FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Argumentation Frameworks

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software & Information Engineering

by

Patrick Bellositz
Registration Number 1027108

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Prof.Dipl.-Ing.Dr.techn. Christian Georg Fermüller

Vienna, 1st June, 2015

Patrick Bellositz	Christian Georg Fermüller

Erklärung zur Verfassung der Arbeit

Patrick Bellositz
Eichkogelstraße 10/10/9, 2353 Guntramsdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Juni 2015

Patrick Bellositz

Acknowledgements

I want to thank my family for supporting me and Dr. Fermüller for guiding me in the process of creating this thesis.

Abstract

This bachelor thesis explains argumentation frameworks and provides their most important semantics. It further analyzes the relations between different types of extension semantics.

The main part of this thesis is the the developement of a JAVA application which is capable of creating custom argumentation frameworks, their representation as graphs and the computation of extensions. The application is created in a way such that students can study argumentation frameworks in an easily digestable way including step-by-step explanations of how to compute extensions accompanied by coloured highlighting of important aspects of the frameworks' graphs.

Contents

CHAPTER 1

Introduction

Most humans argue daily. An argument is made, which is followed by a counter-argument and so on until there is a resolution. This way decisions can be made that satisfy the needs of the arguing parties, be it in a large group of people or a single person weighing all the information they have, the decision depending on the arguments that "win".

In 1995 Dung introduced *argumentation frameworks* as a basic theory to implement human argumentation mechanisms on computers in [1].

First we will define argumentation frameworks and analyse semantics for them representing both credulous and sceptical approaches to reasoning. An example is provided to illustrate the diffences in computing the sets (called extensions) these semantics yield.

Next we examine how the different extension types are related to each other. We also will look at additional semantics and compare them to Dung's original ones they build upon using additional examples.

In the last chapter we take a look at a computer program, specifically written along with this bachelor thesis, that allows the user to define argumentation frameworks (or, alternatively, choose from a set of examples) and that explains how the extensions of these frameworks are computed.

The goal of this thesis is to enable students to more easily acquire knowledge about argumentation frameworks by giving them the necessary information and providing them with a tool to aid them in their understanding of this topic.

CHAPTER 2

Definitions

To work with argumentation frameworks, first we must define their properties. In this section we will introduce the basic semantics for argumentation frameworks. If not otherwise mentioned the following definitions were proposed by Dung in [1].

2.1 Argumentation Frameworks

First we define argumentation frameworks as pairs of arguments and attacks relations.

Definition 1. An *argumentation framework* F is a pair (A, R), where A is a set of arguments and R is an attack relation.

The aforementioned attacks are defined as follows.

Definition 2. An *attack relation* $R \subseteq A \times A$ is a set of pairs (a, b), where $a, b \in A$ means *a attacks b.*

Remark 1. Let S be a set of arguments. If $a \in S$ and there is an attack $(a, b) \in R$ we say *S attacks b.*

Example 1. Imagine we have three arguments a_1, a_2, and b.

a_1 = "Blue is the most beautiful of all colours."
b = "No, black is much more beautiful!"
a_2 = "That's wrong, black isn't even a colour."

We can see that argument b attacks argument a_1 and vice versa. Additionally argument a_2 attacks argument b.

This results in the framework $F = (A, R)$, where $A = \{a_1, a_2, b\}$ and $R = \{(b, a_1), (a_1, b), (a_2, b)\}$.

It is also possible to represent every framework as a graph (V, E), where $V = A$ and $E = R$. The graph representing the framework in our example looks like this:

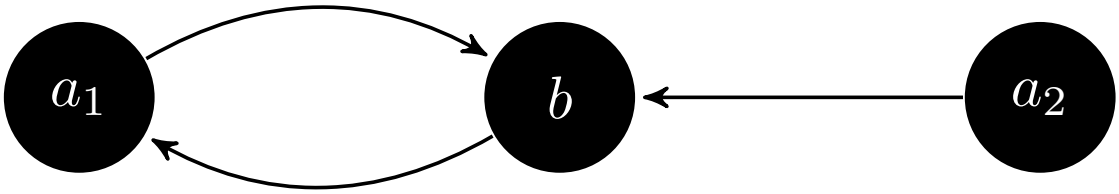

Figure 2.1: An argumentation framework about colours.

Remark 2. From now on, when mentioning A or R, we refer to the sets of an arbitrary, fixed argumentation framework $F = (A, R)$, unless otherwise stated.

The semantics of argumentation frameworks is captured by singling out certain so-called extensions intended to collect maximal subsets of arguments that are jointly maintainable according to various criteria with respect to the other arguments present in the argumentation framework.

2.2 Conflict-free sets

As a basis for extensions we introduce *conflict-free sets*. Since no extension should contain arguments that are in conflict with each other, there should be no attack consisting of two arguments of the same set.

Definition 3. Given an argumentation framework $F = (A, R)$, let $S \subseteq A$ be a set of arguments. It is *conflict-free*, if $\nexists(a, b) \in R$, where $a, b \in S$.
The set of all conflict-free sets of F is denoted $cf(F)$.

Example 2. (Continuation of Example 1.) As no argument attacks itself, $\{a_1\}$, $\{a_2\}$ and $\{b\}$ each are conflict-free. $\{a_1, a_2\}$ is also a conflict-free set, since there exists no attack consisting of both a_1 and a_2. The empty set is always conflict-free.

Since there is an attack between b and each of the other arguments, there are no other conflict-free sets.

It follows that $cf(F) = \{\emptyset, \{a_1\}, \{a_2\}, \{b\}, \{a_1, a_2\}\}$.

2.3 Admissible extensions

To be able to find a set of arguments that can defend itself, it does not suffice that a set is not in conflict with itself. Additionally each of its arguments should also be defended

from outside attackers as is the case with admissible semantics.

Definition 4. An argument a is *defended* by a set S, iff for every attack $(b, a) \in R$ there is an attack (c, b), where $c \in S$.

Definition 5. Let S be a conflict-free set. It is called an *admissible extension* if it defends each $a \in S$.
The set of all admissible extensions of an argumentation framework F is denoted $adm(F)$.

Example 3. (Continuation of Example 2.) Of the conflict-free sets only $\emptyset$ and $\{a_2\}$ don't get attacked. Therefore they are admissible extensions. $\{a_1\}$ defends itself from its only attacker b via the attack (a_1, b), making it an admissible extension. $\{a_1, a_2\}$ gets attacked via the attack (b, a_1), but a_1 gets defended through (a_1, b) and (a_2, b), also making it an admissible extension.
$\{b\}$ is not admissible since it doesn't defend its argument against the attacks by a_1 and a_2.

It follows that $adm(F) = \{\emptyset, \{a_1\}, \{a_2\}, \{a_1, a_2\}\}$.

2.4 Preferred extensions

Most given argumentation frameworks produce big numbers of admissible extension. Preferred semantics reduce the number of extensions, only taking the biggest ones. The resulting preferred extensions represent a credulous approach of reasoning, since they contain all arguments that are contained in admissible extensions.

They were defined in [2] as follows:

Definition 6. Let S be an admissible extension. It is called a *preferred extension* if for each $S' \subseteq A$, that is an admissible extension, $S \not\subset S'$.
The set of all preferred extensions of an argumentation framework F is denoted $prf(F)$.

Example 4. (Continuation of Example 3.) Since $\emptyset \subset \{a_2\}$, $\emptyset$ is not a preferred extension. Since $\{a_1\} \subset \{a_1, a_2\}$ and $\{a_2\} \subset \{a_1, a_2\}$, $\{a_1\}$ and $\{a_2\}$ are not a preferred extensions. Since all other admissible extensions are proper subsets of $\{a_1, a_2\}$, it is a preferred extension.

It follows that $prf(F) = \{\{a_1, a_2\}\}$.

As we can see, all arguments contained in admissible extensions still are contained in a preferred extension.

2.5 Stable extensions

Additionally we can define a stricter version of the admissible extension, that not only requires arguments to be defended, but also attacks all arguments not contained within it.

Definition 7. Let S be a conflict-free set. It is called a *stable extension* if for each $a \notin S$ there exists an attack $(b, a) \in R$ where $b \in S$.
The set of all stable extensions of an argumentation framework F is denoted $st(F)$.

Example 5. (Continuation of Example 2) The conflict-free set $\emptyset$ doesn't attack any of the other arguments, $\{a_1\}$ and $\{a_2\}$ only attack b, not attacking a_2 or a_1 respectively. $\{b\}$ doesn't attack a_2. Since they don't attack all arguments they don't contain, these sets are not stable extensions.
$\{a_1, a_2\}$ attacks the only other argument (i.e. b, via attack (a_2, b)) and therefore is a stable extension.

It follows that $st(F) = \{\{a_1, a_2\}\}$.

2.6 Complete extensions

Credulous reasoning contrasts sceptical reasoning. An extension matching sceptical reasoning within the context of argumentation frameworks is the complete extension.

Definition 8. Let S be an admissible extension. It is called a *complete extension* if for each $a \notin S$, S does not defend a.
The set of all complete extensions of an argumentation framework F is denoted $co(F)$.

Example 6. (Continuation of Example 3.) The admissible extension $\emptyset$ is not a complete extension because it defends a_2 which it doesn't contain. The same is true for $\{a_1\}$. $\{a_2\}$ defends a_1 and is therefore also not a complete extension.
$\{a_1, a_2\}$ attacks b and as there are no other arguments which could be defended it is a complete extension.

It follows that $co(F) = \{\{a_1, a_2\}\}$.

2.7 Grounded extension

As before, we can again reduce the number of extensions. In this case we search for the common denominator, meaning exactly those arguments all complete extensions can "agree" on. This set is called the grounded extension as defined in [2].

Definition 9. The (unique) *grounded extension* is defined by $\bigcap_{i=1}^{n} S_i$, where $\{S_1, ..., S_n\}$ is the set of all complete extensions.
The grounded extension of an argumentation framework F is denoted $gr(F)$

Example 7. (Continuation of Example 6.) Since there is only one complete extension $\{a_1, a_2\}$, it also is the grounded extension $gr(F)$.

CHAPTER 3

Relations and additional semantics

As we have seen in the previous chapter, there often are sets meeting the criteria for more than one extension type. In this chapter we look at the relations between extension types, explore their features and provide additional extension semantics. In advance the relations between extension types, including additional extension types, are shown in Figure 3.1 which was adapted from [4].

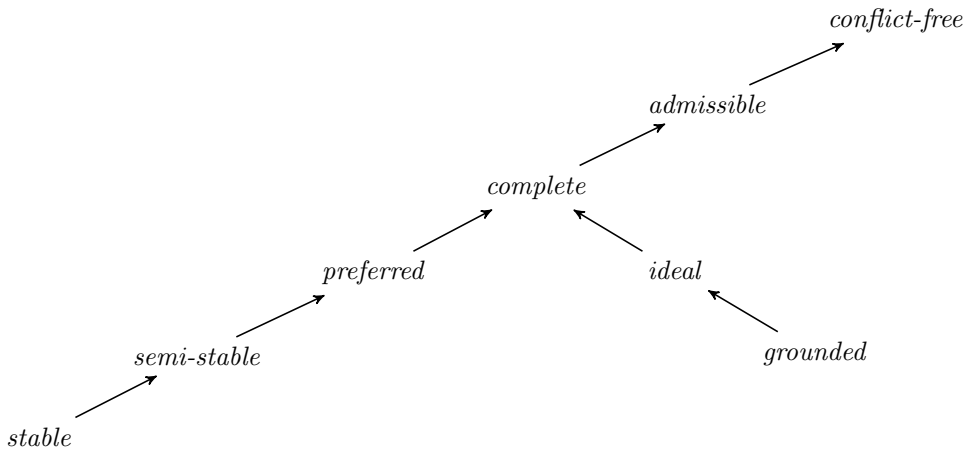

Figure 3.1: Relations between sets of an argumentation framework. An arrow from X to Y means that, if a set is X it also is Y

3.1 Relations between extension types

In this section we will examine relations between different types of extensions.

As per the definitions of the extensions there are the following relations:

$$adm(F) \subseteq cf(F) \tag{3.1}$$
$$st(F) \subseteq cf(F) \tag{3.2}$$
$$prf(F) \subseteq adm(F) \tag{3.3}$$
$$co(F) \subseteq adm(F) \tag{3.4}$$

Additional to relations already given in their definitions, there are further relations to be mentioned.

The following Lemma is provided in [1] and regards the relation between stable and preferred extensions.

Lemma 1. Each stable extension is a preferred extension, but not every preferred extension is a stable extension.

$$st(F) \subseteq prf(F) \tag{3.5}$$
$$prf(F) \not\subseteq st(F) \tag{3.6}$$

Therefore each stable extension also is an admissible extension.

Example 8. Let F be an argumentation framework (A, R) such that $A = \{a, b, c\}$ and $R = \{(b, a), (b, c), (a, a), (c, b)\}$.

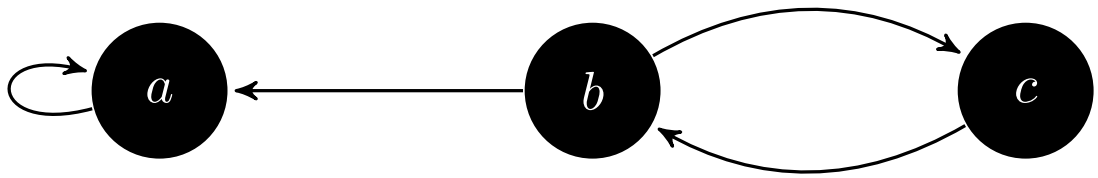

Figure 3.2: An argumentation framework F, where $st(F) \subsetneq prf(F)$

It can be computed that $cf(F) = adm(F) = \{\emptyset, \{b\}, \{c\}\}$. So we only have to consider three extensions for further computations.

As can be seen, $\{b\}$ is the only conflict-free set that attacks all other arguments. Therefore $st(F) = \{\{b\}\}$ and it becomes obvious that $st(F) \subsetneq prf(F)$.

Further we look at the relation between preferred, complete and grounded extensions. Dung provides the following Lemma in [1].

Lemma 2. Each preferred extension is a complete extension, but not vice versa.

$$prf(F) \subseteq co(F) \tag{3.7}$$
$$co(F) \not\subseteq prf(F) \tag{3.8}$$

Example 9. Let F be an argumentation framework (A, R) such that $A = \{a, b, c, d\}$ and $R = \{(a, b), (b, a), (c, b), (c, d), (d, c)\}$.

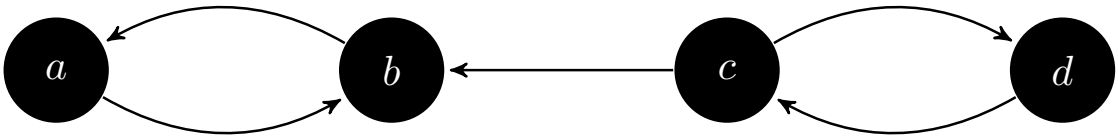

Figure 3.3: An argumentation framework F, where $prf(F) \subset co(F)$

To compute complete extensions as well as preferred extensions we use the set of admissible extensions $adm(F) = \{\emptyset, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}, \{b, d\}\}$.

All admissible extensions except $\{c\}$ contain all the arguments they defend. Hence they are complete. $\{c\}$ defends a, which it doesn't contain and therefore is not complete. It follows that $co(F) = \{\emptyset, \{a\}, \{d\}, \{a, c\}, \{a, d\}, \{b, d\}\}$

Of the admissible extensions $\emptyset, \{a\}$ and $\{d\}$ are subsets of $\{a, d\}$, while $\{c\}$ is a subset of $\{a, c\}$. Those are not preferred extensions. We get $prf(F) = \{\{a, c\}, \{a, d\}, \{b, d\}\}$, since these sets are not subsets of each other.

As we can see (3.7) and (3.8) are both satisfied.

Regarding the relation between grounded and complete extensions,[1] contains the following information:

Lemma 3. The grounded extension is the least (w.r.t. set inclusion) complete extension.

$$gr(F) \in co(F) \tag{3.9}$$

3.2 Additional semantics

In this section we will look at a few additional extension semantics and how they relate to the ones mentioned above.

3.2.1 Semi-stable extensions

The only one of Dung's original extensions that doesn't exist for every argumentation framework is the stable extension. A related, but less strict extension was definded in [3], the semi-stable extension. In [4] it is defined as follows:

Definition 10. Let S be a complete extension. It is called a *semi-stable extension* if the set $S \cup \{a \in A \mid S \text{ attacks } a\}$ is maximal w.r.t. set inclusion.
The set of all semi-stable extensions of an argumentation framework F is denoted $sst(F)$.

Since there always is a complete extension, there is always a semi-stable extension. Proofs for the following relations are provided in [3].

$$st(F) \subseteq sst(F) \tag{3.10}$$
$$sst(F) \subseteq prf(F) \tag{3.11}$$

The two following examples can originally be found in [3]:

Example 10. Let F be an argumentation framework (A, R) such that $A = \{a, b, c, d\}$ and $R = \{(a, a), (a, c), (b, c), (c, d)\}$

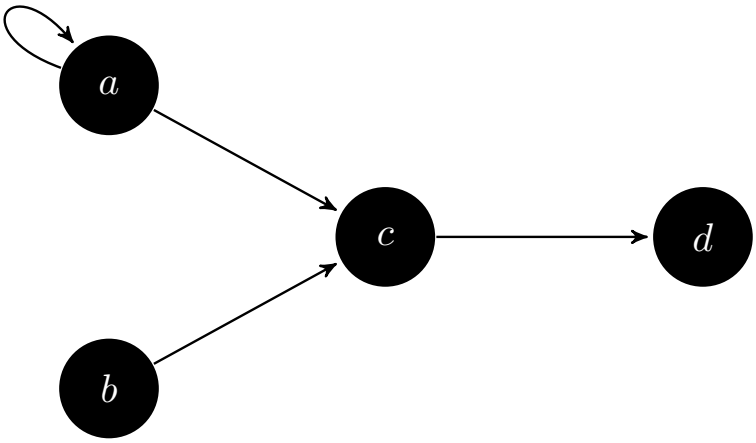

Figure 3.4: An argumentation framework F, where $st(F) \subset sst(F)$

It can be computed that $adm(F) = \{\emptyset, \{b\}, \{b, d\}\}$ and $co(F) = \{\{b, d\}\}$.

There is no extension within $adm(F)$, that attacks all arguments it doesn't contain. Therefore $st(F) = \emptyset$.

$\{b, d\}$ is the only complete extension, $\{b, d\} \cup \{c\}$ is maximal w.r.t. set inclusion, so it is semi-stable. It follows that $sst(F) = \{\{b, d\}\}$.

As we can see (3.10) is satisfied.

Example 11. Let F be an argumentation framework (A, R) such that $A = \{a, b, c, d, e\}$ and $R = \{(a, b), (b, a), (b, c), (c, d), (d, e), (e, c)\}$

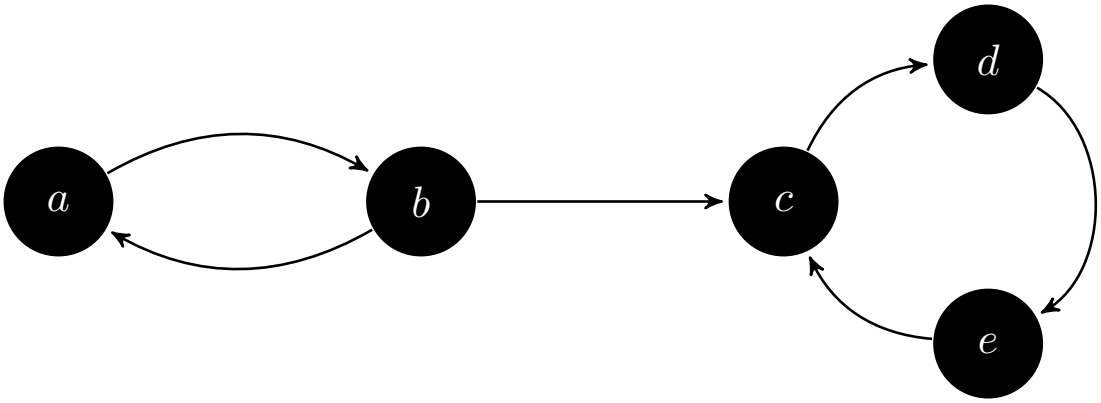

Figure 3.5: An argumentation framework F, where $sst(F) \subset prf(F)$

It can be computed that $co(F) = \{\emptyset, \{a\}, \{b, d\}\}$ and $prf(F) = \{\{a\}, \{b, d\}\}$.

To obtain $sst(F)$ we need to compute $S \cup \{a \in A \mid S \text{ attacks } a\}$ for all $S \in co(F)$. $\emptyset \cup \emptyset = \emptyset$, $\{a\} \cup \{b\} = \{a, b\}$ and $\{b, d\} \cup \{a, c, e\} = A$, only A being maximal w.r.t. set inclusion. Therefore $sst(F) = \{\{b, d\}\}$.

As we can see (3.11) is satisfied.

3.2.2 Ideal extension

There also is an alternative semantics to the grounded extension. Ideal extension semantics were introduced in [5] and also produce a unique sceptical set, albeit a less strict one.

Definition 11. Let S be an admissible extension. It is called the *ideal extension* if it is the maximal (w.r.t. set inclusion) admissible extension contained in every preferred extension.
The ideal extension of an argumentation framework F is denoted $id(F)$.

Ideal extension is a superset of the grounded extension according to [6]. In [7] it is shown that the ideal extension is always a complete extension.

$$gr(F) \subset id(F) \tag{3.12}$$
$$id(F) \in co(F) \tag{3.13}$$

Example 12. Let F be an argumentation (A, R) such that $A = \{a, b, c, d, e\}$ and $R = \{(a, b), (b, d), (c, b), (d, a), (d, e), (e, c)\}$.

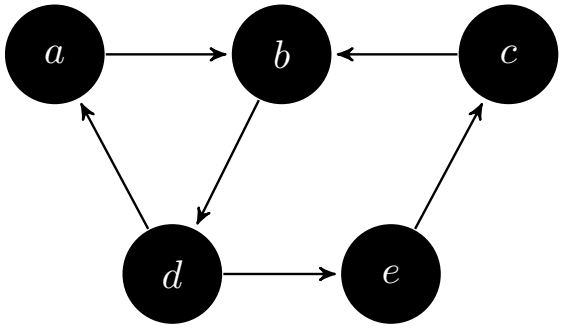

Figure 3.6: An argumentation framework F, where $gr(F) \subset id(F)$ and $id(F) \in co(F)$

We know that $prf(F) = \{\{c, d\}\}$ and $co(F) = \{\emptyset, \{c, d\}\}$.
It follows that $gr(F) = \emptyset$, while $id(F) = \{c, d\}$, so that (3.12) and (3.13) hold.

CHAPTER 4

Implementation

4.1 Introduction

In this chapter the usage and implementation details of the aforementioned program illustrating the computation of the different extension types is provided. The source code of the program as well as an executable .jar file can be found at `https://github.com/e1027108/bakk/`. The program contains the following functionality:

- Definition of argumentation frameworks
- Loading of preset argumentation frameworks
- Visualisation of argumentation frameworks as graphs
- Computation of extensions (including step-by-step explanations)
- Highlighting of extensions in the graph

The application consists of two views: the input mask and the graph view.

4.2 Input mask

On starting the application the user is presented with an input mask. It consists of ten rows representing the arguments of an argumentation framework. The number of arguments is limited to ten because a higher number of arguments doesn't provide any further benefit to a user wanting to learn about the basics of argumentation frameworks. Each row consists of a checkbox and two textfields.

Each checkbox is labelled with the name of the argument the row represents in capital letters (A ... J). If a checkbox is selected the program will use the represented argument for further computations and visualisations.

The first textfield of each row contains the argument's description. This can be a string of any form. The argument description is optional because it is only used in a cosmetic, non-functional fashion in the rest of the program.

The second textfield of each row contains the names of the arguments the argument in question attacks. Argument names are case-insensitive and can optionally be seperated either by ',' or spaces. Duplicate attacks are ignored. This means the attack fields accept the following syntax:

$$[a - jA - J\ ,]*$$

There are some preset examples to be chosen from via the drop-down menu labelled "choose preset". Once an example is chosen, selection and text fields are updated corresponding to the chosen preset. Presets include examples given in [1] and [2] as well as examples used in this thesis.

On clicking the "show graph" button, the application checks if the attack fields conform to the syntax for selected arguments and if they contain arguments that are not selected (meaning they don't exist and can therefore not be attacked). If there is invalid input an error message is displayed, if not the argumentation framework is created and the graph view displayed.

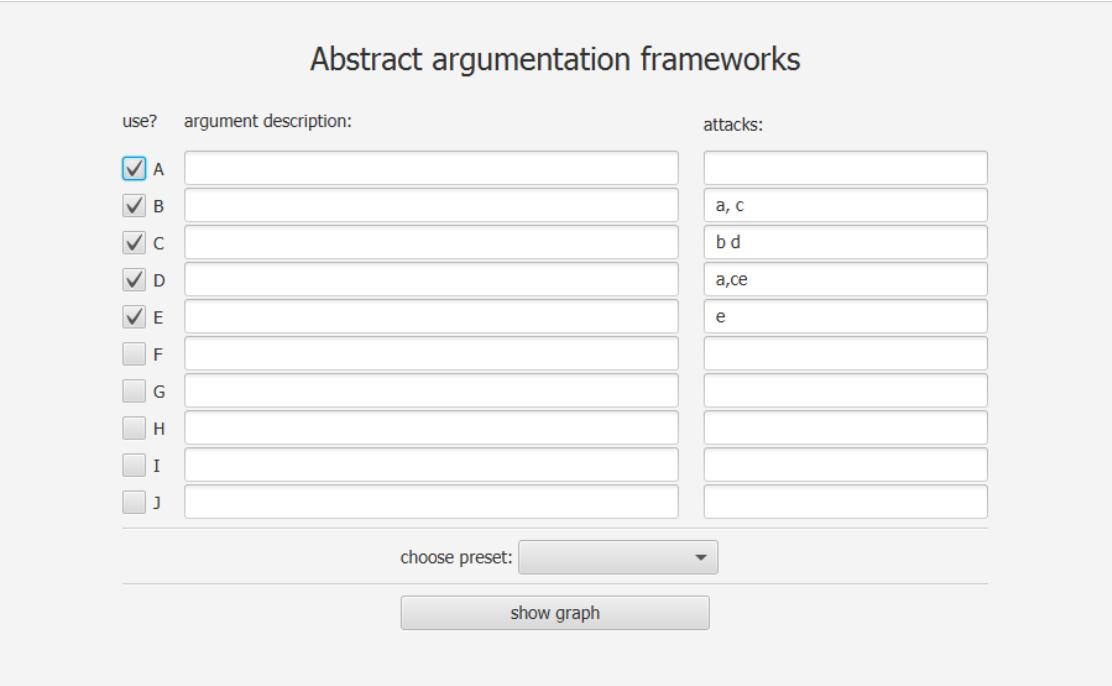

Figure 4.1: Input mask

Figure 4.1 shows the input mask with input representing the argumentation framework $F = (A, R)$ with $A = \{a, b, c, d\}$ and $R = \{(a, b), (b, c), (c, a), (d, b)\}$.

4.3 Graph view

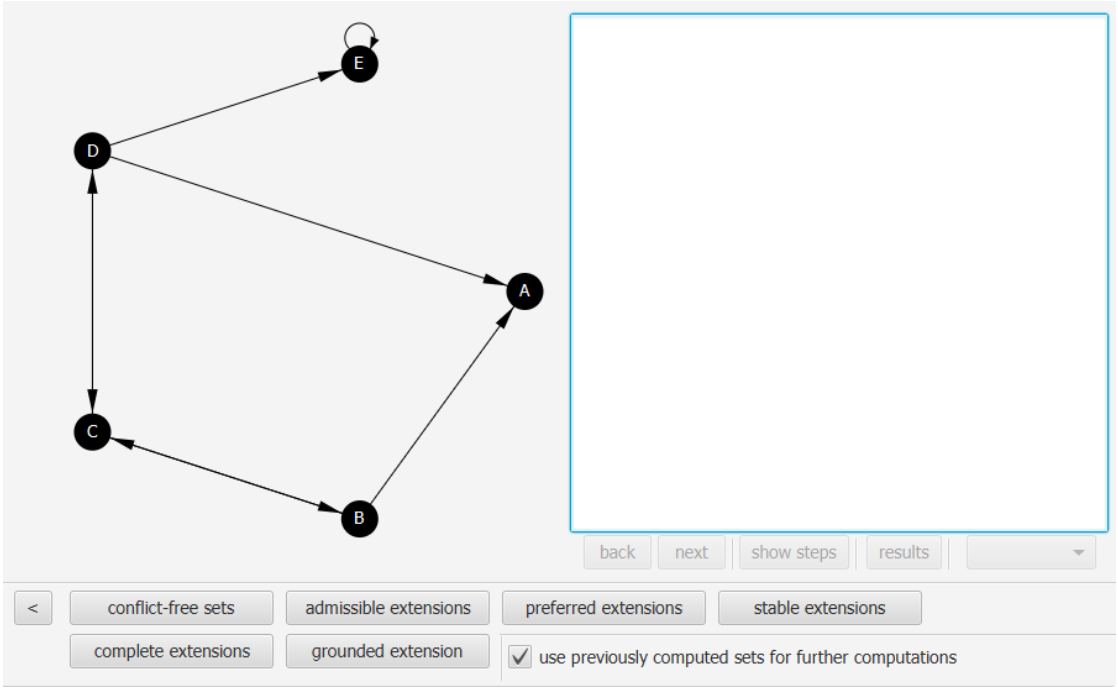

Figure 4.2: Graph view resulting from Figure 4.1

Upon loading the graph view (shown in Figure 4.2) the user can see the graph representing the argumentation framework defined in the input mask to the left. Every argument is represented by a node labelled with that arguments name. The argument nodes are arranged in a circle for better distinguishability.

It is possible to view the arguments' descriptions by hovering over them. In case no description was given for an argument, the application displays this information.

Figure 4.3: Tooltip as shown hovering over an argument without description.

The user then has the option to choose to compute one of six types of sets:

- conflict-free set
- admissible extension
- preferred extension
- stable extension
- complete extension
- grounded extension

If the checkbox to "use previously computed sets for further computations" is selected, each computation takes into account the results of preceding computations, if applicable. If this option is not selected the program will compute every extension type needed, prior to computing the chosen extension.
That means if conflict-free sets were already computed one can compute stable extensions without the need to compute conflict-free sets again, but when trying to compute preferred extensions the set of admissible extensions would have to be freshly computed (the computation of the admissible extensions using the already computed conflict-free extensions).

Each computation triggers the text area to the right (from now on called the explanation area) to display explanations of how the computed extensions came to pass. These explanations can either be shown step-by-step, all at once or be skipped so only the results of the computation are shown. Using previously computed extensions the explanation area does not show the explanation for already computed extensions again.

Figure 4.4 shows the explanation shown during the computation of the complete extensions of the argumentation framework shown in Figure 4.1. Note that in this case, admissible extensions (as a prerequisite to compute complete extensions) did not need to be computed, because they had been computed beforehand.

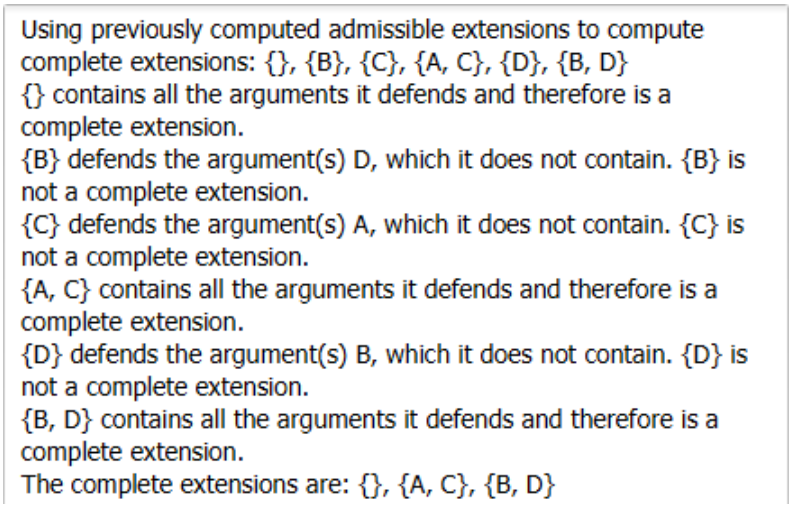

Figure 4.4: Text explaining the computation of complete extensions.

To further illustrate the explanations shown the application recolours the graph for each step, using three easily distinguishable colours:

Green is used for arguments included in the considered set and their relevant attacks needed to qualify for an extension type.
Red is used for conflicting arguments and attacks preventing a set of arguments from qualifying for an extension type.
Blue is used for arguments missing from a set for it to qualify for an extension type.

Figure 4.5 is an example where all three colours are used to illustrate the applications reasoning in computing complete extensions.

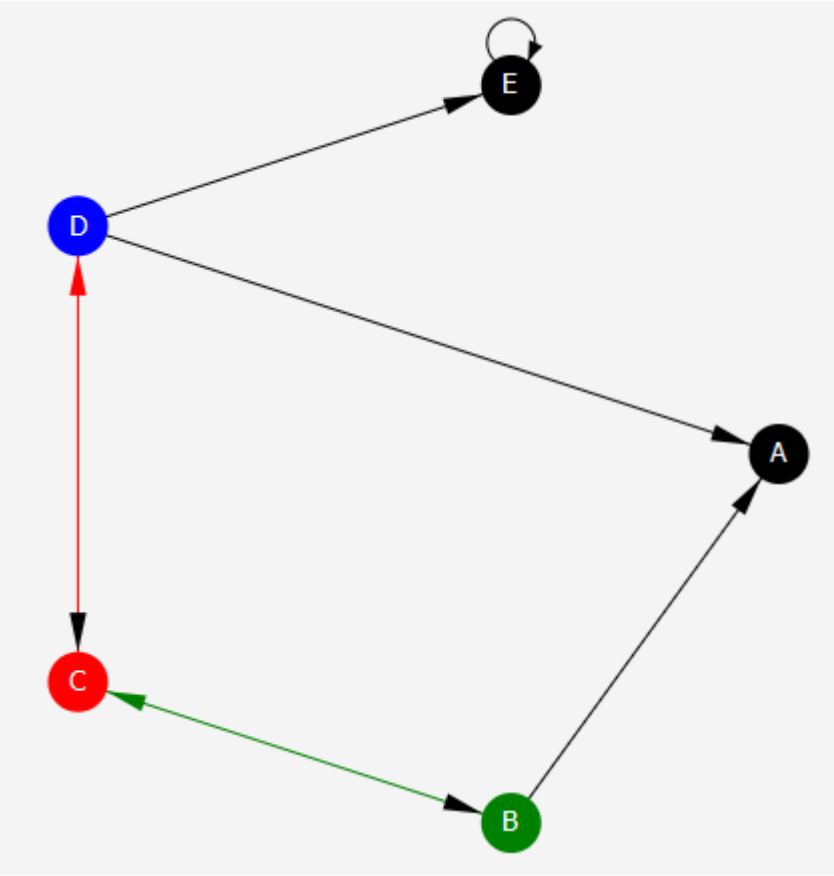

Figure 4.5: $\{B\}$ defends the argument D, which it does not contain. $\{B\}$ is not a complete extension.

After showing the results of a computation a dropdown menu becomes availiable beneath the right bottom corner of the explanation area, containing all computed extensions. They can be selected and are then highlighted in green in the graph.

The graph view also allows the user to go back to the input mask using the '<' button.

The input mask still contains the input from before switching to the graph view to allow the user to make small adjustments without having to redefine the whole framework.

4.4 Implementation details

This section lays out which technologies where used to create the program, what class structure was used to represent argumentation frameworks and what these classes do.

4.4.1 Technologies and Resources used

The application was written in Java (version 1.8) using eclipse Kepler as a working environment.

As the graphical user interface JavaFX was used, the design being created as .fxml files via JavaFX Scene Builder.

Java Universal Network/Graph Framework (JUNG) was used for the data structure of the graphs.

4.4.2 Program structure

There exist four classes in the application which model argumentation frameworks:

- Argument.java
- Attack.java
- Framework.java
- Extension.java

These classes have the following relationships:

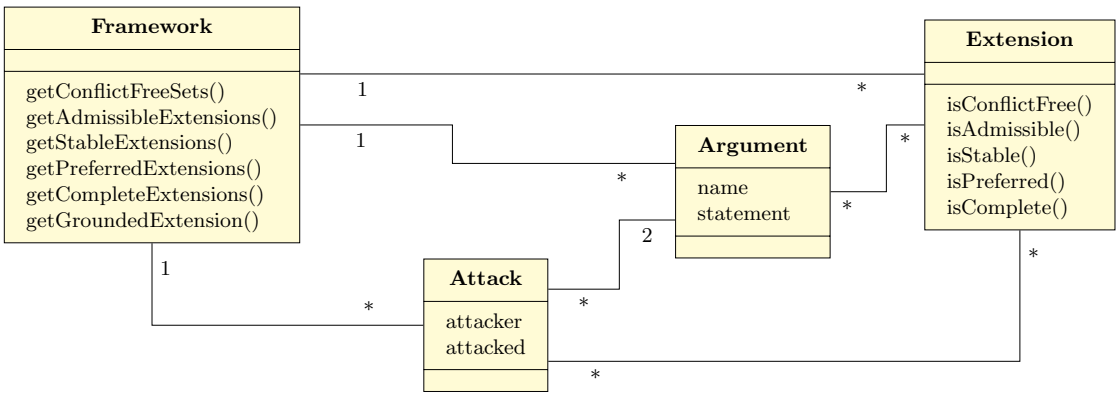

Figure 4.6: Argumentation framework class diagram

Argument

An *Argument* object is a pair consisting of a key and a value. The key is the name of the argument (representing an argument $a \in A$), while the value is the argument's statement.

Attack

An *Attack* object stores a pair of Argument objects. The first argument is called the "attacker", the second the "attacked" argument representing an attack $(a, b) \in R$.

Framework

A *Framework* object represents an argumentation framework $F = (A, R)$. It contains a list of all arguments (A) and a list of all attacks (R). It also stores all the extensions.

If a user wants to compute the extensions of a specific type (except the grounded extension), the framework calls the corresponding method of every extension candidate to find out if it indeed is of that extension type. For this purpose the framework can store multiple lists of extension candidates (e.g. The framework can store a list of conflict-free sets, which then can be used to compute admissible or stable extensions). After each candidate has returned its status, the frameworks returns a list of extensions that satisfy the criteria of the extension type in question.

To compute the grounded extension the framework simply compares the arguments of its complete extensions, without any algorithms of the extension object being called.

Extension

An *Extension* object represents a set $S \subseteq A$. For that purpose it stores a list of Argument objects. Every extension object stores a reference to the framework it belongs to, to compute and store lists of outgoing and incoming attacks.

Each extension object can compute if it is of a certain extension type (except for grounded extensions, see above).

List of Figures

Bibliography

[1] P. M. Dung, „On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-persons games“, *Artificial Intelligence*, vol. 77, no. 2, pp. 321–357, 1995.

[2] U. Egly, S. A. Gaggl, and S. Woltran, „Answer-set programming encodings for argumentation frameworks“, *Argument & Computation*, vol. 1, no. 2, pp. 147–177, 2010.

[3] M. W. A. Caminada, W. A. Carnielli, and P. E. Dunne, „Semi-stable semantics“, *Logic and Computation*, vol. 22, no. 5, pp. 1207–1254, 2011.

[4] N. Gorogiannis and A. Hunter, „Instantiating abstract argumentation with classical logic arguments: postulates and properties“, *Artificial Intelligence*, vol. 175, no. 9-10, pp. 1479–1497, 2011.

[5] J. J. Alferes, P. M. Dung, and L. M. Pereira, „Scenario semantics of extended logic programs“, in *Proceedings of the Second International Workshop on LPNMR*, P. L. M. and N. A., Eds., ser. Lecture Notes in Computer Science, MIT Press, 1993, pp. 334–348, ISBN: 0-262-66083-0.

[6] P. M. Dung, P. Mancarella, and F. Toni, „Computing ideal sceptical argumentation“, *Artificial Intelligence*, vol. 171, no. 10-15, pp. 642–674, 2007.

[7] P. Baroni and M. Giacomin, „On principle-based evaluation of extension-based argumentation semantics“, *Artificial Intelligence*, vol. 171, no. 10-15, pp. 675–700, 2007.