Planning

CHAPTER 11

Outline

◊ Search vs. planning
◊ STRIPS operators
◊ Partial-order planning
function SIMPLE-PLANNING-AGENT( percept) returns an action
static: KB, a knowledge base (includes action descriptions)
p, a plan, initially NoPlan
i, a counter, initially 0, indicating time
local variables: G, a goal
current, a current state description

Tell(KB, MAKE-PERCEPT-SENTENCE( percept, i))
current ≔ STATE-DESCRIPTION(KB, i)
if p = NoPlan then
    G ≔ ASK(KB, MAKE-GOAL-QUERY(i))
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, i))
i ≔ i + 1
return action

Search vs. planning
Consider the task get milk, bananas, and a cordless drill

Standard search algorithms seem to fail miserably:

After-the-fact heuristic/goal test inadequate
Search vs. planning contd.

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

Planning in situation calculus

$PlanResult(p, s)$ is the situation resulting from executing $p$ in $s$

$PlanResult([], s) = s$

$PlanResult([a], s) = PlanResult(p, Result(a, s))$

Initial state $At(Home, S_0) \land \neg Have(Milk, S_0) \land \ldots$

Actions as Successor State axioms

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

Query

$s = PlanResult(p, S_0) \land At(Home, s) \land Have(Milk, s) \land \ldots$

Solution

$p = [Go(Supermarket), Buy(Milk), Buy(Bananas), Go(HWS), \ldots]$

Principal difficulty: unconstrained branching, hard to apply heuristics
**STRIPS operators**

Tidily arranged actions descriptions, restricted language

**ACTION:** Buy(x)

**PRECONDITION:** At(p), Sells(p, x)

**EFFECT:** Have(x)

[Note: this abstracts away many important details!]

Restricted language ⇒ efficient algorithm

Precondition: conjunction of positive literals

Effect: conjunction of literals

\[
\text{At}(p) \land \text{Sells}(p, x) \\
\text{Buy}(x) \\
\text{Have}(x)
\]

---

**State space vs. plan space**

Standard search: node = concrete world state

Planning search: node = partial plan

Defn: open condition is a precondition of a step not yet fulfilled

Operators on partial plans:

- add a link from an existing action to an open condition
- add a step to fulfill an open condition
- order one step wrt another

Gradually move from incomplete/vague plans to complete, correct plans
A plan is complete iff every precondition is achieved
A precondition is achieved iff it is the effect of an earlier step and no possibly intervening step undoes it

Example Partial Order Planner Diagram

Bold arrows – causal links
- steps added to achieve precondition of other steps
Planner protects causal links
- no step that will delete precondition will be placed between causally linked steps
Light arrows - ordering constraints
Section 11.6 – Partial Order Planning Algorithm

**Clobbering and promotion/demotion**

A clobberer is a potentially intervening step that destroys the condition achieved by a causal link. E.g., $Go(Home)$ clobbers $At(HWS)$:

- **Demotion:** put before $Go(HWS)$
- **Promotion:** put after $Buy(Drill)$
Example: Blocks world

"Sussman anomaly" problem

Start State

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

Goal State

\[ \text{Clear}(x) \ \text{On}(x,z) \]
\[ \text{PutOnTable}(x) \]
\[ \neg\text{On}(x,z) \ \text{Clear}(z) \ \text{On}(x,\text{Table}) \]

+ several inequality constraints

Example contd.

START

\[ \text{On}(C,A) \ \text{On}(A,\text{Table}) \ \text{On}(R,\text{Table}) \ \text{On}(C) \]

\[ \text{On}(C,z) \ \text{On}(z) \]
\[ \text{PutOnTable}(C) \]
\[ \text{On}(A,z) \ \text{On}(z) \]
\[ \text{PutOn}(A,B) \]
\[ \text{On}(B,z) \ \text{On}(z) \]
\[ \text{C} \]

FINISH

\[ \text{PutOn}(A,B) \]
\[ \text{clobbers} \ \text{C}(B) \Rightarrow \text{order after} \]
\[ \text{PutOn}(B,C) \]
\[ \text{PutOn}(B,C) \]
\[ \text{clobbers} \ \text{C}(C) \Rightarrow \text{order after} \]
\[ \text{PutOnTable}(C) \]

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

\[ \text{C} \]

\[ \text{B} \]

\[ \text{C} \]
• Robot world
• Actions:
  • Move from place to place within same room (doors between rooms in both rooms)
  • Push object (box) from location x to location y as long as Shakey and box are both at location x and the box is pushable
  • Climb onto a box from the floor
  • Climb down from a box onto floor
  • Turn on light switches from on top of box if box is located at the switch
  • Turn off light switches

STRIPS Actions:

- Go(x,y) \rightarrow At(S,y)
- Push(b,x,y) \rightarrow At(b,y) \land At(s,y)
- Climb(b) \rightarrow On(S,b)
- Down(b) \rightarrow On(S,F) \land At(S,b)
- TurnOn(ls) \rightarrow ~TurnedOn(ls)
- ~TurnedOn(ls) \rightarrow On(S,F) \land At(S,b)