UNIT III

PLANNING

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

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

PLANNING WITH STATE-SPACE SEARCH

The most straightforward approach is touse state-space search. Because the descriptions of actions in a planning problem specify
both preconditions and effects, it is possible to search in either direction: either forward fromthe initial state or backward from the goal, as shown in Figure 11.5

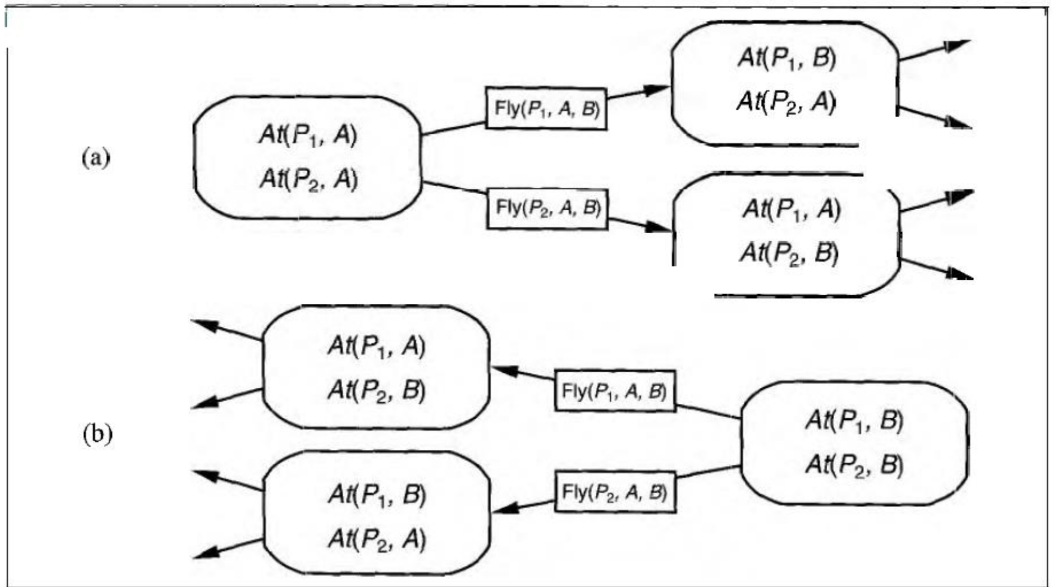


Figure 11.5 Two approaches to searching for a plan. (a) Forward (progression) state-space search, starting in the initial state and using the problem's actions to search forward for the goal state. (b) Backward (regression) state-space search: a belief-state search (see page 84) starting at the goal state(s) and using the inverse of the actions to search backward for the initial state.

Forward state-space search

Planning with forward state-space search is sometimes called **progression** planning, because it moves in the forward direction.

The formulation of planning problems as state-space search problems is as follows:

The **initial state** of the search is the initial state from the planning problem. In general, each state will be a set of positive ground literals; literals not appearing are false.

The **actions** that are applicable to a state are all those whose preconditions are satisfied. The successor state resulting from an action is generated by adding the positive effect literals and deleting the negative effect literals.

The **goal test** checks whether the state satisfies the goal of the planning problem. The **step cost** of each action is typically 1. Although it would be easy to allow different costs for different actions, this is seldom done by STRIPS planners.

First forward search does not address the irrelevant action problem- all applicable actions are considered from each state. Second, the approach quickly bogs down without a good heuristic

Consider an air cargo problem with 10 airports, where each airport has *5* planes and 20 pieces of cargo.

The goal is to move all the cargo at airport A to airport B.

There is a simple solution to the problem: load the 20 pieces of cargo into one of the planes at A, fly the plane to B, and unload the cargo.

But finding the solution can be difficult because the average branching factor is huge:

each of the 50 planes can fly to 9 other airports, and each of the 200 packages can be either unloaded (if it is loaded), or loaded into any plane at its airport (if it is unloaded). On average, let's say there are about 1000 possible actions, so the search tree up to the depth of the obvious solution has about 1000^{41} nodes.

Backward state-space search

Backward state-space search was described briefly as part of bidirectional search

We noted there that backward search can be difficult to implement when the goal states are described by a set of constraints rather than being listed explicitly. In particular, it is not always obvious how to generate a description of the possible **predecessors** of the set of goal states.

The main advantage of backward search is that it allows us to consider only **relevant** actions. An action is relevant to a conjunctive goal if it achieves one of the conjuncts of the goal.

For example, the goal in our 10-airport air cargo problem is to have 20 pieces of cargo at airport B, or more precisely,

$$At(C_1, B) \; A \; At(C_2, B) \; A \ldots A \; At(C_{20}, B) \, .$$

Now consider the conjunct *At (C1,*B) .

Working backwards, we can seek actions that have this as an effect.

There is only one: Unload *(C1,p ,* B) ,w here plane *p* is unspecified.

Searching backwards is sometimes called **regression** planning. The principal question in regression planning is this: what are the states from which applying a given action leads to the goal? Computing the description of these states is called **regressing** the goal through the action.

consider the air cargo example. We have the goal

$$At(C_1, B) \wedge At(C_2, B) \wedge \ldots \wedge At(C_{20}, B)$$

The action will work only if its preconditions are satisfied. Therefore, any predecessor state must include these preconditions:

$$In(C_1, p) \; \mathrm{A} \; At(p,B)$$

Thus, the predecessor description is

$$In(C_1, p) \wedge At(p, B) \wedge At(C_2, B) \wedge \ldots A\ At(C_{20}, B).$$

Given a goal description G, let A be an action that is relevant and consistent. The corresponding predecessor is as follows:

a Any positive effects of A that appear in G are deleted.
a Each precondition literal of A is added, unless it already appears.

for example, the predecessor description in the preceding paragraph is satisfied by the initial state

$$In(C_1, P_{12}) \wedge At(P_{12}, B) \wedge At(C_2, B) \wedge \ldots \wedge At(C_{20}, B)$$

with substitution *{p/P12}*T. he substitution must be applied to the actions leading from the state to the goal, producing the solution

$$[Unload(C_1, P_{12}, B)].$$

Heuristics for state-space search

It turns out that neither forward nor backward search is efficient without a good heuristic function.

in STRIPS planning, the cost of each action is 1, so the distance is the number of actions.

Finding the exact number is NP hard, but it is possible to find reasonable estimates most of the time without too much computation. We might also be able to derive an **admissible** heuristic-one that does not overestimate.

There are two approaches

relaxed problem(admissible Heuristics)

subgoal independence assumption: the cost of solving a conjunction of subgoals is approximated by the sum of the costs of solving each subgoal *independently.*

The components of a planning system with suitable examples.

Planning refers to the use of methods that focus on ways of decomposing the original problem into appropriate subparts and on ways of recording and handling interaction among the subparts as they are detected during the problem solving process.

Components Of Planning System

The important components of planning are

1. Choosing the best rule to apply next based on the best variable available heuristic information.
2. Applying the chosen rule to compute the new problem state that arises from its application.
3. Detecting when a solution has been found.
4. Detecting dead ends so that they can be abandoned and the system's effort directed in correct direction.
5. Repairing an almost correct solution.

1. Choosing rules to apply:

- First isolate a set of differences between the desired goal state and the current state.
- Detect rules that are relevant to reduce the differences.

If several rules are found, a variety of heuristic information can be exploited to choose among them.

This technique is based on the means end analysis method.

2. Applying rules:

Applying the rules is easy.

Each rule specifies the problem state that would result from its application.

We must be able to deal with rules that specify only a small part of the complete problem state.

Different ways to do this are

Describe, for each action, each of the changes it makes the state description.

A state was described by a set of predicates representing the facts that were true in that state. Each state is represented as a predicate.

The manipulation of the state description is done using a resolution theorem prover.

3. Detecting a solution

A planning system has succeeded in finding a solution to a problem when it has found a sequence of operators that transforms the initial problem state into the goal state.

Any of the corresponding reasoning mechanisms could be used to discover when a solution has been found.

4. Detecting dead ends

As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution. The same reasoning mechanisms that can be used to detect a solution can often be used for detecting a dead end.

If the search process is reasoning forward from the initial state, it can prune any path that leads to a state from which the goal state cannot be reached.

If the search process is reasoning backward from the goal state, it can also terminate a path either because it is sure that the initial state cannot be reached or little progress is being made.

In reasoning backward, each goal is decomposed into sub goals. Each of them may lead to a set of additional sub goals. Sometimes it is easy to detect that there is now way that the entire sub goals in a given set can be satisfied at once. Other paths can be pruned because they lead nowhere.

5. Repairing an almost correct solution

Solve the sub problems separately and then combine the solution to yield a correct solution. But it leads to wasted effort.

The other way is to look at the situations that result when the sequence of operations corresponding to the proposed solution is executed and to compare that situation to the desired goal. The difference between the initial state and goal state is small. Now the problem solving can be called again and asked to find a way of eliminating a new difference. The first solution can then be combined with second one to form a solution to the original problem.

When information as possible is available, complete the specification in such a way that no conflicts arise. This approach is called least commitment strategy.

It can be applied in a variety of ways.

- To defer deciding on the order in which operations can be performed.
- Choose one order in which to satisfy a set of preconditions, we could leave the order unspecified until the very end. Then we could look at the effects of each of the sub solutions to determine the dependencies that exist among them. At that point, an ordering can be chosen.

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. This means that they cannot take advantage of problem decomposition

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

Consider the simple problem of putting on a pair of shoes. We can describe this as a formal planning problem as follows:

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

Any planning algorithm that can place two actions into a plan without specifying which comes first is called a **partial-order planner.** Figure 11.6 shows the partial-order plan that is the solution to the shoes and socks problem.

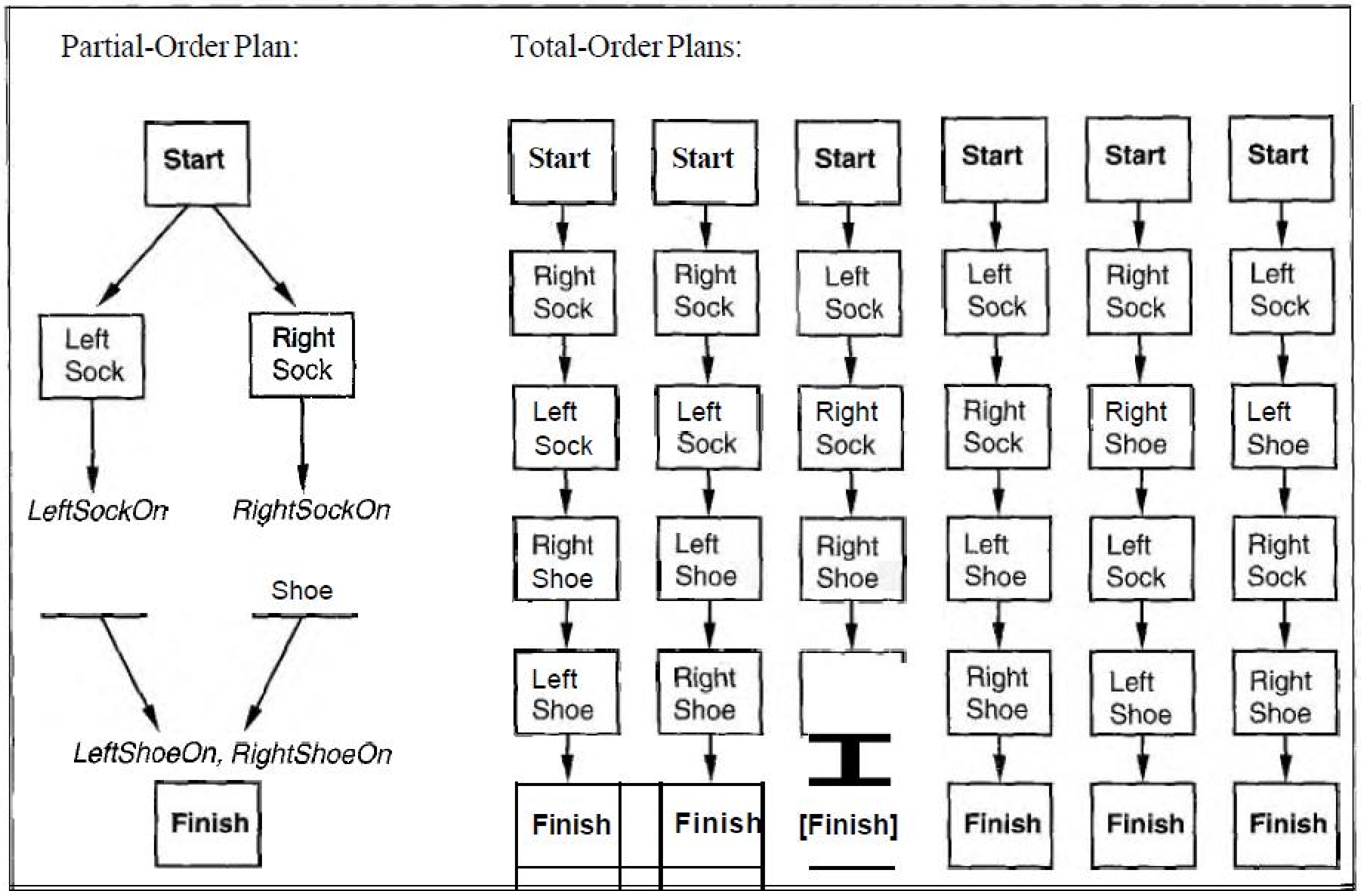


Figure 11.6 A partial-order plan for putting on shoes and socks, and the six corresponding linearizations into total-order plans.

Each plan has the following four components, where the first two define the steps of the plan and the last two serve a bookkeeping function to determine how plans can be extended:

- 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.

- A set of **ordering constraints.** Each ordering constraint is of the form

$$A \prec B,$$

read as "A before B" and means that action A xlust 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

$$RightSock \xrightarrow{RightSockOn} RightShoe$$

- **a** A set of **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

For example, the final plan in Figure 11.6 has the following components

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

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

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

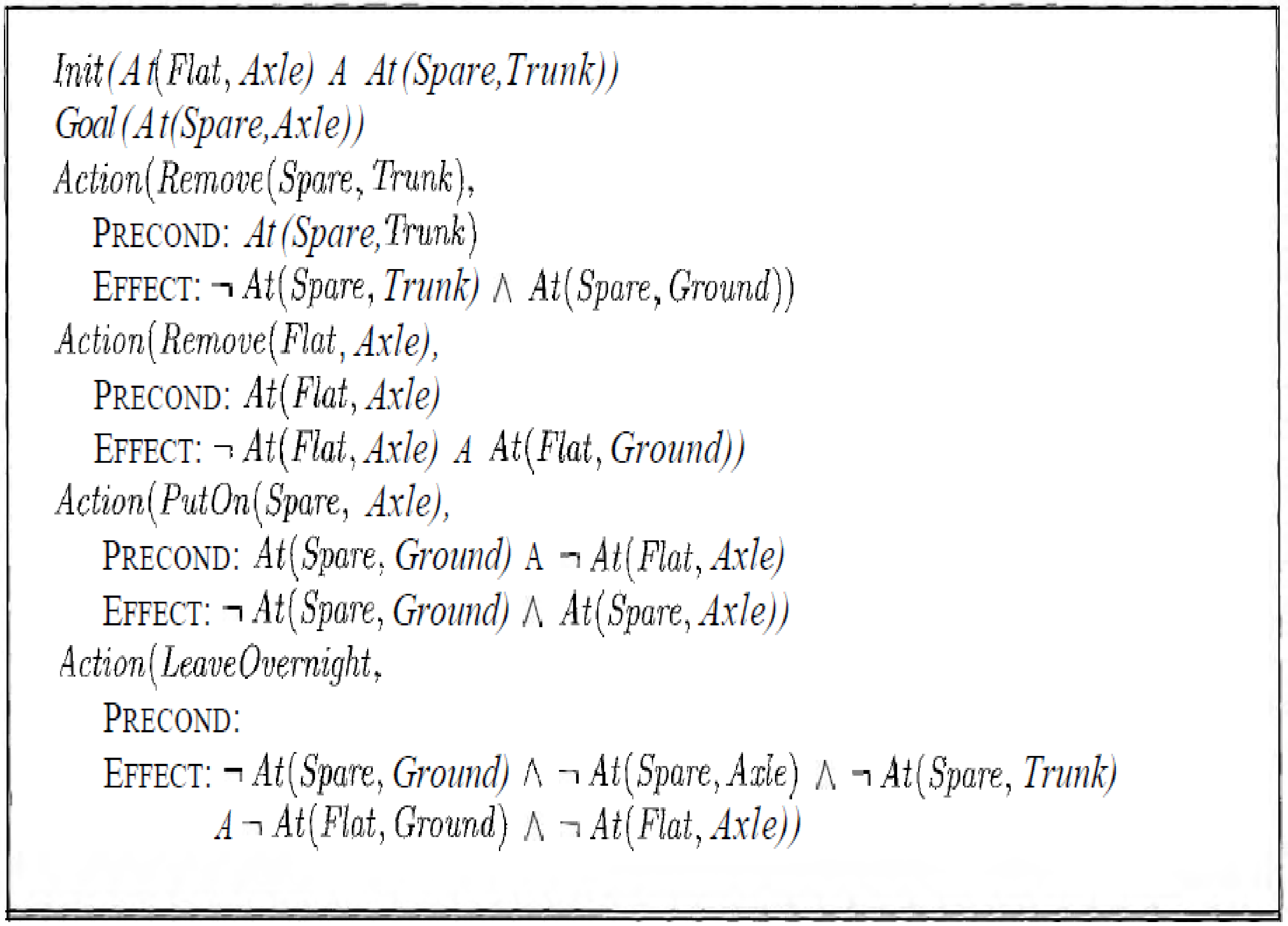
$\text{RightShoe} \xrightarrow{\mathit{RightShoeOn}} \mathit{Finish}, \mathit{LeftShoe} \xrightarrow{\mathit{LeftShoeOn}} \mathit{Finish}\}$

$\text{Open Preconditions:}\{\}$.

We define a **consistent plan** as a plan in which there are no cycles in the ordering constraints and no conflicts with the causal links.

A consistent plan with no open preconditions is a **solution.**

A partial-order planning example(FLAT 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*))

Figure 11.7 The simple flat tire problem description.

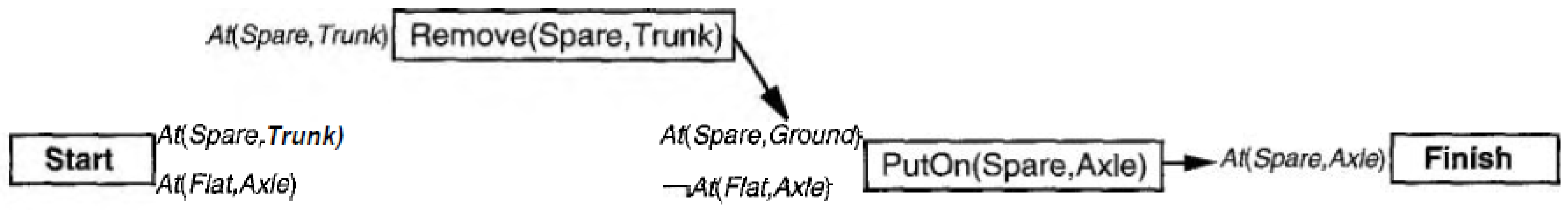


Figure 11.8 The incomplete partial-order plan for the tire problem, after choosing actions for the first two open preconditions. Boxes represent actions, with preconditions on the left and effects on the right. (Effects are omitted, except for that of the *Start* action.) Dark arrows represent causal links protecting the proposition at the head of the arrow.

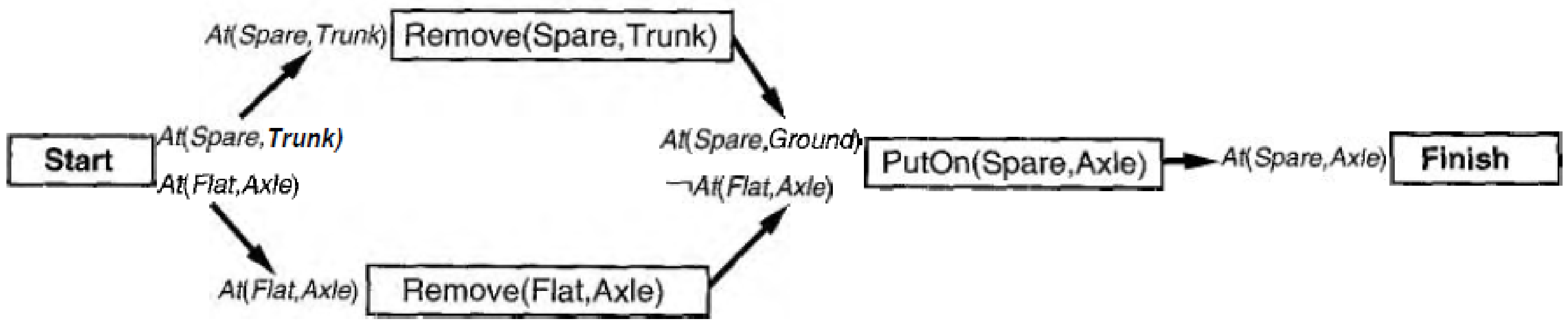


Figure 11.10 The final solution to the tire problem. Note that *Remove*(*Spare*, Trunk) and *Remove*(*Flat*, *Axle*) can be done in either order, as long as they are completed before the *Put*On *(Spare,Axle)* action.

Heuristics for partial-order planning

Compared with total-order planning, partial-order planning has a clear advantage in being able to decompose problems into subproblems. It also has a disadvantage in that it does not represent states directly, so it is harder to estimate how far a partial-order plan is from achieving a goal.

The heuristic function is used to choose which plan to refine. Given this choice, thealgorithm generates successors based on the selection of a single open precondition to work on.