Knowledge representation and reasoning
Lecture 20: Planning 3

Natasha Alechina
natasha.alechina@nottingham.ac.uk

Plan of the lecture

- SET and SEM
- Partial order planning (based on Russell and Norvig, Russell's slides)
- Overview of planning algorithms and systems

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

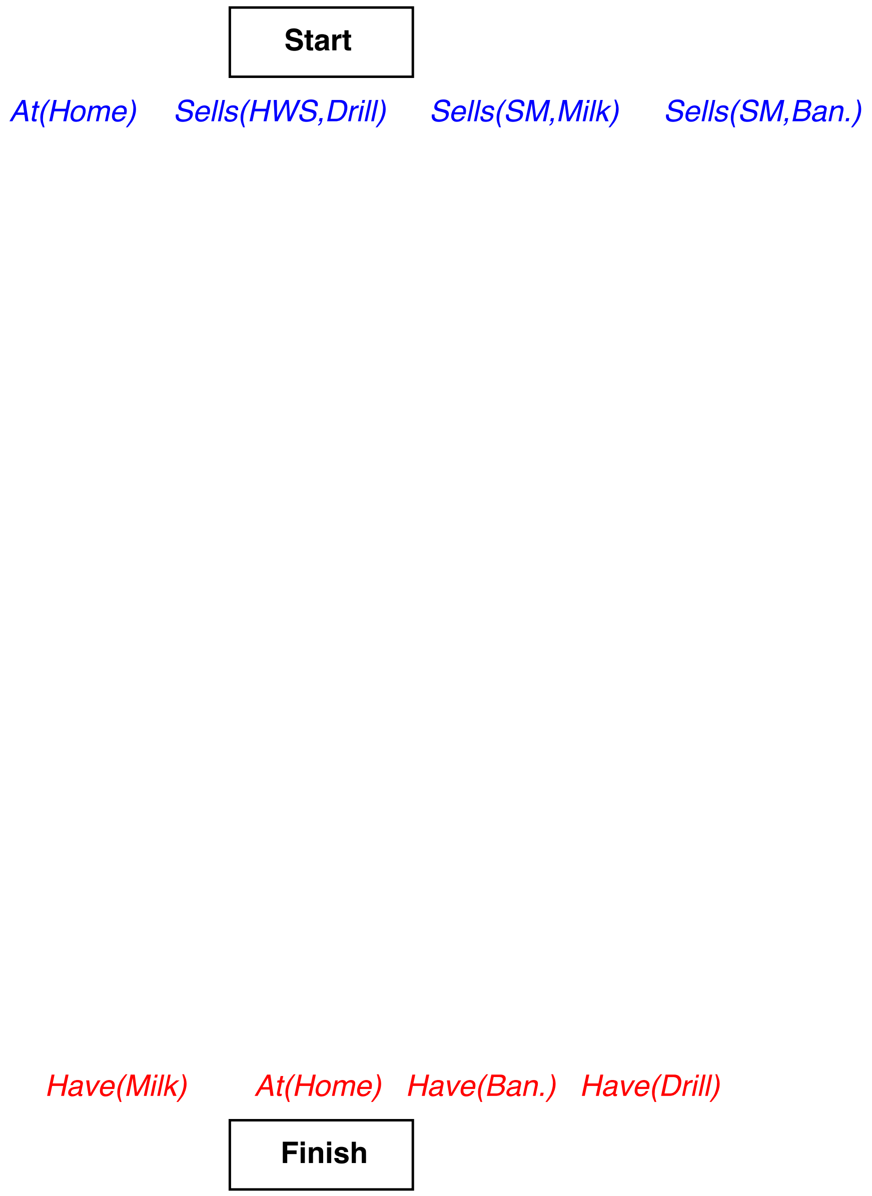
- $Start$ step has the initial state description as its effect
- $Finish$ step has the goal description as its precondition
- causal links from outcome of one step to precondition of another
- temporal ordering between pairs of steps

Open condition = precondition of a step not yet causally linked

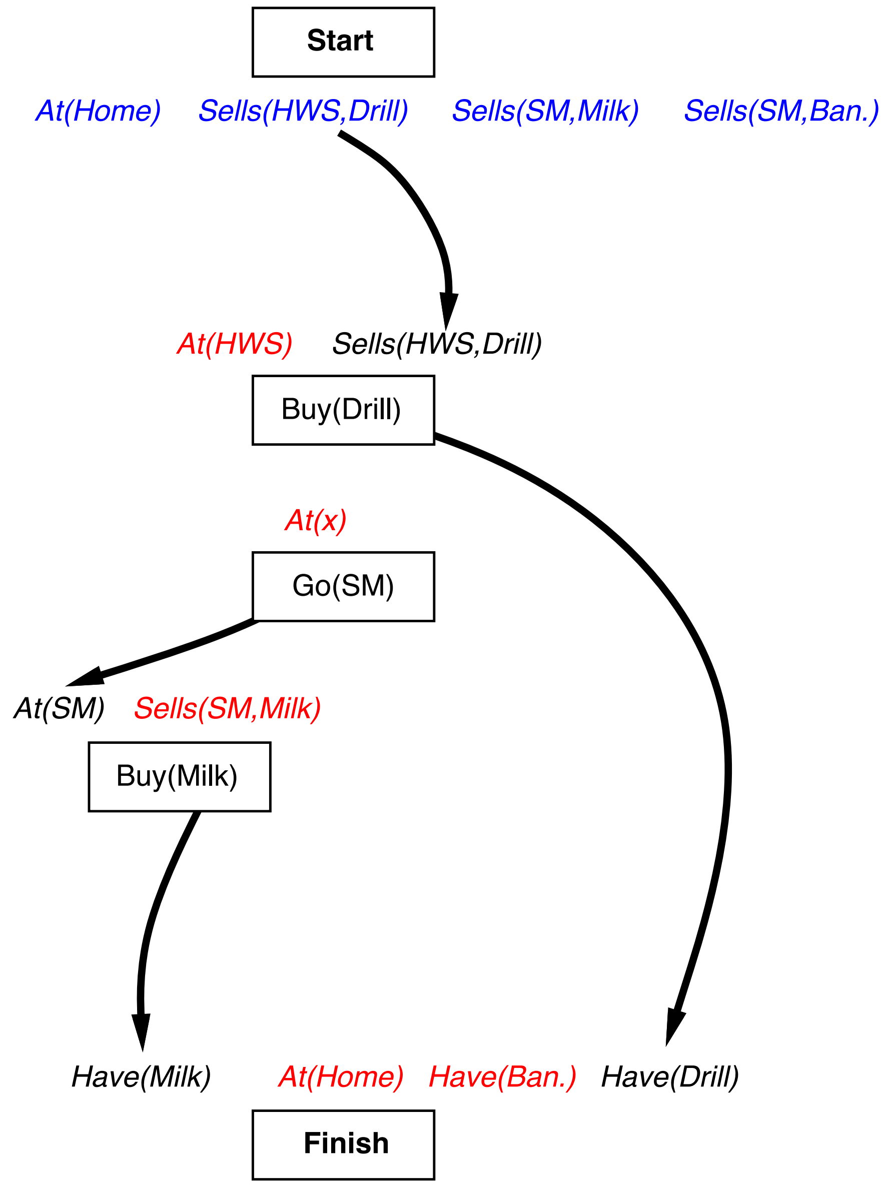
A plan is complete iff every precondition is achieved

A precondition is achieved iff it is the effect of an earlier step
and no possibly intervening step undoes it

Example

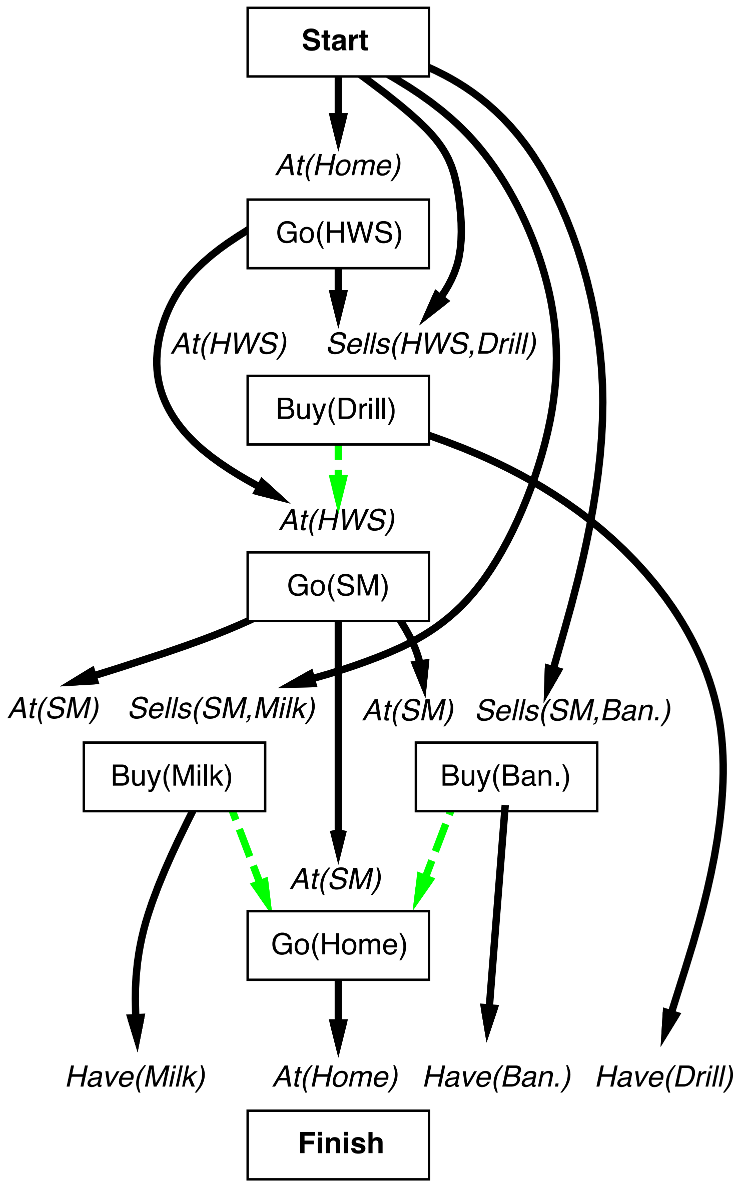

Start
At(Home)
Sells(HWS,Drill)
Sells(SM,Milk)
Sells(SM,Ban.)
Have(Milk)
At(Home)
Have(Ban.)
Have(Drill)
Finish

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 $S_{need}$, $c$
```

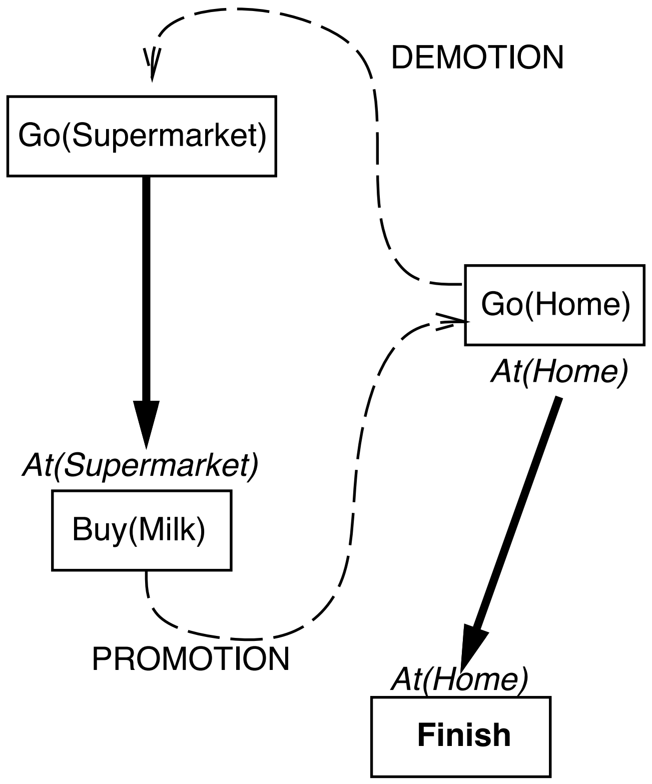
POP algorithm contd.

```
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

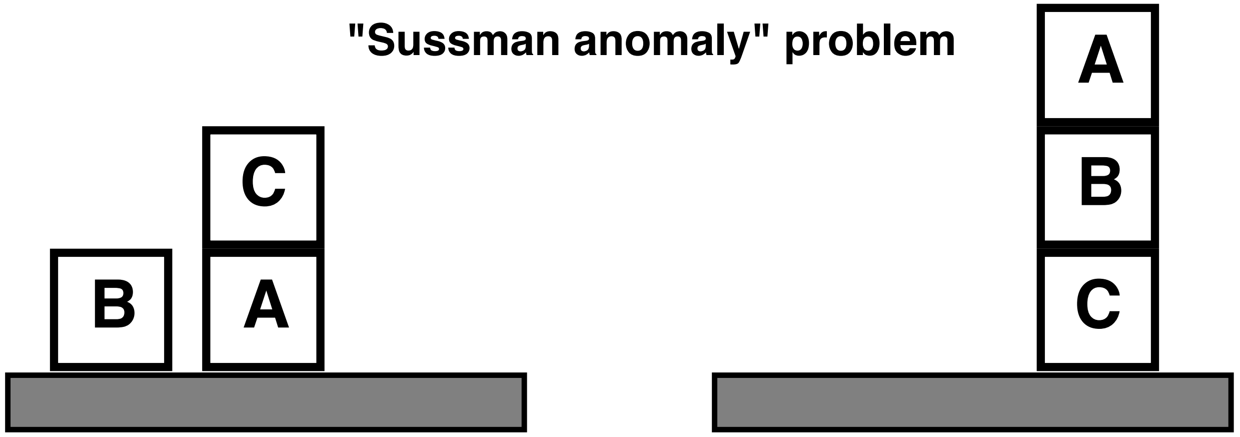
POP is sound, complete, and systematic (no repetition)

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

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)

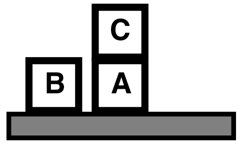
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

Example contd.

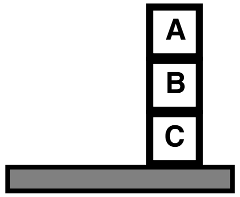
START

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

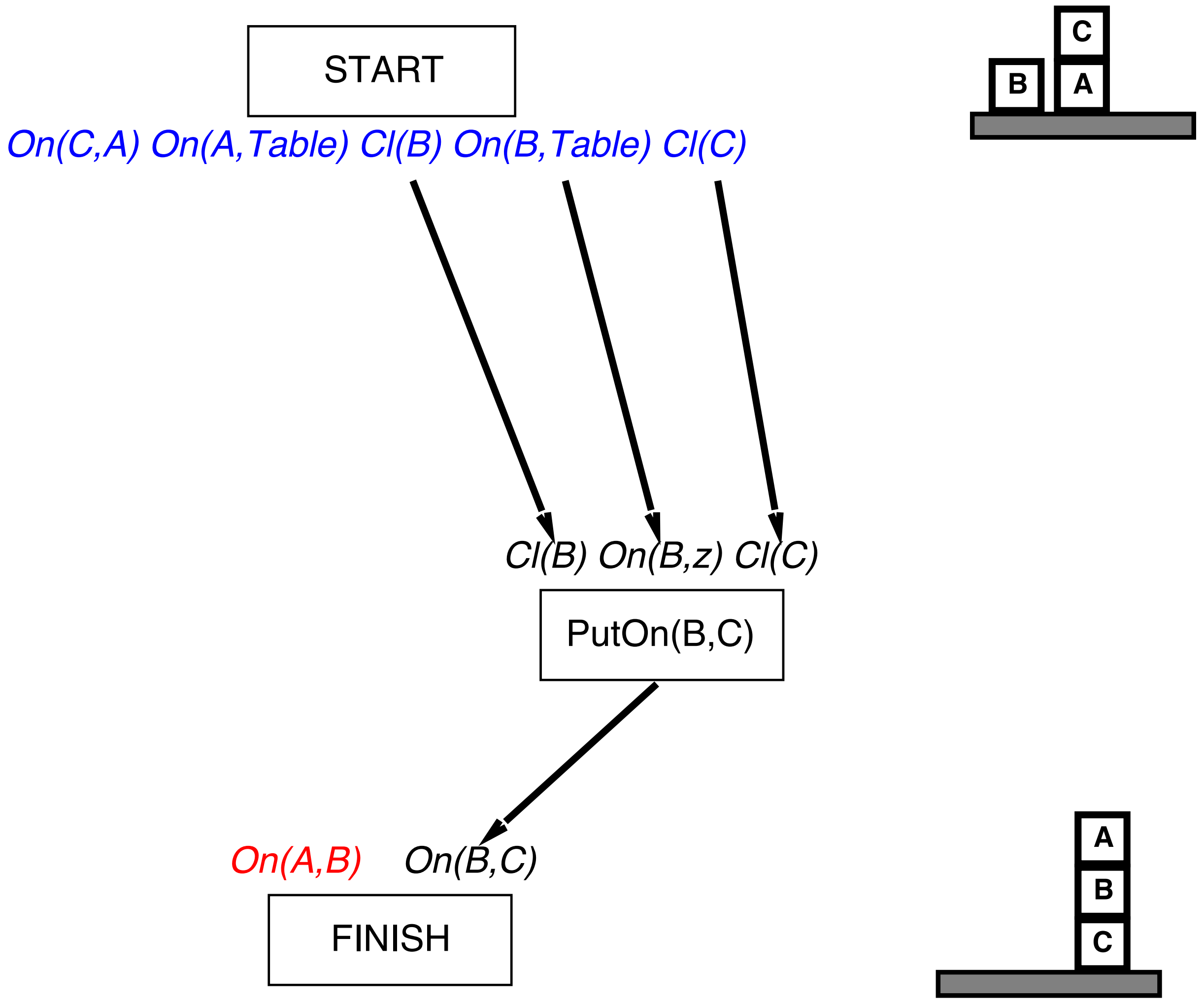


On(A,B) On(B,C)

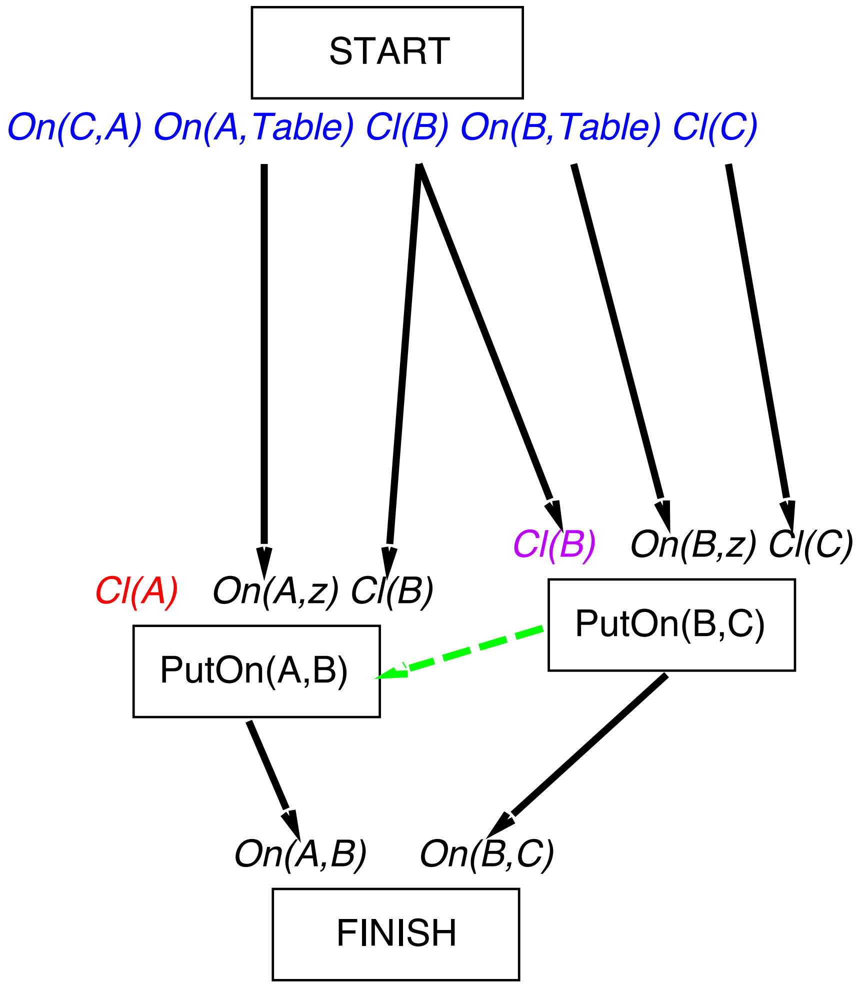
FINISH

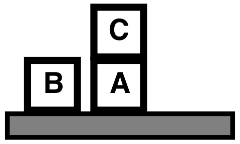


Example contd.

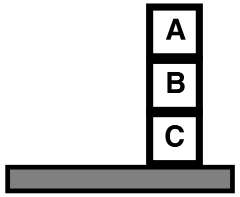


Example contd.

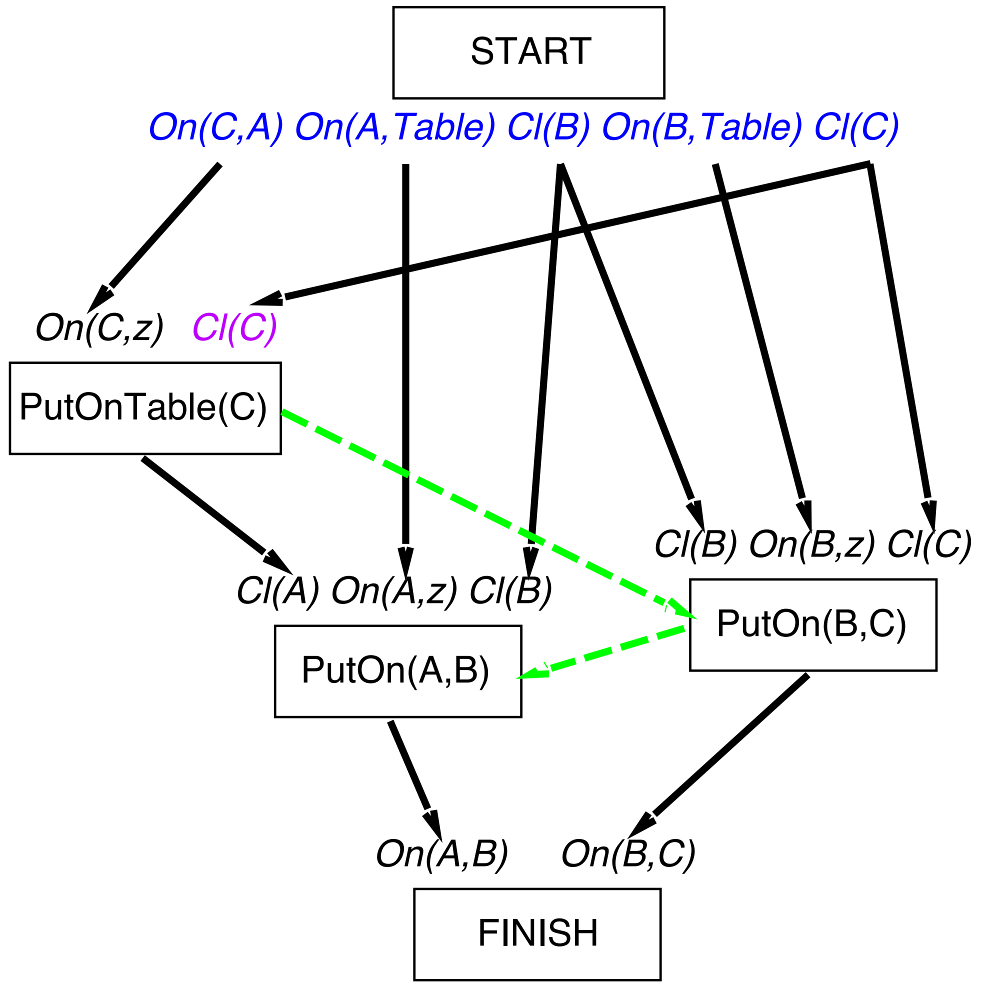


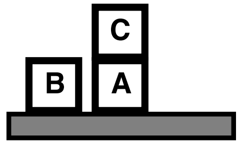


PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)



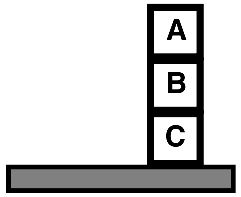
Example contd.





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

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



Other planning approaches

- SatPlan: reduction to propositional reasoning, and checking for satisfiability.
- Can check whether a plan of fixed length exists
- Hierarchical planning (planning with abstraction): first find a plan with high-level abstract actions, then refine it
- Planning with time and resources, and scheduling plans

The complexity of classical planning

PlanSAT is the question whether there exists any plan that solves a given planning problem

Bounded PlanSAT is the question whether there exists a plan of length k or less

PlanSAT is about **satisficing** (want any solution, not necessarily the cheapest or the shortest)

Bounded PlanSAT can be used to ask for the **optimal** solution

If in the PDDL language we do not allow functional symbols, both problems are decidable

Complexity of both problems is PSPACE (can be solved by a Turing machine which uses polynomial amount of space)

NP $\subseteq$ PSPACE (PSPACE is even harder than NP)

Some of the top-performing systems

International planning competition winners (from Russell and Norvig, 3rd edition):

Year	Track	Winning systems (approaches)
2008	Optimal	Gamer (symbolic bi-directional search)
2008	Satisficing	LAMA (fast forward search with FF heuristic)
2006	Optimal	SATPlan, MAXPlan (boolean satisfiability)
2006	Satisficing	SGPlan (forward search, partition into independent subproblems
2004	Optimal	SATPlan (boolean satisfiability)
2004	Satisficing	Fast Diagonally Forward (forward search with causal graph)
2002	Automated	LPG (local search, constraint satisfaction)
2002	Hand-coded	TLPLan (temporal action logic with control rules for forward searc
2000	Automated	FF (forward search)
2000	Hand-coded	TalPlanner (temporal action logic with control rules for forward sea
1998	Automated	IPP (planning graphs); HSP (forward search)

What follows for the algorithms

If a planning system based on a particular planning algorithm is very fast it does not mean necessarily that the algorithm is ‘better’

Sat solvers are very fast, but one may argue that it is not very practical to propositionalise planning problems

Forward planning with good heuristics can be very fast but again may be not always practically possible

Partial order planners are considered to be very flexible and generally useful, although they don’t feature in the winners table...

Next lecture

- Reasoning with uncertainty
- Brachman and Levesque, chapter Chapter 12, slides 202-216.
- Revision advice
- I can also answer revision questions during the lab on Monday
- No lecture next Thursday (13th of December)