Carneades Argumentation Evaluation System and Burden of Proof

Prepared for: Artificial Intelligence Large Practical, Assignment 2
Prepared by: Yashvir Surana, School of Informatics, The University of Edinburgh
18 December 2015
Student Number: s1368177

CONTEXT

Introduction & Theoretical Background

"Carneades is a formal, mathematical model of argument structure and evaluation which applies proof standards to determine the acceptability of statements on an issue-by-issue basis. The main original contribution of Carneades is its method for allocating the burden of proof to the proponent or respondent, for each premise separately, using premise types, proof standards and the dialectical status of statements." - [GPW2007], The Carneades model of argument and burden of proof, Thomas F. Gordona, Henry Prakkenb, Douglas Waltonc, 2007. Our extension follows the same ideology but implements the same principle by using a graph-like mechanism.

Trade-offs faced while deciding upon the structure of this extension included following the principles informally defined by Freeman and Farley, 2006 in the DART argumentation system. Although it does provide a heuristic flavour to the dialogue scheme, it wasn't the optimal solution to be used because systems based on DART cannot explicitly model argumentation theoretics as sufficiently required for this task as Carneades. [3]

Another trade-off was to whether or not use Toulmin structures for arguments. [4] However, Toulmin structures are the optimum representation because they represent both for and against arguments properly.

Verheij's work in [6] virtual arguments describes very efficiently, the usage of pro and con arguments along with warrants in the Toulmin sense. [5]

Critical questions are represented by assumptions and exceptions in our system. Showing that an exception holds in an argument disproves it and shifts the burden of proof, a crucial element of our system. Theoretically, merely asking a critical question is not enough to shift the burden of proof, it has to be backed up by another argument whose conclusion is the exception to the argument in question.[5]

For example, disproving whether a witness is reliable can be done by simply showing he has been unreliable in the past and if he has committed perjury once, what's to make the defendant believe that he's not doing it again?

DESIGN

Requirements of Assignment 1

The primary requirement of Assignment 1 was to design, implement and test a reader extension with file reading capability for caes.py (a python version by Ewan Klein of the carneades argumentation evaluation system) which would initialise a CAES by reading in a file as opposed to issuing separate python commands.
The secondary requirement was to devise an appropriate input syntax which would take into account propositions (both negative and positives), arguments, relative weights, audience assumptions, and associated proof standards. It should also have the capability for the user to add comments if desired.
The tertiary requirement was to write deserialisation code in python that would convert text into appropriate data structures that could be used by CAES. This should not only lead to initialisation but also perform error-checking and throw relevant messages for the user to be able to fix them if needed.[1]

Requirements of Assignment 2

The primary requirement of Assignment 2 was to extend the above system to permit arguments being made cumulatively one after the other i.e. in a dialogue form. Ideally, this exchange of dialogues should take place between a prosecution and defence, like in a legal trial. Both parties must bring evidence and after the introduction of these, a shifting burden of proof must be modelled. Ideally, the system should select the opening argument and the consequent arguments from a list of available arguments. Acceptability of a statement should be considered via the CAES method acceptable(). Level of abstraction shown in the table on the left from pp.888-890 of [GPW2007] is adequate and no distinction between burden of production and burden of persuasion is required.[2]

Proponent	Arg_id	Argument	Assumptions	*murder* accept?	BoP
P		*murder*		no	**P**
P	arg1	[*killing, malice*], ∼[§*187 excluded*] ⇒ *murder*	§187 *is valid*	yes	**D**
D	arg2	[*self-defense*], ∼[§*197 excluded*] ⇒ §*187 excluded*	§197 *is valid*	no	**D**
D	arg3	[*W1 'self-defense'*], ∼[*W1 not credible*] ⇒ *self-defense*	§197 *is valid*	no	**P**
P	arg4	[*W2 'time to run away'*], ∼[*W2 not credible*] ⇒ ¬*self-defense*	§197 *is valid*	no	**P**

Design Choices and Alternatives for Assignment 1

Alternatives included JSON, XML, using different markers to split the whole chunk of data into arguments, and then attributes of the arguments line-by-line, and creating a grammar-based lexer-parser as learnt in INF2B-Processing Formal and Natural Languages last year. JSON approach was chosen because it is the most elementary way to represent objects in my opinion and a case can be viewed as an object and its specifics: propLiterals, args, audienceAssumptions, weights, proofStandards can be viewed as characteristics of every individual argument. It is also more easy to read than XML in a text editor. While deciding the approach, I faced a major trade-off - JSON doesn't support comments natively. But the task requirement made it obligatory to include comments-functionality, so I decided to include a regular expression filter which removes comments before giving it to the JSON loader, essentially working like a minifier. Since I decided to input arguments enclosed as one entity enclosed by quotes and parts separated by '|', ',' and '&', I had to use the split function and create two helper functions which propLiteral-ified (convert to propLiteral) an entity or a list of entities. CAES initialisation steps were directly applied while processing the file to minimise run time.

Design Choices and Alternatives for Assignment 2

Firstly, regarding the overall structure of code, there were two approaches: 1) creating functions, structuring everything in a pedantic object oriented manner, or 2) coding everything in one continuous flow in one function. I chose the latter, because I personally thought that if the code is actually going to be used by a non-computer-scientist such as a lawyer, it might be easier for him to understand a line-by-line approach with comments at hand.

Secondly, the dilemma of choosing the introduction argument, I thought would best be dealt by choosing the root of the directed graph of all the arguments exploiting the indegree parameter. This way, one can choose what comes next significantly better. Arguments may not always be ordered in the optimal form, the system developed here takes care of that by sorting arguments in an order generating from the root or the opening argument. So the consecutive argument reinforces the preceding argument or disproves it. This is also dealt with in a depth-first manner. Since a graph-like approach was chosen, deciding which party is the proposer and burden of proof also highly depends on the notion that a premise of one argument could be the conclusion of another argument and vice-versa.
Lastly, the output of the table showing shifts of burden of proof at every argument being introduced was chosen to be in a table form in the console using characters, since installation of third-party libraries can be time-consuming just to show the same output in a slightly more aesthetic manner.

IMPLEMENTATION

Reader

The input file, structured according to the instructions in the README file, is opened in the reader.
Comments starting with // or enclosed in /* comment */ are removed from the file.
Resulting chunk of text is fed to a JSON loader and returns a dictionary.
A loop separates each argument and its various objects and characteristics, converts them into PropLiterals and stores everything in relevant lists.
The same loop also checks for ValueErrors, making sure that the data conforms with the syntax.
The same lists, easily indexable are now converted to data structures that can be accepted by CAES for an initialisation. (This step is skipped for the Burden of Proof extension)
CAES is initialised using the instructions followed from caes.py

Burden of Proof Extension

Directed graph of the arguments is used and indegree parameter exploited to find out the opening argument.
Remainder of the arguments are now ordered by following a pseudo-depth-first mechanism. For every premise of the opening argument, the implementation looks for an argument whose conclusion matches the premise, so it can be used to either reinforce or disprove the argument later. This is similar to iterative deepening search with depth of one level.
After the arguments are ordered, the implementation searches for arguments untouched by such a mechanism, usually which do not share an edge with the main graph vertices and appends them to the end.
A row with all the headers of the output table is declared and an opening ‘charge filing’ is appended.
In any legal case, prosecution always has the opening argument, and thus relevant arguments, proposer-id, and burden of proof is assigned.
Lists of data to be used by CAES called cumulatively after every argument is added are prepared.
A loop creates rows with all the information and CAES.acceptable() at every iteration and takes the number of elements from the above lists equal to the number of iteration. This facilitates the cumulative execution of CAES and data is added to the table.
In the above, arg_id, argument, assumptions, CAES.acceptable(chargeFiledInitially) are directly accessible and filled, but proposer of the argument and who has the burden of proof must be assigned newly at every iteration. This utilises a graph-based approach by using inter-twined lists and previous proposer to assign a current proposer. This in conjunction with the result of CAES.acceptable(chargeFiledInitially) is used to determine whether the burden of proof has shifted or is with the same proposer.

EVALUATION

Working Example

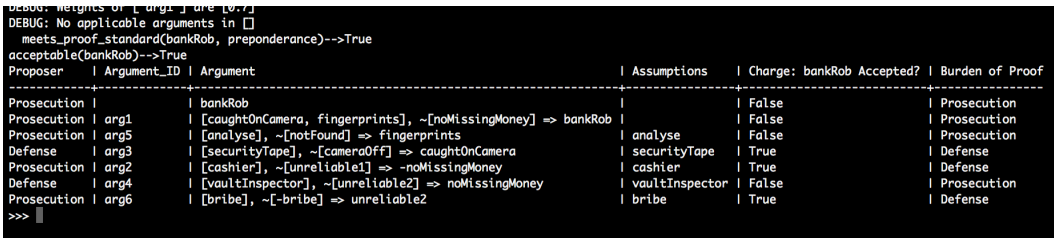
Below is a screenshot of a test run on a sample case as follows:

>>> from reader import Read

>>> c=Read()

>>> c.readFile('bankRob_case.json','bankRob')

We can see that the first row in the table is the initial charge filing and has no argument ID or assumption. Its a charge by prosecution and BOP lies with them. Second row represents the opening argument. To keep it realistic, we haven't assumed anything in this. Charge is still not accepted and BOP still lies with the prosecution. In the next argument (third row) presented fingerprints are analysed and although the charge is still not accepted and the BOP is still with prosecution, this evidence is casting reasonable doubt. Now, in an attempt to stop the case from being made, defence asks for further proof and security tape from camera comes to light (fourth row). This fulfils the two premises of the opening argument and therefore the charge is now acceptable and BOP has shifted to the defence. To weaken their (defence's) case, prosecution has brought in a cashier as a witness (fifth row) whose testimony reinforces the current state of the trial and BOP is still with defence. In the sixth row, the defence calls for the vault inspector to testify and since there is no missing money (a very strong point), bank robbery is invalid again and BOP is again shifted on prosecution. Prosecution has brought forward evidence (paper trail) of a bribe being offered to the vault inspector (seventh row), which as a critical question or an exception and thus renders the vault inspector's testimony incorrect and bankRob charge holds again with BOP shifted back to defence. This is how shifting of BOP takes place in our system.



```
DEBUG: No applicable arguments in []
  meets_proof_standard(bankRob, preponderance)-->True
acceptable(bankRob)-->True
Proposer    | Argument_ID | Argument                                                    | Assumptions    | Charge: bankRob Accepted? | Burden of Proof
------------+-------------+-------------------------------------------------------------+----------------+---------------------------+----------------
Prosecution |             | bankRob                                                     |                | False                     | Prosecution
Prosecution | arg1        | [caughtOnCamera, fingerprints], ~[noMissingMoney] => bankRob |                | False                     | Prosecution
Prosecution | arg5        | [analyse], ~[notFound] => fingerprints                      | analyse        | False                     | Prosecution
Defense     | arg3        | [securityTape], ~[cameraOff] => caughtOnCamera              | securityTape   | True                      | Defense
Prosecution | arg2        | [cashier], ~[unreliable1] => -noMissingMoney                | cashier        | True                      | Defense
Defense     | arg4        | [vaultInspector], ~[unreliable2] => noMissingMoney          | vaultInspector | False                     | Prosecution
Prosecution | arg6        | [bribe], ~[-bribe] => unreliable2                           | bribe          | True                      | Defense
>>>
```

The fact that we achieved the above result is tantamount to our implementation meeting its full functional requirements except error reporting (since the syntax was proper) and Burden of Proof being modelled successfully.

Error Reporting

Firstly, arguments not being properly entered and similar ill-entries raise ValueErrors like shown below

```
>>> c.readFile('case4.json','stoleCash')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/yashvirsurana/Desktop/AILP1/src/carneades/reader.py", line 85, in readFile
    raise ValueError('Please enter an argument in the form of -> conclusion|premise1,premise2 or conclusion|premise&exception for argument '+str(i+1))
ValueError: Please enter an argument in the form of -> conclusion|premise1,premise2 or conclusion|premise&exception for argument 3
>>> c.readFile('case5.json','stoleCash')
```

Secondly, data not being entered properly in JSON syntax also returns an error. This includes comments not being in accordance with the prescribed instructions as well.

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/yashvirsurana/Desktop/AILP1/src/carneades/reader.py", line 67, in readFile
    raise ValueError('Incorrect Syntax, Please check if the input is in a valid JSON syntax') #RETURN Error if input is not JSON
ValueError: Incorrect Syntax, Please check if the input is in a valid JSON syntax
>>>
```

NOTE: case1.json (fraud), case2.json (wikiReliable), case3.json (stoleCash), case4.json (stoleCash), case5.json (stoleCash) are cases made specifically for testing purposes and do not make any sense from legal point of view and burden of proof shifting will not work on them. For testing of burden of proof shifting, please use (bankRob_case.json, 'bankRob') and (murder_case.json, 'murder') for evaluation purposes.

CONCLUSION

The goal of this assignment was to model the notion of burden of proof, similar to real word legal dynamics where each side, namely the prosecution and the defence, introduce and exchange arguments in order to shift the burden of proof and prove the statement they are claiming to be true. The goal was achieved successfully as one can see in the Evaluation section. The key strengths and weaknesses of the system are shown below.

Key Strengths of the CAES - Extension with Burden of Proof Modelling

1. Effectively outputs a dialogical result with shifting burden of proof.
2. Resulting table displayed in console, hence no need to install new libraries.
3. Selects an opening argument.
4. Selects which argument to follow up / back up previous argument with.
5. JSON based input makes it easy for a lay man to input a case.
6. Added comments functionality in JSON based input.
7. Throws ValueErrors when input is not structured properly or doesn't conform with JSON syntax. (including comments)

Key Weaknesses of the CAES - Extension with Burden of Proof Modelling

1. Doesn't account for the distinction between burden of production and burden of persuasion. (Although this wasn't required in the task specification, it is still a weakness of the system)
2. Doesn't account for emotional sentiments behind legal arguments.
3. Doesn't account for cases that make no sense from a legal point of view (For example, case3.json), reads them but there is no shifting of burden of proof from prosecution.

REFERENCES

[GPW2007], The Carneades model of argument and burden of proof, Thomas F. Gordona, Henry Prakkenb, Douglas Waltonc, 2007. Available: http://ac.els-cdn.com/S0004370207000677/1-s2.0-S0004370207000677-main.pdf?_tid=164e40ba-9c86-11e5-bba2-00000aab0f26&acdnat=1449453721_2875c41996f11415fb2058c7b587b41f

[1] S. Anderson, "AI Large Practical Assignment 2," 2015. [Online]. Available: http://www.inf.ed.ac.uk/teaching/courses/ailp/assignments/assignment1-2015.pdf

[2] S. Anderson, "AI Large Practical Assignment 2," 2015. [Online]. Available: http://www.inf.ed.ac.uk/teaching/courses/ailp/assignments/assignment2-2015.pdf

[3] Freeman and Farley's DART System, 1996.

[4] Henry Prakken, "Logical Tools for Modelling Legal Argument", 1997. [Online]. Available: https://books.google.co.in/books?id=WxXvCAAAQBAJ&pg=PA263&lpg=PA263&dq=freeman+farley+dart+system&source=bl&ots=_rbLdI6axT&sig=8-nKOf4siSGAUTwQmJEmgyD_iO4&hl=en&sa=X&ved=0ahUKEwjVqf-O9-bJAhXEkY4KHavTDhgQ6AEIHjAA#v=snippet&q=DART&f=false

[5 The Carneades Argumentation Framework – Using Presumptions and Exceptions to Model Critical Questions by Thomas F. Gordon and Douglas Walton [Online.] Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.5067&rep=rep1&type=pdf

[6] Bart Verheij, Virtual Arguments, TMC Asser Press, The Hague, 2005.