UNIVERSITY OF ABERDEEN

Masters Thesis

Rankings Arguments & Extensions for Sets of Attacking Arguments

Author: Bhaskar Sinha

Supervisor: Dr. Bruno Yun

A thesis submitted in fulfillment of the requirements for the degree of Master of Science
in the

School of Natural & Computing Sciences

November 4, 2020
Declaration of Authorship

I, Bhaskar Sinha, declare that this thesis titled, “Rankings Arguments & Extensions for Sets of Attacking Arguments” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.

- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.

- Where I have consulted the published work of others, this is always clearly attributed.

- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.

- I have acknowledged all main sources of help.

- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Bhaskar Sinha

Date: 12 Aug 2020
“When you have something to say, silence is a lie.”

Jordan B. Peterson
A significant amount of work has been done on Argumentation, which allows one to model ideas and beliefs using arguments to draw conclusions and inferences. One of the focal points of research, as of late, has involved frameworks with sets of attacking arguments, where arguments can jointly attack other arguments in groups. While mechanisms for ranking arguments in binary abstract argumentation frameworks do exist, not many such methods exist that allow one to rank arguments in the context of sets of attacking arguments. As part of this thesis, we propose new ranking-based semantics for the specific context of frameworks with sets of attacking arguments and characterise them with respect to existing desirable properties defined in the literature. We also introduce a novel framework for decision-making that combines the notions of ranking-based semantics and extension-based semantics to arrive at conclusions and inferences. An application was implemented to demonstrate the proposed ranking-based functions and decision-making framework as well as to provide a visualisation of the argumentation framework. The developed application was evaluated by both experts and non-experts as having an intuitive design and a helpful visualisation of input argument frameworks.
Acknowledgements

I would like to express my deep and sincere gratitude to my supervisor Dr. Bruno Yun (University of Aberdeen) for granting me the opportunity to conduct research on the topic of Argumentation and for his continuous guidance and help in working on this thesis. In particular, for his patience and for helping me understand the concepts and notions pertaining to the research topic. I am extremely grateful for all that he has offered to me.

I am extremely grateful to my parents and sister for their love and for all the sacrifices they have made to enable me to have a chance to work on this research topic. I am extremely grateful for their patience, continuous guidance and love.
# Contents

Declaration of Authorship iii

Abstract vii

Acknowledgements ix

1 Introduction 1
   1.1 Argumentation-based Reasoning 1
   1.1.1 Arguments ........................................ 2
   1.1.2 Attacks ............................................ 2
   1.1.3 Acceptance ......................................... 2
   1.1.4 Applications ...................................... 3
   1.2 Argumentation with Sets of Attacking Arguments ............... 4
   1.3 Research Question .................................. 5
   1.4 Outline of the Thesis ................................ 5

2 Argumentation Frameworks 7
   2.1 Argumentation Frameworks and Their Semantics ............... 7
   2.2 Extension-based and Labeling-based Semantics ............... 8
   2.3 Ranking-based Semantics ................................ 10
   2.4 Summary .............................................. 12

3 A new framework for decision-making 13
   3.1 Ranking-Semantics for SETAFs 13
   3.1.1 Extending Existing Ranking-Semantics .................. 13
   3.1.2 Characterising Ranking-Semantics ..................... 16
   3.2 Combining Extensions and Rankings for Decision-Making .... 18
   3.3 Summary .............................................. 19

4 Implementation 21
   4.1 Technologies and Architecture 21
   4.2 ASPARTIX Input ...................................... 23
   4.3 Algorithms .......................................... 23
   4.3.1 Computing Rankings ................................ 23
   4.3.2 Computing Preferred Extensions ..................... 24
   4.4 User Interface ....................................... 27
   4.5 Feature Showcase & Testing 28
   4.5.1 Feature: File Upload & Argument & Attack Parsing ....... 28
   4.5.2 Feature: Rank Computation 28
   4.5.3 Feature: Extension Computation 29
   4.5.4 Feature: Visualisation 29
   4.5.5 Performance Measurements 29
   4.5.6 Testing ............................................ 30
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Dung Acceptability Semantics</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>An Argumentation Framework with Sets of Attacking Arguments</td>
<td>8</td>
</tr>
<tr>
<td>4.1</td>
<td>Web-App Architecture</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td>Candidate Tree Construction</td>
<td>26</td>
</tr>
<tr>
<td>4.3</td>
<td>Performance Measurement Trend</td>
<td>30</td>
</tr>
<tr>
<td>A.1</td>
<td>Landing Page</td>
<td>37</td>
</tr>
<tr>
<td>A.2</td>
<td>Demo Page</td>
<td>38</td>
</tr>
<tr>
<td>A.3</td>
<td>Response Populated</td>
<td>38</td>
</tr>
<tr>
<td>A.4</td>
<td>File Upload Input</td>
<td>38</td>
</tr>
<tr>
<td>A.5</td>
<td>Parsed Arguments &amp; Attacks</td>
<td>39</td>
</tr>
<tr>
<td>A.6</td>
<td>Extensions Table</td>
<td>39</td>
</tr>
<tr>
<td>A.7</td>
<td>Ranking Function Drop Down</td>
<td>39</td>
</tr>
<tr>
<td>A.8</td>
<td>Ranking Table</td>
<td>40</td>
</tr>
<tr>
<td>A.9</td>
<td>Argument Highlighting</td>
<td>40</td>
</tr>
<tr>
<td>A.10</td>
<td>Attack Highlighting</td>
<td>41</td>
</tr>
<tr>
<td>A.11</td>
<td>Extension Highlighting</td>
<td>41</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Nh-Categoriser strength values ........................................ 11
3.1 Average Burden-Based Function Strength Values ..................... 14
3.2 Min Burden-Based Function Strength Values ........................... 15
3.3 Discussion-based Semantic Values ....................................... 15
3.4 Mean Categoriser Strength Values ....................................... 16
3.5 Property Satisfied by the Ranking-based Semantics .................... 18
3.6 Ranking on Extensions w.r.t. Categoriser and Discussion-based Semantics .................................................. 18
4.1 Candidate Generation for Preferred Extensions .......................... 25
4.2 Preferred Extension Candidates .......................................... 25
4.3 Rankings and Arguments Tables ......................................... 28
4.4 Attacks and Extensions Tables ........................................... 28
4.5 Performance Measurements .............................................. 29
Dedicated to My Mother, Father & Sister, without whom I would not be where I am today and not have achieved all that I have
Chapter 1

Introduction

This first chapter introduces the concepts that we will be working with in this thesis, i.e. argumentation-based reasoning, argumentation frameworks with sets of attacking arguments and argumentation for decision-making. Next, we shall put forward the research topic chosen and finally, the outline for this thesis.

1.1 Argumentation-based Reasoning

Argumentation is the study of how conclusions can be reached through logical reasoning. It involves studying rules of inference, logic and procedural rules in both artificial and real world settings. It can also be considered a method which humans use to justify their solutions to problems, be it social or economic [1].

A significant amount of work has been done to analyse the structure of arguments and to build systems that can exchange arguments. For instance, Alvarado [2] and Birnbaum [3] proposed systems that could understand editorials and engage in political dialogues and Cohen, in [4], published an in-depth analysis of argument structure. However, more and more researchers are now focusing on how arguments interact by “distancing” themselves from the argument structure and how the arguments are built [5, 6, 7]. This is referred to as “abstract argumentation” in the literature.

Abstract argumentation was originally proposed as a way to understand how common people carry out their reasoning. The field of artificial intelligence has seen considerable work done involving the analysis of structure of arguments and building systems that can engage in the exchange of arguments. A statement would involve a set of arguments that propose an idea or a proposition.

In his seminal paper [1], Dung put forward an abstract, formal and simple theory that captured the notion of acceptability of arguments. His theory of argumentation was built upon an abstract argumentation framework (AAF) consisting of two elements: (1) a set of arguments and (2) a binary relation on the arguments representing the attack relations between said arguments. An argument is considered an abstract entity that represents data or a proposition. Toulmin provided an excellent philosophical account of the general structure of arguments in [8]. The relation between argumentation in the form of a dialogue game and classical monotonic logic was studied by Lorenz et al in [9].

Following this, Dung also put forward the concepts of conflict-freeness between arguments, admissibility of a set of arguments and the notions of extensions for any given framework. Such concepts allow one to generate inferences, in the form of “non-conflicting” sets of arguments, using the arguments and the attack relations between them [1]. The main intuition behind his argumentation-based reasoning,
inspired from existing work on Game Theory, is that a statement is believable or acceptable if it can be argued successfully against its attacking arguments. In other words, a statement is acceptable if an argument that is in its support is successfully defended from all arguments that attack it.

1.1.1 Arguments

As noted previously, arguments are representations of ideas, statements, facts or propositions that can vary from fully abstract to fully structured. The structure for such arguments varies according to the system used to build them and the format of the input data. For instance, in [10], the authors have an argument structure that start with basic facts and axioms and then develop further using rules, lemmas and theorems. Pollock focused mainly on building arguments from defeasible reasons and generating associated attacks called rebuts and undercuts [11]. Similarly, one of the most famous argumentation framework, the ASPIC+ framework [12], is composed of arguments that are built using strict and defeasible rules, ordinary premises and axioms. We can also recognise a plethora of argumentation frameworks built [13, 14, 15] following Besnard and Hunter’s deductive argumentation [16] approach where arguments are composed of premises and a conclusion.

In essence, the various models of argumentation used different methods for forming and structuring arguments. ASPIC+ allows one to use preferences to resolve attacks into defeats. On the other hand, in defeasible reasoning, arguments are constructed by starting from perceptual and memory states and then moving on to beliefs, from these beliefs onto new ones and so on and so forth. As part of this thesis, we will work in the context of abstract argumentation and we assume that the argumentation graph is given as an input.

1.1.2 Attacks

Attacks are the concrete representations of the conflicts between the ideas, propositions or even the inferences underlying the arguments. One example of such attacks can be the disagreement between arguments with conflicting conclusions [10]. As was noted previously, Dung’s abstract argumentation frameworks represent attacks using a binary relation on the arguments [1]. In the literature, rebuttals and defeats are also considered a variation of an attack represented in a different manner and for a different purpose (preferences).

In the ASPIC+ framework arguments can be attacked on a conclusion of an inference (rebutting attack), on an inference step itself (undercutting attack), or on an ordinary premise of the argument (undermining attack). An attack is called a defeat when an argument can be used as a counter argument to another argument, if said argument successfully attacks the other. Whether an attack from an argument on another succeeds as a defeat, depends on the relative strengths of said arguments, i.e. whether an argument is strictly stronger or more preferred to the other.

As is the case with arguments, attacks are represented in varying ways according to the needs and intuition of the framework being employed.

1.1.3 Acceptance

Here, we shall look at the notions of acceptability of arguments in an argumentation framework. Dung [1] put forward notions of acceptability wherein sets of arguments
1.1. Argumentation-based Reasoning

could be thought of as the inferences or conclusions of the arguments and their attack relations. The ideas of conflict-free argument sets, admissibility of arguments and extensions of argument frameworks allowed one to generate inferences and/or conclusions from a given argument framework. Several different types of such extensions, or admissible sets of arguments that can together be considered acceptable were initially put forward [1]:

- Complete
- Grounded
- Stable
- Preferred

For the definitions of these acceptability (or extension-based) semantics, please refer to Chapter 2. The argumentation frameworks in Figure 1.1 (extracted from [17]) demonstrate these acceptability semantics for an arbitrary argumentation framework. For a detailed introduction of these acceptability semantics (and other existing semantics), we kindly ask the reader to refer to the paper by Baroni et al [18].

1.1.4 Applications

Argumentation has been implemented in the form of varying different software and systems in multiple application fields. A prominent example is the work done on OSCAR in [19]. Other examples of implementations of abstract argumentation include Araucaria [20], the Zeno system [21] and the Rationale tool [22]. In [17], the
Chapter 1. Introduction

authors proposed an argumentation-based tool, called ECOBIOCAP, to select the most efficient packaging for the agri-food business.

The authors of [10] highlighted that argumentation has held a prominent role in the field of law wherein parties defend their own positions and challenge those of their opponents. In [23], the authors especially accentuated the significant research done involving computational modeling of argumentation for legal reasoning. Similarly, argumentation has been applied in the fields of medical diagnosis [24], health promotion [25], evidence aggregation [26] and design of software or other artefacts [27].

1.2 Argumentation with Sets of Attacking Arguments

The previous section introduced the concept of abstract argumentation frameworks as well as its applications and a few of its implementations. This section will look at how and why said framework has been generalised to allow sets of attacking arguments. The framework put forward by Dung is significant in that it allows one to model and structure arguments as well as their relations which in turn allows one to draw inferences and conclusions from said framework. However, the lack of the “and” connective between arguments, wherein sets of arguments could attack other arguments, is a hindrance when trying to generate frameworks modelling complex scenarios. Also, when building arguments or attacks from an underlying logical language that does not include conjunctions, one would not be able to model attacks corresponding to ternary conflicts. This can be achieved using argumentation frameworks with sets of attacking arguments (also called SETAFs) wherein arguments can jointly attack another. Furthermore, when instantiating arguments from inconsistent Datalog± knowledge bases, it has been shown that SETAFs can help reduce the number of arguments generated for such an argumentation framework considerably [28].

The aforementioned problems are addressed in [29] wherein modifications are done to the Dung framework that would allow for sets of attacking arguments. This is done by generalising Dung’s framework by allowing sets of arguments to attack a single argument. Modifications are made to the framework in the form of redefining argumentation systems, conflict free sets, acceptance and admissibility semantics in the context of a framework comprising sets of attacking arguments.

The following section addresses the concepts employed when having to generate decisions given an argumentation framework. As was noted in the previous section, several acceptance semantics in the form of extensions were put forward by Dung that allowed one to draw inferences and conclusions given a framework comprising of arguments and their attack relations. Such semantics offered a way to classify the arguments into two categories: those inside the extension and those outside of the extension. Please note that the labeling-based semantics [30, 18] are able to provide one more level of acceptability for the arguments by labeling the arguments either “in”, “out” or “undecided”. Indeed, although there is a bijection between extensions and labellings, labeling-based semantics are able to distinguish whether an argument that is outside of an extension is “undecided” or “out”.

As the demand for more gradual definitions of an argument acceptability grew, Amgoud et al. first proposed ranking-based semantics as a way to offer a more refined degree of acceptance for each argument in the framework as opposed to the previous three levels of acceptability (in, out and undecided) [6]. This is achieved using semantics that are able to assign rank to arguments based on the structure of the
1.3 Research Question

The following section explores the research topic chosen as part of this thesis. We previously introduced the concepts of argumentation frameworks, frameworks with sets of attacking arguments and semantics that allow one to make decisions. As part of this thesis, our aim is two-fold. First, we want to study how existing ranking-based semantics can be modified to allow them to function with SETAFs. Second, we want to create a framework for decision-making by combining the concepts of extension-based semantics and ranking-based semantics in the context of SETAFs.

The primary research question of this project is:

“How can we rank extensions in the context of SETAFs?”

Our research problem can be reformulated into a subset of more precise research questions as follows:

- Can we translate existing ranking-based semantics for AAFs to work in the context of SETAFs?
- Can we develop a framework that combines the notions of ranking-based and extension-based semantics to make decisions with SETAFs?
- Is it possible to develop a tool that could demonstrate such features?

1.4 Outline of the Thesis

This thesis is split into five chapters. The first chapter consists of an introduction wherein the concepts being worked with are introduced. The second chapter involves defining the background work of the several argumentation frameworks used in this thesis as well as their ranking-based and extension-based semantics that will be worked with as part of this thesis. Following this, in the third chapter, we explore the “aggregation” modifications that are proposed on the existing categoriser, burden-based and discussion-based ranking-based semantics and characterise the desirable properties, defined by Yun et al. [33], that are satisfied by these new ranking-based semantics. We also introduce a new framework (inspired from
the work of Bonzon et al. [34] and Yun et al. [35]) for decision-making that combines extension-based and ranking-based semantics. Next, the fourth chapter explores the tool implemented as part of this project. We look at the features the tool provides as well as algorithms that are used when computing argument rankings and extensions in the context of SETAFs. We also look at the user interface and finally we showcase the features implemented and perform testing and evaluation on the tool. Finally, the fifth chapter looks at the related research, recalls the results and provides some conclusions that can be drawn from the research done as well as possible future research avenues.
Chapter 2

Argumentation Frameworks

2.1 Argumentation Frameworks and Their Semantics

In an abstract argumentation framework (AAF), the information is represented through a set of abstract arguments that represent statements or propositions. Conflicts between these arguments are represented through a binary relation on the set of arguments. In other words, such a framework is represented using a directed graph wherein nodes are arguments and edges are the attack relations [1].

**Definition 1 (AAF)** An AAF is a pair $AF = (A, R)$ where $A$ is a set of abstract arguments and $R$ is a binary relation on $A$, called an attack relation.

The objective for such a framework was to facilitate abstraction away from the underlying language and its structure of arguments. Attack relations in such a framework do not allow for the “and” connective between arguments. The following generalisation allows for relations between attacking sets of arguments and the attacked argument [29]. An Argumentation Framework with sets of attacking arguments, or SETAF, is composed of a set of arguments and a set of group attacks. Such an argument framework can be considered a generalization of AAFs described above.

**Definition 2 (SETAF)** A SETAF is a tuple $AF = (A, C)$ where $A$ is a set of arguments and $C \subseteq (2^A \setminus \emptyset) \times A$ is a set of sets of attacking arguments. We write $S \triangleright a$ to denote that $S \subseteq A$ attacks $a \in A$. Conversely, we write $S \not\triangleright a$ when it is not the case that $S \triangleright a$. $S$ is called the attacker of $a$.

**Example 1** Let $AF_1 = (A, C)$ such that $A = \{a, b, c, d, e, f\}$ and $C = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8\}$, where $r_1 = \{a, c, d\} \triangleright b, r_2 = \{a, b\} \triangleright c, r_3 = \{b\} \triangleright d, r_4 = \{c, e\} \triangleright d, r_5 = \{d\} \triangleright e, r_6 = \{b, f\} \triangleright e, r_7 = \{a\} \triangleright f$ and $r_8 = \{d\} \triangleright f$. $AF_1$ is represented in Figure 2.1.

For the purposes of this thesis, let us consider the fictional case wherein the objective is to determine whether a person should be allowed drink alcohol. In such a case, a person is allowed to drink alcohol if aged over 18 years old or if accompanied by an adult or has parent permission. The person does not require parent permission if they are married. Finally, the person is allowed to drive if aged over 18 years old or has parent permission. Using these rules, the following arguments and their implications are put forward:

- **a**: “aged under 18 years old”
- **b**: “allowed to drink alcohol”
- **c**: “not accompanied by an adult”
- **d**: “no parent permission”
Chapter 2. Argumentation Frameworks

Figure 2.1: An Argumentation Framework with Sets of Attacking Arguments

- $e$: “married”
- $f$: “allowed to drive”

In Definition 3, we formally define the set of direct attackers of an argument.

**Definition 3 (Direct Attacker)** Given a SETAF $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$ and $x \in \mathcal{A}$, the set of direct attackers of $x$ is $R_1(x) = \{Y \subseteq \mathcal{A} \text{ such that } (Y, x) \in \mathcal{C}\}$.

Such a set of arguments can also be referred to as the direct attackers of said argument and is often referred to as $R^-(x)$.

In Definition 4, we define a direct defender of an argument $x$ as a set of arguments that attacks at least one direct attacker of $x$.

**Definition 4 (Direct Defender)** Given a SETAG, $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$, and $x \in \mathcal{A}$, the set of direct defenders of $x$ is $R_2(x) = \{Z \subseteq \mathcal{A} \text{ such that } z \in Z, y \in R_1(x), \text{ we have } (z, y) \in \mathcal{C}\}$.

Such a set of arguments can also be referred to as the direct defenders of said argument and is often referred to as $R^+(x)$.

2.2 Extension-based and Labeling-based Semantics

Given a SETAF, one can make use of several available acceptability semantics that allow one to generate a set of arguments that are together acceptable, given the arguments as well as their attack relations. The following definition introduces such semantics for a SETAF $\mathcal{AF}$ and $\mathcal{S} \subseteq \mathcal{A}$ [29].
Definition 5 (Extension-based semantics) Given an argumentation framework $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$, a set of arguments $S \subseteq \mathcal{A}$ is conflict-free in $\mathcal{AF}$ iff it does not attack itself. Formally, there is no $a \in S$, $S_1 \subseteq S$ such that $S_1 \triangleright a$. We say that $S$ defends an argument $a \in \mathcal{A}$ iff for every $Y \in 2^\mathcal{A}$ such that $(Y, a) \in \mathcal{C}$, there exists $y \in Y$, $S_1 \subseteq S$ such that $(S_1, y) \in \mathcal{C}$. A conflict free set $S$ is admissible if it defends all its arguments against each of their attackers. Such a set is:

- complete if each argument defended by $S$ belongs to $S$
- preferred if it is a maximal (w.r.t. $\subseteq$) admissible set of $\mathcal{AF}$
- stable if it attacks each argument in $\mathcal{A} \setminus S$
- grounded if it is the minimal (w.r.t. $\subseteq$) complete extension of $\mathcal{AF}$

Example 2 (Cont’d Example 1) Referring back to $\mathcal{AF}_1$ defined in Example 1, the

- Complete extensions comprise of $\{a\}$, $\{a,c,d\}$ and $\{a,b,e\}$
  - aged under 18 years old
  - aged under 18 years old, not accompanied by an adult and no parent permission
  - aged under 18 years old, allowed to drink alcohol and married
- Preferred extensions comprise of $\{a,c,d\}$ and $\{a,b,e\}$
  - aged under 18 years old, not accompanied by an adult and no parent permission
  - aged under 18 years old, allowed to drink alcohol and married
- Stable extensions comprise of $\{a,b,e\}$ and $\{a,c,d\}$
  - aged under 18 years old, not accompanied by an adult and no parent permission
  - aged under 18 years old, allowed to drink alcohol and married
- Grounded extension is $\{a\}$
  - aged under 18 years old

Extensions are generated in terms of labelings assigned to arguments that constitute the argumentation framework. Arguments are assigned one of the following three labelings

- IN Arguments that are in an extension
- OUT Arguments that out of an extension
- UNDEC Arguments that have not been assigned a labeling

Notation 1 (Set of Extensions) $\mathcal{E}^\text{pr}_{\mathcal{AF}}$ (resp. $\mathcal{E}^\text{st}_{\mathcal{AF}}$ and $\mathcal{E}^\text{gr}_{\mathcal{AF}}$) denotes the set of preferred (resp. stable and grounded) extensions generated for a SETAF $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$

Notation 2 (Argument Labeling) $\text{IN}_\mathcal{E}$ (resp. $\text{OUT}_\mathcal{E}$ and $\text{UNDEC}_\mathcal{E}$) denotes the set of arguments that are labeled IN (resp. labeled OUT and UNDEC) in an extension $\mathcal{E}$.
Chapter 2. Argumentation Frameworks

The extension-based acceptability semantics described above offer an IN or OUT labeling in terms of acceptability of arguments. While one can generally work with such semantics, complex scenarios and certain applications would require a different approach to generate acceptability of arguments.

For instance, when attempting to generate conclusions relative to an argumentation framework consisting of arguments describing a case of law or the diagnosis of a medical condition for a patient [23, 24, 25], modelling and generating conclusions for the same using only IN and OUT acceptability semantics can lead to certain conclusions being rejected outright or labeled as UNDEC. It should be noted that though extensions outright define arguments that are \( \text{in} \) and can together be accepted, labelings provide a better idea of the state of arguments for each extension. Labelings allow one to define arguments that have not been assigned a labeling by labeling them as “undecided”. One cannot obtain such information using only the extension semantics. One would infer a better idea of the state of arguments by assigning a labeling to all arguments for each extension. In such cases, it would be beneficial to produce a degree of acceptability for all possible conclusions which in turn would allow an end user to better gauge and therefore choose the best possible conclusion using said degree of acceptability [36].

2.3 Ranking-based Semantics

The following section introduces the notion of ranking arguments in order of most acceptable (least attacked) to least acceptable (most attacked). Offering a degree of acceptability would provide a better means of generating most acceptable arguments as opposed to arguments that are outright accepted or rejected.

**Definition 6 (Ranking)** A ranking on a set \( X \) is a binary relation \( \preceq \) on \( X \) such that \( \preceq \) is total, reflexive and transitive.

We use the notation \( a \preceq b \) to show that \( a \) is at least as acceptable as \( b \). \( a \preceq b \) iff \( a \preceq b \) and \( b \not\preceq a \). Finally, we use \( a \simeq b \) iff \( a \preceq b \) and \( b \preceq a \). The above notation is followed since for certain ranking semantics, the lower the strength values of an argument, the more acceptable it is. This may not be true of the functions defined here but the same is followed for the sake of consistency.

**Definition 7 (Ranking-based semantics)** A ranking-based semantics \( \sigma \) associates to any SETAF \( AF = (A, \mathcal{C}) \) a ranking \( \preceq_{AF} \) on \( A \).

In order to generate rankings on arguments, the nh-categoriser may be used [33]. The following nh-categoriser function assigns a strength value to each argument by considering only the min strength value of arguments in an attack since the author’s intuition is that such an attack would be void if any of the arguments are removed from the attack. It follows therefore that the strength of a set attack is the strength value of the argument with the lowest strength [33].

**Definition 8 (nh-categoriser function)** The nh-categoriser function \( \text{Cat} : A \to [0, 1] \) is defined as \( \forall x \in A : \)

\[
\text{Cat}(x) = \begin{cases} 
1 & \text{if } R_1(x) = \emptyset \\
\frac{1}{1 + \sum_{S \in R_1(x)} \min_{s \in S} \text{Cat}(s)} & \text{otherwise}
\end{cases}
\]
The categoriser-based ranking-based semantics is based on the nh-categoriser function, wherein an argument is ranked lower if the value returned by the nh-categoriser function is lower. It should be noted that the categoriser-based ranking-based semantics was the first ranking-semantics defined for the SETAF framework.

**Definition 9 (Categoriser-based ranking-based semantics)** The categoriser-based ranking semantic associates to any $AF = (A, C)$ a ranking $\preceq_{Cat}^{AF}$ on $A$ such that $\forall x, y \in A, x \preceq_{Cat}^{AF} y$ iff $Cat(x) \geq Cat(y)$

**Example 3 (Cont’d Example 1)** The nh-categoriser function would assign the strength values as specified in Table 2.1 to arguments

<table>
<thead>
<tr>
<th>Cat(a)</th>
<th>Cat(b)</th>
<th>Cat(c)</th>
<th>Cat(d)</th>
<th>Cat(e)</th>
<th>Cat(f)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.69</td>
<td>0.59</td>
<td>0.45</td>
<td>0.54</td>
<td>0.41</td>
</tr>
</tbody>
</table>

**Table 2.1:** Nh-Categoriser strength values

Using assigned values, the following ranking on arguments is generated with the categoriser-based ranking-based semantics:

$a \preceq_{Cat}^{AF} b \preceq_{Cat}^{AF} c \preceq_{Cat}^{AF} e \preceq_{Cat}^{AF} d \preceq_{Cat}^{AF} f$

Next, we shall look at the existing ranking-based semantics that have been defined for abstract argumentation frameworks. In this thesis, we decide to only focus on the discussion and burden-based ranking-based semantics. These two semantics were chosen because of their simplicity, efficiency and intuitions.

The burden-based semantics assigns at each step $i$, a burden number to every argument according to the burden of its direct attackers.

**Definition 10 (Burden-Based Function)** For any $AF = (A, R)$ for any $x \in A$ and $i \in \mathbb{N}$,

$$Bur_i(x) = \begin{cases} 1 & \text{if } i = 0 \\ 1 + \sum_{(b,x) \in R} \frac{1}{Bur_{i-1}(b)} & \text{otherwise} \end{cases}$$

Next, we shall look at the discussion-based function to rank arguments. The discussion-based function assigns at each step $i$, the count of paths of size $i$ ending on each argument. It is defined as per the following

**Definition 11 (Discussion-Based Function)** For any $AF = (A, R), x \in A$ and $i \in \mathbb{N}$

$$Dis_i(x) = \begin{cases} -|R^+(x)| & \text{if } i \text{ is odd} \\ |R^-(x)| & \text{if } i \text{ is even} \end{cases}$$

where $R^+(x)$ (resp. $R^-(x)$) indicates the set of all possible paths of attacks (resp. defences) of an argument $x$ of length $i$. A path of size $n$ from elements $a$ to $b$ is defined as a sequence $s = (a_1, a_2, a_3...a_n)$ of elements in $A$, such that $a_1 = a$, $a_n = b$ and $\forall i \in \{1,2,3,...n\}$, we have that $(a_{i-1}, a_i) \in R$.

For both, burden-based and discussion-based semantics, ranks are assigned by lexicographically comparing the burden or path counts at each step for all arguments.
The above section introduces the background of the concepts that are being worked on or with as part of this thesis. The notions of abstract argumentation frameworks (AAFs) as well as their generalization in the form of frameworks comprising of sets of attacking arguments (SETAFs) and the ideas inherent to them are introduced and defined. We also looked at the semantics available to rank arguments in such Argument Frameworks. In the following section, we shall propose new functions that may be employed to generate ranks on arguments for such argument frameworks.

2.4 Summary

Chapter 2 in a Nutshell

- We introduced Dung’s abstract argumentation framework and the existing semantics (extension-based, labeling-based and ranking-based)
- We showed how AAFs can be extended with sets of attacking arguments
- We recalled several ranking-based semantics for AAFs and SETAFs:
  - Categoriser
  - Burden-based semantics
  - Discussion-based semantics
Chapter 3

A new framework for decision-making

The previous chapters defined the notions of the argumentation frameworks as well as their semantics that we will be working with as part of this thesis. In the following section, we will put forward new ranking functions that work with SETAFs. Further, we shall explore a method that allows one to generate a Ranking on Extensions for a given Argumentation Framework using the generated rankings on Arguments.

In order to translate existing ranking-based semantics to work with SETAFs, we would be computing the strength of a set of attacking arguments as opposed to the strength of an attack comprising one argument, as is the case for Abstract Argumentation Frameworks. The computation of such a strength value is usually done using an aggregation function. For instance, the categoriser-based ranking-based semantics [33], assigns the attack strength, the minimum strength value of arguments that form said attack. In this section, several new functions are put forward that can be considered extensions of existing ranking-based semantics for AAFs that compute strengths of attacks by applying different aggregation functions to strengths of members of an attack. This allows one to apply different weights to members of each attack. We shall use the aggregation functions ‘Min’ and ‘Mean’ in the context of existing ranking-based semantics where applicable.

3.1 Ranking-Semantics for SETAFs

In this section, we look at the methodologies employed in order to translate existing binary argumentation framework ranking-based semantics to work with SETAFs. We study the burden-based and discussion-based ranking-based semantics as described in [6]. Unless otherwise mentioned, the following functions use the concept of lexicographical ordering on the generated sets of strengths values for each argument. Such an order is a generalization wherein words are alphabetically ordered based on the alphabetical order of the component letters. In the following case, i.e. sets of numerical strength values of arguments, sets are ordered by comparing component numbers in each set against their counterpart numbers in other sets.

3.1.1 Extending Existing Ranking-Semantics

As part of this thesis, said burden-based semantics is extended to work with SETAFs using the following two ranking-based semantics. One of which is the Average Burden-Based function which may be defined as the following
Definition 12 (Average Burden-Based Function) For a SETAF $\mathcal{A.F} = (A, C)$ for any $x \in A$ and $i \in \mathbb{N}$,

$$Bur_i(x) = \begin{cases} 1 & \text{if } i = 0 \\ 1 + \sum_{S \in R_i(x)} \frac{|S|}{\sum_{b \in S} Bur_{i-1}(b)} & \text{otherwise} \end{cases}$$

The Average Burden-Based ranking function uses the average of strengths values of arguments of attackers so as to consider the strengths of all attackers of said argument. This is in contrast to the nh-categoriser function which considers that the strength of a set of attackers is the strength of its argument with minimum strength value. This allows all of the argument strengths of attackers to count towards the strength value of an argument.

Example 4 (Cont’d Example 1) Referring back to $\mathcal{A.F}_1$ defined in Example 1, the Average Burden-Based function would assign burden values as specified in Table 3.1 to arguments for given steps.

<table>
<thead>
<tr>
<th>Step</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1.5</td>
<td>1.67</td>
<td>1.9</td>
<td>1.73</td>
<td>2.33</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1.66</td>
<td>1.8</td>
<td>2.25</td>
<td>2.05</td>
<td>2.53</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1.59</td>
<td>1.75</td>
<td>2.12</td>
<td>1.92</td>
<td>2.44</td>
</tr>
</tbody>
</table>

Table 3.1: Average Burden-Based Function Strength Values

Scores for strength values are calculated until the values converge or they differ from the previous strength value by a certain amount. Usually, this number is of the order of 0.01. Using burden values generated above, the following ranking is generated by lexicographically comparing the sets of burden values

$$a \preceq_{\mathcal{A.F}} b \preceq_{\mathcal{A.F}} c \preceq_{\mathcal{A.F}} d \preceq_{\mathcal{A.F}} e \preceq_{\mathcal{A.F}} f$$

Similarly, the Min Burden-Based function is defined as follows

Definition 13 (Min Burden-Based function) For a SETAF $\mathcal{A.F}_1 = (A, C)$ for any $x \in A$ and $i \in \mathbb{N}$,

$$Bur_i(x) = \begin{cases} 1 & \text{if } i = 0 \\ 1 + \sum_{S \in R_i(x)} \frac{1}{\min_{b \in S} Bur_{i-1}(b)} & \text{otherwise} \end{cases}$$

Example 5 (Cont’d Example 1) Referring back to $\mathcal{A.F}_1$ defined in Example 1, the Min Burden-Based function would assign burden values as specified in Table 3.2 to arguments for given steps.
3.1. Ranking-Semantics for SETAFs

Using burden values generated, the following ranking is generated by lexicographically comparing the burden values

\[ a \preceq_{\mathcal{A}_F} b \preceq_{\mathcal{A}_F} c \preceq_{\mathcal{A}_F} e \preceq_{\mathcal{A}_F} d \preceq_{\mathcal{A}_F} f \]

The Discussion-based semantics, as seen in the previous chapter, ranks arguments by comparing the number of paths ending in said argument. In case multiple arguments have the same number of attackers or defenders for a given path size, the path size is increased further until a difference is found. The definition of path is slightly changed to allow this function to work with SETAFs in the following manner

**Definition 14 (Paths for SETAFs)** A path of size \( n \) from elements \( a \) to \( b \) is defined as a sequence \( s = (S_1, S_2, S_3...S_n) \) of set of arguments in \( \mathcal{A} \), such that \( a \in S_1 \), \( b \in S_n \) and \( \forall i \in \{1, 2, 3,...n\} \), we have that \( c \in S_i \) and \( (S_{i-1}, c) \in \mathcal{C} \).

This semantic does not require further translation to allow it to work with SETAFs since it only requires generating counts of path sizes for each argument and therefore works out of the box.

**Example 6 (Cont’d Example 1)** Referring back to \( \mathcal{A}_F \) defined in Example 1, the discussion-based semantic would assign values as specified in Table 3.3 to arguments for given steps.

<table>
<thead>
<tr>
<th>Step</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1.83</td>
<td>2.33</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2.05</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1.99</td>
<td>2.49</td>
</tr>
</tbody>
</table>

**Table 3.2: Min Burden-Based Function Strength Values**

Using the path counts generated, the following ranking is generated by lexicographically comparing the strength values

\[ a \preceq_{\mathcal{A}_F} b \preceq_{\mathcal{A}_F} c \preceq_{\mathcal{A}_F} f \preceq_{\mathcal{A}_F} d \preceq_{\mathcal{A}_F} e \]

In addition to the above semantics, the following semantic is proposed which can be considered a modification of the nh-categoriser defined previously in Definition 9.

**Definition 15 (Mean Categoriser)** The mean categoriser \( \text{Mct} : \mathcal{A} \to [0,1] \) is defined as \( \forall x \in \mathcal{A} \)

\[
\text{Mct}(x) = \begin{cases} 
1 & \text{if } R_1(x) = \emptyset \\
1 + \frac{1}{\sum_{S \in R_1(x)} \left( \frac{\sum_{s \in S} \text{Mct}(s)}{|S|} \right)} & \text{otherwise}
\end{cases}
\]
Example 7 (Cont’d Example 1) Referring back to $\mathcal{AF}_1$ defined in Example 1, the Mean Categoriser semantic would assign values as specified in Table 3.4 to arguments for given steps.

<table>
<thead>
<tr>
<th>Mct(a)</th>
<th>Mct(b)</th>
<th>Mct(c)</th>
<th>Mct(d)</th>
<th>Mct(e)</th>
<th>Mct(f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.59</td>
<td>0.55</td>
<td>0.47</td>
<td>0.51</td>
<td>0.41</td>
</tr>
</tbody>
</table>

Table 3.4: Mean Categoriser Strength Values

Using strength values generated, the following ranking is generated

\[ a \preceq_{\mathcal{AF}} b \preceq_{\mathcal{AF}} c \preceq_{\mathcal{AF}} e \preceq_{\mathcal{AF}} d \preceq_{\mathcal{AF}} f \]

3.1.2 Characterising Ranking-Semantics

The following section explores the properties that are satisfied by the Ranking functions defined in the previous section. The availability of various ranking functions allows one to choose a function that can best solve the problem at hand. Depending on the problem being addressed, one might choose from any of the available functions mentioned previously. By specifying the properties that are satisfied by each of the functions can help one choose the function that can best solve the problem at hand. The properties the functions are evaluated against here are as defined in [33].

Definition 16 (Isomorphism) An Isomorphism between two argumentation frameworks $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$ and $\mathcal{AF}' = (\mathcal{A}', \mathcal{C}')$ is a bijective function $\lambda : \mathcal{A} \rightarrow \mathcal{A}'$ such that for every $S \in 2^\mathcal{A}$ and $a \in \mathcal{A}$, $(S, a) \in \mathcal{C}$ iff $\{\lambda(s) | s \in S\}, \lambda(a) \in \mathcal{C}'$. With a slight abuse of notation, we will note that $\mathcal{AF}' = \lambda(\mathcal{AF})$ [33].

The Abstraction property states that the name of arguments should not be taken into account in the ranking.

Property 1 (Abstraction) We say that $\sigma$ satisfies Abstraction if for any $\mathcal{AF}, \mathcal{AF}'$ and isomorphism $\lambda$ such that $\mathcal{AF}' = \lambda(\mathcal{AF})$, we have $a \preceq_{\mathcal{AF}} b$ iff $\lambda(a) \preceq_{\mathcal{AF}'} b$ [33].

For a given $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$ and $\mathcal{AF}' = (\mathcal{A}', \mathcal{C}')$, we have $\mathcal{AF} \oplus \mathcal{AF}'$ as the argumentation framework $(\mathcal{A} \cup \mathcal{A}', \mathcal{C} \cup \mathcal{C}')$.

The Indepance property states that two arguments with no paths connecting them should not influence each other.

Property 2 (Independance) We say that $\sigma$ satisfies the Independence property iff for any $\mathcal{AF}, \mathcal{AF}'$ such that $\mathcal{AF}' = \lambda(\mathcal{AF})$, we have $a \preceq_{\mathcal{AF}} b$ iff $\lambda(a) \preceq_{\mathcal{AF}'} b$ [33].

The Void Precedence property states that non-attacked arguments should be ranked higher than attacked arguments.

Property 3 (Void Precedence) We say that $\sigma$ satisfies Void Precedence iff for any $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$ and $(a, b) \in \mathcal{A}$ such that $R^{-1}_1(a) = \emptyset$ and $R^{-1}_1(b) \neq \emptyset$, we have $a \prec_{\mathcal{AF} \oplus \mathcal{AF}'} b$ [33].

The Self Contradiction property states that self-contradicting arguments should be ranked lower than non self-contradicting arguments.
Property 4 (Self Contradiction) We say that $\sigma$ satisfies Self Contradiction iff for any $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$ and $a, b \in \mathcal{A}$ such that there exists $S_1 \in R^-_1(a)$ with $a \in S_1$ and there exists no $S_2 \in R^-_1(b)$ with $b \in S_2$, we have $b \prec_{\mathcal{AF}} a$ [33].

The Cardinality Precedence property states that if an argument $a$ has more attackers than an argument $b$, then $a$ should be ranked lower than $b$.

Property 5 (Cardinality Precedence) We say that $\sigma$ satisfies Cardinality Precedence iff for any $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$ and $a, b \in \mathcal{A}$ such that $|R^-_1(a)| > |R^-_1(b)|$, we have $b \prec_{\mathcal{AF}} a$ [33].

The Defense Precedence property states that if $a, b$ are two arguments with the same number of attackers and $a$ is defended but $b$ is not then $a$ should be ranked higher than $b$.

Property 6 (Defense Precedence) We say that $\sigma$ satisfies Defense Precedence iff for any $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$ and $a, b \in \mathcal{A}$ such that $|R^-_1(a)| = |R^-_1(b)|$ and $|R^-_2(a)| = \emptyset$ and $|R^-_2(b)| \neq \emptyset$, we have $a \prec_{\mathcal{AF}} b$ [33].

The Total property states that two arguments should always be comparable.

Property 7 (Total) We say that $\sigma$ satisfies the Total property iff for any $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$ and $a, b \in \mathcal{A}$, we have $a \preceq_{\mathcal{AF}} b$ or $b \prec_{\mathcal{AF}} a$ [33].

The Non-Attacked Equivalence property states that two non attacked arguments should be ranked equivalently.

Property 8 (Non-Attacked Equivalence) We say that $\sigma$ satisfies Non-Attacked Equivalence iff for any $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$ and $a, b \in \mathcal{A}$ such that $R^-_1(a) = R^-_1(b) = \emptyset$, we have $a \simeq_{\mathcal{AF}} b$ [33].

Definition 17 (Simple & Distribute Defense) For a $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$ and $a \in \mathcal{A}$, the defense of $a$ is simple iff for every direct defender $S$ of $a$, there is a unique $S' \in R^-_1(a)$ and $s' \in S'$ such that $(S, s') \in \mathcal{C}$. The defense of $a$ is distributed iff for every direct attacker $S$ of $a$, there is at most one direct defender $S'$ of $a$ such that $(S', s) \in \mathcal{C}$ where $s \in S$ [33].

The Distributed Defense property states that if an argument $a$ has a simple and distributed defense and an argument $b$ has a simple but not distributed defense, then $a$ should be ranked higher than $b$.

Property 9 (Distributed Defence) We say the $\sigma$ satisfies the Distributed Defence iff for any $\mathcal{AF} = (\mathcal{A}, \mathcal{C})$ for any $a, b \in \mathcal{A}$ such that $|R^-_1(a)| = |R^-_1(b)|$ and $|R^-_2(a)| = |R^-_2(b)|$ and that the defense of $a$ is simple and distributed and that the defense of $b$ is simple but not distributed, we have $a \prec_{\mathcal{AF}} b$ [33].

The following table shows the properties satisfied by the NHC(nh-categoriser), ABB(average burden-based), MBB(min burden-based), DB(discussion-based) and MC(mean categoriser) ranking-based semantics defined above.


**Chapter 3. A new framework for decision-making**

<table>
<thead>
<tr>
<th>Property</th>
<th>NHC</th>
<th>ABB</th>
<th>MBB</th>
<th>DB</th>
<th>MC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstraction</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Independence</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Void Precedence</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Self Contradiction</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Cardinality Precedence</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Defence Precedence</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Total</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Non-Attacked Equivalence</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Distributed Defence</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

*Table 3.5: Property Satisfied by the Ranking-based Semantics*

### 3.2 Combining Extensions and Rankings for Decision-Making

In the cases where multiple extensions are generated, one can make use of the rankings on arguments to generate a ranking on extensions. Such a ranking allows one to better choose the best extension(s) from the list of generated extensions. The function approach below is inspired from the work on ranking extensions defined in the literature [34, 35].

In order to generate a ranking on extensions, the rankings of arguments are used in conjunction with the presence of said arguments in a given extension to rank extensions. The ranks of arguments labeled IN for extensions are compared against other extensions to generate rankings.

Here we shall put forward a new framework that can be employed to generate a ranking on a set of extensions given an argumentation framework with ranking on arguments and its set of extensions.

**Definition 18 (Extension Ranking function)** An extension ranking function $F_{N}$ associates to a set of extensions $E_{AF}$ a ranking $\preceq_{F_{N}}^{AF}$ on $E_{AF}$

The Lexicographical Extension ranking function is proposed that ranks extensions by lexicographically comparing the ranks of IN labeled arguments of each extension.

**Definition 19 (Lexicographical Extension Ranking function)** For a given SETAF $AF = (A,C)$ and $E_{AF}^{x}$ the corresponding sets of extensions for a semantics $x$. The $F_{N}^{x}$n assigns a ranking $\preceq_{Lex}^{AF}$ to $E_{AF}^{x}$ s.t. if $Ext_{1}, Ext_{2} \in E_{AF}^{x}$, $S_{1} \subseteq 1N_{Ext_{1}}$, $S_{2} \subseteq 1N_{Ext_{2}}$ and $S_{1} \preceq_{Lex} S_{2}$, then $Ext_{1} \preceq_{Lex} Ext_{2}$

**Example 8 (Cont’d Example 1)** Referring back to $AF_{1}$ defined in Example 1, $F_{N}^{x}$n assigns the rankings as specified in Table 3.6 to generated preferred extensions using the Categoriser and discussion-based semantics.

<table>
<thead>
<tr>
<th>$F_{N}^{x}<em>{Lex}(E</em>{AF}^{pr})$</th>
<th>IN</th>
<th>OUT</th>
<th>UNDEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a, b, e$</td>
<td>$c, d, f$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>$a, c, d$</td>
<td>$b, e, f$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

*Table 3.6: Ranking on Extensions w.r.t. Categoriser and Discussion-based Semantics*
Per the descriptions of the arguments as specified in Example 1, the labelings generated imply either of the following:

- **Extension 1**
  - a aged under 18 years old
  - b allowed to drink
  - e married

- **Extension 2**
  - a aged under 18 years old
  - c not accompanied by an adult
  - d no parent permission

As can be seen in Table 3.6, depending on the argument ranking function chosen, different rankings are assigned to labelings. In case the categoriser is chosen, Extension 1 is ranked higher whereas when the discussion-based semantics is chosen, Extension 2 is ranked higher.

### 3.3 Summary

Chapter 3 in a Nutshell

- We translated existing ranking-based semantics for SETAFs and proposed four new ranking-based semantics:
  - Average burden-based function
  - Min burden-based function
  - Discussion-based function
  - Mean categoriser

- We studied the satisfaction of existing desirable properties by our new ranking-based semantics

- We introduced a new framework for ranking extensions using the rankings on arguments and the set of extensions for SETAFs
Chapter 4

Implementation

The previous sections explored the desired outcomes for a tool implementing ranking function. The following section addresses the methodologies, algorithms followed in order to build such a tool. In the first section below, we shall look at the technologies used as well as the architecture implemented. Next, we shall address the format an input file for such a tool should follow. Next, the algorithms used to compute rankings and extensions are addressed, followed by a description of the desired User Interface. Finally, we show the features implemented by means of screenshots of the implemented tool, the testing conducted and the evaluation performed in terms of possible Scalability and the User Experience collected by means of a feedback form.

4.1 Technologies and Architecture

The following section explores the technologies and architectures employed in the implementation of the concepts outlined in the previous section. As mentioned previously, the development of a web-app is proposed that allows a user to generate rankings and extensions given a SETAF.

The server-side programming language used is Java 1.8. It allows for the use of Streams that enables faster parallel processing of lists and maps as and when required as well as Executors that enable multi-threaded processing. The Web-App is developed using the Spring Web MVC framework. Spring MVC provides a Model-View-Controller architecture that enables one to develop flexible and loosely coupled web applications. Spring Boot is not used here since Spring MVC allows for a greater amount of customization of the configuration of the Web-App. The Model-View-Controller architecture offered by Spring allows for fast, flexible and highly reusable code development. Such an architecture allows for the separation of Input, Business and UI logics.

The User Interface (UI) is developed using JSP, JavaScript and jQuery. Java Server Pages (JSP) allow for the dynamic construction of web pages and works well with the Spring framework. jQuery allows for easy modification of elements on an HTML page without having to work with the DOM for any given page. Argument Frameworks are visualised using the arbor.js library that is built on top of jQuery. The Arbor library allows for easy visualisation of nodes as well as edges between said nodes. It also allows for modification of colors of nodes as well as edges which in turn would allow for highlighting arguments, their attack relations as well as arguments in a given extension. A great function provided by this library is that it organises input arguments and edges by spacing them out as evenly as possible while at the same time allowing for inputs like gravity and repulsiveness of the system and mass of nodes.
In cases where the number of arguments and attacks are high, the processing time to generate extensions can become exponentially high resulting in HTTP requests to the server timing out after 30 seconds. To combat this, WebSockets are used to allow the server to process extension generation in the background. To achieve this, SockJS and STOMP are used on the front end for clients. On the server side, the Spring WebSockets and Spring Messaging libraries are used. Websockets are a communication protocol that allow for full duplex communication over TCP connections. STOMP, Simple Text Oriented Messaging Protocol, is a protocol that allows for messages to be sent and received between a client and servers using sockets. SockJS is a client side library written in JavaScript that allows clients to establish socket connections with clients and servers. Finally, Spring Sockets is a library that enables Socket connections to be established using a Spring MVC server and the Spring Messaging library allows servers to send and receive messages from clients.

Finally, the Web-App is hosted on Heroku. Heroku is a PaaS (Platform as a Service) that allows users to run applications in the cloud. The Web-App developed is compiled into a WAR file which is in turn used to deploy the application onto Heroku. Heroku was chosen as the hosting service since it provides a free tier with limited database storage and one dyno. Since this app does not require any storage capabilities, Heroku satisfies the requirements for this app. Conversely, since the app is running on a free tier server, the dyno goes to sleep after one hour of inactivity. As a result, the web-app might be at times slow to load up since the server would have to wake up the inactive dyno. Similar alternatives such as AWS and Google Cloud can be used too but I am more comfortable working with Heroku given my past experience with it.

Figure 4.1 shows the architecture of the Web-App implemented.
4.2 ASPARTIX Input

For our implementation, inputs from user are accepted in the ASPARTIX format [https://www.dbai.tuwien.ac.at/research/argumentation/aspartix/]. This format consists of a sequence of statements that either indicate an argument, target of an attack or the support of an attack. The following is an example of such an input file:

1. `arg(a).` ... a is an argument
2. `att(x,b).` ... the attack named x attacks b
3. `mem(x,a).` ... the argument a is in support of attack x

Example 9 (Cont’d Example 1) Referring back to Example 1, the ASPARTIX file for such an argument framework would look like the following:

```
att(r1,b). mem(r1,a). mem(r1,c). mem(r1,d).
att(r2,c). mem(r2,a). mem(r2,b).
att(r3,d). mem(r3,b).
att(r4,d). mem(r4,c). mem(r4,e).
att(r5,e). mem(r5,d).
att(r6,e). mem(r6,b). mem(r6,f).
att(r7,f). mem(r7,a).
att(r8,f). mem(r8,d).
```

4.3 Algorithms

4.3.1 Computing Rankings

The ranking-based semantics and functions outlined in the previous sections are implemented by following the computation of strength values of arguments for each step as mentioned in the definitions above. The termination criteria for these functions differs from function to function as required. For the categoriser, mean-categoriser, average and min burden-based functions, strengths are generated in incremental steps until said values converge to a value by a degree of a set interval. For our implementation, this value is set to 0.01. As for the discussion-based semantic, path counts for attacks and defences are generated in incremental steps until unique step counts are generated for each argument.

The following algorithm details the method followed to compute strength values for arguments when working with the nh-categoriser function.
Algorithm 1: nh-categoriser Strength Computation

Result: Strength values for arguments

initialize strengths = 1;
MAX_STEPS = 50;
PRECISION_VAL = 0.01;

while $i < MAX_STEPS$ or isPrecisionAchieved do
    isPrecisionAchieved = true;
    while argument $x$ in framework do
        $\text{Cat}(x) = 1 / (1 + \min(\text{Cat}(a \in R_1^-(x))))$;
        isPrecisionAchieved = newStrengthValue - prevStrengthValue <= PRECISION_VAL;
    end
    $i++$
end

A similar method is followed when implementing other ranking functions with
the difference being a different aggregation function being employed to generate
strengths for arguments at each step.

4.3.2 Computing Preferred Extensions

The following section addresses the method used in order to generate preferred
extensions. These are generated by following the process as specified in [32]. This
process consists of the computation of candidates by constructing a tree where-in
nodes are built using the recursive addition and removal of arguments to the candidate.
A candidate can be defined as per the following

Definition 20 (Candidate) The partition of $A$ into three sets $(I, O, U)$, where $U$ denotes
the set of arguments still not assigned a labelling, $I$ the arguments that are in a preferred
extension and $O$ the arguments that are out.

The division of $A$ into candidates as above leads to the construction of a tree that
is used to compute all possible permutations of candidates for preferred extensions.
Next, we define the Sets of Arguments that are attacked by a given set of arguments
as well as the Sets of Arguments that were if added to a given set of arguments, the
set would attack itself

Definition 21 (Arguments Attacked) For a given set $S \subseteq A$

$$S^- = \{ a \in A : \exists T \subseteq S \text{ s.t. } T \triangleright a \}$$

The set above, $S^-$ can be thought of as the set of arguments that are attacked by $S$.

Definition 22 (Self Attack Arguments) For a given set $S \subseteq A$

$$S^\leftarrow = \{ a \in A : \exists T \subseteq S, b \in S \text{ s.t. } T \cup \{ a \} \triangleright b \}$$

The set above, $S^\leftarrow$ can be thought of as the of arguments that if added to $S$ would
make it attack itself.

Given the above, child nodes for each candidate node are computed by the addition
and removal of a given argument.
4.3. Algorithms

**Definition 23 (Child Nodes/Candidates)** Given a candidate \( \mathcal{C}A \) and an argument \( a \in \mathcal{A} \),

\[
\mathcal{C}A - a = (I_{\mathcal{C}A}, O_{\mathcal{C}A} \cup \{a\}, U_{\mathcal{C}A} \setminus \{a\})
\]

and

\[
\mathcal{C}A + a = (I_{\mathcal{C}A} \cup \{a\}, O_{\mathcal{C}A} \cup \Delta^+_A \cup \Delta^-_A, U_{\mathcal{C}A} \setminus (\{a\} \cup \Delta^+_A \cup \Delta^-_A))
\]

where

- \( I_{\mathcal{C}A} \) is the set of IN arguments in candidate \( \mathcal{C}A \)
- \( O_{\mathcal{C}A} \) is the set of OUT arguments in candidate \( \mathcal{C}A \)
- \( U_{\mathcal{C}A} \) is the set of UNDEC arguments in candidate \( \mathcal{C}A \)
- \( \Delta^+_A = \{b \in U_{\mathcal{C}A} : \exists T \subseteq I_{\mathcal{C}A} \text{ s.t. } T \cup \{a\} \not\succ b\} \)
- \( \Delta^-_A = \{b \in U_{\mathcal{C}A} : \exists T \subseteq I_{\mathcal{C}A}, c \in I_{\mathcal{C}A} \text{ s.t. } T \cup \{a\} \not\prec T \cup \{a, b\} \not\prec c \not\prec T \cup \{a, b\} \not\prec a\} \)

Once all possible candidates have been generated, each are checked for conflict-freeness and admissibility. Filtered candidates are then checked if they satisfy the preferred extension semantic as defined in Definition 5.

Figure 4.2 shows the construction of a candidate tree according to the process outlined above. The following example shows the candidates generated when constructing the tree for arguments in Example 1.

**Example 10 (Candidate Generation cont’d Example 1)** Table 4.1 shows the arguments belonging to the IN, OUT and UNDEC sets when arguments are added or removed to a candidate. Assume all arguments belong to UNDEC set in Candidate \((\mathcal{C}A)\) initially.

<table>
<thead>
<tr>
<th>Candidate</th>
<th>IN</th>
<th>OUT</th>
<th>UNDEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Ca)</td>
<td>(a,b,c,d,e,f)</td>
<td>(a)</td>
<td>(b,c,d,e)</td>
</tr>
<tr>
<td>(Ca + a)</td>
<td>(a)</td>
<td>(f)</td>
<td>(b,c,d,e)</td>
</tr>
<tr>
<td>(Ca - a)</td>
<td>(a)</td>
<td></td>
<td>(b,c,d,e,f)</td>
</tr>
<tr>
<td>(Ca + a + b)</td>
<td>(a,b)</td>
<td>(c,d,f)</td>
<td>(e)</td>
</tr>
<tr>
<td>(Ca + a - b)</td>
<td>(a)</td>
<td>(b,f)</td>
<td>(c,d,e)</td>
</tr>
<tr>
<td>(Ca - a + b)</td>
<td>(b)</td>
<td>(a,d)</td>
<td>(c,e,f)</td>
</tr>
<tr>
<td>(Ca - a - b)</td>
<td>(a,b)</td>
<td></td>
<td>(c,d,e,f)</td>
</tr>
</tbody>
</table>

**Table 4.1:** Candidate Generation for Preferred Extensions

Candidate trees are constructed recursively as shown above. In the case of Example 1, extensions are found on the candidates as specified in Table 4.2.

<table>
<thead>
<tr>
<th>Candidate</th>
<th>IN</th>
<th>OUT</th>
<th>UNDEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Ca + a + b + e)</td>
<td>(a,b,e)</td>
<td>(c,d,f)</td>
<td></td>
</tr>
<tr>
<td>(Ca + a - b + c + d)</td>
<td>(a,c,d)</td>
<td>(b,e,f)</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4.2:** Preferred Extension Candidates

When generating candidate trees as above for a high number of arguments, generating candidates for every possible combination of arguments addition and removal
can lead to processing times that are exponential. [32] defines a method to perform pruning on the candidate tree being constructed. The following outlines the conditions that can be checked against to help determine whether further traversal along a path in the tree is desired.

**Notation 3 (Preferred Extensions of Candidate)** \( \text{pref}(\mathcal{C}, \mathcal{A}) \) are the Preferred Extensions of candidate \( \mathcal{C}, \mathcal{A} \)

**Condition 1** For a candidate \( \mathcal{C}, \mathcal{A} \), if \( I_{\mathcal{C}, \mathcal{A}} \cup U_{\mathcal{C}, \mathcal{A}} \subset S \) for some admissible set \( S \), then \( \text{pref}(\mathcal{C}, \mathcal{A}) = \emptyset \)

**Condition 2** For a candidate \( \mathcal{C}, \mathcal{A} \) and \( a \in U_{\mathcal{C}, \mathcal{A}} \), if

- for all sets \( S \), where \( S \triangleright a \), it holds that \( S \cap I_{\mathcal{C}, \mathcal{A}} \neq \emptyset \) and
- \( a \not\in (I_{\mathcal{C}, \mathcal{A}} \cup U_{\mathcal{C}, \mathcal{A}})^{\triangleright} \) and
- \( a \not\in (I_{\mathcal{C}, \mathcal{A}} \cup U_{\mathcal{C}, \mathcal{A}} \setminus a)^{\leftarrow} \)

then \( \text{pref}(\mathcal{C}, \mathcal{A} - a) = \emptyset \)

**Condition 3** For a candidate \( \mathcal{C}, \mathcal{A} \) and \( a \in U_{\mathcal{C}, \mathcal{A}} \) which is attacked at least one set of arguments, if

\[ \forall T, \mathcal{R} \text{ where } T \triangleright \mathcal{R} \text{ and } \mathcal{R} \triangleright a \text{, it holds that } T \cap O_{\mathcal{C}, \mathcal{A}} \neq \emptyset, \text{ then } \text{pref}(\mathcal{C}, \mathcal{A}) = \emptyset \]

Using the above conditions, the candidate tree can be pruned, avoiding searching for extensions along certain paths in the tree.
4.4 User Interface

The interface for such a web-app would comprise of inputs that would allow a user to choose and submit arguments and attacks, a ranking semantic and an acceptance semantic. Arguments and attacks would be accepted using a file upload input while the ranking and acceptance semantic could be chosen using drop down selects. The inputs above would be part of a form that on submit would result in the rankings, extensions and visualize the input arguments and attacks being displayed.

Figure A.1 shows the landing page for the application. Figure A.2 shows the demo page when first visited by a user. Upon submitting a valid request, the response is populated on the page as shown in Figure A.3.

Arguments and attacks would be visualized on a canvas the dimensions of which could be modified using inputs made available to the user. This would allow the user to better visualize the argument framework when the number of arguments increase. Finally, the ranking and acceptance results would be shown in the form of tables. Each row would show the relevant details for each argument. Four such tables would be populated -

- **Rankings**
  Displays arguments, as well as their strength values, sorted using generated strength values. When working with burden-based functions, the application displays a sequence of strength values that indicate the strength values at each step. For the discussion-based semantics, the application displays the path sizes for each step in the form of a sequence of values. Finally, for the nh-categoriser and mean categoriser, the application displays the final strength values that are obtained once the set precision for values has been obtained.

- **Arguments**
  Displays the parsed arguments ranked according to the strength values generated. Clicking on a row highlights the selected argument by highlighting the specific Node associated with the argument as well as it’s incoming attacks and attack supports. Incoming attack edges are highlighted in red while support to attacks are highlighted in green.

- **Attacks**
  Displays the attacks parsed in the form of Attack Label, Attacked Argument and Attacking Members. Clicking on a row highlights the selected attack relation by highlighting the node specific to the attack label as well as the attacked argument and attack supporting arguments. Attacked nodes and attack support nodes are highlighted in blue and green colours respectively. Support and attack edges are highlighted green and red respectively.

- **Extensions**
  Displays the Extensions generated for the input Argument Framework. Clicking on a row highlights the IN and OUT arguments as dictated by the chosen extension. IN arguments are highlighted green while OUT arguments are highlighted red.

Example 11 (Generated Tables Cont’d Example 1) Referring back to Example 1, the Rankings, Arguments, Attacks and Extensions would be populated in the following manner.
Chapter 4. Implementation

TABLE 4.3: Rankings and Arguments Tables

<table>
<thead>
<tr>
<th>Argument</th>
<th>Strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>0.6902845</td>
</tr>
<tr>
<td>c</td>
<td>0.5916164</td>
</tr>
<tr>
<td>e</td>
<td>0.5384851</td>
</tr>
<tr>
<td>d</td>
<td>0.44867808</td>
</tr>
<tr>
<td>f</td>
<td>0.40838364</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
</tr>
<tr>
<td>e</td>
<td>4</td>
</tr>
<tr>
<td>d</td>
<td>5</td>
</tr>
<tr>
<td>f</td>
<td>6</td>
</tr>
</tbody>
</table>

TABLE 4.4: Attacks and Extensions Tables

<table>
<thead>
<tr>
<th>Attack Label</th>
<th>Members</th>
<th>Attacked</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>a,c,d</td>
<td>b</td>
</tr>
<tr>
<td>r2</td>
<td>a,b</td>
<td>c</td>
</tr>
<tr>
<td>r3</td>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>r4</td>
<td>c,e</td>
<td>d</td>
</tr>
<tr>
<td>r5</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>r6</td>
<td>b,f</td>
<td>e</td>
</tr>
<tr>
<td>r7</td>
<td>a</td>
<td>f</td>
</tr>
<tr>
<td>r8</td>
<td>d</td>
<td>f</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IN</th>
<th>OUT</th>
<th>UNDEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>a,b,e</td>
<td>c,d,f</td>
<td>a,c,d</td>
</tr>
<tr>
<td>a,c,d</td>
<td>b,e,f</td>
<td></td>
</tr>
</tbody>
</table>

In case when an empty file is uploaded to the server, an error is returned to the user in the form of an alert showing the error. When the uploaded file has an error in the formatting of the file, an error alert is shown to the user intimating them that the input file is not formatted properly.

Finally, the entire source code for this application can be found at the GitHub repository available at https://github.com/hasinha/cs5917.

4.5 Feature Showcase & Testing

The following section outlines the features of the application developed as well as the testing and performance measurement conducted.

4.5.1 Feature: File Upload & Argument & Attack Parsing

A user can upload a file in the ASPARTIX for SETAF format to the server. Figure A.4 shows the UI element allowing the user to upload a file.

Once a file upload request is submitted, the server returns the parsed arguments and attacks which are populated in the form of tables as shown in Figure A.5.

4.5.2 Feature: Rank Computation

The application allows the user to generate ranks for a given argument framework in the form of a drop down. The options available to the user comprise of

- Nh-Categoriser
- Discussion-Based
- Mean Burden-Based
• Min Burden-Based
• Mean Categoriser

Figure A.7 shows the UI element allowing the user to choose a ranking function.

Upon submitting the request, the argument ranks and their respective strengths are populated in a table as show in Figure A.8.

### 4.5.3 Feature: Extension Computation

Currently the user is only able to generate Preferred Extensions for a submitted argument framework. Generated extensions are populated in the form of a table as show in Figure A.6.

### 4.5.4 Feature: Visualisation

Parsed arguments and attacks are visualised as shown in Figure 2.1. In addition to this visualisation, the application allows the user to highlight a specific node and its incoming and outgoing attacks, attack relations and generated extensions. As mentioned in Section 4.4, clicking on one of the entities mentioned above results in the relevant nodes and edges being highlighted. This is further illustrated in Figures A.9, A.10 and A.11.

### 4.5.5 Performance Measurements

The performance of the application is tested by submitting files comprising argument frameworks of varying number of arguments and attacks. For the purposes of testing, these argument frameworks are generated by randomly generating arguments as well as their attacks. As part of this test, argument attacks are assigned randomly. Any given argument may be attacked by a range of 0 - 4 other arguments. These attacks again would comprise of members chosen at random. The attack members could comprise again of upto 4 arguments.

The following table shows the times taken to generate rankings and extensions for given number of arguments:

<table>
<thead>
<tr>
<th>Argument Count</th>
<th>Iteration 1(ms)</th>
<th>Iteration 2(ms)</th>
<th>Iteration 3(ms)</th>
<th>Average(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>42</td>
<td>32</td>
<td>7</td>
<td>27</td>
</tr>
<tr>
<td>10</td>
<td>138</td>
<td>38</td>
<td>10</td>
<td>62</td>
</tr>
<tr>
<td>15</td>
<td>149</td>
<td>35</td>
<td>21</td>
<td>68</td>
</tr>
<tr>
<td>20</td>
<td>185</td>
<td>42</td>
<td>35</td>
<td>87</td>
</tr>
<tr>
<td>25</td>
<td>292</td>
<td>154</td>
<td>115</td>
<td>187</td>
</tr>
<tr>
<td>30</td>
<td>342</td>
<td>212</td>
<td>207</td>
<td>253</td>
</tr>
<tr>
<td>35</td>
<td>1296</td>
<td>864</td>
<td>877</td>
<td>1012</td>
</tr>
<tr>
<td>40</td>
<td>34134</td>
<td>28263</td>
<td>31465</td>
<td>31287</td>
</tr>
<tr>
<td>45</td>
<td>59271</td>
<td>31502</td>
<td>36249</td>
<td>42340</td>
</tr>
<tr>
<td>50</td>
<td>89136</td>
<td>72476</td>
<td>77963</td>
<td>79768</td>
</tr>
</tbody>
</table>

**Table 4.5: Performance Measurements**

It should be noted that computing the rankings of arguments takes anywhere between 1 and around 40 milliseconds. Most of the time taken shown above comprises...
the time taken to compute the Preferred Extensions. This behaviour is explored further in Section 4.6.1.

Figure 4.3 shows the line graph plotted for the performance measurements done for the application.

UI performance is measured in terms of FPS when visualisation of parsed arguments and attacks is being performed. The FPS count consistently remains at 55 FPS for the input argument frameworks above. It should be noted that testing has not been performed against a greater number of arguments. No lag is present when entity highlighting on canvas is performed either.

4.5.6 Testing

The developed application is tested primarily using Manual Testing. An Argument Framework input file is created for which the Rankings and Extensions are known. Such a file is submitted to the application and the generated outputs are compared against the expected outputs. For the purposes of this test, an input file using \( AF_1 \) as defined in Example 1 is constructed. The output rankings and extensions for each of the ranking functions and acceptability semantic are generated and verified against those specified in Sections 3.1 and 4.3.2.

Though Unit Tests could have been designed to test in detail the computation of ranks and extensions, the same could not be achieved owing to time constraints.
4.6 Evaluation

4.6.1 Scalability

The following section addresses the scalability and, in turn, the possible performance improvements of the application developed. As noted in the section measuring the performance of the application, generation of ranks is not time and memory intensive. The time and memory complexity when generating extensions on the other hand increases exponentially with the increase in the number of arguments and attacks between them. This is evident because as mentioned previously, when generating extensions, all possible combinations of argument removal and addition have to be generated for extension candidates. As the number of arguments increases, the number of possible candidates for extensions increases dramatically. The presence of termination criteria for argument addition and removal are implemented do not aid much in this case.

In order to further reduce the time taken to generate these candidates, this processing is done in parallel using an ExecutorService. The ExecutorService allows one to spawn several threads which in turn allows the server to process candidate generation in parallel. The application currently is able to spawn a maximum of 40 threads when generating extensions. As was seen in the performance measurements, such a thread count is not enough to allow the server to generate extensions in an acceptable amount of time when the number of arguments goes beyond 35. One could reduce this time further by increasing the thread count available to the ExecutorService but this would come at the cost of higher memory consumption. In essence, for the methodology implemented here, the time complexity can be reduced by increasing the available thread count provided enough Memory and CPU cores are available.

The application is hosted on a Heroku PaaS free tier server. Such a server provides the application with around 512 MB of memory and 1 Web worker. As a result of the low thread count, the demo application will sometimes not respond if it is busy processing the result of another request. Similarly, due to the low memory available, the server might return an error status of 503-Service Unavailable if it encounters an OutOfMemoryException when processing a request with high number of arguments. Increasing the number of available workers or dynos on Heroku would alleviate such problems.

4.6.2 Feedback

User evaluation for the implemented application was conducted by means of a feedback form. Experts and non-experts were asked to try out the application following which their responses for the feedback were collected by means of a Google Feedback form. The feedback form can be found at https://forms.gle/dYWHHoXiVPVq3Hi67. As part of this evaluation, we received 5 responses to the feedback form.

When asked about the intuitiveness of the application, users said the application was generally intuitive and that the visualisation feature of the application was particularly helpful in understanding the argumentation framework. All of the users responded with positive feedback with regards to the visualisation feature. With regards to the helpfulness of the ranking functions, users generally responded with positive feedback. One of the major drawbacks, however, of the application, according to users, was that the ranking functions were not explained well. Currently,
the application only includes definitions of the implemented functions. A text to go along with said definitions would significantly increase the explainability of said functions.

In order to better explain the outcomes of the ranking functions, the results of these functions could have been explained better for the relevant function chosen. As far as visualisation is concerned, a legend and consistent highlighting colours would further aid in better understanding the parsed arguments and their attack relations. Post highlighting an argument or attack relation, a way to reset the visualisation would have further helped. Lastly, a way to store histories of computed ranks using several functions would help users compare the outcomes generated for each function.

4.7 Summary

<table>
<thead>
<tr>
<th>Chapter 4 in a Nutshell</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Technologies and architecture employed to build application for demonstrating the functions proposed</td>
</tr>
<tr>
<td>• ASPARTIX input format definition</td>
</tr>
<tr>
<td>• Ranking function algorithm definition</td>
</tr>
<tr>
<td>• Preferred extension computation algorithm definition</td>
</tr>
<tr>
<td>• Application user interface characteristics and feature showcase</td>
</tr>
<tr>
<td>• Manual tests and performance measurements</td>
</tr>
<tr>
<td>• Possible scalability and improvements of application developed</td>
</tr>
<tr>
<td>• Evaluation conducted on implemented application by experts and non-experts and feedback collected</td>
</tr>
</tbody>
</table>
Chapter 5

Discussion and Conclusion

This chapter will delve into the related research that has been conducted on the research topic chosen as well as the results and conclusions that can be drawn from the research outlined in the previous chapters. Finally, we will look at the future work that can be done relevant to the research topic.

5.1 Related Research

1. In the case where multiple extensions are available for a given argumentation framework, it is important to enable the user to choose the best possible extension given the argument rankings.
   
   - In the work of [37], the authors create an argumentation framework from an inconsistent knowledge base and reuse the “inconsistency knowledge” from the data to rank extensions.
   - In the work of [35], the authors introduce a framework for aggregating the output of ranking-based semantics to rank extensions. The analysis of any related properties is very superficial. Later on, the work of [34] studied how ranking-based semantics can be used to refine the output of extensions/labeling semantics and vice-versa.
   - In the work of [17], they authors studied how ranking extensions can be applied to the choice of packaging for the agro-business.

2. For a given Argumentation Graph, consisting of multiple conclusions arrived at via different paths, output rankings to enable one to choose a path that has better acceptability or believability.
   
   - In the work of [38], the authors propose a very simple structured argumentation framework and the extensions to rank arguments’ conclusions.

3. Extending ranking-based semantics defined for binary AF in the context of SETAF.
   
   - In the work of [13], they first introduce the Categoriser ranking-based semantics in the case of binary AF which was later re-defined for the SETAF case in [PAPER BRUNO AAAI 2020]
   - In the paper of [39], they introduce an extension of Dung’s abstract argumentation frameworks that include social voting on the arguments: the Social Abstract Argumentation Frameworks (SAF). They also propose a family of semantics where a model is a solution to the equation system with one equation for each argument, based on its social support and its
To compare SAFs with the existing ranking-based semantics, we chose to ignore the social support of arguments by giving them the same value.

- In the paper of [6], they introduce both the Discussion-based ranking-based semantics and the Burden-based ranking-based semantics.
- The semantics proposed by [40] takes into account all the ancestors branches of an argument (defender and attacker) stored in tupled values.
- In [41], the authors compute the strength of an argument using a two-person zero-sum strategic game.

### 5.2 Results and Conclusion

In this section we shall look at the results and conclusions we can draw from the research done as part of this thesis on the research topic. In the previous chapters we introduced the concepts that we worked with and then proceeded to make modifications to said concepts to allow them to work with SETAFs.

#### 5.2.1 New ranking-based semantics

We saw the existing ranking-based semantics that have defined for abstract argumentation frameworks. The ones explored here comprised of the discussion-semantics and burden-based semantics. Further, we looked at the nh-categoriser function and the categoriser, a ranking-based semantics for SETAFs. We extended the existing AAF ranking-based semantics by adding aggregation functions for computing the strengths of arguments based on the strengths of their attacking arguments.

The mean and minimum aggregation functions were applied to the discussion-based and burden-based semantics. The average burden-based function assigns strengths to arguments based on the average of the strengths of their attacking arguments. The minimum burden-based function assigns the strengths to arguments based on the minimum of strengths of their attacking arguments.

The discussion-based semantics did not require much modification except for the definition of a path. In the case of AAFs, a path would be a sequence of arguments. In the case of SETAFs, a path would comprise a sequence of sets of arguments since arguments can be attacked by sets of arguments. Once a path has been defined in the context of SETAFs, the existing discussion-based semantics works with SETAFs.

Finally, a modification of the nh-categoriser, the mean categoriser is proposed that assigns strengths based on average instead of minimum strengths of attacking arguments.

Next, we looked at the properties of ranking-based semantics that were satisfied by the modified ranking functions. The satisfaction of these properties portrays an idea about the ranking functions that allows one to best choose an appropriate function for the problem at hand. Said properties allow one to choose a ranking function according to the idea one desires the function should address. For instance, if one desires a function which ranks arguments with more attackers lower than those with fewer attackers, one would look for a function that satisfies the Cardinality Precedence property. Off the functions proposed, one could choose to work with the discussion-based function since said property is satisfied by it.
5.2.2 Preferred Extension Computation

We employed the algorithm put forward by [32] to generate preferred extensions in the context of SETAFs. This is done by recursively generating candidates by adding and removing candidates from “in” labelings. Termination criteria were also implemented that allowed the system to check against conditions that would indicate whether further generation of candidates was desirable. Performance measurements of the algorithm employed was also done. The huge time complexity when having to generate extensions for large number of arguments too was explored in the previous chapter as well as possible improvements that could be made to reduce the time complexity.

5.2.3 Framework for Ranking SETAFs Extensions

Chapter 3 put forward a new framework that allowed one to rank extensions using the generated ranking on arguments. Rankings on arguments are generated using the existing nh-categoriser as well as the modified proposed new functions based on the discussion-based and burden-based semantics for abstract argumentation frameworks. Extensions are generated using the implementation based on the algorithm specified previously.

Once rankings on arguments and extensions are generated, a ranking on extensions is generated by lexicographically comparing ordered ranks for arguments that are labeled “in” in generated extensions.

5.2.4 Application

As part of the research done, we developed a tool the goal of which is to demonstrate the functions that have been proposed as part of this thesis. One of the main features the application allows the user to do is to generate a ranking on arguments using any of the ranking functions previously specified. In addition to this, the application allows the user to generate extensions. For now, only the preferred extension semantic has been implemented. A ranking on extensions is also generated using the rankings and extensions generated.

Another primary feature of the application is the visualisation of an argument framework. The application is able to visualise arguments in the form of circle shapes and attack relations in the form of square shapes. Arguments and attack relations are displayed to the user in the form of tables alongside the canvas visualising the argument framework. The application further allows users to highlight arguments, attacks and extensions by clicking on rows in displayed tables. Relevant nodes are highlighted by coloring nodes and its incoming and outgoing attacks and graying out the remaining nodes and edges.

In the previous chapter we looked at the evaluation conducted by experts and non-experts by means of a feedback form. We explored the further improvements and changes that could be made to the application to better explain the functions implemented as well as the outcomes generated.

5.3 Future Work

In this section we shall look at the work that can be done in the future on the concepts addressed as part of this thesis.
5.3.1 Theoretical Work

The following section addresses the possible theoretical work that can be done to add on to and improve the work done as part of this thesis.

The new extension ranking function framework as of yet has not been tested against properties. To better characterise such a framework, it would help to define properties as well as whether the framework satisfies said properties. This work has not been done currently and could be achieved in the future.

The research topic chosen saw us work primarily with ranking-based semantics for SETAFs and extensions. Having temporal argumentation frameworks with the ranking functions proposed above for SETAFs would further help build the feature set of the application and would help one model complex real world scenarios wherein arguments can be valid depending on the time at which the model is being evaluated.

In addition to this, one could also look at integrating bipolar weighted argumentation frameworks to work with the application developed. Such an implementation would allow one to introduce assigning initial weights to arguments as well as attack and support relations which in turn would allow the application to model even more complex real world scenarios.

Another point of contention is the time complexity when computing preferred extensions for argument frameworks consisting of a large number of arguments. An alternative implementation wherein preferred extensions can be generated without dramatically increasing the time complexity of the computation would significantly help improve the performance of the implemented tool.

5.3.2 Changes w.r.t. Tool

In the following section we shall address the changes that can be made to the application implemented to further enhance its usability and user experience.

The application developed as of now only allows users to generate rankings on arguments and extensions. A further improvement could be made to the application to allow users to export said results as a text file.

As far as visualisation is concerned, the application could be improved by adding a better way of highlighting nodes and edges. Further, a legend could be added that would aid users in better understanding the nodes and edges plotted on the canvas.

The application could be scaled vertically by hosting the application using a service that would be able to provide the application with a greater number of threads and a higher amount of memory. These changes would aid the application when computing extensions which as we saw in the previous chapter has a high cost in terms of time complexity.
Appendix A

Figures

A.1 List of Figures

Figure A.1: Landing Page
Figure A.2: Demo Page

Figure A.3: Response Populated

Figure A.4: File Upload Input
A.1. List of Figures

Figure A.5: Parsed Arguments & Attacks

Figure A.6: Extensions Table

Figure A.7: Ranking Function Drop Down
Appendix A. Figures

**Figure A.8:** Ranking Table

**Figure A.9:** Argument Highlighting
A.1. List of Figures

**Figure A.10:** Attack Highlighting

**Figure A.11:** Extension Highlighting
Appendix B

Installation

The source code is included herewith as a zip file or can be found at the following url https://github.com/hasinha/cs5917. There are several ways to run the application, two of which are explored below.

B.1 Prerequisites

Please ensure the following tools are installed

- Maven
- Tomcat
- >= JDK 1.8 (if working with eclipse)
- >= JRE 1.8

B.2 Eclipse

The simplest way to run this application is through an IDE. Here we shall provide instructions for the Eclipse IDE.

The following details the steps that should be followed to setup and run the application.

1. Start Eclipse
2. Import Existing Project
3. Create new Tomcat Server
4. Add project to new tomcat instance
5. Navigate to events.js(line: 141) and modify the socket connection url to /CS5197/chat
6. Start Server
7. Navigate to http://localhost:8080/CS5197/ to access application

B.3 WAR file local deploy

Another way to run the application is to build a WAR file and run the Tomcat instance of it. To generate a WAR file, run the following in the root directory of the codebase-
Appendix B. Installation

mvn install

Running this will output a target folder in the same directory which will contain a WAR file named “CS5197-0.0.1-SNAPSHOT.war”.

1. Copy/Move WAR file to $CATALINA_HOME/webapps location.
2. Navigate to Tomcat installation directory “bin” containing startup scripts.
3. Start Tomcat instance by executing startup.sh
5. Tomcat instance can be shutdown by executing shutdown.sh script.

Alternatively, one can use the tomcat manager to deploy this application. For this to work, please ensure the tomcat user in $TOMCAT_HOME/conf/tomcat_users.xml has the manager-gui role defined and available. The following steps should be followed-

1. Start Tomcat instance by executing startup.sh
2. navigate to http://localhost:8080/manager
3. Navigate to the section “WAR file to deploy” under the Deploy section.
4. Select the WAR file previously generated to deploy.
5. Deploy & navigate to http://localhost:8080/warFileName
Bibliography


