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

Planning with state-space search

- 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
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 all 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

Air cargo domain example

▶ Goal can be described as
  \[ \text{At}(C_1, B) \land \text{At}(C_2, B) \land \ldots \land \text{At}(C_{20}, B) \]
▶ 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:
  \[ \text{In}(C_1, p) \land \text{At}(p, B) \land \ldots \land \text{At}(C_{20}, B) \]
▶ \( \text{Load}(C_1, p) \) would be inconsistent (negates \( \text{At}(C_1, B) \))
Backward state-space search

- 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 (planning) 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 \((\text{subgoal decomposition})\)
  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 \textbf{totally ordered} sequences of actions
- Better not to commit ourselves to complete chronological ordering of tasks \((\text{least commitment} \text{ strategy})\)
- 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 \textbf{linearisations} of partial-order plan
- Search in \textbf{plan space} rather than state spaces (because your search is over ordering constraints on actions, as well as transitions among states).
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 $A \rightarrow B$ 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

- **Initial plan**:
  Actions: \{`Start`, `Finish`\}, Orderings: \{`Start ⇋ `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 $A \rightarrow B$ and $A \prec B$ to plan. If $A$ new, add $A$ and `Start ⇋ 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 $A \rightarrow B$ 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)

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

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`(`Spares`, `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 \( \text{At}(\text{Spare}, \text{Axle}) \) of \( \text{Finish} \)
  Only applicable action = \( \text{PutOn}(\text{Spare}, \text{Axle}) \)
- Pick \( \text{At}(\text{Spare}, \text{Ground}) \) from \( \text{PutOn}(\text{Spare}, \text{Axle}) \)
  Only applicable action = \( \text{Remove}(\text{Spare}, \text{Trunk}) \)
- Situation after two steps:

Partial-order planning example (3)

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

Partial-order planning example (4)

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

Partial-order planning example (5)

- Now choose \( \text{Remove}(\text{Flat}, \text{Axle}) \) instead of \( \text{LeaveOvernight} \)
- Next, choose \( \text{At}(\text{Spark}, \text{Trunk}) \) precondition of \( \text{Remove}(\text{Spare}, \text{Trunk}) \)
  Choose \( \text{Start} \) to achieve this
- Pick \( \text{At}(\text{Flat}, \text{Axle}) \) precondition of \( \text{Remove}(\text{Flat}, \text{Axle}) \), 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:
    
    Action\((\text{Move}(b, x, y))\),
    
    PRECOND: \(\text{On}(b, x) \land \text{Clear}(b) \land \text{Clear}(y)\)
    
    EFFECT: \(\text{On}(b, y) \land \text{Clear}(x) \land \neg \text{On}(b, x) \land \neg \text{Clear}(y)\)
    
    achieves \(\text{On}(A, B)\) under substitution \(\{b/A, y/B\}\)
  - Applying this substitution yields
    
    Action\((\text{Move}(A, x, B))\),
    
    PRECOND: \(\text{On}(A, x) \land \text{Clear}(A) \land \text{Clear}(B)\)
    
    EFFECT: \(\text{On}(A, B) \land \text{Clear}(x) \land \neg \text{On}(A, x) \land \neg \text{Clear}(B)\)
    
    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
  
  \(\text{Move}(A, x, B)^{\text{On}(A, B)} \rightarrow \text{Finish}\) would be added
  
- If another action has effect \(\neg \text{On}(A, z)\) then this is only a conflict if \(z = B\)
  
- Solution: insert 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

- 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