Abstract 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 the concept of the abstract argumentation framework, its properties as well as its extensions. It further analyzes the relations between different types of extensions.

The main part of the thesis is the modelling of the explained concepts as code and the developement of a JAVA application which is capable of creating custom argumentation frameworks, their visualisation 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 the extensions are computed accompanied by colored highlighting of important aspects of the frameworks' graphs.

Contents

Introduction

Most humans argue on a daily basis. Often argument is followed by counter-argument is followed by another counter-argument until one of the participants is defeated.

Each of these arguments consists of the following ingredients:

- Premises

- Conclusions

A counter-argument now tries to either disprove a premise or a conclusion. For example:

$$\underbrace{\text{All birds can fly.}}_{\text{premise}} \; \underbrace{\text{Tweety is a bird.}}_{\text{premise}} \; \underbrace{\text{Therefore Tweety can fly.}}_{\text{conclusion}}$$

One now could bring up the fact, that penguins are birds that can't fly, thus disproving the first premise. It could also be argued, that Tweety is not a bird, but rather a virtual character which can not be compared to the real world, attacking the second premise.

Now lets change our argument slightly:

$$\underbrace{\text{All birds can fly.}}_{\text{premise}} \; \underbrace{\text{Tweety can fly.}}_{\text{premise}} \; \underbrace{\text{Therefore Tweety is a bird.}}_{\text{conclusion}}$$

Now the conclusion of the argument could also be attacked.

In reality arguments are not always contructed in a way, where premises and conclusions can so easily be spotted, because they are often implicit. For example

Dirty things need to be cleaned. The floor is dirty.

consists only of premises and does not have an explicit conclusion.

While humans automatically grasp implicit premises or conclusions for simple arguments, this is not always the case. Complicated discussions or negotiations often are too complex to immediately understand. Computers gernerally also don't understand arguments very well. To use the optimal set of arguments one could now formalise the arguments and especially how they relate to each other to come to an optimal solution.

In this paper we look at the concept of argumentation frameworks as proposed by Dung in year. Following that we look at an application written for this thesis and specifically designed to illustrate and explain how such sets of arguments can be obtained.

CHAPTER 2 ■

Definitions

To work with argumentation frameworks, first we must define their properties. This section will lay out the basic structure of argumentation frameworks as well as provide extension types that can be used for further analysis.

2.1 Argumentation Frameworks

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

Definition 2. *Attack relations* $R \subseteq A \times A$ represent attacks as 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 3 arguments a_1, a_2, and b.

$$
\begin{aligned}
a_1 \quad &= \text{"Blue is the most beautiful of all colors."} \\
b \quad &= \text{"No, black is much more beautiful!"} \\
a_2 \quad &= \text{"That's wrong, black isn't even a color."}
\end{aligned}
$$

We can see that these arguments contain three attacks. 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 $R = \{(b, a_1), (a_1, b), (a_2, b)\}$.

A big argument framework in might become hard to read with increasing set sizes, so it is also possible to represent every framework as a graph (V, E), where $V = A$ and $E = R$. The graph of our example looks like this:
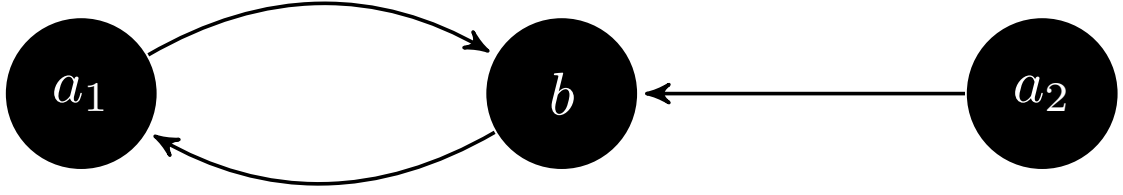
Figure 2.1: An argument framework about colors.

Depending on their properties, arguments can be grouped into *extensions* in order to obtain additional knowledge about the framework.

2.2 Conflict-free sets

As a basis we introduce the *conflict-free set*, since no extension should contain arguments that are in conflict with each other (i.e. there can't be attacks within an extension).

Definition 3. Let S be a set of arguments. It is *conflict-free*, if $\forall a \forall b \; a, b \in S, (a, b) \notin R$. The set of all conflict-free sets of an argumentation framework 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 relation containing a_1 and a_2. The empty set is always conflict-free.

Since there is an attack relation 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't be reasoned against, it does not suffice if that set is only not in itself conflicted, but each argument should also not be an invitation for an easy counter-argument. Therefore it is necessary to only accept arguments into a set that don't hurt its defendability.

Definition 4. An argument a is *defended* by a set S, if for every attack $(b, a) \in R$ there is an attack (c, b), where $c \in S$. If this is the case S *defends* a.

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 $adm(F)$.

4

Example 3. (Continuation of Example 2) Of the conflict-free sets only $\emptyset$ and $\{a_2\}$ don't get attacked. Therefore they are is admissible extensions. $\{a_1\}$ defends itself from its only attacker b via the attack relation (a_1, b), making it an admissible extension. $\{a_1, a_2\}$ gets attacked via the attack relation (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.

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

2.4 Preferred extensions

Using admissible extensions we now can reduce the number of relevant extensions by eliminating redundant extensions, all containing the same arguments, by only taking the biggest ones, that include smaller extensions. The resulting preferred extensions are an example of a credulous reasoning, since they contain all arguments that are contained in admissible extensions.

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 $pr(F)$.

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, even though the number of results is smaller.

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 needs to attack 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 is 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 $\{b\}$, missing an attack on $\{a_2\}$ or $\{a_1\}$

respectively. $\{b\}$ misses an attack on $\{a_2\}$. They are not stable extensions.

$\{a_1, a_2\}$ attacks all other arguments (i.e. b, attack relation (a_2, b)) and therefore is a stable extension.

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

2.6 Complete extensions

In contrast to credulous reasoning stands sceptical reasoning. The extension fitting sceptical reasoning within the context of argument 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 relevant extensions. In this case we search for the common denominator, meaning exactly those arguments all complete extensions can "agree" on. This results in the grounded extension

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 $gr(F)$.

Observations

As we have seen in the previous section, often there may be overlaps between the different extension types. In this section we will discuss those relations between extension types. Additionally we will explore their unique features.

3.1 Relations between extension types

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

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

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 7. Let F be an argumentation framework (A, R) such that $A = a, b, c$ and $R = (b, a), (b, c), (a, a), (c, b)$.
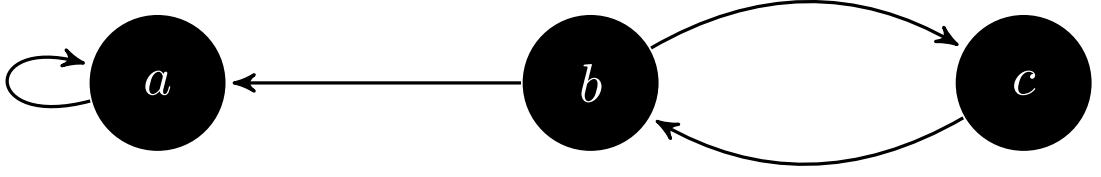
Figure 3.1: Example for the difference between stable and preferred extensions

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 $st(F) = \{\{b\}\}$.

We know that $\{b\}$ has to be a preferred extension, because of Lemma 1. $\{c\}$ attacks only it's attacker b, but not a. Therefore it is preferred and $prf(F) = \{\{b\}, \{c\}\}$.

Lemma 2. The grounded extension is always complete.

$$gr(F) \subseteq co(F) \tag{3.7}$$

$$co(F) \not\subseteq prf(F) \tag{3.8}$$
$$prf(F) \not\subseteq co(F) \tag{3.9}$$
$$\exists F \ prf(F) \cap co(F) \neq \emptyset \tag{3.10}$$
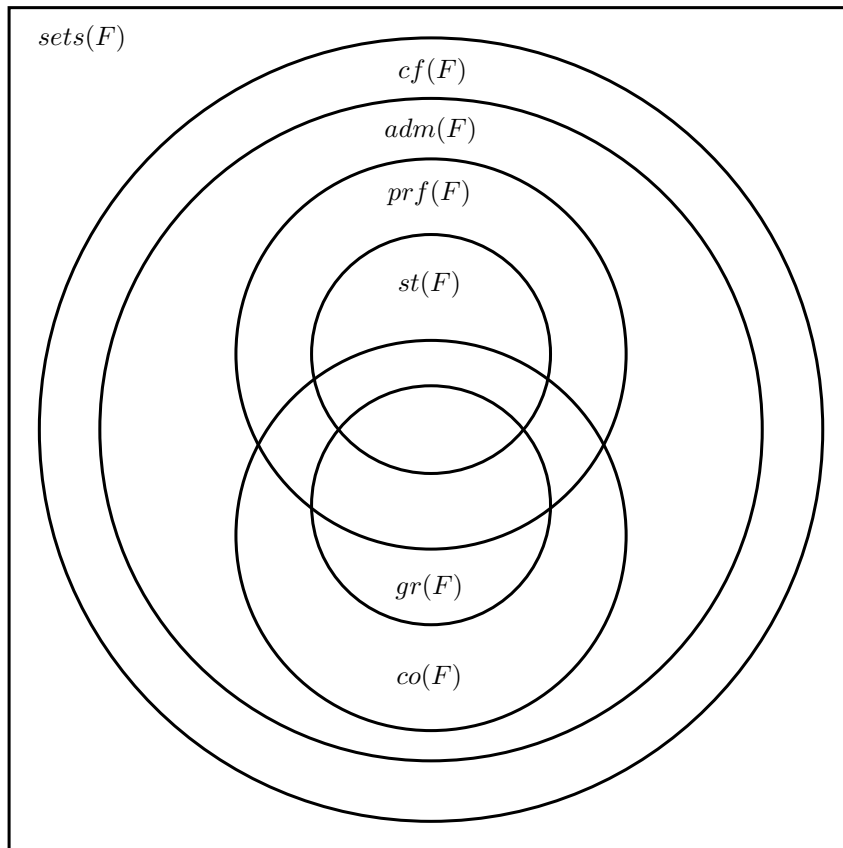$$\exists F \ st(F) \cap co(F) \neq \emptyset \tag{3.11}$$

Figure 3.2: Relations between extensions

Implementation

4.1   Introduction

In this section the usage and implementation details of the aforementioned program illustrating the computation of the different extension types is provided. 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 labeled with the name of the argument the row represents. 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 optional description of the argument in question. The argument description is optional because it is only used in a cosmetic, non-functional way 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 be, but don't have to be, seperated either by ',' or spaces.

The "show graph" button checks input in selected rows for problems. It detects attacks against non-existent arguments, while treating multiple attacks of an argument against another argument are ignored. It further provides error messages, so the user knows about issues. Note that unselected rows are ignored, even if there is a description or

attacks already defined. This enables the user to quickly add or remove arguments for comparison in the next window.

To make using the application easier, every item seen on screen shows a tooltip explaining the purpose of that element.

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

Once an argument framework is created, "show graph" clicked and no problems detected the graph view is shown.
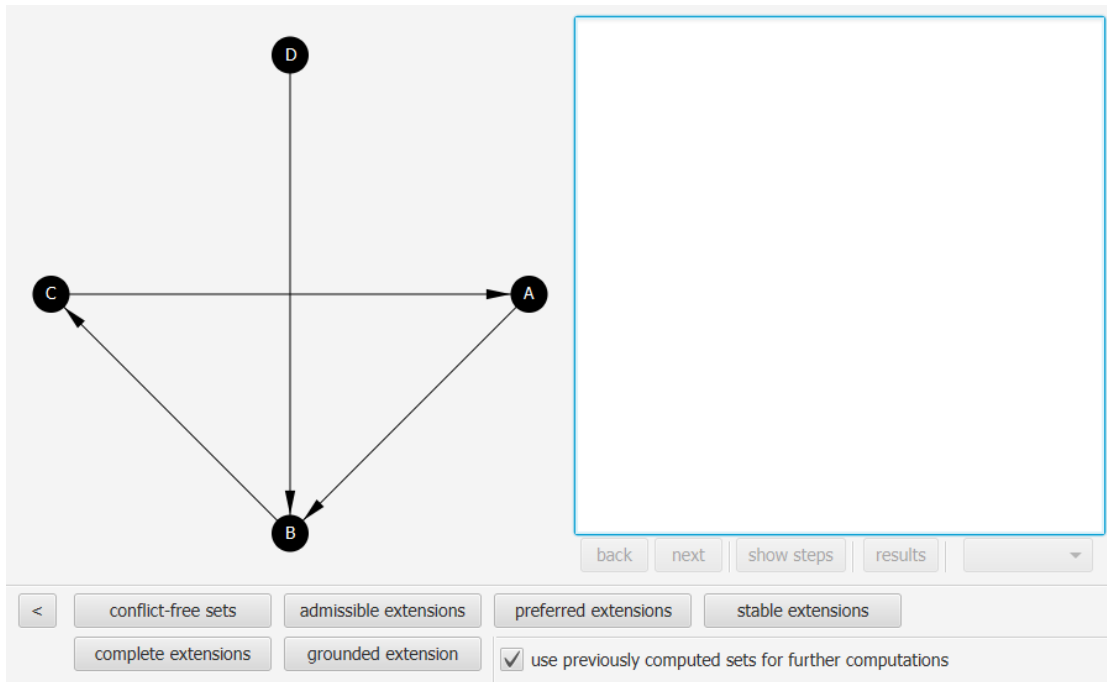
Figure 4.2: Graph view resulting from Figure 4.1

Upon loading the graph view the user can see the graph representing the argument 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 laid out in a circle for better distinguishability.

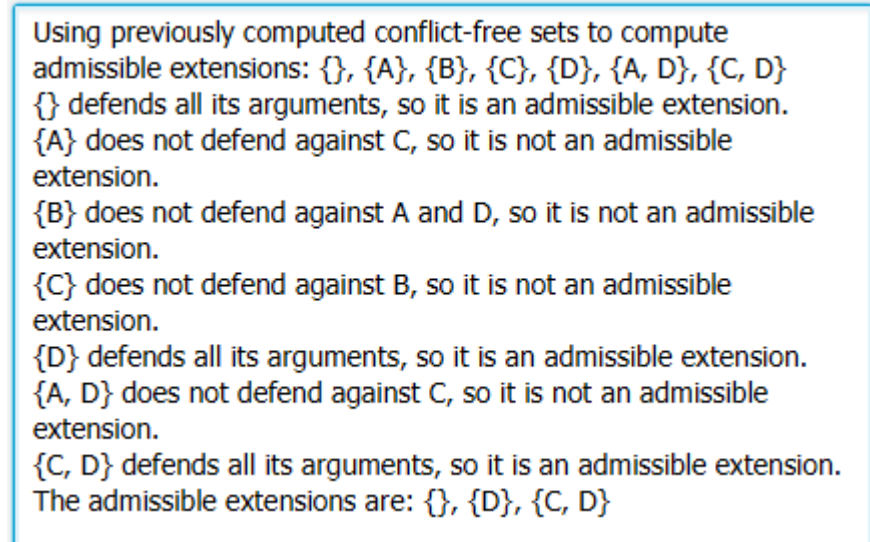The user then has the option to choose to compute one of six extension types:

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

If the checkbox to "use previoucly 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 (called the explanation area) to the right to display explanations of how the computed extensions came to pass. This explanation can either be shown step-by-step, all at once or be skipped and only the results be shown. Using previoucly computed extensions the explanation area does not show the explanation for already computed extensions again.

> Using previously computed conflict-free sets to compute admissible extensions: {}, {A}, {B}, {C}, {D}, {A, D}, {C, D}
> {} defends all its arguments, so it is an admissible extension.
> {A} does not defend against C, so it is not an admissible extension.
> {B} does not defend against A and D, so it is not an admissible extension.
> {C} does not defend against B, so it is not an admissible extension.
> {D} defends all its arguments, so it is an admissible extension.
> {A, D} does not defend against C, so it is not an admissible extension.
> {C, D} defends all its arguments, so it is an admissible extension.
> The admissible extensions are: {}, {D}, {C, D}

Figure 4.3: Text explaining the computation of admissible extensions.

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

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.
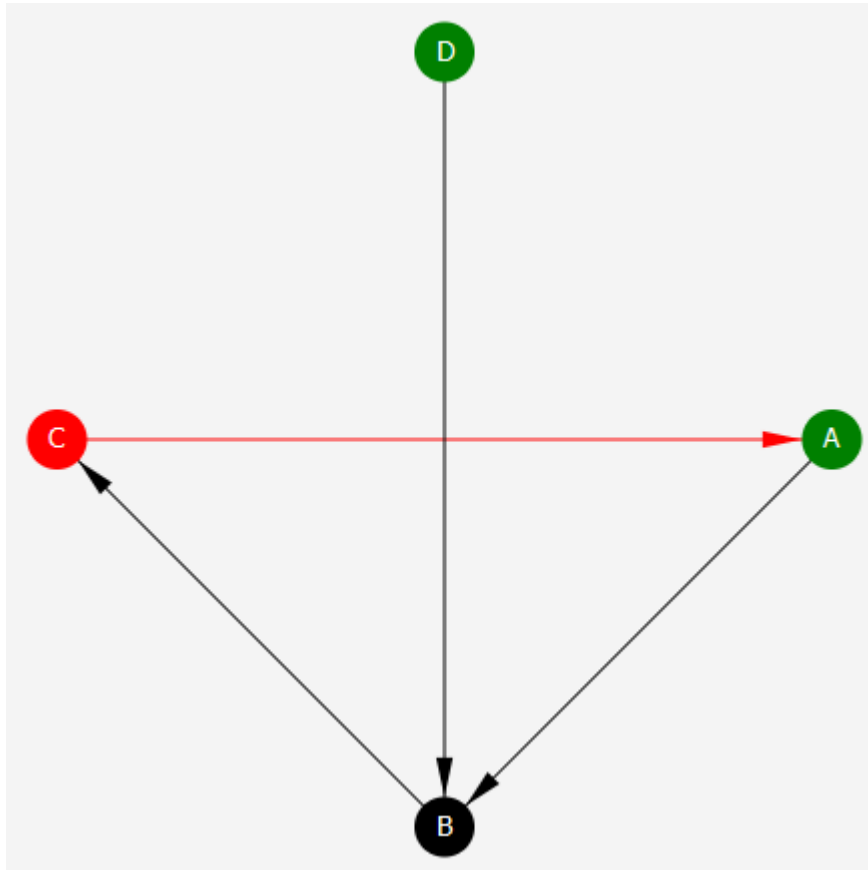
14

Figure 4.4: {A,D} does not defend against C, so it is not an admissible extension.

After showing the results of a computation a dropdown menu becomes availiable to the right bottom side 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 will still contain the input from before switching to the graph view.

4.4 Implementation details

This section lays out which technologies where used to create the program, how they where used and explains why the program was implemented in such a way.

4.4.1 Technologies used

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

To control 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 frameworks' graphs.

4.4.2 Data structure

Rather than implementing the sets A and R separately, I instead opted for storing each outgoing attack of an argument directly within the argument. The general data structure used is

$F = \{a_1, ..., a_n\}$, where $a_i = D$ and $D \subseteq F$, D being arguments attacked by a_i.

A simple example of a naive algorithm using that data structure follows:

Algorithm 4.1: Check whether a set is conflict-free

Data: set S of arguments
Result: whether S is conflict-free

1 foreach $a \in S$ **do**
2   **foreach** $b \in S$ **do**
3   **if** $b \in a$ **then**
4   return $False$;
5   **end**
6   **end**
7 end
8 return $True$;