AI Large Practical

Alan Smaill

School of Informatics

Oct 4, 2017

AILP: Assignment 2

School of informatics

- The second 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 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 3, which will be assessed primarily on a report of further extensions and experiments.

Starting point

School of informatics

- 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	27 Sep	Fri 13 Oct, 16:00	0%
A2	4 Oct	Fri 10 Nov, 16:00	50%
A3	8 Nov	Wed 20 Dec, 16:00	50%

Initial system and paper

There is an account of Haskell version of initial system in this paper:

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

- Does a good job of distilling the Carneades framework into abstractions that lend themselves to implementation.
- `https://github.com/Smaill/carneades-1` is a reimplementation of the Haskell code in Python; still beta. Probably helpful for you to look at the Haskell version as well as the Python.
- If (when?) you find bugs in the Python, let me know; even better, fix the bug and we can incorporate the fix.

task 1: Richer notion of proposition

School of informatics

Carneades as you find it has a simple notion of proposition, like a Boolean proposition. You should allow more flexibility:

- Allow unary and binary predicates;
- Allow constants ("names");
- Allow quoted statements to be names.

The last might seem strange;
notice however situations where we want to reason using statements like:

The expert's domain is medicine.
The expert said the victim suffered heart attack.

School of informatics

- Initial docstring of module caes.py constructs a argumentation model — a Carneades Argument Evaluation Structure (CAES) — and evaluates some arguments in that structure.
- To use the model to investigate a series of problems, and to set up experiments, it would be convenient to load the information required for initialising a CAES from a file.
- Task 2: Develop extra functionality so that propositions, arguments and other configuration data required for a CAES instance can all be provided via text files.

Arguments with exceptions

School of informatics

Our basic notion of argument:

- Propositions are atomic statements, possibly negated; they can be represented as simple strings, with a boolean flag.
- An argument consists of a set of propositions as *premises*, a further set of *exceptions* and a single *conclusion*.
- We read this as saying that if the premises are all established, and none of the exceptions can be established, then the conclusion is justified.
- Furthermore, an audience is modeled as a set of assumptions, and an assignment of *weights* to each argument.
  - A premise is established if it belongs to the set of assumptions; some of the proof standards make appeal to the weights of arguments.

Assignment ctd

School of informatics

- Implement a string-based way of expressing argumentation data, rather than always having to call explicit constructors. Your implementation should support the grammar sketched in the assignment.
- You will need to be able to deserialise your input data into the CAES data structures.
- The aim is to make life easier for person trying to write the input data!
- Design choices:
  - add classes / functions to caes.py
  - separate module

Testing

School of informatics

You should provide 3 test files with between 5 and 30 arguments, and indicate (in comments) what your system returns on selected queries. Your test files do not need to use the richer notion of proposition mentioned above, but your system will be tested on input that does use that extension.

You need to provide a short script or other means that will allow your system to be run on DICE with a specific command:

```
% runCaes <testfile>
```

You will need to ensure that your system runs on DICE when run by someone else using the above command!
Hint: find someone else to run your code on DICE.

Next week

- No lecture;
- Drop-in sessions for queries will start.