AI Large Practical

Alan Smaill

School of Informatics

Sep 25 2013

AILP: Assignment 1

- The first assignment is now on-line on the course web page.

  Today, we will look through what you are being asked to do as part of this assignment.

  As we saw last time, the general area is that of argumentation systems, and we will build on an existing Haskell implementation.

- While the assessment here is on your submitted programs, it will be a good idea to look at some of the associated literature now in preparation for part 2, which will be assessed primarily on a report of further extensions and experiments.

Starting point

- One approach to argumentation is with the Carneades framework, which by now has a fair sized literature; this is specifically designed for legal argumentation, and different standards of argument that are applied when reaching legal judgements.

- A bunch of material gathered together (look at sections 1–5):

  `http://www.sciencedirect.com/science/article/pii/S0004370207000677`

- And on argumentation in general, some slides by Besnard and Hunter are useful in giving a bigger picture:

  `http://www.ecsqaru.org/ECSQARU2007/elements.pdf`

  NB, they start from a more conventional logical approach.

Due dates

Reminder:

Assignment	Issue	Due	Weight
A1	25 Sep	Fri 25 Oct, 16:00	50%
A2	30 Oct	Thu 12 Dec, 16:00	50%

Initial system and paper

There is an account of the initial system in this paper:

`www.cs.nott.ac.uk/~bmv/Papers/tfp2012_abstract.pdf`

It describes concisely the the implementation; the code is not long, but does rely on some Haskell libraries.

Literate Haskell

The code is written in an extension of Haskell called *literate Haskell*. This allows a smooth presentation of the code in the paper mentioned above, and ensures that the code in the paper is actually the code of the system.

It also allows automatic generation of documentation, via the `haddock` tool:

`http://www.haskell.org/haddock`

. . . could be useful, though it is not essential to write your own extensions in this way.

Structuring system development

- It is good to split any even medium sized development into modules, and Haskell modules are well-suited to the task.
- You may decide to split the original program – at any event it is bad practice simply to add extra material in the body of that code.
- Better to work out what functionality the different modules should provide, and how they fit together.
- The gentle introduction:

  `http://www.haskell.org/tutorial/modules.html`

The assigment: building a front-end

In the code as given, the module `ExampleCAES.lhs` constructs in Haskell a particular argumentation configuration, and evaluates some arguments in that context.

To use Carneades to investigate a series of problems, and to set up experiments, we do not want to have to have a new module for each example, nor to express the arguments and other data directly in Haskell.

Thus the assigment asks you to develop a front-end to the system, so that argumentation data can be given in text files, which themselves can be used as input to the argumentation analysis engine.

Arguments with exceptions

The basic notion of argument that is considered has the following features:

- Propositions are just atomic statements, possibly negated. So they can be represented as strings, with a boolean flag.
- An argumentation step (or just argument) consists of a set of propositions as *premisses*, a further set of *exceptions* and a single *conclusion*.

  We read this as saying that *if* the premisses are all extablished, *and* none of the exceptions can be extablished, *then* the conclusion is justified.

  Furthermore, a *weight* is associated with each argument. (It is a good question to ask where these weights might come from in a real-life situation.)

Assignment Part 1

You should consider an input language for argumentation data — as it stands, the arguments are entered in the internal Haskell syntax, which is clumsy for writers and readers.

For a particular context, there are three sorts of argumentation data which the system makes use of:

1. Arguments, with weights
2. Assumptions
3. Standards of proof, associated with propositions under consideration.

Assignment Part 1 ctd

For the example in `ExampleCAES`, these correspond to:

Arguments	`mkArg ["kill", "intent"] [] "murder"`
	`mkArg ["witness"] ["unreliable"] "intent"`
	`mkArg ["witness2"] ["unreliable2"] "-intent"`

Assumptions	`mkAssumptions ["kill", "witness",`
	`"witness2","unreliable2"]`

Standard	`standard (_,"intent") =`
	`beyond_reasonable_doubt`

Note that:

- the arguments are elsewhere associated with weights
  they have (internal, Haskell) identifiers
- the code for `standard` also has a default value

Part 1 ctd

Thus you should design an appropriate format for such a collection of argumentation data.

Some choices:

- A single file for each case, or separate files for arguments, assumptions and standards?
- A formal grammar description, or an informal characterisation?

You will need to be able to parse your input data in order to convert it to a form acceptable to the argumentation analysis routines;
your choice of how to do this may affect your other choices here.

Testing

You should provide 3 test files with between 5 and 30 arguments, and indicate (in comments) what your system returns on selected queries.

It is also a good idea to include a README file in your directory; this will make the tester's life easier.

Next week

- Lecture at 9:00, same venue;
- Initial problems, assessment information, more pointers.
- Also drop-in sessions will start next week.