Classical Planning

A plan is a collection of actions for performing some task (reaching some goal)

If we have a robot we want the robot to

1. Decide what to do, and
2. Figure out what actions it needs to do in order to accomplish its goals
We want to change the world to suit our needs.

**Problem:** Need to reason about what the world will be like after taking certain actions

**Goal:** Kate has coffee and has food in the fridge and the bookshelf is fixed

**Currently:** Robot is at home, has no coffee, coffee is not made, no food in the fridge, …

**To Do:** Go to the kitchen, make coffee, bring it to Kate, go to the store,…
Planning

• Planning is basically searching over sets of states while also reasoning over the effects of actions

• Optimal plan will be the one with smallest number of actions

• This is a lot like search BUT
  - Representation is extremely important
Planning vs Search

Consider the task get milk, bananas, and a hammer. Standard search fails miserably.
Planning Languages

- By using a structured and restricted planning language we can do better than standard search algorithms
  - Connect state and action descriptions
  - Allow the adding of actions in any order
  - Establish independent subproblems and solve the separately
Domain

- Set of typed objects (usually represented as propositions)
- B and Shakey are OK, but x and Robot(x) are not

States

- Conjunctions of first-order predicates over objects
- $\text{On}(A,B) \land \text{On}(B,C)$ is allowed but not $\text{On}(x,y) \land \text{On}(y,z)$

Closed-World Assumption

- Any conditions not mentioned in a state are assumed to be false
- This is required to overcome the Frame Problem
Block World

Domain: A, B and C

States:
OnTable(A) \land OnTable(B) \land On(C,A) \land HandEmpty()
Goals

- Conjunctions of positive ground literals
- OnTable(A) ∧ On(B, A) ∧ On(B, C) ∧ HandEmpty()
**STRIPS**

**Actions** are specified by their **preconditions** and their **effects**

Fly(p, from, to)

**PRECOND:** $\text{At}(p, \text{from}) \land \text{Plane}(p) \land \text{Airport}(\text{from}) \land \text{Airport}(\text{to})$

**EFFECT:** $\neg \text{At}(p, \text{from}) \land \text{At}(p, \text{to})$

Description of how the state changes when the action is executed. Variables in the effect must be included in the original parameter list.

Effects are sometimes represented as **Add-lists** and **Delete-lists**.
Add-list: propositions that become true
Delete-list: propositions that become false

Description of what must be true in order for the action to be executed. (Conjunction of function-free positive literals)
STRIPS

Semantics:

• If the precondition is false in a world state then the action changes nothing (it can not be applied)

• If the precondition is true
  • Delete items from the Delete-list
  • Add items in the Add-list
  • Order of operations is important

Solution:

• Action sequence that when executed in the start state results in a state that satisfies the goal
Example

- Init(At(Flats, Axles) ^ At(Spare, Trunks))

- Goal( At(Spare, Axle))

- Action(Remove(Spare, Trunk),
  - PRECOND: At(Spare, Trunk)
  - EFFECT: ~At(Spare, Trunk) ^ At(Spare, Ground))

- Action(Remove(Flat, Axle),
  - PRECOND: At(Flat, Axle)
  - EFFECT: ~At(Flat, Axle) ^ At(Flat, Ground) ^ Clear(Axle))

- Action(PutOn(Spare, Axle),
  - PRECOND: At(Spare, Ground) ^ Clear(Axle)
  - EFFECT: ~At(Spare, Ground) ^ At(Spare, Axle))

- Action(LeaveOverNight,
  - PRECOND:
  - EFFECT: ~At(Spare, Ground) ^ ~At(Spare, Axle) ^ ~At(Spare, Trunk) ^ ~At(Flat, Ground) ^ ~At(Flat, Axle))
Example

Define the action *Move* object from someplace to another place
Planning as Search

- **Progression Planning (Forward Planning)**
  - This is precisely search like we saw earlier in the course
  - You need good heuristics but these can be domain independent

- **Regression Planning (Backward Planning)**
  - Start from the goal state
  - Find consistent, relevant actions
    - Consistent: it can not undo any desired literals
    - Relevant: it must achieve one of the conjuncts of the goal
Example

Initial State:
- Clear(c)
- Clear(a)
- Clear(b)
- OnTable(b)
- OnTable(a)
- OnTable(c)
- HandEmpty()

Goal:
- Clear(a)
- Clear(c)
- On(b,c)

Pickup(x)
- P: OnTable(x), Clear(x), HandEmpty
- E: Holding(x), ~OnTable(x), ~HandEmpty

PutDown(x)
- P: Holding(x)
- E: OnTable(x), Clear(x), HandEmpty,
  ~Holding(x)

Stack(x,y)
- P: Holding(x), Clear(y)
- E: On(x,y), Clear(x),
  HandEmpty, ~Clear(y),
  ~Holding(x)

UnStack(x,y)
- P: Clear(x), On(x,y), HandEmpty
- E: Clear(y), Holding(x),
  ~Clear(x), ~On(x,y),
  ~HandEmpty
Planning Graphs

It can be useful to represent planning problems as planning graphs
- For deriving heuristics
- For running particular algorithms

Planning graphs consist of **levels**
- $S_0$ has a node for each literal that holds in the initial state
- $A_0$ has nodes for each action that could be taken in $S_0$
- $S_i$ contains all literals that could hold given the actions taken in level $A_{i-1}$
- $A_i$ contains all actions who’s preconditions could hold in $S_i$
Planning Graphs

Init: Have (Cake)
Goal: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
PRECOND: Have(Cake)
EFFECT: ~Have(Cake) ∧ Eaten(Cake)

Action: Bake(Cake)
PRECOND: ~Have(Cake)
EFFECT: Have(Cake)
Planning Graphs

**Persistence Actions:** Once a literal appears, then it can persist if no action negates it (no-op)

**Mutual Exclusion Links (Mutex):** Record conflicts between actions that can not occur together
Planning Graphs

Mutual Exclusion Links (Mutex): Record conflicts between actions that can not occur together

- **Inconsistent Effects**: (actions) An effect of one negates the effect of another
- **Interference**: (actions) One deletes a precondition of another
- **Competing Needs**: (actions) Mutually inconsistent preconditions
- **Inconsistent Support**: (states) One is a negation of another OR all ways of achieving them are mutually exclusive
Using Planning Graphs

Observations

• Graph is polynomial in the size of the planning problem.

• If any goal literal does not appear in the final level then the problem is unsolvable.
Using Planning Graphs

Heuristics

- For a single goal literal \( g \), the level in which it first appears is an admissible heuristic (level-cost(\( g \)))

- For multiple goal literals \( (g_1 \land g_2 \land \ldots) \)
  - Max-level heuristic: Max level-cost(\( g_i \)) (admissible)
  - Level-sum heuristic: \( \sum \) level-cost(\( g_i \)) (may be inadmissible)
  - Set-level heuristic: Level where all goal literals appear and are not mutex (admissible and dominates max-level)
Example: Planning Graphs

Level-cost?
Max-level?
Level-sum?
Set-level?
GraphPlan

- Start with an empty graph
- Iterate until you find a solution
  - Graph expansion
  - Analyze graph for mutex
  - Check if there is a possible solution
  - If yes, extract-solution
Solution extraction is backward search through the planning graph

\textbf{Extract-Solution}(S_G, t)

- If \( t=0 \) return solution
- For each proposition \( s \) in \( S_G \)
  - Choose an action in \( A_{t-1} \) to achieves \( s \)
- If any pair of actions chosen are mutex then backtrack
- \( S_{G'} = \) set of preconditions for chosen actions
- \textbf{Extract-Solution}(S_{G'}, t-1)
Example

- Init(At(Flat, Axle) \land At(Spare, Trunk))

- Goal(At(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) \land Clear(Axle))

- Action(PutOn(Spare, Axle),
  - PRECOND: At(Spare, Ground) \land Clear(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)
Example
GraphPlan Properties

• Sound and complete
  - Search must terminate
  - Any plan found is a sound plan

• Optimal
  - Finds shortest length plan assuming that multiple actions may occur at the same time

• Time complexity
  - Polynomial time to construct the planning graph
  - However planning is PSPACE-complete. Thus, extraction may be intractable