Class Notes - 2/10/2004 – 2/12/2004

I. Demonstrate how graph plan works using the Flat Tire Example:

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: none
EFFECT: ¬At(Spare, Ground) ^ ¬At(Spare, Axle) ^ ¬At(Spare, Trunk)
^ ¬At(Flat, Ground) ^ ¬At(Flat; Axle))

First, we will construct a planning graph. (Note: In these notes, we will use S for Spare, A for Axle, F for Flat, and G for Ground.)

Review: Graph plan, and divide execution into time steps. There are two types of nodes (proposition level and action) and three types of links.

Insert Graph here,
S0 contains all the literals.

So

At(F, A)

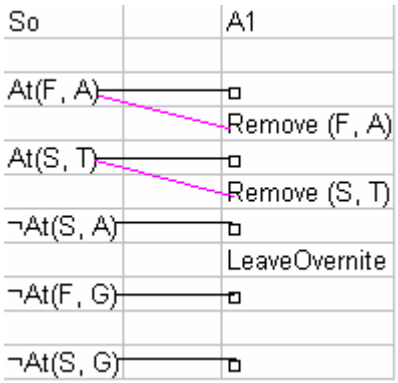
At(S, T)

¬At(S, A)

¬At(F, G)

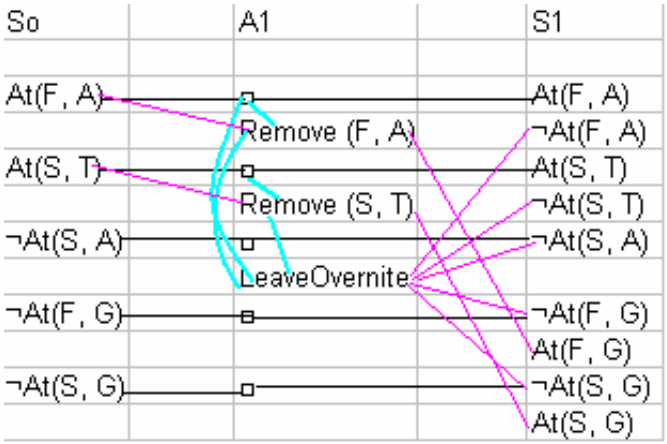
¬At(S, G)

A1 contains all the actions that can be executed if preconditions are satisfied. For each of the literals, there is also action for [called] No-Op.



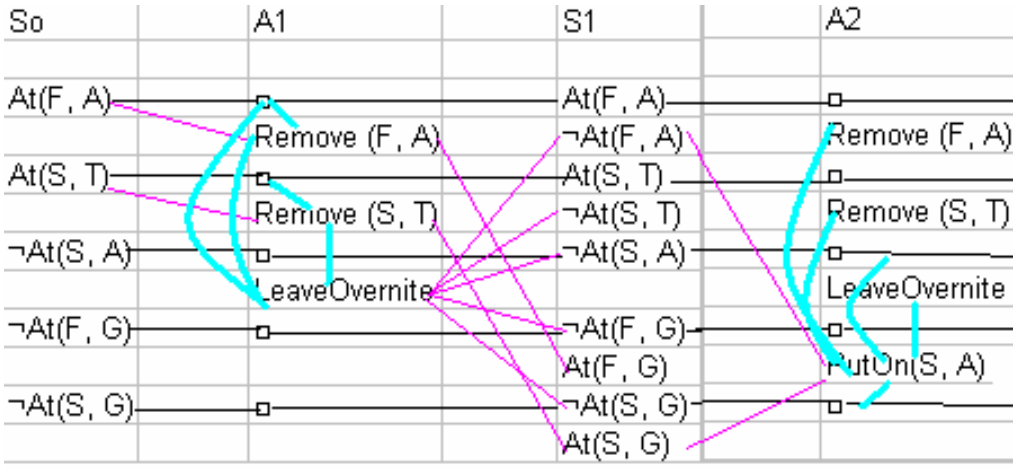
[LeaveOvernight]

S1. Contains most of the literals, but the goal is not achieved yet; therefore, we will continue to search.

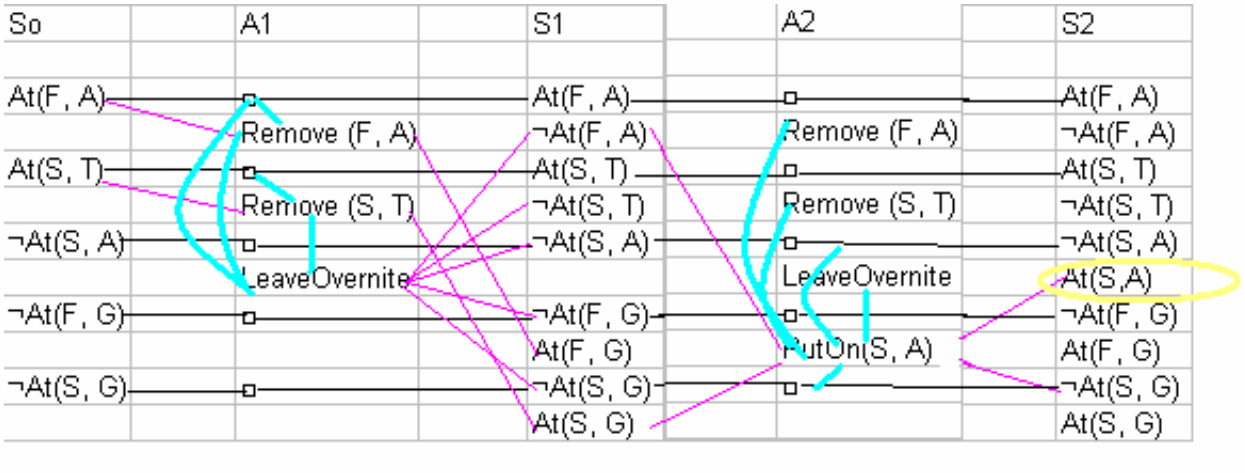


Notice that we have also identified that mutex pairs (in blue). Recall that there are three types of mutex. 1) two actions cannot occur at the same time if they have inconsistent effect. 2) action has precondition interference with the effect of the other. And 3) one action require negate the precondition of the other.

A2. Contains a newly added action PutOn.



S2. All possible literals are here. The goal is also founded here. At this stage, we can try to extract the solution using a backward search.



Solution extraction:

G = { At (S, A)}

The only action that can lead to the achievement of the goal is this one:
A2 = {PUTON (S, A)}

The goal now has been changed, our new goal has become: [it is the precondition of PUTON(S,A)]
G1 = {Not[¬] At (F, A), At (S, G)}

Now we need to find a set of actions that achieve this:

For (Not (At (F, A)) there are two options:
{Remove(F, A), LeaveOvernight}

For (At (S, G)) we have only one option:
{Remove(S, T)}

Only two combinations can result in the achievement of G1:

{Remove (F, A), Remove (S, T)}
{LeaveOvernight, Remove (S, t)}

Since one of the rules for selecting the correct action is that no pair of mutex should exist [in the set of selected actions], we can eliminate the second set of combination.

A1 = {Remove (F, A), Remove (S, T)}
G0 = {At (S, T), At (F, A)}
Since this is the initial step [since G0 holds in the initial state], we know we have found a solution.

II. Partial Order Planning

The idea of partial order planning is least commitment (only specify needed details, leave others open).

We often encounter problems where there is no need to commit to a strict sequencing of actions. For example:
In the Spare Tire Problem, On A1 {Remove(F, A), Remove (S, T)}, we don't really care which action comes first.

The search-space for Partial Order Planning is plane space.

One of the advantages of Partial Order Planning is that in real life, we can decompose problems into parts, then combine the results. This type of problem solving cannot be implemented in other types of planning we have learned so far.

For example:

Point A and point B are located on either side of a lake. In order to reach from one point to another, one must cross the bridge. For this example, we can divide the search into two parts: one from point A to the bridge and one from point B to the bridge.

This kind of search cannot easily be done using state-space search.

What must be represented in search problems:
A) Possible solution
B) Check to see if the solution is indeed the solution.
C) Generate new possible solutions based on current solution.

Represent Partial Planning as possible solution. [POP as search – Possible solution]
Partial Plan:

1) A set of actions that make up the steps of the plan.
Example: {A, B, C, D}

2) A set of ordering constraints.
Example: {A < B}
Which reads "A before B" and means that A has to be executed before B.
Note: it must describe a proper partial order. (Example A < B and B[A] > B is a contradiction)

3) A set of causal links
Example: $\{A \xrightarrow{p} B\}$
Which reads "A achieves p for B". It means that P is an effect of A and a precondition of B.

4) A set of open preconditions (preconditions that are not achieved by some action in plan).

Example: [Two special actions that are added to planning problem by POP algorithm]
Action Start
Precondition: none.
Effect: initial condition.

Action Finish
Precondition: all the literals in the goal of the planning problem.
Effect: none.

We can generate a new partial plan by adding new actions to the existing ones, adding order constraints, and changing the causal link.

Then we will check to see if open precondition is empty. If it is, we have a plan.

A plan is consistent if there is no cycle in ordering constraint and no conflict:

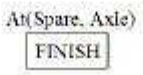
- $\{ A < B, B < A \} \subseteq OC$
- C conflicts with $A \xrightarrow{p} B$ if C has $\neg p$ as an effect and C could occur after A and before B

If P is a consistent partial plan with OP = ϕ, then every linearization that obeys the ordering constraints will be a valid sequential plan.

Example: the flat tire problem

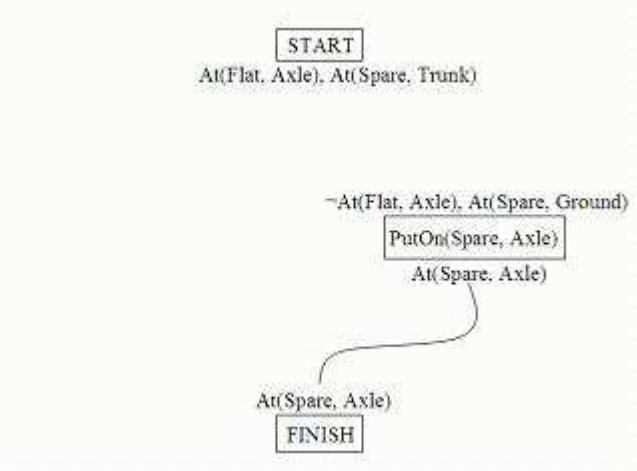
Situation 0:

OC = { Start < Finish }
CL = ϕ
OP = { At(Spare, Axle) }



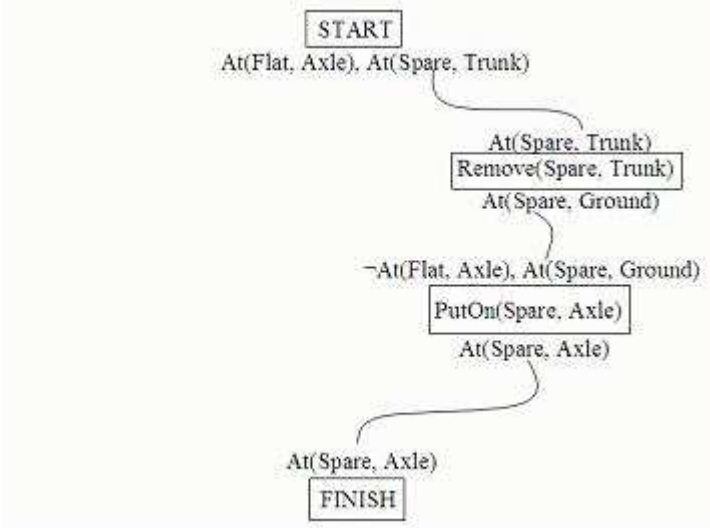
Situation 1:

Action 1: PutOn(Spare, Axle)
OC = { Start < 1, 1 < Finish }
CL = PutOn(Spare, Axle) $\xrightarrow{At(Spare,Axle)}$ Finish
OP = { ¬At(Flat, Axle), At(Spare, Ground) }



Situation 2:

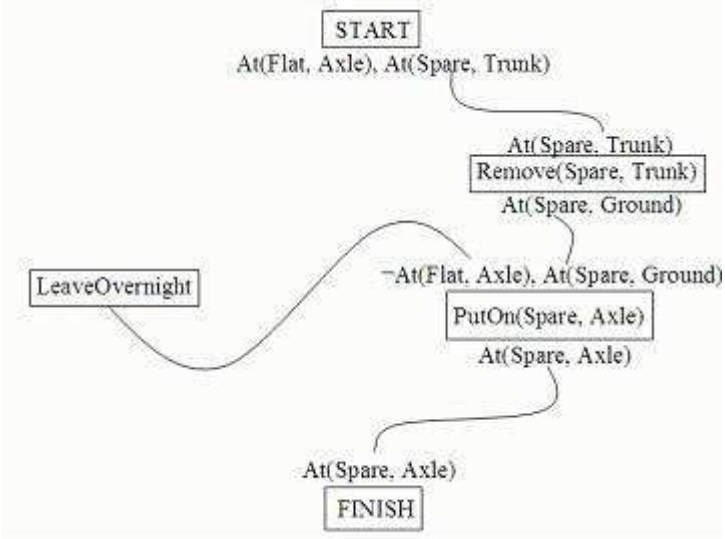
Action 2: Remove(Spare, Trunk)
OC = { Start < 2, 2 < 1, 1 < Finish }
CL = Remove(Spare, Trunk) $\xrightarrow{At(Spare,Ground)}$ PutOn(Spare,Axle)
Start $\xrightarrow{At(Spare,Trunk)}$ Remove(Spare, Trunk)
OP = { ¬At(Flat, Axle) }



Situation 3:

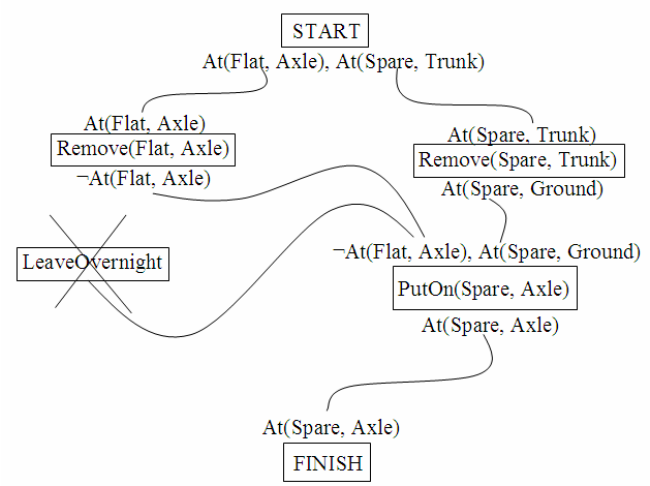
Action 3: LeaveOvernight
CL = LeaveOvernight $\xrightarrow{At(Spare,Ground)}$ PutOn(Spare, Axle)

In the set of ordering constraints, LeaveOvernight should be before Action 1, but it conflicts with Remove(Spare, Trunk) $\xrightarrow{At(Spare,Ground)}$ PutOn(Spare,Axle)
So we would have LeaveOvernight before Action 2, which would also be a conflict, and would lead us to have it before Start.
The action LeaveOvernight is thus impossible.



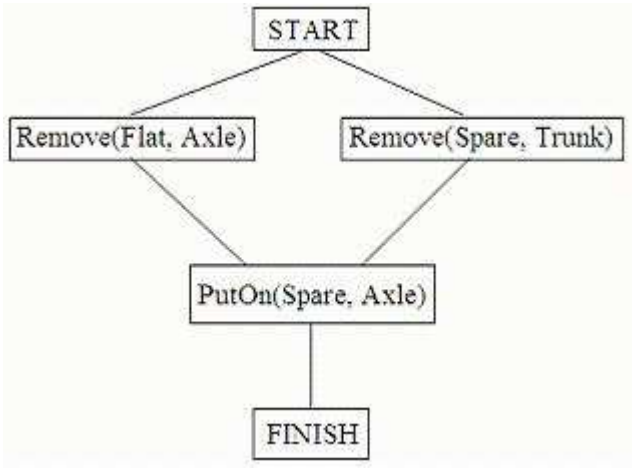
Situation 4:

Action 4: Remove(Flat, Axle)
OC = { Start < 4, 4 < 1, 4 < Finish }
CL = Remove(Flat, Axle) $\xrightarrow{\neg At(Flat,Axle)}$ PutOn(Spare,Axle)
Start $\xrightarrow{At(Flat,Axle)}$ Remove(Flat, Axle)
OP = { }



The set of open preconditions is empty, which means that we have a plan. So we have the following linearization:

Start, Remove(Flat, Axle), Remove(Spare, Trunk), PutOn(Spare, Axle), Finish
Start, Remove(Spare, Trunk), Remove(Flat, Axle), PutOn(Spare, Axle), Finish



Summary

1. Start with < {Start, Finish}, {Start < Finish}, ϕ, Goal >
2. If there is an open precondition of an action B, pick some *p*, find an action [A] that achieves *p*.
    - A $\xrightarrow{p}$ B (add it to the set of Causal Links)
    - If A is new: Start < A < Finish
    - Find C that conflicts with A and B [A $\xrightarrow{p}$ B], and resolve it (put C < A or B < C).

Added 1:

If the set of open precondition is empty then returns the plan, if there exists some open precondition p such that for every action A achieving p, adding A results in an inconsistent plan, the algorithm stops with failure.

Added 2:

We also talk about the subgoal independent assumption. The Sussman anomaly shows that this assumption is not always good. To date, this assumption has been removed by planners. However, this assumption is often used to compute heuristics.