Planning vs problem solving
- Situation calculus
- Plan-space planning

Planning as Problem Solving
- Planning:
  - Start state (S)
  - Goal state (G)
  - Set of actions
- Can be cast as "problem-solving" problem
- But, what if initial state is not known exactly? E.g., start in bottom row in 4x4 world, with goal being C.
- Do search over "sets" of underlying states to find a set of actions that will reach C for any starting state.

Planning as Logic
- The problem solving formulation in terms of sets of atomic states is incredibly inefficient because of the exponential blowup in the number of sets of atomic states.
- Logic provides us with a way of describing sets of states.
- Can we formulate the planning problem using logical descriptions of the relevant sets of states?
- This is a classic approach to planning: situation calculus, use the mechanism of FOL to do planning.
- Describe states and actions in FOL and use theorem proving to find a plan.

Situation Calculus
- Reify situations: [reify = name, treat them as objects] and use them as predicate arguments.
  - At(Robot, Room6, S₉) where S₉ refers to a particular situation
- Result function: a function that describes the new situation resulting from taking an action in another situation.
  - Result(MoveNorth, S₁) = S₆
- Effect Axioms: what is the effect of taking an action in the world
  - ∀ x.a. Present(x,a) ∧ Portable(x) → Holding(x, Result(Grab, s))
  - ∀ x.a. ¬ Holding(x, Result(Drop, s))
- Frame Axioms: what doesn’t change
  - ∀ x.a. color(x,a) = color(x, Result(Grab, s))
  - Can be included in Effect axioms
Planning in Situation Calculus

- Use theorem proving to find a plan
- Goal state: $\exists s. \text{At(Home, } s) \land \text{Holding(Gold, } s)$
- Initial state: $\text{At(Home, } s_0) \land \neg \text{Holding(Gold, } s_0) \land \text{Holding(Rope, } s_0)$ ...
- Plan: $\text{Result(North, Result(Grab, Result(South, } s_0)))}$
  - A situation that satisfies the requirements
  - We can read out of the construction of that situation what the actions should be.
  - First, move South, then Grab and then move North.

Special Properties of Planning

- Reducing specific planning problem to general problem of theorem proving is not efficient.
- We will build a more specialized approach that exploits special properties of planning problems.
  - Connect action descriptions and state descriptions [focus searching]
  - Add actions to a plan in any order
  - Sub-problem independence
  - Restrict language for describing goals, states and actions

STRIPS representations

- States: conjunctions of ground literals
  - $\text{In(robot, } r_3) \land \text{Closed(door}_r) \land ...$
- Goals: conjunctions of literals
  - $(\text{implicit } r) \land \text{In(Robot, } r) \land \text{In(Charger, } r)$
- Actions (operators)
  - Name (implicit $r$): $\text{Go(here, there)}$
  - Preconditions: conjunction of literals
    - $\text{At(here)} \land \text{path(here, there)}$
  - Effects: conjunctions of literals [also known as post-conditions, add-list, delete-list]
    - $\text{At(there)} \land \neg \text{At(here)}$
  - Assumes no inference in relating predicates (only equality)

Strips Example

- Action
  - $\text{Buy(x, store)}$
    - Pre: $\text{At(store)}, \text{Sells(store, } x)$
    - Eff: $\text{Have(x)}$
  - $\text{Go(x, y)}$
    - Pre: $\text{At(x)}$
    - Eff: $\text{At(y)}, \neg \text{At(x)}$
- Goal
  - $\text{Have(Milk)} \land \text{Have(Banana)} \land \text{Have(Drill)}$
- Start
  - $\text{At(Home)} \land \text{Sells(SM, Milk)} \land \text{Sells(SM, Banana)} \land \text{Sells(HW, Drill)}$

Planning Algorithms

- Progression planners: consider the effect of all possible actions in a given state.
- Regression planners: to achieve a goal, what must have been true in previous state.
  - $\text{Have(M)} \land \text{Have(B)} \land \text{Have(D)}$
  - $\text{Buy(M,store)}$
    - $\text{At(store)} \land \text{Sells(store,M)} \land \text{Have(B)} \land \text{Have(D)}$
- Both have problem of lack of direction – what action or goal to pursue next.

Plan-Space Search

- Situation space – both progressive and regressive planners plan in space of situations
- Plan space – start with null plan and add steps to plan until it achieves the goal
  - Decouples planning order from execution order
  - Least-commitment
    - First think of what actions before thinking about what order to do the actions
  - Means-ends analysis
    - Try to match the available means to the current ends
Graph Plan

- A propositional planner, that is, there are no variables
- Simpler – don’t have to worry about matching
- Bigger – if you have six blocks, you need 36 propositions to represent all On(x,y) assertions

1. Make a plan graph of depth k
2. Search for a solution
3. If succeed, return a plan
4. Else k=k+1
5. Go to 1.

Plan Depth

A plan of depth k
- has k times steps
- may have multiple parallel actions per time step

Scheduling: tasks are fixed

Planning vs Scheduling

Planning: find steps and schedule

PSPACE-complete

Graph Plan: find plans of a given depth

Scheduling: tasks are fixed

NP-Complete
Making the Plan Graph

- Start with initial conditions
- Add actions with satisfied preconditions
Making the Plan Graph

- Start with initial conditions
- Add actions with satisfied preconditions
- Add all effects of actions at previous levels
- Add maintenance actions
**Mutually Exclusive Actions**

Two action instances at level $i$ are mutex if:
- **Inconsistent effects**: effect of one action is negation of effect of another
- **Interference**: one action deletes the precondition of the other
- **Competing needs**: the actions have preconditions that are mutex at level $i-1$

**Mutually Exclusive Propositions**

Two propositions at level $i$ are mutex if:
- **Negation**: they are negations of one another
- **Inconsistent support**: all ways of achieving the propositions at level $i-1$ are pairwise mutex.

**Solution Extraction**

- If all the literals in the goal appear at the deepest level and not mutex, then search for a solution for each subgoal at level $i$
  - For each subgoal at level $i$
    - Choose an action to achieve it
    - If it’s mutex with another action, Fail
  - Repeat for preconditions at level $i-2$

**Birthday Dinner Example**

- **Goal**: $\neg$garb $\land$ dinner $\land$ present
- **Init**: $\neg$garb $\land$ clean $\land$ quiet
- **Actions**:
  - **Cook**
    - Pre: clean
    - Effect: dinner
  - **Wrap**
    - Pre: quiet
    - Effect: present
  - **Carry**
    - Pre: garb
    - Effect: $\neg$garb $\land$ clean
  - **Dolly**
    - Pre: garb
    - Effect: $\neg$garb $\land$ quiet
<table>
<thead>
<tr>
<th>Post</th>
<th>Pre</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>garb</td>
<td>clean</td>
<td>quiet</td>
</tr>
<tr>
<td>dinner</td>
<td>present</td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>wrap</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>carry</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>dolly</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>clean</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>wrap</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>carry</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>dolly</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>clean</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>wrap</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>carry</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>dolly</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>clean</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>wrap</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>carry</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>dolly</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>clean</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>wrap</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>carry</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>dolly</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>clean</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>wrap</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>carry</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>dolly</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>clean</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>wrap</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>carry</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>dolly</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>clean</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>wrap</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>carry</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>dolly</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>clean</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>wrap</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>carry</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>dolly</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>clean</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>wrap</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>carry</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>dolly</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>clean</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>wrap</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>carry</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>dolly</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>clean</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>wrap</td>
<td>dinner</td>
<td>present</td>
</tr>
<tr>
<td>carry</td>
<td>garb</td>
<td>quiet</td>
</tr>
<tr>
<td>dolly</td>
<td>dinner</td>
<td>present</td>
</tr>
</tbody>
</table>
Extensions

- Lots of time optimizations
- Disjunctive preconditions
- Universally quantified (sort of) preconditions and effects
- Conditional planning