Planning

Why Planning ?

We have already discussed

- Search based problem solving
- Logical planning agents
  - Action and state
- How forward and backward search algorithms can be used
- Some complex planning

We need to explore the problems that are not constrained to consider only totally ordered sequence of action

- We will consider fully observable , deterministic , finite , static and discrete environment they are called classical planning

3.2 Partial Order Planning…

- ✓ Forward and backward state-space search are particular forms of **totally ordered plan search.**
- ✓ They explore only strictly linear sequences of actions directly connected to the **start or goal.**
- ✓ Rather than work on **each subproblem separately,** they must always make decisions about how to **sequence actions** from all the subproblems.

Cont…

- ✓But it is preferable to work on several subgoals **independently, solves them** with several subplans, and then **combines the subplans**.

- ✓The **planner** can work on "obvious" or **"important' decisions first,** rather than being forced to work on steps in chronological order.

- ✓The general strategy of delaying a choice during search is called a **least commitment** strategy.

Example : Putting on a pair of shoes...

Plan…

$$
\begin{array}{l}
\mathrm{Goal}(RightShoeOn \wedge LeftShoeOn) \\
Init() \\
Action(RightShoe, \textsc{Precond}{:}RightSockOn, \textsc{Effect}{:}RightShoeOn) \\
Action(RightSock, \textsc{Effect}{:}RightSockOn) \\
Action(LeftShoe, \textsc{Precond}{:}LeftSockOn, \textsc{Effect}{:}LeftShoeOn) \\
\mathrm{Action}(LeftSock, \textsc{Effect}{:}LeftSockOn) .
\end{array}
$$

Plan…

Goal($RightShoeOn \wedge LeftShoeOn$)
$Init()$
$Action(RightShoe$, PRECOND:$RightSockOn$, EFFECT:$RightShoeOn)$
$Action(RightSock$, EFFECT:$RightSockOn)$
$Action(LeftShoe$, PRECOND:$LeftSockOn$, EFFECT:$LeftShoeOn)$
Action($LeftSock$, EFFECT:$LeftSockOn$) .

✓A planner should be able to come up with the **two-action** sequence:

i. **Rightsock followed by Rightshoe** to achieve the first conjunct of the goal

ii. **Leftsock followed by LeftShoe** for the second conjunct.

✓Then the **two sequences can be combined** to yield the **final plan**.

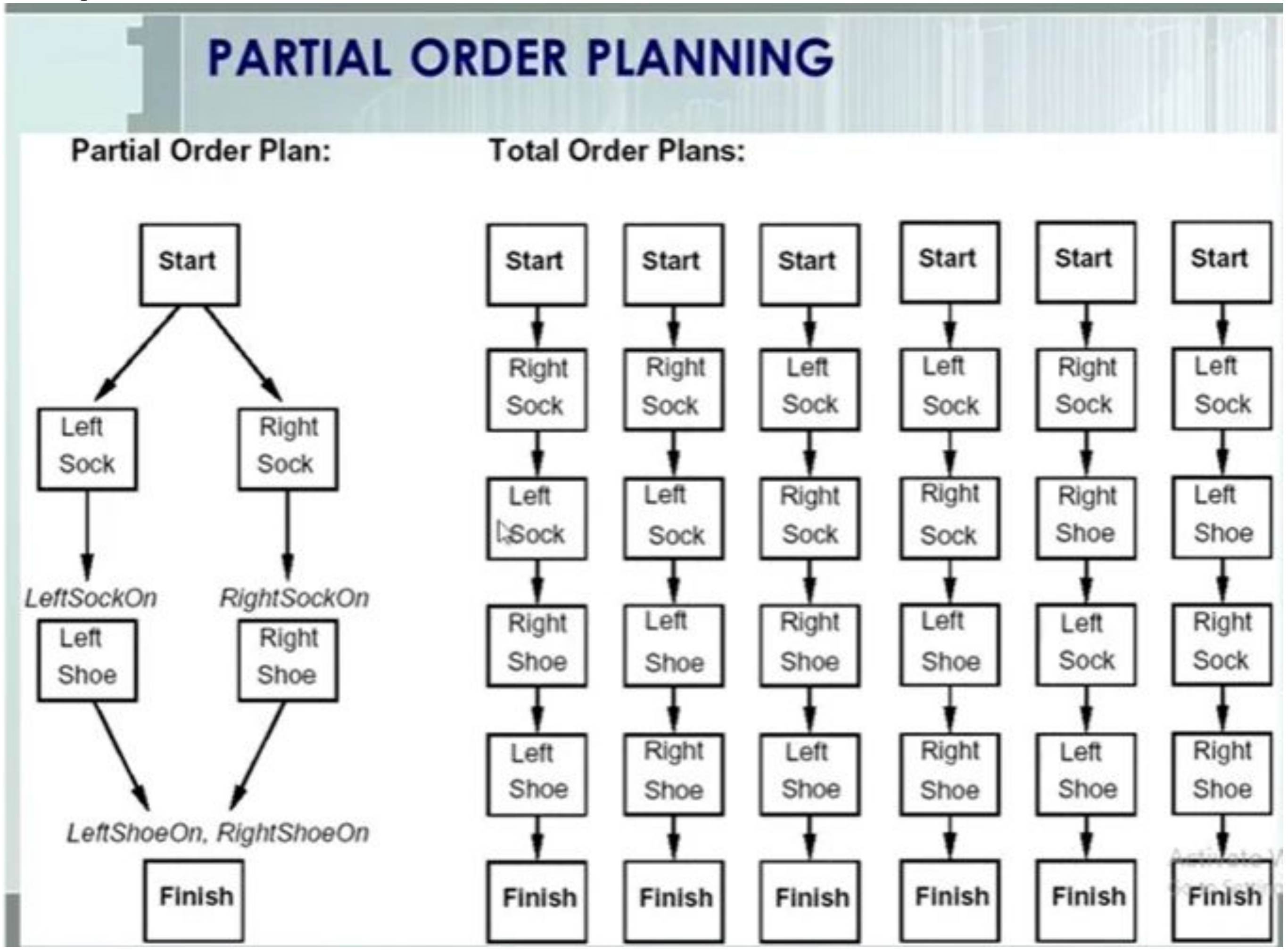
Cont…

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

- ✓ In the following figure, the solution is represented as a **graph of actions, not a sequence.**

- ✓ And also the **"dummy"** actions called **Start** and **Finish**, which mark the **beginning** and **end** of the plan.

Planning graph



Cont…

- ✓ The partial-order solution corresponds to **six possible total-order plans**;

- ✓ Each of these is called a **linearization of the partial-order plan.**

- ➢ **Planning as a search:**
- ✓ Partial-order planning can be implemented **as a search** in the space of partial-order plans.

- ✓ That is, we start with an **empty plan.**

- ✓ Then we consider ways of refining the plan until we come up with a **complete plan** that solves the problem.

Cont…

✓ Each plan has the following **four** components:

a. **First two** define the **steps of the plan.**

b. **Last two** serve a **bookkeeping function** to determine how plans can be extended:

i. **Actions**

ii. **Ordering constraints**

iii. **Casual links**

iv. **Open pre-conditions**

1. Actions…

- ✓ A set of actions that make up the **steps of the plan.**
- ✓ These are taken from the set of actions in the **planning problem.**
- ✓ The **"empty"** plan contains just the **Start** and **Finish** actions.
- ✓ **Start has no preconditions** and has as its effect all the literals in the initial state of the planning problem.
- ✓ **Finish has no effects** and has as its preconditions the goal literals of the planning problem.

ii. Ordering Constraints…

- ✓ These are a **set of ordering constraints.**
- ✓ Each ordering constraint is of the form $A \prec B$, which is read as **"A before B“.**
- ✓ It means that action **A** must be executed sometime before action **B,** but not necessarily immediately before.
- ✓ Any cycle-such as $A \prec B$ and $B \prec A$-represents a contradiction, so an ordering constraint cannot be added to the plan if it creates a cycle.

iii. Causal links…

- ✓ A causal link between two **actions A and B** in the plan is written as $A \xrightarrow{p} B$ and is read as **"A achieves p for B."**

- ✓ For example, the causal link $\mathit{RightSock} \xrightarrow{\mathit{RightSockOn}} \mathit{RightShoe}$ asserts that RightSockOn is an effect of the RightSock action and a precondition of RightShoe.

- ✓ It means, the plan may not be extended by adding a new action **'C'** that **conflicts with the causal link.**

Cont…

- ✓ Sometimes, the causal links also called as **protection intervals,** because the link $A \xrightarrow{p} B$ protects **“p”** from being negated over the interval from A to B.

iv Open preconditions:

- ✓ A precondition is open **if it is not achieved by some action in the plan.**

- ✓ Planners will work to reduce the set of **open preconditions to the empty set**, without introducing a contradiction.

Cont…

Actions: $\{RightSock$, RightShoe, $LeftSock$, $LeftShoe$, Start, Finish)

Orderings: $\{RightSock \prec$ RightShoe, $LeftSock \prec LeftShoe\}$

Links: $\{RightSock \xrightarrow{RightSockOn}$ RightShoe, $LeftSock \xrightarrow{LeftSockOn}$ LeftShoe,

RightShoe $\xrightarrow{RightShoeOn} Finish, LeftShoe \xrightarrow{LeftShoeOn} Finish\}$

Open Preconditions: { } .

➢**Consistent plan:** It is a plan in which there are no cycles in the ordering constraints and no conflicts with the causal links.

✓And a consistent plan with **no open preconditions is a solution.**

✓Every linearization of a partial-order solution is a total-order solution whose execution from the initial state will reach a goal state.

Example for POP: The spare tire problem…

- **Problem statement:**

  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.

Action Description language (ADL)

Simple flat tire description…

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*))

Planning Graph

Complete – Consistent Plan…

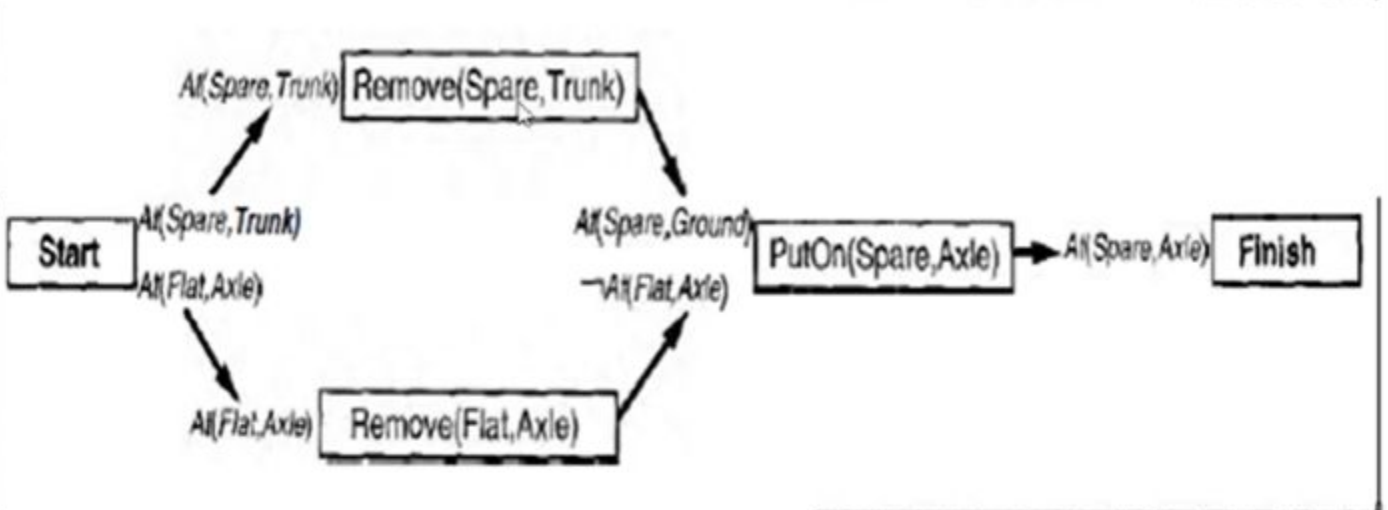


Figure 11.10 The final solution to the tire problem. Note that *Remove*(*Spare*, Trunk) and *Remove*(*Flat*, *Axle*) can be done in either order, as long as they are completed before the *PutOn*(*Spare*,*Axle*) action.