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

Backward state-space search

- 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
  \[ At(C_1, B) \land At(C_2, B) \land \ldots \land At(C_{20}, B) \]
- To achieve \( At(C_1, B) \) 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 \( \text{In}(C_1, p) \land At(p, B) \), and should not include \( At(C_1, B) \) (otherwise irrelevant action)
- Full predecessor:
  \[ \text{In}(C_1, p) \land At(p, B) \land \ldots \land At(C_{20}, B) \]
- \( \text{Load}(C_1, p) \) would be inconsistent (negates \( At(C_1, B) \))
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 (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:
  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 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, preconditions=goal literals)
- **Ordering constraints** on actions `A ≺ B` (A must occur before B); contradictory constraints prohibited
- **Causal links** between actions `A ⇒ 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 `¬p (A ≺ C and C ≺ 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 ≺ RightShoe, LeftSock ≺ LeftShoe}`
  - Links: `{RightSock⇒RightShoe, LeftSock⇒LeftShoe, RightShoe⇒Finish, LeftShoe⇒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!)

Partial-order planning example (1)

```
Init(At(Flat, Axle) ∧ At(Spare, Trunk)).
Goal(At(Spare, Axle)).
Action(Decrease(Spare, Trunk)).
  Precond: At(Spare, Trunk)
  Effect: ¬At(Spare, Trunk) ∧ At(Spare, Ground))
Action(Decrease(Flat, Axle)),
  Precond: At(Flat, Axle)
  Effect: ¬At(Flats, Axle) ∧ At(Flat, Ground))
Action(PutOn(Spare, Axle),
  Precond: At(Spare, Ground) ∧ ¬At(Flat, Axle)
  Effect: ¬At(Spare, Ground) ∧ At(Spare, Axle))
Action(Decrease(Spare, Axle)),
  Precond: At(Spare, Ground) ∧ ¬At(Flat, Axle)
  Effect: ¬At(Spare, Ground) ∧ At(Spare, Axle))
```

Goal test: check whether there are open preconditions (only consistent plans are generated)
Partial-order planning example (2)

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

Partial-order planning example (3)

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

Partial-order planning example (4)

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

Partial-order planning example (5)

- Now choose \(\text{Remove}(\text{Flat, Axle})\) instead of \(\text{LeaveOvernight}\).
- Next, choose \(\text{At}(\text{Spark, Trunk})\) precondition of \(\text{Remove}(\text{Spare, Trunk})\).
  Choose \(\text{Start}\) to achieve this.
- Pick \(\text{At}(\text{Flat, Axle})\) precondition of \(\text{Remove}(\text{Flat, Axle})\), 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:

  \[
  \text{Action}(\text{Move}(b, x, y), \\
  \text{Precond}: \text{On}(b, x) \land \text{Clear}(b) \land \text{Clear}(y) \\
  \text{Effect}: \text{On}(b, y) \land \text{Clear}(x) \land \lnot \text{On}(b, x) \land \lnot \text{Clear}(y))
  \]

  achieves \(\text{On}(A, B)\) under substitution \(\{b/A, y/B\}\)
- Applying this substitution yields

  \[
  \text{Action}(\text{Move}(A, x, B), \\
  \text{Precond}: \text{On}(A, x) \land \text{Clear}(A) \land \text{Clear}(B) \\
  \text{Effect}: \text{On}(A, B) \land \text{Clear}(x) \land \lnot \text{On}(A, x) \land \lnot \text{Clear}(B))
  \]

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

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

Also has an effect on links, e.g. in example above

\[
\text{Move}(A, x, B) \rightarrow \text{On}(A, B)
\]

\(\text{Finish}\) would be added
- If another action has effect \(\lnot \text{On}(A, z)\) 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