UNIT III
Planning

The task of coming up with a sequence of actions that will achieve a goal is called **planning**

classical planning environments

- ✓ fully observable, deterministic,finite, static (change happens only when the agent acts), and discrete (in time, action, objects, and effects). These are called. In contrast,
- ✓ **Nonclassical** planning is for partially observable or stochastic environments and involves a different set of algorithms and agent designs

THE PLANNING PROBLEM

- Agent Environment **STATES** are represented as valuations of state variables.
- An action can be represented as a **procedure or a program.**
- The procedures are used to compute values of state variables.
- After the execution of procedure (i.e after the action),the environment state will be changed, towards the goal.

Planning algorithms

- representation of planning problems-states, actions, and goals
- Algorithms are nothing but logical structure of the problem.
- To define an efficient algorithm language is very important.
- STRIPS language(Standford Research Institute and Problem Solver).

Representation of States

- Planners decompose the world into logical conditions and represent a state as a conjunction of positive literals.
- Literals in first-order state descriptions must be **ground (single data not variables)**and **function-free.** Literals such as At(x, y) or At(Father(Fred), Sydney) are not allowed.
- The **closed-world assumption** is used, meaning that any conditions that are not mentioned in a state are assumed false.

Representation of goals

- **. A** goal is a partially specified state, represented as a conjunction of positive ground literals

such as Rich ∧ Famous or $At(P_2, \text{Tahiti})$

- A propositional state *s* **satisfies** a goal **g** if *s* contains all the atoms(ground literal)in g.
- The propositional state Rich^Famous^Happy satisfies the goal state Rich^Famous.

Representation of actions

Representation of actions. An action is specified in terms of the preconditions that must hold before it can be executed and the effects that ensue when it is executed. For example, an action for flying a plane from one location to another is:

$$Action(Fly(p, from, \text{to}),$$
$$\text{PRECOND}: At(p, \text{from}) \text{ A } Plane(p) \text{ A } Airport(from) \wedge Airport(to)$$
$$\text{EFFECT}: \neg At(p, \text{from}) \text{ A } At(p, \text{to}))$$

Action schema

- An action schema consists of three parts:
- The action name and parameter list-for example, Fly(p, from, to)-serves to identify the action.
- The **precondition** is a conjunction of function-free positive literals stating what must be true in a state before the action can be executed. Any variables in the precondition must also appear in the action's parameter list.
- The **effect** is a conjunction of function-free literals describing how the state changes when the action is executed.

Add list and Delete list

- Two categories
- Add list-positive literals
- Delete list-negative literals
- Action is **applicable** in any state that satisfies the precondition; otherwise, the action has no effect.

$$At(P_1, JFK) \wedge At(P_2, SFO) \wedge Plane(P_1) \wedge Plane(P_2) \wedge Airport(JFK) \wedge Airport(SFO).$$

This state satisfies the precondition

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

with substitution $\{p/P_1, from/JFK, to/SFO\}$ (among others—see Exercise 11.2). T
the concrete action $Fly(P_1, JFK, SFO)$ is applicable.

STRIPS **Language**	ADL **Language**
Only positive literals in states: *Poor* ∧ *Unknown*	Positive and negative literals in states: $\neg$*Rich* ∧ $\neg$*Famous*
Closed World Assumption: Unmentioned literals are false.	Open World Assumption: Unmentioned literals are unknown.
Effect P ∧ $\neg Q$ means add P and delete Q.	Effect P ∧ $\neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q.
Only ground literals in goals: *Rich* ∧ *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* ∧ *Famous*	Goals allow conjunction and disjunction: $\neg$*Poor* ∧ *(Famous V 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 $(p : Plane)$.

Figure 11.1 Comparison of STRIPS and ADL languages for representing planning problems. In both cases, goals behave as the preconditions of an action with no parameters.

What is STRIPs?

- It is a automated planning technique used to find a goal by executing a domain from the initial state of domain.
- With STRIPS we can first describe the world(initial state and goal state)by providing objects,actions,preconditions and effects.
- To describe the world we used two categories of term
  - ✓ States-initial state and goal state
  - ✓ Action schema-objects ,actions ,preconditions and effects

STRIPS-States,goals,Actions

- States: conjunction of ground, function-free and positive literals.
- To describe states, closed assumptions is used(the world model contains everything,the agent needs to know.
- Goals:conjunction of literals,may contain variables.
- Action:precondns that must hold before execution and the effects after exeution.

STRIPS Action Schema

- It includes
- Action name& parameter list
- Precond:a conjunction of function free positive literals.The action variables must also appear in precondition.
- Effect:a conjunction of function free literals(positive or negative)

Eg:

Action:But(x)

Preconditon;At(p),Sells(p,x)

Effect:Have(x)

STRIPS - Air cargo transport

Figure 11.2 shows an air cargo transport problem involving loading and unloa
and off of planes and flying it from place to place. The problem can be def
actions: *Load*, *Unload*, and *Fly*. The actions affect two predicates: $In(c, \mathrm{p})$ m
c is inside plane p, and $At(x, a)$ means that object x (either plane or cargo)

- Solution

details consistently. The following plan is a solution to the problem:

$$[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK$$
$$Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P2, SFO$$

$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\land Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1, JFK) \land At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p, from) \land At(p, to))$

Figure 11.2 A STRIPS problem involving transportation of air cargo between airports.

Partial Order Planning-The shoe and sock problem

- Works on several subgoals independently
- Solves them with subplans
- Combines the subplans
- Flexibility in ordering the subplans
- Least commitment strategy:

  delaying a choice during search

Goal(*RightShoeOn* A *LeftShoeOn*)
Init()
Action(*RightShoe*, PRECOND:*RightSockOn*, EFFECT:*RightShoeOn*)
Action(*RightSock*, EFFECT:*RightSockOn*)
Action(*LeftShoe*, PRECOND:*LeftSockOn*, EFFECT:*LeftShoeOn*)
Action(*LeftSock*, EFFECT:*LeftSockOn*) .

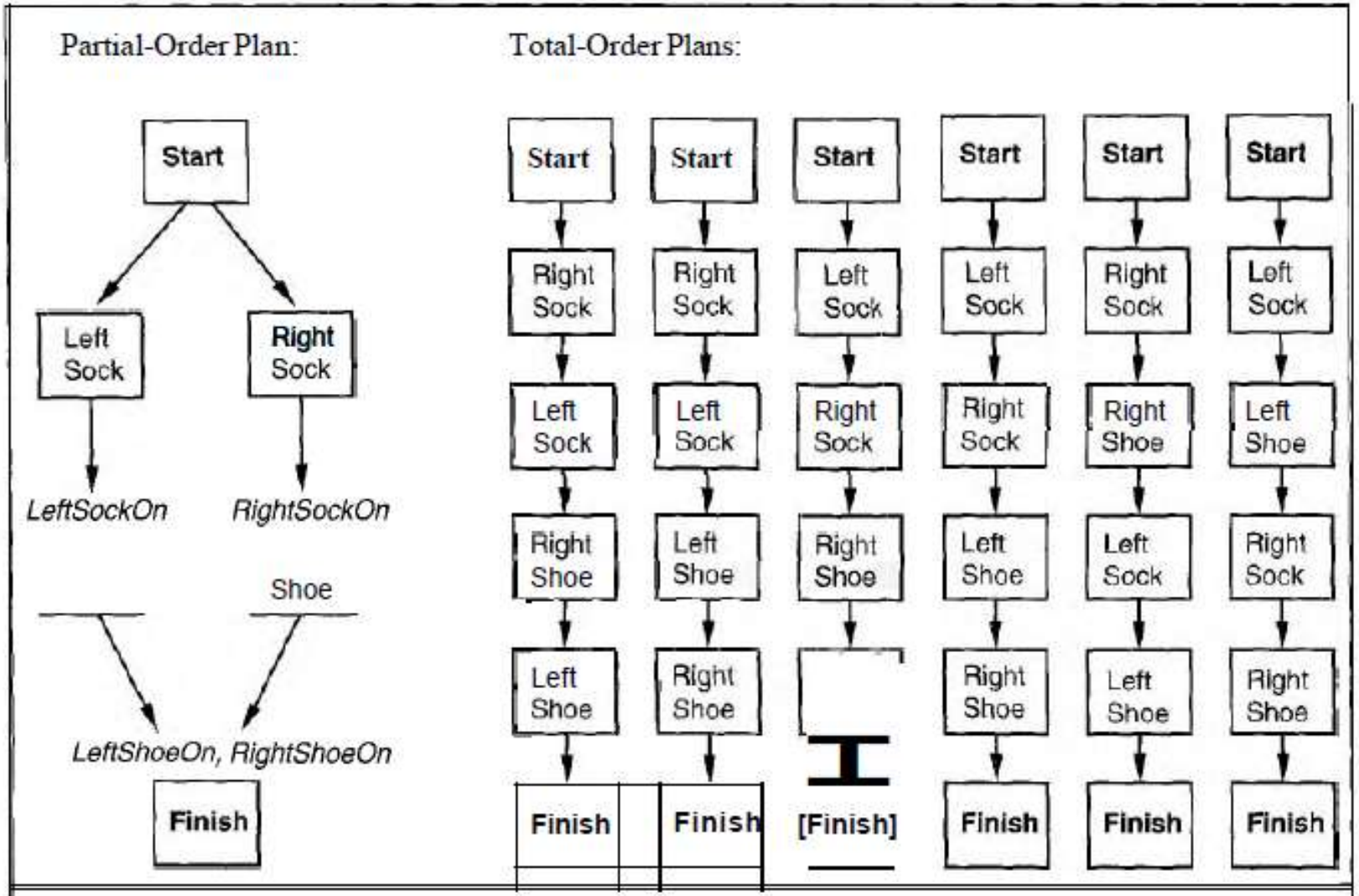

Figure 11.6 A partial-order plan for putting on shoes and socks, and the six corresponding linearizations into total-order plans.

How to define Partial Order plan

- A set of **actions** that make up the steps of the plan.
- A set of **ordering constraints.** Each ordering constraint is of the form A - B, which is read as "A before B" and means that action A must be executed sometime before action B, but not necessarily immediately before.
- A set of **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

  $$RightSock \xrightarrow{RightSockOn} RightShoe$$

  asserts that *RightSockOn* is an effect of the *RightSock* action and a precondition of *RightShoe.* It also asserts that *RightSockOn* must remain true from the time of action *RightSock* to the time of action *RightShoe.* In other words, the plan may not be
- A set of **open preconditions.** A precondition is open if it is not achieved by some action in the plan

Solultion –shoe problem

For example, the final plan in Figure 11.6 has the following components (not shown are the ordering constraints that put every other action after Start and before Finish):

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: { }.

POP-Changing a flat tire

- Consider the problem of changing a flat tire.
- The goal is to have a good spare tire,properly mounted onto the car's axle
- The intial stata a flat tire on the axle and a good spare tire in the trunk.
- There are 4 actions
  - Remove the spare from the trunk
  - Remove the falt tire from the axle
  - Putting the spare on the axle
  - Leaving the car unattended overnight.

  We assume that the car is in .particularly bad neighborhood so that the effect of leaving it overnight is that the tire disaapear.

A partial-order planning example

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

$Goal(At(Spare, Axle))$

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

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

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

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

Figure 11.7 The simple flat tire problem description.