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>
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)

Finish
Example

Start

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

At(HWS)  Sells(HWS,Drill)

Buy(Drill)

At(x)

Go(SM)

At(SM)  Sells(SM,Milk)

Buy(Milk)

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

Finish
Example

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 \textbf{Choose-Operator}(plan, operators, S\textsubscript{need}, c)

\begin{itemize}
  \item \textbf{choose} a step \( S\textsubscript{add} \) from \textit{operators} or \textit{Steps(plan)} that has \( c \) as an effect
  \item if there is no such step then fail
\end{itemize}

\begin{itemize}
  \item add the causal link \( S\textsubscript{add} \rightarrowrightarrow S\textsubscript{need} \) to \textit{Links(plan)}
  \item add the ordering constraint \( S\textsubscript{add} \prec S\textsubscript{need} \) to \textit{Orderings(plan)}
  \item if \( S\textsubscript{add} \) is a newly added step from \textit{operators} then
    \begin{itemize}
      \item add \( S\textsubscript{add} \) to \textit{Steps(plan)}
      \item add \( \text{Start} \prec S\textsubscript{add} \prec \text{Finish} \) to \textit{Orderings(plan)}
    \end{itemize}
\end{itemize}

procedure \textbf{Resolve-Threats}(plan)

\begin{itemize}
  \item for each \( S\textsubscript{threat} \) that threatens a link \( S\textsubscript{i} \rightarrowrightarrow S\textsubscript{j} \) in \textit{Links(plan)} do
    \begin{itemize}
      \item \textbf{choose} either
        \begin{itemize}
          \item \textit{Demotion}: Add \( S\textsubscript{threat} \prec S\textsubscript{i} \) to \textit{Orderings(plan)}
          \item \textit{Promotion}: Add \( S\textsubscript{j} \prec S\textsubscript{threat} \) to \textit{Orderings(plan)}
        \end{itemize}
      \item if not \textit{Consistent(plan)} then fail
    \end{itemize}
\end{itemize}
```

Chapter 11 12
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 $At(Supermarket)$:

Demotion: put before $Go(Supermarket)$

Promotion: put after $Buy(Milk)$
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

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

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.

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

\[
\text{On}(A,B) \quad \text{On}(B,C)
\]
Example contd.

\[
\begin{align*}
&\text{START} \\
&\text{On(C,A) On(A,Table) Cli(B) On(B,Table) Cli(C)} \\
&\text{Cl(B) On(B,z) Cli(C)} \\
&P\text{utOn(B,C)} \\
&\text{On(A,B) On(B,C)} \\
&\text{FINISH}
\end{align*}
\]
On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

PutOn(A,B)

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

PutOn(B,C)

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

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

FINISH

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

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

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