Introduction to Planning

What is Planning?

- Planning is a search problem that requires to find an efficient sequence of actions that transform a system from a given starting state to the goal state.

- Planning is an activity where agent has to come up with a sequence of actions to accomplish a target.

What's Given?

- Initial state of the problem
- Goal state of the problem
- A finite set of actions:
  - pre-conditions: a finite set of conditions for the action to be performed
  - post-conditions: a finite set of conditions that will be changed after the action is performed

What's Output?

- A sequence of actions that meet the following criteria
  - every action matches the current system state
  - can transform system from initial state to goal state
  - the total cost of the actions is below a specified value

Typical assumptions

- Atomic time: Each action is indivisible
- No concurrent actions allowed, but actions need not be ordered w.r.t each other in the plan
- Deterministic actions: action results completely determined — no uncertainty in their effects
- Agent is the sole cause of change in the world
- Agent is omniscient with complete knowledge of the state of the world
- Closed world assumption where everything known to be true in the world is included in the state description and anything not listed is false
- Fully Obervable, Deterministic, Finite, Static, Discrete

Planning problem

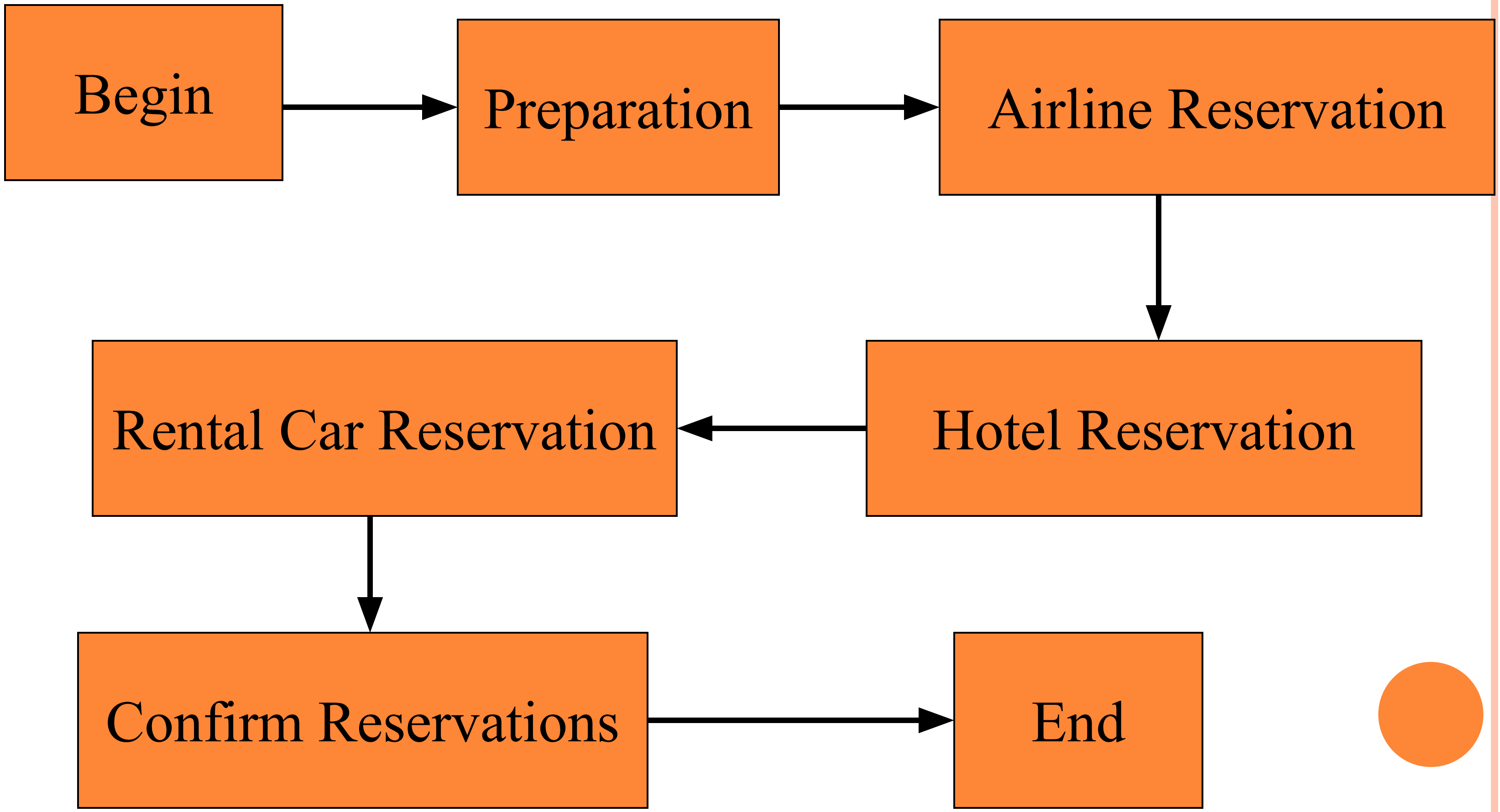
- Find a **sequence of actions** that achieves a given **goal** when executed from a given **initial world state**
- That is, given
  - a set of *operator descriptions* defining the possible primitive actions by the agent,
  - an *initial state* description, and
  - a *goal state* description or predicate,

  compute a plan, which is
  - a sequence of operator instances which after executing them in the initial state changes the world to a goal state
- Goals are usually specified as a conjunction of goals to be achieved

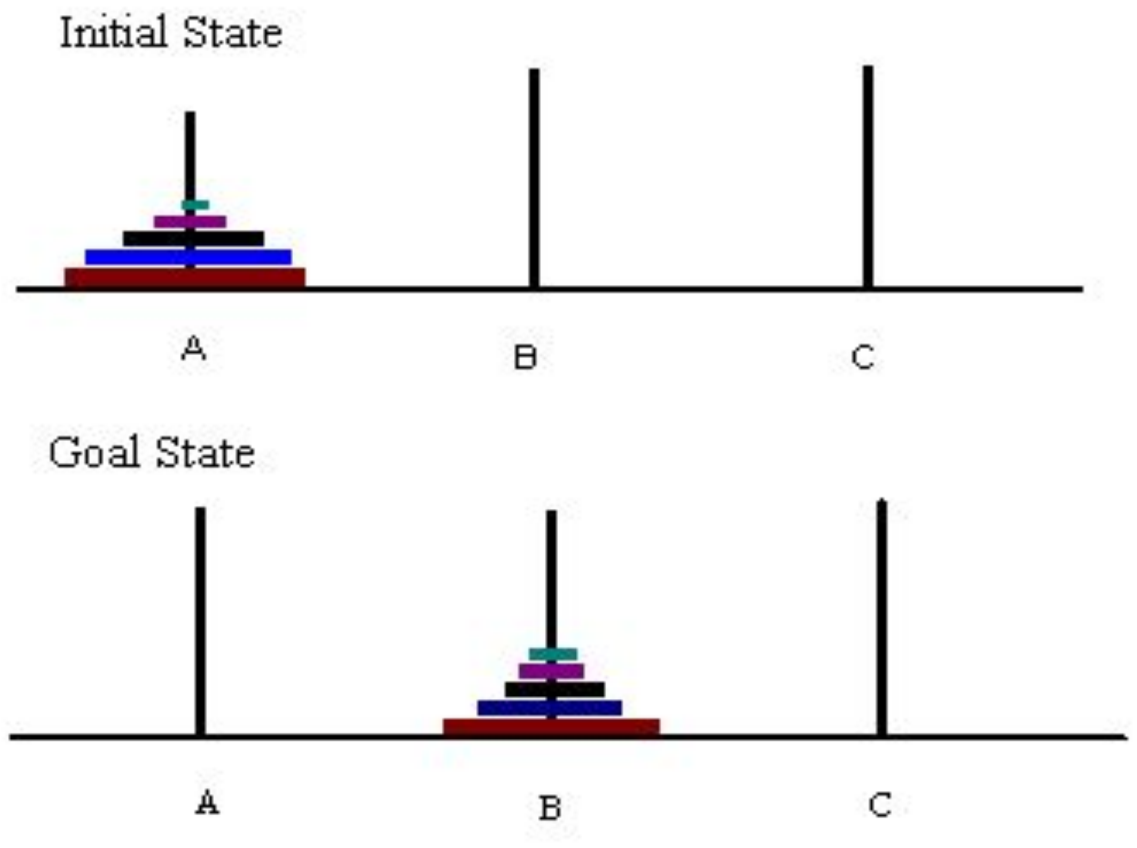
Planning as a Real-World Problem

- Planning problem has a wide range of applications in the real world
  - planning in daily life
  - game world
  - workflow management

Planning a Trip



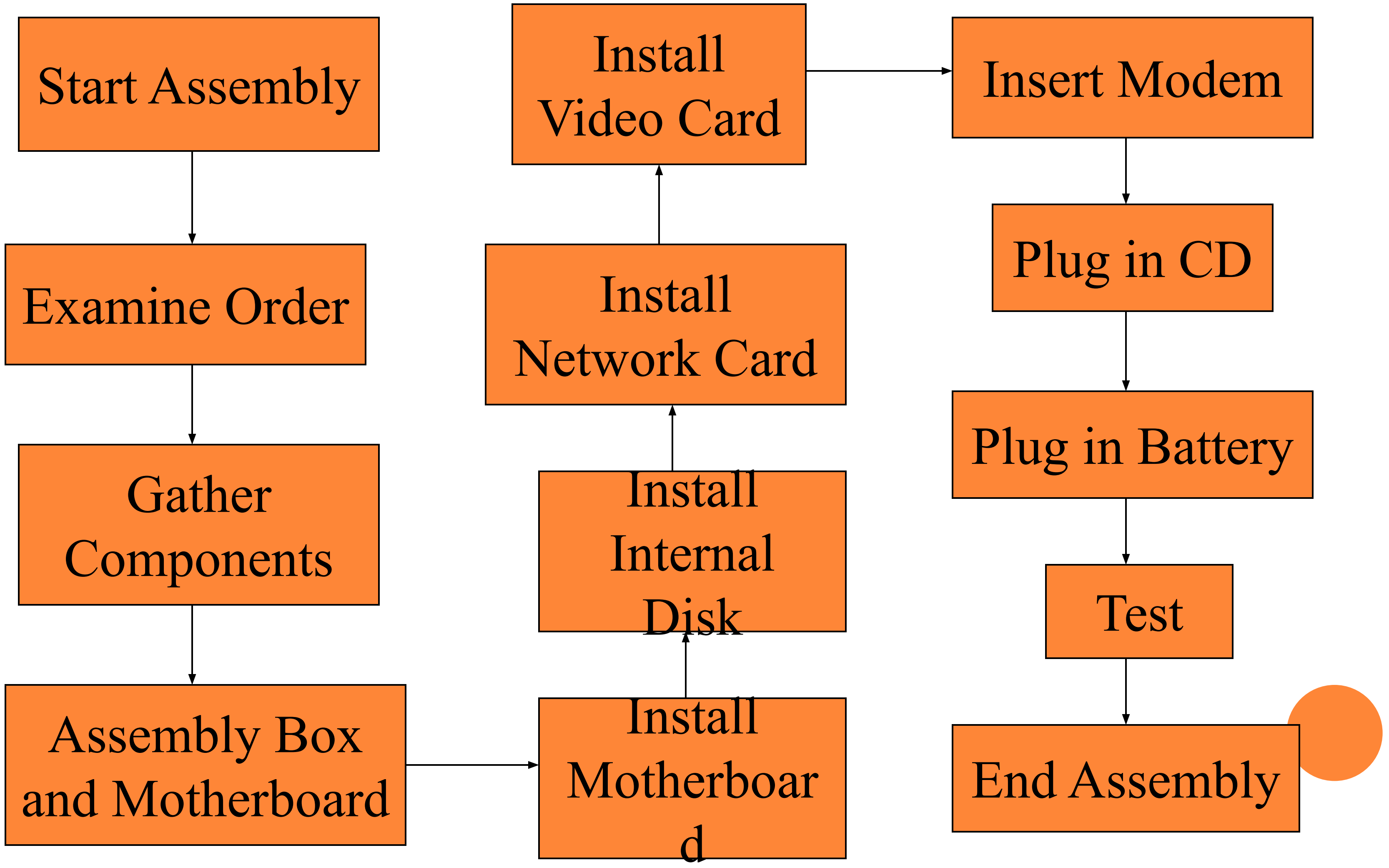
Towers of Hanoi



Sliding-Tile Puzzle

1	3	6	11
9	8	12	
10	4	13	15
2	7	14	5

Planning in Workflow Management



Planning Languages

- Languages must represent..
  - States
  - Goals
  - Actions
- Languages must be
  - Expressive for ease of representation
  - Flexible for manipulation by algorithms

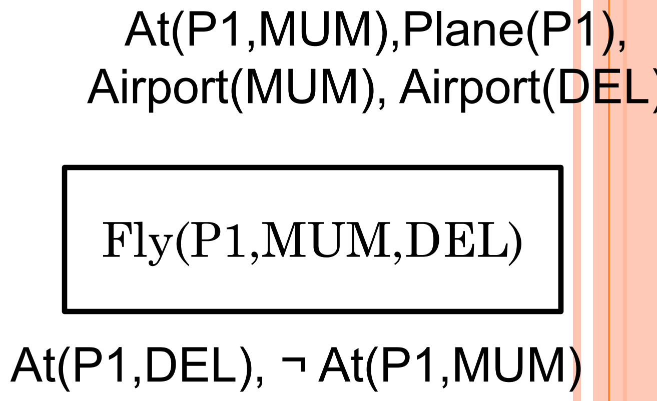
Representation of states

- A state is represented with a conjunction of positive literals
- Using
  - Logical Propositions: *Poor* $\wedge$ *Unknown*
  - FOL literals: *At(Plane1,Mumbai)* $\wedge$ *At(Plan2,Delhi)*
  - *Literals must be ground and function free*
- FOL literals must be ground & function-free
  - Not allowed: *At(x,y)* or *At(Father(Fred),Sydney)*
- Closed World Assumption
  - What is not stated are assumed false

Representation of Goals

- Goal is a <u>partially</u> specified state
- A proposition satisfies a goal if it contains all the atoms of the goal and possibly others..
  - Example: Rich $\wedge$ Famous $\wedge$ Miserable satisfies the goal Rich $\wedge$ Famous

Representation of actions



?Action Schema

- Action name & parameter list
- Preconditions (conjunction of positive function free positive literals)
- Effects (conjunction of function free literals)

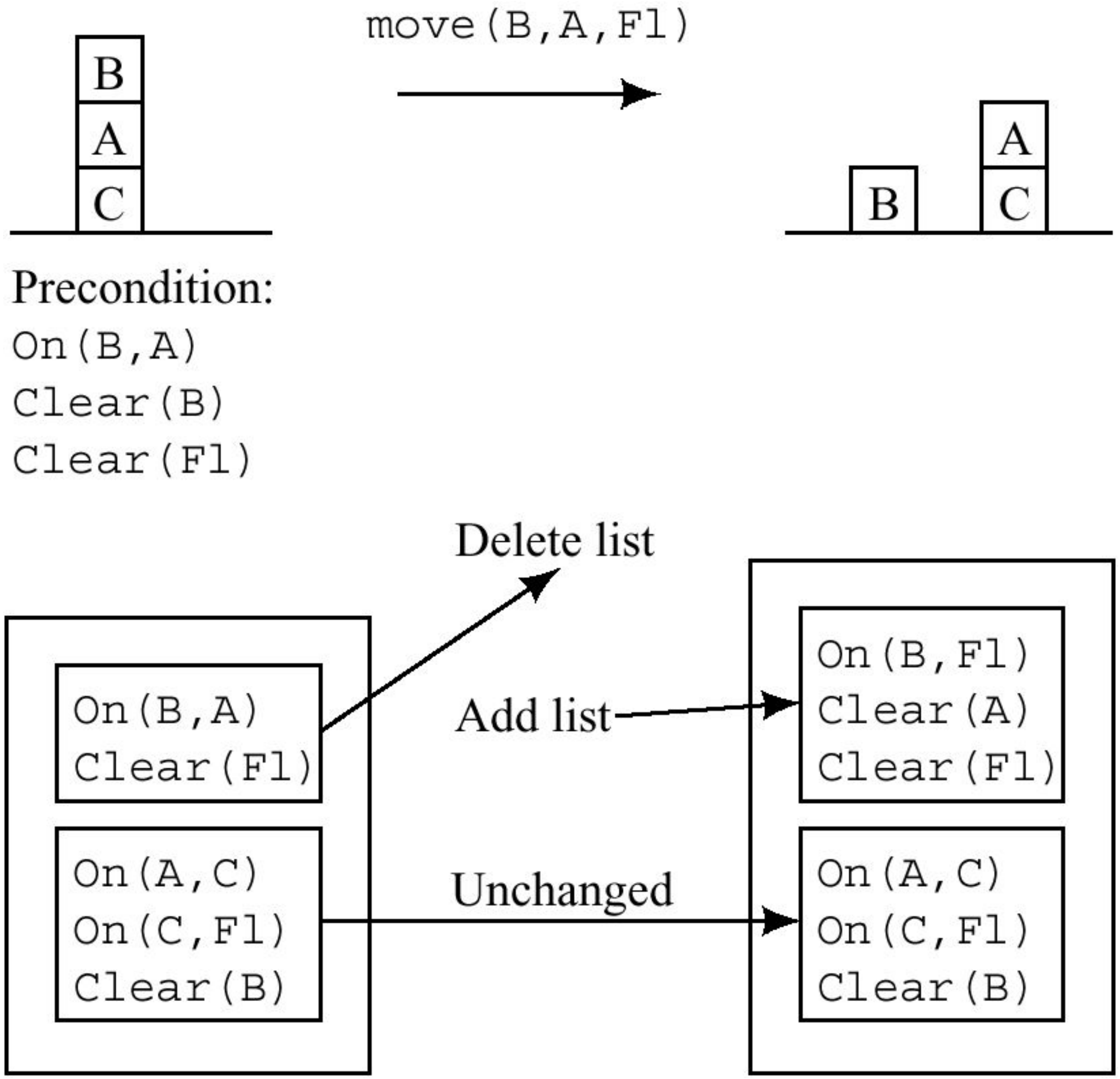
?Example Action schema

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))

Example: the move operator



Solution to planning problem

- It is an action sequence that, when a executed in initial state , results in a state that satisfies a goal

Applying an Action

- Find a substitution list θ for the variables
  - of all the precondition literals
  - with (a subset of) the literals in the current state description
- Apply the substitution to the propositions in the effect list
- Add the result to the current state description to generate the new state
- Example:
  - Current state: At(P1,MUM) $\wedge$ At(P2,SFO) $\wedge$ Plane(P1) $\wedge$ Plane(P2) $\wedge$ Airport(MUM) $\wedge$ Airport(DEL)
  - It satisfies the precondition with θ={p/P1,from/MUM, to/DEL)
  - Thus the action Fly(P1,MUM,DEL) is applicable
  - The new current state is: At(P1,DEL) $\wedge$ At(P2,DEL) $\wedge$ Plane(P1) $\wedge$ Plane(P2) $\wedge$ Airport(MUM) $\wedge$ Airport(DEL)

Languages for Planning Problems

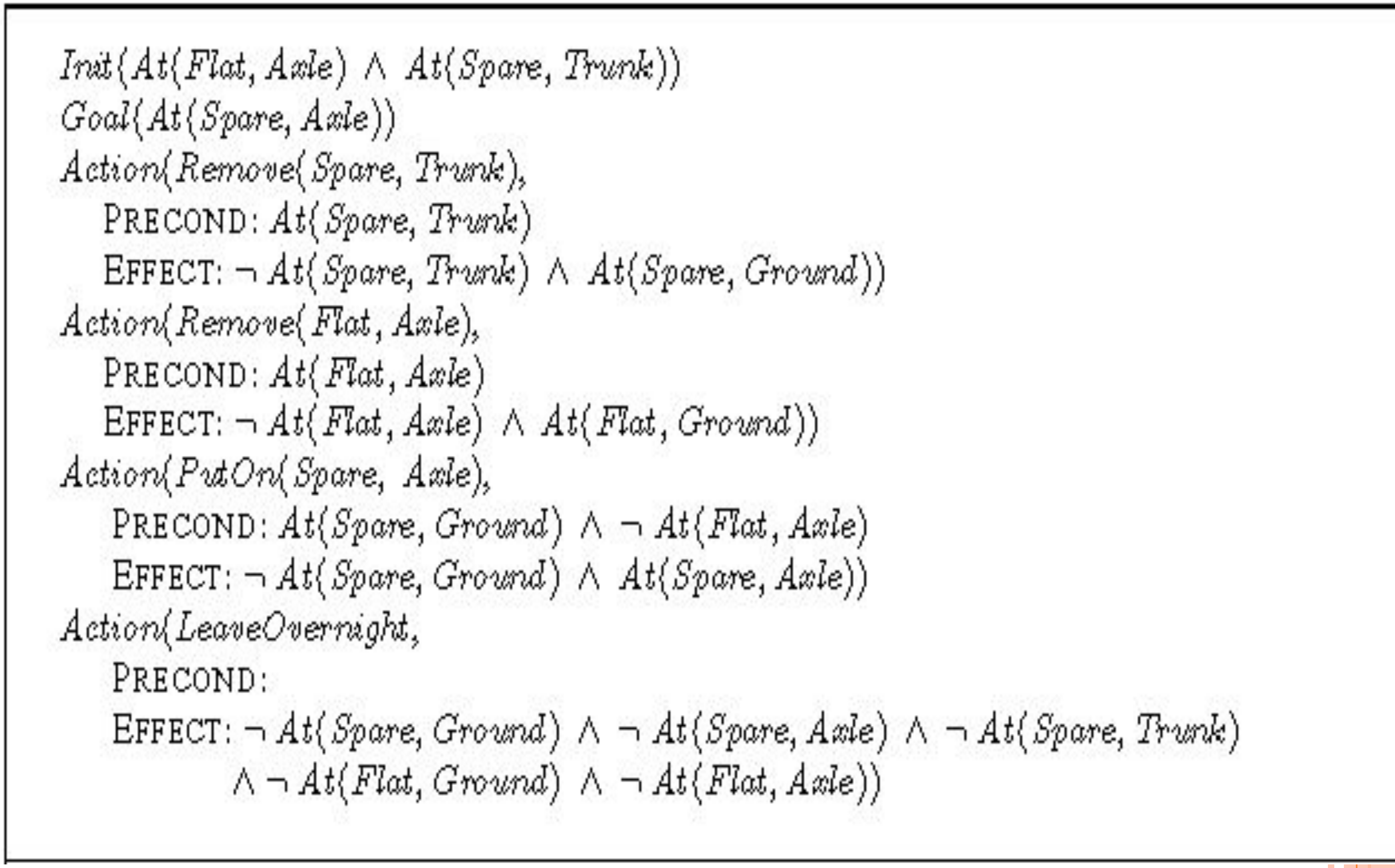
- STRIPS
  - Stanford Research Institute Problem Solver
  - Historically important
- ADL
  - Action Description Languages

STRIPS **Language**	ADL **Language**
Only positive literals in states: $Poor \wedge Unknown$	Positive and negative literals in states: $\neg Rich \wedge \neg Famous$
Closed World Assumption: Unmentioned literals are false.	Open World Assumption: Unmentioned literals are unknown.
Effect $P \wedge \neg Q$ means add P and delete Q.	Effect $P \wedge \neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q.
Only ground literals in goals: $Rich \wedge Famous$	Quantified variables in goals: $\exists x\, At(P_1, x) \wedge At(P_2, x)$ is the goal of having P_1 and P_2 in the same place.
Goals are conjunctions: $Rich \wedge Famous$	Goals allow conjunction and disjunction: $\neg Poor \wedge (Famous \vee Smart)$
Effects are conjunctions.	Conditional effects allowed: **when** P: E means E is an effect only if P is satisfied.
No support for equality.	Equality predicate $(x = y)$ is built in.
No support for types.	Variables can have types, as in $(p : Plane)$.

Example: Spare Tire Problem

- Initial State
- Goal State
- Actions:
  - *Remove(Spare,Trunk), Remove(Flat, Axle)*
  - *PutOn(Spare,Axle)*
  - *LeaveOvernight*

STRIP FOR SPARE TIRE PROBLEM



```
Init(At(Flat, Axle) ∧ At(Spare, Trunk))
Goal(At(Spare, Axle))
Action(Remove(Spare, Trunk),
    PRECOND: At(Spare, Trunk)
    EFFECT: ¬ At(Spare, Trunk) ∧ At(Spare, Ground))
Action(Remove(Flat, Axle),
    PRECOND: At(Flat, Axle)
    EFFECT: ¬ At(Flat, Axle) ∧ At(Flat, Ground))
Action(PutOn(Spare, Axle),
    PRECOND: At(Spare, Ground) ∧ ¬ At(Flat, Axle)
    EFFECT: ¬ At(Spare, Ground) ∧ At(Spare, Axle))
Action(LeaveOvernight,
    PRECOND:
    EFFECT: ¬ At(Spare, Ground) ∧ ¬ At(Spare, Axle) ∧ ¬ At(Spare, Trunk)
            ∧ ¬ At(Flat, Ground) ∧ ¬ At(Flat, Axle))
```

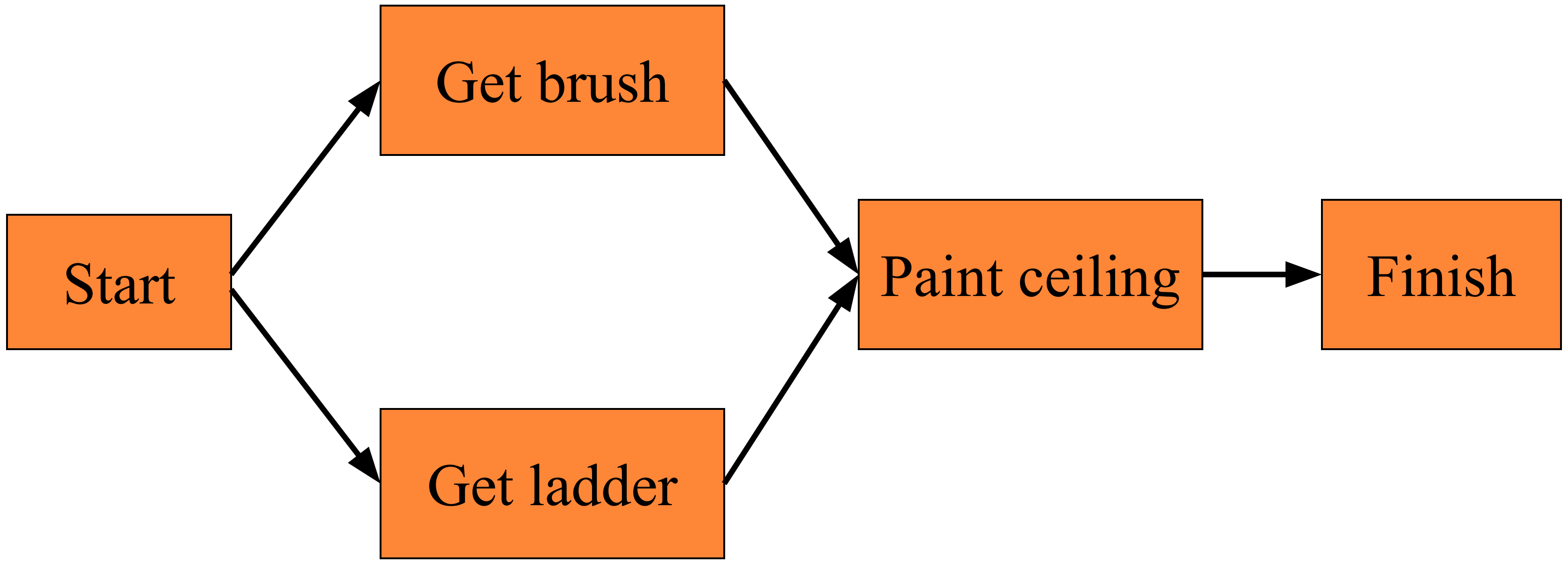
Example: Blocks World

- Initial state
- Goal
- Actions:
  - *Move*(b,x,y)
  - *MoveToTable*(b,x)

Partial-Order Plan versus Total-Order Plan

- Partial-order plan
  - consists partially ordered set of actions
  - sequence constraints exist on these actions
- Total-order plan
  - consists totally ordered set of actions

Partial-Order Plan



Total-Order Plan

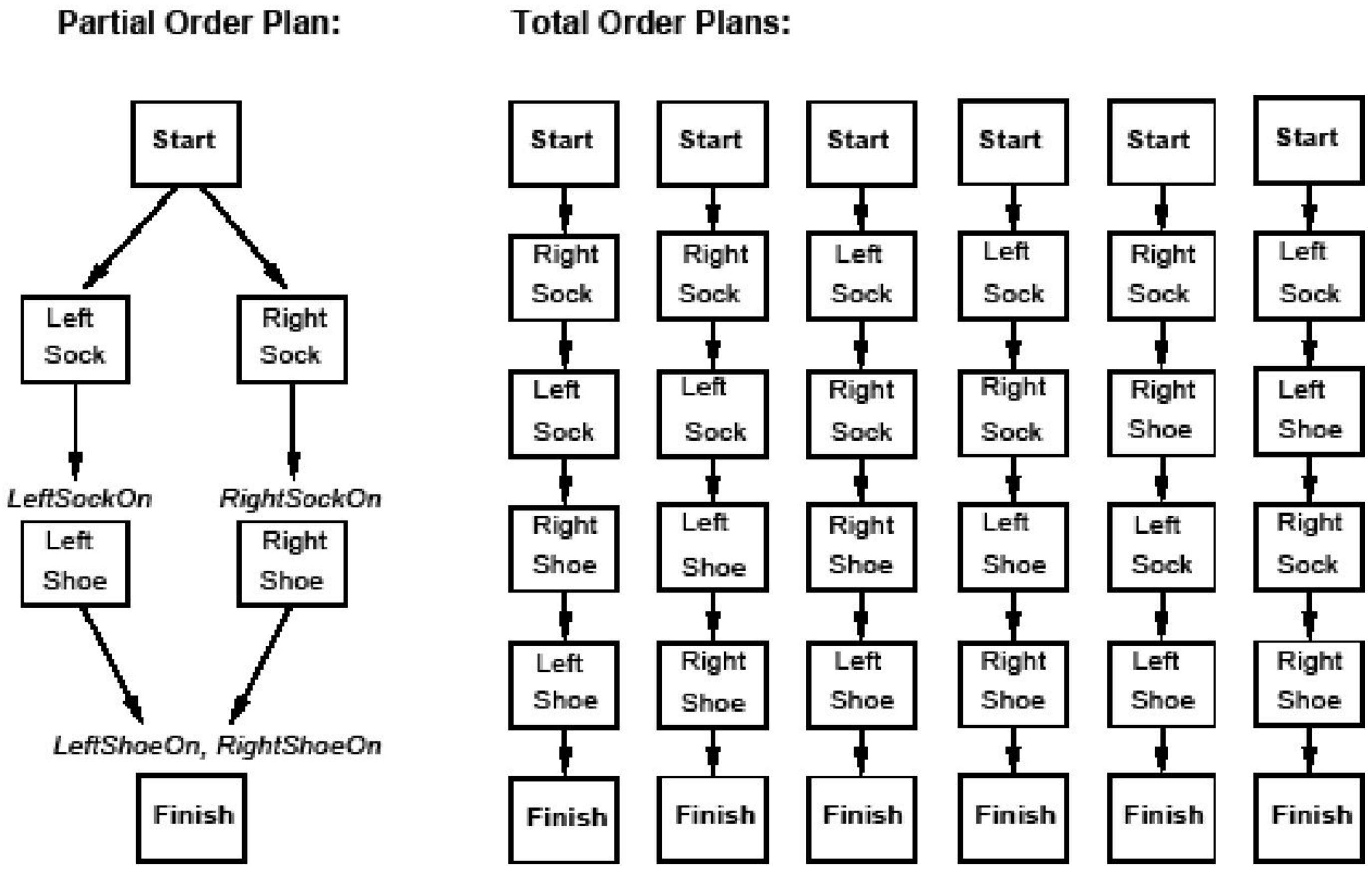
$$\begin{tikzcd}
\boxed{\text{Start}} \arrow[r] & \boxed{\text{Get ladder}} \arrow[r] & \boxed{\text{Get brush}} \arrow[r] & \boxed{\text{Paint ceiling}} \arrow[r] & \boxed{\text{Finish}}
\end{tikzcd}$$

$$\begin{tikzcd}
\boxed{\text{Start}} \arrow[r] & \boxed{\text{Get brush}} \arrow[r] & \boxed{\text{Get ladder}} \arrow[r] & \boxed{\text{Paint ceiling}} \arrow[r] & \boxed{\text{Finish}}
\end{tikzcd}$$

Partial Order Planning (POP)

- State-space search
  - Yields totally ordered plans (linear plans)
- POP
  - Works on subproblems independently, then combines subplans
  - Example
    - Goal(RightShoeOn $\wedge$ LeftShoeOn)
    - Init()
    - *Action*(RightShoe, Precond: RightSockOn, Effect: RightShoeOn)
    - *Action*(RightSock, Effect: RightSockOn)
    - *Action*(LeftShoe, Precond: LeftSockOn, Effect: LeftShoeOn)
    - *Action*(LeftSock, Effect: LeftSockOn)

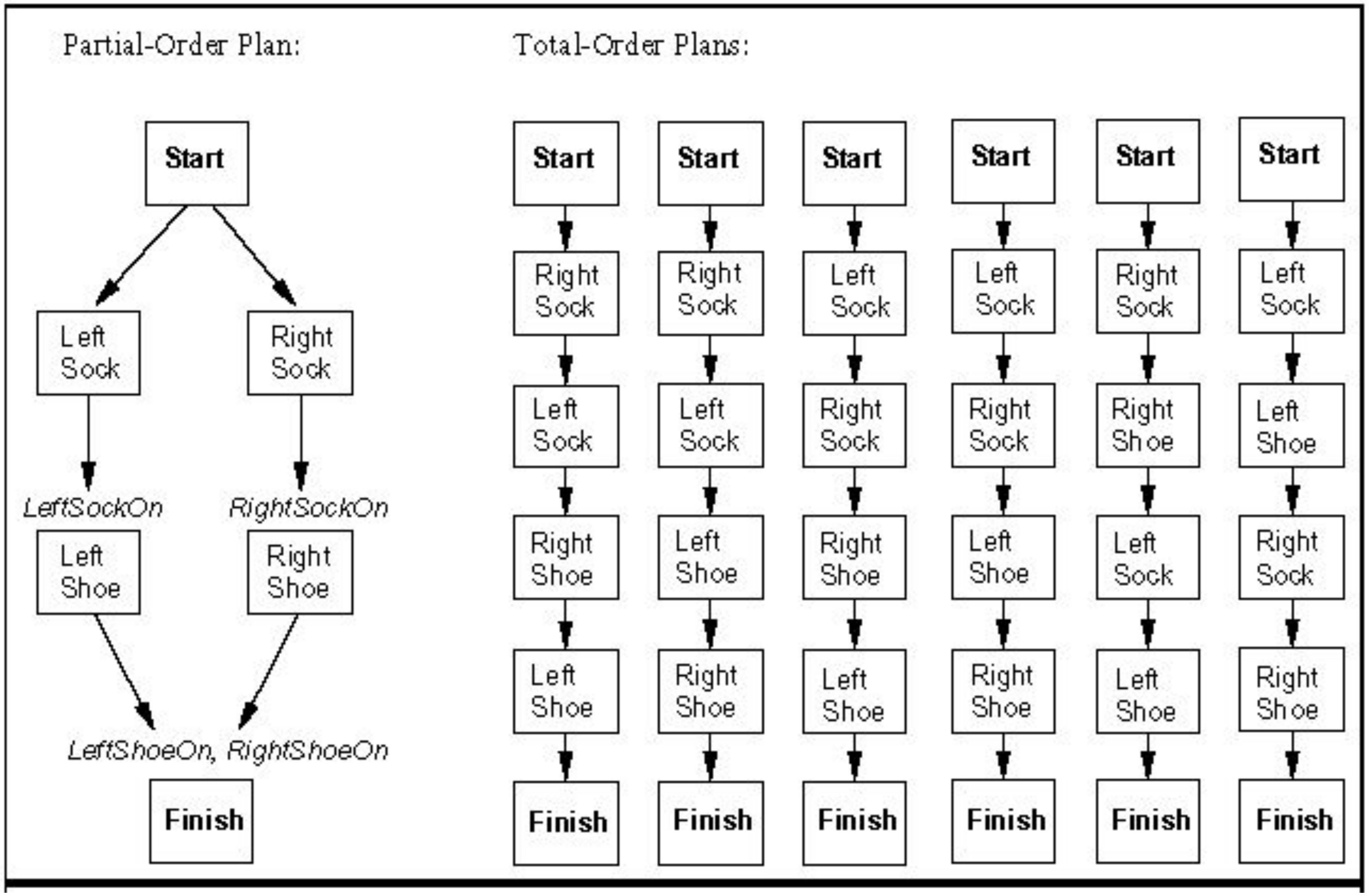
POP Example & its linearization



Components of a Plan

1. A set of actions
2. A set of ordering constraints
   - A α B reads "A before B" but not necessarily immediately before B
   - Alert: caution to cycles A α B and B α A
3. A set of causal links (protection intervals) between actions
   - A $\xrightarrow{p}$ B reads "A achieves *p* for B" and p must remain true from the time A is applied to the time B is applied
   - Example "RightSock $\xrightarrow[\textit{RightSockOn}]{}$ RightShoe
4. A set of open preconditions
   - Planners work to reduce the set of open preconditions to the empty set w/o introducing contradictions

A partial order plan for putting shoes and socks



Partially ordered plans

Partially ordered collection of steps with

- $Start$ step has the initial state description as its effect
- $Finish$ step has the goal description as its precondition
- causal links from outcome of one step to precondition of another
- temporal ordering between pairs of steps

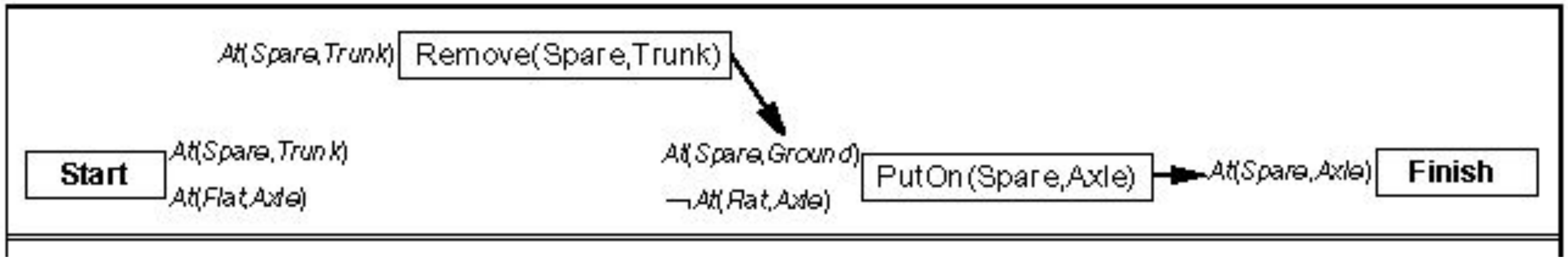
Open condition = precondition of a step not yet causally linked

A plan is complete iff every precondition is achieved

A precondition is achieved iff it is the effect of an earlier step
and no possibly intervening step undoes it

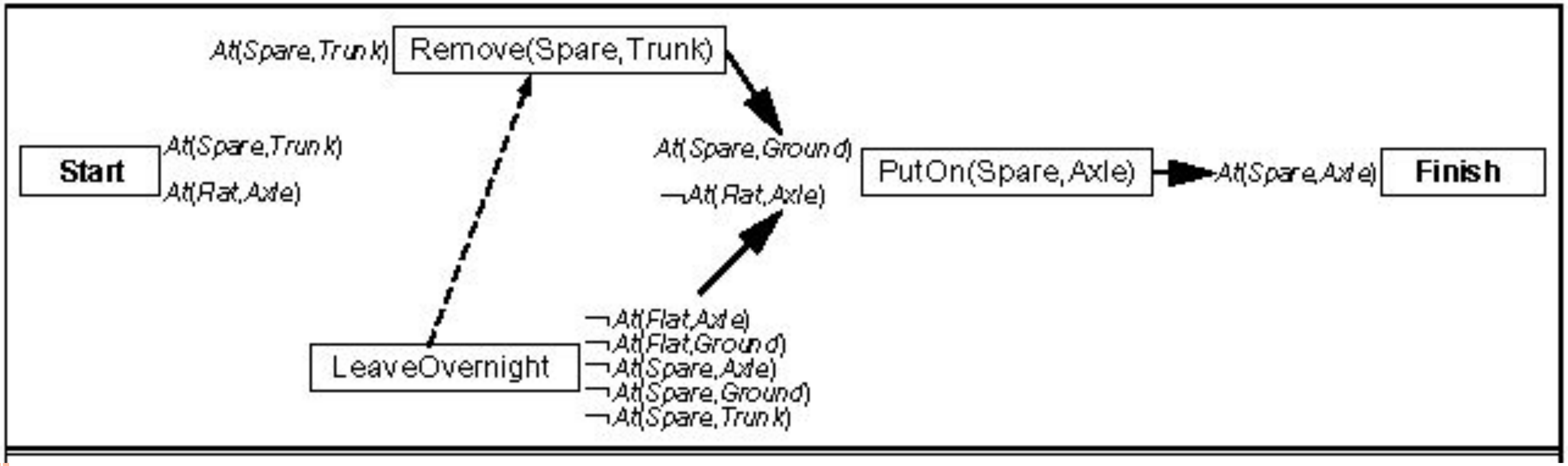
The initial partial plan for Spare tire

- From initial plan, pick an open precond (At(Spare,Axle)) and choose an applicable action (PutOn))
- Pick precond At(Spare,ground) and choose an applicable action Remove(Spare,trunk)



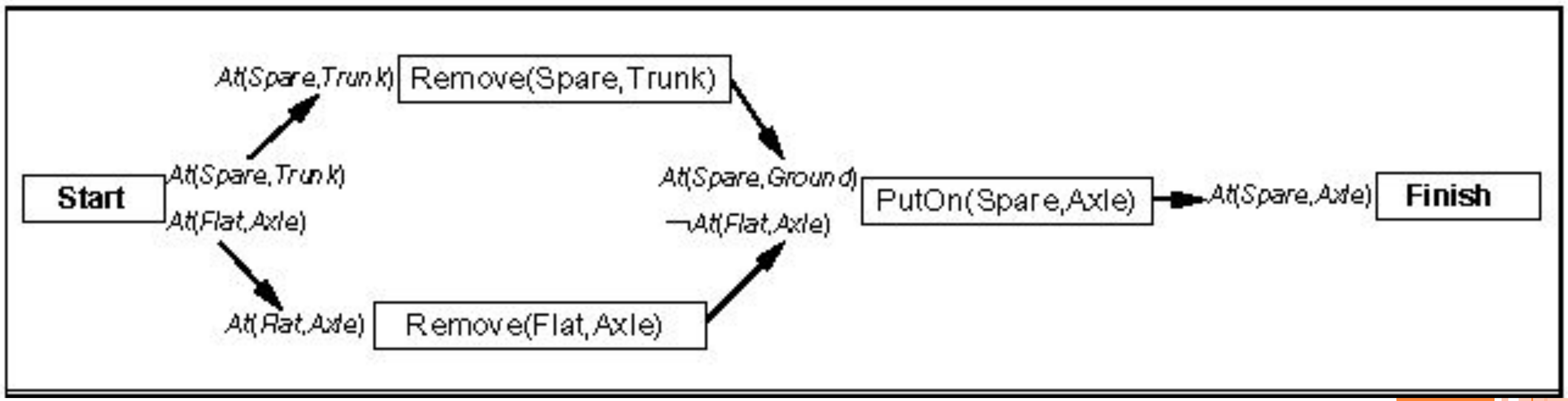
Spare-tire, continued

- Pick precond ~At(Flat,Axle) and choose Leaveovernight action.
- Because it has ~At(Spare,ground) it conflicts with "Remove"



Flate-tire, continued

- Removeovernight doesn't work so:
- Consider ~At(Flat,Axle) and choose Remove(Flat,axle)
- Pick At(Spare,Trunk) precond, and Start to achieve it.



Hierarchical planning

Example : One level planner

- Planning for ”Going to Goa this Cristmas”
    - Switch on computer
    - Start web browser
    - Open Indian Railways website
    - Select date
    - Select class
    - Select train
    - … so on
- Practical problems are too complex to be solved at one level

Hierarchy in Planning

- Hierarchy of actions
  - In terms of *major* action or *minor* action
- Lower level activities would detail more precise steps for accomplishing the higher level tasks.

Example

- Planning for ”Going to Goa this Cristmas”
  - Major Steps :
    - Hotel Booking
    - Ticket Booking
    - Reaching Goa
    - Staying and enjoying there
    - Coming Back
  - Minor Steps :
    - Take a taxi to reach station / airport
    - Have dinner on beach
    - Take photos

Motivation

- Reduces the size of search space

  Instead of having to try out a large number of possible plan ordering, plan hierarchies limit the ways in which an agent can select and order its primitive operators

  If entire plan has to be synthesized at the level of most detailed actions, it would be impossibly long.

General Property

- ***Postpone*** attempts to solve mere details, ***until*** major steps are in place.
- Higher level plan may run into difficulties at a lower level, causing the need to return to higher level again to produce appropriately ordered sequence.
- Planner
    - Identify a hierarchy of conditions
    - Construct a plan in levels, postponing details to the next level
    - Patch higher levels as details become visible

Ref : [1,2]

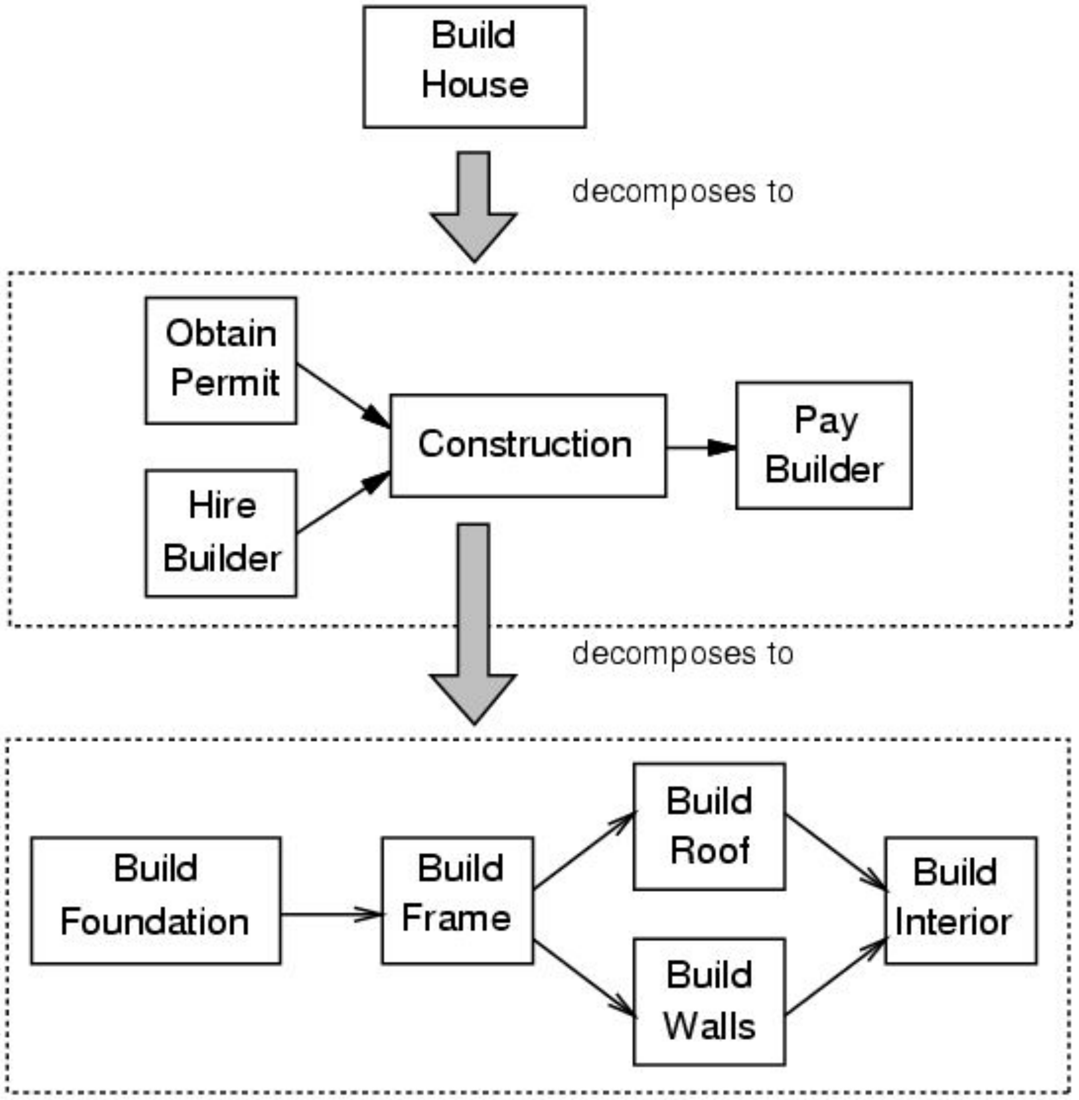
Example

- Actions required for “Travelling to Goa”:
    - Opening makemytrip.com (1)
    - Finding flight (2)
    - Buy Ticket (3)
    - Get taxi(2)
    - Reach airport(3)
    - Pay-driver(1)
    - Check in(1)
    - Boarding plane(2)
    - Reach Goa(3)

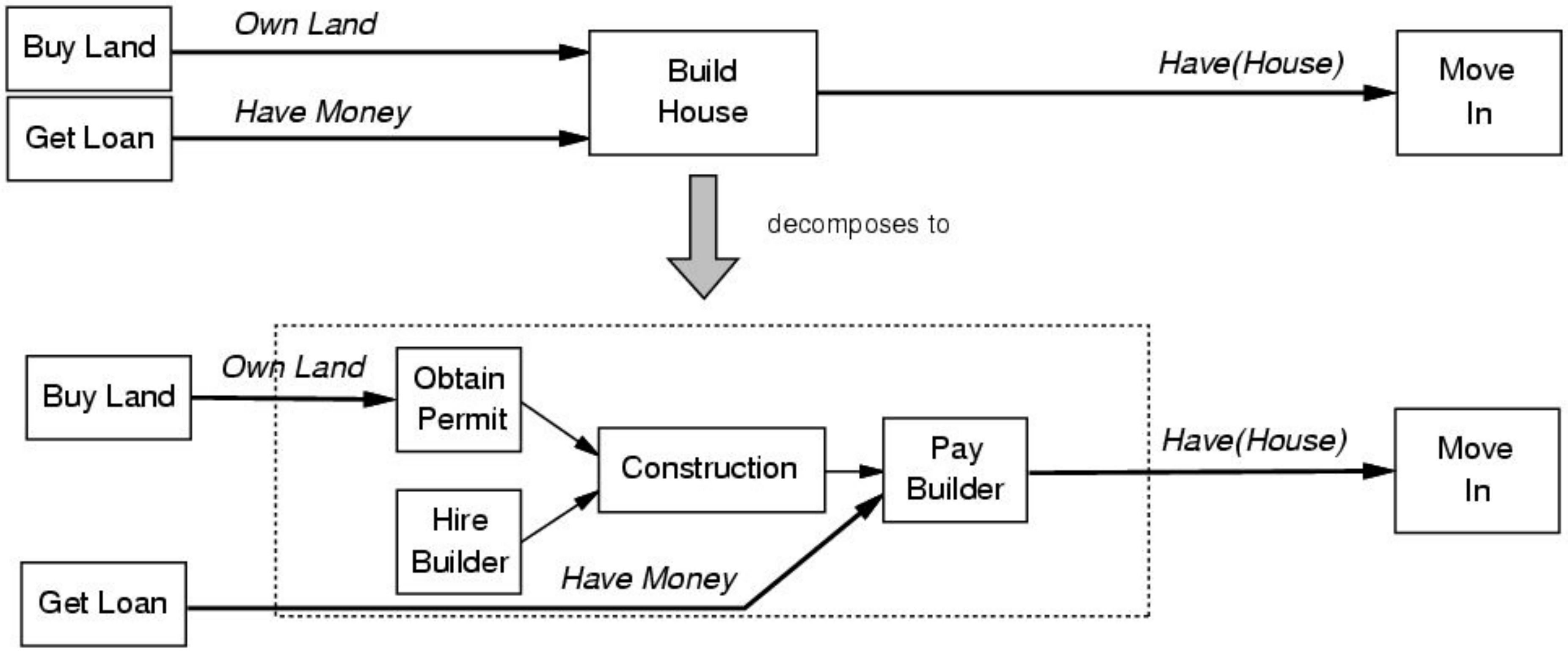
Example

- 1st level Plan :
  - Buy Ticket (3), Reach airport(3), Reach Goa(3)
- 2nd level Plan :
  - Finding flight (2), Buy Ticket (3), Get taxi(2), Reach airport(3), Boarding plane(2), Reach Goa(3)
- 3rd level Plan (final) :
  - Opening makemytrip.com (1), Finding flight (2), Buy Ticket (3), Get taxi(2), Reach airport(3), Pay-driver(1), Check in(1), Boarding plane(2), Reach Goa(3)

Hierarchical Decomposition



Task Reduction



Task Network

- Collection of ***task*** and ***constraints*** on those tasks
- $((n_1, \alpha_1), \ldots, ((n_m, \alpha_m), \phi)$,
  - where:
  - n1, ..., nm are the names of the tasks
  - α1, ..., αm are the labels associated with the tasks, which provide additional information about the tasks (e.g., their parameters, preconditions, effects, etc.)
  - ϕ is a boolean formula expressing the constraints on the tasks
  - Each task is represented by a tuple (ni, αi), where ni is the name of the task and αi is its label.
  - The label αi can be any data structure that provides additional information about the task, such as its parameters or preconditions.
  - For example, if the task is "move(p1, p2)", where p1 and p2 are the positions of two objects, the label αi could be a tuple (p1, p2).
    - Truth constraint : (n, p, n') means p will be true immediately after n and immediately before n'.
    - Temporal ordering constraint : n $\prec$ n' means task n precedes n'.

Ref : [3]

Conditional Planning

UNCERTAINTY

- The agent might not know what the initial state is
- The agent might not know the outcome of its actions

The plans will have branches rather than being straight line plans, includes *conditional steps*

- $\textbf{if} < \textit{test} > \textbf{then}\ \textit{plan}_A\ \textbf{else}\ \textit{plan}_B$

Simply get plans ready for all possible contingencies

Modeling uncertainty

Actions sometimes fail → disjunctive effects

- Example: moving left sometimes fails
- *Action*(*Left*,PRECOND: *AtR*,EFFECT: *AtL* ∨ *AtR*)
- *Conditional effects*: effects are conditioned on secondary preconditions

  Action(*Suck*, PRECOND: ;,

  EFFECT: (**when** *AtL: CleanL*) ∧ (**when** *AtR: CleanR*))
- Actions may have both disjunctive and conditional effects:

  Moving sometimes dumps dirt on the destination square only when that square is clean

  Action(*Left*, PRECOND: *AtR*;,

  EFFECT: *AtL* ∨ (*AtL* ∧ **when** *CleanL*: ¬ *CleanL*))