UNIVERSITY OF ABERDEEN

MASTER THESIS

A Temporal Argumentation Framework with Sets of Attacking Arguments

Author: Jinlong ZHU
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, Jinlong ZHU, declare that this thesis titled, “A Temporal Argumentation Framework with 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: ______________________________

Date: ________________________________
Abstract

School of Natural & Computing Sciences

Master of Science

A Temporal Argumentation Framework with Sets of Attacking Arguments

by Jinlong ZHU

Argumentation is a field of Artificial Intelligence that specialises in knowledge representation and reasoning. Argumentation mimics the mechanism that humans use to reason by the means of Logic. Nowadays, argumentation is used in various real-life disciplines and implemented in various tools. In this work, we define a new argumentation framework, called “Temporal Argumentation Framework with Sets of Attacking arguments”, by adding the notion of time to the Argumentation Framework with Sets of Attacking Arguments (SETAF). This new argumentation framework, in which the set of arguments and the attack relation between the arguments are only valid for a certain period, is more expressive as it is able to model more complex scenarios. As a result, the relation of attacks and defences between arguments are not permanent. We also extended the usual argumentation semantics extensions for this new framework and implemented them in a concrete application.
Acknowledgements

From the topic selection of the thesis, the acquisition of materials to the writing of the thesis, I received the help of my teachers and classmates. First of all, I would like to thank my supervisor, Dr. Bruno Yun, who provided valuable suggestions for my research and study, taught me many professional skills, and patiently guided me to complete the project. I admire his profound knowledge and responsible attitude very much. Secondly, I want to thank all the teachers in our college who taught me a lot of professional knowledge. Finally, I want to thank my classmates, friends and parents for supporting me and for everything they have done for me.
Contents

Declaration of Authorship iii

Abstract v

Acknowledgements vii

1 Introduction 1
  1.1 Argumentation with Sets of Attacking Arguments 1
  1.2 Argumentation and Time 2
  1.3 Research Question 2
  1.4 Outline of the thesis 2

2 Background 3
  2.1 Dung’s Abstract Argumentation Framework 3
  2.2 Timed Argumentation Framework 4
  2.3 Argumentation Frameworks with Sets of Attacking Arguments 5

3 Temporal Frameworks with Sets of Attacking Arguments 7
  3.1 Temporal SETAF 8
  3.2 Semantics for T-SETAF 10
  3.3 A New Input Format for T-SETAF 11
  3.4 Extracting SETAF from T-SETAF 11

4 Implementation 15
  4.1 Technologies and Architecture 15
  4.2 Development Tools and Environment 16
  4.3 Algorithms 17
    4.3.1 Finding All Subsets of Set 17
    4.3.2 Computing Conflict-free Extensions 17
    4.3.3 Computing Admissible Extensions 18
    4.3.4 Computing Complete Extensions 19
    4.3.5 Computing Preferred Extensions 20
    4.3.6 Computing Grounded Extension 20
  4.4 User Interface 21
  4.5 Testing 23

5 Conclusion 25

Bibliography 27
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>AFF for Example</td>
<td>3</td>
</tr>
<tr>
<td>2.2</td>
<td>TAF for Example</td>
<td>4</td>
</tr>
<tr>
<td>2.3</td>
<td>SETAF for Example</td>
<td>5</td>
</tr>
<tr>
<td>3.1</td>
<td>T-SETAF for Example</td>
<td>8</td>
</tr>
<tr>
<td>3.2</td>
<td>Temporal Distribution</td>
<td>9</td>
</tr>
<tr>
<td>3.3</td>
<td>T-SETAF INPUT FORMAT</td>
<td>12</td>
</tr>
<tr>
<td>3.4</td>
<td>Representation of T-SETAF</td>
<td>12</td>
</tr>
<tr>
<td>3.5</td>
<td>The SETAF Extracted from T-SETAF</td>
<td>13</td>
</tr>
<tr>
<td>4.1</td>
<td>Architecture</td>
<td>16</td>
</tr>
<tr>
<td>4.2</td>
<td>Enter the user interface</td>
<td>21</td>
</tr>
<tr>
<td>4.3</td>
<td>Input the path of the xxx.apx file</td>
<td>22</td>
</tr>
<tr>
<td>4.4</td>
<td>Generate tsetaf</td>
<td>22</td>
</tr>
<tr>
<td>4.5</td>
<td>Show tsetaf graph</td>
<td>23</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In the real world, humans use argumentation in their daily lives and argumentation theory has been studied since the time of Aristotle. In logic and philosophy, argumentation is a mechanism that mimic human reasoning by drawing conclusions through logical reasoning. Although argumentation theory has been an individual discipline since 1970s, the use of argumentation in Artificial Intelligence has truly blossomed only in the late 1980s. Argumentation is now used in many applications from disciplines ranging from linguistics, philosophy, discourse analysis, social psychology and many other [17, 12, 13, 5].

In his seminal paper [10], Dung proposed to represent arguments and attacks as nodes and arcs in directed graphs, called Abstract Argumentation Framework (AAF). Then, he uses a game-theory approach to draw conclusion by computing groups of non-conflicting arguments called “extensions” [4].

This approach of representing arguments as “abstract” has been widely adopted and extended in the recent work [15, 1, 2, 3]. In [3], the authors extend the AAF into a bipolar argumentation framework (BAF) by adding support relation enabling arguments to support each others.

In [15, 1, 2], the authors extend BAFs by adding weights on arguments to represent the reliability of an argument. Each successful extension of the original AAF enabled to capture new intuitions that were previously not accounted for. In this thesis, we are interested in a particular extension of the AAF called Argumentation Framework with Sets of Attacking Arguments (SETAF).

1.1 Argumentation with Sets of Attacking Arguments

Nielsen and Parsons’ introduced the Argumentation Framework with Sets of Attacking Arguments (also called SETAF) [16], which is an AFF with the added concept of “joint attack”, i.e. groups of arguments can attack a single argument.

They added value of these joint attacks is that they enable us to represent particular phenomena such as the “accrual” phenomena (multiple arguments attacking together are stronger than individual arguments attacking alone).

Moreover, SETAF have been increasingly studied in the recent years [11, 18]. In [11], Flouris and Bikakis completed the formal characterization of SETAF by redefining the usual extension and labelling-based semantics and studying the links between them. In [18], the authors studied how ranking-based semantics (originally defined for AAF) can be re-defined for SETAF.
1.2 Argumentation and Time

The topic of time is also important in argumentation as arguments can be introduced at different times, and multiple arguments can only be applied at specific points in time. It is also possible that arguments may have different strengths depending on when and how they are stated. More importantly, as time goes by, some arguments may become invalid or unusable.

For all these reasons, we need to consider the timely development of information by clearly indicating the moment when these arguments can be used.

In [6], the authors model the notion of time on the classic bipolar abstract argumentation frameworks and propose different semantics of temporal acceptability. In [7], the authors propose an abstract argument framework in which the relationship between arguments is only valid within a certain time interval.

All existing work in argumentation dealing with time has been mainly focused on binary argumentation frameworks [7, 8, 6] and how SETAF interact with time has not been studied.

1.3 Research Question

Our project’s research question is:

“How can we extend SETAF with the notion of time and what is the impact of time on the output on the existing semantics?”

To answer this question, we added the notion of time for SETAF by adding an availability interval for each argument, so that the attack relation defined only for a limited amount of time. In terms of implementation, we first defined the new format for inputting T-SETAF based on the existing. secondly, we allow the user to interact with the argumentation graph by providing a graphical representation. Finally, we implemented the new semantics in the tool.

1.4 Outline of the thesis

This report is composed of five chapters. In Chapter 1, we introduce the general research context and our research question. In Chapter 2, we give definitions and examples of the three argumentation frameworks: Abstract Argumentation Framework (AAF), Timed Argumentation Framework (TAF) and the Argumentation Framework with Sets of Attacking Arguments (SETAF). In Chapter 3, we introduce T-SETAF and show how the semantics can be extended to this new framework. We introduce the input format of T-SETAF, and the conversion between T-SETAF and SETAF. Chapter 4 describe the architecture of the tool, the algorithm, the user interface and the evaluation. In Chapter 5, we summarize the project, provide some conclusions to the research questions, the deficiencies of the project and the future work.
Chapter 2

Background

2.1 Dung’s Abstract Argumentation Framework

In his seminal paper [10], Dung introduces the Abstract Argumentation Framework (AAF) as a directed graph where nodes are abstract arguments and arcs represent attacks between arguments.

Definition 1 (AAF) An AAF is $S = (A, E)$ where $A$ is a set of arguments and $E \subseteq A \times A$ is a binary set of attacks.

Example 1 Let $S$ be an AAF such that $A = \{A, B, C, D, E, F\}$ and $E = \{(A, B), (A, E), (B, C), (D, C), (E, F)\}$ represented in Figure 2.1.

![Figure 2.1: AFF for Example](image)

From an AAF, there are multiple ways to extract useful conclusions. For instance, one can compute meaningful sets consisting of non-conflicting arguments (called extensions) to decide whether several arguments can be accepted together. Other techniques such as ranking semantics are able to compute a ranking on arguments from the AAF. The intuition would be that the “best” arguments would be those that are less attacked.

We quickly recall below the usual extension semantics first defined in [10] for AAFs.

Definition 2 (Extension semantics) Let $S = (A, E)$, $E \subseteq A$ and $A \in A$. We say that:

- $E$ is conflict free iff there exists no arguments $A, B \in A$ such that $(A, B) \in E$.
- $E$ defends $A$ iff for every argument $B \in A$, if we have $(B, A) \in E$ then there exists $C \in E$ such that $(C, B) \in E$.
- $E$ is admissible iff it is conflict free and defends all its arguments.
• $E$ is a complete extension iff $E$ is an admissible set which contains all the arguments it defends.

• $E$ is a preferred extension iff it is maximal (with respect to set inclusion) admissible set.

• $E$ is a stable extension iff it is conflict-free and for all $A \in A \setminus E$, there exists an argument $B \in E$ such that $(B, A) \in E$.

• $E$ is a grounded extension iff $E$ is a minimal (for set inclusion) complete extension.

**Example 2 (Cont’d Example 1)** Using the AAF of Example 1, the admissible extensions are $\{f, a\}, \{a\}, \{d\}, \{a, d\}, \{a, f, d\}$, the complete extensions are $\{d\}, \{a, f, d\}$, the preferred extensions are $\{a, f, d\}$ and the grounded extension is $\{d\}$.

### 2.2 Timed Argumentation Framework

In [8, 9], a novel timed argumentation framework is proposed as an extension of Dung’s AFF. This framework called *Timed Abstract Framework* (TAF) combines arguments and temporal notions. In this formalism, arguments are only valid for a certain period of time, this particular period of time is called availability interval.

**Definition 3 (Availability Intervals)** Let $E$ be a set of elements and $T$ be a set of time-points. An availability interval for $E$ on $T$ is $\text{Av}: E \rightarrow \mathcal{X}_T$.

**Definition 4 (TAF)** A TAF is $Q = (A, R, \text{Av})$, where $A$ is a set of arguments, $R$ is a relation between arguments and $\text{Av}$ is availability function for time arguments.

**Example 3** Based on the AFF from Example 1, we can add an availability interval to each argument to obtain the TAF of Figure 2.2. The label $[0, 80]$ above $A$ means that the argument $A$ is valid only between the time-points 0 and 80 (included). Similarly, the label $(0, 60)$ above $C$ means that $C$ is valid between the time-points 0 and 60 (excluded).

![Figure 2.2: TAF for Example](image-url)

In [7], the authors studied properties of TAF by defining the notion of acceptability of arguments. Next, in [6], the authors proposed the *Timed Bipolar Argumentation Framework* (T-BAF) which is an extension of TAF that enhances the framework...
with positive (support) and negative (attack) interactions between arguments varying over time. They also translated the notions of acceptability for T-BAF by introducing the notion of \( t\text{-profile} \), binding arguments to sets of time intervals. In [14], the authors explained how to combine time with logic. Their contributions were twofold: (1) they showed how propositional logic can be used to express time by representing the relationships between intervals and (2) how arguments and attacks can be constructed upon this temporal knowledge.

### 2.3 Argumentation Frameworks with Sets of Attacking Arguments

In [16], the authors proposed the argumentation framework with sets of attacking arguments (SETAF) as a generalization of Dung’s framework. The difference with Dung’s framework is that SETAF allow arguments to jointly attack other arguments. They also provided new definitions and proofs to mirror Dung’s results on this more general framework. In [11], the authors provided a more complete characterization of SETAF, including the handling of various semantics not considered in the original publication of [16], a finer-grained representation of all acceptable semantics using labelling, and properties to allow the transition between extensions and labelings. Lastly, in [18], the authors generalised existing postulates for ranking-based semantics to SETAF. They introduced the first ranking-based semantics for SETAF by introducing a variant of the \( h\text{-categoriser} \) called NH-categoriser.

A SETAF is a pair composed of a set of arguments and a set of group attacks.

**Definition 5 (SETAF)** A SETAF is \( \Omega = (A, R) \) where \( A \) is a set of arguments and \( R \subseteq (2^A \setminus \emptyset) \times A \) is a set of sets of attacking arguments.

**Example 4** Let \( \Omega = (A, R) \) be a SETAF where \( A = \{A, B, C, D, E, F\} \) and \( R = \{(\{A\}, B), (\{A\}, C), (\{B\}, A), (\{B, C\}, D), (\{C\}, A), (\{D\}, E), (\{E\}, F), (\{F\}, D)\} \) represented in Figure 2.3.

![Figure 2.3: SETAF for Example](image-url)
Chapter 3

Temporal Frameworks with Sets of Attacking Arguments

In this project, we consider time as intervals (continuous period of time) following the intuition of [6]. Each time interval contains two time points, the beginning of the time interval and the end of the time interval. In my project, there are two forms of time interval. The first is a numerical form. For example, the value of the beginning of the time interval is 80 and the value of end of the time interval is 90. For this form, it is required that the value of the beginning of the time interval is less than the value of end of the time interval. The second is time-points. The beginning of the time interval is 09/10/2020 and the end of the time interval is 20/12/2020. The beginning of the time interval must be before the end of the time interval.

**Definition 6 (Time-points)** We say that $\mathcal{T}$ is a set of time-points if there is a total, reflexive and transitive binary relation $\preceq$ on $\mathcal{T}$.

A time interval $I$ refers to a continuous period of time contained in two different time points. The first time-point is called the startpoint of $I$, and the second time-point is called the endpoint of $I$. The time interval is said to be closed if it includes both the startpoint and endpoint. It is open if it does not contain the startpoint and endpoint. Otherwise, $I$ is semi-closed, i.e. it includes one of the aforementioned time-points but not both [6]. Please note that contrary to [6], we will use the notation ',' instead of '-' in intervals.

**Definition 7 (Time interval)** A time interval $I$ on $\mathcal{T}$ is $\langle a, b \rangle$ where $a, b \in \mathcal{T}$ such that $a \preceq b$ and $\langle$ is either $[ or ( and ) is either $]$ or $\rangle$. If $I = [a, b]$ then $I$ is said to be closed, else $I = (a, b)$ then $I$ is open, otherwise $I$ is semi-closed. We use the notation $I^- = a$ to denote the startpoint of $I$ and $I^+ = b$ to denote the endpoint.

**Example 5** Let $\mathcal{T} = \{09/07/2020, 11/07/2020\}$ and $09/07/2020 \preceq 11/07/2020$. The interval $I_1 = [09/07/2020, 11/07/2020]$ is open, it contains the startpoint and endpoint. Let $I_2 = (09/07/2020, 11/07/2020)$ is semi-closed, it only contains the endpoint but does not contains the startpoint. Let $I_3 = (09/07/2020, 11/07/2020)$ is closed, it does not contains startpoint and end point.

If $I_1$ and $I_2$ are two time intervals, we denote by $I_1 \cup I_2$ the combined time of $I_1$ and $I_2$. Multiple time intervals can be combined into a set of time intervals. For example, we will use $\{(1, 4), (15, 20)\}$ to represent $(1, 4) \cup (15, 20)$. We call a time intervals set on $\mathcal{T}$, or just intervals set, a finite set of time intervals on $\mathcal{T}$. We denote by $\mathcal{X}_\mathcal{T}$ the set of all possible time interval sets on $\mathcal{T}$. 
3.1 Temporal SETAF

In this section, we introduce our new formalism to integrate time into SETAF. The intuition of [6] is that an argument can only be legal for a specific interval of time in the argumentation process. In the next definition, we introduce Budan’s notion of availability interval.

The availability interval of arguments means that arguments are only valid for a continuous period of time. In many cases, the relationship between the arguments is not permanent. Availability means that their relationship (including attack and defend) is only established within a certain period of time. Interval refers to the time interval. A certain period of time contains a pair of time points, namely the startpoint and the endpoint. Between the pair of time points, the relationship between the arguments is valid, but the relationship is not valid outside the pair of time points.

A Temporal Argumentation Framework with Sets of Attacking arguments (T-SETAF) is composed of a set of arguments, a set of attacking arguments and an availability intervals that binds a time interval set to each argument. The attack relationship of one argument set to another argument is only valid for a period of time. The concept of T-SETAF adds the concept of time to the framework with sets of attacking arguments (SETAF).

**Definition 8 (T-SETAF)** A T-SETAF on $T$ is a tuple $\mathcal{AF} = (\mathcal{A}, \mathcal{R}, \mathcal{Av})$ such that $\mathcal{A}$ is a set of arguments, $\mathcal{R} \subseteq (2^\mathcal{A} \setminus \emptyset) \times \mathcal{A}$ and $\mathcal{Av}$ is an availability intervals for $\mathcal{A}$ on $T$.

**Example 6** Let us consider the T-SETAF $\mathcal{AF} = (\mathcal{A}, \mathcal{R}, \mathcal{Av})$, where $\mathcal{A} = \{A, B, C, D, E, F\}$, $\mathcal{R} = \{(\{A\}, B), (\{A\}, C), (\{B\}, A), (\{B, C\}, D), (\{C\}, A), (\{D\}, E), (\{E\}, F), (\{F\}, D)\}$ and the $\mathcal{Av}$ is defined such that $\mathcal{Av}(A) = \{(0, 80)\}, \mathcal{Av}(B) = \{[0, 60]\}, \mathcal{Av}(C) = \{(20, 80)\}, \mathcal{Av}(D) = \{[70, 90]\}, \mathcal{Av}(E) = \{(40, 90)\}$ and $\mathcal{Av}(F) = \{(50, 120)\}$. 

![Figure 3.1: T-SETAF for Example](image)

The attack relation between the arguments in the T-SETAF is related to availability interval of the arguments involved in the attack. Each argument has its availability interval. Only when the availability interval of the all the arguments in the set of attacking arguments overlaps with the availability interval of the attacked argument, the attack relation between them become effective, an attainable attack will occur.

**Definition 9 (Attainable attack)** Let $\mathcal{AF} = (\mathcal{A}, \mathcal{R}, \mathcal{Av})$ be a T-SETAF, $A \in \mathcal{A}$ and $B \subseteq \mathcal{A} \setminus \emptyset$ such that $(B, A) \in \mathcal{R}$. The attack $(B, A)$ is said to be attainable if $\mathcal{Av}(A) \cap \bigcap_{b \in B} \mathcal{Av}(b) \neq \emptyset$. 
The attack \((B, A)\) is attainable on \(I\) iff \(Av(A) \cap \left(\bigcap_{b \in B} Av(b)\right) \cap I \neq \emptyset\). We denote by \(R^I_{AF}\), the set of all attacks in \(AF\) that are attainable on \(I\).

**Example 7** The attack \(\{(A), B\}\) is attainable on \((0, 60]\) and the attack \(\{(B, C), D\}\) is not attainable because \((20, 80]\) \(\cap \) \([0, 60]\) \(\cap \) \([70, 90]\) = \(\emptyset\).

When an argument set \(A\) attack an argument set \(B\), at the same time, the argument set \(A\) is attacked by an argument set \(C\), in this cases, the argument set \(B\) is defended by the argument set \(C\). However, if there is no intersection between the availability interval of \(C\) attacking \(A\) and the availability interval of \(A\) attacking \(B\). This time interval called threat interval, the argument set \(C\) can not defend the argument set \(B\) by attack the argument set \(A\) in this time interval.

**Definition 10 (Threat interval)** Let \(AF = (A, R, Av)\) be a T-SETAF, \(A \in A\) and \(B \subseteq A \setminus \emptyset\) such that \((B, A) \in R\). The threat interval of \(B\) to \(A\), denoted \(\tau_{BA}\), is defined as \(Av(A) \cap \left(\bigcap_{b \in B} Av(b)\right)\).

**Example 8** The threat interval of \(B\) to \(A\) is \([0, 60]\) because \(Av(A) \cap Av(B) = (0, 80] \cap [0, 60] = [0, 60]\). Similarly, the threat interval of \(C\) to \(A\) is \([20, 80]\) because \(Av(A) \cap Av(C) = (0, 80) \cap [20, 80] = [20, 80]\).

Given an argument set \(A\), an argument set \(B\) and an argument set \(C\) and a time interval \(I\). In time interval \(I\), the argument set \(A\) attacked the argument set \(B\) and the argument set \(B\) attacked the argument set \(C\). In this case, \(A\) defend \(C\) by attack \(B\).

**Definition 11 (Defender)** Let \(AF = (A, R, Av)\) be a T-SETAF, \(D, E \subseteq A \setminus \emptyset\) and \(F \in A\) such that \((E, F) \in R\) and there exists \(e \in E\) with \((D, e) \in R\). The interval in which \(D\) defends \(F\) against \(E\) is \(\delta_{DE} = \tau_D \cap \tau_F\). We say that \(D\) provides a full defence for \(F\) against \(E\) iff \(\delta_{DE} = \tau_E\).

**Example 9** Consider a T-SETAF \(AF = (A, R, Av)\), where \(A = \{A, B, C, D\}\), \(R = \{\{(A, B), C\}, \{(C), D\}\}\), and \(Av(A) = [5, 30]\), \(Av(B) = [0, 30]\), \(Av(C) = [3, 15]\) and \(Av(D) = [0, 7]\). It holds that all the attacks of \(R\) are attainable. The threat interval \(\tau_{[C]}\)
\[ = Av(C) \cap Av(D) = [3, 7]. \] The set \( \{A, B\} \) defends \( D \) against \( C \), the defense interval \( \delta_{[A,B]}^D[C] = \tau_{[A,B]}^C \cap \tau_{[C]}^D = [5, 15] \cap [3, 7] = [5, 7] \). The set \( \{A, B\} \) does not provide full defense of \( D \) against \( \{C\} \) in this case.

In a T-SETAF system, acceptability must be concerned with time. A set of arguments \( A \) provides a full defense to an argument \( B \) by attack every attacker of \( B \) in a specific time interval.

Definition 12 (Acceptable) Let \( \mathcal{AF} = (\mathcal{A}, \mathcal{R}, Av) \) be a T-SETAF, \( A \in \mathcal{A} \) and \( S \subseteq \mathcal{A} \setminus \emptyset \). We say that \( A \) is acceptable with respect to \( S \) if for every \( X \subseteq \mathcal{A} \setminus \emptyset \) such that \( (X,A) \in \mathcal{R} \), \( S \) defends \( A \) from \( X \).

Analyzing the acceptability need to consider a strict time interval. For a set of the time interval and a time interval, need to establish a definition to express the relation between them.

Definition 13 (t-connected [8]) Let \( S \subseteq \mathcal{X}_T \) be a set of intervals and \( I \in \mathcal{X}_T \) be an interval of time. \( S \) is said to be t-connected on \( I \), denoted \( S \leftrightarrow I \), if (1) there exists \( X \in S \) such that \( I^- \in X \) and \( I^+ \in X \), or (2) \( \exists Y \in S \) such that \( I^+ \in Y \) and \( (S \ominus Y^+) \leftrightarrow [I^-, (Y^- - 1)] \), where \( S \ominus i = \{i : I \in S \text{ and } i \notin I\} \) for some set of intervals \( S \) and some moment of time \( i \).

Example 10 Let \( S \) be the set of availability intervals s.t. \( S = \{[01/09/2020, 05/09/2020], [05/09/2020, 10/09/2020], [05/10/2020, 05/11/2020]\} \) and \( I = [06/09/2020, 07/09/2020] \). Let us show that \( S \) is t-connected to \( I \). Let \( X = [05/09/2020, 10/09/2020] \in S \), we have that \( I^+ = 07/09/2020 \in X \), \( I^- = 06/09/2020 \in X \). We then have that \( S \) is said to be t-connected on \( I \). This satisfies condition (1) of Definition 12.

Example 11 Let \( S = \{[0,3], [3,5], [7,10]\} \) and \( I = [2,4] \). Let us show that \( S \) is t-connected to \( I \). Let \( Y = [3,5] \in S \), we have that \( I^+ = 4 \in Y \), \( Y^+ = 5 \), \( S \ominus Y^+ = \{[0,3], [7,10]\} \). We then have \( S \ominus Y^+ \) is t-connected to \( [2,2] \) because \( 2 \in [0,3] \). This satisfies condition (2) of Definition 13.

Definition 14 (Interval-based acceptability function) [8] The interval-based acceptability function is defined as \( F^I_{\mathcal{AF}}(S) = \{A : A \text{ is acceptable with respect to } S \text{ in } I\} \).

Example 12 Based on the T-SETAF from example 6. Let \( S = \{A\} \subseteq \mathcal{A} \), the interval \( I = (0,80) \). Argument \( D \) is acceptable with respect to \( S \) in \( I \) because \( Av(D) = [70,90] \cap I \neq [] \) and \( \forall X \in A \) such that \( (X,D) \) that \( S \) defends \( D \) from \( X \) in \( I \). \( F^I_{\mathcal{AF}}(S) = \{D : D \text{ is acceptable with respect to } S \text{ in } I\} \) is interval-based acceptability function.

3.2 Semantics for T-SETAF

The set \( tCE^I_{\mathcal{AF}} \) contains all arguments that are acceptable with respect to \( tCE^I_{\mathcal{AF}} \).

Definition 15 (Timed complete extension) Let \( \mathcal{AF} = (\mathcal{A}, \mathcal{R}, Av) \) be a T-SETAF and \( I \) an interval. The timed complete extensions in \( I \) of \( \mathcal{AF} \), noted as \( tCE^I_{\mathcal{AF}} \) are the fix points of the function \( F^I_{\mathcal{AF}} \).

The set \( tGE^I_{\mathcal{AF}} \) contains arguments that can be defended to the grounds while they are available in \( I \) [8].
Definition 16 (Timed grounded extension) [8] Let $\mathcal{AF} = (A, R, Av)$ be a T-SETAF and $I$ an interval. The timed grounded extension in $I$ of $\mathcal{AF}$, noted as $t\text{GE}_I^{\mathcal{AF}}$, is the least fix point of the function $F^I_{\mathcal{AF}}$.

The set $t\text{PE}_I^{\mathcal{AF}}$ are the maximal set of arguments (w.r.t. inclusion) among the timed complete extensions $t\text{CE}_I^{\mathcal{AF}}$.

Definition 17 (Timed preferred extension) Let $\mathcal{AF} = (A, R, Av)$ be a T-SETAF and $I$ an interval. The timed preferred extension in $I$ of $\mathcal{AF}$, noted as $t\text{PE}_I^{\mathcal{AF}}$, is the maximal fix point of function $F^I_{\mathcal{AF}}$.

Example 13 Based on the T-SETAF from Example 3.1, let us consider the interval $I = [50, 60]$. Admissible extension are $\{e\} \{a\} \{b\} \{c\} \{b,c\} \{e, a\} \{e,b,c\}$. Timed complete extension are $\{e\} \{e, a\} \{e,b,c\}$. Timed grounded extension is $\{e\}$. Timed preferred extension is $\{e,b,c\}$.

3.3 A New Input Format for T-SETAF

T-SETAF adds the notion of time to each argument, so its input format is also different from SETAF. It adds the concept of time to the input format of SETAF.

The Answer Set Programming Argumentation Reasoning Tool on the Formal Argumentation at DBAI website\(^1\) provides input formats for many argumentation frameworks. We added the time concept format to the SETAF input format to create the input format of T-SETAF.

The following Figure 3.3 shows the specific input format. The left part of the figure is a T-SETAF system, and the right part of the figure is the format of the input file corresponding to the T-SETAF system.

- arg(a): a is an argument
- att(x,b): the attack named x attacks argument b
- timelist(a,x): the time interval set of argument a is x
- mem1(x,t): the time interval t is in x
- mem2(x,b): the argument b is in x

3.4 Extracting SETAF from T-SETAF

In a T-SETAF system, each argument has its time interval, and each argument is only valid within its time interval set. Therefore, the attack relationship between the arguments and argument set is not permanent, and the attack relationship is also valid within a specific period of time.

Since T-SETAF is obtained by adding the notion of time to each argument based on SETAF, if we give a specific time interval, we can extract a SETAF. To find out which arguments in T-SETAF have valid time that coincides with the given specific time interval. We check if the valid time of a certain argument overlaps with the given specific time interval, it proves that the argument is valid within this time. If the arguments with an attack relation are valid within this time, then the attack

\(^1\)https://www.dbai.tuwien.ac.at/research/argumentation/aspartix/
Chapter 3. Temporal Frameworks with Sets of Attacking Arguments

Figure 3.3: T-SETAF Input Format

Given a TSETAF \( Q = (A, R, Av) \) with
\( A = \{a, b, c, d\} \) and
Attack \( R = \{(b,c,a), (a,b),(d,a)\} \)
\( Av = \{(a, (0, 80]), (b, (0, 100]), (c, (20,90]), (d, (0,120])\} \)

That we have 4 arguments \( a, b, c, d \) and three attacks
\( r_1 = (a, b), r_2 = (b, c), r_3 = (d, a) \)

\begin{tabular}{|c|c|}
\hline
TSETAF & INPUT FILE \\
\hline
\hline
arg(a) & time_list[a, L1] \\
arg(b) & time_list[b, L2] \\
arg(c) & time_list[c, L3] \\
arg(d) & time_list[d, L4] \\
\hline
\end{tabular}

When we gave a specific time interval of \((70, 120)\), the T-SETAF in 3.4 was converted to SETAF in 3.5.

Figure 3.4: Representation of T-SETAF
3.4. Extracting SETAF from T-SETAF

**Figure 3.5:** The SETAF Extracted from T-SETAF
Chapter 4

Implementation

This chapter introduces the implementation of the project. Please recall that the research question of our project is: "How can we extend SETAF with the notion of time and what is the impact of time on the output of the existing semantics?". Based on this research direction, it is clear that the core task of the project is to include the concept of time in SETAF.

The implementation of the project is divided into three stages: preliminary preparation for project implementation, analysis and design stage and execution stage. The preliminary preparation mainly includes the estimation of project development time, technical assessment, and project risk assessment. The analysis and design stage determines the architecture used by the project, the programming language to be used for writing the project, the technical means used in development, the source and input form of the data used in the project, and the functions included in the project. The execution phase mainly includes code writing, testing and evaluation.

This part mainly introduces the last two stages of project implementation, which are mainly divided into architecture, programming language, technical means, user interface and use case modules to describe in detail.

4.1 Technologies and Architecture

In software development, many projects use a three-tier structure. However, the three-tier architecture requires a long development period, involves more technologies, and targets a wide range of user groups. The targeted user groups for this project are all academic staffs working in the Argumentation domain or with basic notions in Argumentation. For this reason, the tool that we developed only implemented the core functions.

The core functions of the project include the parsing and reading of .apx files, the export of a SETAF into ASPARTIX, Visualisation of the T-SETAF and the STEAF, conversion from T-SETAF to the STEAF, calculation of the semantic extension of the STEAF, implementation of the Nh-categoriser and the implementation of the interface.

This project chooses Java as the development language. First of all, compared to other programming languages, I am more familiar with Java-based programming. Secondly, Java is an object-oriented language, and many libraries are provided internally. By using these libraries, the difficulty of development can be simplified and the development time can be shortened. we used IO streams for file reading and writing. The IO stream contains the Writer class and the Reader class. These two classes have buffer-based subclasses BufferedWriter and BufferedReader, which read and write data in character form. Due to the buffer, the efficiency of reading

\(^1\)https://docs.oracle.com/javase/8/docs/api/java/io/package-summary.html
and writing is higher than that of byte-wise reading and no buffer. We used Regular Expression to parse the data from the input file. Regular expressions is used to segment the parsed data to get the desired data. Parsed data is stored in HashSets or HashMaps, and most of the core functions are implemented based on the inter-collection operating. The graph is constructed using Java’s GraphStream library. The GraphStream library is not only powerful but also easy to use, as long as you import the corresponding jar package, you can use it directly. The following figure 4.1 represents the architecture diagram of the project.

![Architecture Diagram](image)

**Figure 4.1: Architecture**

The main functional modules of the project are shown in the architecture. First read the .apx file through the file operation in the IO stream, and read each line in the file through the BufferedReader, and use regular expressions to split each line of data read, and store the obtained data in the corresponding HashSet or in the HashMap collection. T-SETAF can be converted to SETAF at a given time interval, and dynamic graphics about T-SETAF and SETAF can be displayed through the GraphStream library. The data stored in the collection is written into the .apx file through the BufferedWriter, and the semantic extensions can be calculated through the operation between the collections.

### 4.2 Development Tools and Environment

This project does not use database storage data and Java Web development user interface, it is just a pure Java project. I selected IntelliJ IDEA as the IDE because IntelliJ IDEA is today’s mainstream Java integrated development environment tool software, with strong integration capabilities, internal integration of a series of tools such as github makes the operation easier, it has a powerful code prompt function for developers brought great convenience.

The development environment of this project is Java SE Development Kit 8. Compared with the earlier version, many new features have been added. Including a series of Lambda expressions, Date Time AP, etc. Compared to other Java SE

---

2https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html
3https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html
4https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html
5http://graphstream-project.org/
Development Kits, it is more stable, and many large companies are still using Java SE Development Kit 8.

### 4.3 Algorithms

The algorithms used in this project are mainly for computing semantic extension functions. Calculating semantics includes calculating the set of all subsets of arguments, calculating conflict-free extensions, obtaining Admissible extensions based on Conflict-free extensions, and calculating Complete extensions, Preferred extensions and Grounded extensions. Since the result returned by almost every method in computing extended semantics will be used as the parameter of the next method, the accuracy and efficiency of each function must be guaranteed.

#### 4.3.1 Finding All Subsets of Set

Since Java does not directly provide a method to calculate all subsets of a set, I implemented this method through an algorithm. There are two methods of calculating subsets that I have mastered, one is to use recursion to solve the subsets, and the other is to use bit operations to calculate all the subsets. Compared with the two methods, it is more convenient to use bit operations to calculate all subsets of the set.

The test case used in this project contains 5 arguments. From this, we know that the set contains 2 to the fifth power of subsets, which contains a total of 32 subsets. First, get the length of the set n, and then traverse from 0 to n-1. When traversing, bit operations are performed on each data of 0, 1, 2..., and the corresponding digits are judged one by one, which is the binary representation. The specific code implementation is as follows:

```java
public HashSet<HashSet<Argument>> getSubset() {
    int n = setOfArguments.size();
    ArrayList<Argument> ArrayArguments = new ArrayList<>(setOfArguments);

    HashSet<HashSet<Argument>> result = new HashSet<>();
    // Run a loop from 0 to 2^n
    for (int i = 0; i < (1 << n); i++) {
        HashSet<Argument> setToBeConstructed = new HashSet<>();
        int m = 1; // m is used to check set bit in binary representation.
        // Print current subset
        for (int j = 0; j < n; j++) {
            if ((i & m) > 0) {
                setToBeConstructed.add(ArrayArguments.get(j));
            }
            m = m << 1;
        }
        result.add(setToBeConstructed);
    }
    return result;
}
```

#### 4.3.2 Computing Conflict-free Extensions

Calculating conflict-free extensions is to traverse all the subsets of the set through the elimination method, and determine whether each subset satisfies the condition of
conflict-free extension, and the condition is that the arguments in the same set cannot directly have an attack relationship. If an argument set satisfies this condition, it is a conflict-free extension.

We first traverse all the subsets of the set to find out each argument and the attack set that attacks this argument. If a subset has both this argument and the set of arguments that attack it, then this subset is not a conflict-free extension. The specific code implementation is as follows:

```java
public HashSet<HashSet<Argument>> getConflictFree() {
    HashSet<HashSet<Argument>> allSubsets = getSubset();
    HashSet<HashSet<Argument>> result = new HashSet<>();
    for (HashSet<Argument> X : allSubsets) {
        Boolean IsXConflictFree = true;
        for (Relation R : mapOfRelation) {
            if (X.contains(R.getAttacked()) &&
                X.containsAll(R.getSetOfAttacker()))
                IsXConflictFree = false;
        }
        if (IsXConflictFree) result.add(X);
    }
    return result;
}
```

```java
public HashSet<Relation> getAttackersOfArgument(Argument a) {
    HashSet<Relation> set = new HashSet<>();
    for (Relation relation : mapOfRelation) {
        if (relation.getAttacked().equals(a)) {
            set.add(relation);
        }
    }
    return set;
}
```

### 4.3.3 Computing Admissible Extensions

According to the definition of admissible extension, the set in the obtained conflict-free extension is traversed to determine whether each argument in the traversed set is defended by the arguments in this set. If the existence argument in the set cannot be defended by the arguments in the set, then the set does not belong to the admissible extension.

Based on this judgment logic, we use a method of defining Boolean variables, but when the next condition is not met, the program returns to the previous level and finally judges based on the value of the defined Boolean variable. The implemented code is as follows:

```java
public HashSet<HashSet<Argument>> getAdmissible() {
    HashSet<HashSet<Argument>> admissible_set = new HashSet<>();
    for (HashSet<Argument> C : getConflictFree()) {
        Boolean IsCAdmissible = true;
        for (Argument c : C) {
            Boolean IsDefendedAgainstAllAttacks = true;
            for (Relation R : getAttackersOfArgument(c)) {
                Boolean IscDefendedAgainstR = false;
                for (Argument r : R.getSetOfAttacker())
                    if (!C.contains(r)) { // If the argument c is not defended against the attacker r
                        IsDefendedAgainstAllAttacks = false;
                        break;
                    }
                if (IsDefendedAgainstAllAttacks) IscDefendedAgainstR = true;
            }
            if (IscDefendedAgainstR) admissible_set.add(C);
        }
    }
    return admissible_set;
}
```

```java
public HashSet<Relation> getAttackersOfArgument(Argument a) {
    HashSet<Relation> set = new HashSet<>();
    for (Relation relation : mapOfRelation) {
        if (relation.getAttacked().equals(a)) {
            set.add(relation);
        }
    }
    return set;
}
```
4.3. Algorithms

for (Relation C1 : getAttackersOfArgument(r)) {
    if (C.containsAll(C1.getSetOfAttacker())) {
        IsDefendedAgainstR = true;
    }
} 

if (!IsDefendedAgainstR)
    IsDefendedAgainstAllAttacks = false;

if (!IsDefendedAgainstAllAttacks)
    IsCAdmissible = false;
}

if (IsCAdmissible)
    admissible_set.add(C);

return admissible_set;

---

4.3.4 Computing Complete Extensions

The conditions that a set is a complete extension must meet are: first, the set is an admissible extension, and second, it must contain all the arguments defended by it. Therefore, by traversing the admissible extension, we exclude the sets that do not meet the second condition one by one. Judge whether all the arguments defended by it are in a set: first, we find all the arguments that are not in this set, then we get the attack set of these arguments through the previously defined method, and then find the attacker who attacked each argument in the set, and finally judge whether our initial argument set contains these attackers. If it does, this argument set belongs to the admissible extension. The implemented code is as follows:

```java
public HashSet<HashSet<Argument>> getComplete() {
    HashSet<HashSet<Argument>> completeSet = new HashSet<>();
    for (HashSet<Argument> C : getAdmissible()) {
        Boolean IsComplete = true;
        for (Argument c : NotInSet(C)) {
            Boolean IsDefendedByC = true;
            for (Relation R : getAttackersOfArgument(c)) {
                Boolean IsDefendedAgainstR = false;
                for (Argument r : R.getSetOfAttacker()) {
                    for (Relation C1 : getAttackersOfArgument(r)) {
                        if (C.containsAll(C1.getSetOfAttacker())) {
                            //It means C defends c against R
                            IsDefendedAgainstR = true;
                        }
                    }
                }
            }
        }
    }

    if (!IsDefendedAgainstR)
        IsDefendedByC = false;
    
    if (IsDefendedByC)
        IsComplete = false;
    }

    if (IsComplete)
        completeSet.add(C);
    }
```
public HashSet<Argument> NotInSet(HashSet<Argument> C) {
    HashSet<Argument> result = new HashSet<>();
    for (Argument a : setOfArguments) {
        if (!C.contains(a))
            result.add(a);
    }
    return result;
}

4.3.5 Computing Preferred Extensions

The definition of a preferred extension is the maximal among the complete extensions. Traversing each set in the complete extensions, remove the set in the complete extensions to which set is traversed, and then judge whether the removed set is a subset of the complete extensions, this complete extensions is after removing the traversed set. If it is not a subset of it, add this set of arguments to the preferred extension. The specific code is as follows:

```java
HashSet<HashSet<Argument>> preferredSet = new HashSet<>();
HashSet<HashSet<Argument>> complete = getComplete();
for (HashSet<Argument> C : complete)
{
    Boolean IsCMaximal = true;
    HashSet<HashSet<Argument>> complete1 = getComplete();
    complete1.remove(C);
    if (!complete1.containsAll(C)) {
        IsCMaximal = false;
    }
    if (IsCMaximal) {
        preferredSet.add(C);
    }
}
return preferredSet;
```

4.3.6 Computing Grounded Extension

The definition of grounded extension is the minimal among the complete extensions. Traversing the complete extensions to get a set, and then determine whether this set is a subset of each set in the complete extensions. If this condition is met, the set is grounded extension. The specific implementation is as follows:

```java
public HashSet<HashSet<Argument>> getGrounded() {

    HashSet<HashSet<Argument>> groundedSet = new HashSet<>();
    for (HashSet<Argument> C : getComplete())
    {
        Boolean IsCGroundedSet = true;
        for (HashSet<Argument> D : getComplete())
        {
            if (!D.containsAll(C)) {
```
4.4 User Interface

The above is the algorithm implementation of this project in calculating the semantic extension function, but only the specific code of each function module is given. The complete code can be found on github.

4.4 User Interface

This project has been packaged into a jar file and uploaded to my personal github account. Users can download and use it on github.

The specific usage methods for project operation are as follows:

- Download the jar file from github link to the personal computer.
- Use the terminal of the personal computer to enter the directory where the jar file is stored, and enter the code `java -jar TSETAF.jar` to enter the main interface of the project. As shown in the figure below:

  ![Figure 4.2: Enter the user interface](image)

- Enter the file path of xxx.apx. As shown in the figure below:
- Enter the number corresponding to the function in the interface to execute the corresponding function.

---

6https://github.com/Aaron1206/TSETAF
Chapter 4. Implementation

FIGURE 4.3: Input the path of the xxx.apx file

FIGURE 4.4: Generate tsetaf
4.5 Testing

To test our implementation, we have run the jar file on different devices and found that the project can run normally. All functions included in the project can operate normally and get correct results. Since the test file provided by this project only contains 6 arguments, we increased the number of arguments to 50 and found that the tool can operate normally.
Chapter 5

Conclusion

In this report, we propose a new framework called Temporal Argumentation Framework with Sets of Attacking arguments (T-SETAF), which is a framework that adds the notion of time to Argumentation Framework with Sets of Attacking arguments (SETAF). First of all, we recalled Dung’s Abstract Argumentation Framework (AAF) and the usual argumentation semantics. Secondly, we introduced the Timed Argumentation Framework with the notion of time, and introduced the related definitions. Next, we introduced the Argumentation Framework with Sets of Attacking arguments (SETAF). Finally, we combined the time-related notion in TAF and SETAF to propose a new framework (T-SETAF). In the part of introducing T-SETAF, we gave a series of time-related definitions such as the definition of time points, time interval, and availability interval. Secondly, it gives the definition of T-SETAF and the definition involved in the framework, including Attainable attack, Threat interval, Defender. Finally, the semantics for SETAF are defined, including admissible extensions, complete extensions, preferred extensions and grounded extension. After introducing this framework, we defined the input format of T-SETAF and implemented some functions, mainly including parsing the examples of input format into T-SETAF demo, visualisation of T-SETAF, implementing the NH-categoriser and calculating the semantic extension of T-SETAF.

T-SETAF adds the concept of time to SETAF, which supports joint attacks, and is an extension of SETAF. This project does not provide ranking-based semantics and labellings, and the examples given in this project are abstract. In terms of implementation, first of all, the realized function is relatively single, and secondly, the project’s structure is very simple, excluding the database and user page, and the user experience may not be good enough.

In the future work, we will continue to improve the theoretical knowledge of T-SETAF, including a series of works such as semantics based on ranking, related extensions and labellings in T-SETAF. In terms of implementation, we will also add more functions and improve our user interface to make the user experience better.
Bibliography


[9] Maria Laura Cobo and Guillermo R Simari. “An approach to timed abstract argumentation”. In:


