Informatics 2D: Reasoning and Agents

Alex Lascarides

School of Informatics

The University of Edinburgh

Lecture 17b: Partial Order Planning

Where are we?

Last time. . .

- Planning using state-space search (forward/backward)

Now:

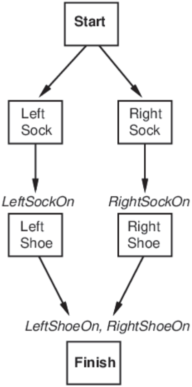
- **Partial-order Planning**

Partial-order planning

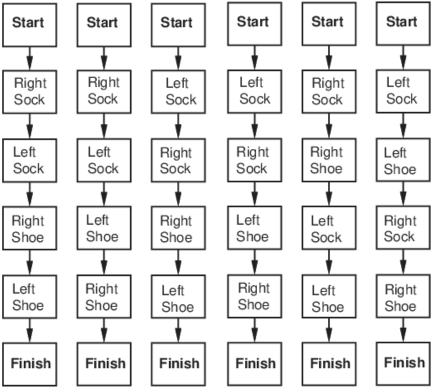
- State-space search planning algorithms consider **totally ordered** sequences of actions
- Better not to commit ourselves to complete chronological ordering of tasks (**least commitment** strategy)
- **Basic idea:**
  1. Add actions to a plan without specifying which comes first unless necessary
  2. Combine 'independent' subsequences afterwards
- Partial-order solution will correspond to one or several **linearisations** of partial-order plan
- Search in **plan space** rather than state spaces (because your search is over ordering constraints on actions, as well as transitions among states).

Example: Put your socks and shoes on

Partial-Order Plan:



Total-Order Plans:



Partial-order planning (POP) as a search problem

Define POP as search problem over plans consisting of:

- **Actions**; initial plan contains dummy actions *Start* (no preconditions, effect=initial state) and *Finish* (no effects, precondition=goal literals)
- **Ordering constraints** on actions $A \prec B$ (A must occur before B); contradictory constraints prohibited
- **Causal links** between actions $A \xrightarrow{p} B$ express A achieves p for B (p precondition of B, effect of A, must remain true between A and B); inserting action C with effect $\neg p$ ($A \prec C$ and $C \prec B$) would lead to **conflict**
- **Open preconditions:** set of conditions not yet achieved by the plan (planners try to make open precondition set empty without introducing contradictions)

The POP algorithm

- Final plan for socks and shoes example (without trivial ordering constraints):
  Actions: $\{RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish\}$
  Orderings: $\{RightSock \prec RightShoe, LeftSock \prec LeftShoe\}$
  Links: $\{RightSock \stackrel{RightSockOn}{\rightarrow} RightShoe,$
  $LeftSock \stackrel{LeftSockOn}{\rightarrow} LeftShoe,$
  $RightShoe \stackrel{RightShoeOn}{\rightarrow} Finish,$
  $LeftShoe \stackrel{LeftShoeOn}{\rightarrow} Finish\}$
  Open preconditions: $\{\}$
- **Consistent plan** = plan without cycles in orderings and conflicts with links
- **Solution** = consistent plan without open preconditions
- Every linearisation of a partial-order solution is a total-order solution (implications for execution!)

The POP algorithm

- Initial plan:
  Actions: $\{Start, Finish\}$, Orderings: $\{Start \prec Finish\}$,
  Links: $\{\}$, Open preconditions: Preconditions of *Finish*
- Pick p from open preconditions on some action B, generate a consistent successor plan for every A that achieves p
- Ensuring consistency:
  1. Add $A \xrightarrow{p} B$ and $A \prec B$ to plan. If A new, add A and $Start \prec A$ and $A \prec Finish$ to plan
  2. Resolve conflicts between the new link and all actions and between A (if new) and all links as follows:
     If conflict between $A \xrightarrow{p} B$ and C, add $B \prec C$ or $C \prec A$
- Goal test: check whether there are open preconditions (only consistent plans are generated)

Partial-order planning example (1)

$Init(At(Flat, Axle) \land At(Spare, Trunk)).\quad Goal(At(Spare, Axle)).$

$Action(Remove(Spare, Trunk),$

Precond: $At(Spare, Trunk)$

Effect: $\neg At(Spare, Trunk) \land At(Spare, Ground))$

$Action(Remove(Flat, Axle),$

Precond: $At(Flat, Axle)$

Effect: $\neg At(Flat, Axle) \land At(Flat, Ground))$

$Action(PutOn(Spare, Axle),$

Precond: $At(Spare, Ground) \land \neg At(Flat, Axle)$

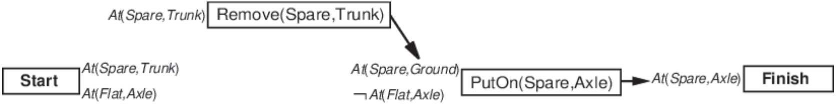
Effect: $\neg At(Spare, Ground) \land At(Spare, Axle))$

$Action(LeaveOvernight,$ Precond:

Effect: $\neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk)$
$\land \neg At(Flat, Ground) \land \neg At(Flat, Axle))$

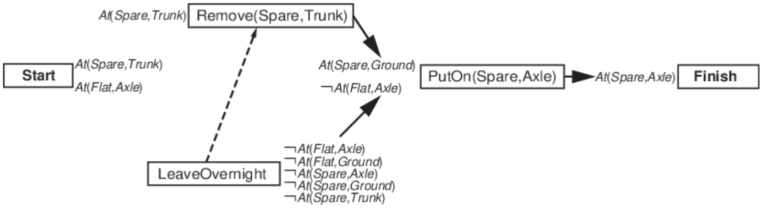
Partial-order planning example (2)

- Pick (only) open precondition $At(Spare, Axle)$ of $Finish$
  Only applicable action $= PutOn(Spare, Axle)$
- Pick $At(Spare, Ground)$ from $PutOn(Spare, Axle)$
  Only applicable action $= Remove(Spare, Trunk)$
- Situation after two steps:



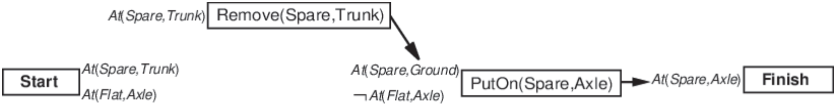
Partial-order planning example (3)

- Pick $\neg At(Flat, Axle)$ precondition of $PutOn(Spare, Axle)$
  Choose $LeaveOvernight$, effect $\neg At(Spare, Ground)$
- Conflict with link
  $Remove(Spare, Trunk) \overset{At(Spare,Ground)}{\rightarrow} PutOn(Spare, Axle)$
- Resolve by adding $LeaveOvernight \prec Remove(Spare, Trunk)$
  Why is this the only solution?



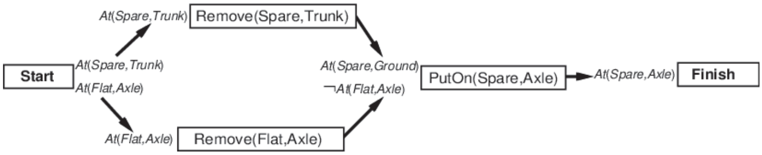
Partial-order planning example (4)

- Remaining open precondition $At(Spare, Trunk)$, but conflict between *Start* and $\neg At(Spare, Trunk)$ effect of *LeaveOvernight*
- No ordering before *Start* possible or after $Remove(Spare, Trunk)$ possible
- No successor state, backtrack to previous state and remove *LeaveOvernight*, resulting in this situation:



Partial-order planning example (5)

- Now choose *Remove*(*Flat*, *Axle*) instead of *LeaveOvernight*
- Next, choose *At*(*Spark*, *Trunk*) precondition of *Remove*(*Spare*, *Trunk*)
  Choose *Start* to achieve this
- Pick *At*(*Flat*, *Axle*) precondition of *Remove*(*Flat*, *Axle*), choose *Start* to achieve it
- Final, complete, consistent plan:



Dealing with unbound variables

- In first-order case, unbound variables may occur during planning process
- Example:

$$\begin{aligned} & Action(Move(b,x,y), \\ & \quad \text{Precond:} On(b,x) \wedge Clear(b) \wedge Clear(y) \\ & \quad \text{Effect:} On(b,y) \wedge Clear(x) \wedge \neg On(b,x) \wedge \neg Clear(y)) \end{aligned}$$

achieves $On(A,B)$ under substitution $\{b/A, y/B\}$

- Applying this substitution yields

$$\begin{aligned} & Action(Move(A,x,B), \\ & \quad \text{Precond:} On(A,x) \wedge Clear(A) \wedge Clear(B) \\ & \quad \text{Effect:} On(A,B) \wedge Clear(x) \wedge \neg On(A,x) \wedge \neg Clear(B)) \end{aligned}$$

and x is still unbound (another side of the least commitment approach)

Dealing with unbound variables

- Also has an effect on links, e.g. in example above $Move(A, x, B) \stackrel{On(A,B)}{\rightarrow} Finish$ would be added
- If another action has effect $\neg On(A, z)$ then this is only a conflict if $z = B$
- Solution: insert **inequality constraints** (in example: $z \neq B$) and check these constraints whenever applying substitutions
- Remark on heuristics: Even harder than in total-order planning, e.g. adapt most-constrained-variable approach from CSPs

Summary

- Partial-order planning
- The POP algorithms
- POP as search in planning space
- POP example
- POP with unbound variables
- Next time: **Planning and Acting in the Real World**