Efficiency and Explainability of Support Relations in Argumentation Frameworks

Zwierd Grotenhuis
Supervised by Henry Prakken
Artificial Intelligence
Department of Humanities
Utrecht University
z.grotenhuis@students.uu.nl
7.5 ECTS

June 22, 2020

Abstract

Discussions that happen daily in our lives can be formalized into abstract frameworks that help us determine the dialectical status of the arguments given in the discussion. This study aims to contribute to ongoing research about different types of extensions to the abstract framework introduced by Dung (1995). Specifically, it asks whether adding support relations increases efficiency or explainability when determining its status with relation to the old framework.
After making a program to automate the evaluation of such frameworks, the results have indicated that there is no such increase. However, other benefits may be gained by adding support relations or by researching different kinds of extensions to Dung's framework. This is an interesting topic for future research.

Contents

1 Introduction

1.1 Scientific Embedding

To understand the research question that this study aims to answer, some background information is needed. Dung (1995) studied the argumentation based inference mechanism we humans use, and tried to come up with ways to formalize this and implement that mechanism into computers. He came up with a very simple yet elegant way to display arguments and their relation to each other. He defined an *abstract argumentation framework* as a pair of arguments and the 'defeat' relation between them. In figure 1 (taken from Prakken, 2018), we can see some examples of what this argumentation framework looks like. This argumentation framework can then be extended by a *labelling*, where an argument is *in* if and only if all arguments defeating it are *out*, and an argument is *out* if and only if it is defeated by an argument that is *in*, as explained by Caminada (2006). This labelling is a way to determine the status of all arguments in an argumentation framework. If we are interested in the dialectical status of a single argument, we can do so through *argument games*. These will be explained in chapter 2.3. The advantage of using argument games to determine the status of an argument is that the entire AAF does not need to be evaluated.

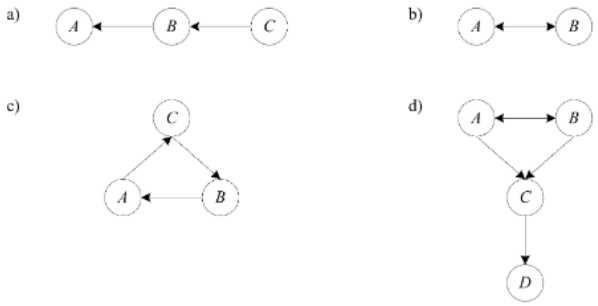


Figure 1: four argumentation frameworks

The reason we are studying this argumentation based inference mechanism in AI is that it is considered a very natural way for us to formalize the way humans argue. With proper inference mechanisms it is easier to determine which arguments are justified if it is not that clear at first sight.

In the example in Figure 2 there are no other ways to label according to our definition of in and out, but as frameworks become increasingly complex, it turns out that it is possible for frameworks to be labelled in multiple ways. For our research, we are interested in the grounded semantics. The grounded semantics means we are looking to determine a labelling where as few arguments are labelled in as possible, with respect to the subargument relation. The nice thing about grounded semantics is that there is always exactly one labelling possible.

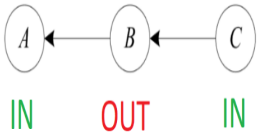


Figure 2: example of labelling

Since these frameworks can get quite large and complex, research has been done to automatically determine the dialectical status of an argument by numerous researchers, such as Vreeswijk (2003). The framework introduced by Dung has also been extended by Cayrol and Lagasquie-Schie (2005). They introduced notions of support relations (*bipolar argumentation frameworks*) between arguments in abstract argumentation.

1.2 Research Question

During this study we will be trying to find out if this extension of the basic argumentation framework will lead to any new insights in labelling. More specifically, the extension of basic argumentation frameworks might lead to more efficiency in evaluation, as the algorithm might discard tree paths before they would be discarded without the support relation. Maybe the new extension can lead to a form of *lazy evaluation*, so our program can go through large argumentation frameworks in a smaller time. The extension might also lead to more insight in the reasoning behind the evaluation of a framework.
This leads us to the following research question: *"What is the effect of adding support relations to the argumentation framework introduced by Dung on the explainability and efficiency behind evaluating its dialectical status?"*

1.3 Method

Before answering the research question, we need to do some research in existing literature about ways to determine the dialectical status of an argument. Then we will answer the research question using this information, by writing a program that can parse a given argumentation framework with support relations, and will give us the status of a given argument, as well as how the program has decided on its outcome. In order to do this we will need to change some rules of the argument games as defined by Vreeswijk and Prakken (2000), to encompass our extension with support relations. We will then compare the reasoning behind the evaluation with and without the support relations, and see if there are any notable differences in number of reasoning steps. We can then conclude if adding support relations has any (positive) effect on the reasoning done by our program. If there is time, we might also research the effect of adding support relations on the preferred semantics, but initially we will focus on the grounded semantics.
The paper can be structured into the subjects:

1. Labelling
2. Argument game
3. Programming
4. Results
5. Conclusion

2 Terminology

As a starting point for our research I will take some definitions from the literature, and I will give a recap of the definitions that are important to our research.

2.1 Argumentation Frameworks

Dung (1995) has introduced widely used formal definitions of argumentation frameworks and related concepts.

An abstract argumentation framework (AAF) is a tuple $< A, Def >$ consisting of a set A of arguments and a binary defeat relation ($Def \subseteq A$ x A). An argument A defeats argument B if and only if $(A, B) \in Def$, and that A strictly defeats B if and only if A defeats B and B does not defeat A. An argument set S defeats an argument A if any of the arguments in S defeat A.

We can label each argument in an AAF with *in* or *out*. An argument is labelled *in* if and only if all arguments defeating it are *out*, and an argument is labelled *out* if and only if some argument defeating it is *in*.

In this research, we are mainly focused on the grounded semantics, which means we are aiming to minimize the set of arguments labelled *in* in our argument set. In the grounded semantics, an argument A is:

1. *Justified* if and only if A is in.
2. *Overruled* if and only if A is not in in but defeated by an argument that is in.
3. *Defensible* otherwise.

Important to note is that in the grounded semantics, there is always exactly one unique labelling.

2.2 Support Relation

Cayrol and Lagasquie-Schie (2005) have extended Dung's frameworks, and introduced the *Bipolar Argumentation Frameworks* (BAF). These frameworks are similar to the AAF, adding a support relation S. This leads to the following definition:

A bipolar argumentation framework (BAF) is a tuple $< A, R, S >$ consisting of a set A and a binary attack relation (R $\subseteq$ A x A) (referred to as *defeat relation* in Dung's AAF), and a binary support relation (S $\subseteq$ A x A) such that if A supports B and C attacks A then C attacks B.

Support relations are not transitive by definition.

From a set of attack and support relations, complex attacks are obtained. These complex attacks are divided into supported attacks and secondary attacks.

- A supported attack from A to B is a sequence $A_1\ r_1\ A_2\ ...\ A_{n-1}\ r_{n-1}\ A_n$, with $n \leq 3$, where $A_1 = A$ and $A_n = B$, such that $r_i = S(1 \leq i \leq n-2)$ and $r_{n-1} = R$.
- A secondary attack from A to B is a sequence $A_1\ r_1\ A_2\ ...\ A_{n-1}\ r_{n-1}\ A_n$, with $n \leq 3$, where $A_1 = A$ and $A_n = B$, such that $r_1 = R$ and $r_i = S(2 \leq i \leq n-1)$.

Informally, this means that a supported attack is a path through a bipolar interaction graph where the first series of arguments are support relations, followed by an attack relation. A secondary attack is an attack followed by a number of support relations.

Every BAF can be reformulated as an AAF with the addition of a secondary attack, and vice versa. Given arbitrary arguments A, B and C where A supports B and C attacks A, then given the definition of the *secondary attack*, we can rewrite this into an AAF where C attacks A and C attacks B.

2.3 Argument Game

Prakken (1999) proposed an argument game to decide the dialectical status of an argument in the grounded semantics. The idea is an argument game between two players, a proponent and an opponent, where the proponent starts with an argument that needs evaluation. The opponent will then try to defeat this argument, after which the proponent will try to do the same, and so on. The winner is the player who makes the last move. Whoever has the winning strategy decides the dialectical status of the argument. The player with the winning strategy is the player that can win given any move the opponent makes.

To decide if an argument is in the grounded extension (if the argument is in in the grounded semantics, it is part of the grounded extension), the opponent is favored. His moves are allowed to be defeating, while the proponent's moves have to be strictly defeating. The proponent is also not allowed to repeat his moves, and backtracking is not allowed for either player. A player wins the argument game if and only if the other player has no legal moves.

This can be formalized into the following ruleset:

1. The opponent's moves have to defeat the proponent's argument
2. The proponent's moves have to strictly defeat the proponent's argument
3. The proponent may not repeat his moves.

4. Backtracking is not allowed for either player.
5. A player wins the argument game if and only if the other player has no legal moves.

The argument game with these rules satisfied is called the *G-game*. This game is both complete and sound for the grounded semantics, as Prakken has proven.

3 Labelling

One of the ways to determine the status of an argument is using labelling. To implement the support relation into our AAF, we would need to change our definitions of labelling. We will focus on implementing the secondary attack in our research.

3.1 Secondary attack

A good new definition for labelling of an AAF with secondary attacks has been given by Prakken (2013). An argument A is labelled *in* if and only if

1. All arguments that defeat A are labelled *out*, and
2. All arguments $B_1, ..., B_n$ that support A are labelled *in*.

An argument is labelled *out* if and only if

1. Some argument that defeats A is labelled *in*, or
2. Some argument B that supports A is labelled *out*.

4 Argument Game

We would also need to change our definitions for the argument game for the secondary attack.

4.1 Secondary attack

Here is the ruleset of the new argument game, now with the secondary attack.

1. The opponent's moves have to defeat the proponent's argument OR an argument that supports the proponent's argument.
2. The proponent's moves have to strictly defeat the proponent's argument OR an argument that supports the proponent's argument.
3. The proponent may not repeat his moves.
4. Backtracking is not allowed for either player.
5. A player wins the argument game if and only if the other player has no legal moves.

5 Programming

Our program needs to determine the dialectical status of our arguments. This can be done via argument games or via labelling. Since the argument game has been used more in literature, and argument games are more efficient for determining single arguments, we will also use this method of deciding the status of our arguments. First we will need to implement the basic argument game for grounded semantics, which we will do in C#, and after that we can look at changing our code so that support relations are also included.

5.1 Input

Our program will take a text file as input with a BAF. An example input file has been shown below, on the left side, where the right side are comments to explain what is meant (not included in the input text file).

4	Number of arguments
(A, B, C, D)	Arguments
2	Number of attack relations
$(C; A, B; A)$	Attack relations
1	Number of support relations
$(A; D)$	Support relations

After this file has been read, the user will be asked what argument has to be evaluated.

5.2 Structure

The program has defined the following classes:

- Argument, which has a title
- Relation, which contains two arguments and the type of relation (i.e. attack or support)
- DisputeTree, which contains a root node for our dispute tree
- Node, which contains an argument and a list of child nodes.

We will be using *Depth First Search* to construct the *Dispute Tree*, which will contain all paths that can be taken, decided by the legal moves given the argument framework, starting at the argument that needs to be evaluated.

This algorithm uses the GetLegalMoves function, which is the one we have adjusted to take support relations into consideration. Inspiration is taken from Modgil and Caminada (2009), definition 8, to determine legal moves for regular AAFs.

```
if PropOnTurn
{
        find all strictly-defeating attacks on the argument given by OPP,
        add to list;
        find all direct supports on the argument given by OPP,
        add their attacks to the list;
        remove arguments already played by PRO;
}
else
{
        find all attacks on the argument given by PRO, add to list;
        find all direct supports on the argument given by PRO,
        add their attacks to the list;
}
```

From this dispute tree, we will determine if there are any winning strategies, by checking each argument's status. This is done with the recursive algorithm shown below. Informally, it starts at the root of the dispute tree, which is the argument that needs to be evauluated, and determines if this node leads to a guaranteed win for the proponent, given that he plays optimally. Notice that if it is the proponent's turn, any move that the opponent can do that does not lead to a winning path makes it a non-winning strategy, so we dont have to evaluate the other paths in the tree.
On the other hand, if it's the opponent's turn, a single move that the proponent does that leads to a winning path means the strategy might be a winning strategy, so we also dont have to evaluate all possible paths if we have already found a winning path.
.

```
public static bool CheckIfWin(Node node, bool onPropTurn)
{
        create stack currentPath;
        push node to currentPath;
        invert onPropTurn;
        if node has no children
        {
                if (onPropTurn)
                {
                        return true;
                }
                else
                {
```

```
                return false;
            }
        }
        else
        {
            foreach (var child in node.children)
            {
                bool childWon = CheckIfWin(child, onPropTurn);
                if (onPropTurn == true && !childWon) // If it's the
                proponent's turn, any move that the opponent can do
                 that does not lead to a win makes it a non-winning
                  strategy.
                {
                    return false;
                }
                if ((onPropTurn == false && childWon)) // If it's the
                 opponent's turn, a single move that the proponent
                  does that leads to a win means the strategy might
                  be a winning strategy.
                {
                    push node to currentPath;
                    return true;
                }
            }
            if (onPropTurn) // if all children are checked and there
             are no moves that the opponent can do that lead to a loss,
              then the strategy might be a winning stategy.
            {
                push node to currentPath;
                return true;
            }
            else // if all children are checked and the proponent
             cannot play a single move that leads to a win, this
             strategy is not a winning strategy.
            {
            pop currentPath;
            return false;
            }
        }
}
```

Once we have determined that a winning strategy is found, we have stored the path that lead to this winning strategy. The program will also give some input on how it has come to its conclusion, and based on this we can see if there's a difference between evaluating a BAF or a semantically identical AAF.

5.3 Output

Given the argumentation framework shown in *5.1 Input*, the output of the program will look something like this:

```
Choose the number corresponding to the argument you want to evaluate:
1: A
2: B
3: C
4: D
5: E
3
Evaluating argument C...
Possible moves for opponent:
D
Determining the status of D
Possible moves for proponent:
B E
Determining the status of B
Possible moves for opponent:
D C
Determining the status of D
Possible moves for proponent:
E
Determining the status of E
E is a winning move!
Determining the status of C
C is not a winning move...
Then B is not part of a winning strategy either.

_____________________________________________________

Determining the status of E
E is a winning move!
Then C may be part of a winning strategy!

_____________________________________________________

A winning strategy has been found!
C —> D —> E

Press any key to exit.
```

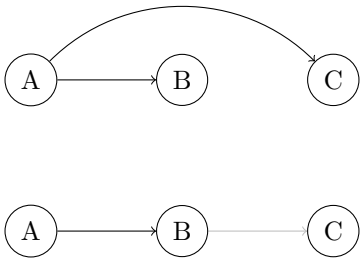
6 Results

The program has been run on the following AAF/BAFs. On the top part of the image, an AAF can be seen while the bottom has an identical BAF with support relations added. For each of the frameworks, the number of expanded nodes has been displayed, as well as whether the evaluated argument has a winning strategy (marked green and red). The number of expanded nodes relates to the number of calculations our program has to do, so a low amount of nodes means that our program can quickly decide the dialectical status of a given argument. With this number we can determine if there is any efficiency gain in adding support relations. Note that a black line indicates an attack relation while a grey line indicates a support relation. The red and green marking indicates whether the program has found a winning strategy. Of course, these should always be the same whether we are checking the BAF or AAF.

For every one of these BAFs, they have been rewritten into an AAF: If there is a support relation from argument A to argument B and a set of arguments S attacks A, we rewrote this so that S attacks A and S attacks B.

6.1 Simplest

Below we see the simplest possible AAF that can be rewritten into a BAF.

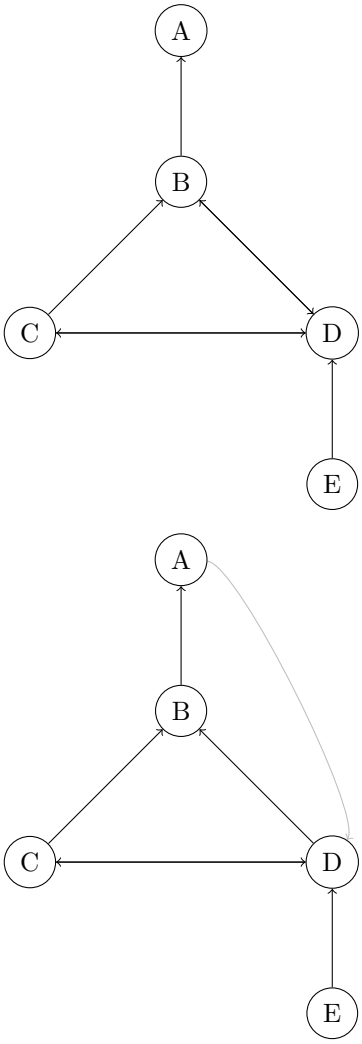


(1)

	A	B	C
AAF	2	1	2
BAF	2	1	2

6.2 Medium 1

Below we see a slightly more complex AAF and its BAF-counterpart. Inspiration is taken from Modgil and Caminada (2006), figure 3.

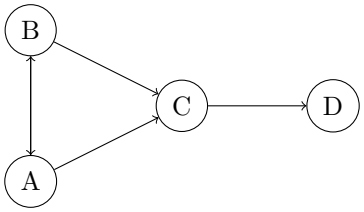


(2)

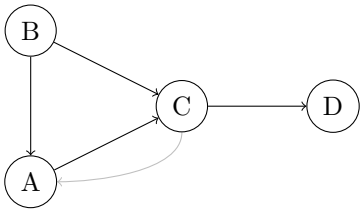
	A	B	C	D	E
AAF	5	4	3	2	1
BAF	19	4	7	10	1

6.3 Medium 2

Another slightly more complex AAF and its BAF-counterpart. Inspiration is taken from Prakken (2018)), example 4.2.9.



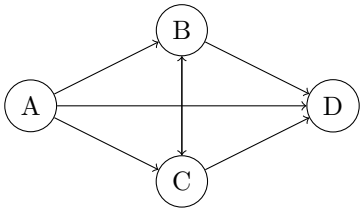
(3)



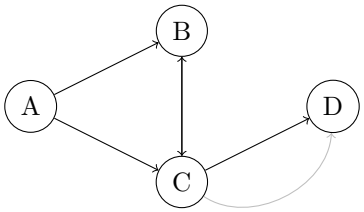
	A	B	C	D
AAF	2	2	2	6
BAF	2	4	4	8

6.4 Medium 3

Another slightly more complex AAF and its BAF-counterpart. Inspiration is taken from Prakken (2018)), excercise 4.8.11.



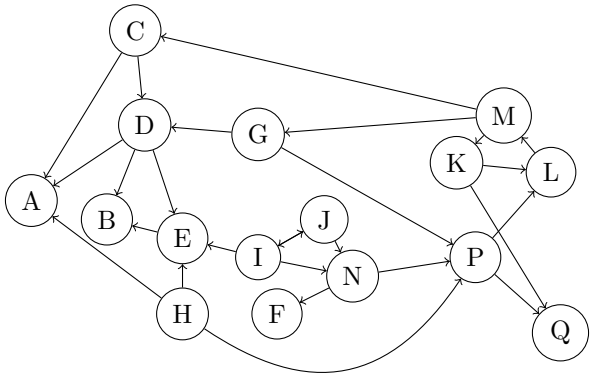
(4)



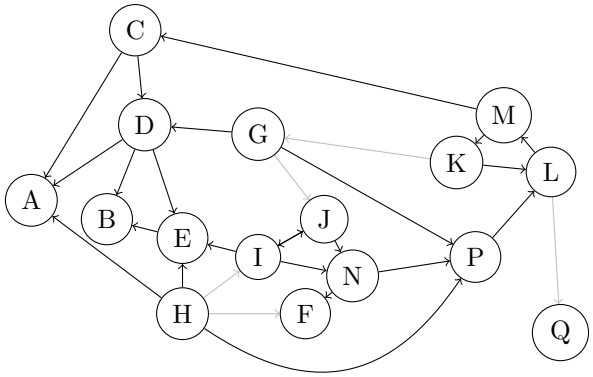
	A	B	C	D
AAF	1	2	4	2
BAF	1	2	4	4

6.5 Complex

A very complex framework, taken from Prakken (2018), figure 5.2, and again modified for our own purposes. For this complex framework, a timer has been implemented to see if there is any notable difference between the evaluation of the two frameworks.



(5)



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	P	Q
AAFNodes	14	30	14	14	2	6	14	1	2	2	12	12	12	2	6	14
AAFTime (ms)	64	121	45	52	5	16	54	4	7	5	41	39	41	6	20	54
BAFNodes	14	30	14	14	2	6	14	1	2	2	12	12	12	2	6	14
BAFtimes (ms)	54	143	50	49	5	18	48	4	8	7	40	40	40	4	25	48

7 Conclusion

To answer our research question *"What is the effect of adding support relations to the argumentation framework introduced by Dung on the explainability and efficiency behind evaluating its dialectical status?"*, we need to answer the two subquestions.

7.1 Efficiency

Something interesting can be seen in our results: it seems that the BAF actually has more expanded nodes for some of the argumentation framework, and evaluating an identical AAF has fewer or equal expanded nodes. For example, in 6.2 we see a difference in output. Let's take argument A. In both frameworks, the opponent can only play B. Here is a difference: in the BAF, the proponent is allowed to play D, as it strictly defeats B. In the similarly rewritten AAF though, B is not strictly defeated by D. This results in our program expanding the path to D first, where it will find out after a few steps into the game that this is not a winning strategy. Note that the fact that D can be played by proponent in the BAF is irrelevant for the evaluation of the program, as D is not part of a winning strategy because the opponent can play B to defeat D's supporter: B. It only makes it so that more nodes have to be expanded. This type of failure to detect a losing strategy early can be solved with extra checks. However, this would greatly increase overall computation time and is not more efficient.

Besides that, finding the legal moves for every support relation, we have to check all attack relations to see if there is a supporter that can be attacked. This makes it so that when adding 1 support relation, we have to do $n + n$ checks, where n is the number of attacks. Whereas if we would have rewritten to an AAF, we would only have to do $n + 1$ checks in total. This increase in computation time is not even taken into consideration in the expansion nodes shown above, but it can be very significant. It is interesting to see that, on the scale of 6.5's complex frameworks, this does not yet have a noticable effect on runtime. In future research it might be interesting to find out at which scale the difference becomes noticable.

Another thing to keep in mind is that sometimes the program will explore different paths first for the BAF compared to the AAF. If, for example, the opponent can play arguments A, B and C, exploring C first compared to A might lead to a faster conclusion that there is no winning strategy for this argument. This adds some variation to the number of expanded nodes that is tied to the BAF/AAF duo. This does not seem to have a significant effect on the argumentation frameworks we have explored.

All in all, one would think that the BAF actually is *less* efficient than our original AAF, even though the results only slightly back this up. It is certainly a

possibility that these effects are only noticable on extremely large argumentation frameworks, where there are thousands of arguments and even more relations between them.

7.2 Explainability

As far as explainability is concerned, our algorithm is very clear in which decisions it makes, and there is no real gain when adding support relations. The output is mostly the same, except for arguments that become strictly-defeating when a secondary attack is rewritten into two regular attack relations, at which point it takes more steps to discover that a path is not winning. This results in the output to become more convoluted and thus less easy to follow.

We can now answer our research question. Adding support relations has a negative efect on the explainability and efficiency behind evaluation. Based on this conclusion, support relations should be avoided when formalizing arguments and aiming for efficiency, unless the extra information that support relations offer are highly valued.

7.3 Future research

In future research it might be interesting to see how this conclusion changes when adding supported attacks or considering transitive support relations, since these all have an effect on the computation time but might also give new insights in the dialectical status of arguments. Adding these concepts would also make it easier to form an argumentation framework from real world examples, since there are more ways to relate arguments to eachother. Another thing to look into is how the extensions of the argumentation framework can lead to different evaluations in other semantics, such as the preferred semantics.

8 Literature

References

[1] H. Prakken, Argument. In S.O. Hanson & V.F. Hendricks (eds.): "Introduction to Formal Philosophy", *Springer Undergraduate Texts in Philosophy, Springer International Publishing AG*, pp. 63-79, 2018.

[2] Cohen, A., Parsons, S., Sklar, E. I. and McBurney, P., A characterization of types of support between structured arguments and their relationship with support in abstract argumentation, *International Journal of Approximate Reasoning, 94*, pp. 76-104, 2018.

[3] S. Modgil and M. Caminada, Proof theories and algorithms for abstract argumentation rameworks. *Argumentation in AI , I. Rahwan and G. Simari (eds)*, pp. 105-132, Springer-Verlag, 2009.

[4] Dung, P. M., On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence, 77(2)*, p. 321-357, 1995.

[5] Modgil, S and Prakken, H, The ASPIC+ framework for structured argumentation: a tutorial, *Argument and Computation 5*: pp. 31–62, 2014

[6] GVreeswijk, G.A.W, An algorithm to compute minimally grounded and admissible defence sets in argument systems. *Proc. 1st International Conference on Computational Models of Argument*, pp. 109–120, UK, 2006.

[7] Cayrol, C and Lagasquie-Schiex, M.C, On the acceptability of arguments in bipolar argumentation frameworks, *Proceedings of the Eight European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pp. 378–389, 2005.

[8] Vreeswijk, G.A.W and Prakken, H., Credulous and sceptical argument games for preferred semantics. *Proc. 7th European Workshop on Logic for Artificial Intelligence*, pp. 239–253, 2000

[9] Prakken, H, Dialectical proof theory for defeasible argumentation with defeasible priorities (preliminary report). *Proceedings of the 4th ModelAge Workshop on Formal Models of Agents. Springer Lecture Notes in AI, Springer Verlag*, pp. 202-215, 1999.

[10] Prakken, H, Relating ways to instantiate abstract argumentation frameworks. In K. Atkinson, H. Prakken and A. Wyner (eds.) From Knowledge Representation to Argumentation in AI, Law and Policy Making. A Festschrift in Honour of Trevor Bench-Capon on the Occasion of his 60th Birthday, *College Publications*, pp. 167-189, 2013.

[11] Prakken, H, On support relations in abstract argumentation as abstractions of inferential relations. *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI2 014)*, pp. -740, 2014.

9 Appendix

9.1 Code Implementation

Full source code and a compiled .exe file can be found in the attached file. Users can add their own frameworks in the .txt folder.