Planning

COMP3431 Robot Software Architectures
Planning

A planner finds sequences of actions that will cause transitions from an initial state through intermediates states to a goal state
Actions

• Transitions from one state to the next are achieved by *actions*.

• Must specify how actions work

• Must work out correct sequence of actions to reach goal
Action Models

• Action action(<parameters>)

• PRECOND: <conditions that must be true to apply this action>

• EFFECTS: <conditions that become true or false after executing the action>
Action Example

**Action** Fly(p, from, to)

PRECOND: Plane(p) \land At(p, from) \land Airport(from) \land Airport(to)

EFFECT: \neg At(p, from) \land At(p, to))

- positive and negative literals in effects can be separated into an *add list* and a *delete list*
Example

Init:    Airport(MEL) ∧ Airport(SYD) ∧ Plane(P1) ∧ Plane(P2) ∧ Cargo(C1) ∧ Cargo(C2) ∧
            At(C1, SYD) ∧ At(C2, MEL) ∧ At(P1, SYD) ∧ At(P2, MEL)

Goal:    At(C1, MEL) ∧ At(C2, SYD)

Action   Load(c, p, a)
            PRECOND:     At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
            EFFECT:     ¬ At(c, a) ∧ In(c, p)

Action   Unload(c, p, a)
            PRECOND:     In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
            EFFECT:     At(c, a) ∧ ¬ In(c, p)

Action   Fly(p, from, to)
            PRECOND:     At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
            EFFECT:     ¬ At(p, from) ∧ At(p, to)
Progression and Regression

- Forward Search

- Backward Search
Backward Regression

\[ g' = (g - \text{Add}(a)) \cup \text{Precond}(a) \]

- \( g' \) is the regression from goal \( g \) over action \( a \)
- I.e. going backwards from \( g \), we look for an action, \( a \), that has preconditions and effects that satisfy \( g' \)
Planning and TR Programs

Action :-

- goal → do_nothing
- precond → action
- ....
- start → action

- TR Programs list actions from a plan, keeping preconditions
- Each rule below should be the regression of the rule above
Sussman’s Anomaly

- Goal: \( \text{On}(A, B) \land \text{On}(B, C) \)

- Try achieving \( \text{On}(A, B) \) first
  
  \[
  [\text{move}(c,a,\text{floor}), \text{move}(a,\text{floor},b), \text{move}(a,b,\text{floor}), \text{move}(b,\text{floor},c)]
  \]

- Trying \( \text{On}(B, C) \) first
  
  \[
  [\text{move}(b,\text{floor},c), \text{move}(b,c,\text{floor}), \text{move}(c,a,\text{floor}), \text{move}(a,\text{floor},b)]
  \]

- Should be:
  
  \[
  [\text{move}(c,a,\text{floor}), \text{move}(b,\text{floor},c), \text{move}(a,\text{floor},b)]
  \]
WARPLAN


• WARPLAN tries to interleave actions by protecting goals.
  • Achieve on(A,B): [move(c,a,floor), move(a,floor,b)]
  • Protect on(A,B)

• Now try on(B,C) by appending actions to end of plan
  • If it tries to undo a protected goal, move backwards through plan and try to slot new plan in.
Warplan

• \([\text{move}(c, a, \text{floor}), \text{move}(a, \text{floor}, b), \text{move}(a, b, \text{floor}), \ldots]\)

• \([\text{move}(c, a, \text{floor}), \ldots, \text{move}(a, \text{floor}, b)]\)

  Try inserting plan for on(B,C) here

• check that goals before and after are preserved
Partially Ordered Plans
Partial-Order Planning

Init: \( \text{Tire(Flat)} \land \text{Tire(Spare)} \land \text{At(Flat, Axle)} \land \text{At(Spare, Boot)} \)

Goal: \( \text{At (Spare, Axle)} \)

Action Remove(obj, loc)

- PRECOND: \( \text{At(obj, loc)} \)

- EFFECT: \( \neg \text{At(obj, loc)} \land \text{At(obj, Ground)} \)

Action PutOn(t, Axle)

- PRECOND: \( \text{Tire(t)} \land \text{At(t, Ground)} \land \neg \text{At(Flat, Axle)} \)

- EFFECT: \( \neg \text{At(t, Ground)} \land \text{At(t, Axle)} \)
Partial-Order Planning

Start

At(Spare,Boot)
At(Flat,Axle)

At(Spare,Axle)

Finish

Remove(Spare,Boot)

At(Spare,Boot)
At(Flat,Axle)

At(Spare,Axle)

PutOn(Spare,Axle)

At(Spare,Ground)
¬ At(Flat,Axle)

At(Spare,Axle)

Finish

Remove(Spare,Boot)

At(Spare,Boot)
At(Flat,Axle)

At(Spare,Axle)

PutOn(Spare,Axle)

At(Spare,Ground)
¬ At(Flat,Axle)

At(Spare,Axle)

Finish

Remove(Spare,Boot)

At(Spare,Boot)
At(Flat,Axle)

At(Spare,Axle)

PutOn(Spare,Axle)

At(Spare,Ground)
¬ At(Flat,Axle)

At(Spare,Axle)

Finish

Remove(Flat,Axle)

At(Flat,Axle)

Remove(Flat,Axle)

At(Flat,Axle)

¬ At(Flat,Axle)
Forward Planning

- Forward planners are now among the best.
- Use heuristics to estimate costs
- Possible to use heuristic search, like A*, to reduce branching factor.