Planning and Search

Classical Planning: Partial order planning

Outline

- ♢ Totally vs partially ordered plans
- ♢ Partial-order planning
- ♢ Examples

Totally vs partially ordered plans

So far we produced a linear sequence of actions (totally ordered plan)

Often it does not matter in which order *some of the actions* are executed

For problems with independent subproblems often easier to find a **partially ordered plan**: a plan which is a set of actions and a set of constraints $Before(a_i, a_j)$

Partially ordered plans are created by a search through a space of plans (rather than the state space)

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

Example

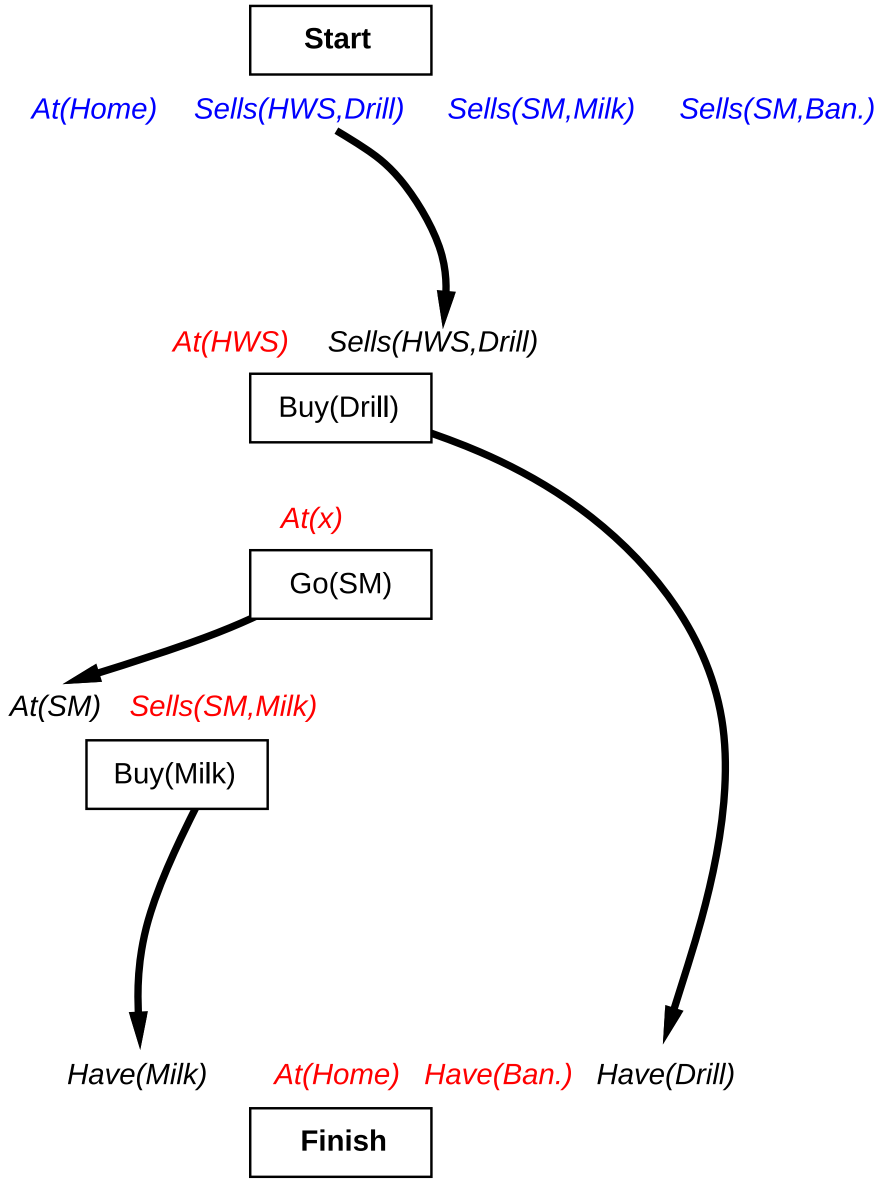
Start

At(Home) *Sells(HWS,Drill)* *Sells(SM,Milk)* *Sells(SM,Ban.)*

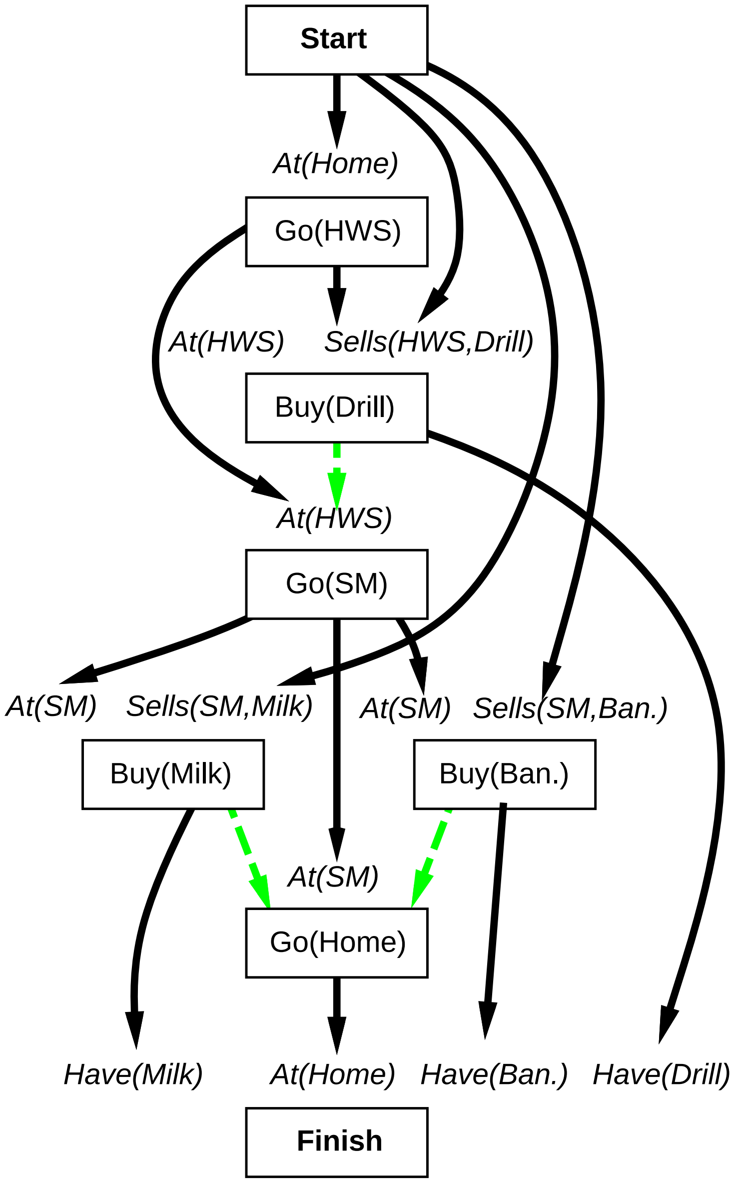
Have(Milk) *At(Home)* *Have(Ban.)* *Have(Drill)*

Finish

Example


Start
At(Home)
Sells(HWS,Drill)
Sells(SM,Milk)
Sells(SM,Ban.)
At(HWS)
Sells(HWS,Drill)
Buy(Drill)
At(x)
Go(SM)
At(SM)
Sells(SM,Milk)
Buy(Milk)
Have(Milk)
At(Home)
Have(Ban.)
Have(Drill)
Finish

Example


Start
At(Home)
Go(HWS)
At(HWS)
Sells(HWS,Drill)
Buy(Drill)
At(HWS)
Go(SM)
At(SM)
Sells(SM,Milk)
At(SM)
Sells(SM,Ban.)
Buy(Milk)
Buy(Ban.)
At(SM)
Go(Home)
Have(Milk)
At(Home)
Have(Ban.)
Have(Drill)
Finish

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 sketch

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

plan ← Make-Minimal-Plan(*initial, goal*)
loop do
  if Solution?(*plan*) **then return** *plan*
  S_{need}, c ← Select-Subgoal(*plan*)
  Choose-Action(*plan, actions*, 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

POP algorithm contd.

procedure CHOOSE-ACTION(*plan*, *actions*, S_{need}, c)

choose a step S_{add} from *actions* 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 *actions* **then**
    add S_{add} to STEPS(*plan*)
    add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

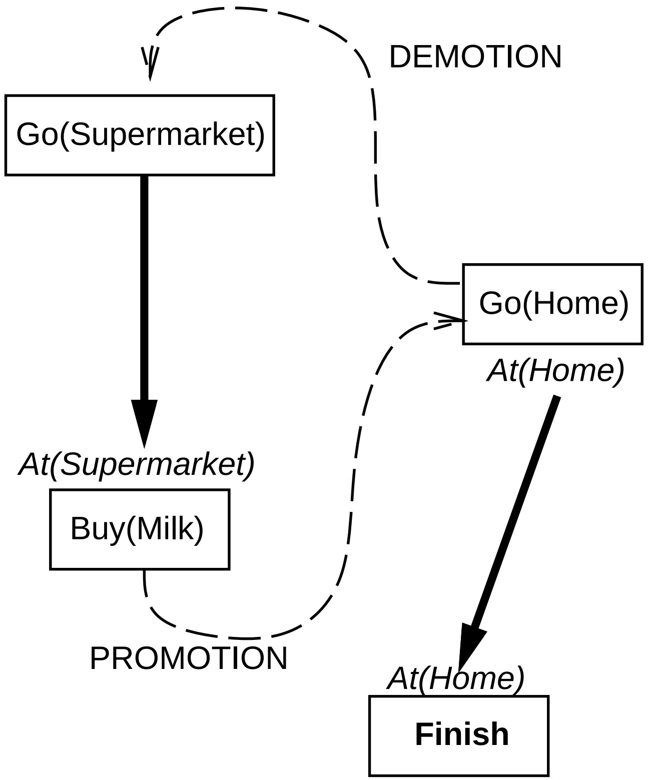
---

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 and promotion/demotion

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

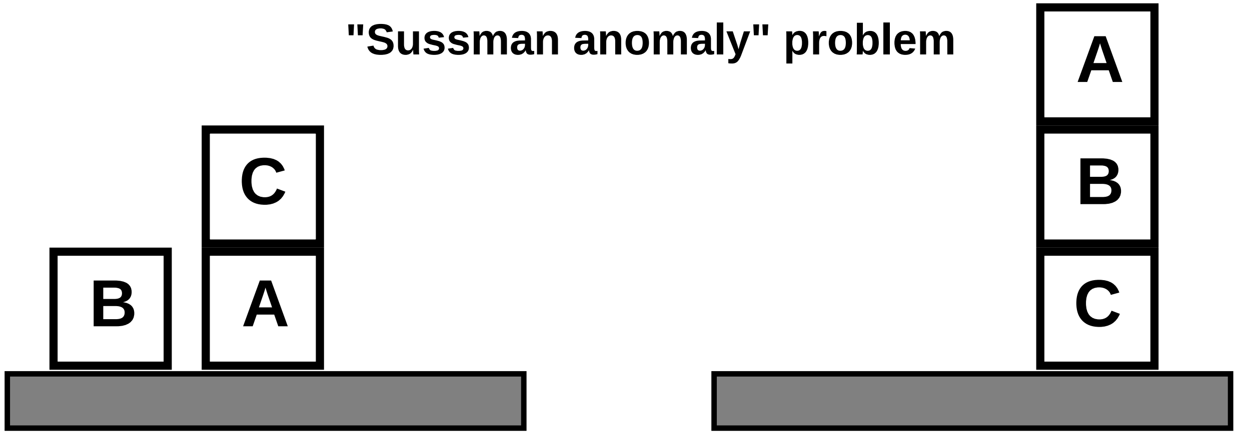
POP is sound, complete, and systematic (no repetition)

Can be made efficient with good heuristics derived from problem description

Particularly good for problems with many loosely related subgoals

Example: Blocks world

"Sussman anomaly" problem



Start State　　　　Goal State

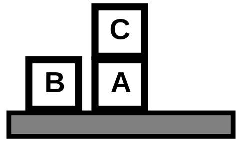
PutOn(x,y)	PutOnTable(x)
Preconditions: *Clear(x) On(x,z) Clear(y)*	Preconditions: *Clear(x) On(x,z)*
Effects: *~On(x,z) ~Clear(y) Clear(z) On(x,y)*	Effects: *~On(x,z) Clear(z) On(x,Table)*

+ several inequality constraints

Example contd.

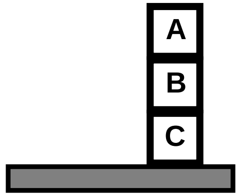
START

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

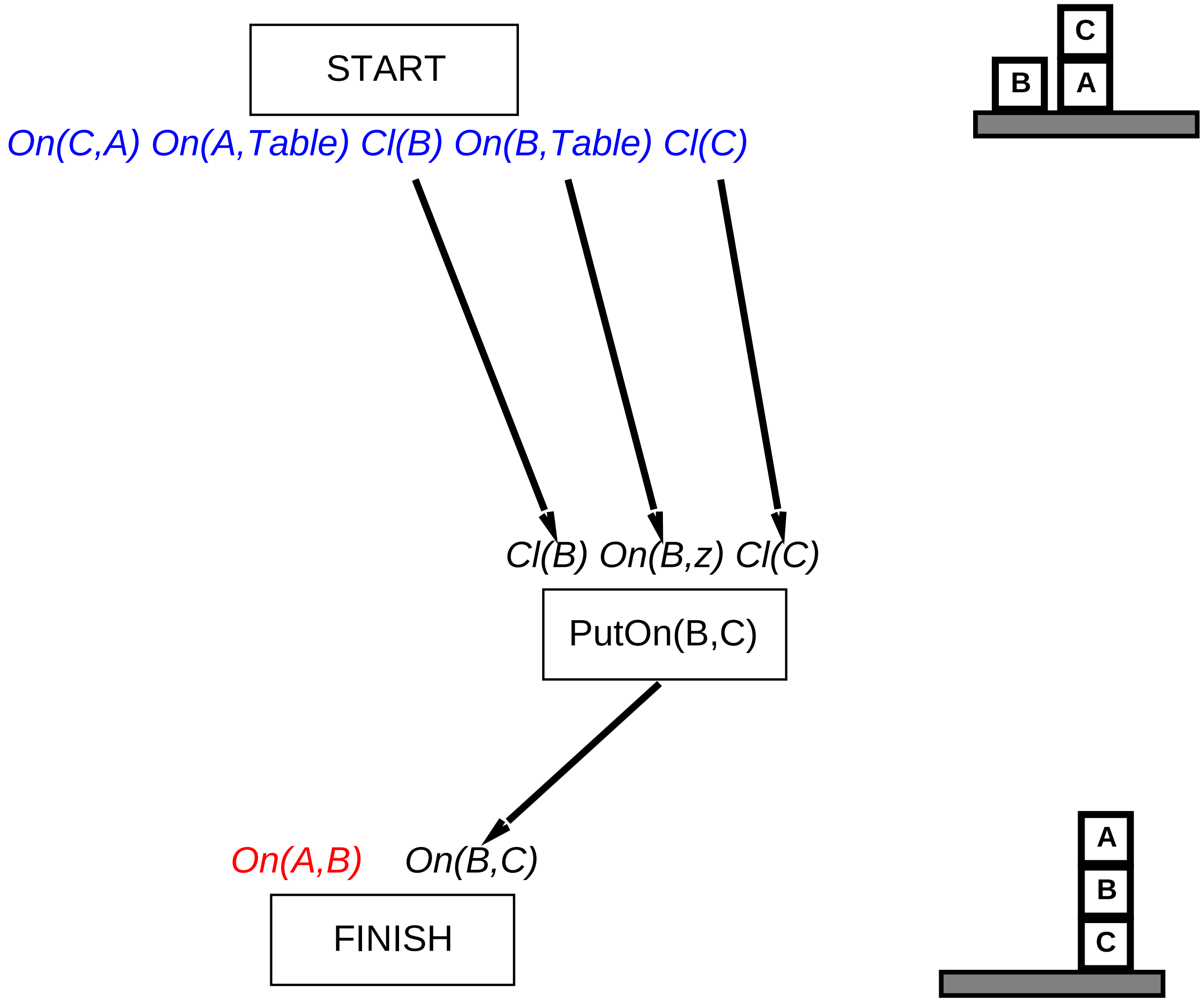


On(A,B) *On(B,C)*

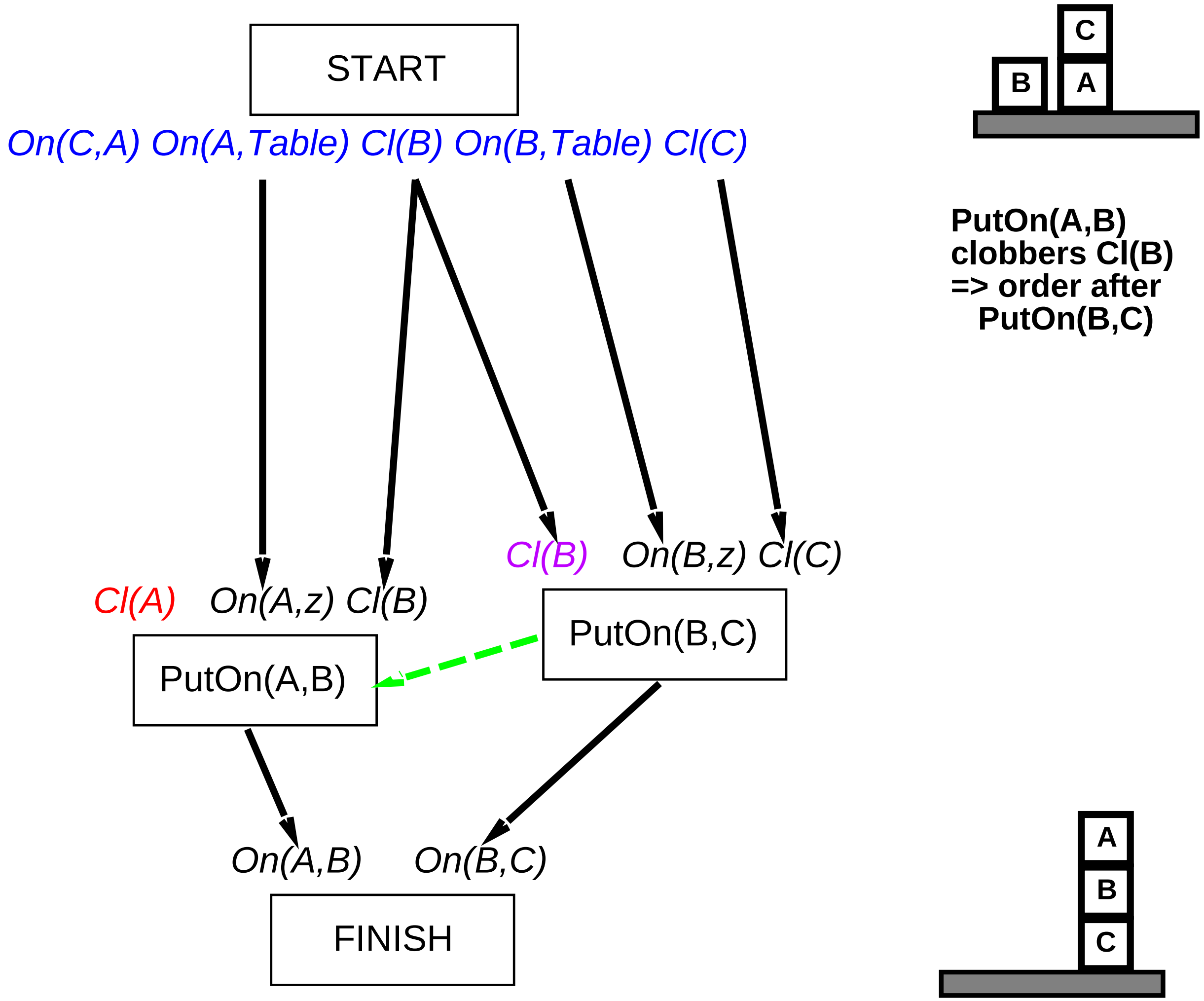
FINISH



Example contd.

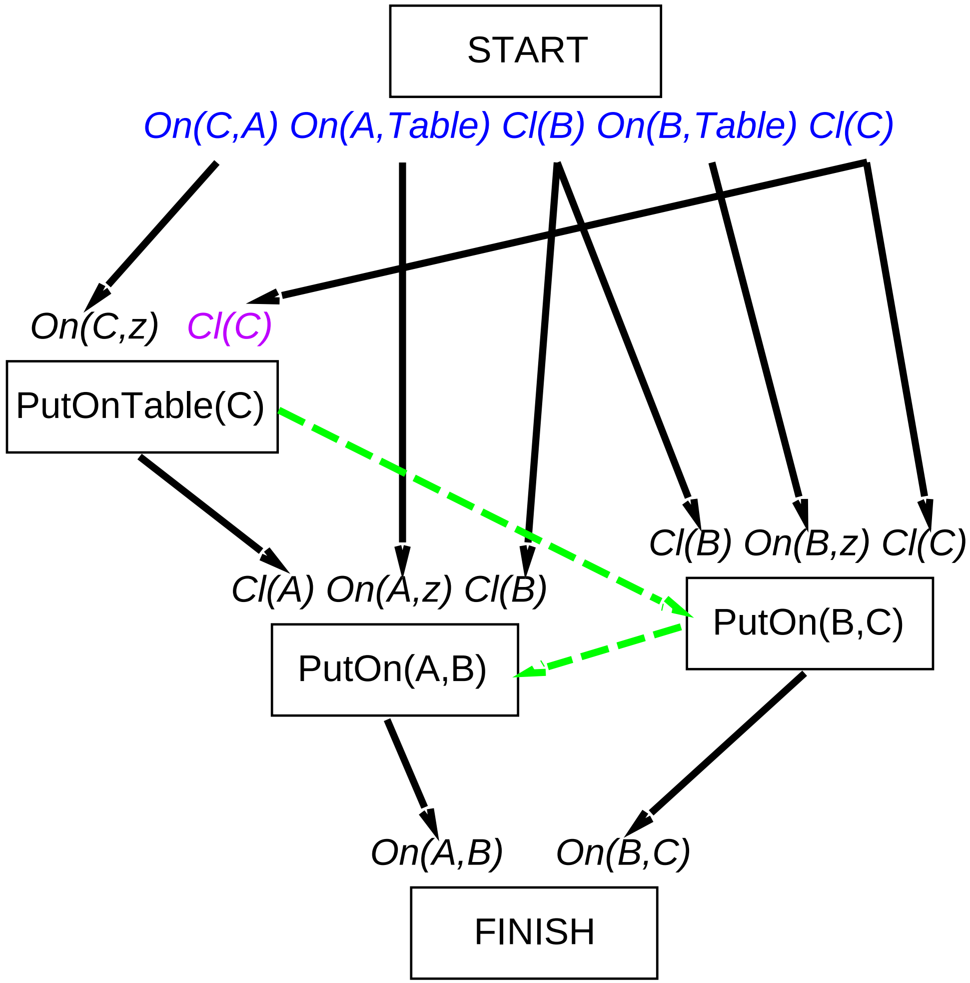


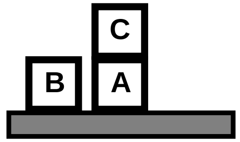
Example contd.



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

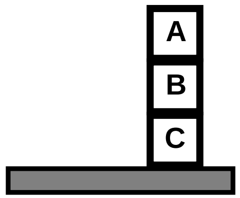
Example contd.





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

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



Exercise

Assume planning domain definition from lecture 10:

$Move(b, x, y)$:
  PRECOND: $On(b, x) \wedge Clear(b) \wedge Clear(y)$
  EFFECT: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y)$

$Movetotable(b, x)$:
  PRECOND: $On(b, x) \wedge Clear(b)$
  EFFECT: $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x)$

and the following planning problem: the initial state is $On(B, A) \wedge On(D, C) \wedge Clear(B) \wedge Clear(D) \wedge On(A, Table) \wedge On(C, Table)$. The goal is $On(A, B) \wedge On(C, D) \wedge On(B, Table) \wedge On(C, Table)$. Solve this problem using partial order planning; trace the search from the initial empty plan to a complete solution, explaining every step.

Next lecture

GraphPlan, SATplan.

Comparison of classic planning algorithms.

Chapter 10 in 3rd edition, 11 in 2nd edition.

(As far as I can tell, there is not a lot of difference between the 2nd and 3rd edition on planning; in fact POP is covered in more detail in the 2nd edition).