today’s topics:

- logic-based agents (see notes from last time)
- planning
What is Planning?

- Key problem facing agent is *deciding what to do*.
- We want agents to be *taskable*: give them *goals* to achieve, have them decide for themselves how to achieve them.
- Basic idea is to give an agent:
  - representation of goal to achieve;
  - knowledge about what actions it can perform; and
  - knowledge about state of the world;
and to have it generate a *plan* to achieve the goal.
- Essentially, this is *
  
  *automatic programming.*
Plan to achieve goal

Planner

goal
state of environment
possible actions
• Question: How do we represent…
  – goal to be achieved;
  – state of environment;
  – actions available to agent;
  – plan itself.

• We show how all this can be done in first-order logic…
• We’ll illustrate the techniques with reference to the *blocks world*.
• Contains a robot arm, 3 blocks (A, B and C) of equal size, and a table-top.
• Initial state:
To represent this environment, need an ontology.

- $On(x, y)$: obj $x$ on top of obj $y$
- $OnTable(x)$: obj $x$ is on the table
- $Clear(x)$: nothing is on top of obj $x$
- $Holding(x)$: arm is holding $x$
• Here is a first-order logic representation of the blocks world described above:

\begin{align*}
  &\text{Clear}(A) \\
  &\text{On}(A, B) \\
  &\text{OnTable}(B) \\
  &\text{OnTable}(C) \\
  &\text{Clear}(C)
\end{align*}

• Use the \textit{closed world assumption}: anything not stated is assumed to be \textit{false}.
• A goal is represented as a first-order logic formula.

• Here is a goal:

\[ \text{OnTable}(A) \land \text{OnTable}(B) \land \text{OnTable}(C) \]

• Which corresponds to the state:

A B C

• Actions are represented using a technique that was developed in the STRIPS planner.
• Each action has:
  – a name
    which may have arguments;
  – a pre-condition list
    list of facts which must be true for action to be executed;
  – a delete list
    list of facts that are no longer true after action is performed;
  – an add list
    list of facts made true by executing the action.

Each of these may contain variables.
Example 1:
The *stack* action occurs when the robot arm places the object $x$ it is holding is placed on top of object $y$.

\[
\begin{align*}
\text{Stack}(x, y) \\
\text{pre} & \quad \text{Clear}(y) \land \text{Holding}(x) \\
\text{del} & \quad \text{Clear}(y) \land \text{Holding}(x) \\
\text{add} & \quad \text{ArmEmpty} \land \text{On}(x, y)
\end{align*}
\]
• Example 2:

The *unstack* action occurs when the robot arm picks an object $x$ up from on top of another object $y$.

$$UnStack(x, y)$$

**pre**  $On(x, y) \land Clear(x) \land ArmEmpty$

**del**  $On(x, y) \land ArmEmpty$

**add**  $Holding(x) \land Clear(y)$

Stack and UnStack are *inverses* of one-another.
• Example 3:
  The *pickup* action occurs when the arm picks up an object $x$ from the table.

  \[
  Pickup(x)
  \]
  \[
  \text{pre } Clear(x) \land OnTable(x) \land ArmEmpty
  \]
  \[
  \text{del } OnTable(x) \land ArmEmpty
  \]
  \[
  \text{add } Holding(x)
  \]

• Example 4:
  The *putdown* action occurs when the arm places the object $x$ onto the table.

  \[
  PutDown(x)
  \]
  \[
  \text{pre } Holding(x)
  \]
  \[
  \text{del } Holding(x)
  \]
  \[
  \text{add } Holding(x) \land ArmEmpty
  \]
• What is a plan?
A sequence (list) of actions, with variables replaced by constants.
• So, to get from:
• We need the set of actions:

\[ \text{Unstack}(A) \]
\[ \text{Putdown}(A) \]
\[ \text{Pickup}(B) \]
\[ \text{Stack}(B, C) \]
\[ \text{Pickup}(A) \]
\[ \text{Stack}(A, B) \]
In “real life”, plans contain conditionals (IF .. THEN...) and loops (WHILE... DO...), but most simple planners cannot handle such constructs — they construct linear plans.

Simplest approach to planning: means-ends analysis.

Involves backward chaining from goal to original state.

Start by finding an action that has goal as post-condition. Assume this is the last action in plan.

Then figure out what the previous state would have been. Try to find action that has this state as post-condition.

Recurse until we end up (hopefully!) in original state.
function $plan$(
    $d$ : WorldDesc, // initial env state
    $g$ : Goal, // goal to be achieved
    $p$ : Plan, // plan so far
    $A$ : set of actions // actions available)
1. if $d \models g$ then
2.     return $p$
3. else
4.     choose $a$ in $A$ such that
5.     \begin{align*}
    \text{add}(a) \models g & \text{ and } \\
    \text{del}(a) \not\models g
    \end{align*}
6.     set $g = \text{pre}(a)$
7.     append $a$ to $p$
8.     return $plan(d, g, p, A)$
• How does this work on the previous example?
• This algorithm not guaranteed to find the plan…
• … but it is sound: If it finds the plan is correct.
• Some problems:
  – negative goals;
  – maintenance goals;
  – conditionals & loops;
  – exponential search space;
  – logical consequence tests;
The Frame Problem

- A general problem with representing properties of actions:
  How do we know exactly what changes as the result of performing an action?
  If I pick up a block, does my hair colour stay the same?

- One solution is to write frame axioms.
  Here is a frame axiom, which states that SP’s hair colour is the same in all the situations $s'$ that result from performing $Pickup(x)$ in situation $s$ as it is in $s$.

$$\forall s, s'. Result(SP, Pickup(x), s) = s' \Rightarrow HCol(SP, s) = HCol(SP, s')$$
• Stating frame axioms in this way is unfeasible for real problems.
• (Think of all the things that we would have to state in order to cover all the possible frame axioms).
• STRIPS solves this problem by assuming that everything not explicitly stated to have changed remains unchanged.
• The price we pay for this is that we lose the advantages of using logic:
  – Semantics goes out of the window
• However, more recent work has effectively solved the frame problem (using clever second-order approaches).
Sussman’s Anomaly

- Consider we have the following initial state and goal state:

\[
\begin{array}{c}
\text{B} \\
\text{C}
\end{array}
\quad \rightarrow \quad
\begin{array}{c}
\text{A} \\
\text{B} \\
\text{C}
\end{array}
\]

- What operations will be in the plan?
• Clearly we need to *Stack* B on C at some point, and we also need to *Unstack* A from C and *Stack* it on B.

• Which operation goes first?

• Obviously we need to do the *Unstack* first, and the *Stack* B on C, but the planner has no way of knowing this.

• It also has no way of “undoing” a partial plan if it leads into a dead end.

• So if it chooses to *Stack*(A, C) after the *Unstack*, it is sunk.

• This is a big problem with linear planners

• How could we modify our planning algorithm?
• Modify the middle of the algorithm to be:

1. if \( d \models g \) then
2. return \( p \)
3. else
4. choose \( a \) in \( A \) such that
5. \( \text{add}(a) \models g \) and
6. \( \text{del}(a) \not\models g \)
6a. \( \text{no\_clobber}(\text{add}(a), \text{del}(a), \text{rest\_of\_plan}) \)
7. set \( g = \text{pre}(a) \)
8. append \( a \) to \( p \)
9. return \( \text{plan}(d, g, p, A) \)

• But how can we do this?
• We will give an answer in the next lecture.
Summary

- This lecture has looked at planning.
- We looked mainly at a logical view of planning, using STRIPS operators.
- We also discussed the frame problem, and Sussman’s anomaly.
- Sussman’s anomaly motivated some thoughts about partial-order planning.
- We will cover partial order planning in more detail in the next lecture.