Planning

Meet Shakey
Planning

- What is planning?
  - Find a sequence of actions that achieves a desired result
  - Haven’t we done planning already?

- Planning with TreeSearch
  - Initial state: our, well, initial state
  - Successor function: which actions can we perform?
  - isGoal: Have we achieved the desired state?

Sketch of an idea

- Let’s say that we have some statements about the blocks world, like
  - On(B,A)
  - Clear(B)
  - On(A,T)
  - ~Clear(A)

- Can we “prove” that we can move block B?
  - On(B,A) ^ Clear(B) => On(B,T) ^ Clear(A)  (??)

- But adding this to the KB introduces contradictions!
  - E.g., Clear(A) ^ ~Clear(A)
Frame Axioms

"Time Is What Prevents Everything From Happening At Once..." - John Wheeler (1911-2008)

- An action divides time into a “before” and “after”.
  - Different things are true---
  - Some are changed explicitly by the action
  - Some “continue to be”

- **Frame axioms**: The way we we describe the “after” in terms of the “before” and the action effects

PDDL

- Planning Domain Definition Language (PDDL)
  - Expresses typical frame axioms automatically
  - Database semantics
    - Closed world (fluent are false by default)
    - Two constants (Bob, Mr. Henderson) *always* refer to different objects
  - Based on STRIPS language (1971) used by Shakey

- Environment
  - fully observable, deterministic, finite, static, discrete

- Objectives
  - conjunctions of goal propositions
PDDL Action Schema

- **Example:**
  - **ACTION:** Fly(p, from, to)
  - **PRECOND:** At(p, from) ^ Plane(p) ^ Airport(from) ^ Airport(to)
  - **EFFECT:** -At(p, from) ^ At(p, to)

- **In comparison to FOL**
  - **FOL:** Variables (use unification), Predicates, Functions, arbitrary connectives
  - **PDDL:** Variables (use unification), Predicates, No functions, conjunctions only

- **Goals**
  - Cannot have a vacuum which wants at least one clean room: Clean(Room1) v Clean(Room2)

- **Effects**
  - Sets values of propositions (overriding earlier values)
  - Everything else "continues to be"

A Blocks-World Problem

- **Initial:**
  - On(A,C) ^ On(B,Table) ^ On(C,Table) ^ Clear(A) ^ Clear(B)

- **Goal:**
  - On(B,C)

- **Define the clear predicate?**
  - Clear(x) = \( \forall y. \neg \text{On}(y,x) \)
  - But we can’t: not part of the syntax of PDDL
  - Instead, values of Clear predicate are updated by actions
A Blocks-World Problem

\[
\text{On}(A, C) \land \text{On}(B, \text{Table}) \land \text{On}(C, \text{Table}) \land \text{Clear}(A) \land \text{Clear}(B)
\]

\[
\text{Move}(b, x, y) \quad ("\text{move } b \text{ from } x \text{ to } y")
\]

**precondition:** \(\text{On}(b, x) \land \text{Clear}(b) \land \text{Clear}(y)\)

**effect:** \(\text{On}(b, y) \land \text{Clear}(x) \land \neg \text{On}(b, x) \land \neg \text{Clear}(y)\)

- Consider goal state: \(\text{On}(B, C)\)
  - Can be unified with action \(\text{Move}(b, x, y)\)
    - \(\theta = \{b/A, x/C, y/B\}\)
  - Precondition satisfied?
- Resulting state
  - \(\neg \text{On}(A, C) \land \text{On}(B, \text{Table}) \land \text{On}(C, \text{Table}) \land \neg \text{On}(A, B) \land \text{Clear}(C)\)

A small problem

\[
\text{On}(A, C) \land \text{On}(B, \text{Table}) \land \text{On}(C, \text{Table}) \land \text{Clear}(A) \land \text{Clear}(B)
\]

\[
\text{Move}(b, x, y) \quad ("\text{move } b \text{ from } x \text{ to } y")
\]

**precondition:** \(\text{On}(b, x) \land \text{Clear}(b) \land \text{Clear}(y)\)

**effect:** \(\text{On}(b, y) \land \text{Clear}(x) \land \neg \text{On}(b, x) \land \neg \text{Clear}(y)\)

- What happens if we perform \(\text{Move}(b, x, \text{Table})\)?
  - The table has a “special” property!
- How do we fix this?
Shakey the Robot

- Natural language interface
- STRIPS-style planner
  - Real-world implementation of blocks-world like problem

Shakey and STRIPS
Searching for Plans

- Given a plan (sequence of actions) and an initial state, can test whether plan achieves goal
- Q: How to generate solution plans?
- A: search (as always...)

Forward State-Space Search

- Also called progression planning
- Planning as state space search:
  - Represent states by sets of positive ground literals
    - Literals not appearing are false or don’t matter
    - Initial state: given by planning problem
  - Action applicable in a state iff preconditions satisfied
  - Successor states generated by adding positive effect literals and deleting negative effect literals
  - Goal test succeeds iff state satisfies goal sentence
  - Step cost = 1 (typically)
Forward Search: Complexity

- In the absence of function symbols, the state space of a planning problem is finite
  - Therefore any complete graph-search algorithm will be a complete planning algorithm

- But will it be efficient?
  - Many irrelevant actions
    - all applicable actions are considered in each state
  - What is branching factor for blocks world with $N$ blocks?
  - Need good heuristic functions

Backward State-Space Search

- Also called regression planning
- Generates predecessors starting from goal state
  - Find action $A$ whose effect unifies with goal (or part)
  - New “goal” is set of conditions for this action to be applicable
  - Computing these conditions is called regressing the goal through the action.
  - Delete positive effects of $A$ that appear in goal
  - Add precondition literals of $A$

- **Advantage:** need only consider relevant actions
- **Disadvantage:** dealing with interactions among goal propositions
Regression Example

- Goal: $\text{On}(B,C)$
- Choose action:
  - $\text{Move}(B,\text{Table},C)$
  - Achieves $\text{On}(B,C)$, has preconditions $\text{Clear}(B)$, $\text{Clear}(C)$, $\text{On}(B,\text{Table})$
- Choose action:
  - $\text{MoveToTable}(A,C)$
  - Achieves $\text{Clear}(C)$, has preconditions $\text{Clear}(A)$, $\text{On}(A,C)$
- Remaining conditions satisfied in initial state

---

Sussman Anomaly (Linear Planning)

- Initial state: $\text{On}(B,\text{Table}), \text{Clear}(B), \text{Clear}(C), \text{On}(C,A), \text{On}(A,\text{Table})$
- Goal: $\text{On}(A,B), \text{On}(B,C)$
Next Time

- Planning Graphs and real-world planning