

The problem of variable instantiations

Lecture 15

Dealing with variables in POP

Now it is time to deal with the remaining problem: **how to deal with the variables**. We will see that **unification plays a key role** again in planning.

Reference:

- Textbook Section 11.7

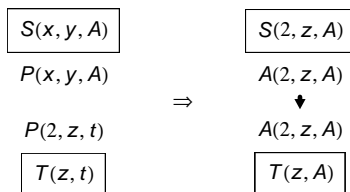
- Up til now, instantiation of variables is “magic”: when we need it, it will be there.
- We **can actually implement** this magic: whenever we choose an operator, we choose its variables as well.
- But just like blind use of FOL rules, this is **not very efficient** because it results into a lot of back-tracking.
- Instead, we want **variables to be chosen by unifications** during the planning process.
- **Where can we put the unification?** We can do it when we **find operators for pre-conditions**.
- Whenever we find an operator to cause a pre-condition, we allow operators with an effect which **unifies with the pre-condition**.

AI(0270)

AI(0270)-15.1

Example

So the unification takes place like this:



Instead of blindly choosing values for x, y, z and t , we choose it **because we have chosen the particular operator**.

Because the operator does not dictate a value for z , it remains *unbounded*. But we have **made sure that $y = z$ by unification**.

AI(0270)-15.2

AI(0270)-15.3

Representation of plans

Now our intermediate plans may contain **unbounded variables**, and when we search, we **bound the variables** by unifications.

We need to modify our representation a bit to accommodate this. A plan consists of:

- A set of operators, including the Start operator and the Finish operator. Each has a **different set of variables**.
- A set of **bindings** that is already made to the operators.
- A set of **causal links** in the form $S_1 \xrightarrow{c} S_2$. Here, c should be an effect of S_1 and a pre-condition of S_2 **after the bindings are made**.
- A set of **ordering constraints** in the form $S_1 \prec S_2$.
Again, if $S_1 \xrightarrow{c} S_2$, then we automatically have $S_1 \prec S_2$.

Our modification to the algorithm

Before we use unification to choose variables, we have these in POP **after we pick an open condition c** :

- Find a step S_2 that is not after S_1 which has effect c , ...
- **Choose** a new step S_3 with effect c

To use unification to choose variable, we change them to...

- Find a step S_2 that is not after S_1 which has effect c_1 , where $UNIFY(c, c_1) = \theta$... Add θ to the set of bindings.
- **Choose** a new step S_3 with effect c_1 , where $UNIFY(c, c_1) = \theta$ Add θ to the set of bindings.

So the changes are pretty trivial, except one thing...

Is it a threat to causal links?

In the past, when plans have variables instantiated, we say...

- A step S_3 conflict with (threaten) a causal link $S_1 \xrightarrow{c} S_2$ if S_3 **delete c and can be executed between S_1 and S_2** .

But now... both c and the delete list of S_3 can have **unbounded variable!**

E.g., if we have protected $At(SuperMarket)$ between S_1 and S_2 , but another step S_3 deletes $At(x)$. What to do?

- We can resolve it by **making a binding for x now**. If x is chosen to be *SuperMarket*, we **use promotion or demotion**. But then this binding will be done blindly, not via unification. \Rightarrow Slow.
- We can resolve it by **choosing either to add a constraint $x \neq SuperMarket$ to S_3 , or making $x = SuperMarket$ and do promotion or demotion immediately**. But then we need dealing with inequality.

AI(0270)-15.4

AI(0270)-15.5

An easier approach

Or we can say that it is **not (yet) a conflict**. We can **recognize a conflict only when the variables are actually bounded**.

- How to **detect** such conflicts? In the simplest solution, we can **check the whole plan every time we have added a causal link**.
But this is a bit costly: for each causal link, we need to check each step to see whether it can be executed between the causal link, and whether it deletes the condition. We can make it a bit faster by some bookkeeping.
- How to know **a solution is found**? What is our **requirements** now if variables can be **uninstantiated**?
- Answer part 1: perhaps we don't have such case.

If **there is no variable in effects** that is **not mentioned in preconditions**, there is no way to make a solution with uninstantiated variables. And this is the most common cases.

AI(0270)-15.6

Variable in the effect but not in precondition...

- But what if it is not the case, and **some variables in the effect is really not in precondition**?
- The first question: **what such steps really means**?
- It means that **we can choose any variable** in the constraint.
- This is really quite rare. But if we really have this, our final plan can have variables.
- Then a solution requires a special notion of a step causes (or **achieves**) a condition:

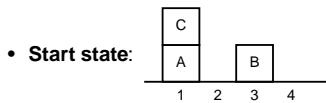
A step S_1 achieves a condition c of step S_2 after it if: (1) S_1 has c as its effect after binding, and (2) no step with a effect $\neg c'$ which can unify with $\neg c$ can execute between S_1 and S_2 .

- Usually you need to **instantiate all variables** anyway to say this.

AI(0270)-15.7

One more example: the Sussman Anomony

Let's try again the example that **fails the recursive planner**. (The failure is called "Sussman Anomony".) We focus on **how and when to do unification**, and when **conflicts** are handled.



• **Goal:** $On(A, B) \wedge On(B, C)$

• **Operator:**

Move(block, here, there):

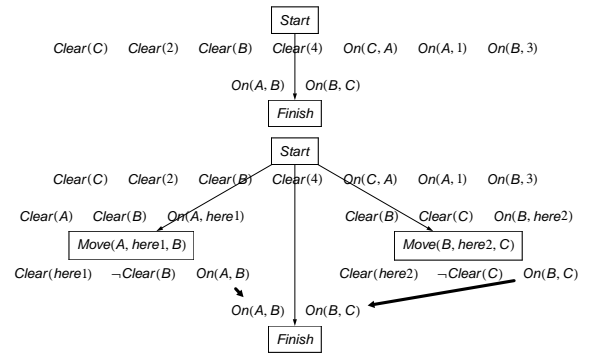
Pre-condition: $On(block, here) \wedge Clear(block) \wedge Clear(there)$,

Effect: $On(block, there) \wedge Clear(there) \wedge \neg Clear(here)$

AI(0270)-15.8

Step 1

As always, the first couple of steps are pretty trivial.



AI(0270)-15.9

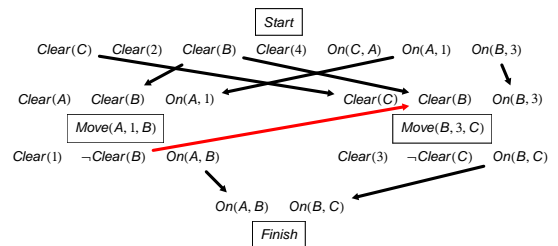
Before the next step...

- We have already done some unifications. Indeed, **to choose a step, we need to perform unification**.
- So we unified *block1* with *A*, *there1* with *B*, *block2* with *B*, and *there2* with *C*. These are the current **set of binding**.
- Note also that there is really **no backtracking possible**. This is the nice thing: we are only committing to things we must commit.
- We can leave *here1* and *here2* unbounded for **later binding**.
- On the other hand, there is one possible conflict here: the step *Move(A, here1, B)* might interfere with the pre-condition *Clear(B)*.
- Since currently we don't have a causal link on the *Clear(B)* condition, we won't be checking it yet.
Yet, we can **leave a mark** so that we know to check it when a causal link is added.

AI(0270)-15.10

Step 2

Now let's add more causal links and see what happens...

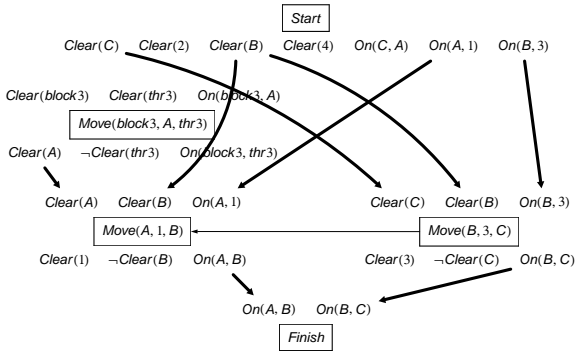


Once we have the causal link, we find a conflict, and has to resolve it. Do promotion in this case.

Now we have one more thing is needed: *Clear(A)*. Add a step...

AI(0270)-15.11

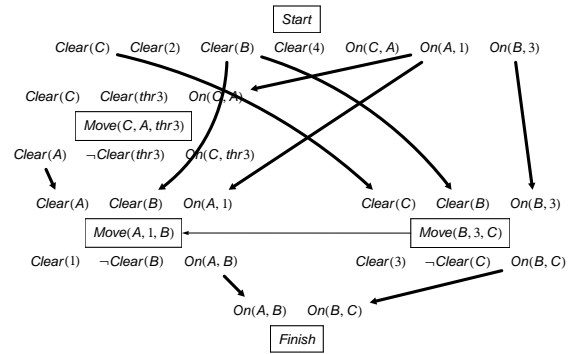
Step 3



Now we have more potential conflicts: $\neg\text{Clear}(thr3)$ can conflict with any *Clear* conditions.

AI(0270)-15.12

Step 4

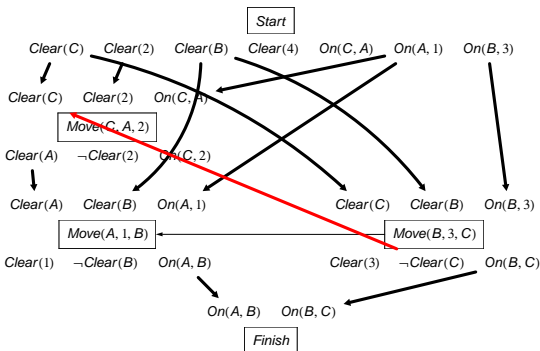


But we will ignore it, because *thr3* is still unbounded. If we start from the most constrained variable, we will start trying with *block3*.

AI(0270)-15.13

Step 5

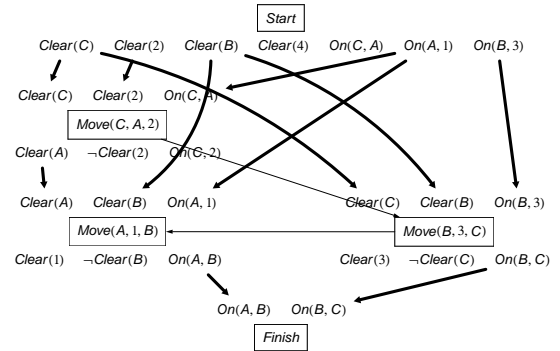
We get another potential conflict: *Move(B, 3, C)* can delete *Clear(C)* needed in *Move(C, A, thr3)*. But we wait until we have a causal link.



AI(0270)-15.14

Step 6

Now it is clear that we need a promotion. So we get the final plan...



AI(0270)-15.15

Some observations

- The optimal solution is found, unlike the recursive planner.
- It can easily be seen that **each solution has a corresponding plan**, and that **we can always search the plan**. So POP is a complete algorithm.
- In this case, all the “fine” ordering constraints are **done by promotions and demotions**.
This is a good sign that we might experience Sussman anomaly.
- Variables are chosen **only when we need to establish a pre-condition** via a **causal link**. So we can always use unification.
The “from” step gives a place for performing unification.
- Our problem is a “proper” one, in which **all variables are introduced in pre-conditions**. So the resulting plan is **fully instantiated**.

AI(0270)-15.16