Planning

CS 475

March 13, 2001

1 Introduction

A *planning problem* $\mathcal{P}$ is given by a triple (D, I, G) where D is a domain description, I is the initial state, and G is the final state. A *solution* to a planning problem (D, I, G) is a sequence of actions (in D) that, when executed from a state satisfying I, will bring the world to a state satisfying G.

To be able to find a solution to a planning problem $\mathcal{P}$ **using computers** we need to

- represent $\mathcal{P}$ to the computer, and
- have a program (*planner*) that is capable of solving $\mathcal{P}$.

In addressing the first issue, we need

- representation language for D (we have learned one language - the *Situation Calculus*)
- representation language for I and G

For simplicity, we will only consider the case where I and G are set of fluents.

Example 1.1 *In situation calculus notation, the planning problem in the previous lesson given in the next picture*

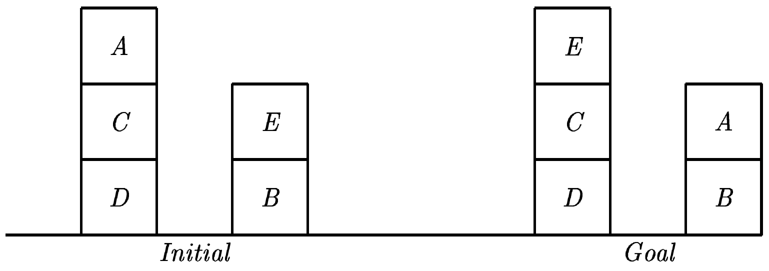

Figure 1: A Planning Problem in Block World

is given by $(\mathcal{T}', I, G)$ where $\mathcal{T}'$ is obtained from the situation calculus theory (previous note) by removing the set of axioms about the initial situations. I is the set of fluents which are true in S_0 (wrt. $\mathcal{T}$) – which is the set

$$\{On(A,C), On(B,T), On(E,B), On(C,D), On(D,T)\},$$

and G is the set

$$\{On(A,B), On(B,T), On(E,C), On(C,D), On(D,T)\}.$$

NOTE: There are another representation languages that can be used to represent and reason about actions as well (See the literature mentioned in Russel & Norvig's book.) There are also attempt to develop a more expressive language for goal representation. For instance, a goal can be a preferable trajectory instead of a set of fluents.

A *planner* is a program that generates solutions given a planning problem. In the last class, we have seen that a general purpose problem solver (GPPS) that solves problem by searching can be used to as a planner. Under this framework, an action is an operator and a state is a set of fluents whereas a situation is an action sequence. A search tree coincides with the situation tree of the problem assuming that we represent the problem in situation calculus. So, we can use all the search techniques to speed up the planning process. In the worst case, however, the planning process is **NP**-complete, and hence, no technique can help!

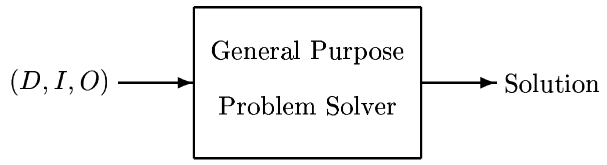

Figure 2: Planning as Search

Using a GPPS means that we need to have an implementation of the transition function between states. If we are going to implement the *Res* function we would need a theorem prover. This is generally computational impossible. This can be improved if we limit ourselves to a representation language that can make this process faster. One approach in this direction is the representation used in **STRIPS**. In this language, an action is represented by an operator with three components: a name, a set of preconditions, and a set of effects (both positive and negative effects). For example, selling a house will cause us to have money but no longer have the house – this is represented by $OP(ACTION : sellHouse, PRECONDITION : \{hasHouse\}, EFFECT : \{hasMoney, \neg hasHouse\})$. Another way to improve the performance of a GPPS is try to limit the search space. Experimental has showed that taking specific information of planning problems is a good way.

2 Partial-Order Planning

Using a GPPS to solve a planning problem, we search through the *state space* to find solutions. Remember what is a the search space of a problem? Planners that search through the search space to find plans are called *progression*.

Another way to find plans is to search backward, from the goal to the initial state. Planners working this way are called *regression* planners. We have seen that in GPPS, searching backward does not seem to be a good idea. Why then it can be good in planning? The reason for this confidence lies in the fact that planning has a special structure that makes it possible.

Instead of searching through the state space backward, we search through the *plan space* which is the set of ALL incomplete plans which we call a *partial plan*. We start out with a partial-plan and then extend it until we get a complete plan that solves the problem. This means that we are going to define a new problem whose search space is the plan space and the goal is to find a complete plan. The initial state for the problem could be plan!

Since we are going to use the search algorithm again, we need to define what are the operators of this new problem. A *refinement operator* takes a partial plan and adds constraints to it. A *modification operator* does anything else.

3 Representation for Plans

A *plan* is defined as a tuple (S, O, V, C) where

- S is a set of plan steps. Each step is one of the operators for the problem. (Each step is an action).
- O is a set of step ordering constraints. Each ordering constraint is of the form $s_i \prec s_j$, which is read as "s_i is before s_j" and means that s_i must occur sometimes before step s_j but not necessary immediately before s_j.
- V is a set of variable binding constraints. Each variable constraint is of the form $v = x$, where v is a variable in some step, and x is either a constant of another variable.
- C is a set of **causal links**. A causal link is written as $s_i \xrightarrow{c} s_j$ and read as "s_i achieves c for s_j." Causal links serve to record he purposes of steps in the plan: a purpose of s_i is to achieve the precondition c of s_j.

The set O contains a contradiction if it contains both $s_i \prec s_j$ and $s_j \prec s_i$ for some steps s_i and s_j.

The set V contains a contradiction if it contains both $v = A$ and $x = B$ for some variable x and two constants A and B.

A plan is *consistent* if there is no contradiction in the ordering or binding constraints sets.

A *complete plan* is one in which every precondition of every step is achieved by some other step. A step achieves a condition if the condition is one of the effects of the step, and if no other step can possibly cancel out the condition. A complete plan is also called a *total-order plan.*

A *solution* is a plan that is consistent and complete.

Example 3.1 *Consider the two partial-order plans in the following pictures:*

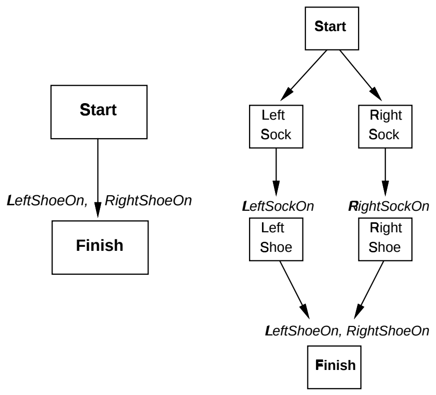

Figure 3: A partial-order plan to achieve the goal of having the shoe on!

The partial order plan on the right is represented by: (S, O, V, C) *where*

- $S = \{start, left\_sock, right\_sock, left\_shoe, right\_shoe, finish\}$
- $O = \{start \prec left\_sock, start \prec right\_sock, left\_sock \prec left\_shoe,$ $right\_sock \prec right\_shoe, left\_shoe \prec finish, right\_shoe \prec finish\}$
- $V = \emptyset$
- $C = \{left\_sock \overset{leftSockOn}{\longrightarrow} left\_shoe, right\_sock \overset{rightSockOn}{\longrightarrow} right\_shoe,$ $left\_shoe \overset{leftShoeOn}{\longrightarrow} finish, right\_shoe \overset{rightShoeOn}{\longrightarrow} finish\}.$

4 A Partial-Order-Planning Example

Given the shopping domain with the following operators:

- $OP(ACTION : Go(x),$
  $PRECONDITION : \{At(y)\},$
  $EFFECT : \{At(x), \neg At(y)\}),$
- $OP(ACTION : Buy(x),$
  $PRECONDITION : \{At(store), Sells(store, x)\},$
  $EFFECT : \{Have(x)\}).$

To encode the initial/goal state with
$I = \{At(Home), Sells(SM, Banana), Sells(SM, Milk), Sells(HWS, Drill)\}$ and
$G = \{At(Home), Have(Banana), Have(Milk), Have(Drill)\}$ we add the following operator to the domain:

- $OP(ACTION : Start,$
  $PRECONDITION : \emptyset,$
  $EFFECT : I),$
- $OP(ACTION : Finish,$
  $PRECONDITION : G,$
  $EFFECT : \emptyset).$

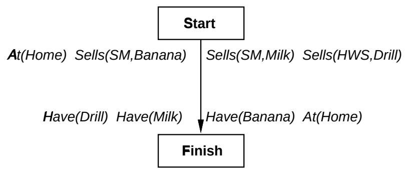
Start
At(Home) Sells(SM,Banana) Sells(SM,Milk) Sells(HWS,Drill)
Have(Drill) Have(Milk) Have(Banana) At(Home)
Finish

Figure 4: The initial plan for the shopping problem

Refinement - Step 1 - Achieving the *Sells* precondition by adding causal links

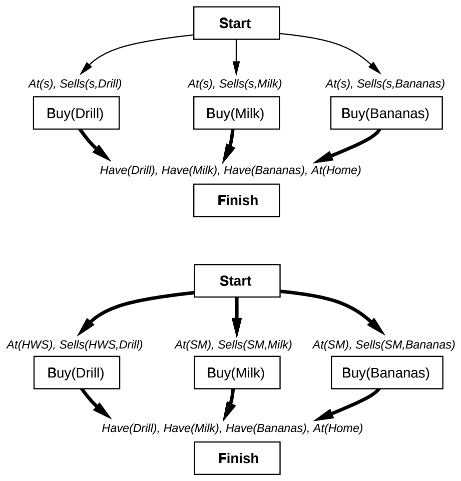

Figure 5: Refinement - Step 1

Refinement - Step 2 - Achieving the *At* precondition for the *Buy* actions

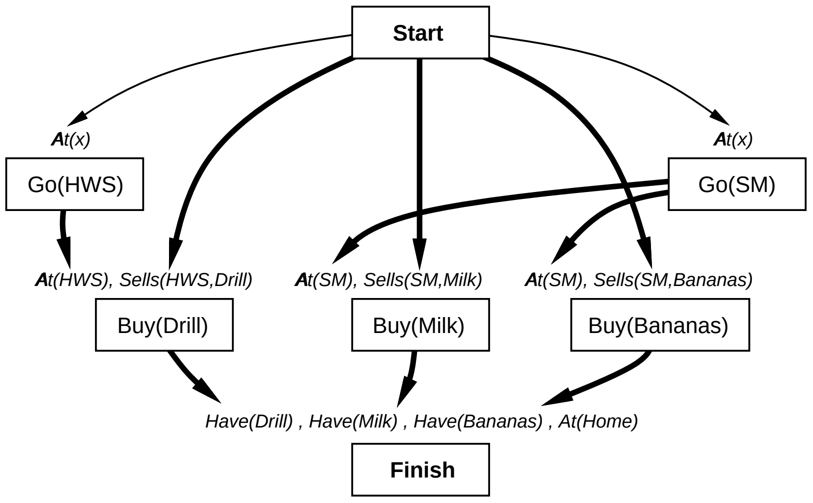
Start
At(x)
At(x)
Go(HWS)
Go(SM)
At(HWS), Sells(HWS,Drill)
At(SM), Sells(SM,Milk)
At(SM), Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill) , Have(Milk) , Have(Bananas) , At(Home)
Finish

Figure 6: Refinement - Step 2

Problem: we can not go to two places at the same time:

Protected *causal links* and *threat*: $Start \overset{At(Home)}{\longrightarrow} Go(SM)$ is a protected causal link. Adding the step $Go(HWS)$ is a threat of the link.

Resolving the problems of the shopping domain

Getting a solution

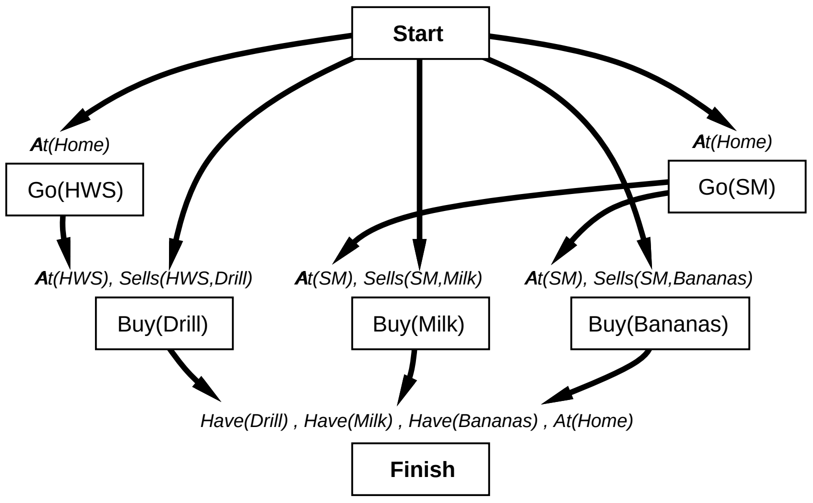
Start
At(Home)
Go(HWS)
At(Home)
Go(SM)
At(HWS), Sells(HWS,Drill)
At(SM), Sells(SM,Milk)
At(SM), Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill) , Have(Milk) , Have(Bananas) , At(Home)
Finish

Figure 7: Problem

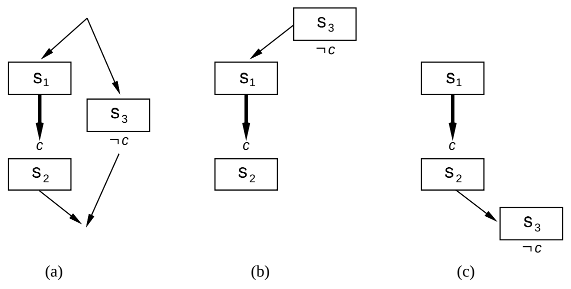

Figure 8: Demotion (b) and Promotion (c)

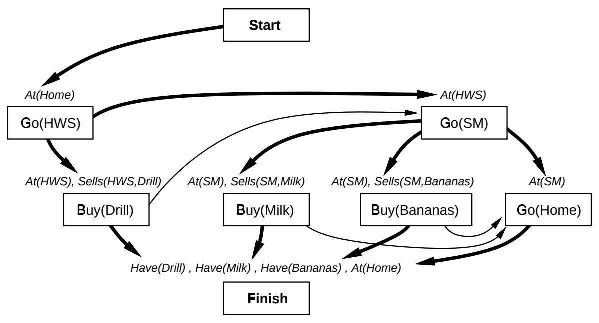

Figure 9: Demotion $Go(SM)$

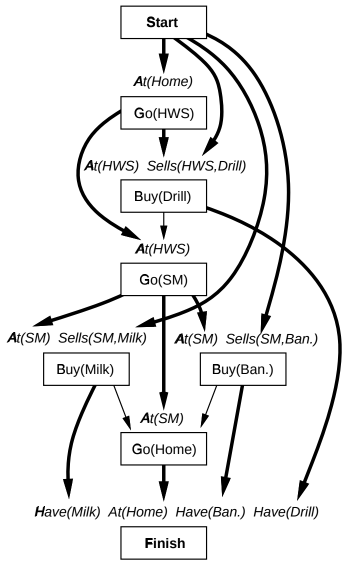

Figure 10: A solution