Planning

A plan is a sequence of actions

- It should lead from an initial state to some desired target state.
- Actions can be represented like predicates, e.g.
  fly(g-avrl, sfo, jfk).
- But they aren't predicates: they are clearly not true or false.
- States can be represented by logical formulæ, e.g.
  aboard(fred, g-avrl) ∧ at(g-avrl, sfo)
- Here the parts really are like predicates, but they are called Fluents
- They can change between true and false as events progress
- States are always just conjunctions
  No ∨'s: at this stage they can't represent uncertainty

Actions have Preconditions

- The action can only be taken if the precondition is true
  e.g. for the action fly(P, From, To), the precondition might be
  at(P, From) ∧ is-aeroplane(P) ∧ is-airport(From) ∧ is-airport(To)
  Usually they only contain ∧'s and ¬'s.
- Actions also have Effects, which are also logical formulæ
  e.g. for the action fly(P, From, To), the effect might be
  ¬at(P, From) ∧ at(P, To)
  The effect is always just a conjunction, maybe with some ¬'s
- An Action Schema is used to connect everything together:
  Action(fly(P, From, To),
  PRECOND: at(P, From) ∧ aero(P) ∧ airpt(From) ∧ airpt(To)
  EFFECT: ¬at(P, From) ∧ at(P, To))
- Ugly syntax, just to save work
- A Ground Action has all of its variables replaced by actual values, e.g.
  fly(g-avrl, sfo, jfk)
- A ground action is Applicable in a state
  If that state implies its precondition

A Problem in this world consists of

- A Goal: the state we want things in at the end
- An initial state: how things are before we start
  The goal and initial states must be Ground States:
  they contain no variables, only constants
- A list of the actions that can be used

The solution to a problem is a Plan, just a sequence of actions:

[get-on(fred, g-avrl), fly(g-avrl, sfo, jfk), get-off(fred, g-avrl)]

A PDDL (Planning Domain Definition Language) example:

Init(at(c1, sfo) ∧ at(c2, jfk) ∧ at(g-avrl, sfo) ∧ at(ja8089, jfk)
∧ is-cargo(c1) ∧ is-cargo(c2) ∧ is-plane(g-avrl) ∧ is-plane(ja8089)
∧ is-airport(sfo) ∧ is-airport(jfk))
Goal(at(c1, jfk) ∧ at(c2, sfo))

Action(load(Cargo, Plane, Airport),
PRECOND: at(Cargo, Airport) ∧ at(Plane, Airport)
∧ is-cargo(Cargo) ∧ is-plane(Plane) ∧ is-airport(Airport),
EFFECT: ¬at(Cargo, Airport) ∧ inside(Cargo, Plane))
Action(unload(Cargo, Plane, Airport),
PRECOND: inside(Cargo, Plane) ∧ at(Plane, Airport)
∧ is-cargo(Cargo) ∧ is-plane(Plane) ∧ is-airport(Airport),
EFFECT: ¬inside(Cargo, Plane) ∧ at(Cargo, Airport))
Action(fly(Plane, From, To),
PRECOND: at(Plane, From) ∧ is-plane(Plane) ∧ is-airport(From)
∧ is-airport(To),
EFFECT: ¬at(Plane, From) ∧ at(Plane, To))

A very famous example

- The "blocks" world
- "Pick up a big red block"

Init(on(a, table) ∧ on(b, table) ∧ on(c, a)
∧ is-block(a) ∧ is-block(b) ∧ is-block(c)
∧ clear(b) ∧ clear(c) ∧ clear(table))
Goal(on(a, b) ∧ on(b, c))
Action(move-to-block(Block, From, To),
PRECOND: on(Block, From) ∧ clear(Block) ∧ clear(To)
∧ is-block(Block) ∧ is-block(To)
∧ Block≠From ∧ Block≠To ∧ From≠To,
EFFECT: on(Block, To) ∧ clear(From) ∧ ¬on(Block, From) ∧ ¬clear(To))
Action(move-to-table(Block, From),
PRECOND: on(Block, From) ∧ is-block(Block) ∧ is-block(From)
∧ clear(Block),
EFFECT: on(Block, table) ∧ clear(From) ∧ ¬on(Block, From))

Formulating a plan

- Finding a plan is just another search
- But we are searching for a big complicated thing this time: a plan
- For any real example, the state space will be very big
  We will need a good heuristic
- Closed World assumption: states don't *need* to include things that are false but usually they do
  An effect including ¬ just removes that fluent from the state.
- Partially ordered plans?

Forward search

- Start at initial state
- Unify current state with preconditions for each action
- Whenever successful,
  Apply the substitution to the action to find a step in the plan
  Apply the substitution to the effects and
  add them to the current state to find the next state

Backward search, or Regression search

- Start at the goal state
- Unify current state with effects of each action
  - but don't allow any effects that negate any part of the goal
  - what if the successful plan involves a desired fluent being false for just a little while?
- Whenever successful,
  - Apply the substitution to the action to find a step in the plan
  - Generate the next state by
    - removing any positive fluents in the effect from the goal,
    - adding any positive fluents in the precondition,
    - removing any negative fluents in the effect,
    - adding any negative fluents in the precondition.
  - So in this case, states *do* have to include negative fluents too.
- Many times, a backward search can have far fewer next states to explore at each step

Heuristics

- An admissible heuristic never over-estimates the remaining cost
- Sometimes relaxing the problem reveals a good heuristic:
  - an exact cost in the relaxed problem can be a heuristic in the original
  - but for that to be practical, the relaxed problem must be very quick and easy to solve.
- Maybe just ignore all the preconditions, that certainly won't over-estimate
- Any goal fluent can be made true with just one action
  - If it can be made true at all, that is
- Back to the eight puzzle for an example:
  - Action(slide(Tile, From-square, To-square),
    - PRECOND: is-tile(Tile) ∧ is-empty(To-square)
      - ∧ in(Tile, From-square) ∧ adjacent(From-square, To-square),
    - EFFECT: in(Tile, From-square) ∧ is-empty(From-square)
      - ∧ ¬in(Tile, To-square) ∧ ¬is-empty(To-square))
- Ignoring *all* the preconditions is silly
  - You'd even try moving things that aren't tiles
- Ignore is-empty(To-square) ∧ adjacent(From-square, To-square)
  - Any tile can move anywhere in one go
  - The heuristic is the number of out-of-place tiles
- Only ignore is-empty(To-square)
  - Any tile can move to any adjacent square even if it's occupied
  - The heuristic is the Manhattan Distance

High-level actions (HLAs)

- A complete plan for an autonomous robot can have *very* many actions
  - To get from one place to another, a robot must activate its tiny little motors in exactly the right order for every single step taken
- Fortunately, plans in the real world tend to be very hierarchical
- One single *very* high-level action can be a complete plan

[move-from-to(lab, repair-shop)]

- this can be resolved into a plan involving *quite* high-level actions

  [move-from-to(lab, corridor-outside-lab),
  move-from-to(corridor-outside-lab, corridor-outside-repair-shop),
  move-from-to(corridor-outside-repair-shop, repair-shop)]
- each of those actions are resolved by their own individual plans involving slightly lower-level actions
- and so on, all the way down to plans involving the most basic actions which the robot can actually physically do, e.g. activate or deactivate a motor
- All of those sub-plans can be discovered individually when the time comes
- and just concatenated together

Non-determinism

- Perhaps you can't be sure what the state of the world is

  you haven't got a sensor for that particular thing
- Or perhaps you can't be sure what effect an action will actually cause

  turning the wheels might not move the vehicle, it might be muddy
- When a problem can be solved by more than one possible plan a choice must be made
- Angelic selection: the agent can choose which plan to take

  Only requires that just one of the possible plans would work
- Dæmonic selection: something else, the environment, forces the choice

  Requires that every single possible plan must work
- But really, a plan isn't a solution if it doesn't achieve the goal

  A plan is a sequence of actions that solves a problem.