Classical AI Planning: STRIPS Planning

Jane Hsu

Remember: Problem-Solving Agent

```
function SIMPLE-PROBLEM-SOLVING-AGENT( p) returns an action
    inputs: p, a percept
    static: s, an action sequence, initially empty
            state, some description of the current world state
            g, a goal, initially null
            problem, a problem formulation
    state ← UPDATE-STATE(state, p)
    if s is empty then
        g ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, g)
        s ← SEARCH( problem)
        action ← RECOMMENDATION(s, state)
    s ← REMAINDER(s, state)
    return action
```

Note: This is offline problem-solving. Online problem-solving involves acting w/o complete knowledge of the problem and environment.
The Planning Problem

To find an executable sequence of actions that achieves a given goal when performed starting in a given state.

Roots of Planning

- problem solving
- state-space search

EXAMPLE: “Shakey” the Robot (SRI)
- a robot that roamed the halls of SRI in the early 1970’s
- actions were based on STRIPS plans
A Simple Planning Agent

```sql
function SIMPLE-PLANNING-AGENT(percept) returns an action
    static: KB, a knowledge base (includes action descriptions)
    p, a plan (initially, NoPlan)
    t, a time counter (initially 0)
    local variables: G, a goal
                             current, a current state description
    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    current ← STATE-DESCRIPTION(KB, t)
    if p = NoPlan then
        G ← ASK(KB, MAKE-GOAL-QUERY(t))
        p ← IDEAL-PLANNER(current, G, KB)
        if p = NoPlan or p is empty then
            action ← NoOp
        else
            action ← FIRST(p)
            p ← REST(p)
            TELL(KB, MAKE-ACTION-SENTENCE(action, t))
            t ← t+1
    return action
```

Algorithm: A Simple Planning Agent

1. Generate a goal to achieve
2. Construct a plan to achieve goal from the current state
3. Execute plan until finished
4. Begin again with new goal
   - Use percepts to build a model of the current world state
   - IDEAL-PLANNER: Given a goal, algorithm generates a plan of actions
   - STATE-DESCRIPTION: given percept, return initial state description in format required by planner
   - MAKE-GOAL-QUERY: used to ask KB what the next goal should be
Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*

Standard search algorithms seem to fail miserably:

![Diagram showing a decision tree with nodes labeled with actions like 'Go To Pet Store', 'Buy a Dog', 'Talk to Parent', etc.]

After-the-fact heuristic/goal test inadequate

---

Search vs. planning

Planning systems do the following:

1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

<table>
<thead>
<tr>
<th>States</th>
<th>Search</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actions</td>
<td>Lisp data structures</td>
<td>Logical sentences</td>
</tr>
<tr>
<td>Goal</td>
<td>Lisp code</td>
<td>Preconditions/outcomes</td>
</tr>
<tr>
<td>Plan</td>
<td>Sequence from $S_0$</td>
<td>Logical sentence (conjunction)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Constraints on actions</td>
</tr>
</tbody>
</table>

Copyright (C) 2003 Jane Hsu
Basic Representation for Planning

- Most widely used approach: uses STRIPS language

- **states**: conjunctions of function-free ground literals (i.e., predicates applied to constant symbols, possibly negated); e.g.,
  \[ \text{At(Home)} \land \neg \text{Have(Milk)} \land \neg \text{Have(Bananas)} \land \neg \text{Have(Drill)} \ldots \]

- **goals**: also conjunctions of literals; e.g.,
  \[ \text{At(Home)} \land \text{Have(Milk)} \land \text{Have(Bananas)} \land \text{Have(Drill)} \]
  but can also contain variables (implicitly universally quant.); e.g.,
  \[ \text{At(x)} \land \text{Sells(x, Milk)} \]

Planning in Situation Calculus

\[ \text{PlanResult}(p, s) \] is the situation resulting from executing \( p \) in \( s \)

\[ \text{PlanResult}([], s) = s \]

\[ \text{PlanResult}([a], s) = \text{PlanResult}(p, \text{Result}(a, s)) \]

**Initial state** \( \text{At(Home, } S_0) \land \neg \text{Have(Milk, } S_0) \land \ldots \)

**Actions** as Successor State axioms

\[ \text{Have(Milk, Result}(a, s)) \iff [(a = \text{Buy}(\text{Milk}) \land \text{At(Supermarket, s)} \lor (\text{Have(Milk, s)} \land a \neq \ldots)] \]

**Query**

\[ s = \text{PlanResult}(p, S_0) \land \text{At(Home, s)} \land \text{Have(Milk, s)} \land \ldots \]

**Solution**

\[ p = [\text{Go(Supermarket)}, \text{Buy}(\text{Milk}), \text{Buy(Bananas)}, \text{Go(HWS)}, \ldots] \]

Principal difficulty: unconstrained branching, hard to apply heuristics
Planner vs. Theorem Prover

- **Planner**: ask for sequence of actions that makes goal true if executed
- **Theorem prover**: ask whether query sentence is true given KB

The Frame Problem

- **Assumption**: actions have local effects
- **We need** frame axioms for each action and each fluent that does not change as a result of the action
  - example: frame axioms for *move*:
  - If a block is on another block and *move* is not relevant, it will stay the same.
    - **Positive**:
      \[ (\text{On}(x, y, s) \land (x \neq u)) \rightarrow \text{On}(x, y, \text{do(move}(u, v, z), s)) \]
    - **Negative**:
      \[ (\neg\text{On}(x, y, s) \land [(x \neq u) \lor (y \neq z)]) \rightarrow \neg\text{On}(x, y, \text{do(move}(u, v, z), s)) \]
Other Problems in Planning

- **The qualification problem**: qualifying the antecedents for all possible exception. Needs (but impossible!) to enumerate all exceptions
  - \(\neg\)heavy and \(\neg\)glued and \(\neg\)armbroken \(\Rightarrow\) can-move
  - \(\neg\)bird and \(\neg\)cast-in-concrete and \(\neg\)dead... \(\Rightarrow\) flies
  - Solutions: default logics, nonmonotonic logics

- **The ramification problem:**
  - If a robot carries a package, the package will be where the robot is. But what about the frame axiom, when can we infer about the effects of the actions and when we cannot.

- Not everything true can be inferred
  - On(C,F1) remains true but cannot be inferred

Assumptions in Classical Planning

- **Percepts**
  - Perfect perception
  - Complete knowledge agent is omniscient

- **Actions**
  - Instantaneous actions
  - Atomic time
  - No concurrent actions allowed
  - Deterministic actions (effects are completely specified)

- **Environment**
  - Static environment
  - Completely observable environment
  - Agent is the sole cause of change in the world
Assumptions

- Closed World Assumption
  - If ground terms cannot be proved to be true, they can be assumed to be false.

- Domain Closure Assumption
  - All objects in the domain are explicitly named.

- Unique Names Assumption
  - If ground terms cannot be proved to be equal, they can be assumed to be unequal.

STRIPS Representation

- STRIPS is the simplest and the second oldest representation of operators in AI.
- When the initial state is represented by a database of positive facts, STRIPS can be viewed as being simply a way of specifying an update to this database.
Example: “The Blocks World”

- **Domain:**
  - set of cubic blocks sitting on a table

- **Actions:**
  - blocks can be stacked
  - can pick up a block and move it to another position
  - can only pick up one block at a time

- **Goal:**
  - to build a specified stack of blocks

Representing States & Goals

- **STRIPS:**
  - describes states & operators in a restricted language

- **States:**
  - a conjunction of “facts” (ground literals that do not contain variable symbols)

- **Goals:**
  - a conjunction of positive literals
Representing States & Goals

**Initial State:**
- ontable(A) ∧
- ontable(B) ∧
- on(C, B) ∧
- clear(A) ∧
- clear(C) ∧
- handempty

**Goal:**
- ontable(B) ∧
- on(C,B) ∧
- on(A,C) ∧
- clear(A) ∧
- handempty

Graphical Representation: Initial State

- On(C, A)
- Clear(Fl)
- On(A, Fl)
- Clear(B)
- On(B, Fl)
- Clear(C)

Start

T
Graphical Representation: Goal State

Representing Actions

1. Action description
   
   Name: pickup(x)

2. Precondition
   
   Preconditions: ontable(x), clear(x), handempty

3. Effect
   
   Effect: holding(x), ~ontable(x), ~clear(x), ~handempty
Actions in “Blocks World”

- **pickup(x)**
  - picks up block ‘x’ from table

- **putdown(x)**
  - if holding block ‘x’, puts it down on table

- **stack(x,y)**
  - if holding block ‘x’, puts it on top of block ‘y’

- **unstack(x,y)**
  - picks up block ‘x’ that is currently on block ‘y’

An operator is APPLICABLE if all preconditions are satisfied.

Blocks World - Operator

- **Move(x, y, z)**
  - Move block x that is above y to above z
  - **PC**: On(x,y), Clear(x), Clear(z)
  - **D**: Clear(z), On(x, y)
  - **A**: On(x,z), Clear(y), Clear(FL)
Graphical Representation: Operator

**Operator**

\[ \text{Clear(y)} \quad \text{On(x, z)} \quad \text{Clear(Fl)} \]

**PC**

\[ \text{On(x, y)} \quad \text{Clear(x)} \quad \text{Clear(z)} \]

Progression Situation-Space

**Algorithm:**

1. Start from initial state
2. Find all operators whose preconditions are true in the initial state
3. Compute effects of operators to generate successor states
4. Repeat steps 2-3, until a new state satisfies the goal conditions
Regression Situation-Space

**Algorithm:**
1. Start with goal node corresponding to goal to be achieved
2. Choose an operator that will add one of the goals
3. Replace that goal with the operator’s preconditions
4. Repeat steps 2-3 until you have reached the initial state

STRIPS: Goal-Stack Planning

Given a goal stack:
1. Initialize: Push the goal to the stack.
2. If the top of the stack is satisfied in the current state, pop.
3. Otherwise, if the top is a conjunction, push the individual conjuncts to the stack.
4. Otherwise, check if the add-list of any operator can be unified with the top, push the operator and its preconditions to the stack.
5. If the top is an action, pop and execute it.
6. Loop 2-5 till stack is empty.
STRIPS Planning

STRIPS in Action

STATE DESCRIPTION
CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
HANDEMPTY

GOAL STACK
ON(C,B) & ON(A,C)

goal decomposition

---

not promising
(why is this?)

---

production rule

state(x, y)
- P&D: HOLDING(x), CLEAR(y)
- A: HANDEMPTY, ON(x,y), CLEAR(x)

F-rule

Solution = {}

---

STRIPS in Action (Continued)
production rule

Solution = {}
Intelligent Systems & Applications

STRIPS Planning

Substitute \{A/y\}, then apply unstack(C,A) then stack(C,B)

Solution = \{unstack(C,A), stack(C,B)\}

production rule

Solution = \{unstack(C,A), stack(C,B)\}
Intelligent Systems & Applications

STRIPS Planning 18

Copyright (C) 2003 Jane Hsu

goal decomposition

production rule

Solution = \{ \text{unstack}(C,A), \text{stack}(C,B) \}

Apply pickup(A)
And then stack(A,C)

Solution plan = \{ \text{unstack}(C,A), \text{stack}(C,B), \text{pickup}(A), \text{stack}(A,C) \}

Copyright (C) 2003 Jane Hsu