PLANNING

1 What is Planning?

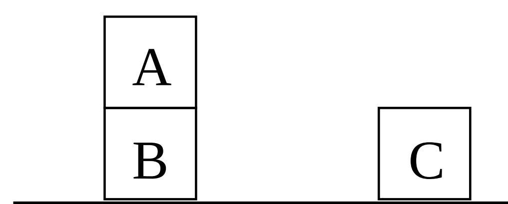
- Key problem facing *agent* is *deciding what to do*.
- We want agents to be *taskable*: give them *goals* to achieve, have them decide for themselves how to achieve them.
- Basic idea is to give an agent:
  - representation of goal to achieve;
  - knowledge about what actions it can perform; and
 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

  and to have it generate a *plan* to achieve the goal.
- Essentially, this is

  automatic programming.

- Question: How do we *represent*...
  - goal to be achieved;
  - state of environment;
  - actions available to agent;
  - plan itself.
- 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 FOL representation of the blocks world described above:

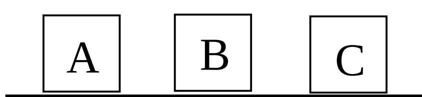
$$Clear(A)$$
$$On(A, B)$$
$$OnTable(B)$$
$$OnTable(C)$$
$$Clear(C)$$

- Use the *closed world assumption*: anything not stated is assumed to be *false*.

- A *goal* is represented as a FOL formula.
- Here is a goal:

$$OnTable(A)OnTable(B)OnTable(C)$$

- Which corresponds to the state:



- *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.

$$\begin{array}{ll} & Stack(x,y) \\ \text{pre} & Clear(y) \wedge Holding(x) \\ \text{del} & Clear(y) \wedge Holding(x) \\ \text{add} & ArmEmpty \wedge On(x,y) \end{array}$$

- Example 2:

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

$$\begin{array}{ll} & UnStack(x,y) \\ \text{pre} & On(x,y) \wedge Clear(x) \wedge ArmEmpty \\ \text{del} & On(x,y) \wedge ArmEmpty \\ \text{add} & Holding(x) \wedge Clear(y) \end{array}$$

  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.

$$\begin{array}{ll} & Pickup(x) \\ \text{pre} & Clear(x) \wedge OnTable(x) \wedge ArmEmpty \\ \text{del} & OnTable(x) \wedge ArmEmpty \\ \text{add} & Holding(x) \end{array}$$

- Example 4:

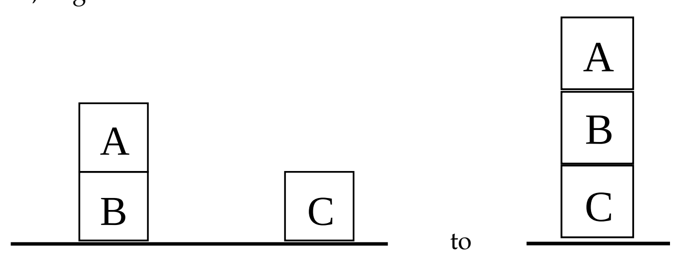
  The *putdown* action occurs when the arm places the object x onto the table.

$$\begin{array}{ll} & PutDown(x) \\ \text{pre} & Holding(x) \\ \text{del} & Holding(x) \\ \text{add} & Holding(x) \wedge ArmEmpty \end{array}$$

- What is a plan?

  A sequence (list) of actions, with variables replaced by constants.

- So, to get from:



- We need the set of actions:

$$\begin{array}{c} Unstack(A) \\ Putdown(A) \\ Pickup(B) \\ Stack(B, C) \\ Pickup(A) \\ Stack(A, B) \end{array}$$

- 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.

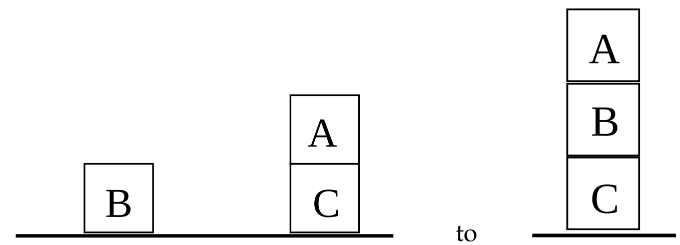

```
function plan(
    d : WorldDesc,        // initial env state
    g : Goal,             // goal to be achieved
    p : Plan,             // plan so far
    A : set of actions    // actions available)
if d ⊨ g then
        return p
else
        choose a in A such that
                add(a) ⊨ g and
                del(a) ⊭ g
        set g = pre(a)
        append a to p
        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;

Sussman's Anomaly

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



- 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 |= g then
2.        return p
3.   else
4.        choose a in A such that
5.             add(a) |= g and
6.             del(a) |≠ g
6a.            no_clobber(add(a), del(a), rest_of_plan)
7.        set g = pre(a)
8.        append a to p
9.        return plan(d, g, p, A)
```

Summary

- This lecture briefly introduced simple (classical planning).
- It showed how to use means-ends analysis to create a linear plan when the world is represented using STRIPS operators.
- We also talked about some of the problems with this approach.
- In particular we talked about Sussman's anomaly, which leads us to *partial order planning*, the topic of the next lecture.