Planning
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
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></th>
<th>Search</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>Lisp data structures</td>
<td>Logical sentences</td>
</tr>
<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|p], s) = PlanResult(p, Result(a, s))

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

Actions as Successor State axioms

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

Query

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

Solution

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

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

Tidily arranged actions descriptions, restricted language

**ACTION:** \( \text{Buy}(x) \)

**PRECONDITION:** \( \text{At}(p), \text{Sells}(p, x) \)

**EFFECT:** \( \text{Have}(x) \)

[Note: this abstracts away many important details!]

Restricted language \( \Rightarrow \) efficient algorithm
- Precondition: conjunction of positive literals
- Effect: conjunction of literals

\[
\begin{array}{c}
\text{At}(p) \quad \text{Sells}(p, x) \\
\hline
\text{Buy}(x) \\
\hline
\text{Have}(x)
\end{array}
\]
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
**POP algorithm sketch**

```plaintext
function POP(initial, goal, operators) returns plan

    plan ← MAKE-MINIMAL-PAN(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
```

ALMA Slides ©Stuart Russell and Peter Norvig, 1998
POP algorithm contd.

procedure \textsc{Choose-Operator}(plan, operators, S_{need}, c)

choose a step \( S_{add} \) from operators or \textsc{Steps}(plan) that has \( c \) as an effect
if there is no such step then fail
add the causal link \( S_{add} \rightarrow S_{need} \) to \textsc{Links}(plan)
add the ordering constraint \( S_{add} < S_{need} \) to \textsc{Orderings}(plan)
if \( S_{add} \) is a newly added step from operators then
    add \( S_{add} \) to \textsc{Steps}(plan)
    add Start < \( S_{add} \) < Finish to \textsc{Orderings}(plan)

procedure \textsc{Resolve-Threats}(plan)

for each \( S_{\text{threat}} \) that threatens a link \( S_i \rightarrow S_j \) in \textsc{Links}(plan) do
    choose either
    \textit{Demotion}: Add \( S_{\text{threat}} < S_i \) to \textsc{Orderings}(plan)
    \textit{Promotion}: Add \( S_j < S_{\text{threat}} \) to \textsc{Orderings}(plan)
if not \textsc{Consistent}(plan) then fail
end

POP is sound, complete, and \underline{systematic} (no repetition)

Extensions for disjunction, universals, negation, conditionals
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 \( Buy(Drill) \)
"Sussman anomaly" problem

Start State

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

\[ \text{PutOn}(x,y) \]

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

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

Goal State

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

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

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

+ several inequality constraints
Example contd.

START

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

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

FINISH
Example contd.

\[
\begin{align*}
On(C,A) & \quad On(A, Table) & Cl(B) & \quad On(B, Table) & Cl(C) \\
Cl(B) & \quad On(B, z) & Cl(C) & \quad PutOn(B, C) \\
On(A, B) & \quad On(B, C) & \quad FINISH
\end{align*}
\]
Example contd.

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

PutOn(A,B)

Finishes with:

FINISH
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

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

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