Planning

Chapter 11
Outline

♦ Search vs. planning
♦ STRIPS operators
♦ Partial-order planning
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
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>
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 $\Rightarrow$ efficient algorithm

- Precondition: conjunction of positive literals
- Effect: conjunction of literals

A complete set of STRIPS operators can be translated into a set of successor-state axioms
Partially ordered plans

Partially ordered collection of steps with
  *Start* step has the initial state description as its effect
  *Finish* step has the goal description as its precondition
  causal links from outcome of one step to precondition of another
  temporal ordering between pairs of steps

Open condition = precondition of a step not yet causally linked

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

Start

At(Home)  Sells(HWS,Drill)  Sells(SM,Milk)  Sells(SM,Ban.)

Have(Milk)  At(Home)  Have(Ban.)  Have( Drill)
Example

Start

\[ \text{At(Home)} \quad \text{Sells(HWS,Drill)} \quad \text{Sells(SM,Milk)} \quad \text{Sells(SM,Ban.)} \]

\[ \text{At(HWS)} \quad \text{Sells(HWS,Drill)} \]

Buy(Drill)

\[ \text{At(x)} \]

Go(SM)

\[ \text{At(SM)} \quad \text{Sells(SM,Milk)} \]

Buy(Milk)

\[ \text{Have(Milk)} \quad \text{At(Home)} \quad \text{Have(Ban.)} \quad \text{Have(Drill)} \]

Finish
Start

At(Home)

Go(HWS)

At(HWS)  Sells(HWS,Drill)

Buy(Drill)

At(HWS)

Go(SM)

At(SM)  Sells(SM,Milk)  At(SM)  Sells(SM,Ban.)

Buy(Milk)  Buy(Ban.)

At(SM)

Go(Home)

Have(Milk)  At(Home)  Have(Ban.)  Have(Drill)

Finish
Planning process

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 to remove possible conflicts

Gradually move from incomplete/vague plans to complete, correct plans

Backtrack if an open condition is unachievable or
if a conflict is unresolvable
POP algorithm sketch

function POP(initial, goal, operators) returns plan

    plan ← Make-Minimal-Plan(initial, goal)
    loop do
        if Solution?(plan) then return plan
        S_need, c ← Select-Subgoal(plan)
        Choose-Operator(plan, operators, S_need, c)
        Resolve-Threats(plan)
    end

function Select-Subgoal(plan) returns S_need, c

    pick a plan step S need from Steps(plan)
    with a precondition c that has not been achieved
    return S need, c
**POP algorithm contd.**

```plaintext
procedure CHOOSE-OPERATOR(plan, operators, S_need, c)
    choose a step S_add from operators or STEPS(plan) that has c as an effect
    if there is no such step then fail
    add the causal link S_add \xrightarrow{c} S_need to LINKS(plan)
    add the ordering constraint S_add \prec S_need to ORDERINGS(plan)
    if S_add is a newly added step from operators then
        add S_add to STEPS(plan)
        add Start \prec S_add \prec Finish to ORDERINGS(plan)
```

```plaintext
procedure RESOLVE-THREATS(plan)
    for each S_threat that threatens a link S_i \xrightarrow{c} S_j in LINKS(plan) do
        choose either
        Demotion: Add S_threat \prec S_i to ORDERINGS(plan)
        Promotion: Add S_j \prec S_threat to ORDERINGS(plan)
        if not CONSISTENT(plan) then fail
    end
```
A clobberer is a potentially intervening step that destroys the condition achieved by a causal link. E.g., \( \text{Go(Home)} \) clobbers \( \text{At(Supermarket)} \):

- **Demotion**: put before \( \text{Go(Supermarket)} \)
- **Promotion**: put after \( \text{Buy(Milk)} \)
Properties of POP

Nondeterministic algorithm: backtracks at choice points on failure:
- choice of \( S_{add} \) to achieve \( S_{need} \)
- choice of demotion or promotion for clobberer
- selection of \( S'_{need} \) is irrevocable

POP is sound, complete, and systematic (no repetition)

Extensions for disjunction, universals, negation, conditionals

Can be made efficient with good heuristics derived from problem description

Particularly good for problems with many loosely related subgoals
Example: Blocks world

"Sussman anomaly" problem

Start State

Goal State

\[ \begin{align*}
\text{Clear}(x) \quad \text{On}(x,z) \quad \text{Clear}(y) \\
\text{PutOn}(x,y) \\
\sim \text{On}(x,z) \quad \sim \text{Clear}(y) \\
\text{Clear}(z) \quad \text{On}(x,y)
\end{align*} \]

\[ \begin{align*}
\text{Clear}(x) \quad \text{On}(x,z) \\
\text{PutOnTable}(x) \\
\sim \text{On}(x,z) \quad \text{Clear}(z) \quad \text{On}(x,\text{Table})
\end{align*} \]

+ several inequality constraints
Example contd.

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

On(A,B) On(B,C)

FINISH
Example contd.

On(C,A)  On(A,Table)  Cl(B)  On(B,Table)  Cl(C)

Cl(B)  On(B,z)  Cl(C)

PutOn(B,C)

On(A,B)  On(B,C)

FINISH
Example contd.

START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

PutOn(A,B)

On(A,z) Cl(B)

PutOn(B,C)

On(B,z) Cl(C)

FINISH

PutOn(A,B)

On(A,B) Cl(B)

On(A,B)

On(B,C) Cl(C)

PutOn(B,C)

Cl(A)

On(A,z)

Chapter 11 18
Example contd.

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

On(C,z) Cl(C)

PutOnTable(C)

Cl(A) On(A,z) Cl(B)

PutOn(A,B)

Cl(B) On(B,z) Cl(C)

PutOn(B,C)

On(A,B) On(B,C)

FINISH