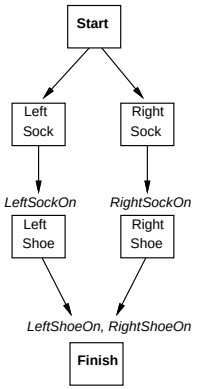
PLANNING II

Partial Order Planning

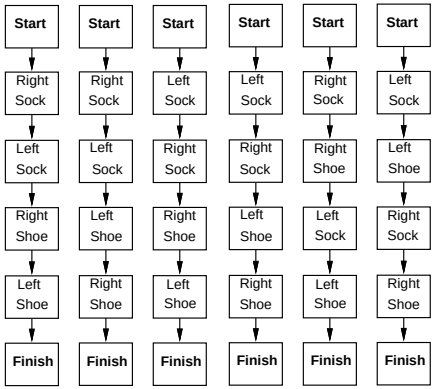
- The answer to the problem we ended the last lecture with is to use partial order planning.
- Basically this gives us a way of checking before adding an action to the plan that it doesn't mess up the rest of the plan.
- The problem is that in this recursive process, we don't know what the rest of the plan is.
- Need a new representation *partially ordered plans*.

Representation

Partial Order Plan:



Total Order Plans:

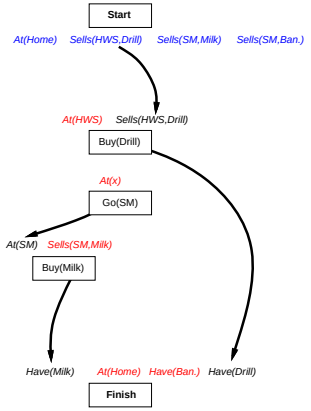


Partially ordered plans

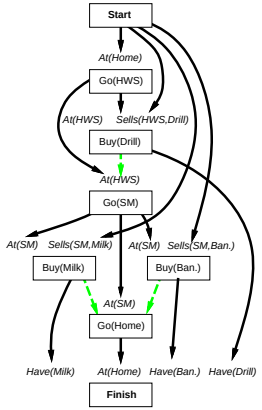
- *Partially ordered* collection of steps with
  - $Start$ step has the initial state description as its effect
  - $Finish$ step has the goal description as its precondition
  - *causal links* from outcome of one step to precondition of another
  - *temporal ordering* between pairs of steps
- *Open condition* = precondition of a step not yet causally linked
- A plan is *complete* iff every precondition is achieved
- A precondition is *achieved* iff it is the effect of an earlier step and no *possibly intervening* step undoes it

Plan construction

Plan construction (2)



Plan construction (3)



Planning process

- Operators on partial plans:
  - *add a link* from an existing action to an open condition
  - *add a step* to fulfill an open condition
  - *order* one step wrt another to remove possible conflicts
- Gradually move from incomplete/vague plans to complete, correct plans
- Backtrack if an open condition is unachievable or if a conflict is unresolvable

POP algorithm

function POP(*initial*, *goal*, *operators*) **returns** *plan*

  plan ← MAKE-MINIMAL-PLAN(*initial*, *goal*)
  loop do
    if SOLUTION?(*plan*) **then return** *plan*
    $S_{need},\ c$ ← SELECT-SUBGOAL(*plan*)
    CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, c)
    RESOLVE-THREATS(*plan*)
  end

---

function SELECT-SUBGOAL(*plan*) **returns** $S_{need},\ c$

  pick a plan step S_{need} from STEPS(*plan*)
    with a precondition c that has not been achieved
  return $S_{need},\ c$

procedure CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, c)

  choose a step S_{add} from *operators* or STEPS(*plan*) that has c as an effect
  if there is no such step **then fail**
  add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
  add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
  if S_{add} is a newly added step from *operators* **then**
    add S_{add} to STEPS(*plan*)
    add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

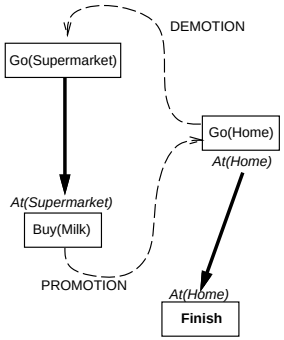
---

procedure RESOLVE-THREATS(*plan*)

  for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
    choose either
      Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
      Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
    if not CONSISTENT(*plan*) **then fail**
  end

Clobbering

- A *clobberer* is a potentially intervening step that destroys the condition achieved by a causal link. E.g., $Go(Home)$ clobbers $At(Supermarket)$:



Demotion: put before $Go(Supermarket)$

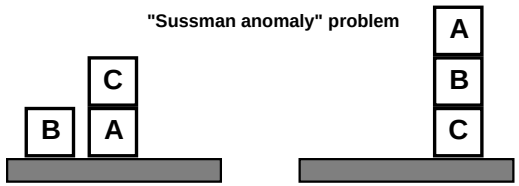
Promotion: put after $Buy(Milk)$

Properties of POP

- Nondeterministic algorithm: backtracks at *choice* points on failure:
  - choice of S_{add} to achieve S_{need}
  - choice of demotion or promotion for clobberer
  - selection of S_{need} is irrevocable
- POP is sound, complete, and *systematic* (no repetition)
- Extensions for disjunction, universals, negation, conditionals
- Can be made efficient with good heuristics derived from problem description
- Particularly good for problems with many loosely related subgoals

Example

"Sussman anomaly" problem



Start State | Goal State

Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y) Clear(z) On(x,y)

Clear(x) On(x,z)

PutOnTable(x)

~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

Example (2)

START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

On(A,B) On(B,C)

FINISH

Example (3)

START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

Cl(B) On(B,z) Cl(C)

PutOn(B,C)

On(A,B) On(B,C)

FINISH

Example (4)

START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

Cl(A) On(A,z) Cl(B)

PutOn(A,B)

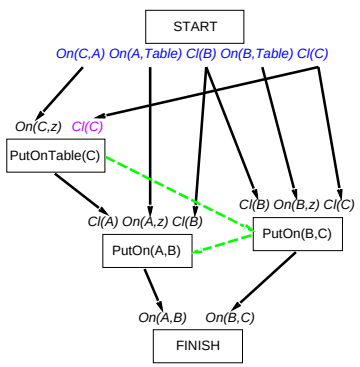
Cl(B) On(B,z) Cl(C)

PutOn(B,C)

On(A,B) On(B,C)

FINISH

Example (5)



PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

PutOn(B,C) clobbers Cl(C) => order after PutOnTable(C)

Summary

- This lecture has looked at a more advanced approach to planning.
  - Partial order planning
- This requires a new way of looking at the world, but the payoff is a more robust approach.
- We also looked at the POP algorithm, . . .
- . . . and saw how it could solve the Sussman anomaly.