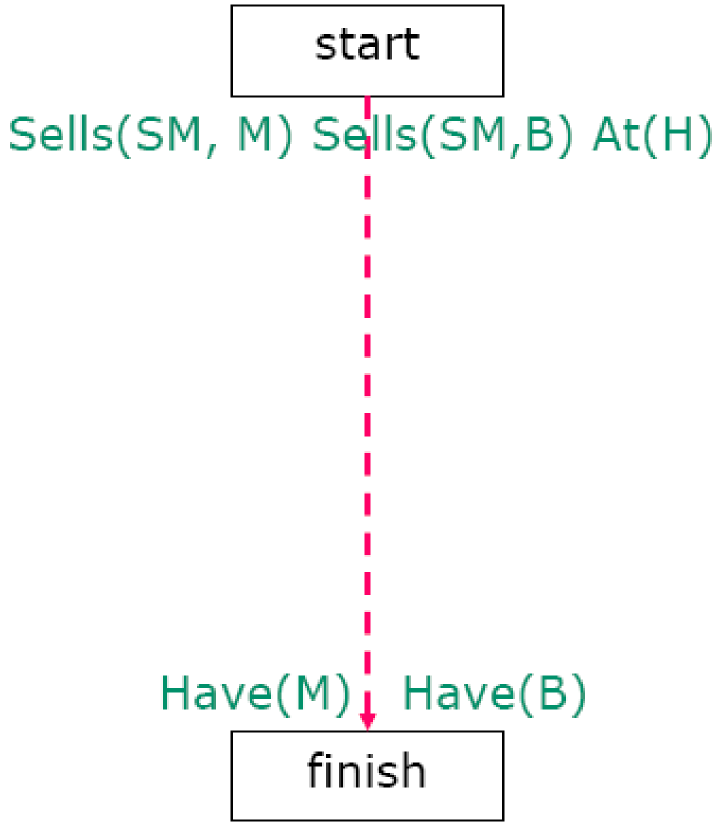
- Steps: {start, finish}
- Ordering: {start < finish}

- start
  - Pre: none
  - Effects: start conditions
- finish
  - Pre: goal conditions
  - Effects: none

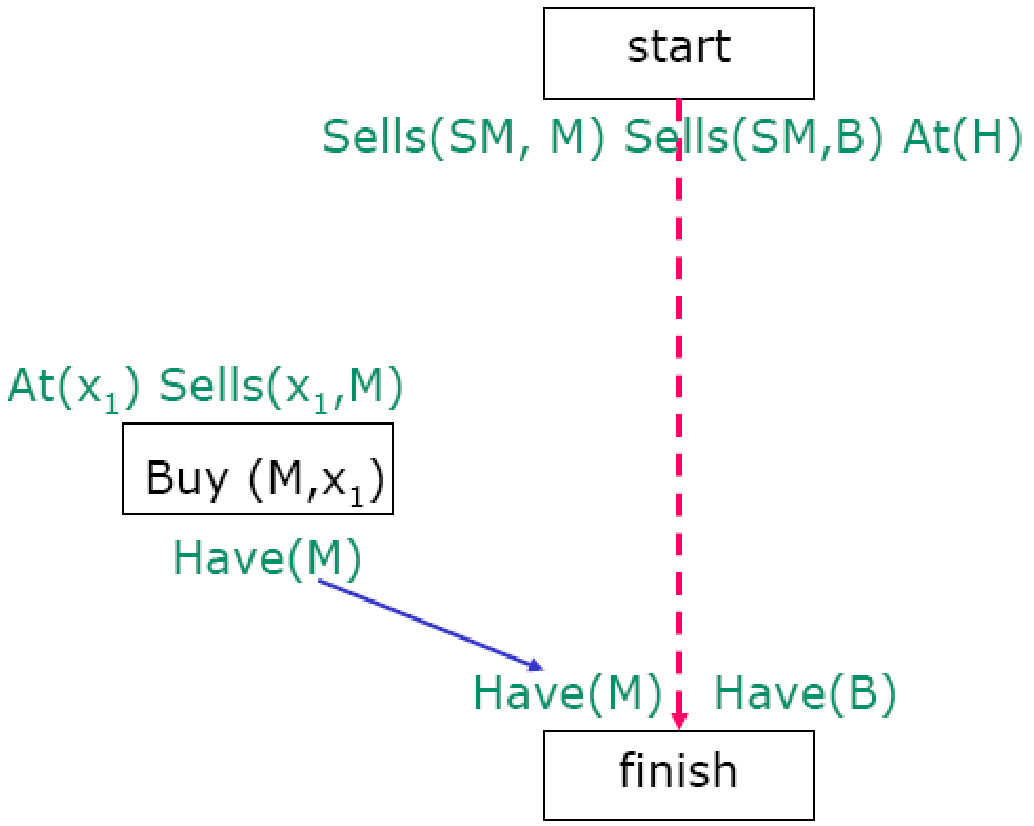
Plan Completeness

- A plan is complete iff every precondition of every step is achieved by some other step.
- $S_i \square_c S_j$ ("step I achieves c for step j") iff
  - $S_i < S_j$
  - $c \in \text{effects}(S_i)$
  - $\neg \exists S_k. \neg c \in \text{effects}(S_k)$ and $S_i < S_k < S_j$ is consistent with the ordering constraints
- A plan is consistent iff the ordering constraints are consistent and the variable binding constraints are consistent.

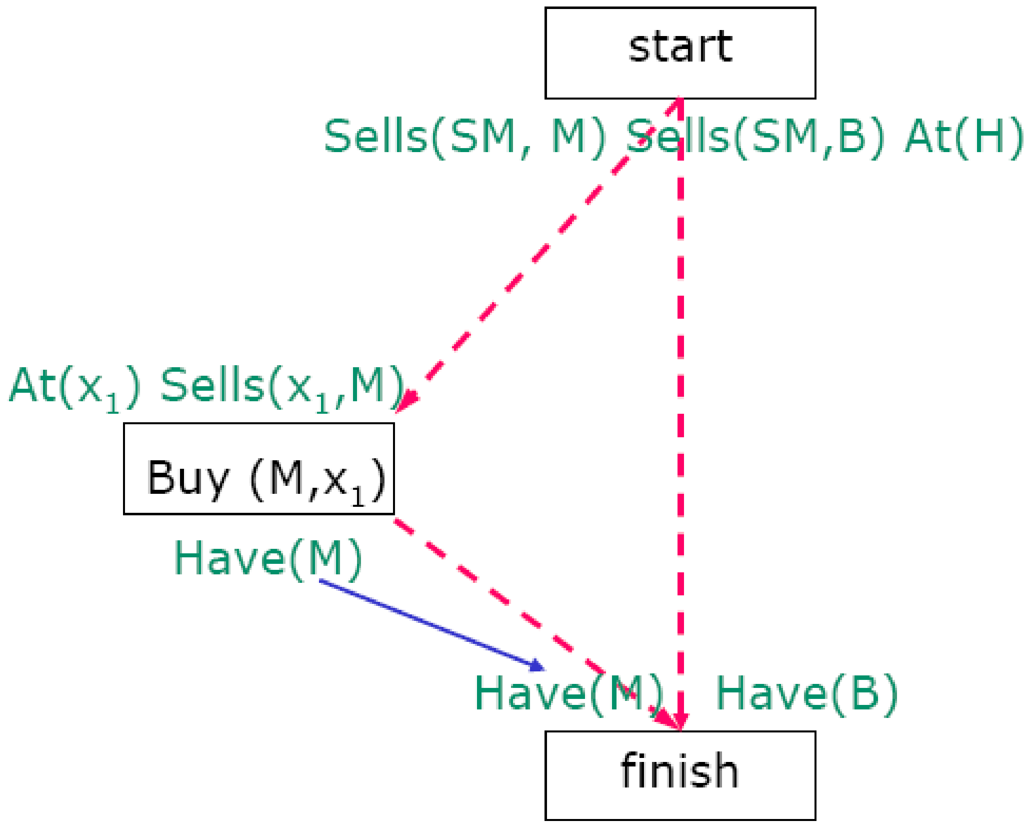
PO Plan Example



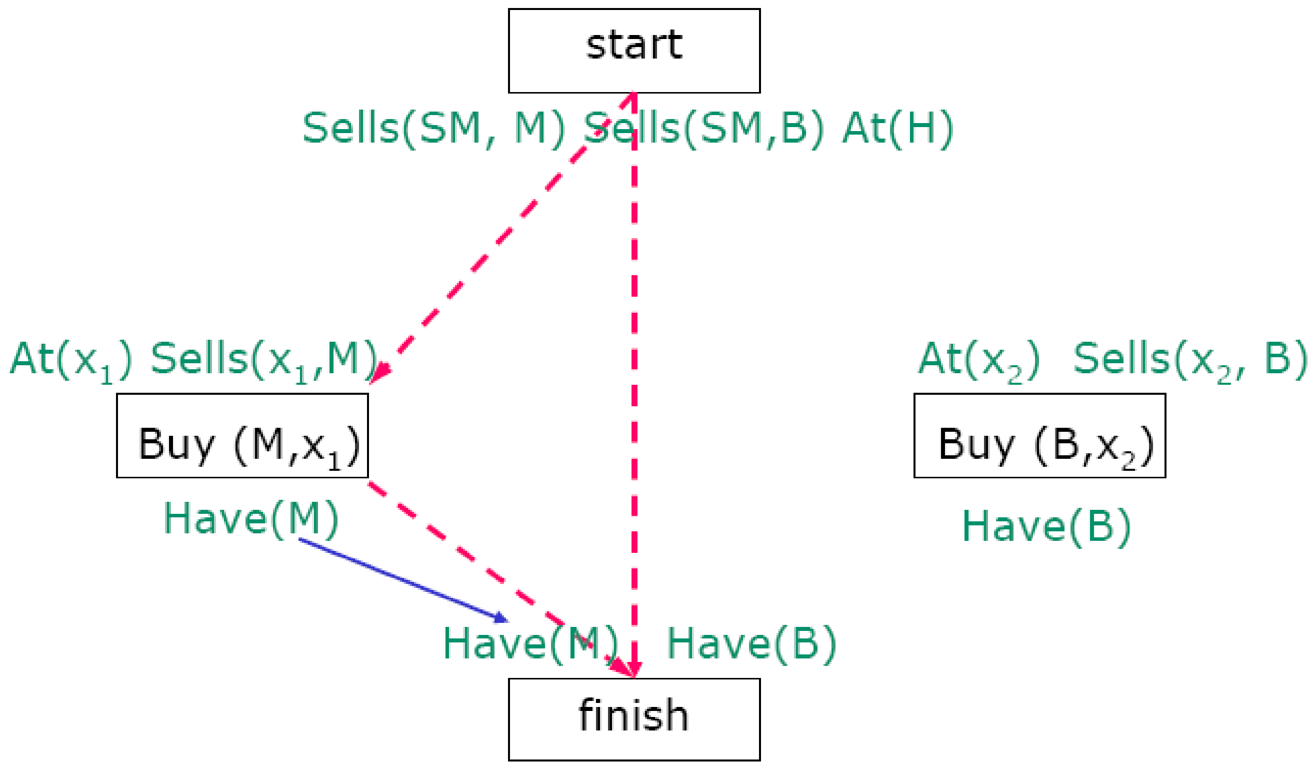
PO Plan Example



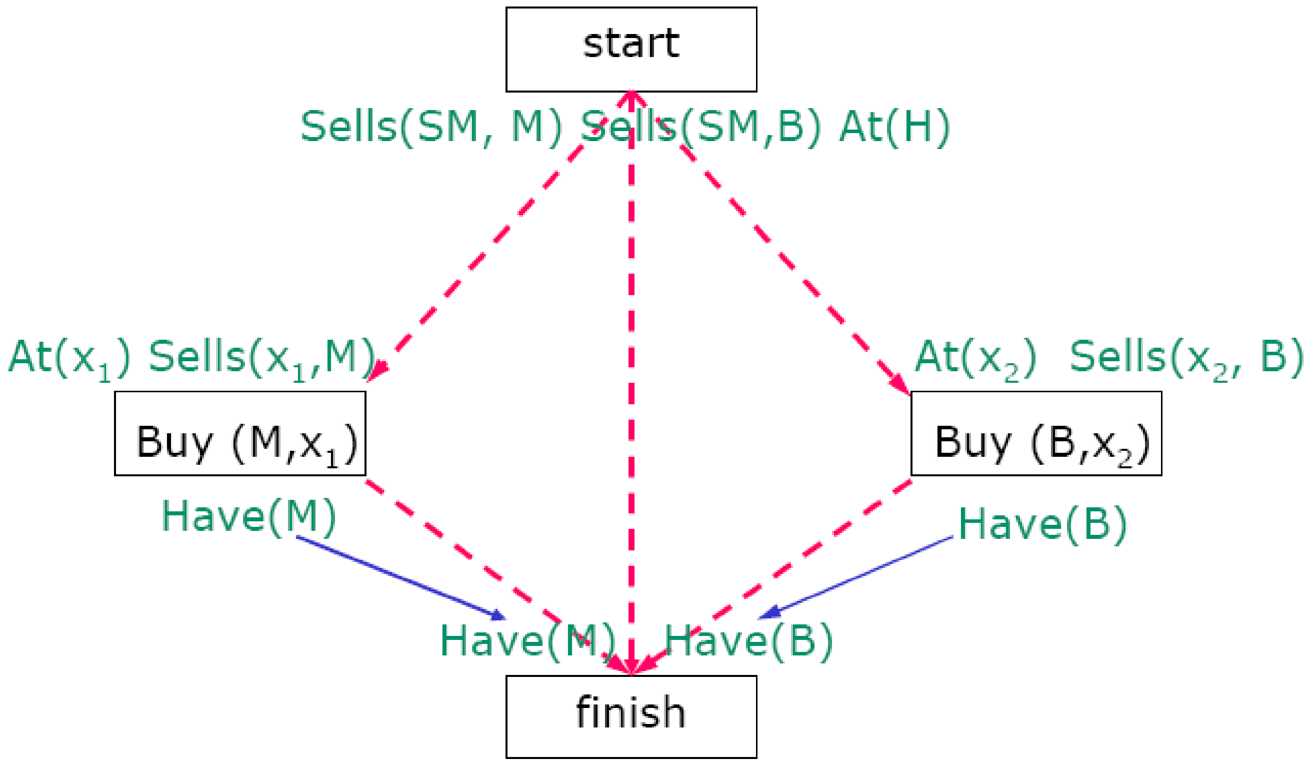
PO Plan Example



PO Plan Example

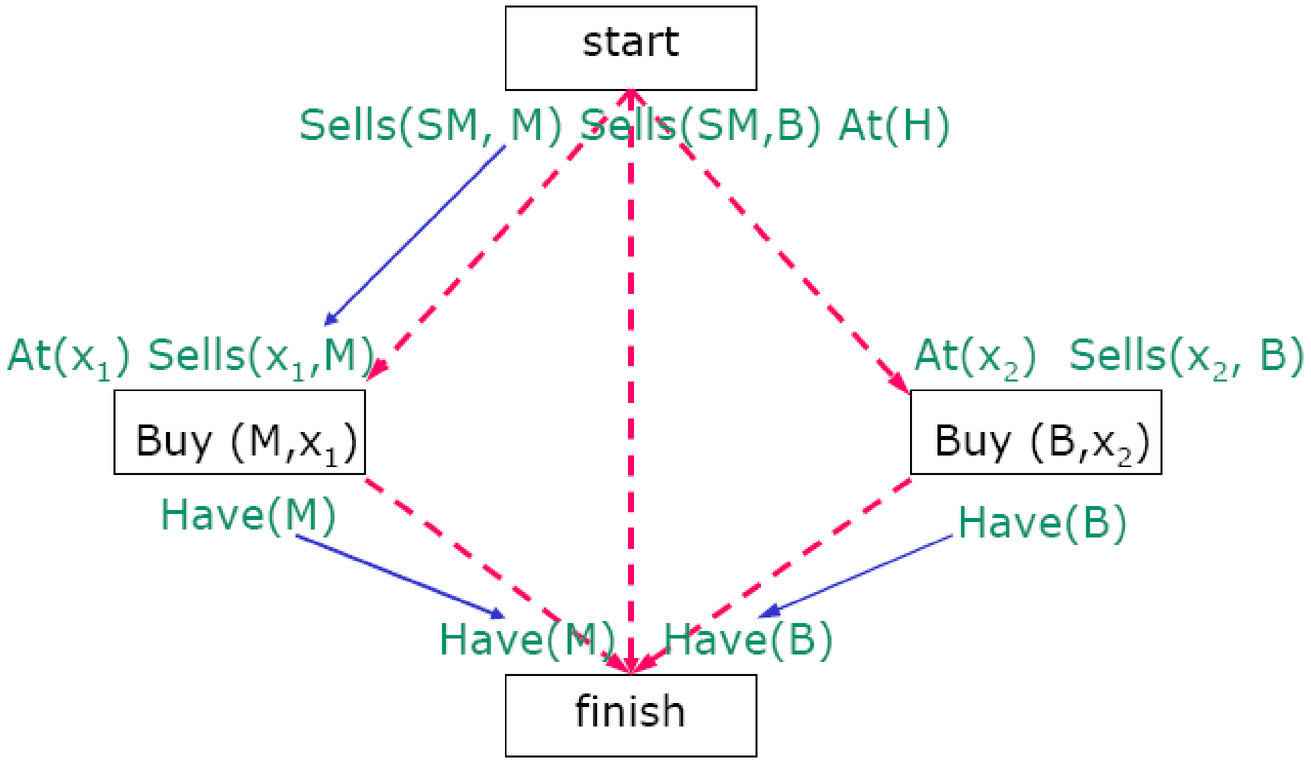


PO Plan Example



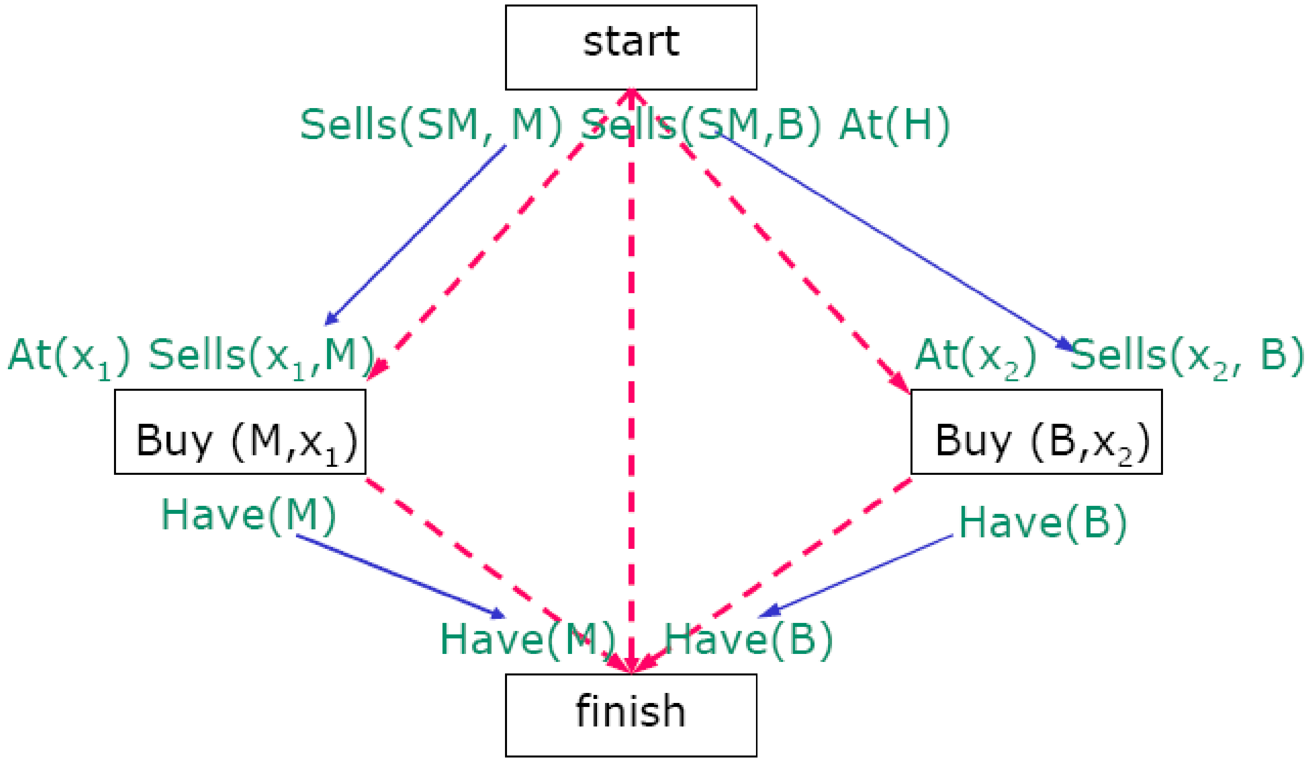
PO Plan Example

x_1 = SM



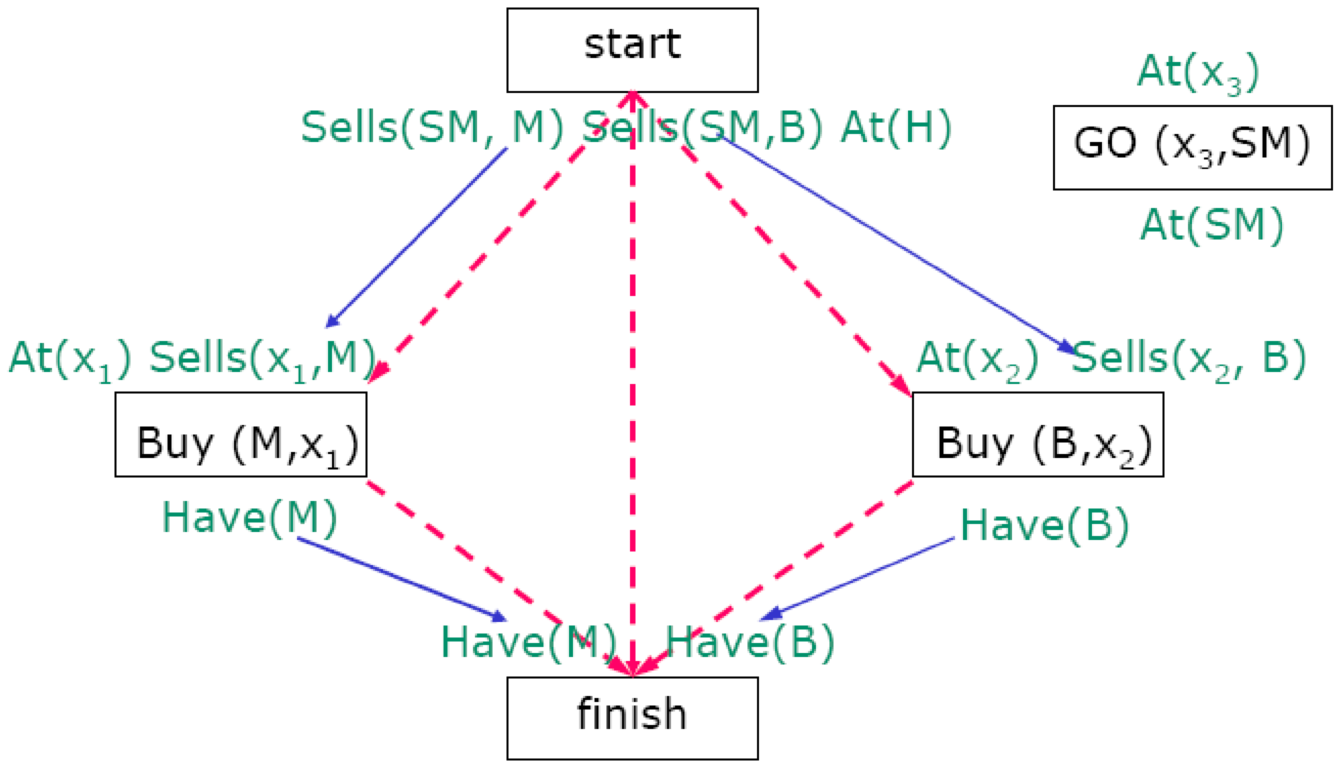
PO Plan Example

x_1 = SM
x_2 = SM



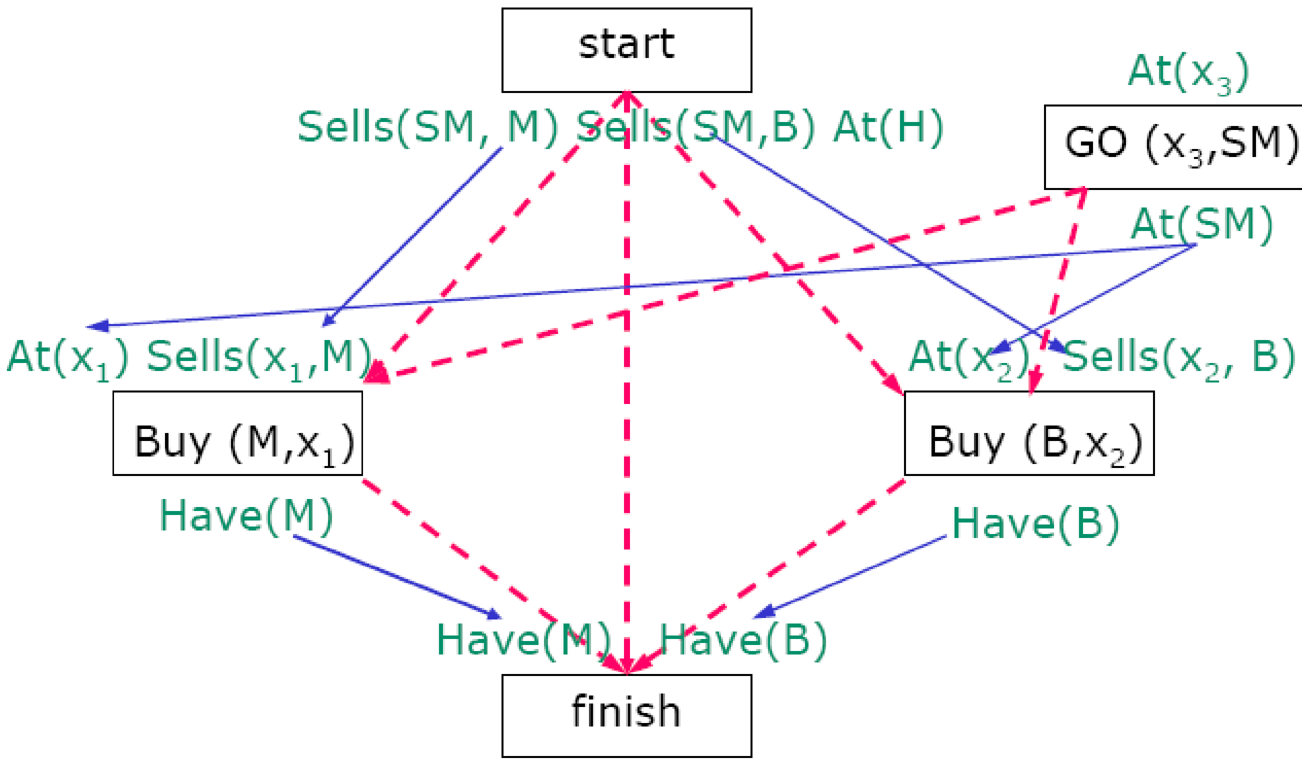
PO Plan Example

x_1 = SM
x_2 = SM



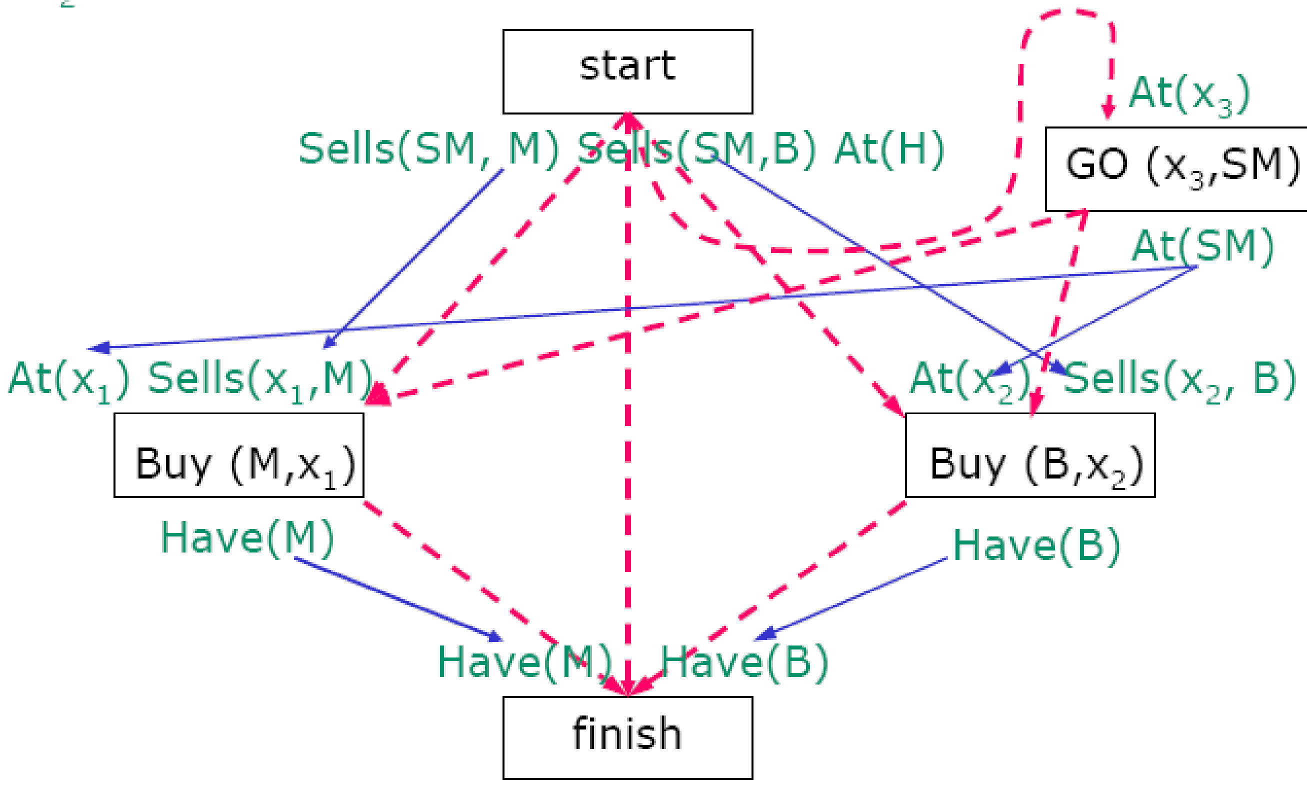
PO Plan Example

x_1 = SM
x_2 = SM



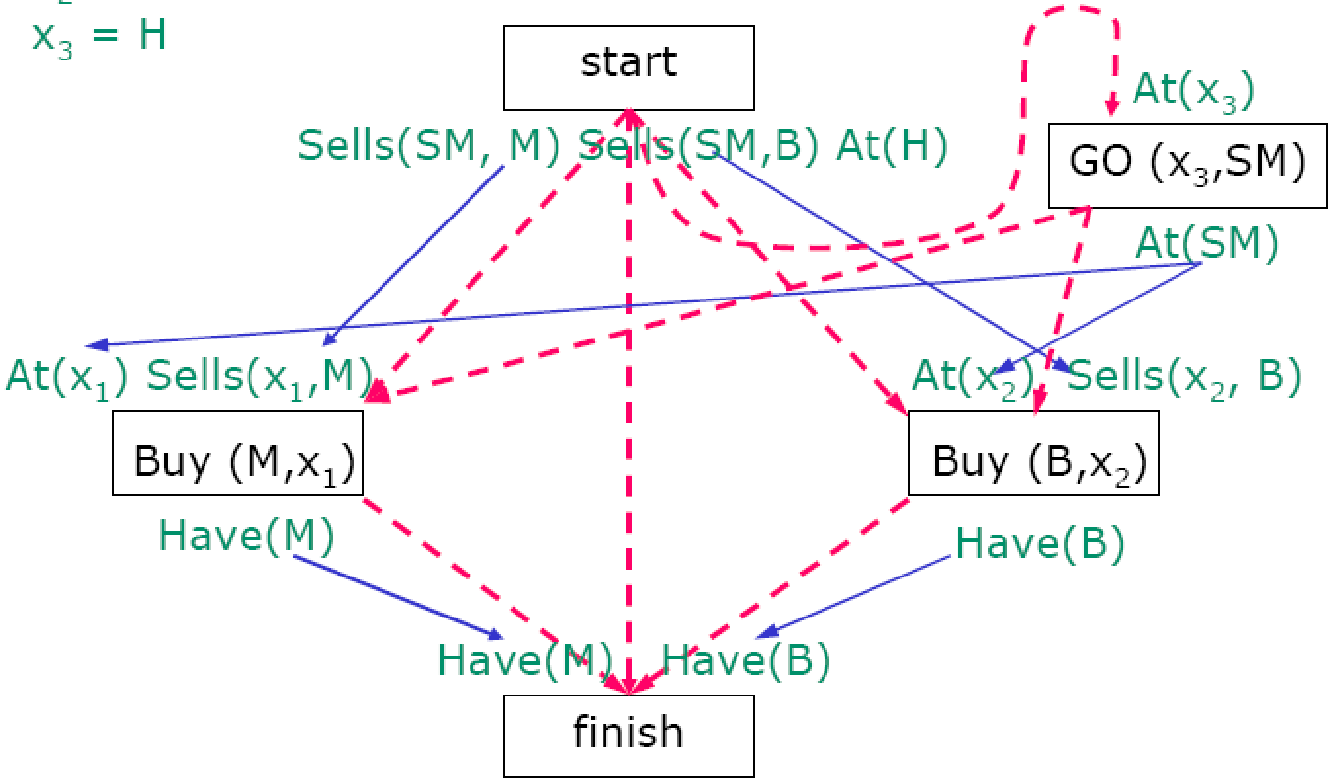
PO Plan Example

x_1 = SM
x_2 = SM



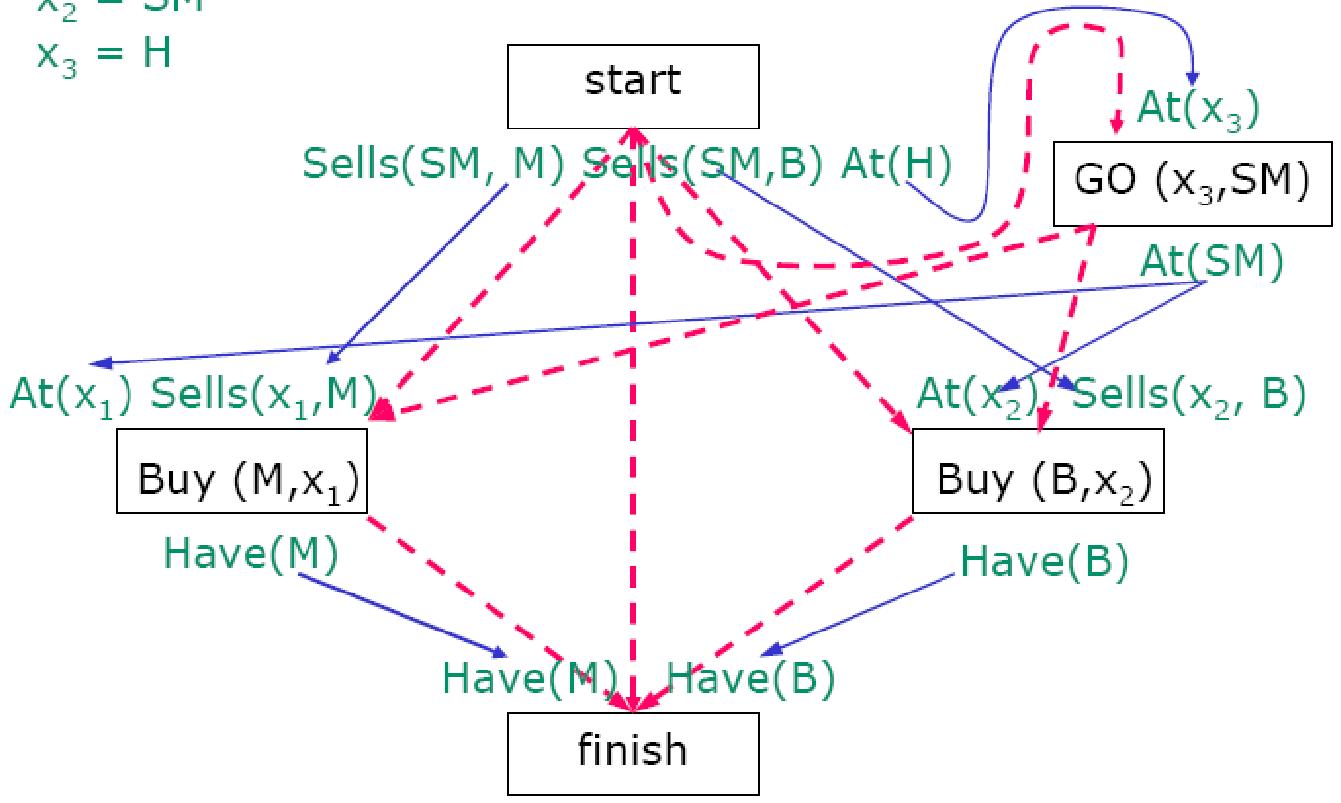
PO Plan Example

x_1 = SM
x_2 = SM
x_3 = H



PO Plan Example

x_1 = SM
x_2 = SM
x_3 = H


start
Sells(SM, M) Sells(SM,B) At(H)
At(x_3)
GO (x_3,SM)
At(SM)
At(x_1) Sells(x_1,M)
At(x_2) Sells(x_2, B)
Buy (M,x_1)
Buy (B,x_2)
Have(M)
Have(B)
Have(M) Have(B)
finish