POP: A Partial-Order Planner

In this lecture, we look at the operation of one particular partial-order planner, called POP. POP is a regression planner; it uses problem decomposition; it searches plan space rather than state space; it build partially-ordered plans; and it operates by the principle of least-commitment.

In our description, we'll neglect some of the fine details of the algorithm (e.g. variable instantiation) in order to gain greater clarity.

1 POP plans

We have to say what a plan looks like in POP. We are dealing with partially-ordered steps so we must give ourselves the flexibility to have steps that are unordered with respect to each other. And, we are searching plan-space instead of state space, so we must have the ability to represent unfinished plans that get refined as planning proceeds.

A plan in POP (whether it be a finished one or an unfinished one) comprises:

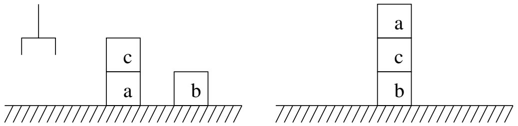
- A set of plan *steps*. Each of these is a STRIPS operator, but with the variables instantiated.
- A set of *ordering constraints*: $S_i \prec S_j$ means step S_i must occur sometime before S_j (not necessarily immediately before).
- A set of *causal links*: $S_i \xrightarrow{c} S_j$ means step S_i achieves precondition c of step S_j.

So, it comprises actions (steps) with constraints (for ordering and causality) on them.

The algorithm needs to start off with an *initial plan*. This is an unfinished plan, which we will refine until we reach a solution plan.

The initial plan comprises two dummy steps, called Start and Finish. Start is a step with no preconditions, only effects: the effects are the start state of the world. Finish is a step with no effects, only preconditions: the preconditions are the goal.

By way of an example, consider this start state and goal state:



Initial state   Goal state

These would be represented in POP as the following initial plan:

```
  Plan(STEPS: {S1: Op( ACTION:   Start,
                       EFFECT:   clear(b) ∧ clear(c) ∧
                                 on(c, a) ∧ ontable(a) ∧
                                 ontable(b) ∧ armempty),
               S2: Op( ACTION:   Finish,
                       PRECOND:  on(c, b) ∧ on(a, c))},
  ORDERINGS:  {S1 ≺ S2},
      LINKS: {})
```


This initial plan is refined using POP's *plan refinement operators*. As we apply them, they will take us from an unfinished plan to a less and less unfinished plan, and ultimately to a solution plan. There are four operators, falling into two groups:

- *Goal achievement operators*
  - *Add new step*: Add a new step S_i which has an effect c that can achieve an as yet unachieved precondition of an existing step S_j. Also add the following constraints: $S_i \prec S_j$ and $S_i \xrightarrow{c} S_j$ and Start $\prec S_i \prec$ Finish.
  - *Reuse existing step:* Use an effect c of an existing step S_i to achieve an as yet unachieved precondition of another existing step S_j. And add just two constraints: $S_i \prec S_j$ and $S_i \xrightarrow{c} S_j$.
- Causal links must be *protected* from *threats*, i.e. steps that delete (or negate or *clobber*) the protected condition. If S threatens link $S_i \xrightarrow{c} S_j$:
  - *Promote*: add the constraint $S \prec S_i$; **or**
  - *Demote*: add the constraint $S_j \prec S$

The goal achievement operators ought to be obvious enough. They find preconditions of steps in the unfinished plan that are not yet achieved. The two goal achievement operators remedy this either by adding a new step whose effect achieves the precondition, or by exploiting one of the effects of a step that is already in the plan.

The promotion and demotion operators may be less clear. Why are these needed? POP uses problem-decomposition: faced with a conjunctive precondition, it uses goal achievement on each conjunct separately. But, as we know, this brings the risk that the steps we add when achieving one part of a precondition might interfere with the achievement of another precondition. And the idea of promotion and demotion is to add ordering constraints so that the step cannot interfere with the achievement of the precondition.

Finally, we have to be able to recognise when we have reached a *solution plan*: a finished plan.

A solution plan is one in which:

- every precondition of every step is achieved by the effect of some other step;
- all possible clobberers have been suitably demoted or promoted; and
- there are no contradictions in the ordering constraints, e.g. disallowed is $S_i \prec S_j$ and $S_j \prec S_i$; also disallowed is $S_i \prec S_j$, $S_j \prec S_k$ and $S_k \prec S_i$.

Note that solutions may still be partially-ordered. This retains flexibility for as long as possible. Only immediately prior to execution will the plan need *linearising*, i.e. the imposition of arbitrary ordering constraints on steps that are not yet ordered. (In fact, if there's more than one agent, or if there's a single agent but it is capable of multitasking, then some linearisation can be avoided: steps can be carried out in parallel.)

2 The POP algorithm

In essence, the POP algorithm is the following:

Make the initial plan, i.e. the one that contains only the Start and Finish steps.
while the plan is not a solution plan
{ Choose one of the following:
1. Achieve an unachieved precondition by adding a new step;
2. Achieve an unachieved precondition using an existing step;
3. Protect a link by promotion;
4. Protect a link by demotion;
}

But what the above fails to show is that planning involves search. At certain points in the algorithm, the planner will be faced with choices (alternative ways of refining the current unfinished plan). POP must try one of them but have the option of returning to explore the others.

There are basically two main 'choice points' in the algorithm:

- In goal achievement, a condition c might be achievable by any one of a number of new steps and/or existing steps. For each way of achieving c, a new version of the plan must be created and placed on the agenda.

  Question. A condition c might be achievable by new steps or existing steps. When placing these alternatives on the agenda, why might we arrange for the latter to come off the agenda before the former?

- When resolving threats, POP must choose between demotion and promotion.

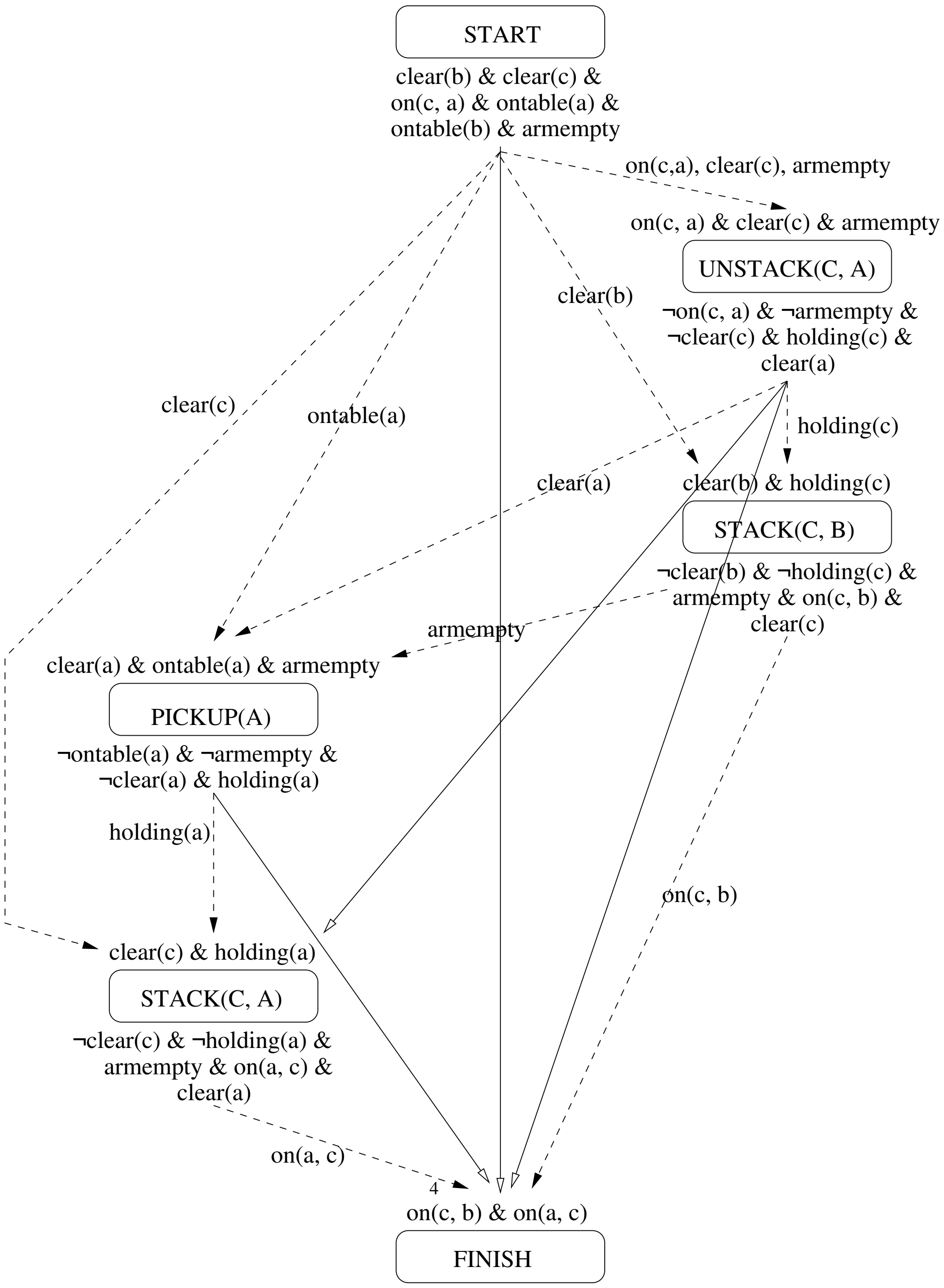
(Some people think that the choice of which precondition to achieve next also gives rise to search. But, in fact, all preconditions must eventually be achieved, and so these aren't alternatives. The choice can be made irrevocably.)

Provided your implementation of POP uses a complete and optimal search strategy, then POP itself is complete and optimal.

However, the number of choices at each point can still be high and the unfinished plans that we store on the agenda can be quite large data structures, so we typically abandon completeness/optimality to keep time and space more manageable. Search strategies that are more like depth-first search might be preferable. And we might use heuristics to order alternatives or even to prune the agenda.

In the lecture, we will dry-run the POP algorithm.

Afterwards, convince yourself that POP is a regression planner, that it uses problem decomposition, that it searches plan space, that it build partially-ordered plans and that it operates by the principle of least commitment.



Exercise (Past exam question)

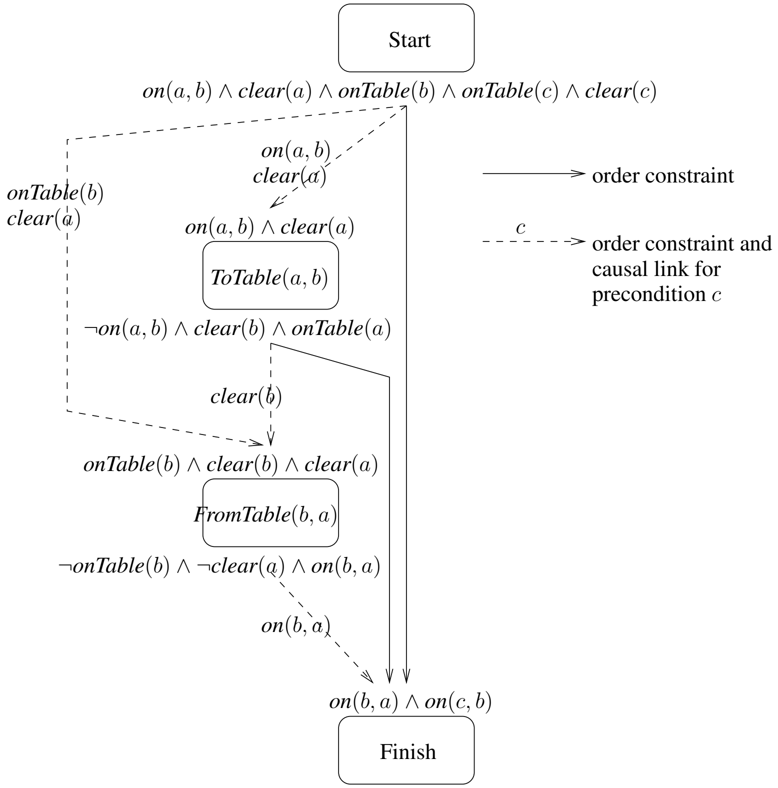
1. An A.I. planner operates in a simplified Blocks World. The only operators in its repertoire move a block x from the table to another block y:

   Op(ACTION: *FromTable*(x, y),
   PRECOND: *onTable*$(x) \wedge$ *clear*$(x) \wedge$ *clear*(y),
   EFFECT: $\neg$*onTable*$(x) \wedge \neg$*clear*$(y) \wedge$ *on*(x, y))

   and move a block x from block y to the table:

   Op(ACTION: *ToTable*(x, y),
   PRECOND: *on*$(x, y) \wedge$ *clear*(x),
   EFFECT: $\neg$*on*$(x, y) \wedge$ *clear*$(y) \wedge$ *onTable*(x))

   Here is an incomplete plan of the kind that could be built by the POP planner covered in lectures:



   (a) Give the *start* world state and *goal* of this plan.

   (b) Copy this plan onto your answer sheet. (Copy just the boxes and arrows; there is no need to copy the preconditions & effects.)
   - Choose an unachieved precondition in the plan.
   - Add a *new step* to the plan to achieve your chosen precondition. Draw it onto your copy of the diagram. Include its preconditions & effects, all order constraints and all causal links.
   - If your new step threatened any existing causal links, then state which link(s) were threatened; state what extra ordering constraint(s) you added to protect the threatened link(s); state whether what you did was an example of promotion or demotion; and briefly explain why the extra ordering constraint(s) fix the plan.

   (c) Is the plan now complete? Explain your answer.

2. Write *STRIPS operators* that would enable a planner to build plans that it could give to photocopier repair robots.

   Use the following predicate symbols:

copier(x)	:	x is a photocopier
robot(x)	:	x is a robot
noToner(x)	:	x has no toner
hasToner(x)	:	x has toner
hasPaper(x, n)	:	x has n sheets of paper
at(x, y)	:	x is at y

   You can also use the predicates $<, \leq, >, \geq$ and $=$, the function symbols $+$ and $-$ and the constant symbols 0 and 1 if you wish, all with their usual meanings from arithmetic. (Hint: many students lost marks in the exam by forgetting about the delete-list, i.e. the negated effects.)

   You should write the following three operators:

   - *replaceToner*(x, y): To replace the toner, the copier (y) must be out of toner, a robot (x) must be at the copier and it must have some toner, all of which it puts into the copier.
   - *insertPaper*(x, y, n): To put n sheets of paper into the copier (y), a robot (x) must be at the copier and it must have at least n sheets of paper. (You should assume that the copier has no maximum amount of paper.)
   - *makeCopy*(x, y): To make a copy (using up one sheet of paper), a robot (x) must be at a copier (y) that has toner and that has at least one sheet of paper.