AI Large Practical (2016–17)
Assignment 1

Alan Smaill

28th Sep, 2016

1 The assignment

Submissions for this assignment will receive comments when submitted, but will *not* count towards the final mark for the AILP.

Assignment 1 involves an implementation of a system for representing and evaluating arguments, where arguments are for or against a particular claim, backed up by some supporting evidence.

During this assignment, you should

- look at some presentations of the general area of argumentation systems (there are some starting places on the course page);
- look at an initial implementation of such a system, and aim to get an idea of how it works;
- Devise two argumentation scenarios (one may be similar to well-known examples; the other should be your own idea, more extended with say 20+ nodes.
- Give an explanation in English of the two examples, and
- run them interactively through the supplied Carneades system.

You should write these scenarios yourself. You are not permitted to

- copy other students' work;
- show your own work to other students;

but you are encouraged to have discussions with your colleagues.

Please see: `http://www.inf.ed.ac.uk/teaching/plagiarism.html`

2 Submission

Your are required to submit

- A description in English of the two argumentation scenarios, with comments where appropriate;
- a record of an interactive session with Carneades showing how the argumentation works in the system.

Please use the DICE submit command:

```
% submit ailp 1 <zip-file-of-your-project-directory>

            submit ailp 1 <your-directory>
```

The deadline for Assignment-1 submission is **16:00 on Tuesday 18th October 2016**.

3 Getting the original system

The following GitHub repository contains a Python implementation of the Carneades argumentation system:
`https://github.com/ewan-klein/carneades`
API documentation (using the Sphinx documentation package) can be found at
`http://ewan-klein.github.io/carneades/`
The implementation follows quite closely a Haskell implementation of the Carneades argumentation system, and you may find it useful to consult this:
`http://www.cs.nott.ac.uk/~psxvb/Papers/tfp2012_abstract.pdf`
– this contains a useful worked example.

The recommended method of getting the code from GitHub is to use git's clone command. Here's how it might look on a DICE machine:

```
% git clone https://github.com/ewan-klein/carneades.git
Initialized empty Git repository in .../ewan/carneades/.git/
remote: Counting objects: 407, done.
remote: Compressing objects: 100% (282/282), done.
remote: Total 407 (delta 131), reused 0 (delta 0)
Receiving objects: 100% (407/407), 740.90 KiB | 416 KiB/s, done.
Resolving deltas: 100% (176/176), done.
```

If you want to clone into a different directory, say `myproj`, do this:

```
% git clone https://github.com/ewan-klein/carneades.git myproj
```

However, assuming, that you've cloned into the default directory, `carneades`, you can `cd` into the Python package directory and try running the code. You'll need to use Python 3; the code has been developed with Python 3.4. If you have a peek at the `caes.py` module, youll see that the start of the file contains lines like this:

```
""
First, lets create some propositions using the :class:PropLiteral
constructor. All propositions are atomic, that is, either positive
or negative literals.
>>> kill = PropLiteral(kill)
>>> kill.polarity
True
>>> intent = PropLiteral(intent)
>>> murder = PropLiteral(murder)
>>> witness1 = PropLiteral(witness1)
>>> unreliable1 = PropLiteral(unreliable1)
>>> witness2 = PropLiteral(witness2)
>>> unreliable2 = PropLiteral(unreliable2)
...
```

This is a long 'docstring'; lines starting with >>> represent interactive Python commands. The doctest module can be used to execute them. This is a way to test the behaviour of the code, and to provide illustrative calls for documentation purposes. Towards the bottom of the bottom of this docstring, you will see how to initialise a Carneades Argument Evaluation Structure (CAES) and create the various components of a CAES. The `caes.py` module had logging set to `DEBUG` level, but you can easily comment this out at the top of the file if you want. The module also does some recursive call tracing so that you can get a better idea of what steps are involved in evaluating an argument in a CAES.

```
% cd carneades/src/caes
% python3.4 caes.py
DEBUG: Added proposition murder to graph
DEBUG: Added proposition -murder to graph
DEBUG: Added proposition intent to graph
DEBUG: Added proposition kill to graph
DEBUG: Proposition intent is already in graph
DEBUG: Added proposition -intent to graph
DEBUG: Added proposition witness1 to graph
DEBUG: Added proposition unreliable1 to graph
DEBUG: Proposition -intent is already in graph
```

```
DEBUG: Proposition intent is already in graph
DEBUG: Added proposition witness2 to graph
DEBUG: Added proposition unreliable2 to graph
Calling applicable([witness1], [unreliable1] => intent)
DEBUG: Checking applicability of arg2...
...
```

The `caes.py` module depends on igraph . A Python 3.4 compatible version of `igraph` has been installed on DICE. However, the plotting capability of `igraph` depends on Python bindings for the Cairo library, and these are not currently available on DICE.