Planning and Partial-Order Planning

Sections 11.1-11.3
Outline

- Search vs. planning
- STRIPS operators
- Partial-order planning

Additional reference used for the slides:
Consider the task *get milk, bananas, and a cordless drill*
Standard search algorithms seem to fail miserably:

Search vs. planning

After-the-fact heuristic/goal test inadequate
## Search vs. planning (cont’d)

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

Tidily arranged actions descriptions, restricted language

\[
\begin{align*}
\text{At}(p) & \quad \text{Sells}(p, x) \\
\text{BUY} (x)
\end{align*}
\]

\[
\begin{align*}
\text{Have}(x)
\end{align*}
\]

**ACTION:** \textit{Buy}(x)  \\
**PRECONDITION:** \textit{At}(p), \textit{Sells}(p, x)  \\
**EFFECT:** \textit{Have}(x)
ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p, x)$
EFFECT: $Have(x)$

[Note: this abstracts away many important details!]

Restricted language $\implies$ 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
A partially ordered plan is a 5-tuple \((A, O, C, OC, UL)\)

- \(A\) is the set of actions that make up the plan. They are partially ordered.

- \(O\) is a set of ordering constraints of the form \(A \prec B\). It means \(A\) comes before \(B\).

- \(C\) is the set of causal links in the form \((A, p, B)\) where \(A\) is the supplier action, where \(B\) is the consumer action, and \(p\) is the condition supplied. It is read as “\(A\) achieves \(p\) for \(B\)”.
A partially ordered plan is a 5-tuple \((A, O, C, OC, UL)\)

- **OC** is a set of open conditions, i.e., conditions that are not yet supported by causal links. It is of the form \(p \text{ for } A\) where \(p\) is a condition and \(A\) is an action.

- **UL** is a set of unsafe links, i.e., causal links whose conditions might be undone by other actions.
A plan is *complete* iff every precondition is achieved, and there are no unsafe links. A precondition is *achieved* iff it is the effect of an earlier step and no *possibly intervening* step undoes it.

In other words, a plan is complete when $OC \cup UL = \emptyset$.

$OC \cup UL$ is referred to as the *flaws* in a plan.

When a causal link is established, the corresponding condition is said to be *closed*. 
Example

START

CleanLeftSock       CleanRightSock

FINISH

LeftShoeOn          RightShoeOn

OC =
LeftShoeOn for FINISH
RightShoeOn for FINISH
START

CleanLeftSock

LeftSockOn

LEFT SHOE

LeftShoeOn

FINISH

CleanRightSock

RightShoeOn

OC=
RightShoeOn for FINISH
LeftSockOn for LEFTSHOE
Example (cont’d)

- START
  - CleanLeftSock
    - LEFT SOCK
      - LeftSockOn
        - LEFT SHOE
          - LeftShoeOn
            - FINISH
  - CleanRightSock
    - RightShoeOn

OC = CleanLeftSock for LEFTSOCK RightShoeOn for FINISH
Example (cont’d)

START

CleanLeftSock

LEFT SOCK

LeftSockOn

LEFT SHOE

LeftShoeOn

FINISH

CleanRightSock

OC = RightShoeOn for FINISH
Example (cont’d)

```
START

CleanLeftSock

LEFT SOCK

LeftSockOn

LEFT SHOE

LeftShoeOn

FINISH

CleanRightSock

RIGHT SHOE

RightShoeOn

RightSockOn

OC = RightSockOn for RIGHTSHOE
```
Example (cont’d)

OC = CleanRightSock for RIGHTSOCK

START

CleanLeftSock

LEFT SOCK

CleanRightSock

RIGHT SOCK

LeftSockOn

LEFT SHOE

RightSockOn

RIGHT SHOE

LeftShoeOn

FINISH

RightShoeOn
Operators on partial plans:
  close open conditions:
    add a link from an existing action to an open condition
    add a step to fulfill an open condition
  resolve threats:
    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
function Tree-Search (problem, fringe)
returns a solution, or failure

fringe ← Insert(Make-Node(Initial-State [problem]), fringe)
loop do
  if EMPTY?(fringe) then return failure
  node ← Remove-First(fringe)
  if Goal-Test[problem] applied to State[node] succeeds
     then return Solution(node)
  fringe ← Insert-All(Expand(node, problem), fringe)
The initial state, goal state and the operators are given. The planner converts them to required structures.

**Initial state:**

`MAKE-MINIMAL-PLAN (initial,goal)`

**Goal-Test:**

`SOLUTION?(plan)`

`SOLUTION?` returns true iff OC and UL are both empty.
The **successors function** could either close an open condition or resolve a threat.

**function** SUCCESSORS *(plan)*  
**returns** a set of partially ordered plans

\[
\text{flaw-type} \leftarrow \text{SELECT-FLAW-TYPE} \ (plan) \\
\text{if flaw-type is an open condition then} \\
\quad S_{\text{need}}, c \leftarrow \text{SELECT-SUBGOAL} \ (plan) \\
\quad \text{return} \ \text{CLOSE-CONDITION} \ (plan, \text{operators}, S_{\text{need}}, c) \\
\text{if flaw-type is a threat then} \\
\quad S_{\text{threat}}, S_i, c, S_j \leftarrow \text{SELECT-THREAT} \ (plan) \\
\quad \text{return} \ \text{RESOLVE-THREAT} \ (plan, S_{\text{threat}}, S_i, c, S_j)
\]
function CLOSE-CONDITION (plan, operators, S_{need}, c)
returns a set of partially ordered plans

plans ← ∅
for each S_{add} from operators or STEPS(plan) that has c has an effect do
    new-plan ← plan
    if S_{add} is a newly added step from operators then
        add S_{add} to STEPS (new-plan)
        add START \prec S_{add} \prec FINISH to ORDERINGS (new-plan)
        add the causal link \langle S_{add}, c, S_{need} \rangle to LINKS (new-plan)
        add the ordering constraint \langle S_{add} \prec S_{need} \rangle to
            ORDERINGS (new-plan)
        add new-plan to plans
    end
return new-plans
function RESOLVE-THREAT (plan, $S_{\text{threat}}, S_i, c, S_j$)
returns a set of partially ordered plans

    plans ← ∅

//Demotion:
new-plan ← plan
add the ordering constraint ($S_{\text{threat}} < S_i$) to ORDERINGS (new-plan)
if new-plan is consistent then
    add new-plan to plans

//Promotion:
new-plan ← plan
add the ordering constraint ($S_j < S_{\text{threat}}$) to ORDERINGS (new-plan)
if new-plan is consistent then
    add new-plan to plans

return new-plans
Shopping example

The operators are:
- GO (?x, ?y)
  - preconditions: at(?x)
  - effects: ~at(?x), at(?y)
- BUY (?s, ?i)
  - preconditions: at(?s), ~bought(~i)
  - effects: bought(?i)

Agenda:
- open subgoals:
  - bought(A) for g
  - bought(B) for g
  - at(H) for g

The subgoals that are currently open are italicized.

new Agenda:
- open subgoals:
  - bought(A) for g
  - bought(B) for g
  - at(J) for 1

add a go(J,H) action
for at(H) for g

add a causal link from
INIT for at(H) for g

Ch. 11a – p.25/38–
Shopping example (cont’d)

New agenda:
open subgoals:
bought(A) for g
bought(B) for g
at(H) for 2

Supply at(H) for 2 from INIT–0

Add a go(H, J) action (2)
Shopping example (cont’d)

new agenda: open subgoals:
- bought(B) for g
- bought(A) for 3
- at(J) for 3

Support at(J) from GO(H,J)−2

support ~bought(A) from INIT−0
new agenda: open subgoals:
- bought(B) for g
- at(J) for 3

add BUY(J,A) (3)

Ch. 11a – p.27/38-
Shopping example (cont’d)

new agenda:
open subgoals:
- bought(B) for g
- bought(A) for 3

new agenda:
open subgoals:
- bought(B) for g
- bought(A) for 3

HAVEN’T CONSIDERED THE THREATS YET!
Now, the solution is a possible ordering of this plan. Those are:

2 3 4 1
2 3 1 4
2 4 3 1
2 4 1 3
2 1 3 4
2 1 4 3

It should not be possible to order GO(j,H) before any of the BUY actions.
This is a correct partially ordered plan.
It is complete.
The possible total orders are:
2 3 4 1
2 4 3 1

The agent has to go to Jim’s first.
It order of getting the items does not matter.
The it has to go home.
Another shopping example

START

At(H)  Sells(Hws,Drill)  Sells(Sm,Milk)  Sells(Sm,Ban)

Have(Milk)  at(H)  Have(ban)  Have(Drill)

FINISH
Another shopping example (cont’d)

START

At(H)

Sells(Hws,Drill) Sells(Sm,Milk) Sells(Sm,Ban)

GO(H,Hws)

At(Hws)

BUY(drill)

Have(Milk)

at(H)

Have(ban)

Have(Drill)

FINISH
Another shopping example (cont’d)

START

At(H)

Sells(Hws,Drill) Sells(Sm,Milk) Sells(Sm,Ban)

GO(H,Hws)

At(Hws)

At(Hws)

GO(Hws,Sm)

At(Sm)

At(Sm)

BUY(drill)

BUY(Milk)

BUY(ban)

Have(Milk)

at(H)

Have(ban)

Have(Drill)

FINISH
Another shopping example (cont’d)

START

At(H)

Sells(Hws,Drill) Sells(Sm,Milk) Sells(Sm,Ban)

GO(H,Hws)

At(Hws)

At(Sm)

BUY(Milk)

At(Sm)

BUY(ban)

At(Sm)

GO(Sm,H)

Have(Milk)

at(H)

FINISH

At(Sm)

Have(ban)

Have(Drill)

At(Sm)

to BUY(Milk)

to BUY(Ban)
Another shopping example (cont’d)

START

At(H)

Sells(Hws,Drill) Sells(Sm,Milk) Sells(Sm,Ban)

GO(H,Hws)

At(Hws)

GO(Hws,Sm)

At(Sm)

BUY(Milk)

At(Sm)

BUY(ban)

GO(Sm,H)

Have(Milk)

at(H)

FINISH

Have(ban)

Have(Drill)
A **threatening step** is a potentially intervening step that destroys the condition achieved by a causal link. E.g., $\text{GO}($Sm,H$)$ threatens $\text{At}(\text{Sm})$

Demotion: put before $\text{GO}(\text{Hws},\text{Sm})$

Promotion: put after $\text{GO}(\text{Hws},\text{Sm})$
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 threat resolution
  - 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
Heuristics for POP

- Which plan to select?
- Which flaw to choose?
- More after planning graphs