AILP Report

Luca Scimeca

December 21, 2015

1 Introduction

This paper investigates argumentation theory from an artificial intelligence perspective, it proposes an hypothesis and describes the implementation of an algorithm for argumentation dialogs. The analysis is done through the extension of a system for argument evaluation, namely Carneades. Ultimately, a further discussion, a summary and conclusions are reported. The experiment was realized in the course Artificial Intelligence Large Practical (AILP).

2 Argumentation Theory

2.1 Background

Argumentation theory has become more and more of interest to artificial intelligence. To understand the field of argumentation theory it is fundamental to understand the meaning of *argument*.
In its most minimal description, an argument can be defined as a set of statements (propositions), made up of three parts: a conclusion, a set of premises and an inference from the premises to the conclusion [1]. Different approaches have been taken to model arguments for centuries. Initially, the conclusion for an argument was thought to follow directly from the premises. This model, although logically convenient, was demonstrated not to capture the full power of argumentation as we know it. Significant advancements were achieved in the past century when the initial issues were in part solved by taking a dialogical approach to argumentation (Hamblin, 1970). The advantage of this approach was in considering the context in which an argument was put forward.
The purpose of argumentation is ultimately asserting a proposition and being capable of supporting that proposition with logically valid arguments given the circumstances. Following this line a thought then, an argument is not a simple set of propositions, but rather a confrontation between two parties which doubt and *attack* each other arguments and conclusions. A game where each party attacks the proponent's claims has been shown to be sound a complete for a *credulus argument* [2].
Essential to understanding and analyzing arguments is diagramming. For the purpose of this report, diagrams will be composed of nodes, representative of arguments, and boxes, representative of premises and conclusions for the arguments; arrows connect boxes to nodes and vice-versa (see Figure 1).

2.2 Carneades

The Carneades system is based on the Carneades model of evaluation of arguments. The system takes a dialogical approach to argumentation, more specifically, it supports persuasion dialogs. In such dialogs, arguments are defeasible, i.e. they can be questioned [3]. Two participants, then, argue over the acceptability of their arguments, each trying to *persuade* the other of their claims. The structure of argumentations in Carneades is mainly captured by four elements:

1. Statements: declarative sentences in the system. A statement can be in one of four states: *stated*, *questioned*, *accepted* or *rejected*.

2. Premises: can be *ordinary* premises, *assumptions* and *exceptions*.

3. Arguments: mappings from premises to conclusion statements. They can be *pro* or *con* arguments.

4. Proof Standards: assigned to each statement, hold information on the evidence needed for the statement to be accepted.

To evaluate arguments within Carneades it is necessary to know the context in which statements are declared, that is, the *status* of the statement, its *proof standard* and a partial order on the arguments. A statement is *stated* when it is put forward by one of the parties in an argumentation. In this state, the acceptability of the statement needs to be assessed unless it was determined or assumed beforehand. When a statement is *questioned*, however, its acceptability needs to be assessed at all times. Arguments in the *accepted* or *rejected* states always retain the acceptability value accepted or rejected respectively, unless otherwise proved.
An important tool within the system allows for arguments to be drawn and visualized (see Figure 2)

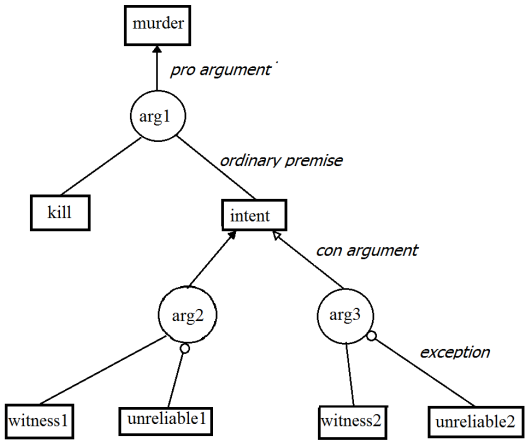

Figure 1: Example of a diagram in argumentation theory

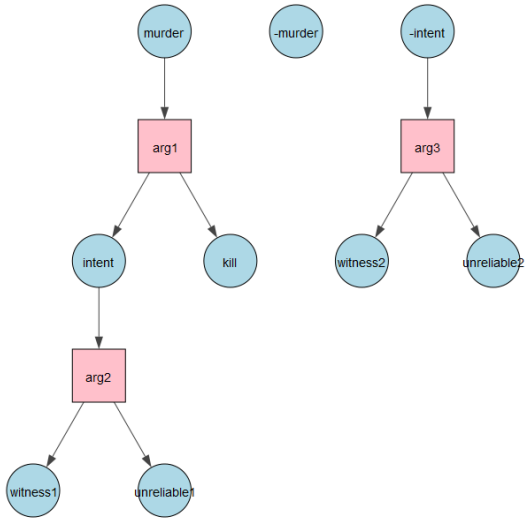

Figure 2: Example of drawn argument set in Carneades

3 Aims and hypothesis

The aim here is modeling a case where tho parties, a proponent and an opponent, use knowledge about the case to argue over a conclusion, represented as a proposition.
To make sense of the Carneades system and the implementation of the extensions, it is fundamental to understand the concept of *Burden of Proof* (BoP). The BoP is the requirement of a party to provide evidence that will change the conclusion from the current position (e.g. accepted to rejected or vice-versa).
Initially, then, the proponent puts forward the statement to be discussed and backs it up with some argument. At this point, if the conclusion of the argument put forward by the proponent is accepted the burden of proof shifts to the opponent, which now needs to provide evidence to make the main statement not accepted again. Following the dialogical aspect of

arguments, the BoP shifts are possible through the use of critical questioning. Questions, in fact, can be asked by either party to doubt the validity of any claims made by the opposition and may or may not need to be backed up by evidence. In the system, critical questions are modeled with assumptions and exceptions for each argument [3]. As the case unfolds, the BoP shifts signify which party needs to provide evidence at any given point. As the BoP is on a party with no more arguments to use, the case closes and the acceptability of the conclusion is decided.

4 Design Specification and Implementation

The requirements addressed in the Carneades expansion were the following:

1. Syntax design to state CAES cases.
2. CAES input reading from file.
3. Case structuring from inputs.
4. Dialog implementation.

4.1 Reading functionality

The input reading functionality in Carneades was implemented through the additional class *Reader* which was added to the *caes.py* python file. In the class, most of the reading functionality is performed by the *load()* method. The method uses the previously imported regular expression *re* module to break the input in three different sections containing *arguments*, *proof standards* and *assumptions* respectively. The syntax designed for the case files is similarly structured, favoring ease of case inputs by allowing the user to insert multiples entries in each of the three sections. Following the input split, the method is structured in three different sections which deal with each split respectively. The first split (i.e. arguments) is also subdivided into components which parse premises, exceptions and conclusions respectively. Each section of the method tokenizes the relevant part of the input syntax and converts the tokens to *propositions* through an external method in the class. As each section is tokenized and converted, all information is loaded in a CAES object and returned by the loading function with the argument set in the case.

4.2 Argumentation dialog implementation

The implementation of the dialog for arguments in Carneades is implemented in an additional python file, namely *argumentation.py*. The file contains a *main* and *make_step* method, performing the case dynamics, and five additional helper functions. As *argumentation.py* is executed, the main() method queries the user for the path to the case file to load in Carneades. Once the input is given, the case is loaded in a CAES through the Carneades *Reader* class (see 'Reading functionality' section) and a list of all possible propositions in the system is shown. The user can then choose which of the proposition should be argued over in the case, and as the value is inserted the case may begin.
The first and second step in a case are two special steps, i.e. they differ algorithmically from the others. The first consists of the Proponent simply putting forward the conclusion to be

argued over. Initially, as no argument is given for the stated proposition, this will never be accepted. The second step sees the Proponent giving an argument in support of the conclusion initially put forward. The main statement can now be queried for acceptability and if the proposition meets its assigned proof standard, the statement is accepted. The algorithm uses an helper function to choose the best argument in favor of the conclusion.
Arguments are picked given their weight in the case; an argument with a higher weight for the audience in a court, has in fact more chances to better support or counter the conclusion argued, and thus make the conclusion ultimately accepted or rejected.
From the third step onward a recurrent pattern for argument evaluation applies for both the Proponent and the Opponent in the case. If the BoP shifts to the Opponent as the Proponent makes his second step (i.e. the conclusion is now accepted), the Opponent needs to counter the move to try and shift the burden of proof back to the Proponent. Here, the Opponent can: attack the conclusion directly with an argument of its own, attack one of the premises of the arguments put forward by the Proponent in favor of the conclusion, or try to prove any of the exceptions to the same arguments, effectively providing enough evidence for those not to hold. Note that these options correspond to *asking* the Opponent critical questions to make the BoP shift (see 'Aims and Hypothesis' section).
After the second step then, the party with the BoP considers all possible propositions to attack, additionally to the propositions to further support their own arguments, and picks the proposition with highest weighted applicable argument, which reasonably gives the party the best chances to quickly shift the BoP. As the proposition is picked, the best applicable argument is applied and a new step is performed with the *make_step()* method. As a party runs out of arguments to put forward and can not counter a move, the final state of the conclusion is returned.
For each argument used by either party in the case, the premises and exceptions are initially assigned the lowest proof standard (i.e. *scintilla*), that is, they are accepted if they are supported by any applicable argument in the case graph. As the dialog proceeds and one of those propositions is attacked or backed up by an argument, the proof standard may change to whichever proof standard was defined in the loaded file. The dynamic change of the proof standard is coherent with the view of statements and critical questioning previously described.

5 Evaluation

5.1 Choice of tests

The *Reader* class added for the Carneades reading functionality was tested with python doctests. The tests are preformed through the use of multiple template files all with one or more errors. The files are loaded in the testing framework through the *Reader* object and the expected results for each error are matched against the actual outputs. The errors in the files cover syntax mistakes (e.g. colons, semicolons, commas, brackets etc), case errors (e.g. weight omission/out of bounds, multiple conclusions, wrong proof standard) and general loading functionality (e.g. exception throwing, correct internal CAES structuring, error feedback and others besides).

The argumentation dialog module was tested with python unittests. The tests are divided in two sections: one tests the helper functions used in the argumentation module to perform arguments evaluation and the other which tests the *main* and *make_step* function performing

the case dynamics. The testing of the dialogs in the second section is done through the use of various complete examples querying the decision making of the system at different points. The tests cover various situations where, the best choice at a given point through the case is: attacking the a conclusion directly, attacking one of the premises in support of an argument whose conclusion the party is trying to disproof, or proving one of the exceptions of the same. The tests also cover both situation where the Proponent ultimately wins or loses additionally to situations with multiple arguments in support of the same proposition, correct proof standard dynamic update and other general cases besides.
To test the module, the *main* method in the argumentation class is passed the file-path and the relevant proposition to prove, so to avoid user prompting. After the method generates all steps in the case, a dialog table containing the argument evaluation at each step and the relative BoP shifts is build, and matched against the expected table for the example. One of the tests in the set is hereby shown to demonstrate the working and testing of the system.

Sample test for dialogical argumentation: Simulated case of inappropriate photography.

Table 1: Dialog table for case of inappropriate photography

case step	current party	argument ID	argument	main statement accepted	BoP
1	PROPONENT	N/A	inappropriate_photography	False	PROPONENT
2	PROPONENT	arg1	[private_property, prohibited], [] => inappropriate_photography	False	PROPONENT
3	PROPONENT	arg4	[proof_of_sign], [unreliable_proof] => prohibited	True	OPPONENT
4	OPPONENT	arg2	[public_property], [of_people_in_privacy] => -inappropriate_photography	True	OPPONENT
5	OPPONENT	arg6	[government_ownership], [] => public_property	False	PROPONENT
6	PROPONENT	arg3	[secluded], [] => of_people_in_privacy	False	PROPONENT

Table 1 shows the expected dialogical table for the events in the case. The case should unfold as follows:

1. The Proponent puts forward the statement to be argued in the case i.e. *inappropriate_photography*.
2. The Proponent picks *arg1* in support of the statement previously put forward. The argument states that the conditions for the photos to have been taken inappropriately follows from two facts, namely: the pictures were taken in a private property, and taking pictures in such property was prohibited. As the proponent puts the argument forward the main statement changes its proof standard to the actual proof standard in the case i.e. *beyond_reasonable_doubt*. This simple argument does not shift the BoP since it is assumed by the court the property is private, but the proposition needs to give further evidence on whether it was indeed prohibited to take pictures in the property.
3. The Proponent, at this point, puts forward *arg4*, giving evidence of the prohibition with a proof of a sign. The proof is assumed to be valid and will remain so until further evidence can prove the opposite. Here, as the arguments so far provided result conclusive, the main statement is accepted and the BoP shifts to the opposition.

4. The Opponent decides to attack the main statement directly, given the most relevant argument for the audience is against that conclusion (i.e. it has the highest weight for proof standard evaluation). The party then puts forward *arg2* making the case that the area where the picture were taken was actually a public property and the pictures must therefore be considered appropriate, unless they were of people in privacy. This argument is not enough to make the BoP, given the Opponent must still prove the property was in fact public.
5. The Opponent backs up the previous claim by supporting the *public_property* premise with an argument stating how the property is actually owned by the government. This move is picked, from a set of moves for proving public ownership, as the strongest argument in support of the premise. At this point, the BoP shifts since it is no more the case that the defended party took inappropriate pictures *beyond reasonable doubt*.
6. The Proponent must now try to shift the BoP back to the Opponent and tries to do so attacking one of the exceptions in an argument put forward by the Opponent. The Proponent in fact tries to make the case that the pictures taken were indeed of people in privacy since they were *secluded* at the time of the shooting. Here, although the argument is valid, it is not enough to show the guilt of the opposition under the current proof standard, and as the Proponent has no more arguments, the Opponent wins the case and the main statement *inappropriate_photography* is not accepted by the court.

After the table containing the correct moves at each step for either party is created, it is matched against the table returned by the system and the result returned as the test result. This and many other similar cases test the various possible scenarios of a typical dialog.

5.2 Test results

The reading functionality implemented within the Carneades implementation passes all tests supplied to it. Given the extensive coverage of the tests, it is reasonable to assume the *Reader* class successfully meets all requirements initially set.

The unittest for the argumentation module cover both the decision making of the system through the helper functions and the final dialogical table with the dynamics of the case at the various steps.
The system passes all tests supplied. The parties (i.e. Opponent and Proponent) do pick the most logical argument at each step following the strategy intended for the system. Given the broad choice of tests, it is safe to assume the system behaves as intended, although further testing could more comprehensively cover the numerous possible situations in different case scenarios.
Note that the tests do not assess the system picking the argument that most quickly shifts the BoP in the system, rather whether or not the system meets the previously set standards. The dialogs then, should unfold with the parties picking arguments given the *strategy* designed for the purpose of the experiment (see 'Discussion and Conclusion' section for further details).

6 Discussion and Conclusion

The argumentation dialog implemented through the Carneades system successfully models persuasion dialogs, where two parties argue over a conclusion. In the system, the concept of BoP drives the *turns* of the two parties which each time put forward the best argument available to them to most quickly change the state of the main proposition argued.
The *best* argument is here picked through a *strategy* method witch evaluates *all* possible propositions to attack or support for a party, creates an ordered list of arguments for each of those and returns a strategy in a list form. The first element in the list corresponds to the proposition to attack (or support), and the argument to do it with. The strategy gives much importance to the weight of the arguments which, given the way propositions are evaluated against proof standards in *caes.py*, reasonably returns the arguments most relevant for BoP shifts. This approach, if reasonable, might commit parties to pick arguments which ultimately require additional steps to change the acceptability of the main statement; consider, for example, a case where high weighted argument needs several other additional arguments to support its premises.
Another approach could see the list returned arranged in a predetermined order, i.e. [exceptions-conclusions-premises], in which case attacking exceptions would strategically be assumed more efficient than, for example, attacking conclusions directly. Such an implementation, however, may similarly pick an exception to counter which ultimately requires several more steps than simply attacking, say, a related premise.
To make the one *best* decision at each step, knowledge of all following steps to the conclusion of the case is required. As each party considers arguments to put forward, then, the whole case to the bottom of the argument set must be unfolded and the argument which leads to a win (where a win is possible) with the least steps should be picked; conversely, if no win is possible, the argument leading to the opposition putting forward the highest number of arguments should be chosen. In this framework, however, argument evaluation would be all but efficient. Moreover, each party should know exactly what the other party will put forward for any given argument used in the case additionally to all arguments and evidence retained by the opposition beforehand; this model then, would not capture (or to a less extent) the reality of a persuasion dialog in real world situations. In a realistic scenario, the parties would in fact be unaware of all possible directions a case could take, and usually would have no knowledge of evidence and arguments the opposition intends to use. Under these conditions, the modeled system more consistently captures the reality of dialogical argumentations.

References

[1] D. Walton. *Argumentation Theory: A Very Short Introduction, in: Argumentation in Artificial Intelligence*, pages 1–22.

[2] G. Vreeswijk and H. Prakken. *Credulous and sceptical argument games for preferred semantics, in: proceedings of JELIA'2000, 7th European Workshop on Logic for Artifical Intelligence*, pages 224–238. Springer Berlin Heidelberg, October 2000.

[3] Thomas F. Gordon, Henry Prakken, and Douglas Walton. The Carneades model of argument and burden of proof. *Artificial Intelligence*, 171(10-15):875–896, 2007.

[4] Iyad Rahwan and Guillermo R. Simari. *Argumentation in Artificial Intelligence*. Springer US, 1 edition, 2009.

[5] Henry Prakken, Chris Reed, and Douglas Walton. Dialogues about the burden of proof. In *The Tenth International Conference on Artificial Intelligence and Law, Proceedings of the Conference, June 6-11, 2005, Bologna, Italy*, pages 115–124, 2005.