Where are we?

Last time . . .

- we defined the planning problem
- discussed problem with using search and logic in planning
- introduced representation languages for planning
- looked at blocks world example

Today . . .

- State-space search and partial-order planning
Most straightforward way to think of planning process: search the space of states using action schemata.

Since actions are defined both in terms of preconditions and effects we can search in both directions.

Two methods:

1. **forward state-space search**: Start in initial state; consider action sequences until goal state is reached.
2. **backward state-space search**: Start from goal state; consider action sequences until initial state is reached.
Planning with state-space search

(a)

At(P₁, A)
At(P₂, A)

Fly(P₁, A, B)

At(P₁, B)
At(P₂, A)

(b)

At(P₁, A)
At(P₂, B)

Fly(P₁, A, B)

At(P₁, B)
At(P₂, B)

At(P₂, A)

Fly(P₂, A, B)
Forward state-space search

Also called progression planning

Formulation of planning problem:

- Initial state of search is initial state of planning problem (= set of positive literals)
- Applicable actions are those whose preconditions are satisfied
- Single successor function works for deterministic planning problems (consequence of action representation)
- Goal test = checking whether state satisfies goal of planning problem
- Step cost usually 1, but different costs can be allowed
Forward state-space search

- Search space is finite in the absence of function symbols
- Any complete graph search algorithm (like A* ) will be a complete graph planning algorithm
- Forward search does not solve problem of irrelevant actions (all actions considered from each state)
- Efficiency depends largely on quality of heuristics

Example:
- Air cargo problem, 10 airports with 5 planes each, 20 pieces of cargo
- Task: move all 20 pieces of cargo at airport A to airport B
- Each of 50 planes can fly to 9 airports, each of 200 packages can be unloaded or loaded (individually)
- Assuming an average of about 1000 actions at every airport, we have $1000^{81}$ nodes in the search tree (although solution trivial)!
In normal search, backward approach hard because goal described by a set of constraints (rather than being listed explicitly)

Problem of how to generate predecessors, but planning representations allow us to consider only relevant actions

Exclusion of irrelevant actions decreases branching factor

In example, only about 20 actions working backward from goal

Regression planning = computing the states from which applying a given action leads to the goal

Must ensure that actions are consistent, i.e. they don’t undo any desired literals
Air cargo domain example

- Goal can be described as

\[ \text{At}(C_1, B) \land \text{At}(C_2, B) \land \cdots \land \text{At}(C_{20}, B) \]

- To achieve \( \text{At}(C_1, B) \) there is only one action, \( \text{Unload}(C_1, p, B) \) (\( p \) unspecified)

- Can do this action only if its preconditions are satisfied.

- So the predecessor to the goal state must include \( \text{In}(C_1, p) \land \text{At}(p, B) \), and should not include \( \text{At}(C_1, B) \) (otherwise irrelevant action)

- Full predecessor: \( \text{In}(C_1, p) \land \text{At}(p, B) \land \cdots \land \text{At}(C_{20}, B) \)

- \( \text{Load}(C_1, p) \) would be inconsistent (negates \( \text{At}(C_1, B) \))
General process of constructing predecessors for backward search given goal description $G$, relevant and consistent action $A$:

- Any positive effects of $A$ that appear in $G$ are deleted
- Each precondition of $A$ is added unless it already appears

Any standard search algorithm can be used, terminates when predecessor description is satisfied by initial (planing) state

First-order case may require additional substitutions which must be applied to actions leading from state to goal
Heuristics for state-space search

- Two possibilities:
  1. Divide and Conquer (subgoal decomposition)
  2. Derive a Relaxed Problem

- Subgoal decomposition is . . .
  - optimistic (admissible) if negative interactions exist (e.g. subplan deletes goal achieved by other subplan)
  - pessimistic (inadmissible) if positive interactions exist (e.g. subplans contain redundant actions)

- Relaxations:
  - drop all preconditions (all actions always applicable, combined with subgoal independence makes prediction even easier)
  - remove all negative effects (and count minimum number of actions so that union satisfies goals)
  - empty delete lists approach (involves running a simple planning problem to compute heuristic value)
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:**
  - Add actions to a plan without specifying which comes first unless necessary.
  - 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:

Start

Left Sock → LeftSockOn → Left Shoe → Finish

Right Sock → RightSockOn → Right Shoe → Finish

Total-Order Plans:

Start

Start → Right Sock → Left Sock → Right Sock → Left Sock → Right Shoe → Left Shoe → Right Shoe → Left Shoe → Right Shoe → Finish

Start → Right Sock → Left Sock → Right Sock → Left Sock → Right Shoe → Left Shoe → Right Shoe → Left Shoe → Right Shoe → Finish

Start → Right Sock → Left Sock → Right Sock → Left Sock → Right Shoe → Left Shoe → Right Shoe → Left Shoe → Right Shoe → Finish

Start → Right Sock → Left Sock → Right Sock → Left Sock → Right Shoe → Left Shoe → Right Shoe → Left Shoe → Right Shoe → Finish

Start → Right Sock → Left Sock → Right Sock → Left Sock → Right Shoe → Left Shoe → Right Shoe → Left Shoe → Right Shoe → Finish

02/03/2017 Michael Herrmann Inf2D 17
Partial-order planning (POP) as a search problem

Define POP as search problem over plans consisting of:

- **Actions**: initial plan contains dummy actions
  - \textit{Start}(no preconditions, effect = initial state) and
  - \textit{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 $C$ ($A \prec C$ and $C \prec B$) with effect $\neg p$ 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 \xrightarrow{RightSockOn} RightShoe, LeftSock \xrightarrow{LeftSockOn} LeftShoe, RightShoe \xrightarrow{RightShoeOn} Finish, LeftShoe \xrightarrow{LeftShoeOn} 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 \( C \prec A \)
  3. Goal test: check whether there are open preconditions (only consistent plans are generated)
Partial-order planning example (1)

\[ \text{Init}(\text{At}(\text{Flat}, \text{Axle}) \land \text{At}(\text{Spare}, \text{Trunk})). \quad \text{Goal}(\text{At}(\text{Spare}, \text{Axle})). \]

\[ \text{Action}(\text{Remove}(\text{Spare}, \text{Trunk}), \]
\[ \quad \text{Precond: } \text{At}(\text{Spare}, \text{Trunk}) \]
\[ \quad \text{Effect: } \neg \text{At}(\text{Spare}, \text{Trunk}) \land \text{At}(\text{Spare}, \text{Ground})) \]

\[ \text{Action}(\text{Remove}(\text{Flat}, \text{Axle}), \]
\[ \quad \text{Precond: } \text{At}(\text{Flat}, \text{Axle}) \]
\[ \quad \text{Effect: } \neg \text{At}(\text{Flat}, \text{Axle}) \land \text{At}(\text{Flat}, \text{Ground})) \]

\[ \text{Action}(\text{PutOn}(\text{Spare}, \text{Axle}), \]
\[ \quad \text{Precond: } \text{At}(\text{Spare}, \text{Ground}) \land \neg \text{At}(\text{Flat}, \text{Axle}) \]
\[ \quad \text{Effect: } \neg \text{At}(\text{Spare}, \text{Ground}) \land \text{At}(\text{Spare}, \text{Axle})) \]

\[ \text{Action}(\text{LeaveOvernight}, \]
\[ \quad \text{Precond:} \]
\[ \quad \text{Effect: } \neg \text{At}(\text{Spare}, \text{Ground}) \land \neg \text{At}(\text{Spare}, \text{Axle}) \land \]
\[ \neg \text{At}(\text{Spare}, \text{Trunk}) \land \neg \text{At}(\text{Flat}, \text{Ground}) \land \neg \text{At}(\text{Flat}, \text{Axle})) \]

02/03/2017 Michael Herrmann Inf2D 17
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:

\[
\begin{align*}
\text{Start} & \quad \text{At}(\text{Spare, Trunk}) \\
\text{At}(\text{Flat, Axle}) & \quad \text{Remove}(\text{Spare, Trunk}) \\
\text{At}(\text{Spare, Ground}) & \quad \text{PutOn}(\text{Spare, Axle}) \\
\text{At}(\text{Spare, Axle}) & \quad \text{Finish}
\end{align*}
\]
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
  $Remove(Spare, Trunk) \xrightarrow{At(Spare,Ground)} PutOn(Spare, Axle)$

- Resolve by adding
  $LeaveOvernight \prec Remove(Spare, Trunk)$
Why is this the only solution?
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 $\text{Remove}(Flat, Axle)$ instead of $\text{LeaveOvernight}$
- Next, choose $\text{At}(Spark, Trunk)$ precondition of $\text{Remove}(Spare, Trunk)$
  Choose $\text{Start}$ to achieve this
- Pick $\text{At}(Flat, Axle)$ precondition of $\text{Remove}(Flat, Axle)$, choose $\text{Start}$ to achieve it
- Final, complete, consistent plan:
Dealing with unbound variables

- In first-order case, unbound variables may occur during planning process.
- Example:
  
  \[
  \text{Action}(\text{Move}(b, x, y), \\
  \text{Precond}: \ On(b, x) \land Clear(b) \land Clear(y) \\
  \text{Effect}: \ On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))
  \]
  
  achieves \(On(A, B)\) under substitution \(\{b/A, y/B\}\).

- Applying this substitution yields
  
  \[
  \text{Action}(\text{Move}(A, x, B), \\
  \text{Precond}: \ On(A, x) \land Clear(A) \land Clear(B) \\
  \text{Effect}: \ On(A, B) \land Clear(x) \land \neg On(A, x) \land \neg Clear(B))
  \]
  
  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
  \[ \text{Move}(A, x, B) \xrightarrow{\text{On}(A,B)} \text{Finish} \] would be added

- If another action has effect \( \neg \text{On}(A, z) \) then this is only a conflict if \( z = B \)

- Solution: insert \textit{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
Sussman anomaly

Start:

Goal 1: $On(A, B)$

Goal 2: $On(B, C)$

After achieving one of the goals, the agent cannot now pursue the other goal without undoing the first one.
Summary

- State-space search approaches (forward/backward)
- Heuristics for state-space search planning
- 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 I