LECTURE 4:
PRACTICAL REASONING

An Introduction to MultiAgent Systems
http://www.csc.liv.ac.uk/~mjw/pubs/imas

4-1

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." (Bratman)
- Practical reasoning is distinguished from *theoretical reasoning* – theoretical reasoning is directed towards beliefs

4-2

Practical Reasoning

- Human practical reasoning consists of two activities:
  - *deliberation*
    deciding *what* state of affairs we want to achieve
  - *means-ends reasoning*
    deciding *how* to achieve these states of affairs
- The outputs of deliberation are *intentions*

4-3

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.
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 such that ϕ and ψ are mutually exclusive.
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.

4-4

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 ϕ was not possible.
6. Under certain circumstances, agents believe they will bring about their intentions.
   It would not normally be rational of me to believe that I would *bring my intentions about; intentions can* fail. *Moreover, it does not make sense that if I believe ϕ is inevitable that I would adopt it as an intention.*

4-5

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!

4-6

Intentions in Practical Reasoning

- Notice that intentions are much stronger than mere desires:

  "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)

4-7

Planning Agents

- Since the early 1970s, the AI planning community has been closely concerned with the design of artificial agents
- Planning is essentially automatic programming: the design of a course of action that will achieve some desired goal
- Within the symbolic AI community, it has long been assumed that some form of AI planning system will be a central component of any artificial agent
- Building largely on the early work of Fikes & Nilsson, many planning algorithms have been proposed, and the theory of planning has been well-developed

4-8

What is Means-End Reasoning?

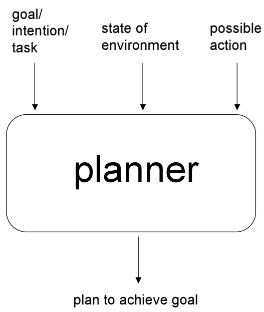
- Basic idea is to give an agent:
  - representation of goal/intention to achieve
  - representation actions it can perform
  - representation of the environment

and have it generate a *plan* to achieve the goal

- Essentially, this is

  automatic programming

4-9



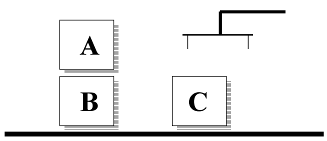
4-10

Planning

- Question: How do we *represent*. . .
  - goal to be achieved
  - state of environment
  - actions available to agent
  - plan itself

4-11

The Blocks World



- We'll illustrate the techniques with reference to the *blocks world* (like last week)
- Contains a robot arm, 3 blocks (A, B, and C) of equal size, and a table-top

4-12

The Blocks World Ontology

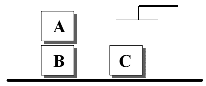
- To represent this environment, need an *ontology*

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

The Blocks World

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

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

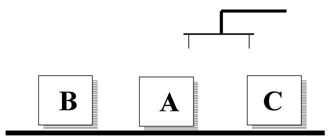


- Use the *closed world assumption*: anything not stated is assumed to be *false*

The Blocks World

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

  $OnTable(A) \wedge OnTable(B) \wedge OnTable(C)$

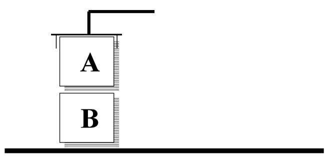


The Blocks World

- *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*

The Blocks World Operators



- Example 1:
  The *stack* action occurs when the robot arm places the object x it is holding is placed on top of object y.

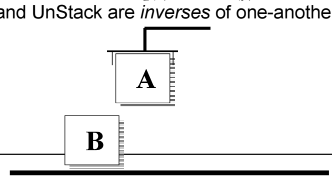
	Stack(x, y)
pre	$Clear(y) \wedge Holding(x)$
del	$Clear(y) \wedge Holding(x)$
add	$ArmEmpty \wedge On(x, y)$

The Blocks World Operators

- 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	$On(x, y) \wedge Clear(x) \wedge ArmEmpty$
del	$On(x, y) \wedge ArmEmpty$
add	$Holding(x) \wedge Clear(y)$

  Stack and UnStack are *inverses* of one-another.



The Blocks World Operators

- Example 3:
  The *pickup* action occurs when the arm picks up an object x from the table.

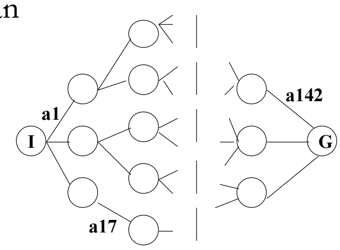
	Pickup(x)
pre	$Clear(x) \wedge OnTable(x) \wedge ArmEmpty$
del	$OnTable(x) \wedge ArmEmpty$
add	$Holding(x)$

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

	Putdown(x)
pre	$Holding(x)$
del	$Holding(x)$
add	$Clear(x) \wedge OnTable(x) \wedge ArmEmpty$

4-19

A Plan



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

4-20

The STRIPS approach

- The original STRIPS system used a goal stack to control its search
- The system has a database and a goal stack, and it focuses attention on solving the top goal (which may involve solving subgoals, which are then pushed onto the stack, etc.)

4-21

The Basic STRIPS Idea

- Place goal on goal stack:

Goal1

- Considering top Goal1, place onto it its subgoals:

GoalS1-2
GoalS1-1
Goal1

- Then try to solve subgoal GoalS1-2, and continue…

4-22

Stack Manipulation Rules, STRIPS

<u>If on top of goal stack:</u>	<u>Then do:</u>
Compound or single goal matching the current state description	Remove it
Compound goal *not* matching the current state description	1. Keep original compound goal on stack 2. List the unsatisfied component goals on the stack in some *new* order
Single-literal goal not matching the current state description	Find rule whose instantiated add-list includes the goal, and 1. Replace the goal with the instantiated rule; 2. Place the rule's instantiated precondition formula on top of stack
Rule	1. Remove rule from stack; 2. Update database using rule; 3. Keep track of rule (for solution)
Nothing	Stop

"Underspecified" – there are decision branches here within the search tree… 4-23

Implementing Practical Reasoning Agents

- 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
```

- (We will not be concerned with stages (2) or (3))

4-24

Implementing Practical Reasoning Agents

- Problem: deliberation and means-ends reasoning processes are not instantaneous. They have a *time cost*.
- Suppose the agent starts deliberating at t_0, begins means-ends reasoning at t_1, and begins executing the plan at time t_2. Time to deliberate is

  $$t_{deliberate} = t_1 - t_0$$
- and time for means-ends reasoning is

  $$t_{me} = t_2 - t_1$$

Implementing Practical Reasoning Agents

- Further suppose that deliberation is *optimal* in that if it selects some intention to achieve, then this is the best thing for the agent. (Maximizes expected utility.)
- So at time t_1, the agent has selected an intention to achieve that would have been optimal *if it had been achieved at* t_0.
  But unless $t_{deliberate}$ is vanishingly small, then the agent runs the risk that the intention selected is no longer optimal by the time the agent has fixed upon it.
- This is *calculative rationality*.
- Deliberation is only half of the problem: the agent still has to determine *how* to achieve the intention.

Implementing Practical Reasoning Agents

- So, this agent will have overall optimal behavior in the following circumstances:

1. When deliberation and means-ends reasoning take a vanishingly small amount of time; or
2. When the world is guaranteed to remain static while the agent is deliberating and performing means-ends reasoning, so that the assumptions upon which the choice of intention to achieve and plan to achieve the intention remain valid until the agent has completed deliberation and means-ends reasoning; or
3. When an intention that is optimal when achieved at time t_0 (the time at which the world is observed) is guaranteed to remain optimal until time t_2 (the time at which the agent has found a course of action to achieve the intention).

Implementing Practical Reasoning Agents

- Let's make the algorithm more formal:

```
Agent Control Loop Version 2
B := B0; /* initial beliefs */
while true do
     get next percept ρ;
     B := brf(B, ρ);
     I := deliberate(B);
     π := plan(B, I);
     execute(π)
end while
```


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

Deliberation

- The *deliberate* function can be decomposed into two distinct functional components:
  - *option generation*
    in which the agent generates a set of possible alternatives;
    Represent option generation via a function, *options*, which takes the agent's current beliefs and current intentions, and from them determines a set of options (= *desires*)
  - *filtering*
    in which the agent chooses between competing alternatives, and commits to achieving them.
    In order to select between competing options, an agent uses a *filter* function.

Deliberation

Agent Control Loop Version 3

```
B := B_0;
I := I_0;
while true do
    get next percept ρ;
    B := brf(B, ρ);
    D := options(B, I);
    I := filter(B, D, I);
    π := plan(B, I);
    execute(π)
end while
```

4-31

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.” 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.” Miffed, you send the wise guy back to the manufacturer, complaining about a lack of commitment. 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. This time, you angrily return Willie for overcommitment. After still more tinkering, the manufacturer sends Willie back, promising no more problems with its commitments. So, being a somewhat trusting customer, you accept the rascal back into your household, but as a test, you ask it to bring you your last beer. Willie again accedes, saying “Yes, Sir.” (Its attitude problem seems to have been fixed.) 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. Back at the plant, when interrogated by customer service as to why it had abandoned its commitments, the robot replies 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.”

4-32

Commitment Strategies

- The following *commitment strategies* are commonly discussed in the literature of rational agents:
  - *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.
  - *Open-minded commitment*
    An open-minded agent will maintain an intention as long as it is still believed possible.

4-33

Commitment Strategies

- An agent has commitment both to *ends* (i.e., the wishes to bring about), and *means* (i.e., 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

4-34

Agent Control Loop Version 4

```
B := B_0;
I := I_0;
while true do
    get next percept ρ;
    B := brf(B, ρ);
    D := options(B, I);
    I := filter(B, D, I);
    π := plan(B, I);
    while not empty(π) do
        α := hd(π);
        execute(α);
        π := tail(π);
        get next percept ρ;
        B := brf(B, ρ);
        if not sound(π, I, B) then
            π := plan(B, I)
        end-if
    end-while
end-while
```

4-35

Commitment Strategies

- Still overcommitted to intentions: 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*)

4-36

```
Agent Control Loop Version 5
B := B0;
I := I0;
while true do
     get next percept ρ;
     B := brf(B, ρ);
     D := options(B, I);
     I := filter(B, D, I);
     π := plan(B, I);
     while not empty(π)
              or succeeded(I, B)
              or impossible(I, B)) do
          α := hd(π);
          execute(α);
          π := tail(π);
          get next percept ρ;
          B := brf(B, ρ);
          if not sound(π, I, B) then
               π := plan(B, I)
          end-if
     end-while
end-while
```


Intention Reconsideration

- Our agent gets to reconsider its intentions once every time around the outer control loop, i.e., when:
  - it has completely executed a plan to achieve its current intentions; or
  - it believes it has achieved its current intentions; or
  - it believes its current intentions are no longer possible.
- This is limited in the way that it permits an agent to *reconsider* its intentions
- Modification: Reconsider intentions after executing every action


```
Agent Control Loop Version 6

B := B0;
I := I0;
while true do
     get next percept ρ;
     B := brf(B, ρ);
     D := options(B, I);
     I := filter(B, D, I);
     π := plan(B, I);
     while not (empty(π)
              or succeeded(I, B)
              or impossible(I, B)) do
          α := hd(π);
          execute(α);
          π := tail(π);
          get next percept ρ;
          B := brf(B, ρ);
          D := options(B, I);
          I := filter(B, D, I);
          if not sound(π, I, B) then
               π := plan(B, I)
          end-if
     end-while
end-while
```


Intention Reconsideration

- But intention reconsideration is *costly*!
  A dilemma:
  - an agent that does not stop to reconsider its intentions sufficiently often will continue attempting to achieve its intentions even after it is clear that they cannot be achieved, or that there is no longer any reason for achieving them
  - an agent that *constantly* reconsiders its attentions may spend insufficient time actually working to achieve them, and hence runs the risk of never actually achieving them
- Solution: incorporate an explicit *meta-level control* component, that decides whether or not to reconsider


```
Agent Control Loop Version 7

B := B0;
I := I0;
while true do
     get next percept ρ;
     B := brf(B, ρ);
     D := options(B, I);
     I := filter(B, D, I);
     π := plan(B, I);
     while not (empty(π)
              or succeeded(I, B)
              or impossible(I, B)) do
          α := hd(π);
          execute(α);
          π := tail(π);
          get next percept ρ;
          B := brf(B, ρ);
          if reconsider(I, B) then
               D := options(B, I);
               I := filter(B, D, I);
          end-if
          if not sound(π, I, B) then
               π := plan(B, I)
          end-if
     end-while
end-while
```


Possible Interactions

- The possible interactions between meta-level control and deliberation are:

Situation number	Chose to deliberate?	Changed intentions?	Would have changed intentions?	*reconsider*(...) optimal?
1	No	—	No	Yes
2	No	—	Yes	No
3	Yes	No	—	No
4	Yes	Yes	—	Yes

Intention Reconsideration

- In situation (1), the agent did not choose to deliberate, and as consequence, did not choose to change intentions. Moreover, if it *had* chosen to deliberate, it would not have changed intentions. In this situation, the *reconsider(…)* function is behaving optimally.
- In situation (2), the agent did not choose to deliberate, but if it had done so, it *would* have changed intentions. In this situation, the *reconsider(…)* function is not behaving optimally.
- In situation (3), the agent chose to deliberate, but did not change intentions. In this situation, the *reconsider(…)* function is not behaving optimally.
- In situation (4), the agent chose to deliberate, and did change intentions. In this situation, the *reconsider(…)* function is behaving optimally.
- An important assumption: cost of *reconsider(…)* is *much* less than the cost of the deliberation process itself.

Optimal Intention Reconsideration

- Kinny and Georgeff's experimentally investigated effectiveness of intention reconsideration strategies
- Two different types of reconsideration strategy were used:
  - *bold* agents
    never pause to reconsider intentions, and
  - *cautious* agents
    stop to reconsider after every action
- *Dynamism* in the environment is represented by the *rate of world change*, γ

Optimal Intention Reconsideration

- Results (not surprising):
  - If γ is low (i.e., the environment does not change quickly), then bold agents do well compared to cautious ones. This is because cautious ones waste time reconsidering their commitments while bold agents are busy working towards — and achieving — their intentions.
  - If γ is high (i.e., the environment changes frequently), then cautious agents tend to outperform bold agents. This is because they are able to recognize when intentions are doomed, and also to take advantage of serendipitous situations and new opportunities when they arise.

BDI Theory and Practice

- We now consider the *semantics* of BDI architectures: to what extent does a BDI agent satisfy a *theory of agency*
- In order to give a semantics to BDI architectures, Rao & Georgeff have developed *BDI logics*: non-classical logics with modal connectives for representing beliefs, desires, and intentions
- The 'basic BDI logic' of Rao and Georgeff is a quantified extension of the expressive branching time logic CTL*
- Underlying semantic structure is a *labeled branching time* framework

BDI Logic

- From classical logic: $\wedge$, 🕓, $\neg$, …
- The CTL* *path quantifiers*:
  - Aϕ 'on all paths, ϕ'
  - Eϕ 'on some paths, ϕ'
- The BDI connectives:
  - (Bel *i* ϕ) *i* believes ϕ
  - (Des *i* ϕ) *i* desires ϕ
  - (Int *i* ϕ) *i* intends ϕ

BDI Logic

- Semantics of BDI components are given via accessibility relations over 'worlds', where each world is itself a branching time structure
- Properties required of accessibility relations ensure belief logic KD45, desire logic KD, intention logic KD
  (Plus interrelationships. . .)

Axioms of KD45

- (1) Bel(p → q) → (Bel p → Bel q) (K)

If you believe that p implies q then if you believe p then you believe q

- (2) Bel p → ¬Bel ¬p (D)

This is the consistency axiom, stating that if you believe p then you do not believe that p is false

- (3) Bel p → Bel Bel p (4)

If you believe p then you believe that you believe p

- (4) ¬Bel p → Bel ¬Bel p (5)

If you do not believe p then you believe that you do not believe that p is true

4-49

Axioms of KD45

It also entails the two inference rules of *modus ponens* and necessitation:

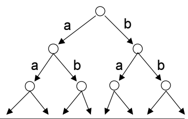
- (5) if p, and p → q, then q (MP)
- (6) if p is a theorem of KD45 then so is Bel p (Nec)

This last rule just states that you believe all theorems implied by the logic

4-50

CTL Temporal Logic (from David Garlan's slides, CMU)

- Branching time logic views a computation as a (possibly infinite) tree or DAG of states connected by atomic events
- At each state the outgoing arcs represent the actions leading to the possible next states in some execution
- Example:
  P = (a → P) ∏ (b → P)



4-51

CTL* Notation

- Variant of branching time logic that we look at is called CTL*, for Computational Tree Logic (star)
- In this logic
  - A = "for every path"
  - E = "there exists a path"
  - G = "globally" (similar to □)
  - F = "future" (similar to ◊)

4-52

Paths versus States

- A and E refer to paths
  - A requires that all paths have some property
  - E requires that at least some path has the property
- G and F refer to states on a path
  - G requires that all states on the given path have some property
  - F requires that at least one state on the path has the property

4-53

CTL* Examples

- AG p
  - For every computation (i.e., path from the root), in every state, p is true
  - Hence, means the same as □p
- EG p
  - There exists a computation (path) for which p is always true

4-54

CTL* Examples continued

- AF p
  - For every path, eventually state p is true
  - Hence, means the same as ◊p
  - Therefore, p is *inevitable*
- EF p
  - There is some path for which p is eventually true
  - I.e., p is “reachable”
  - Therefore, p will hold *potentially*

4-55

Some Useful CTL* Equalities

- From linear temporal logic:
  □P ≡ ~ ◊ ~P
  ◊P ≡ ~ □ ~P
- In CTL* we can say:
  AG p ≡ ~ EF ~p
  EG p ≡ ~ AF ~p
- We can rewrite AG p ≡ ~ EF ~p as
  EF p ≡ ~AG ~p

4-56

BDI Logic

- Let us now look at some possible axioms of BDI logic, and see to what extent the BDI architecture could be said to satisfy these axioms
- In what follows, let
  - α be an *O-formula*, i.e., one which contains no positive occurrences of A
  - ϕ be an arbitrary formula

4-57

BDI Logic

- *Belief goal compatibility*:
  (Des α) → (Bel α)
  States that if the agent has a goal to optionally achieve something, this thing must be an option. This axiom is operationalized in the function *options*: an option should not be produced if it is not believed possible.
- *Goal-intention compatibility*:
  (Int α) → (Des α)
  States that having an intention to optionally achieve something implies having it as a goal (i.e., there are no intentions that are not goals). Operationalized in the *deliberate* function.

4-58

BDI Logic

- *Volitional commitment*:
  (Int *does*(*a*)) → *does*(*a*)
  If you intend to perform some action *a* next, then you do *a* next.
  Operationalized in the *execute* function.
- *Awareness of goals & intentions*:
  (Des ϕ) → (Bel (Des ϕ))
  (Int ϕ) → (Bel (Int ϕ))
  Requires that new intentions and goals be posted as events.

4-59

BDI Logic

- *No unconscious actions*:
  done(*a*) → Bel(*done*(*a*))
  If an agent does some action, then it is aware that it has done the action.
  Operationalized in the *execute* function.
  A stronger requirement would be for the success or failure of the action to be posted.
- *No infinite deferral*:
  (Int ϕ) → A◊(¬(Int ϕ))
  An agent will eventually either act for an intention, or else drop it.

4-60

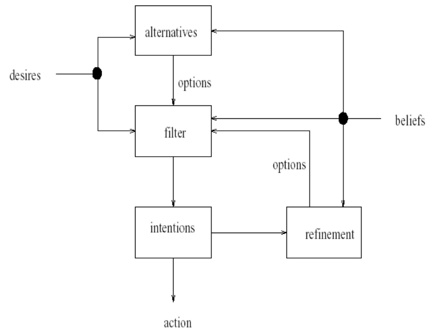
Implemented BDI Agents: IRMA

- IRMA – Intelligent Resource-bounded Machine Architecture – Bratman, Israel, Pollack
- IRMA has four key symbolic data structures:
  - a *plan library*
  - explicit representations of
    - *beliefs*: information available to the agent — may be represented symbolically, but may be simple variables
    - *desires*: those things the agent would *like* to make true — think of desires as *tasks* that the agent has been allocated; in humans, not necessarily logically consistent, but our agents will be! (goals)
    - *intentions*: desires that the agent has *chosen* and *committed to*

IRMA

- Additionally, the architecture has:
  - a *reasoner* for reasoning about the world; an inference engine
  - a *means-ends analyzer* determines which plans might be used to achieve intentions
  - an *opportunity analyzer* monitors the environment, and as a result of changes, generates new options
  - a *filtering process* determines which options are compatible with current intentions
  - a *deliberation process* responsible for deciding upon the 'best' intentions to adopt

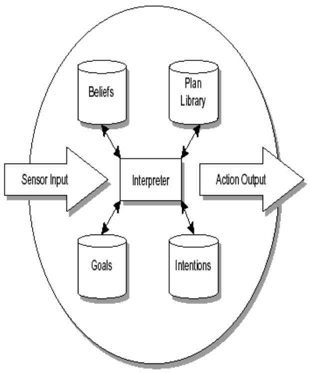
IRMA



Implemented BDI Agents: PRS

- Another BDI-based agent architecture: the PRS – Procedural Reasoning System (Georgeff, Lansky)
- In the PRS, each agent is equipped with a *plan library*, representing that agent's *procedural knowledge*: knowledge about the mechanisms that can be used by the agent in order to realize its intentions
- The options available to an agent are directly determined by the plans an agent has: an agent with no plans has no options
- In addition, PRS agents have explicit representations of beliefs, desires, and intentions, as above

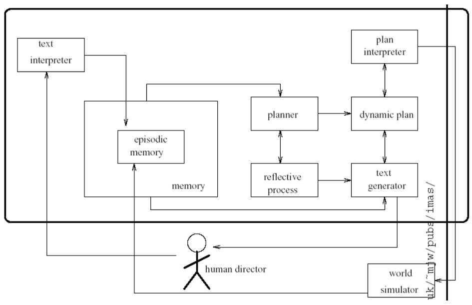
PRS



HOMER – An agent that plans

- Vere & Bickmore developed HOMER: a simulated robot submarine, in a two-dimensional 'Seaworld'
- HOMER takes instructions from a user in a subset of English with about an 800 word vocabulary
- HOMER can plan how to achieve its instructions, (which usually relate to collecting and moving items around the Seaworld), and can then execute its plans, modifying them as required during execution
- HOMER has a limited *episodic memory*, and using this, is able to answer questions about its past experiences

Homer



Dialogues with HOMER

```
EXAMPLE DIALOGUE 1
U:  What is in front of you?
H:  A log.
U:  Do you own the log.
H:  No, I don't.
U:  The log belongs to you.
H:  Oh.
[...]
U:  Turn away from your log.
H:  OK, I am turning.
[LOG IS MOVED]
U:  Turn around.
H:  OK, I'm turning.
H:  I've lost the log!
```

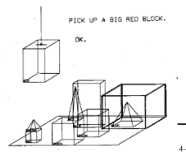

Dialogues with Homer

```
EXAMPLE DIALOGUE 2
U:  Drop the package at the barge next Saturday at 9pm.
H:  OK.
[HOMER GENERATES PLAN]
U:  Are you going to the barge next Saturday?
H:  Yes.
U:  What time?
H:  8.56pm
[USER ADDS CONTRADICTORY INSTRUCTIONS, WHICH HOMER DETECTS.]
```


Comparison with SHRDLU

- **The dialog that was used as a SHRDLU demo:**

Person: PICK UP A BIG RED BLOCK.
Computer: OK. (does it)
Person: GRASP THE PYRAMID.
Computer: I DON'T UNDERSTAND WHICH PYRAMID YOU MEAN.
Person: FIND A BLOCK WHICH IS TALLER THAN THE ONE YOU ARE HOLDING AND PUT IT INTO THE BOX.
Computer: BY "IT", I ASSUME YOU MEAN THE BLOCK WHICH IS TALLER THAN THE ONE I AM HOLDING.
Computer: OK. (does it)
Person: WHAT DOES THE BOX CONTAIN?
Computer: THE BLUE PYRAMID AND THE BLUE BLOCK.
Person: WHAT IS THE PYRAMID SUPPORTED BY?
Computer: THE BOX.
Person: HOW MANY BLOCKS ARE NOT IN THE BOX?
Computer: FOUR OF THEM.
Person: IS AT LEAST ONE OF THEM NARROWER THAN THE ONE WHICH I TOLD YOU TO PICK UP?
Computer: YES, THE RED CUBE.
Person: IS IT SUPPORTED?
Computer: YES, BY THE TABLE.
Person: CAN THE TABLE PICK UP BLOCKS?
Computer: NO.
Person: CAN A PYRAMID BE SUPPORTED BY A BLOCK?
Computer: YES.
Person: CAN A PYRAMID SUPPORT A PYRAMID?
Computer: I DON'T KNOW.



SHRDLU Dialog

Person: STACK UP TWO PYRAMIDS.
Computer: I CAN'T.
Person: WILL YOU PLEASE STACK UP BOTH OF THE RED BLOCKS AND EITHER A GREEN CUBE OR A PYRAMID?
Computer: OK. (does it)
Person: WHICH CUBE IS SITTING ON THE TABLE?
Computer: THE LARGE GREEN ONE WHICH SUPPORTS THE RED PYRAMID.
Person: IS THERE A LARGE BLOCK BEHIND A PYRAMID?
Computer: YES, THREE OF THEM: A LARGE RED ONE, A LARGE GREEN CUBE, AND THE BLUE ONE.
Person: PUT A SMALL ONE ONTO THE GREEN CUBE WHICH SUPPORTS A PYRAMID.
Computer: OK. (does it)
Person: PUT THE LITTLEST PYRAMID ON TOP OF IT.
Computer: OK. (does it)
Person: DOES THE SHORTEST THING THE TALLEST PYRAMID'S SUPPORT SUPPORTS SUPPORT ANYTHING GREEN?
Computer: YES, THE GREEN PYRAMID.
Person: WHAT COLOR IS IT?
Computer: BY "IT", I ASSUME YOU MEAN THE SHORTEST THING THE TALLEST PYRAMID'S SUPPORT SUPPORTS.
Computer: RED
Person: HOW MANY THINGS ARE ON TOP OF GREEN CUBES?
Computer: I'M NOT SURE WHAT YOU MEAN BY "ON TOP OF "IN THE PHRASE "ON TOP OF GREEN CUBES " .
DO YOU MEAN:
1 - DIRECTLY ON THE SURFACE
2 - ANWHERE ON TOP OF ?
Person: 2
Computer: THREE OF THEM.
Person: HAD YOU TOUCHED ANY PYRAMID BEFORE YOU PUT THE GREEN ONE ON THE LITTLE CUBE?
Computer: YES, THE GREEN ONE.

SHRDLU Dialog

Person: WHEN DID YOU PICK IT UP?
Computer: WHILE I WAS STACKING UP THE RED CUBE, A LARGE RED BLOCK AND A LARGE GREEN CUBE.
Person: WHY?
Computer: TO GET RID OF IT.
Person: WHY DID YOU DO THAT?
Computer: TO CLEAN OFF THE RED CUBE.
Person: WHY DID YOU CLEAR OFF THAT CUBE?
Computer: TO PUT IT ON A LARGE GREEN CUBE.
Person: WHY DID YOU DO THAT?
Computer: TO STACK UP THE RED CUBE , A LARGE RED BLOCK AND A LARGE GREEN CUBE.
Person: WHY DID YOU DO THAT?
Computer: BECAUSE YOU ASKED ME TO.
Person: HOW DID YOU DO IT?
Computer: BY PUTTING A LARGE RED BLOCK ON THE TABLE ; THEN LETTING GO OF IT ; THEN PUTTING A LARGE GREEN CUBE ON IT ; THEN LETTING GO OF THAT CUBE ; THEN PUTTING THE RED CUBE ON THAT CUBE; THEN LETTING GO OF THAT CUBE.
Person: HOW MANY OBJECTS DID YOU TOUCH WHILE YOU WERE DOING IT?
Computer: FOUR OF THEM.
Person: WHAT DID THE RED CUBE SUPPORT BEFORE YOU STARTED TO CLEAN IT OFF?
Computer: THE GREEN PYRAMID.
Person: THERE WERE FIVE BLOCKS TO THE LEFT OF THE BOX THEN.
Computer: NO, ONLY FOUR OF THEM: THE RED CUBE, TWO LARGE GREEN CUBES AND A LARGE RED BLOCK.
Person: PUT THE BLUE PYRAMID ON THE BLOCK IN THE BOX.
Computer: OK. (does it)
Person: IS THERE ANYTHING WHICH IS BIGGER THAN EVERY PYRAMID BUT IS NOT AS WIDE AS THE THING THAT SUPPORTS IT?
Computer: YES, THE BLUE BLOCK.
Person: DOES A STEEPLE
Computer interrupting: SORRY, I DON'T KNOW THE WORD "STEEPLE".

SHRDLU Dialog

Person: A "STEEPLE" IS A STACK WHICH CONTAINS TWO GREEN CUBES AND A PYRAMID.
Computer: I UNDERSTAND.
Person: ARE THERE ANY STEEPLES NOW?
Computer: NO.
Person: BUILD ONE.
Computer: OK. (does it)
Person: CALL THE BIGGEST BLOCK "SUPERBLOCK".
Computer: OK.
Person: HAVE YOU PICKED SUPERBLOCK UP SINCE WE BEGAN?
Computer: YES.
Person: WHY DID YOU DROP IT?
Computer: BY "IT", I ASSUME YOU MEAN THE SUPERBLOCK.
I CAN'T EXPLAIN A NON-EXISTENT EVENT.
Person: IS THERE ANYTHING TO THE RIGHT OF THE RED PYRAMID?
Computer: YES, FOUR THINGS: THE BOX , THE RED CUBE, THE BLUE PYRAMID AND THE BLOCK NAMED SUPERBLOCK
Person: THANK YOU.
Computer: YOU'RE WELCOME!

4-73