Robotics and Autonomous Systems

Lecture 19: AgentSpeak and Jason

Richard Williams

Department of Computer Science
University of Liverpool

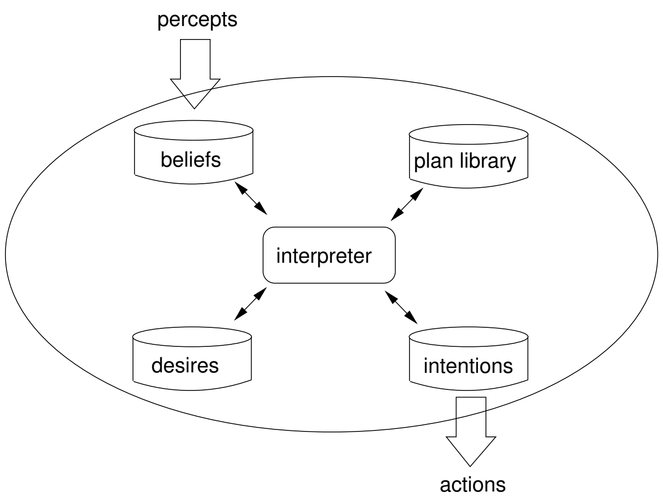
Today

- In this lecture we will begin to look at the tools that you will use for the second assignment:
  - AgentSpeak
  - Jason
- AgentSpeak is a programming language.
- Jason is an environment for building agents.
- They can be combined with Java/LeJOS for building robot controllers.

AgentSpeak

- AgentSpeak is a programming language for BDI agents
- It is an “abstract” programming language aimed for academic research to provide an operationalization of BDI theory
- Presented in 1996 by A. Rao
  - Rao, along with Mike Georgeff did a lot to popularise BDI within the AI world.
- It is based on:
  - the PRS architecture
  - BDI logics
  - Logic Programming (Prolog)
- Language of choice for the Multi-Agent Programming Contest

PRS



- The Procedural Reasoning System.

BDI Logics

- Logics that represent intentional notions:
  $Bel_i(\phi)$
  $Des_i(\phi)$
  $Intend_i(\phi)$
- Logics that encode the properties of these notions:
  $Bel_i(\phi) \wedge Bel_i(\phi \supset \psi) \supset Bel_i(\psi)$
- Logics that encode the relationships between these notions:
  $Intend_i(\phi) \supset Des_i(\phi)$
  $Intend_i(\alpha) \supset Bel_i(\alpha)$

Prolog

- Programming language based on first order logic.
  - PROgramming in LOGic
- Programs are statements in logic:

```
friend(X, Y) :- likes(X, Y).
likes(alice, bob).
```

- Queries are answered using logical inference:

```
friend(alice, bob).
```


Syntax of AgentSpeak

- There are three main language constructs in AgentSpeak:
  - Beliefs
  - Desires
  - Plans
- The architecture of AgentSpeak has four main components:
  - Belief Base
  - Plan Library
  - Set of Events
  - Set of Intentions

Beliefs

- Beliefs are simple Prolog programs.
- Two kinds of statement.
  - Facts
  - Rules
- Facts are statements about what the agent holds to be true.
- Rules are statements about relationships between facts.
  - Can think of them as allowing new facts to be created.

Example facts

- Atomic propositions
  `lecturer(richard)`
  `teaches_comp329(richard)`
- Propositions can be negated
  $\sim$ `young(richard)`
- The symbol $\sim$ should be read "not".

Example rules

- Rules look a lot like rules in Prolog.
- `child(X, Y) :- parent(Y, X).`
- Read a rule `a :- b` as "a, if b" or "if b then a".
- With facts, rules allow an agent to infer things.
- For example:
  `parent(bob,jane)`
  matches
  `parent(Y, X)`
  if `Y` = `bob`, `X` = `jane`
- The agent can infer
  `child(jane, bob)`

Example rules

- Rules are allowed to be more complex than this.
- For example:

```
grandparent(X, Z) :- parent(X, Y)
                     & parent (Y, Z).
```

- The "&" represents conjunction, and is what we usually mean by "and".
- So, given:
  `parent(eric, bob)`
  `parent(bob, jane)`
  the agent can infer:
  `grandparent(eric, jane)`

Example rules

```
grandparent(X, Z) :- parent(X, Y)
                      & parent (Y, Z).
child(X, Y) :- parent(Y, X).
son(X, Y) :- child(X, Y) & male(X).
daughter(X, Y) :- child(X, Y) & female(X).
parent(eric, bob)
parent(bob, jane)
parent(bob, david)
female(jane)
male(david)
```

- What can the agent infer?

Goals

- Goals represent states that the agent wants to bring about:

  Achievement goals
  `!learn(lejos)`
- Goals represent things the agent wants to know:

  Test goals
  `?teaches(richard,Module)`
  `?bank_balance(BB)`
- Test goals are goals in Prolog.
- Queries

More syntax

- The `teaches` in:

  `?teaches(richard,Module)`

  is a predicate
- Expresses a relation, or a property.

  `lecturer(richard)`
- The arguments of predicates are constants:
  - lower case, `bob`

  or variables:
  - uppercase, `Module, BB`

Events

- An agent reacts to events by executing plans.
- Events are changes in the:
  - beliefs; or
  - goals

  of the agent

Events

- AgentSpeak events are:
  - belief addition: `+b`
  - belief deletion: `-b`
  - achievement-goal addition: `+!g`
  - achievement-goal deletion: `-!g`
  - test-goal addition: `+?g`
  - test-goal deletion: `-?g`

Plans

- Plans are recipes for action.
- The context is a conjunction of special logical formulae defining when the plan is applicable.
- The body is a sequence of actions and sub-goals to achieve.

Plans

- An AgentSpeak plan has the following general structure:

  ```
  triggering_event : context <- body
  ```

  where
  - the triggering event denotes the events that the plan is meant to handle.
  - the context represents the circumstances in which the plan can be used.
  - the body represents the actual plan to handle the event if the context is believed true at the time a plan is being chosen
- When the trigger happens, test the context, and if it is true, then execute the plan.

Example plans

- A plan that responds to a change in belief.

  ```
  +green_patch(Rock)
     : not battery_charge(low)
     <- ?location(Rock,Coordinates);
        !at(Coordinates);
        !examine(Rock).
  ```

- When the belief green_patch(Rock) is added.
  (When you realise that the rock has a green patch).
- If battery charge is not low.

  Find the location of the rock.

  Go to that location

  Examine the rock.

Example plans

- A plan that responds to the addition of a goal.

  ```
  +!at(Coordinates)
     : not at(Coordinates)
       & ~ unsafe_path(Coordinates)
     <- move_towards(Coordinates);
        !at(Coordinates).
  ```

- To get to a set of coordinates.
- If not at the coordinates, and there is not an unsafe path to the coordinates

  Move towards the coordinates

  Reset the goal of being at the coordinates
- The recursive setting of the goal allows for plans that partially achieve the goal.

Plans

- So plans are a bit like STRIPS actions:
  - Preconditions
  - What you do

  but they also contain more than one action
- Plans are also a bit like STRIPS plans
  - Sequence of things to do

  but they also have preconditions and subgoals.

not and $\sim$

- In logical languages, especially ones related to Prolog, it is common to have two kinds of negation.
  - Strong, $\sim$
  - Weak, `not`
- One way to think of this is

Syntax	Meaning
ϕ	ϕ is true
$\sim \phi$	ϕ is false
`not` ϕ	The agent does not believe that ϕ is true
`not` $\sim \phi$	The agent does not believe that ϕ is false

where:
  - "is true/false" means "can be proved from its set of beliefs"
  - "does not believe" means "cannot prove from its set of beliefs".

not and $\sim$

- This is negation as failure (to prove).
- Related to the "closed world assumption" that we met before.
- "What I don't tell you is false."

not and $\sim$

- Reconsider our previous program:

```
grandparent(X, Z) :- parent(X, Y)
                        & parent (Y, Z).
child(X, Y) :- parent(Y, X).
son(X, Y) :- child(X, Y) & male(X).
daughter(X, Y) :- child(X, Y) & female(X).
parent(eric, bob)
parent(bob, jane)
parent(bob, david)
female(jane)
male(david)
```

not and ~

- These statements are true:
- son(david, bob)
  not son(bob, brian)
  not ~ son(bob, brian)
- These statements are not true:
  ~ male(david)
  not female(jane)

Actions

- Actions in AgentSpeak are symbolic representations of the actual actions the agent is supposed to do
  - For our NXT robots:

    `setSpeed(10)`,
    `rotateRight()`, or
    `goto(100, 200)`

    might be actions.
- The agent program will use these representations, while the interpreter
  - Jason in our case

  will hook these symbolic representations to the actual actions.
- For us, these will be methods in Java/LeJOS.

Actions

- Note that actions in an AgentSpeak program are logical statements.
- Their position in a plan means the interpreter can recognise them.
- In:

```
+!at(Coordinates)
   : not at(Coordinates)
     & ~ unsafe_path(Coordinates)
   <- move_towards(Coordinates);
      !at(Coordinates).
```

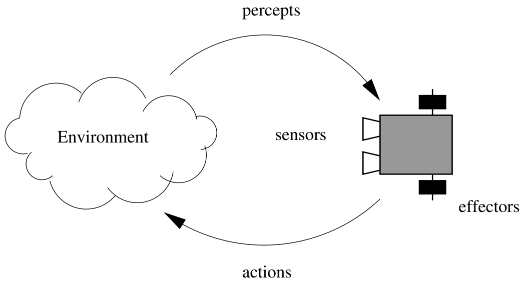
the statement `move_towards(Coordinates)` means make the call `goTo(float x, float y)`

Actions

- Some actions are internal and are prefixed by a "."

Environments

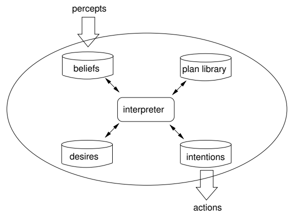
- When an agent program is executed, the agent needs to be connected to an environment.
- Environment provides the percepts and allows for actions.



- Often, the environment can be simulated before deployment.

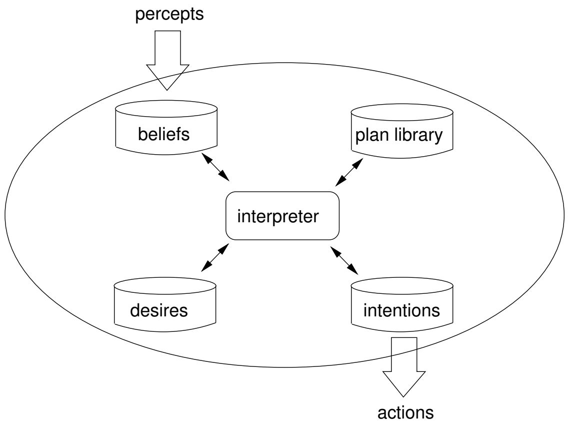
Jason

- Jason is an interpreter for a (richer) version of AgentSpeak implemented in Java.
- Developed by Jomi Hübner and Rafael Bordini over the last ten years or so.
- It enables a platform for the development of agents and multi-agent systems enabling hooks to call Java code



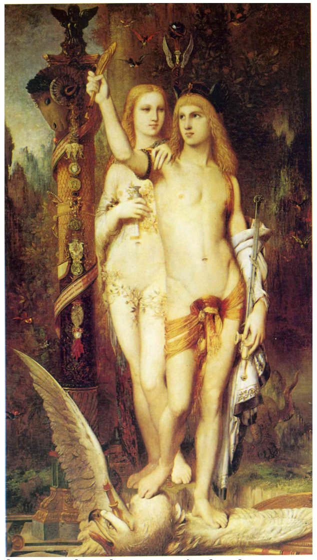
- http://jason.sourceforge.net/

Jason



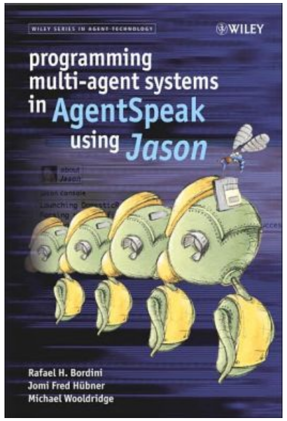
- Beliefs, desires and plans are all in AgentSpeak.
- Actions are calls to Java (and, in our case, LeJOS).

Jason



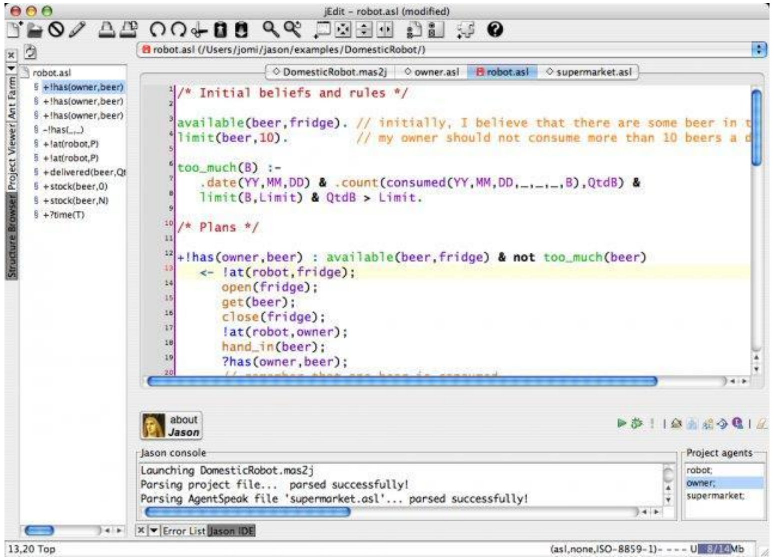
- Logo is Jason (of "Jason and the Argonauts") from a painting by Gustave Moreau.

Jason



- Jason comes with the editor jEdit
- There is also an Eclipse plugin

Jason



HelloWorld in Jason

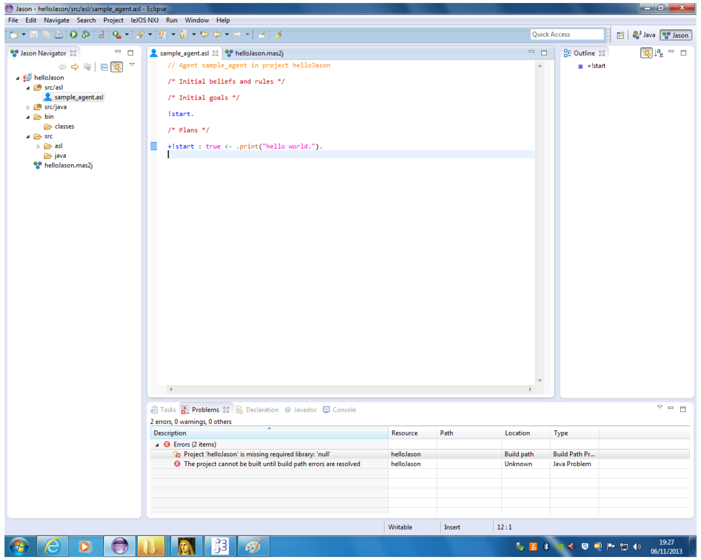
- Create a Jason project "helloworld", and you get:

```
MAS helloworld{

   infrastructure: Centralised

   agents:
      agent1 sample_agent;

   aslSourcePath:
      "src/asl";
}
```


Jason

- `infrastructure`: how the agent system is organised.
- `agents`: the list of agents that make up the system. Here there is just one.
- `aslSourcePath`: path from the MAS file to the agent descriptions.

Jason



Jason

- The agent looks like this:

```
/* Initial beliefs and rules */

/* Initial goals */

!start.

/* Plans */

+!start : true <- .print("hello world.").
```


Jason

- No initial beliefs or rules
- Only goal is the achievement goal `start`.
- The context/precondition for `start` is `true`.
- The plan for `start` is to print “Hello World”.

Summary

- This lecture introduced the syntax of AgentSpeak and discussed its main constructs:
  - beliefs
  - goals
  - plans
- It also introduced the Jason interpreter and produced a simple HelloWorld program
- We will look at more complex Jason programs next time.