

## Partial Order Planning

### Lecture 14

#### Partial Order Planning

We have seen that the recursive planners that we examined in last lecture does not work well: it is slow, forces an ordering unnecessarily, and is incomplete.

In this lecture we see an alternative way to do planning. Unlike the solutions in the last lecture, it is complete, and does not force an ordering if it is not needed.

#### Reference:

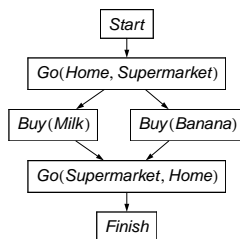
- Textbook Section 11.4–11.6

AI(0270)

AI(0270)-14.1

#### The graphical view

We can draw the plan graphically like this:



To execute the plan, we perform **serialization**: everytime we find one of the actions that has **no arrow pointer to it** and execute it.

In other words, an arrow limits the order so that **one actions is done before the other**.

AI(0270)-14.2

#### A clearer diagram

To help us verify a plan, we will draw more elaborated diagrams for them:

- We show **the conditions and effects of each operator**.
- For consistency, the **Start** step has all the initial states as effects, and the **Finish** step has all the goal conditions as preconditions. All actions must be between Start and Finish.
- We show **the established causes** of each precondition, by drawing **bold arrows**, called **causal-links**, from causes to pre-conditions.  
If a step  $S_1$  causes condition  $c$  of  $S_2$ , we write  $S_1 \xrightarrow{c} S_2$ .
- The arrows we have seen are called **ordering constraints** of the plan. For an arrow from  $S_1$  to  $S_2$ , we write  $S_1 < S_2$   
If we also have a casual link, we will not draw the thin arrow to keep it clean.

A plan is specified by (1) the **steps**, (2) a set of ordering **constraints**, and (3) a set of **causal links**.

AI(0270)-14.4

#### Why the plan is considered to be correct?

What we can do to improve the situation of the last lecture?

- One possibility: **allow parts of a plan to be unordered**, i.e., in any order, while keeping the remainder of the plan **ordered**.
- E.g., if we need to buy banana and milk, both from supermarket, the plan would look like:
  - Go from home to supermarket
  - Buy banana after going to supermarket, but before going home.
  - Buy milk after going to supermarket, but before going home.
  - Go from supermarket to home.
- Note that we are **very careful** so that the **order** between buying banana and buying milk is **not specified**.

In the point of view of the **execution** of the plan:

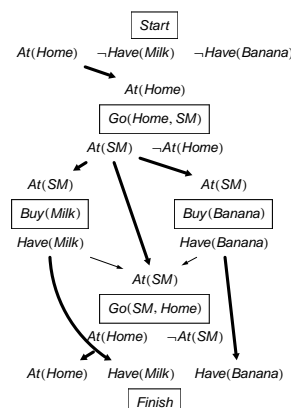
- Any serialization guarantees** that the goal is achieved.  
This is what we really care. But we have to be able to check for this, so...

In the point of view for **verifying** the plan:

- The diagram of the plan has **no cycle**.  
Otherwise we can never serialize it.
- Each precondition of each step  $s_2$  is **caused by a step  $s_1$  before it**. (A precondition that is not caused is said to be **open**.)  
A step  $s_1$  is before another step  $s_2$  if there is a path from  $s_1$  to  $s_2$ . A step  $s_1$  causes a condition  $c$  if  $s_1$  has  $c$  as effect.
- If a precondition of  $s_2$  is caused by a step  $s_1$ , there is **no step  $s_3$  that can execute between  $s_1$  and  $s_2$  which deletes the precondition**.  
 $s_3$  can execute between  $s_1$  and  $s_2$  if neither  $s_3$  is before  $s_1$  nor  $s_3$  is after  $s_2$ .

AI(0270)-14.3

#### Example of plan diagram



Why this is a correct plan?

Clearly, there is no open pre-conditions.

There is only one place where ordering is not specified: *Buy(Milk)* and *Buy(Banana)*

But they **don't interfere each other** by deleting conditions needed by each other. So it is a correct plan.

**Question:** what if the two ordering constraints are not there?

AI(0270)-14.5

### Searching in the plan space

- Our next question is **how to come up with a good plan**.
- The idea: **Start with a plan** that is “minimal”: it just contains the *Start* node and the *Finish* steps, with the ordering constraint  $Start < Finish$ .  
Note that this is a plan. It is just not a correct plan, since some pre-condition of the *Finish* step is not caused by other steps.
- Then we repeatedly **pick an open-condition** do one of the following:
  - Add a **causal link** from another step.
  - **Choose** a step with the open condition as effect. Add it to the plan.
 Both might require us to add ordering constraints to resolve conflicts.
- We can view that **we are doing some sort of uniform-cost search**, with the states being plans, and with the above operators.  
We say that we search in the plan space rather than situation space.

AI(0270)-14.6

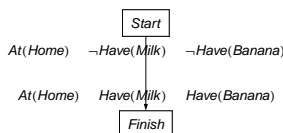
### How we come up with the plan: an example

Rather than showing you the full algorithm at the beginning, Let's start by solving our problem to get milk and banana. Formally:

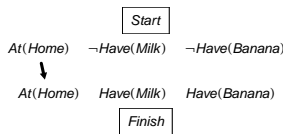
- **Start state:**  $At(Home), \neg Have(Milk), \neg Have(Banana)$
- **Goal:**  $At(Home), Have(Milk), Have(Banana)$
- **Operator**  $Go(here, there)$ :
  - Precondition:  $At(here)$ , with restriction that  $here \neq there$
  - Effect:  $At(there), \neg At(here)$
- **Operator**  $Buy(Milk)$ :
  - Precondition:  $At(SuperMarket)$
  - Effect:  $Have(Milk)$
- **Operator**  $Buy(Banana)$ :
  - Precondition:  $At(SuperMarket)$
  - Effect:  $Have(Banana)$

AI(0270)-14.7

#### First plan



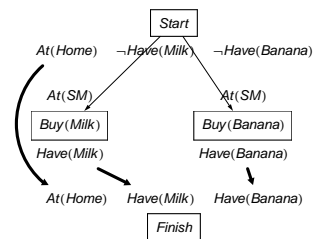
We can add the causal link for  $At(Home)$ , leaving us...



And continuing, we can find the following...

AI(0270)-14.8

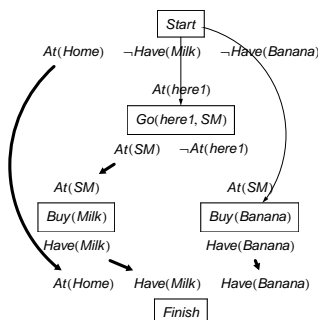
#### The incorrect path...



But now we hit a problem... we need to have a step that establish  $At(SM)$ , which can only be done by adding the  $Go(here, SM)$  action.

AI(0270)-14.9

#### The problem shows up...



We can make *here1* to be *Home*, but then it **conflicts with the causal link** from *Start* to *Finish*.

We can make *here* to be something else, but it just delay the problem. Sooner or later we have to Go from Home, which deletes  $At(Home)$ .

AI(0270)-14.10

#### What to do now?

- Now this part of the search tree **will not** give rise to any solution.
- But will we **get stuck** in it forever? Not quite. Our search algorithm will do uniform cost search (or better, iterative-deepening DFS), and the cost will depend on the number of steps needed.
- The minimum cost solution involves just 4 steps, so in the worst case we just expand three  $Go()$  that adds the  $Go()$  action.
- **We can leave here uninstantiated**, so we don't need to check all possible choices of it. This reduces the branching factor to 1.
- Both the *Buy* steps don't have backtracking choices, because they are the only steps to cause  $Have(Milk)$  and  $Have(Banana)$ .  
We can do better by treating it as CSP and use most-constrained-variable.
- So the only backtracking to do is to **establish  $At(Home)$  by an alternative way**, i.e., having a step for it.

AI(0270)-14.11

### So we get this...



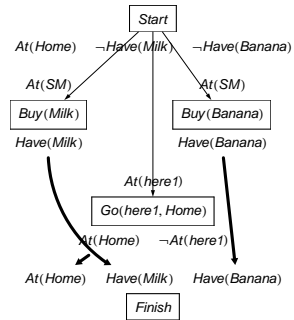
Once we get here, we can do all the trivial things: add a step to establish *Have(Milk)*, and another to establish *Have(Banana)*.

Or we might be **unlucky again** and pick *At(here1)* to establish. But this time we won't make *here1 = Home*, as this results in the disallowed *Go(Home, Home)*.

Let's assume we are lucky this time.

AI(0270)-14.12

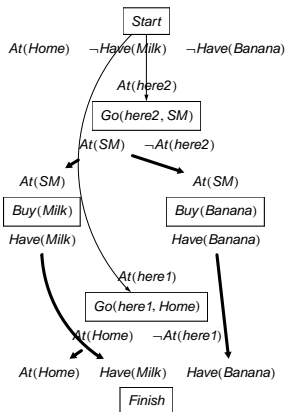
### More progress



Let's assume that we pick *At(SM)* to establish, and add the required step. Otherwise, we will pick *At(here1)* to establish, and add another step anyway.

AI(0270)-14.13

### And we continue to get closer...



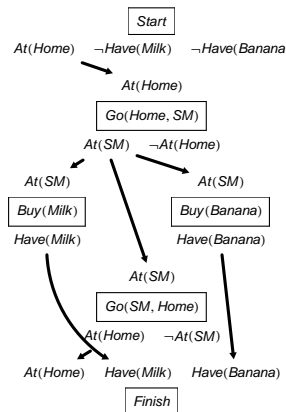
At this point we are close to a correct plan we need, except that the two *At(here)* preconditions.

We can just have *here2 = Home* and **add the causal link**.

We can also have *here1 = SM* and add the causal link. But there is a problem...

AI(0270)-14.14

### Dealing with conflicts



The step *Go(SM, Home)* deletes *At(SM)*...

which is **protected between** *Go(Home, SM)* and *Buy(Milk)*...

and the *Go(SM, Home)* step **can execute between the two steps!**

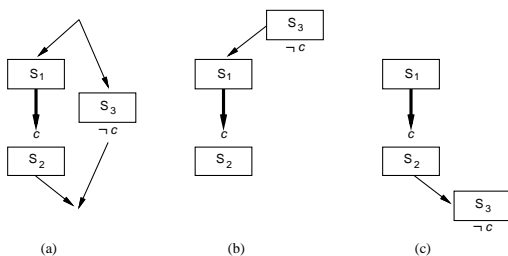
One way is to make sure that *Go(SM, Home)* **must execute before** *Go(Home, SM)* (**demotion**), but the causal link disallow that.

So we make sure *Go(SM, Home)* **must execute after** *Buy(Milk)* and *Buy(Banana)* (**promotion**), by adding ordering constraints.

AI(0270)-14.15

### Promotion and demotion

This is the general case for promotion and demotion...



So if we have the situation depicted by (a), we have a conflict, which can be handled by either (b) demotion or (c) promotion.

Now you should go back to see why the conflict of our first plan cannot be fixed this way.

AI(0270)-14.16

### The algorithm POP (Partial Order Plan)

- Make a plan that contains the *Start* node and the *Finish* steps, with the ordering constraint  $Start < Finish$ .
- While there is an open condition, do the following:
  - **pick a step**  $S_1$  with an **open condition**  $c$ .
  - Choose either...
    - Find a step  $S_2$  that is not after  $S_1$  which has effect  $c$ , and **add a causal link**  $S_2 \xrightarrow{c} S_1$ .
    - **Choose** a new step  $S_3$  with effect  $c$ . Add  $S_3$  to the plan, **add the causal link**  $S_3 \xrightarrow{c} S_1$ , and **add the ordering constraints**  $Start < S_3$  and  $S_3 < S_1$ .
- If there is a conflict, **resolve it** by choosing to do either promotion or demotion (thus **adding ordering constraints**).

AI(0270)-14.17