Introduction to Jason

Introducing *Jason*

Jason is an interpreter for AgentSpeak programs, and a framework to create environments for the development and testing of multi-agent systems. You will not need to be familiar with the full power of *Jason*. In this assignment the environment is provided and you need only create simple plan-rules that govern the agent's behaviour.

The website for *Jason* is `http://jason.sourceforge.net` and it is the best place to start for finding documentation and example code. In particular the documentation section of the site contains links to the manual and a "getting started" tutorial, both of which are useful reference documents.

Setting up *Jason*

An IDE is provided for the easy editing of AgentSpeak programs, running the environment and debugging. To set it up simply follow the instructions below, taken from `http://jason.sourceforge.net/mini-tutorial/getting-started/` (which has more info, and pictures).

1. Download *Jason* from `http://jason.sourceforge.net`
2. Extract the archive to a folder in your userspace. Something like `tar xzf Jason-1.4.2.tgz` should do the trick.
3. Execute *Jason* by running `./jason.sh` from the `bin` directory.
4. You will probably need to setup the location of your Java JDK. Go to the menu Plugins→ Plugins Options→Jason. On a DICE machine, "Java Home" should be set to `/usr/lib/jvm/java/`

At this stage we encourage loading some of the examples and running them, to get an idea of how things work. The guides online may be of some assistance here.

Setting up the Environment

The steps below will take you through loading and running the assignment code:

1. Download the code from and decompress it, e.g. with `tar xzf abs1.tar.gz`.
2. Start *Jason*, and load up `infoSpeak.mas2j`.
3. Edit your code as appropriate, and run or debug with the buttons in the bottom right area of the IDE.

A Brief Overview of an AgentSpeak Agent

This is a very brief overview, for further information it is strongly recommended to read the *Jason* documentation and see the examples, and then attend the tutorial lecture.

The main components of an agent are beliefs, intentions and plans. Beliefs either come from percepts, and are updated automatically each reasoning cycle, or are added from plans. They are predicates, which may be of arity 0. An example of a position belief might be `pos(2,3)`. Plans are of the following form:

```
triggering-event : guard
     <- plan_step;
        plan_step.
```

`triggering-event` - Defines which events may initiate a plan. It could be internal, from plans, or external - from percepts. It can be addition, denoted by `+` or removal denoted by `-`. It may be either a goal or a belief. So for example `+!start` would trigger on the addition of a `start` achievement goal.

`guard` - A conjunction or disjunction of beliefs, logical formulae and certain built-in special predicates, such as `.random(N)`. An example of a guard depending on the position would be `pos(X,Y) & (X == 10)`. Note that variables must begin with an uppercase letter, lowercase or numbers are constant. "_" unifies with anything, and is the "don't care" symbol.

`plan_step` - There may be 0 or more steps, seperated by ; and terminated with a full-stop. These can be actions, in the form `do(action)`, or goals to achieve, in the form of `+!goal`.

Simulation Environment

Percepts

Percepts provided by the world are as follows:

`gsize(N,X,Y)` - the current grid is scenario N, dimensions X by Y (cells 0 to (X-1))

`depot(N,X,Y)` - Grid scenario N, and the gold depot is at X,Y

`pos(X,Y)` - the agent's current position

`carrying_gold` - this belief will be held if the agent has picked up gold. Note the agent can only carry one piece of gold at a time

`cell(X,Y,Object)` - if there's an object in any of the 8 adjacent cells, then these beliefs will hold. Object can be `obstacle`, `gold`, or `ally` for another agent (note that the agent can also perceive itself!)

Actions

The available actions are of the form `do(Action)` where `Action` is:

`up` - to move up (Y-1 - note 0,0 is the top left of the grid)

`down` - to move down (Y+1)

`right` - to move right (X+1)

`left` - to move left (X-1)

`pick` - to pick up some gold, only 1 piece can be carried at a time

`drop` - to drop some gold

Some Sample Code

Provided in the handout files is some sample code which you may find useful.

Helpful Links

- Jason homepage
- Jason Manual - dont bother too much with this
- Jason lecture slides from a course by Michael Wooldridge
- Other lecture slides from Faculdade de Ciencias e Tecnologia Universidade Nova de Lisboa