Imperial College London

Interim Report

Web-Based Argumentation

Author:
Michael Procopiou

Supervisors:
Dr. Xiuyi Fan
Prof. Francesca Toni

May 22, 2014

Contents

1 Introduction **3**
- 1.1 Incorporating derivation engines in a web application. 4
- 1.2 Visualising the derivation trees. 5
- 1.3 Building a Context Free Grammar for the input of the web application. 5
- 1.4 Grounding a framework over a domain. 5
- 1.5 Implementing derivations using Ideal Semantics. 6

2 Background **7**
- 2.1 Argumentation 7
- 2.2 Assumption-Based Argumentation 7
- 2.3 The ABA framework 8
- 2.4 Proving a claim using ABA 10
- 2.5 Computational mechanisms for ABA 11
- 2.6 Proxdd 12
- 2.7 Grapharg 13
- 2.8 ASPARTIX 15
- 2.9 Implementing Ideal semantics dispute derivations 16
- 2.10 Decision Making with ABA 18
- 2.11 Mapping AA to ABA 20

3 Building a web application interfacing the derivation engines. **22**
- 3.1 Overview of the Solution. 22
  - 3.1.1 Client 23
  - 3.1.2 Server 23
  - 3.1.3 Derivation Engines 23
- 3.2 User Interface with Application. 24
  - 3.2.1 Setting up the framework. 24
  - 3.2.2 Configuration of the engines. 24
  - 3.2.3 Visualisation the derivation tree. 24

3.3 Server - Processing the Input. 25
3.4 Server - Interfacing with the Prolog Engine. 25
3.4.1 Need for an interface between C# and Prolog Engines. 26
3.4.2 Using the SwiPIC.dll library as an interface. 26
3.5 Choosing the current implementation. 26
3.5.1 Advantages of current implementation. 27
3.5.2 Disadvantages of current implementation. 27
3.5.3 Proposed alternative (Prolog Web Server). 28

4 Creating an Input Language. 29
4.1 Creating a Context Free Grammar for the Input. 29
4.2 The need for a new input. 29
4.2.1 Formal definition of Input. 30
4.3 Parsing in the input. 30
4.3.1 Building a simple parser for the language. 30
4.3.2 Verifying the correctness of the input. 32
4.3.3 Forcing the existence of a contrary. 32
4.4 Example of an input. 32

5 Grounding a framework over its domain. 33
5.1 The need of specifying a domain for a framework. 33
5.2 Implementing a grounder. 34
5.2.1 Description of the algorithm. 34
5.2.2 Advantages of implementation. 36
5.2.3 Disadvantages of implementation. 37
5.3 Example of expected output. 37

6 Implementing Ideal Semantics dispute derivations. 38
6.1 Extending Proxdd to include ideal semantics. 38
6.2 Implementing ideal semantics. 39
6.3 Example of ideal derivation. 42

7 Project Plan 43
7.1 Planning Implementation 43
7.2 Project Timeline 44
7.3 Planning Future Extensions 45

8 Evaluating Project 47

Chapter 1

Introduction

Argumentation is increasingly becoming a popular aspect of Artificial Intelligence that deals with the creation of knowledge systems and evaluation of decision problems. Past and current research has allowed for the development of various argumentation frameworks that allow the analysis of arguments by formulating them in a formal manner and evaluating them. This project focuses mostly on Assumption-Based Argumentation (ABA) and its implementation through a practical argumentation system. ABA is a form of argumentation where arguments and attacks are notions derived from primitive notions of rules in a deductive system, assumptions and contraries thereof [9]. This framework allows for the evaluation of a conclusion based on whether it is supported by a "winning" set of arguments. Such sets of arguments and assumptions can be determined as "winning/acceptable" based on a number of different semantics that ABA supports, which are discussed in more detail later on in the report.

Therefore, ABA is a potentially powerful tool which one could use to establish the validity of the conclusion put forward. Early successful application of argumentation theory and the ABA framework have occurred in fields such as legal-reasoning, medical diagnosis and decision theory. Within these fields ABA has been used not only because of its ability to determine propositions as acceptable, but also due to its ability to convey the derivation process to the user. There are various computational mechanisms that have been devised to algorithmically compute the acceptability of a claim. Using these mechanisms argumentation engines, such as proxdd [8] and grapharg [3], have been implemented thus allowing for the computation of acceptability of a claim under a defined ABA framework. Nonetheless, these engines are still at a primitive stage and require enhancing for ABA to become a widely accepted and applied tool in the real world.

Namely, there is a lack of an application that is easily accessible to users

and provides both satisfactory performance and useful visualisations of the argument. The difficulties involved with creating such an application are multifold. These involve the performance of the underlying engine when dealing with large scale real problems, the difficulty a user might face in devising a correct ABA framework for their problem and devising a visualisation system to accurately convey the derivation process to the user. Additionally, the engines themselves are still at an early stage and need to be extended. For example the proxdd and grapharg systems lack the ability to derive the acceptability of a claim based on "ideal" semantics. Lastly, there is a lack of generalisation of these systems. At their current state they aim at simply resolving ABA frameworks directly implemented into the engine. This restricts the usability of the system to users who intend to use just ABA and have significant knowledge of ABA to device the initial framework of their problem.

The project's objectives centre around providing solutions to the problems mentioned above by fulfilling the requirement for a web-based argumentation application that will allow users to exploit the existing ABA systems. The web-application should act as a platform that will provide the users with good user experience and the ability to easily and as seamlessly as possible interact with the ABA systems in the back end. Having computed the necessary derivation the user will be provided with the output in a useful and meaningful manner in the form of a visualisation. The end system would be similar to the ASPARTIX system implemented by TU Wien [6]. Having established the web-application, the project will then seek to enhance the existing systems by enabling them to compute based on more semantics mentioned above. Potentially the project might look in the development of a simple API for these engines. Lastly, the web-application might be extended to support Abstract Argumentation through its mapping to ABA (as described by [7]) and allow the direct input of decision problems (as described in [11]).

1.1 Incorporating derivation engines in a web application.

We looked into designing a web application that would provide an online interface to the derivation engines. This includes a clean and simple interface by which the user is able to choose both the engine and the semantics required and would also allow them to define their ABA framework. Figure [TODO] provides a screen shot of the GUI used by the user to interact with the

derivation engines. The implementation and design choices are detailed in sections [TODO].

1.2 Visualising the derivation trees.

We then looked into a suitable output method of the solution received from the derivation engines. This involves the visualisation of the derivation tree on a canvas environment as illustrated in figure [TODO]. The visualisation provides a clear and concise method for the user to evaluate the derivation of the claim they submitted. The implementation of the visualisation is discussed further in section [TODO].

1.3 Building a Context Free Grammar for the input of the web application.

We formally defined a Context Free Grammar for the expected user input and built a parser for it. An example of a valid input is given in [TODO].

[TODO] - Add example of input.

In section [TODO] we elaborate on the semantics of our input language, along with how the parser forced the validity of the user's input.

1.4 Grounding a framework over a domain.

We then extended the language to allow for the specification of a domain over which assumptions, contraries or rules of our framework are valid. In example [TODO] we see how the notation is used to define a domain and the viable variable values.

[TODO] - ADD example.

We also built a grounder that grounds the framework over the domain specified by the user. The grounder acts as a pre-processing step and allows us to generate the grounded version of our frame work as in example [TODO]

[TODO] - grounder output program.

An overview of the impementation of the grounder is provided in section [TODO]

1.5 Implementing derivations using Ideal Semantics.

Using the existing implementation of the admissible based dispute derivations in Proxdd, we introduced the necessary functionality that enables us to derive dispute derivations based on ideal semantics. This includes the introduction and correct updating of the "F" set , along with the implementation of the Fail(S) check as described in section [TODO]. An example of a successful dispute derivation under the ideal semantics is provided in example [TODO].

[TODO] - Ideal example.

The details of the implementation are explored in section [TODO].

Chapter 2

Background

2.1 Argumentation

Argumentation in general involves studying how claims and conclusions can be reached by applying logical reasoning. It is useful when analysing arguments in the form of dialogue and persuasion, especially under the context of philosophy, medicine and law. Research in artificial intelligence is looking to exploit the underlying logic in analysing arguments to allow for this analysis to be carried out in a computerised manner. In order to do so general purpose argumentation frameworks have been devised to allow for analysis of an argument as a computerised argumentation problem. Two such argumentation frameworks are Abstract Argumentation (AA) and Assumption-Based Argumentation (ABA). Argumentation frameworks such as these can provide the general basis required to implement specific frameworks for real-life problems where argumentation could help in decision making and argument analysis. For the web-based application proposed by this project the underlying framework will be ABA.

2.2 Assumption-Based Argumentation

Although ABA is an instance of AA, there are core differences in the way ABA perceives and evaluates arguments. In ABA arguments and attacks are derived from given rules in a deductive system, assumptions and their contraries [9]. An argument is supported by assumptions and is attacked by other arguments when the contrary of an assumption of the initial argument can be deduced by the opposing argument. However, in order for an argument to be deemed "acceptable" or "winning", ABA is equipped with numerous semantics that allow us to determine an "acceptable/winning" set

of assumptions, from which we can deduce an "acceptable" set of arguments. Additionally several computational mechanism have been developed for ABA (see [8] and [3]) that allow a conclusion or claim to be algorithmically evaluate in terms of a "acceptable" set of arguments. The computational nature of these mechanisms allow us to programmatically implement the semantics and allow for computerised argumentation analysis of a framework.

2.3 The ABA framework

The potential power of ABA lies with its ability to represent and evaluate real-world decision problems. An implementation of ABA could prove to be an invaluable tool when it comes to evaluating potential claims, providing support to existing claims or disputing existing claims, in any context as long as it can be generalised and implemented as part of the ABA framework.

The following is an example taken from [9] to demonstrate how ABA can interpret a simple real-world problem, while introducing the ABA framework and its elements.

"You like eating food, this makes you really happy. But you are very health-conscious too, and don't want to eat without a fork and with dirty hands. You are having a walk with some friends, and your friend Nelly is offering you all a piece of lasagne, looking really delicious. You are not quite sure, but typically you carry no cutlery when you go for walks, and your hands will almost certainly be dirty even if the may not look so. Should you go for the lasagne, trying to snatch it quickly from the other friend? You may argue with yourself as follows:

α : let me go for the lasagne, it looks delicious, and eating it will make me happy!

β : but I am likely to have no fork and my hands will be dirty, I don't want to eat it in these circumstances, let the others have it!

α and β can be seen as arguments, and β disagrees with (attacks) α. If these are all the arguments put forward, since no argument is proposed that defends α against β , the decision to go for the lasagne is not dialectically justified. If, however, you put forward the additional argument

γ : who cares about the others, let me get the lasagne, I may have a fork after all

This provides a defence for α against β, and the decision to go for the lasagna becomes dialectically (although possibly not ethically) justified."

This example is illustrative of the breadth of problems that could potentially be analysed by an ABA system. Granted the example is overly simplistic. It has a limited amount of arguments and accounts for just a few

of the possible parameters that could affect whether we are happy or not. Nonetheless, being a general framework the simplicity of the problem does not affect our ability to device a specific framework for this problem from which we can evaluate our claim of being happy (or not being happy).

The ABA frame work is defined by [9] as follows:

Definition 1. An ABA framework is a tuple $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \bar{} \rangle$ where

- $\langle \mathcal{L}, \mathcal{R} \rangle$ is a deductive system, with $\mathcal{L}$ the *language* and $\mathcal{R}$ a set of *rules*, that we assume of the form $\sigma_0 \leftarrow \sigma_1, \ldots, \sigma_m (m \geq 0)$ with $\sigma_i \in \mathcal{L} (i = 0, \ldots, m)$; σ_0 is referred to as the *head* and $\sigma_i, \ldots, \sigma_m$ as the *body* of the rule $\sigma_0 \leftarrow \sigma_1, \ldots, \sigma_m$;
- $\mathcal{A} \subseteq \mathcal{L}$ is a (non-empty) set, referred to as *assumptions*;
- $\bar{}$ is a total mapping from $\mathcal{A}$ into $\mathcal{L}$; $\bar{\alpha}$ is referred to as the *contrary* of α.

Referring back to the example used let us assume that the following mapping exists:

- *p - happy*
- *r - not happy*
- *a - eating*
- *q - good food*
- *b - no fork*
- *c - dirty hands*
- *s - fork*
- *t - clean hands*

Based on this mapping and the definition of the an ABA framework we can now construct a framework that represents the example, as shown in example 1:

Example 1. $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \bar{} \rangle$ may consist of
$\mathcal{L} = \{a, b, c, p, q, r, s, t\}$
$\mathcal{R} = \{p \leftarrow q, a, q \leftarrow, r \leftarrow b, c\}$
$\mathcal{A} = \{a, b, c\}$
$\bar{a} = r, \bar{b} = s, \bar{c} = t$

The framework constructed in example 1 displays the various elements that allow one to represent a real-life decision problem in ABA. It can be seen that the rules related to our problem are defined in terms of inference rules. Additionally, the framework also includes the following elements:

Assumptions are, in our example, actions that can be taken or disputable observations about the argument. In general assumptions can be considered as vulnerable points of an argument that can be used to support or dispute that argument. The aim of ABA is to establish a set of "strong" assumptions that support or dispute the underlying argument.

Non-Assumptions represent goals we wish to achieve or avoid in accordance to the claim we are trying to analyse. These can be based on certain assumption we are considering or might be stand-alone goals.

Contraries are sentences that are opposing to the assumptions of our framework. ABA imposes that each assumption must be mapped to a contrary. For example, the assumption "no fork" in our framework has a contrary of "fork". Thus our framework is now equipped with both the assumptions necessary but also the opposing assumptions that can dispute them.

The description provides a simple overview of what these elements of the framework are and are used in the example. This will allow us to explore how claims are analysed. For the purpose of this report there is no need to explore to a deeper level the framework, however there are numerous publications that explain in these elements in more detail (see [9]).

2.4 Proving a claim using ABA

The ability of ABA to evaluate the support of a claim, lies in its ability to formulate and analyse arguments. Arguments are the deduction of a certain claim from the rules and assumptions provided. The following is a definition of an argument (according to [9]):

Definition 2. an argument is defined as follows:
An argument for (the claim) $\sigma \in \mathcal{L}$ supported by $A \subseteq \mathcal{A}$ ($A \vdash \sigma$ in short) is a deduction for σ supported by A (and some R $\subseteq \mathcal{R}$)

However, within the ABA framework there is also the notion of attack (similarly to AA). An attack between arguments under ABA occurs when one argument deduces the contrary of an assumption of another argument. Attacks can be defined as follows (from [9]):

Definition 3. an attack is defined as follows:
An argument $A_1 \vdash \sigma_1$ attacks an argument $A_2 \vdash \sigma_2$ iff σ_1 is the contrary of one of the assumptions in A_2.

ABA is equipped with various semantics that allow it to determine "winning/acceptable" sets of arguments and assumptions. This enables us to validate whether a claim can be deduced from these "acceptable" arguments and whether the claim is supported by these "winning" assumptions. These semantics can define the "acceptability" of both a set of arguments and a set of assumptions. Both approaches (or views) are defined below (in accordance to [9]):

Argument View Semantics

- *admissible* iff it does not attack itself and it attacks all arguments that attack it;
- *preferred* iff it is maximally (w.r.t. $\subseteq$) admissible;
- *sceptically preferred* iff it is the intersection of all preferred sets of arguments;
- *complete* iff it is admissible and contains all arguments it *defends*, where A defends α iff A attacks all arguments that attack α;
- *grounded* iff it is minimally (w.r.t $\subseteq$) complete;
- *ideal* iff it is maximally (w.r.t $\subseteq$) admissible *and* contained in all preferred sets of arguments;
- *stable* iff it does not attack itself and it attacks all arguments it does not contain.

Assumption View Semantics

- *admissible* iff it does not attack itself and it attacks all assumptions that attack it;
- *preferred* iff it is maximally (w.r.t. $\subseteq$) admissible;
- *sceptically preferred* iff it is the intersection of all preferred sets of assumptions;
- *complete* iff it is admissible and contains all assumptions it *defends*, where A defends α iff A attacks all assumptions that attack α;
- *grounded* iff it is minimally (w.r.t $\subseteq$) complete;
- *ideal* iff it is maximally (w.r.t $\subseteq$) admissible *and* contained in all preferred sets of assumptions;
- *stable* iff it does not attack itself and it attacks all assumptions it does not contain.

2.5 Computational mechanisms for ABA

The power of ABA is unleashed through the various computational mechanisms and tools that have been developed. These allow for ABA frameworks

to be analysed in an algorithmic manner, thus allowing the creation of computerised systems that can carry out this analysis. The underlying features of these mechanisms are as follows (as described by [9]):

- they can be abstracted away as disputes between a *proponent* and an *opponent*, in a sort of (fictional) zero-sum, two-player game: the proponent aims at proving (constructively) that an initially given *sentence* is "acceptable"/"winning", the opponent is trying to prevent the proponent from doing so;
- these disputes are defined as a *sequence of tuples* (referred to as *dispute derivations*), representing the state of the game while it is being played;
- these disputes interleave the construction of arguments and identification of attacks between them with testing whether the input sentence is "acceptable"/"winning";
- the rules of the game allow proponent and opponent to perform various kinds of *filtering* during disputes, but different kinds for different argumentation semantics (for determining whether the input sentence is "acceptable"/"winning";
- the possible outcomes of dispute derivations are as follows:
  - the input sentence is proven to be "acceptable"/"winning" (the dispute derivation is *successful*) or is not proven to be so; and
  - if the dispute derivation is successful, it returns:
    1. the set of all assumptions supporting the arguments by the proponent(referred to as the *defence set*),
    2. the set of all assumptions in the support of arguments by the opponent and chosen by the proponent to be counter-attacked (referred to as the *culprits*),
    3. in the case of the proposal of ([8]), the *dialectical tree* of arguments by proponent and opponent.

With these features in mind several procedure and algorithms have been devised that determine whether a claim is "acceptable/winning" based on several of the semantics mentioned. These include a wide range of tools from flowcharts to logic programming systems, some of which are discussed in the following sections.

2.6 Proxdd

Proxdd (refer to [2]) is a system developed by Dr. Robert Craven that implements dispute derivations to analyse ABA frameworks. Given a constructed

framework it can algorithmically analyse and return a set of winning arguments in accordance to the semantics specified. Proxdd currently supports admissible and grounded semantics. It is implemented using the Prolog logic programming language (for more details see [2]). The underlying computational mechanism used for Proxdd is the SXDD algorithm (defined in [8]).

Example 2. Example of input (from [2])

```
myAsm(a).
myAsm(b).
myAsm(c).
myAsm(d).
myAsm(e).
myAsm(f).
contrary(a, q).
contrary(b, f).
contrary(c, u).
contrary(d, v).
contrary(e, v).
contrary(f, v).
myRule(p, [a,u]).
myRule(q, [b,r]).
myRule(q, [c,s]).
myRule(q, [c,t]).
myRule(u, [a]).
myRule(s, []).
myRule(t, [d]).
myRule(t, [e]).
```

Example 3. Example of output (from [2])

```
DEF: [a]
CUL: [c]
ARG: [1:([a], [] -> p),6:([c], [] -> q),7:([a], [] -> u),
 8:([c], [d] -> q),9:([c], [e] -> q)]
ATT: [1-0,6-1,7-6,8-1,9-1]
```

2.7 Grapharg

Grapharg (refer to [1]) is a system also developed by Dr. Robert Craven. Similarly to Proxdd it also provides dispute derivation for ABA. Furthermore, it is also implemented in Prolog, however the underlying algorithm is

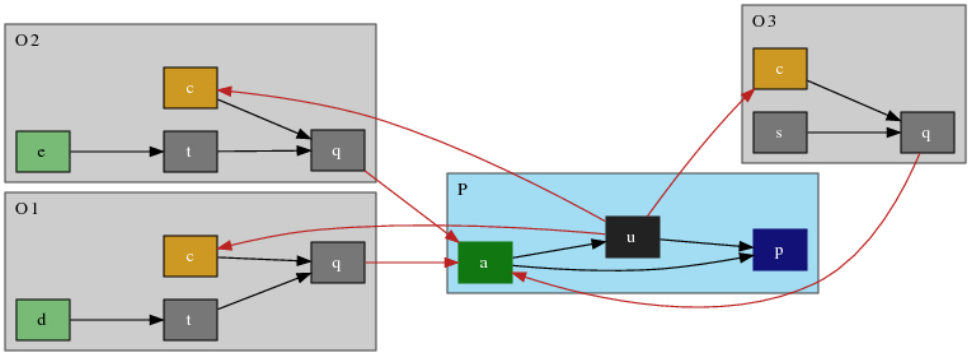
different. Unlike Proxdd it uses a graph-based dispute derivation algorithm that, according to the research's findings, optimises the computation of the dispute derivation (refer to [3]).

In both Proxdd and Grapharg the system takes in a framework in the form shown in example 2 and after the dispute derivation it produces an output similar to example 4. Additionally, by using graphviz, the output can be displayed graphically as in example 5.

Example 4. Example of output (from [1])

```
DEFENCE: [a]
CULPRITS: [c]
PROP JUSTIFICATIONS: [(a,*),(p,1),(u,5)]
OPP JUSTIFICATIONS: [[]-[(c,*),(d,*),(q,4),(t,7)]-[q],
[]-[(c,*),(e,*),(q,4),(t,8)]-[q],[]-[(c,*),
(q,3),(s,6)]-[q]]
ATTACKS: [(q,a),(u,c)]
GRAPH: []
```

Example 5. Example of current visualisation using graphviz (from [1])



However these systems are not user-friendly enough to promote ABA as an easily accessible tool. Both systems require setting up and both are operated using Prolog which many users might not be comfortable with. This means that the systems are not readily accessible to users and the interface provided could be more useful. This project aims at creating a web-application that will enable users to easily and instantly interact with the systems in an online and more user-friendly environment.

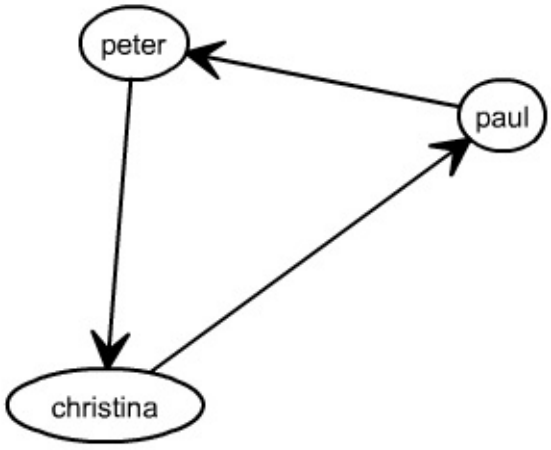
2.8 ASPARTIX

Perhaps the system that best resembles the desirable outcome of the project in terms of user experience and design is the ASPARTIX system (see [6]). ASPARTIX offers a web-application implementation that enables a user to enter a series of arguments and attack relationships between the arguments (see example 6), which is then processed and displayed diagrammatically to the user (see example 7). Our implementation should aim at providing a similar experience; allowing the user to implement their framework and displaying the results diagrammatically. However, ASPARTIX does not implement ABA instead it focuses more on Abstract Argumentation. Nonetheless, elements of the system such as its implementation as a web-application, the user's ability to define a framework and the existence of a canvas on which the visualisation is displayed will have to be integrated in this project's web-application as well.

Example 6. Example of input in ASPARTIX (from [10])

```
arg(peter).
arg(paul).
arg(christina).
att(christina, paul).
att(peter, christina).
att(paul, peter).
```

Example 7. Example of visualisation of ASPARTIX (from [10])



2.9 Implementing Ideal semantics dispute derivations

Both Proxdd and Grapharg currently do not support analysing ABA according to ideal semantics. Nonetheless, the computational mechanisms do exist (see [5]). By extending the engines with this additional computational mechanism, we will allow our web-application to provide analysis under the ideal semantics with minimal additional development effort.

Essentially this form of dispute derivation is an adapted version of that of the admissible semantics. Formally it is defined as follows (according to [5]):

Definition 4. Let $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \bar{} \rangle$ be an assumption based framework. Given a selection function, an *IB-dispute derivation of an ideal support* A for a sentence α is a finite sequence of tuples

$\langle \mathcal{P}_0, \mathcal{O}_0, A_0, \mathcal{C}_0, \mathcal{F}_0 \rangle, \ldots, \langle \mathcal{P}_i, \mathcal{O}_i, A_i, \mathcal{C}_i, \mathcal{F}_i \rangle, \ldots, \langle \mathcal{P}_n, \mathcal{O}_n, A_n, \mathcal{C}_n, \mathcal{F}_n \rangle$
where

$\mathcal{P}_0 = \{\alpha\}$ $\quad \mathcal{A}_0 = \mathcal{A} \cap \mathcal{P}_0$ $\quad \mathcal{O}_0 = \mathcal{C}_0 = \mathcal{F}_0 = \{\}$
$\mathcal{P}_n = \mathcal{O}_n = \mathcal{F}_n = \{\}$ $\quad A = A_n$

and for every $0 \leq i < n$, only one σ in $\mathcal{P}_i$ or one S in $\mathcal{O}_i$ or one S in $\mathcal{F}_i$ is selected, and:

1. If $\sigma \in \mathcal{P}_i$ is selected then

   (a) if σ is an assumption then

   $\mathcal{P}_{i+1} = \mathcal{P}_i - \{\sigma\}$ $\quad A_{i+1} = A_i$ $\quad \mathcal{C}_{i+1} = \mathcal{C}_i$
   $\mathcal{O}_{i+1} = \mathcal{O}_i \cup \{\{\bar{\sigma}\}\}$ $\quad \mathcal{F}_{i+1} = \mathcal{F}_i$

   (b) if σ is not an assumption, then there exists some inference rule $\sigma \leftarrow R \in \mathcal{R}$ such that $\mathcal{C}_i \cap R = \{\}$ and

   $\mathcal{P}_{i+1} = \mathcal{P}_i - \{\sigma\} \cup (R - A_i)$ $\quad A_{i+1} = A_i \cup (A \cap R)$ $\quad \mathcal{C}_{i+1} = \mathcal{C}_i$
   $\mathcal{O}_{i+1} = \mathcal{O}_i$ $\quad \mathcal{F}_{i+1} = \mathcal{F}_i$

2. If S is selected in $\mathcal{O}_i$ then σ is selected in S_u and

(a) if σ is an assumption, then

i. either σ is ignored, i.e.

$$\begin{array}{lll} \mathcal{O}_{i+1} = \mathcal{O}_i - \{S\} \cup & \mathcal{P}_{i+1} = \mathcal{P}_i & A_{i+1} = A_i \\ \{m(\sigma, S)\} & \mathcal{F}_{i+1} = \mathcal{F}_i & \\ \mathcal{C}_{i+1} = \mathcal{C}_i & & \end{array}$$

ii. or $\sigma \notin A_i$ and $\sigma \in \mathcal{C}_i$ and

$$\begin{array}{lll} \mathcal{O}_{i+1} = \mathcal{O}_i - \{S\} & \mathcal{P}_{i+1} = \mathcal{P}_i & A_{i+1} = A_i \\ \mathcal{C}_{i+1} = \mathcal{C}_i & \mathcal{F}_{i+1} = \mathcal{F}_i \cup \{u(S)\} & \end{array}$$

iii. or $\sigma \notin A_i$ and $\sigma \notin \mathcal{C}_i$ and

A. if $\bar{\sigma}$ is not an assumption, then

$$\begin{array}{lll} \mathcal{O}_{i+1} = \mathcal{O}_i - \{S\} & \mathcal{P}_{i+1} = \mathcal{P}_i \cup \{\bar{\sigma}\} & A_{i+1} = A_i \\ \mathcal{C}_{i+1} = \mathcal{C}_i \cup \{\sigma\} & \mathcal{F}_{i+1} = \mathcal{F}_i \cup \{u(S)\} & \end{array}$$

B. if $\bar{\sigma}$ is an assumption, then

$$\begin{array}{lll} \mathcal{O}_{i+1} = \mathcal{O}_i - \{S\} & \mathcal{P}_{i+1} = \mathcal{P}_i & A_{i+1} = A_i \cup \{\bar{\sigma}\} \\ \mathcal{C}_{i+1} = \mathcal{C}_i \cup \{\sigma\} & \mathcal{F}_{i+1} = \mathcal{F}_i \cup \{u(S)\} & \end{array}$$

(b) if σ is not an assumption, then

$$\mathcal{P}_{i+1} = \mathcal{P}_i \qquad A_{i+1} = A_i \qquad \mathcal{C}_{i+1} = \mathcal{C}_i$$

$$\mathcal{F}_{i+1} = \mathcal{F}_i \cup \{S - \{\sigma\} \cup R | \sigma \leftarrow R \in \mathcal{R} \text{ and } R \cap \mathcal{C}_i \neq \{\}\}$$
$$\mathcal{O}_{i+1} = \mathcal{O}_i - \{S\} \cup \{S - \{\sigma\} \cup R | \sigma \leftarrow R \in \mathcal{R} \text{ and } R \cap \mathcal{C}_i = \{\}\}$$

3. If S is selected in $\mathcal{F}_i$ then *Fail(S)* and

$$\begin{array}{lll} \mathcal{O}_{i+1} = \mathcal{O}_i & \mathcal{P}_{i+1} = \mathcal{P}_i & A_{i+1} = A_i \\ \mathcal{C}_{i+1} = \mathcal{C}_i & \mathcal{F}_{i+1} = \mathcal{F}_i - \{S\} & \end{array}$$

Fail(S) is computed as follows:

Definition 5. Let $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \bar{} \rangle$ be an assumption based framework. Given a selection function, a *Fail-dispute derivation* of a multiset of sentences S is a sequence $\mathcal{D}_0, \ldots, \mathcal{D}_n$ such that each $\mathcal{D}_i$ is a set of quadruples of the form $\langle \mathcal{P}, \mathcal{O}, A, \mathcal{C} \rangle$ where

$\mathcal{D}_0 = \{\langle S, \{\}, \mathcal{A} \cup S, \{\}\rangle\}$ $\mathcal{D}_n = \{\}$

and, for every $0 \leq i < n$, quadruple $Q = \langle \mathcal{P}, \mathcal{O}, A, \mathcal{C}\rangle$ is selected in $\mathcal{D}_i$ then either $\mathcal{P} \neq \{\}$ or $\mathcal{O} \neq \{\}$, and

1. If an element O from $\mathcal{O}$ is selected, then
   (a) if $O = \{\}$ then $\mathcal{D}_{i+1} = \mathcal{D}_i - \{Q\}$;
   (b) if $O \neq \{\}$ then let $\sigma \in O$ be the selected sentence in O:
      i. if σ is not an assumption then $\mathcal{D}_{i+1} = \mathcal{D}_i - \{Q\} \cup \{Q'\}$ where Q' is obtained from Q as in step (2.ii) of AB-dispute derivation;
      ii. if σ is an assumption then there are two cases:

      Case 1: $\sigma \notin A$. Then $\mathcal{D}_{i+1} = \mathcal{D}_i - \{Q\} \cup \{Q_0, Q_1\}$ where Q_0 is obtained from Q as in step (2.i.a) and Q_1 is obtained from Q as in steps (2.i.b) or (2.i.c) (as applicable) of AB-dispute derivation;

      Case 2: $\sigma \in A$. Then $\mathcal{D}_{i+1} = \mathcal{D}_i - \{Q\} \cup \{Q_0\}$ where Q_0 is obtained from Q as in step (2.i.a) of AB-dispute derivation;

2. If a $\sigma \in \mathcal{P}$ is selected, then
   (a) if σ is an assumption then $\mathcal{D}_{i+1} = \mathcal{D}_i - \{Q\} \cup \{Q'\}$ where Q' is obtained from Q as in step (1.i) of AB-dispute derivation;
   (b) if σ is not an assumption then $\mathcal{D}_{i+1} = \mathcal{D}_i - \{Q\} \cup \{Q'\}$ where there is a rule $\sigma \leftarrow R$ such that Q' is obtained from Q as in step (1.ii) of AB-dispute derivation.

The procedure defined above is explicitly explained in [5] and is used as the basis of the ideal semantics implementation. The web-application will now be able to analyse according to more semantics.

2.10 Decision Making with ABA

One of the most promising uses of argumentation is the potential of assisting decision making. This relies on being able to implement argumentation in a context that will allow it to compute the suitability of decisions. What makes argumentation exceptionally useful in the decision making context is that it not only proposes a decision, but it also provides the argumentation-based justification of it. This can be useful in real-life scenarios as it allows the user to understand the reasoning behind the decision taken. Research (see

[11]) has proposed the use of Decision frameworks and Decision Functions that are mapped to ABA in order to compute the required decision.

Definition 6. A decision framework *is a tuple* $\langle D, A, G, T_{DA}, T_{GA}\rangle$, *consisting of:*

- a set of decisions $D = \{d_1, \ldots, d_n\}, n > 0$,
- a set of attributes $A = \{\alpha_1, \ldots, \alpha_m\}, m > 0$,
- a set of goals $G = \{g_1, \ldots, g_l\}, l > 0$, and
- two tables: T_{DA}, of size $(n \times m)$, and T_{GA}, of size $(l \times m)$, such that
  - for every $T_{DA}[i,j]^2, 1 \leq n, 1 \leq j \leq m, T_{DA}[i,j]$ is either 1, representing that a decision d_i has attributes a_j, or 0, otherwise;
  - for every $T_{GA}[i,j], 1 \leq i \leq l, 1 \leq j \leq m, T_{GA}[i,j]$ is either 1, representing that goal g_i is satisfied by attribute a_j, or 0, otherwise.

In order to use ABA for decision the decision frameworks and decision functions need to be adapted to ABA. Computational Procedures have been identified for computing strongly dominant decisions, dominant decisions and weakly dominant decisions.

Definition 7. Given a decision framework $df = \langle D, A, G, T_{DA}, T_{GA}\rangle$, in which $|D| = n, |A| = m$ and $|G| = l$, the *strongly dominant ABA framework* corresponding to $\langle D, A, G, T_{DA}, T_{GA}\rangle$ is $df_s = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \mathcal{C}\rangle$, where

- $\mathcal{R}$ is such that : for all $k = 1, \ldots, n; j = 1, \ldots, m$ and $i = 1, \ldots, l$:
  - if $T_{DA}[k,i] = 1$ then $d_k a_i \leftarrow$;
  - if $T_{GA}[j,i] = 1$ then $g_j a_i \leftarrow$;
  - $d_k g_j \leftarrow d_k a_i, g_j a_i$;
- $\mathcal{A}$ is such that: d_k, for $k = 1, \ldots, n; Nd_k g_j$, for $k = 1, \ldots, n$ and $j = 1, \ldots, m$;
- $\mathcal{C}$ is such that: $\mathcal{C}(d_k) = \{Nd_k g_1, \ldots, Nd_k g_n\}$, for $k = 1, \ldots, n$;

$\mathcal{C}(Nd_k g_j) = \{d_k g_j\}$, for $k = 1, \ldots, n$ and $j = 1, \ldots, m$.

Definition 8. Given $df = \langle D, A, G, T_{DA}, T_{GA}\rangle, |D| = n$, and $|A| = m$, let the corresponding strongly dominant ABA framework be $df_s = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \mathcal{C}\rangle$, then the *dominant ABA framework corresponding to* df is $df_D = \langle \mathcal{L}, \mathcal{R}_\mathcal{D}, \mathcal{A}_\mathcal{D}, \mathcal{C}_\mathcal{D}\rangle$, where:

- $\mathcal{R}_\mathcal{D} = \mathcal{R} \cup Ng_j^{\bar{k}} \leftarrow Nd_1 g_j, \ldots, Nd_{k-1} g_j, Nd_{k+1} g_j, \ldots, Nd_N g_j\}$ for $k = 1, \ldots, n$ and $j = 1, \ldots, m$;

- $\mathcal{A}_{\mathcal{D}} = \mathcal{A}$;
- $\mathcal{C}_{\mathcal{D}}$ is $\mathcal{C}$ with $\mathcal{C}(Nd_kg_j) = \{d_kg_j\}$ replace by $\mathcal{C}(Nd_kg_j) = \{d_kg_j, Ng_j^{\bar{k}}\}$, for $k = 1, \ldots, n$ and $j = 1, \ldots, m$.

Definition 9. Given $df = \langle D, A, G, T_{DA}, T_{GA}\rangle, |D| = n$ and $|A| = m$, the *weakly dominant ABA framework corresponding to* df is $df_W = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \mathcal{C}\rangle$, where

- $\mathcal{R}$ is such that: for all $k = 1, \ldots, n; j = 1, \ldots, m$ and $i = 1, \ldots, l$:
  - if $T_{DA}[k, i] = 1$ then $d_ka_i \leftarrow$;
  - if $T_{GA}[j, i] = 1$ then $g_ja_i \leftarrow$;
  - $d_kg_j \leftarrow d_ka_i, g_ja_i$;
    for all $r, k = 1, \ldots, n, r \neq k$; and $j = 1, \ldots, m$:
  - $Sd_rd_k \leftarrow d_rg_j, Nd_kg_j, NSd_kd_r$;
  - $\bar{S}d_kd_r \leftarrow d_kg_j, Nd_rg_j$;
- $\mathcal{A}$ is such that: d_k, for $k = 1, \ldots, n$;
  NSd_kd_r, for $r, k = 1, \ldots, n, r \neq k$;
  Nd_kg_j, for $k = 1, \ldots, n$ and $j = 1, \ldots, m$;
- $\mathcal{C}$ is such that: $\mathcal{C}(d_k) = \{Sd_1d_k, \ldots, Sd_{k-1}d_k, Sd_{k+1}d_k, \ldots, Sd_nd_k\}$, for $k = 1, \ldots, n$;
  $\mathcal{C}(NSd_kd_r) = \{\bar{S}d_kd_r\}$, for $r, k = 1, \ldots, n, r \neq k$;
  $\mathcal{C}(Nd_kg_j) = \{d_kg_j\}$, for $k = 1, \ldots, n$ and $j = 1, \ldots, m$.

By implementing the above mapping and the computational mechanisms provided we can enable our web-application to come closer to being decision-making tool.

2.11 Mapping AA to ABA

AA can be mapped to ABA and thus AA frameworks can be computed using ABA. Specifically the mapping is as follows (according to [7]):

Definition 10. Each AA framework $\mathcal{F} = (Arg, attacks)$ can be mapped onto a *corresponding ABA framework* $ABA(F) = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, \bar{} \rangle$ with

- $\mathcal{A} = Arg$;
- for any $\alpha \in \mathcal{A}, \bar{\alpha} = c(\alpha)$, with
  - $c(a) \notin \mathcal{A}$ and,
  - for $\alpha, \beta \in \mathcal{A}$, if $\alpha \neq \beta$ then $c(\alpha) \neq c(\beta)$;
- $\mathcal{R} = \{c(\alpha) \leftarrow \beta | (\beta, \alpha) \in attacks\}$;

- $\mathcal{L} = \mathcal{A} \cup \{c(\alpha) | \alpha \in \mathcal{A}\}$.

By implementing the mapping within the system, we can now allow it to take as an input an AA framework and evaluate it using the underlying ABA engines. This will make the web-application more flexible as it will be able to compute in accordance to different forms of argumentation.

Chapter 3

Building a web application interfacing the derivation engines.

The project main outcome is creating an extendible web application that provides an interface to various derivation engines for ABA. The application should be user friendly and allow for future integration with other engines. This extensibility can enable the application to establish itself as a common gateway to various derivation engines for ABA. As more engines are added the Web Application can be established as a one-stop-shop for all things ABA.

The following section focuses on providing an overview of the solution implemented. It also explores various design and implementation choices made. Lastly, an evaluation of the current implementation and its alternatives is explored.

3.1 Overview of the Solution.

The web application's architecture is based on a 3-tier system that includes three distinct parts that make up the application. These parts of the application communicate between each other to provide the user with the final desired outcome. The system is made up from the Client, Server and Derivation Engines. The Server acts as a coordinator between the Client and the Derivation Engines. Users interact with the client side of the application which then communicates with the Server. The Server then process the request from the user and submits it to the eligible Derivation Engine. Once the derivation is complete the Server process the solution and sends it back

to the client as illustrated in [TODO].

3.1.1 Client

The Client part of the application is the user's gateway to the argumentation derivation engines. It provides the user with a GUI providing both the means required to submit an argumentation framework and the tools required to view the derivation tree received. It also allows the user to specify which engine and what semantics are to be used.

It was built using tools such ASP.Net and Javascript. ASP.Net, along with HTML was used to create the web pages the user can interact with. Javascript and the D3.js library where used to provide a useful visualisation of the outcome to the user in the form of a derivation tree.

3.1.2 Server

The Server part acts as a coordinator between the Client and the Derivation Engines. Within the Server the input from the Client is processed and prepared to be given to the necessary derivation engine to be analysed. The Server part carries out the following functionality:

- Analyses the parameters from the user and selects the correct derivation engine and semantics.
- Parses the user's input, processes it and generates the relative input for the derivation engine.
- Grounds the framework provided by the user over the domain specified.
- Parses and analyses the output from the derivation engine and builds a JSON string that is passed to the client to build the derivation tree visualisation.

3.1.3 Derivation Engines

The Derivation Engines used for the current implementation include Proxdd and Grapharg which are both implemented in Prolog. The correct derivation engine is initialised with the correct parameters and provided a suitable input it outputs a correct derivation. The derivation engines can be seen as separate entities from the rest of application which implies a plug-in plug-out possibility. By providing a module within the server that interprets and formulates the input and output to the engine, the application should be able to handle the introduction of any new derivation engine it its back end.

3.2 User Interface with Application.

The application, like most web applications, carries out most of the intensive processes in its back end. Therefore, the user interface module carries out little of the functionality of the application, but instead focuses on allowing the user to interact with the engines. The GUI controlled by the User part of the application carries out two tasks. It allows users to input their framework and parameters and it constructs the derivation tree of the solution.

3.2.1 Setting up the framework.

Since the application derives the solution of a user defined argumentation framework, then the user should have the means to input said framework. The GUI provides a text box in which the user can define their ABA framework using a simple language that allows them to define assumptions, contraries and rules over a specified domain. An simple example of such an input can be seen at [TODO]:

[TODO]

The formal context free grammar of the input language along with how it is parsed is explained in detail in section [TODO].

3.2.2 Configuration of the engines.

Additionally the users should be able to choose the configuration of the derivation engine to be used. This includes two main decisions, the engine to be used and the semantics to be used.

The users can choose the engine required by checking the check-box as in example [TODO].

[TODO]

The user can then choose the semantics required by selecting the check-box corresponding to the required semantics, as in example [TODO].

[TODO]

[COULD INCLUDE JUST ONE IMAGE WITH ANNOTATION]

Lastly, the user must specify the claim for which a derivation is required as in example [TODO].

3.2.3 Visualisation the derivation tree.

Once the input provided has been analysed by the derivation engines then the solution found is provided to the user as a derivation tree. Once the Server side finishes processing the solution it provides a JSON string that is

interpreted client-side by javascript and the D3.js visualisation library. The JSON string has a structure as illustrated in example [TODO]. This structure specifies the characteristics of each node in the tree and the edges between the nodes.

[TODO]

On the client side the JSON is processed and the derivation tree it represents is illustrated in the canvas area of the web page as shown in example [TODO]. The tree is annotated and colour coded for clarity and it also allows the user to zoom in and out.

3.3 Server - Processing the Input.

Once the user specifies the framework it then has to be passed to the server to be processed. Within the server several steps take place concerning the processing of the framework. These allow the interpretation of the input and the generation of the corresponding input for the engine. These steps include:

- Parsing the Input and seperating the various elements (as described in [TODO]).
- Checking the validity of the input (as described in [TODO]).
- Grounding the input over a certain domaind (as desribed in [TODO]).
- Generating the derivation engines input (as described in [TODO]).

Due to the modularity of the web application it is possible that new derivation engines could be added to the back-end by implementing alternative to these steps. For example, if a new engine is added than by switching to a different generator we can now use the parsed and grounded input to create an input viable for the new derivation engine.

3.4 Server - Interfacing with the Prolog Engine.

Both Proxdd and Grapharg are implemented in the Prolog programming language which is commonly used in the field of AI. However, it has not been developed with web application development in mind which would hinder any attempts to extend the web application in the future if the whole of the server implementation was carried out in Prolog.

3.4.1 Need for an interface between C# and Prolog Engines.

The following decision choices were all possible:

- Re-implementing both Proxdd and Grapharg in C#
- Using an external library that provides an interface for C# with Prolog.
- Implementing the back-end of the web application in Prolog.

Nonetheless, for reasons explained in more detail in section [TODO] the use of an external library seemed to be the most reasonable choice. The interface allows for seamless integration of Prolog and C#. Additionally, it enables us to focus most of the development work in a programming framework that is more popular and extendible than Prolog.

3.4.2 Using the SwiPIC.dll library as an interface.

The SwiPIC.dll library was chosen as it a provide a simple interface from C# to the SWI-Prolog. It allows for the initialisation of an Instance of the SWI-Prolog Engine and we can then build Prolog queries that can be run. The code snippet at [TODO] provide an example of how the Prolog Engine is initialised and how a query is run:

[TODO ADD EXAMPLE OF CODE]

By using an external library such as this we can easily interact with the derivation engines implemented in Prolog by submitting the query we have constructed based on the user's input and extracting the solution returned by the engine. The interface also enables us to implement the bulk of the application in C# and then use external libraries such as this to interact with any other derivation engines in the future, while keeping the bulk of the implementation common.

3.5 Choosing the current implementation.

There are often enough more than one correct ways of designing and implementing a system. Related decisions are often taken based on the priorities of the project and other limitations. In this subsection I look into justifying some of the design decisions taken and potential disadvantages. Additionally we will look into one of the proposed alternatives for the derivation engines.

3.5.1 Advantages of current implementation.

The main advantage of the 3-tier system is that we provide modularity to the system allowing us to separate various significant elements of the system. In our implementation most of the processing other than the derivation is carried out in the Server tier by the C# code. This makes it highly extendible as a simple switch can be implemented in the Server tier that enables the user to choose from an array of engines. The implementation, being modular, does not require extensive changes in the code to accommodate new derivation engines. As long as an interpreter is created to generate the required JSON string for the client side and the user input is translated to the correct corresponding input for the derivation engine, then the rest of the application should remain unchanged.

In addition to this the modular implementation allows the incorporation of further extensions to the application as part of the pre-processing process in the Server module. As an example, consider that there is a direct mapping that can be implemented between Abstract Argumentation and Assumption Based Argumentation. This mapping can be implemented in a module that is set to run as part of the pre-processing done in the Server module. This can be added to the process in a seamless manner, by which it can be enabled or disabled when required without any additional code changes.

Lastly, the choice to interact with the Prolog engine through an interface rather than the alternatives suggested was done for the purposes of usability and extensibility. Details of this approach are discussed in the last subsection [TODO].

3.5.2 Disadvantages of current implementation.

One of the disadvantages of the current implementation is its reliance on rather exotic external libraries to interface and run the Prolog derivation engines. There is the potential that if the web application is further developed in the future an updated version of this library might be required. However, as the library is not one of the core libraries in C# such an update might take long to be released. Nonetheless, this was taken into account when deciding whether to used this library or not. The application deliberately tries to limit the use of this library to just initialising the engine, submitting a query and extracting the solution. The specifics of the query, for example, are built without the use of this library. Additionally, when deploying the application the environment needs to be configured to account for these libraries.

Another disadvantage of the current implementation is that there is currently a significant amount of processing required to generate the data com-

municated between the tiers in the required format. When a user inputs a framework this must then be translated in the required form for the input of the derivation engine. Once the derivation engine complete the solution then has to be interpreted and the corresponding JSON string must be generated. This takes processing time. An alternative would be to adapt the derivation engines to directly produce the JSON string rather than an alternative output. However, it would be unrealistic to expect derivation engines to conform to these standards. Instead, the current implementation allows for better integration of new engines rather than processing power.

3.5.3 Proposed alternative (Prolog Web Server).

There is one exceptionally interesting alternative implementation that was considered during the design process. SWI-Prolog which is the flavour of Prolog used in our implementation also provides the ability for it to run as a Web Server on its own. The proposed alternative is to establish a web server based on SWI-Prolog that would also include the derivation engines. This web server can then be configured to listen on certain ports for requests that it would be able to handle and reply to. Our web application would then be able to send over the required parameters of the derivation in the form of an Http Request to the Prolog Server. The Prolog Server would then carry out the derivation process and respond with the solution. A major potential advantage of this implementation would be that the Prolog Server could be used by further applications in the future. Requests can be made by various applications and these would be handled accordingly by the Prolog Server.

Nonetheless, there are issues that had to be considered when rejecting this alternative. Configuring the web server and the firewalls between it and our web application would be a very cumbersome task, which could be avoided. Also by creating a completely distinct web server to handle the derivation we are relying our web application on two servers. This creates two potential hazard points for our application. If the Prolog Web Server happens to stop working or face a fault then the web application will not work properly. By integrating Prolog through C# we reduce this potential weakness point. Lastly, the idea of implementing the whole Server side of the application on a Prolog Web Server was rejected as Prolog is not a well supported web development framework such as C# and ASP.Net. This would provide problems when having to extend the application to include new engines which might be built in other programming languages and frameworks such as C++, C, Pythonm etc.

Chapter 4

Creating an Input Language.

A user of the application must be able to define an ABA framework from which he wants to derive a solution for a claim. Therefore a mechanism must exist that allows the user to do so. However, the web application has been designed with extendibility in mind and the input provided should be in a form that makes it useful irrespective of the underlying derivation engine.

4.1 Creating a Context Free Grammar for the Input.

When deciding on the input method there were several parameters that had to be considered. Some of the most important ones were:

- It must be easy for the user to understand and use.
- Should be able to accommodate all the derivation engines implemented now and in the future.
- Should allow for the easy input of both large and small frameworks.

Having considered these factors, the input method chosen for the web application is a simple text input comprised of predetermined commands that can be interpreted by a parser to input the framework to the desired derivation engine.

4.2 The need for a new input.

Perhaps the most important reason for defining formally a new input language to create frameworks is that by abstracting the input language from the derivation engines we can then use a single language for all the potential

derivation engines. Alternatively, the user would have to be aware of the input mechanism for each derivation engine. Instead we chose to abstract the user input from the engines themselves and then create the corresponding required input for the specified engine. Essentially the user remains blissfully ignorant of the specifics of each derivation engine and of the further tiers of the systems. This reduces their ability to interfere with the engines directly and avoids confusion between different input languages.

Furthermore the input language suggested and implemented was designed to be as simple and straight forward as possible. This reflects the fact defining an ABA framework simply requires the definition of assumptions, rules and contraries. Therefore, the language is made up of two statements:

- asm(a,b). used to define an assumption and its contrary.
- b(X) ¡- [a(X)]. used to define a rule.

Additionally, the language we specified allows for the definition of a domain over each of the elements of the framework is valid. By combining this with the grounder described in section [TODO], we can define domain specific framework. This feature of our language also allows us to reduce the amount of lines of input the user has to enter in order to define a specific framework.

4.2.1 Formal definition of Input.

In order to be able to check the input for validity a formal definition must be followed. The Context Free Grammar for our input language is defined below in section [TODO].

[TODO] FORMAL DEFINITION.

4.3 Parsing in the input.

Having formally defined the language we now proceeded with creating an interpreter to parse the user input and take the necessary action. This involves a simple 3 step process (as in figured [TODO]) by which the input is parsed, checked if it is valid and then the corresponding input for the targeted derivation engine is generated.

4.3.1 Building a simple parser for the language.

As noted the input language specified has just two distinct commands that the user can use to specify elements of a framework. This is due to the simplicity of specifying ABA frameworks. The two commands available are:

- *asm(a,b).* - used to define an assumption and its contrary.
- *b(X) <- [a(X)].* - used to define a rule.

The input can be considered as a series of statements separated by the terminal character ".". Each statement specifies either an assumption with its contrary or a rule. Using these two commands interchangeably the user can specify their argumentation framework. By keeping the language simple a simple parser can be created that checks whether each input statement is in one of the two forms.

Once a statement is extracted it is mapped to one of the two cases using the unique identifier tokens "asm(" or "¡-", which are used to identify whether the user is specifying a rule or an assumption. The tokens were chosen to be easy to remember and reuse. The use of the "¡-" operator in the rule definition is exceptionally memorable for users as it is the most common way in which rules are specified in the literature and it is a also a very commonly used operator in logic overall.

Due to the simple and strict nature of the language the parser created is simple in its implementation. It carries out the the following three tasks:

1. Separates the statements by the terminal character ".".
2. Each statement is then identified as either an assumption declaration, rule declaration or an invalid statement.
3. If the statement is valid, it is then separated in its individual parts according to the terminal symbols and the corresponding object (a rule or an assumption/contrary) is created.

In addition to these two commands the language also offers the possibility for the user to define a domain over which certain assumptions and rules are valid. this is done using the tokens defined in [TODO]. The existing engines of Proxdd and Grapharg do not take into account ungrounded assumptions and rules that contained unassigned variables. However, when defining an ABA framework it might be the case that the same assumption holds over various members of a domain.

The language supports the definition of a domain over which an assumption is valid. The functionality of grounding these assumptions over the domain is handled as part of a pre-processing step before the input for the derivation engines is generated. This is explained thoroughly in section [TODO].

If the parser manages to parse the whole of the input successfully, then by the end a web of assumption contrary and rule objects is created, that corresponds to the ABA framework defined.

4.3.2 Verifying the correctness of the input.

One of the key purposes of the parser is not only to interpret the input, but also to verify whether it is valid. Due to the limited amount of statements that are considered valid and the simple linear structure of an input (list of statements) we can simply compare each statement against a mask created using a regular expression. Specifically the regular expressions are defined in [TODO].

[TODO] - Add regular expressions.

If a statement does not fit in these regular expressions then it is an invalid statement and the input in its entirety is considered as invalid. This strict approach to the input ensures that the user has not made any human error that would render it invalid.

4.3.3 Forcing the existence of a contrary.

ABA frameworks themselves have certain rules by which they must adhere to. A common example, that is prone to human error when defining an ABA framework, is that a contrary must be defined for every assumption declared. Our suggested input language is constructed so as to force the user to declare the contrary upon defining a new assumption. This is done by incorporating the definition of a contrary in the declaration of an assumption, as shown in example [TODO].

By forcing the user to define the contrary in the same step as declaring the assumption we ensure that there will always exist a contrary for an assumption. It should be noted that the current implementation restricts the user to defining only one contrary for each assumption. If the need arises in the future then the second term in the assumption command could be redesigned as a list that could specify more than one contraries for a single assumption.

4.4 Example of an input.

To illustrate a valid input corresponding to a valid ABA framework the following example is provided [TODO].

[TODO] - Add input example and formal definition of corresponding framework.

Chapter 5

Grounding a framework over its domain.

As explained in previous sections the current implementation of the derivation engines of both Grapharg and Proxdd does not support the inputting of frameworks that include non-grounded elements. Consider example [TODO].

[TODO] - Add example of non ground assumption.

The use of assumptions and rules specific to a certain domain is very important when it comes to decision making as often enough assumptions and rules change depending the parameters of a situation. Unbounded variables to assumptions and rules cannot be handled and therefore we are forced with creating frameworks that are either valid towards just one specific domain or provide a general overview of what the expected outcome would be.

5.1 The need of specifying a domain for a framework.

Often enough the assumptions and rules of an ABA framework are valid over a specific domain rather being valid globally. However, defining the domain over which elements of the domain are valid allow us to derive a specific solution for a more specific scenario. Often enough, the parameters of a situation might affect which assumptions and rules are valid. Consider the example [TODO]

[TODO] - Find Example, Write it and Walk through it.

Therefore, a mechanism must exist that allows the user to specify over which domain the assumptions or rules hold. One such possibility is for the users to directly specify the parameters of an assumption or rule in the declaration sentence when inputting the framework. This would be a cumbersome

task especially when it comes to exceptionally large frameworks.

Consider, for example an assumption that is valid for 20 different people. When inputting the framework the user would have to declare a new assumption for every individual. This would imply that the user will have to declare the same assumption, but with different parameters, 20 times.

An alternative is to provide the user with the ability to define the framework over which an element is valid and then build a grounder that would bound all unbound variables according to the domain specified.

5.2 Implementing a grounder.

The input language of the application allows the user to define a domain over which assumptions and rules are valid as specified in section [TODO]. However, for the domain to come into effect the elements declared in the input should be grounded over the provided domain. The grounder was implemented in the Server module using C# and is part of the pre-processing of the input (as shown in figure [TODO]) before the corresponding input to the derivation engine is generated. This implies that the input to the derivation engine does not include any unbounded variables, which is a requirement by the current derivation engines Proxdd and Grapharg.

The grounder used in our implementation has been developed from the ground up and is based on the grounder algorithms used by DLV (Add reference [TODO]). The algorithms themselves, along with details of the implementation are provided in section [TODO].

With the existence of a grounder our web application can now handle more real world decision problems that have been formulated in an ABA framework. It can also evaluate the validity of a claim given certain parameters and can provide an analysis of domain specific problems.

5.2.1 Description of the algorithm.

The algorithm implemented is built based on the grounder algorithms used for DLV (reference [TODO]). The objective of the grounder is to take the user specified framework and ground the individual components over the domain they are valid. This is carried out by the algorithm described in this section. The algorithm can be summarised in the following simplified steps:

1. The elements of the framework are placed in the EDB and IDB. If an element is an assumption or a fact (a rule without a body) then they are grounded based on the domain they are specified and added to the

EDB. All other rules that are depended on other predicates are added to the IDB.

2. A dependency graph of the program represented by the IDB is formulated and split into subprograms, thus creating a modular dependency graph made up of Strongly Connected Components.

3. An ordering of the modules of the dependency graph is derived and implemented.

4. The modules are processed in the order defined and the grounded rules within each module are instantiated and added to the grounded program.

As specified the first step of our grounding process is to identify facts that definitely hold in our framework, ground them over the domain specified and add them to the EDB. This includes two special cases:

- *asm(a,b){X=1,2,3; }.* - once assumptions and contraries are defined then they must exist in the ABA framework as they are not reliant on the existence of other predicates.
- *b(X)<- {X=1,2,3; }.* - rules without a body are equivalent to *b(X)<- true* and therefore again are not reliant on the existence of other predicates.

Both of these cases can simply be grounded by calculating all possible combinations of the variables as defined in the domain by the user and then instantiating these predicates with these combinations and adding them to the EDB. The rest of the rules are added to the IDB for further processing and grounding.

Once we have isolated the initial IDB we can now pre-process it and start working towards grounding the various extra rules. This pre-processing allows us to avoid instantiating unnecessary rules that are not achievable since their predicates are not valid under the specified domain. The first step of the pre-processing is generating a dependency graph that replicates the interdependencies between the various rules. This is pretty straight forward to implement and was done by creating a data structure that includes the nodes and the edges between them as specified in [TODO].

Having established the dependency graph we now proceed by partitioning the graph into sub-programs. This is done by identifying strongly connected components (SCC) as defined in definition [TODO]. The dependency graphed

is analysed using Tarjan's strongly connected component (see [TODO]) algorithm and the modules of our graph are identified and used to construct a new Component graph representing the program.

With the modular dependency graph now constructed we must derive an ordering between the modules. The modules are placed into sorted list with the order being defined as in [TODO]. This enables us to first instantiate the rules on which rules from the following modules depend on. Having done so we can minimise the amount of unnecessary rules that would be created as explained in [TODO].

Having completed the pre-processing stage we can now focus on instantiating the rules that represent our framework over the defined domain. This is carried out by the algorithms used for DLV (reference [TODO]) which are described in [TODO]. These instantiate the rules of one module at a time and work incrementally until all of the modules of the ordered list are processed.

For every rule we instantiate it according to the matching algorithm as defined in [TODO]. This algorithm goes through the predicates of a rule and finds valid grounded substitutes of the predicate already existing on the EDB. If a match is found then that substitution of the rule is added to the grounded program and the EDB.

Once all the rules of all the modules have been processed we are now provided with the final grounded framework.

5.2.2 Advantages of implementation.

An easier to implement and more primitive grounder could simply compute all possible values for each variable in our framework and then build a dictionary of these values. This dictionary would then be used to create the grounded rules using all possible combinations of values for the unbounded variables. Nonetheless, this could create an explosion to the size of the framework that might be unnecessary. Consider the following example:

[TODO] - Example of explosion. Assumption with

By implementing a smarter grounded we can avoid the generation of rules that are not achievable. If a rule is made up of predicates that are not valid over the domain specified then the rule is not constructed in the final grounded framework. This allows us to restrict our framework only to rules that could be valid and useful. This also provides a performance boost to the derivation engines as the framework provided for analysis is smaller.

5.2.3 Disadvantages of implementation.

Grounders are an extensive research area especially when it comes to creating highly optimised algorithms. There are several techniques that can be used to improve the performance of a grounder and make it more efficient such as [TODO] reference to Gringo and [TODO] Reference to BJ instantiate for DLV. Our implementation does not focus on making the most of these optimisation techniques. This was deemed reasonable as since the input will be defined by the user will hardly ever be of a size that would provide a considerable performance gain to justify the extra development effort that would be required to implement these optimisation techniques.

Additionally, the current grounder is restricted in when it comes to the input it accepts. Although it serves the formal input language defined it would require additional development to be used in other cases or if the formal language is extended. The most notable limitation of our grounder is its inability to accommodate negative predicates, as in exampe [TODO]. However, in the ABA frameworks we are analysing negative predicates are not often encountered in the body of a rule.

5.3 Example of expected output.

Chapter 6

Implementing Ideal Semantics dispute derivations.

When analysing an ABA framework semantics play an important roles. As explained in section [TODO] Background, when finding a derivation for a claim the semantics by which we do so must be specified. The current derivation engines Proxdd and Grapharg support derivation based on Grounded and Admissible semantics (as defined in [TODO]). As part of our implementation I looked into extending the Proxdd engine to support derivations based on ideal semantics. For an overview of ideal semantics refer to section [TODO].

6.1 Extending Proxdd to include ideal semantics.

Ideal Semantics dispute derivations are heavily based on the admissible semantics based dispute derivations. Proxdd already implements the such a derivation using the sxdd dispute derivation algorithm as described in [TODO]. This algorithm can be adapted to provide a derivation based on ideal semantics by extending it as in the algorithm described at [TODO]. There a are two adaptations to be made to the ab-derivation algorithm so that it can compute ideal semantics based dispute derivations:

1. Extend the algorithm to compute (P,O,D,C,Attacks,Arguments,F) tuples instead of the current (P,O,D,C,Attacks,Arguments) tuples and update F according to the algorithm in section [TODO].

2. Implement the *Fail(S)* check as explained in section [TODO]. This

check is then implemented in an extra step in the algorithm as described in section [TODO].

6.2 Implementing ideal semantics.

When implementing the ideal semantics the process was broken out in three steps:

1. Append the "F" list to the tuples and update it correctly.
2. Implement the *Fail(S)* check procedure.
3. Bring everything together in the final derivation engine.

Our ability to modularise the implementation process was largely dependent on the fact that the implementation is an extension of the ab-dispute derivation algorithm already implemented. This implied that each of the three steps could be implemented and tested individually before connecting the parts in our final derivation engine. The sections that follow ([TODO]) elaborate on the changes that were required, the implementation process and difficulties faced when implementing.

Updating the F set.

The first step was also the step that was less invasive to the current implementation of the derivation engine. This involved appending the "F" set to the tuples used by the sxdd algorithm as described in definition [TODO]. Additionally the updating of the "F" set was not invasive to the rest of the already implemented sxdd algorithm. This can be seen in the algorithm in section [TODO]. Therefore, "F" could be included and updated throughout Proxdd's sxdd algorithm without interfering with ab-dispute derivations or gb-dispute derivations. The "F" set would be updated at any derivation but its content would be irrelevant unless the user specified that ideal semantics were desirable.

Updating the "F" set in accordance with the algorithm in most cases is straight forward. Consider the case where it is the Proponent's turn. In this case the "F" set remains unchanged. As in step [TODO] of example [TODO]. The updating process becomes more complicated when it is the Opponent's turn. On most cases (as seen in the algorithms definition [TODO]) the "F" set is updated by adding the sentences that are unmarked. Marking of sentences is already handled by the existing sxxdd algorithm and we can therefore use the current implementation to add add just the corret sentences.

The most complicated updating of the "F" set takes place at step [TODO]. An update at this step is illustrated by the example [TODO].

[TODO] - Add example that updates the F set accordingly.

Having appended the "F" set we can then test that it is updated correctly, in accordance to the algorithm, without having to implement any of the next steps. This can be done using the ab-dispute derivation semantics and checking the value of the "F" set at each step. The "F" set would simply be ignored and the derivation should still return a valid derivation.

It should be noted that, according to the algorithm in [TODO], the "F" set is also update when it is F's turn and a set of sentences is selected from "F". However, this case was implemented in the third step as it would not come into effect until we need to become invasive with the current implementation.

Carrying out the *Fail(S)* check.

The purpose of including an "F" set that is being updated during the derivation process, is so that we can run the *Fail(S)* check over the set of sentences included in "F". *Fail(S)* is referred to by [TODO] reference as the Fail-dispute derivation of a multiset of sentences. The *Fail(S)* check was implemented in accordance to the algorithm defined in [TODO] and in accordance to [TODO] reference.

[TODO] Add paragraph about what exactly it checks, ask Toni at meeting.

Similarly the *Fail(S)* check plays a Proponent/Opponent games similarly to how the dispute derivation games are played and similarly produces tuples of the structure (P,O,D,C). In fact the new tuple is created using steps from the algorithm in [TODO] as defined in the fail-dispute derivation algorithm in [TODO].

Having implemented the various cases and steps the ideal semantics algorithm was tested by running each step with dummy input and checking whether the output would reflect the expected output, as illustrated by example [TODO]. This ensured the validity of the tuples generated. Having established the validity of these steps we then incorporated them together in a recursive dispute derivation game (see algorithm in [TODO]). This enabled us to check *Fail(S)* as a whole by again inputting a dummy case and checking whether the derivation process was carried out and completed. In this case, due to the semantics being implemented in Prolog, the *Fail(S)* check was expected to either fail or return true if it succeeded, depending on the input we provided. Nonetheless, to ensure the validity of the derivation the process was checked at each step to ensure that the tuples were being generated as

expected.

The current implementation has potential to be optimised further to allow for faster Fail-dispute derivation checks. This would be especially useful in larger scale frameworks. Although the functionality is accurate there is still room for improvement in some areas. One such area can be the selection process by which the algorithm chooses which sentence or set of sentences to check at each turn. For the ab-derivation in Proxdd there is a a selection process that implements an ordering on these sentences before it chooses. Such an optimisation has not been implemented in the current implementation. Currently the Fail-dispute derivation chooses sentences simply by picking off the head of the list each time.

Adding the Ideal Semantics to Proxdd.

The final step involved seamlessly incorporating these new features in the current implementation of Proxdd. The derivation engine uses flags such as "set_ab." and "set_gb." to allow the user to specify which semantics to be used. Naturally, a similar "set_ib." flag was implemented to allow the user to specify the use of ideal semantics.

The major change of the ib-dispute derivation is the inclusion of a "new player" in the dispute derivation game, the set "F". Similarly, to admissible dispute derivation the game is played until both P and O of the currently processed tuple are empty. This would imply the termination of the derivation process having successfully found a derivation solution for the claim provided under the semantics for admissibility. However, in ideal semantics we are looking for a specific subset of the admissible solutions which is defined by whether the *Fail(S)* check is satisfied for all sets in 'F" as defined in [TODO].

Therefore, the first design choice we need to make is when do we attempt to check whether the sets in "F" satisfy the Fail-dispute derivation check. Our solution to this problem is to let the ab-derivation run to completion (i.e both P and O are empty) and then, instead of terminating, identifying that it is F's turn to run the fail-dispute derivation check. Therefore, when incorporating the ideal semantics into Proxdd the "choose_turn" function was adapted as shown in definition [TODO].

[TODO] - Add code of choose_turn.

By implementing F's turn we now have an implementation of the algorithm that finds an admissible derivation and then checks whether this solution is also in accordance to the ideal semantics, as specified in algorithm [TODO].

Additionally to the "choose_turn" function the "derivation" function had

to be adapted as well. Specifically, the base case had to be adapted as shown in [TODO].

[TODO] add new derivation function code.

The current implementation of Proxdd terminates the derivation successfully if both P and O are empty. However as noted already in ideal semantics this is not enough. The derivation base case has been adapted so as to terminate successfully when the "F" set is empty as well. This implies that each set of sentences in the "F" set has successfully passed the *Fail(S)* check and has been removed from "F" in accordance to the algorithm defined in [TODO].

An empty "F" set at the end implies that the solution found is a valid derivation under ideal semantics. If the "F" set is not empty then the base case of the derivation function is never reached, Prolog fails the derivation and attempts a different solution. Thus, enforcing dispute derivations under ideal semantics.

6.3 Example of ideal derivation.

Chapter 7

Project Plan

"Failing to plan is planning to fail" — Winston Churchill

As with any project, planning and time management are imperative. To avoid getting overwhelmed by the workload required by the project, it has been preliminarily planned in the following way. Due to the agile nature of software development and the timeframe in which the implementation must be finalised, the following planning is preliminary and subject to change as the project moves on. For clarity purposes planning was split in three sections; Planning Implementation, Project Timeline, Planning Future extensions.

7.1 Planning Implementation

The application designed and implemented will take the form of a web-application. This will allow the system to be readily accessible to anyone over the internet, rather than having to be designed for a specific operating system and having to have the users to download and set-up the system. Such an approach will enhance the user-experience and make the system more maintainable as updates and additional features will have to be implemented just on the developers' side.

I have decided to use C# and the ASP.NET framework to develop the application. By using a widely supported framework I will ensure that the application functions correctly on a wide range of internet browsers. Additionally, the underlying Model-View-Controller software pattern of ASP.NET suits the problem at hand, as it will allow me to abstract the analysis aspect of the application from the visualisation aspect. This can have exceptional benefits in the future as different Views can be generated to represent other

visualisation options in the future. For the visualisations themselves, I will be using the D3.js package that allows for easy interactive visualisations to be created. Lastly, the computational engines behind the application will be the existing Proxdd and Grapharg systems (described in section 2.6 and section 2.7) which will be invoked through SICSTUS Prolog.

7.2 Project Timeline

Below is the preliminary timetable for the tasks that need to be completed for the project. This timetable is open to revision in case the projects runs into trouble. Nonetheless, an underlying general timeframe exists and is defined by the milestones (shaded in grey). Ideally, the basic functionality required for the project should be completed by the end of March, allowing April and May for implementing potential extensions (see section 2.9, section 2.10, section 2.11) and June for the final project write-up. These milestones can also act as roll-back positions. If the development of the extensions fail the system can always be rolled-back to its previous stable state (last achieved milestone).

Tasks	Start Date	End Date
Read research papers and explore existing applications	16/12/2013	20/1/2014
Decide on implementation languages and tools	16/12/2013	6/1/2014
Write Interim Report	16/1/2014	31/1/2014
Set-up tools and environments	1/2/2014	4/2/2014
Enable communication with engines over the internet	6/2/2014	7/2/2014
Implement Visualisation Canvas	8/2/2014	20/2/2014
Implement Input UI	8/2/2014	20/2/2014
Evaluate system with test examples	20/2/2014	27/2/2014
Finalise web-application	1/3/2014	10/3/2014
Basic Report write-up	1/3/2014	1/4/2014
Look into implementing ideal-semantics dispute derivation	1/4/2014	30/4/2014
Extended Report write-up	1/4/2014	30/4/2014
Look into implementing decision making with ABA	1/5/2014	1/6/2014
Look into implementing AA mapping to ABA	1/5/2014	1/6/2014
Final Report write-up	1/6/2014	17/6/2014

7.3 Planning Future Extensions

Having established the applications the project can, in the future, be extended to enhance the experience or add new functionality. For example, we could allow for the direct input of Decision Problems (see section 2.10) or we could extend the application to support AA argumentation as well by implementing the necessary mapping (see section 2.11). An attempt to these extensions has been scheduled and might be carried out if time allows it.

However, in addition to these planned extensions there are some interesting further enhancements that can be considered in the future. A creation of a stable and useful API for the underlying system would be of great use as it would allow for developers to make use of the system in there developing aspirations. This could provide the necessary platform for developers to launch Argumentation as a commercially viable tool. Additionally, this

could lead to a further extension of the project that will involve the creation of real-life problem solvers that will use the argumentation engines. Due to the planned architecture of the system, this should be simple to do, especially with the inclusion of an API. Lastly, if decision tools are indeed designed using this system, then in the future it might be useful to consider make the web-application also available for hand-held devices. This will allow the tools to be used on the fly by users, making them even more accessible.

Chapter 8

Evaluating Project

When designing and implementing a web-application there are several measures one must consider to evaluate the success of the project. The web-application is meant to be used by a wider range of users. Therefore, performance is not the only aspect of the application I must evaluate. User experience is equally important.

In order to evaluate the user experience and accessibility of the application the web-application should become available to pilot users early on. Through the feedback received, issues can be identified and the experience can be improved accordingly. Use of the web-application requires certain knowledge of argumentation. By creating a simple informative section on the website of the web-application the concept should be introduced to a level that would allow users to provide feedback on their interaction with the application.

Performance wise the system will be tested with a series of example frameworks, to ensure its performance both in validity and in computational speed. The existing systems do compute with a satisfactory speed. The web-application should not hinder this performance significantly. To ensure that the application performs well, the test cases will be vary in terms of complexity and size.

Lastly, certain automated tests will be used during the development phase to ensure that the system is robust against certain likely software issues. One of the aims of the project is for the application to be extendible in the future. The existence of these automated tests has the additional benefit of making the application more extendible by other developers in the future. By providing these tests, we allow the developers to ensure that the initial functionality of the application is not obstructed by any additional functionality they might introduce. Such, tools are important in ensuring that the application is extendible beyond the scope of this project.

Bibliography

[1] Dr. Robert Craven. Grapharg. *http://www.doc.ic.ac.uk/∼rac101/proarg/grapharg.html*.

[2] Dr. Robert Craven. Proxdd. *http://www.doc.ic.ac.uk/∼rac101/proarg/proxdd.html*.

[3] Robert Craven, Francesca Toni, and Matthew Williams. Graph-based disput derivations in assumption-based argumentation. In *TAFA 2013*.

[4] Phan Min Dung, Robert Kowalski, and Francesca Toni. Assumption-based argumentation. In *Argumentation in AI*.

[5] P.M. Dung, P. Mancarella, and F. Toni. Computing ideal sceptical argumentation. *Artificial Intelligence*, 171(1015):657 – 660, 2007. ¡ce:title¿Argumentation in Artificial Intelligence¡/ce:title¿.

[6] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. Aspartix: Implementing argumentation frameworks using answer-set programming. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, volume 5366 of *Lecture Notes in Computer Science*, pages 734–738. Springer, 2008.

[7] Francesca Toni. Reasoning on the web with assumption-based argumentation. In *8th Reasoning Web Summer School volume 7487 of Lecture Notes in Computer Science*.

[8] Francesca Toni. A generalised framework for dispute derivations in assumption-based argumentation. *Artificial Intelligence*, 195(0):1 – 43, 2013.

[9] Francesca Toni. A tutorial on assumption-based argumentation. *Argument & Computation*, 2013.

[10] TU Wien. Aspartix. *http://www.dbai.tuwien.ac.at/proj/argumentation/systempage/*.

[11] Fan Xiuyi and Toni Francesca. Decision making with assumption-based argumentation. In *TAFA 2013*.