1 Outline of Today’s Class

2 Planning
   - Search vs. Planning
   - STRIPS Operators
   - Partial-order Planning

3 Planning and Acting
   - The Real World
   - Conditional Planning
   - Monitoring and Replacing
Search vs. Planning

Consider the task *get milk, bananas, and a cordless drill*

Standard search algorithms seem to fail miserably:

![Diagram showing possible actions and decisions leading to the task completion]

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></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>
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 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

- \text{At(Home)}

Go(HWS)

- \text{At(HWS)}

Buy(Drill)

- \text{At(HWS)}

Go(SM)

- \text{At(SM)}

Buy(Milk)

- \text{At(SM)}

Buy(Ban.)

- \text{At(SM)}

Go(Home)

- \text{At(Home)}

Have(Milk)

- \text{At(Home)}

Have(Ban.)

- \text{Have(Ban.)}

Have(Drill)

Finish
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
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 $S_{need}$, $c$
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} ~> S_{need}* to *Links*(plan)
add the ordering constraint *S_{add} < S_{need}* to *Orderings*(plan)
if *S_{add}* is a newly added step from *operators* then
  add *S_{add}* to *Steps*(plan)
  add *Start < S_{add} < Finish* to *Orderings*(plan)

procedure **Resolve-Threats**(*plan*)

for each *S_{threat}* that threatens a link *S_i ~> S_j* in *Links*(plan) do
  choose either
    Demotion: Add *S_{threat} < S_i* to *Orderings*(plan)
    Promotion: Add *S_j < 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., \textit{Go(Home)} clobbers \textit{At(Supermarket)}:

Demotion: put before \textit{Go(Supermarket)}

Promotion: put after \textit{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

Clear(x) On(x,z) Clear(y)
PutOn(x,y)
~On(x,z) ~Clear(y)
Clear(z) On(x,y)

Clear(x) On(x,z)
PutOnTable(x)
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints
Example Continued

START

$\text{On}(C,A) \ \text{On}(A,\text{Table}) \ \text{Cl}(B) \ \text{On}(B,\text{Table}) \ \text{Cl}(C)$

$\text{On}(A,B) \quad \text{On}(B,C)$

FINISH
Example Continued

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

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

\[
\text{PutOn(B,C)}
\]

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

\[
\text{FINISH}
\]
Example Continued

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

$\text{PutOn}(A, B)$ clobbers $\text{Cl}(B)$
$\Rightarrow$ order after $\text{PutOn}(B, C)$
Example Continued

START

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)
The Real World

START

~Flat(Spare) Intact(Spare) Off(Spare)
On(Tire1) Flat(Tire1)

On(x) ~Flat(x)

FINISH

On(x)

Remove(x)

Off(x) ClearHub

Puton(x)

Off(x) ClearHub

On(x) ~ClearHub

Intact(x) Flat(x)

Inflate(x)

~Flat(x)
Things Go Wrong

**Incomplete information**

Unknown preconditions, e.g., $Intact(Spare)$?

Disjunctive effects, e.g., $Inflate(x)$ causes

\[ Inflated(x) \lor \text{SlowHiss}(x) \lor \text{Burst}(x) \lor \text{BrokenPump} \lor \ldots \]

**Incorrect information**

Current state incorrect, e.g., spare NOT intact

Missing/incorrect postconditions in operators

**Qualification problem:**

Can never finish listing all the required preconditions and possible conditional outcomes of actions
Solutions

Conformant or sensorless planning

Devise a plan that works regardless of state or outcome

Such plans may not exist

Conditional planning

Plan to obtain information (observation actions)

Subplan for each contingency, e.g.,

\[
\text{[Check}(\text{Tire1}), \text{if } \text{Intact}(\text{Tire1}) \text{ then Infl}ate(\text{Tire1}) \text{ else CallAAA}
\]

Expensive because it plans for many unlikely cases

Monitoring/Replanning

Assume normal states, outcomes
Conformant Planning

Search in space of belief states (sets of possible actual states)
Conditional Planning

If the world is nondeterministic or partially observable

then percepts usually provide information,

i.e., split up the belief state
Conditional plans check (any consequence of KB +) percept
[... if C then Plan_A else Plan_B, ...]

Execution: check C against current KB, execute “then” or “else”

Need some plan for every possible percept
(Cf. game playing: some response for every opponent move)
(Cf. backward chaining: some rule such that every premise satisfied

AND–OR tree search (very similar to backward chaining algorithm)
Double Murphy: sucking or arriving may dirty a clean square
Example

Triple Murphy: also sometimes stays put instead of moving

\[ L_1 : \text{Left, if } \text{AtR then } L_1 \text{ else [if CleanL then } [] \text{ else Suck]} \]

or \[ \text{while } \text{AtR do [Left, if CleanL then } [] \text{ else Suck]} \]

“Infinite loop” but will eventually work unless action always fails
“Failure” = preconditions of remaining plan not met
Preconditions of remaining plan

= all preconditions of remaining steps not achieved by remaining steps

= all causal links crossing current time point
On failure, resume POP to achieve open conditions from current state
IPEM (Integrated Planning, Execution, and Monitoring):

keep updating Start to match current state

links from actions replaced by links from Start when done
Example

Start

At(Home)

Go(HWS)

At(HWS)

Sells(HWS,Drill)

Buy(Drill)

Go(SM)

At(SM)

Sells(SM,Milk)

Buy(Milk)

Go(Home)

Have(Milk)

At(Home)

Sells(SM,Ban.)

Sells(SM,Milk)

Buy(Ban.)

At(SM)

Go(HWS)

Have(Ban.)

Have(Drill)

Finish
Example
Example
Emergent Behavior

PRECONDITIONS

START

Color(Chair,Blue) ~Have(Red)

Get(Red)

Have(Red)

Paint(Red)

Color(Chair,Red)

FINISH

FAILURE RESPONSE

Have(Red)

Fetch more red
“Loop until success” behavior *emerges* from interaction between monitor/replan agent design and uncooperative environment.
“Loop until success” behavior *emerges* from interaction between monitor/replan agent design and uncooperative environment.