Robotics and Autonomous Systems

Lecture 17: Practical reasoning agents

Richard Williams

Department of Computer Science
University of Liverpool

Today

- In the previous lecture we looked at:
  - Basic description of an autonomous agent
  - How the intentional stance could be used to describe an agent.
- In this lecture we'll look at how the intentional stance can be used to program agents.
- This will use the basic intentional notions:
  - Belief
  - Desire
  - Intention

  that we saw last time.

Pro-active behaviour

- We said: An intelligent agent is a computer system capable of flexible autonomous action in some environment.
- Where by flexible, we mean:
  - reactive;
  - pro-active;
  - social.
- This is where we deal with the "proactive" bit, showing how we can progran agents to have goal-directed behaviour.

What is Practical Reasoning?

- Practical reasoning is reasoning directed towards actions — the process of figuring out what to do:

  Practical reasoning is a matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes.

Michael Bratman

- Distinguish practical reasoning from theoretical reasoning. Theoretical reasoning is directed towards beliefs.

The Components of Practical Reasoning

- Human practical reasoning consists of two activities:
  - Deliberation
    deciding what state of affairs we want to achieve
    — the outputs of deliberation are intentions;
  - Means-ends reasoning
    deciding how to achieve these states of affairs
    — the outputs of means-ends reasoning are plans.
- Intentions are a key part of this.
- The interplay between beliefs, desires and intentions defines how the model works.

Intentions in Practical Reasoning

1. Intentions pose problems for agents, who need to determine ways of achieving them.

If I have an intention to ϕ, you would expect me to devote resources to deciding how to bring about ϕ.

Intentions in Practical Reasoning

2. Intentions provide a “filter” for adopting other intentions, which must not conflict.
   If I have an intention to ϕ, you would not expect me to adopt an intention ψ that was incompatible with ϕ.
3. Agents track the success of their intentions, and are inclined to try again if their attempts fail.
   If an agent’s first attempt to achieve ϕ fails, then all other things being equal, it will try an alternative plan to achieve ϕ.

Intentions in Practical Reasoning

4. Agents believe their intentions are possible.
   That is, they believe there is at least some way that the intentions could be brought about.
5. Agents do not believe they will not bring about their intentions.
   It would not be rational of me to adopt an intention to ϕ if I believed I would fail with ϕ.
6. Under certain circumstances, agents believe they will bring about their intentions.
   If I intend ϕ, then I believe that under “normal circumstances” I will succeed with ϕ.

Intentions in Practical Reasoning

7. Agents need not intend all the expected side effects of their intentions.
   If I believe $\phi \Rightarrow \psi$ and I intend that ϕ, I do not necessarily intend ψ also. (Intentions are not closed under implication.)
   This last problem is known as the side effect or package deal problem. I may believe that going to the dentist involves pain, and I may also intend to go to the dentist — but this does not imply that I intend to suffer pain!

Intentions = Desires + Commitment

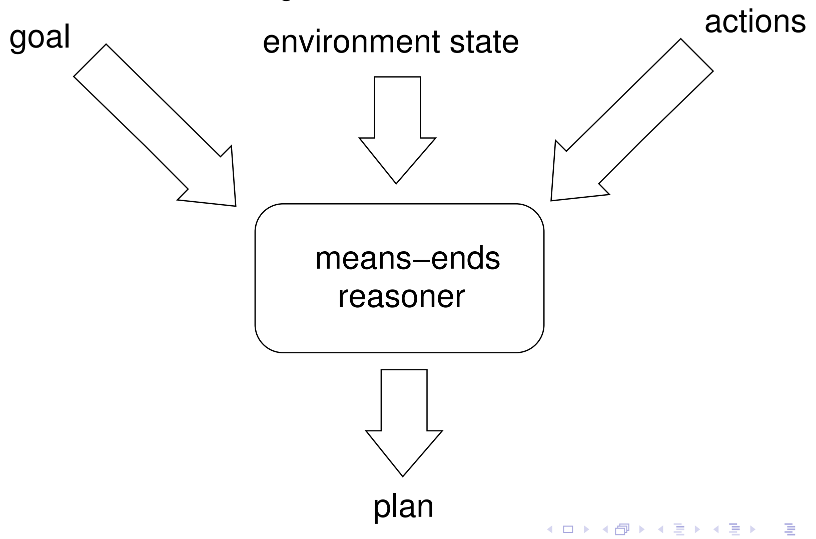
- More than just what you want.

  > *My desire to play basketball this afternoon is merely a potential influencer of my conduct this afternoon. It must vie with my other relevant desires [. . .] before it is settled what I will do. In contrast, once I intend to play basketball this afternoon, the matter is settled: I normally need not continue to weigh the pros and cons. When the afternoon arrives, I will normally just proceed to execute my intentions. (Bratman, 1990)*

- Additional aspect is commitment.

Means-ends reasoning

- Means-ends reasoning is the design of a course of action that will achieve some desired goal.



Means-ends reasoning

- Basic idea is to give a software system:
  - (representation of) goal/intention to achieve;
  - (representation of) actions it can perform; and
  - (representation of) the environment;

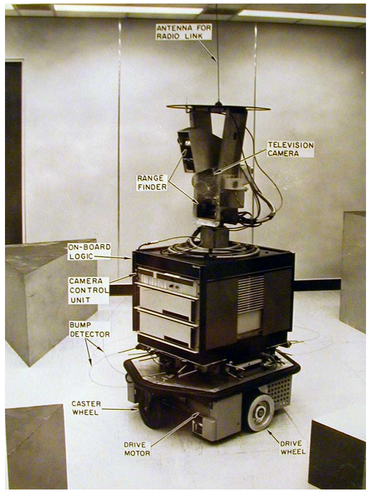
  and have it generate a plan to achieve the goal.
- This is automatic programming.
- Don't have to directly tell the system what to do.
- Let it figure out how to achieve the goal on its own.

Means-ends reasoning

- How do we do this?
- STRIPS, the Stanford Research Institute Problem Solver.

Planning

- Used in Shakey the robot:

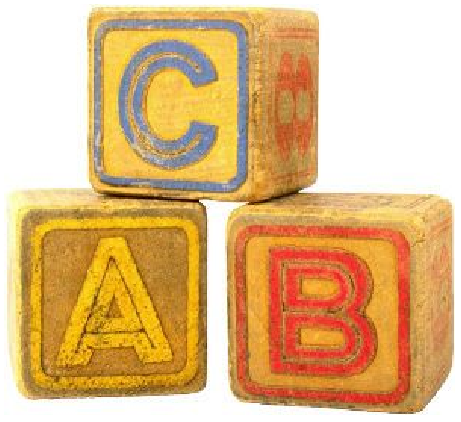


Representations

- Question: How do we represent. . .
  - goal to be achieved;
  - state of environment;
  - actions available to agent;
  - plan itself.
- Answer: We use logic, or something that looks a lot like logic.

STRIPS

- We'll illustrate the techniques with reference to the blocks world.
- A simple (toy) world, in this case one where we consider toys:

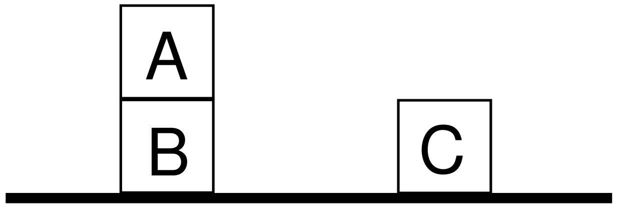


STRIPS

- The blocks world contains a robot arm, 3 blocks (A, B and C) of equal size, and a table-top.

STRIPS

- The aim is to generate a plan for the robot arm to build towers out of blocks.
- For a formal description, we'll clean it up a bit:



STRIPS

- To represent this environment, need an ontology.

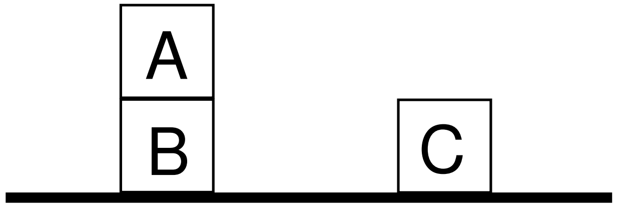
$On(x, y)$	obj x on top of obj y
$OnTable(x)$	obj x is on the table
$Clear(x)$	nothing is on top of obj x
$Holding(x)$	arm is holding x

- A language that allows us to express the things we need to say.

STRIPS

- Here is a representation of the blocks world described above:

$$
\begin{array}{l}
Clear(A) \\
On(A, B) \\
OnTable(B) \\
Clear(C) \\
OnTable(C)
\end{array}
$$



STRIPS

- Use the closed world assumption
  - Anything not stated is assumed to be false.

STRIPS

- A goal is represented as a set of formulae.
- Here is a goal:

$$\{ OnTable(A), \quad OnTable(B), \quad OnTable(C) \}$$

STRIPS

- Actions are represented as follows.
  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.

- The stack action occurs when the robot arm places the object x it is holding is placed on top of object y.

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

STRIPS

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

	$UnStack(x, y)$
pre	$On(x, y)\ \&\ Clear(x)\ \&\ ArmEmpty$
del	$On(x, y)\ \&\ ArmEmpty$
add	$Holding(x)\ \&\ Clear(y)$

Stack and UnStack are inverses of one-another.

STRIPS

- The pickup action occurs when the arm picks up an object x from the table.

	$Pickup(x)$
pre	$Clear(x)\ \&\ OnTable(x)\ \&\ ArmEmpty$
del	$OnTable(x)\ \&\ ArmEmpty$
add	$Holding(x)$

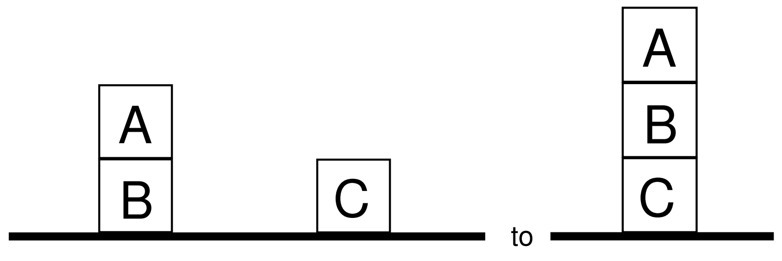
STRIPS

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

	$PutDown(x)$
pre	$Holding(x)$
del	$Holding(x)$
add	$Holding(x)\ \&\ ArmEmpty\ \&\ Clear(x)$

STRIPS

- What is a plan?
  A sequence (list) of actions, with variables replaced by constants.
- So, to get from:



STRIPS

- We need the set of actions:

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

- This is the plan.

STRIPS

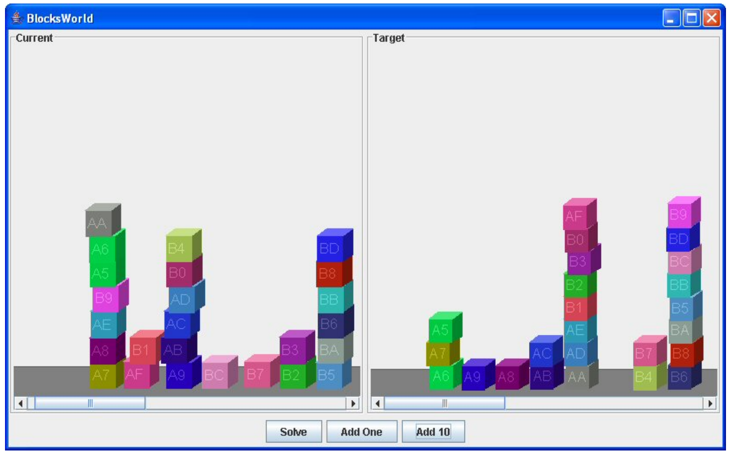
A plan is just a sequence of steps

STRIPS

- Creating the list of actions is not so hard in this case, but gets harder as the number of possible runs gets bigger.

STRIPS

- As here for example:



Implementation

- A first pass at an implementation of a practical reasoning agent:

```
Agent Control Loop Version 1
1.  while true
2.      observe the world;
3.      update internal world model;
4.      deliberate about what intention
          to achieve next;
5.      use means-ends reasoning to get
          a plan for the intention;
6.      execute the plan
7.  end while
```

Implementation

- We will not be concerned with stages (2) or (3) except to say that these are related to the functions *see* and *next* from Lecture #2.
- *see* is as before:

$$see : E \rightarrow Per$$

but instead of the function *next* which took a percept and used it to update the internal state of an agent, we have a belief revision function:

$$brf : \wp(Bel) \times Per \rightarrow \wp(Bel)$$

- *Bel* is the set of all possible beliefs that an agent might have.

Implementation

- Problem: deliberation and means-ends reasoning processes are not instantaneous.
  They have a time cost.
- Suppose that deliberation is optimal in that if it selects some intention to achieve, then this is the best thing for the agent.
- So the agent has selects an intention to achieve that would have been optimal at the time it observed the world.
- The world may change in the meantime.
- Even if the agent can compute the right thing to do, it may not do the right thing.

Implementation

Implementation

- Let's make the algorithm more formal.

```
Agent Control Loop Version 2
1.  $B := B_0$; /* initial beliefs */
2.  while true do
3.      get next percept $\rho$;
4.      $B := brf(B, \rho)$;
5.      $I := deliberate(B)$;
6.      $\pi := plan(B, I)$;
7.      $execute(\pi)$
8.  end while
```

Implementation

- Where:
  - $B \subseteq Bel$ is the agent's set of beliefs;
  - I is the agent's set of intentions;
  - *plan* is exactly what we discussed above; and
  - *execute* is a function that executes each action in a plan.
- Note that the deliberation and planning steps are what is required to make the agent proactive
  - Will work towards some set of intentions.
- Updating beliefs each time aorund the loop makes it reactive.
  - Able to respond to changes in the environment.

Implementation

- How might we implement these functions?

Deliberation

- How does an agent deliberate?
  - begin by trying to understand what the options available to you are;
  - choose between them, and commit to some.

  Chosen options are then intentions.
- The *deliberate* function can be decomposed into two distinct functional components:
  - option generation; and
  - filtering.

Deliberation

- In option generation, the agent generates a set of possible alternatives.
- Represent option generation via a function, *options*:

$$options : \wp(Bel) \times \wp(Int) \rightarrow \wp(Des)$$

  where *Del* is the set of all possible desires and *Int* is the set of all possible intentions.
- This takes the agent's current beliefs and current intentions, and from them determines a set of options (= desires).

Deliberation

- In filtering, the agent chooses between competing alternatives, and commits to achieving them.
- In order to select between competing options, an agent uses a filter function.

$$filter : \wp(Bel) \times \wp(Des) \times \wp(Int) \rightarrow \wp(Int)$$

Deliberation

```
Agent Control Loop Version 3

1.  $B := B_0$;
2.  $I := I_0$;
3.  while true do
4.      get next percept $\rho$;
5.      $B := brf(B, \rho)$;
6.      $D := options(B, I)$;
7.      $I := filter(B, D, I)$;
8.      $\pi := plan(B, I)$;
9.      $execute(\pi)$
10. end while
```

- where $D \subseteq Des$.

Commitment Strategies

Some time in the not-so-distant future, you are having trouble with your new household robot. You say "Willie, bring me a beer." The robot replies "OK boss."

P. R. Cohen and H. J. Levesque (1990). Intention is choice with commitment. Artificial intelligence, 42(2), 213-261.

Commitment Strategies

Twenty minutes later, you screech “Willie, why didn’t you bring me that beer?” It answers “Well, I intended to get you the beer, but I decided to do something else.”

P. R. Cohen and H. J. Levesque (1990). Intention is choice with commitment. Artificial intelligence, 42(2), 213-261.

Aside

Commitment strategies

Under-committed

Commitment Strategies

After retrofitting, Willie is returned, marked “Model C: The Committed Assistant.” Again, you ask Willie to bring you a beer. Again, it accedes, replying “Sure thing.” Then you ask: “What kind of beer did you buy?” It answers: “Genessee.” You say “Never mind.” One minute later, Willie trundles over with a Genessee in its gripper. [. . .]

P. R. Cohen and H. J. Levesque (1990). Intention is choice with commitment. Artificial intelligence, 42(2), 213-261.

Commitment strategies

Over-committed

Commitment strategies

After still more tinkering [. . .] you ask [the robot] to bring you your last beer. [. . .] The robot gets the beer and starts towards you. As it approaches, it lifts its arm, wheels around, deliberately smashes the bottle, and trundles off.

P. R. Cohen and H. J. Levesque (1990). Intention is choice with commitment. Artificial intelligence, 42(2), 213-261.

Commitment strategies

[. . .] the robot [says] that according to its specifications, it kept its commitments as long as required — commitments must be dropped when fulfilled or impossible to achieve. By smashing the bottle, the commitment became unachievable.

P. R. Cohen and H. J. Levesque (1990). Intention is choice with commitment. Artificial intelligence, 42(2), 213-261.

Commitment strategies

Not really the right way to end a commitment.

Degrees of Commitment

How committed should a robot be? Two possibilities:

- Blind commitment
  A blindly committed agent will continue to maintain an intention until it believes the intention has actually been achieved. Blind commitment is also sometimes referred to as fanatical commitment.
- Single-minded commitment
  A single-minded agent will continue to maintain an intention until it believes that either the intention has been achieved, or else that it is no longer possible to achieve the intention.

Degrees of Commitment

- An agent has commitment both to ends
  - the state of affairs it wishes to bring about

  and means
  - the mechanism via which the agent wishes to achieve the state of affairs
- Currently, our agent control loop is overcommitted, both to means and ends.
  Modification: replan if ever a plan goes wrong.

Commitment strategies

- To write the algorithm down we need to refine our notion of plan execution.
- If π is a plan, then:
  - $\mathit{empty}(\pi)$ is true if there are no more actions in the plan.
  - $\mathit{hd}(\pi)$ returns the first action in the plan.
  - $\mathit{tail}(\pi)$ returns the plan minus the head of the plan.
  - $\mathit{sound}(\pi, I, B)$ means that π is a correct plan for I given B.
- Now we can say the following:

Commitment strategies

- The next version of the control loop:

```
Agent Control Loop Version 4

1.  $B := B_0$;
2.  $I := I_0$;
3.  while true do
4.      get next percept $\rho$;
5.      $B := brf(B, \rho)$;
6.      $D := options(B, I)$;
7.      $I := filter(B, D, I)$;
8.      $\pi := plan(B, I)$;

⋮
```

Commitment strategies

```
Agent Control Loop Version 4 (cont)
⋮
9.      while not $empty(\pi)$ do
10.         $\alpha := hd(\pi)$;
11.         $execute(\alpha)$;
12.         $\pi := tail(\pi)$;
13.         get next percept $\rho$;
14.         $B := brf(B,\rho)$;
15.         if not $sound(\pi, I, B)$ then
16.             $\pi := plan(B, I)$
17.         end-if
18.     end-while
19. end-while
```

Commitment strategies

- Makes the control loop more reactive, able to change intention when the world changes.
- Still overcommitted to intentions (means).
  Never stops to consider whether or not its intentions are appropriate.
- Modification: stop to determine whether intentions have succeeded or whether they are impossible:

  single-minded commitment.

Commitment strategies

- Next version:

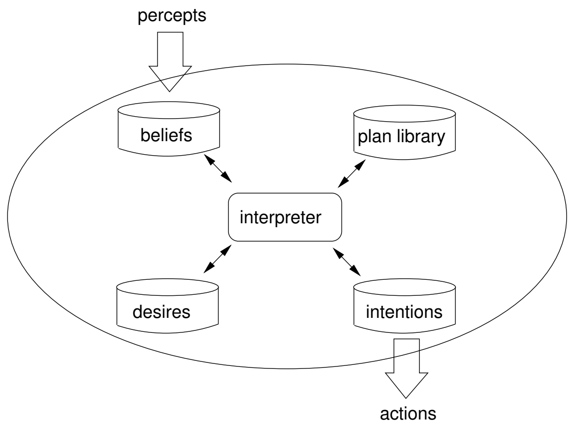
```
Agent Control Loop Version 5

1.  $B := B_0$;
2.  $I := I_0$;
3.  while true do
4.      get next percept $\rho$;
5.      $B := brf(B,\rho)$;
6.      $D := options(B, I)$;
7.      $I := filter(B, D, I)$;
8.      $\pi := plan(B, I)$;
⋮
```

Commitment strategies

```
⋮
9.      while not $empty(\pi)$
                or $succeeded(I, B)$
                or $impossible(I, B)$) do
10.         $\alpha := hd(\pi)$;
11.         $execute(\alpha)$;
12.         $\pi := tail(\pi)$;
13.         get next percept $\rho$;
14.         $B := brf(B,\rho)$;
15.         if not $sound(\pi, I, B)$ then
16.             $\pi := plan(B, I)$
17.         end-if
18.     end-while
19. end-while
```

PRS

- We now make the discussion even more concrete by introducing an actual agent architecture:



- The Procedural Reasoning System.

PRS

- The key feature in PRS is to ease the task of planning
- Besides data structures for beliefs, desires and intentions (represented as first-order logic formulae) PRS contains a plan library
- The library consists of "pre-cooked" plans constructed by the programmer.
- Plans are triplets consisting of:
  - a goal: the post-condition of the plan
  - a context: the pre-condition of the plan
  - a body: the body of the plan containing actions and possibly sub-goals that need to be achieved to carry out the plan

PRS

- At start-up, the system will have some initial beliefs, a library of plans, and a top-level goal
  (Something like a main method)
- The goals to be achieved are then pushed into a stack (the intention stack)
- The system then searches for plans:
  - whose goal is the first goal in the stack; and
  - whose context is compatible with the current beliefs
- Among the set of plans satisfying those constraints, the system will choose one.
  (Possibly based on some ranking or utility of the different plans)
- The chosen plan is then executed

  . . . and new goals might be pushed into the stack

Example PRS

- This is an example that uses the JAM system
  - Implementation of PRS
  - Running the bloacks world example from before
- First a statement of the goal and world state.

```
GOALS:

    ACHIEVE blocks_stacked;

FACTS:

// Block1 on Block2 initially so need to clear
// Block2 before stacking.

    FACT ON "Block1" "Block2";
    FACT ON "Block2" "Table";
    FACT ON "Block3" "Table";
    FACT CLEAR "Block1";
    FACT CLEAR "Block3";
    FACT CLEAR "Table";
    FACT initialized "False";
```

JAM

- Now “plans” that move blocks. Here is a top level one.

```
Plan: {
NAME: "Top-level plan"
DOCUMENTATION:
    "Establish Block1 on Block2 on Block3."
GOAL:
    ACHIEVE blocks_stacked;
CONTEXT:
BODY:
    EXECUTE print "Goal is Block1 on Block2 on Block2 on Table.\n";
    EXECUTE print "World Model at start is:\n";
    EXECUTE printWorldModel;
    EXECUTE print "ACHIEVEing Block3 on Table.\n";
    ACHIEVE ON "Block3" "Table";
    EXECUTE print "ACHIEVEing Block2 on Block3.\n";
    ACHIEVE ON "Block2" "Block3";
    EXECUTE print "ACHIEVEing Block1 on Block2.\n";
    ACHIEVE ON "Block1" "Block2";
    EXECUTE print "World Model at end is:\n";
    EXECUTE printWorldModel;
}
```

JAM

- Here is a lower level plan.

```
Plan: {
NAME: "Stack blocks that are already clear"
GOAL:
    ACHIEVE ON $OBJ1 $OBJ2;
CONTEXT:
BODY:
    EXECUTE print "Making sure " $OBJ1 " is clear\n";
    ACHIEVE CLEAR $OBJ1;
    EXECUTE print "Making sure " $OBJ2 " is clear.\n";
    ACHIEVE CLEAR $OBJ2;
    EXECUTE print "Moving " $OBJ1 " on top of " $OBJ2 ".\n";
    PERFORM move $OBJ1 $OBJ2;
UTILITY: 10;

FAILURE:
    EXECUTE print "\n\nStack blocks failed!\n\n";
}
```

JAM

- Another lower level plan.

```
Plan: {
NAME: "Clear a block"
GOAL:
    ACHIEVE CLEAR $OBJ;
CONTEXT:
    FACT ON $OBJ2 $OBJ;
BODY:
    EXECUTE print "Clearing " $OBJ2 " from on top of " $OBJ "\n";
    EXECUTE print "Moving " $OBJ2 " to table.\n";
    ACHIEVE ON $OBJ2 "Table";

EFFECTS:
    EXECUTE print "CLEAR: Retracting ON " $OBJ2 " " $OBJ "\n";
    RETRACT ON $OBJ1 $OBJ;

FAILURE:
    EXECUTE print "\n\nClearing block " $OBJ " failed!\n\n";
}
```

JAM

- And another lower level plan.

```
Plan: {
NAME: "Move a block onto another object"
GOAL:
    PERFORM move $OBJ1 $OBJ2;
CONTEXT:
    FACT CLEAR $OBJ1;
    FACT CLEAR $OBJ2;
BODY:
    EXECUTE print "Performing low-level move action"
    EXECUTE print " of " $OBJ1 " to " $OBJ2 ".\n";

EFFECTS:
    WHEN : TEST (!= $OBJ2 "Table") {
EXECUTE print "    Retracting CLEAR " $OBJ2 "\n";
RETRACT CLEAR $OBJ2;
    };
    FACT ON $OBJ1 $OBJ3;
    EXECUTE print "    move: Retracting ON " $OBJ1 " " $OBJ3 "\n";
    RETRACT ON $OBJ1 $OBJ3;
    EXECUTE print "    move: Asserting CLEAR " $OBJ3 "\n";
    ASSERT CLEAR $OBJ3;
    EXECUTE print "    move: Asserting ON " $OBJ1 " " $OBJ2 "\n\n";
    ASSERT ON $OBJ1 $OBJ2;

FAILURE:
    EXECUTE print "\n\nMove failed!\n\n";
}
```

PRS/JAM

- Key thing in this example is that we don't write a program to do specific things with the blocks.
- Rather we:
  - Say how the world is
  - Say how blocks can be moved
  - Say how we want the world to be

  and let the system figure out the moves.

Summary

- This lecture has covered a lot of ground on practical reasoning.
- We started by discussing what practical reasoning was, and how it relates to intentions.
- We then looked at planning (how an agent achieves its desires) and how deliberation and means-ends reasoning fit into the basic agent control loop.
- We then refined the agent control loop, considering commitment strategies
- Finally, we looked at an implemented system that is similar to the one we will use.
- Soon we'll see how we can control the NXT in this way.