FernUniversität in Hagen
Faculty of Mathematics and Computer Science

AIG
Artificial Intelligence Group

Implementation of Bipolar Argumentation Frameworks using Answer Set Programming

Bachelor's Thesis

in partial fulfillment of the requirements for
the degree of Bachelor of Science (B.Sc.)
in Informatics

submitted by

Timon Wagener

First examiner:	Prof. Dr. Matthias Thimm Artificial Intelligence Group
Advisor:	Lars Bengel Artificial Intelligence Group

Statement

Ich erkläre, dass ich die Bachelorarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Bachelorarbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Bachelorarbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

	Yes	No
I agree to have this thesis published in the library.	☒	☐
I agree to have this thesis published on the webpage of the artificial intelligence group.	☒	☐
The thesis text is available under a Creative Commons License (CC BY-SA 4.0).	☒	☐
The source code is available under a GNU General Public License (GPLv3).	☒	☐
The collected data is available under a Creative Commons License (CC BY-SA 4.0).	☒	☐

Hannover, 22.02.2023 [illegible]

(Place, Date) (Signature)

Zusammenfassung

Als einer der vier menschlichen rhetorischen Modi trägt die Argumentation eine entscheidende Rolle bei der Art und Weise, wie wir mit anderen Menschen interagieren. Argumente können dabei in zwei verschiedenen Beziehungen zu einander stehen. Auf der einen Seite können sie sich gegenseitig unterstützen. Auf der anderen Seite können Argumente auch dazu eingesetzt werden um andere Argumente anzugreifen. In der Informatik ist die Forschung daran interessiert, diese grundlegenden Eigenschaften der menschlichen Argumentation in Computermodellen zu implementieren. Mit dem Modell der bipolaren Argumentationssysteme können beide Relationen formuliert werden. Die Art und Weise, wie die Unterstützungsrelation interpretiert werden soll, hängt stark vom Kontext des Argumentationssystems ab. Dies führte zu mehreren unterschiedlichen Interpretationen der Unterstützungsrelation wie deduktiv, notwendig und evidentiell. Für die Implementierung ist das Paradigma der deklarativen Programmierung von Answer Set Programming ein sehr vielversprechender Ansatz. Während Implementierungen, für eine abstrakte Interpretation der Unterstützungrelation in der Literatur zu finden sind, gibt es diese nicht für eine deduktive, notwendige oder evidentielle Interpretation. Das Ziel dieser Bachelorarbeit ist die Entwicklung und Evaluierung eines auf Answer Set Programming basierenden Solvers für bipolare Argumentationssysteme mit der deduktiven, notwendigen oder evidentiellen Interpretation der Unterstützungsrelation.

Abstract

As one of the four human rhetorical modes, argumentation has a major impact on the way of how we are interacting with others. Arguments can have two different relations, they can support or they can attack each others. In Computer Science, research is interested in implementing these fundamental properties of human argumentation. With the model of bipolar argumentation frameworks both relations can be formulated. The way the support relation should be interpreted is highly related to the context of the argumentation problem, resulting to several different interpretations such as deductive, necessary and evidential. For implementation, the declarative programming paradigm of answer set programming is a very promising approach. Whereas such implementations, using answer set programming, for an abstract interpretation of support can be found in literature, none can be found for the deductive, the necessary as well as for the evidential interpretation of support. The aim of this bachelor thesis is the implementation and evaluation of an answer set programming based solver for bipolar argumentation frameworks with the deductive, necessary or evidential interpretation of support.

Acknowledgements

I owe a big gratitude to members of the University, my family and friends who have supported me in completing this work and my degree.

In particular, I want to thank my advisor Lars Bengel, for answering my questions at any time of the day. His notes and suggestions have been a great help for writing this thesis. In addition to that I want to thank Professor Doctor Matthias Thimm for the opportunity of writing this thesis in the Artificial Intelligence Group and his support during the time of writing.

Special gratitude is dedicated to my lovely wife Lena Wagener, for encouraging me to start my studies of Computer Science. Without her love, moral support and backing during the last three years, I would not be able to complete my degree.

In addition to that I want to thank my family, especially my parents and my sister, for always supporting my curiosity and believing in me.

Finally I want to thank Ian Jannasch for his notes and questions, that helped me a lot finding the right phrasings.

Contents

1 Introduction 1

2 Abstract Argumentation Frameworks 3
- 2.1 Introduction . . . 3
- 2.2 Acceptability . . . 4

3 Bipolar Argumentation Frameworks 7
- 3.1 Introduction . . . 7
- 3.2 Interpretation of the Support Relation . . . 8
- 3.3 Acceptability . . . 12
  - 3.3.1 Illustration of Acceptability . . . 14

4 Answer Set Programming 18
- 4.1 Syntax . . . 18
- 4.2 Semantics . . . 19
- 4.3 Functionality . . . 20

5 Related Work 25
- 5.1 ASPARTIX . . . 25
- 5.2 TweetyProject . . . 26

6 ASP-Encodings for Bipolar Argumentation Frameworks 28
- 6.1 Modelling Bipolar Argumentation Frameworks . . . 28
- 6.2 Basic Concepts . . . 30
- 6.3 Admissible Sets . . . 33
- 6.4 Preferred Extensions . . . 34
- 6.5 Complete Extensions . . . 39
- 6.6 Grounded Extension . . . 40
- 6.7 Stable Extensions . . . 42

7 Evaluation of ASP-Encodings 43
- 7.1 Generation of Test cases . . . 43
- 7.2 Experimental Design . . . 43
- 7.3 Results . . . 45
  - 7.3.1 Impact of the ratio of support relations: . . . 47
  - 7.3.2 Impact of the amount of arguments: . . . 49
  - 7.3.3 Impact of the semantics: . . . 51
  - 7.3.4 Impact of the type of admissibility: . . . 51
  - 7.3.5 Impact of the problem type: . . . 53
  - 7.3.6 Impact of the support interpretation: . . . 55
  - 7.3.7 Impact of the implementation: . . . 57

8 Conclusion 58

1 Introduction

We are drowning in information but starved for knowledge.
∼ John Naisbitt (1982) ∼

Argumentation is a major and very complex property of human intelligence. It is hard to find a universal declaration, however the following definition was found in [28] and illustrates the complexity of human argumentation:

> Argumentation is a verbal, social, and rational activity aimed at convincing a reasonable critic of the acceptability of a standpoint by putting forward a constellation of propositions justifying or refuting the proposition expressed in the standpoint.

This definition concentrates on the intuitive interpretation of argumentation, the defense of opinion in every day scenarios. When humans are confronted by contrary opinion they try to falsify this opinion with counterarguments. The arguments of the contrary opinion will be attacked with own arguments and each of the involved party tries to prove the correctness of the own opinion.

But human argumentation is not limited to defending and expressing the own opinion in every day scenarios, it is also used for understanding new problems by using logical reasoning. When humans are facing new problems they try to find a solution by using knowledge about similar problems. Humans try to understand the new problem and begin to weight different solutions, which can be interpreted as different arguments. The most reliable argument (resp. solution) will be chosen to overcome the problem.

In our fast changing world we are often confronted by new situations and new topics. No one of us ever thought that topics like pandemic, war and climate change will become that present in our daily lives. Almost every day we are forced to make new decisions with consequences that are unpredictable. Before making these decisions we try to weight the different options to choose the best or at least harmful decision for ourselves. This decision is highly influenced by information from news and opinions from other people.

Because of our almost unlimited access to the internet it is easier than ever to get information about a certain topic. The source of information, which is the basis of our opinion making, shifted from newspapers and books more and more to social media. Social media platforms are often unregulated, which leads to the possibility of expressing opinion for everyone who has access to the internet. This leads to an overflow of information and opinions in our daily lives.

Social media platforms are using different algorithms to present us only content we are interested in. But this also leads to the fact that we get a one sided view of information on these platforms. It becomes more difficult to proof the reliability of information on the one hand and easier to influence our opinion by presenting incomplete or one sided news on the other hand.

Platforms are trying to overcome this by using Artificial Intelligence (AI) systems to determine the acceptability of information. Therefore a lot of work has been done to determine the characteristics of human argumentation and to explore ways of implementing these characteristics on computers.

This work will present two very important models to formulate human argumentation: ***Abstract Argumentation Frameworks*** and ***Bipolar Argumentation Frameworks***. In section 2 the concept of abstract argumentation frameworks will be presented and in section 3 bipolar argumentation frameworks will be introduced.

When modeling human argumentation as a framework one is interested in determining the different acceptable sets of arguments computationally. For determining the acceptable sets the declarative programming paradigm of ***Answer Set Programming*** is a promising approach. This paradigm will be introduced in section 4, followed by presenting an overview of related work in section 5. Especially the ***Argumentation Reasoning Tool ASPARTIX*** will be introduced that builds the basis the developed solver. The aim of this bachelor thesis is the development of answer set programming based encodings for computing the acceptable sets of three different interpretations of bipolar argumentation frameworks. These encodings will be presented in Section 6. In Section 7 the developed encodings will be tested for efficiency and performance against an existing Java implementation. In the last section the most important parts of this work will be summed up.

2 Abstract Argumentation Frameworks

In this section a brief introduction to the concept of abstract argumentation frameworks will be presented. After illustrating the basic structure, different definitions regarding the acceptability of single arguments respectively sets of arguments will be presented. This section is concluded by an illustration of these definitions.

2.1 Introduction

Fundamental progress to explore ways to implement mechanisms of human argumentation has been done by Dung [8], resulting to the model of abstract argumentation frameworks, which is defined as follows:

Definition 1. *An **abstract argumentation framework** (AF), is a tuple $(\mathcal{A}, R_{att})$ that consists of a finite set of arguments $\mathcal{A}$ and a binary relation $R_{att} \subseteq \mathcal{A} \times \mathcal{A}$ called the attack relation.*

For illustrating this definition the two arguments x and $y \in \mathcal{A}$ should be considered. If $xR_{att}y$ exists, this means that argument x attacks argument y. Besides attacking other arguments, one argument can also defend another argument. The notion of defense was defined by Dung on the basis of the notion of attacks as the follows: a set of arguments E defends the argument x if and only if E attacks all attackers of x. A set E attacks an argument y if at least one argument $z \in E$ attacks the argument y. An AF can be represented by a directed graph, whose nodes are the arguments and edges are the attacks. For a better understanding of the concept of AFs, Example 1, that is extracted from [4], can be considered.

Example 1. *Tom, Ben and Dan want to go hiking. They have the following preference regarding the weather:*

1. *sunny*
2. *sunny and cloudy*
3. *cloudy but not rainy weather*

They will cancel the hike only if the weather is rainy. But clouds could be a sign of rain. In the morning they look at the sky and see that the weather is cloudy. The following exchange of informal arguments occurs between Tom, Ben and Dan:

t_1*: Today we have time,lets go hiking.*
b*: The weather is cloudy, clouds are a sign of rain, we better cancel the hike.*
t_2*: These clouds are early patches of mist, the day will be sunny*
(and we can go hiking).
d*: These clouds are not early patches of mist, however it will not rain anyways*
(and we can go hiking).

In this example there are several conflicts between the arguments: argument b attacks argument t_1, argument b is attacked by arguments d and t_2, argument d also attacks argument t_2, the resulting AF can be seen in Figure 1.

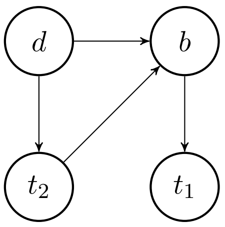

Figure 1: Abstract argumentation framework from Example 1.

As mentioned in the introduction of this work, the aim of modelling a framework is to compute the acceptable sets of arguments of $\mathcal{A}$. Intuitively a set of arguments is acceptable if it can defend all its elements against all attacks. In the upcoming subsection the properties and semantics of acceptable sets of arguments for AFs, will be presented in detail.

2.2 Acceptability

In [8] Dung has defined, that the acceptability of an argument depends on its inclusion in some sets, that are called acceptable sets or extensions. These sets are underlying specific requirements. A very intuitive requirement is that such a set is free of conflict, that means no argument of the set should attack another argument of the set. This is described by the property of ***conflict-free***:

Definition 2. *Let $(\mathcal{A}, R_{att})$ be an AF. A subset $S \subseteq \mathcal{A}$ is **conflict-free** if and only if $\nexists x, y \in S$ such that $xR_{att}y \in S$.*

The next property was mentioned before and describes the fact, that a set of arguments is only acceptable if it defends all its members against all attacks from other arguments. The respective property is called ***collective defense***:

Definition 3. *Let $(\mathcal{A}, R_{att})$ be an AF. A subset $S \subseteq \mathcal{A}$ **defends collectively** an argument x if and only if $\forall y \notin S$ and $yR_{att}x$, there exists an argument $z \in S$ so that $zR_{att}y$.*

Based on these properties several semantics regarding acceptability were definded and presented by Dung in [8]. The combination of the properties of conflict-freeness and collective defense lead the definition of ***admissible*** sets:

Definition 4. *Let $(\mathcal{A}, R_{att})$ be an AF. A subset $S \subseteq \mathcal{A}$ is an **admissible** set if and only if S is conflict-free and defends collectively all its elements.*

An AF can have multiple different admissible sets, in general the admissible sets with the highest number of included arguments, with respect to set-inclusion, are of special interest. These sets are defined as the ***preferred*** extensions:

Definition 5. *Let $(\mathcal{A}, R_{att})$ be an AF. A subset $S \subseteq \mathcal{A}$ is a* ***preferred*** *extension of $(\mathcal{A}, R_{att})$ if and only if S is maximal, with respect to set inclusion, among the admissible sets of $\mathcal{A}$.*

The exclusion of a single argument from a set is mostly caused by the fact, that it is attacked by an argument that is included in the set. If it would not be attacked, then it would be included in the preferred extension. This follows to the fact that sets that attack all arguments that are not included in the set, are also of special interest. Such sets are defined as ***stable*** extensions:

Definition 6. *Let $(\mathcal{A}, R_{att})$ be an AF. A subset $S \subseteq \mathcal{A}$ is a* ***stable*** *extension of $(\mathcal{A}, R_{att})$ if and only if S is conflict-free and S attacks all elements that does not belong to S.*

In contrast to stable extensions, sets that include all arguments, that are defended by the set are also desirable. Such a set is defined as a ***complete*** extension:

Definition 7. *Let $(\mathcal{A}, R_{att})$ be an AF. A subset $S \subseteq \mathcal{A}$ is a* ***complete*** *extension of $(\mathcal{A}, R_{att})$ if and only if S is an admissible set and each argument that is defended by S, belongs to S.*

In contrast to preferred extensions, which are characterized by maximality, the minimal complete extension of an AF is also a desirable set, that is defined as the ***grounded*** extension:

Definition 8. *Let $(\mathcal{A}, R_{att})$ be an AF. A subset $S \subseteq \mathcal{A}$ is the* ***grounded*** *extension of $(\mathcal{A}, R_{att})$ if and only if S is minimal, with respect to set-inclusion, among the complete extensions of $(\mathcal{A}, R_{att})$.*

For a better understanding of the properties and the different semantics, the following example with the AF shown in Figure 2, that was extracted from [13] should be considered.

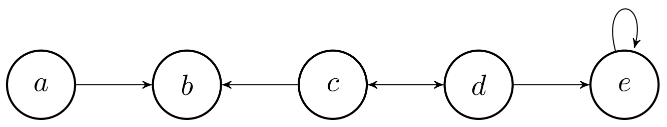

Figure 2: Abstract argumentation framework.

Example 2. *The AF is defined as: $\mathcal{A} = \{a, b, c, d, e\}$ and $R_{att} = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\}$. The AF consists of the following admissible sets: $\{\{\emptyset\}, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}\}$. Regarding to Definition 4, the empty set is always an admissible set. The maximal admissible sets (w.r.t. set-inclusion) were defined as the preferred extensions. In this example the two preferred extensions $\{\{a, c\}, \{a, d\}\}$ can be observed. In contrast to this the set $\{a, d\}$ is the only stable extension of the AF. The set $\{a, c\}$ is conflict-free, but the argument e, which does not belong to the set is not attacked and cannot be included because of the requirement of conflict-freeness. In addition to that the sets $\{\{a\}, \{a, c\}, \{a, d\}\}$ represent the complete extensions and therefore the set $\{a\}$ is the grounded extension of the AF, because it is the minimal complete extension (w.r.t. set-inclusion).*

In this section the concept of AFs, was introduced briefly. These frameworks consider only one possible kind of interaction between the arguments, the attack relation. This can be considered as the weakness of the model. With a detailed look at Example 1 another connection between the arguments can be found. The arguments d and t_2 both support the hiking project and therefore support the argument t_1. Such a connection between arguments cannot be modeled by an AF. For modeling such a situation a new kind of interaction is mandatory, the ***support relation***. The addition of a new relation results to the concept of a new framework, the ***bipolar argumentation frameworks***, that will be introduced in the upcoming section.

3 Bipolar Argumentation Frameworks

The model of bipolar argumentation frameworks was introduced by Cayrol and Lagasquie-Schiex in [6], by adding the support relation to AFs. Subsequently to the introduction of the basic structure, three different interpretations of the support relation and definitions regarding the acceptability of arguments, will be presented in this section. In the last subsection the differences in determining the acceptable sets of arguments, regarding the different interpretations of support, will be introduced.

3.1 Introduction

In [6] the following definition regarding bipolar argumentation frameworks can be found:

Definition 9. *A **bipolar argumentation framework** (BAF) $(\mathcal{A}, R_{att}, R_{sup})$ consists of: a finite set of arguments $\mathcal{A}$, a binary relation $R_{att} \subseteq \mathcal{A} \times \mathcal{A}$ called the attack relation and another binary relation $R_{sup} \subseteq \mathcal{A} \times \mathcal{A}$ called the support relation. These binary relations must satisfy the following consistency constraint: $R_{att} \cap R_{sup} = \emptyset$.*

For illustrating this definition the two arguments x and $y \in \mathcal{A}$ should be considered. If $xR_{att}y$ (resp. $xR_{sup}y$) exists, this means that argument x attacks (resp. supports) argument y. BAFs can also be represented as directed graphs, for that two kinds of edges are necessary, the attack relation ($\rightarrow$) and the support relation ($\rightsquigarrow$). Considering Example 1, the BAF from Figure 3 will be received.

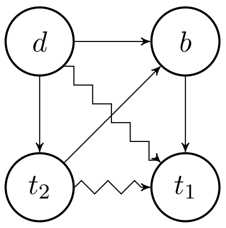

Figure 3: Bipolar argumentation framework from Example 1.

With the following extensive example, which was extracted from [4], the importance of the need for a new kind of interaction between arguments, can be underlined. Such a complex and real-world related scenario, about an exchange during a meeting of a newspaper, cannot easily be modelled using AFs alone.

Example 3. *The following arguments should be considered:*
a: Assuming agreement and no right of censor, information Y concerning X will be published.
b_1: X is the prime minister who may use the right of censor.
c_0: We are in democracy and even a prime minister cannot use the right of censor.
c_1: Also X has resigned and is no longer the prime minister.

d: Channel 1 has officially announced the resignation.
b_2: Y is a private information, so X will prevent the publication.
e: Y is an important information.
c_2: Any information concerning a prime minister is public information.
c_3: Y is of national interest, so Y cannot be considered as private information.
In this example several conflicts can be found: For example the arguments b_1 and b_2 attack argument a, the arguments c_0 and c_1 attack the argument b_1. The importance of a support relation is highlighted by the argument d, because it does not directly attack the argument b_1 and therefore is not defending the argument c_1. The argument d confirms the statement given by argument c_1, which leads to a support relation. The resulting BAF can be seen in Figure 4.

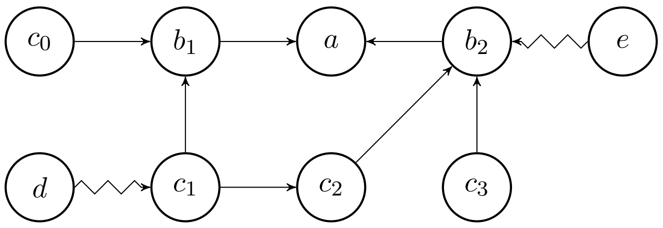

Figure 4: Bipolar argumentation framework from Example 3.

With the extension of AFs by adding the support relation extended issues occur. Two major problems will be introduced in detail in the following subsections.

3.2 Interpretation of the Support Relation

One major issue for BAFs is that there is no universal interpretation of the support relation. The respective interpretation is highly related to the context of the argumentation system. In [5] the detailed characteristics of the support relation were studied. According to Cayrol and Lagasquie-Schiex, the support relation can be interpreted in three different ways, with the following defintions:

1. **Deductive support:** If $xR_{sup}y$ then the acceptance of x implies the acceptance of y, and in consequence the non-acceptance of y implies the non-acceptance of x.

2. **Necessary support:** If $xR_{sup}y$ then the acceptance of x is necessary for the acceptance of y, or equivalently the acceptance of y implies the acceptance of x.

3. **Evidential support**: Enables to distinguish between *prima-facie* and standard arguments. *Prima-facie* arguments are defined as arguments that do not require support from other arguments to stand, while standard arguments must be supported by at least one *prima-facie* argument.

With the addition of the support relation new kinds of attacks can be observed that are based on the interaction between the direct attacks and the supports. These attacks were notated as ***complex attacks*** in [5]. With a detailed look on Example 3, it can be found that d supports a direct attacker of b_1. This can be considered as a negative interaction between d and b_1, which is weaker than the negative interaction between the direct attacker c_1 and b_1. Such a complex attack is called a ***supported attack***.

In comparison to that another argument b_3 can be assumed that says: "Channel 1 published a rectification that the resignation were fake-news". This argument would directly attack the argument d and in addition to that there is a negative interaction between the argument b_3 and c_1, because c_1 is supported by d. Such a complex attack is called a ***secondary attack***, in literature also notated as ***indirect attack***. Supported and secondary attacks were defined as follows in [6]:

Definition 10. *There is a* ***supported attack*** *from x to y if and only if there is a sequence $x_1R_1...R_{n-1}x_n$, $n \geq 3$, with $x_1 = x$, $x_n = y$, $\forall i = 1...n-2$, $R_i = R_{sup}$ and $R_{n-1} = R_{att}$.*

Definition 11. *There is a* ***secondary attack*** *from x to y if and only if there is a sequence $x_1R_1...R_{n-1}x_n$, $n \geq 3$, with $x_1 = x$, $x_n = y$, $R_1 = R_{att}$ and $\forall i = 2...n-1$, $R_i = R_{sup}$.*

These complex attacks were defined in 2005 for BAFs. Eight years later Cayrol and Lagasquie-Schiex introduced the three different interpretations of support. In [5] the authors analysed the impact of the introduced interpretations of support on the definition of supported and secondary attacks. This leads to specific complex attacks for each type of interpretation of support.

To illustrate the differences between the three possible interpretations of support and their complex attacks, the following examples should be considered:

Deductive Interpretation of support:

Example 4. *(Deductive support): This example was extracted from [5].*
At the last day of the season Liverpool leads the championship, followed by Chelsea. The following statements regarding to the title can be formulated:

- *if Liverpool wins the last match (lw), then Liverpool wins the Premier League (lc).*
- *if Liverpool loses with more than 4 goals in difference (ll), Chelsea wins the Premier League (cc).*
- *if the best defender of Liverpool is injured (bpi), Liverpool loses with more than 4 goals.*

Regarding to this example the argument lw supports the argument lc, so the acceptance of lw implies the acceptance of lc, and as a consequence the non-acceptance

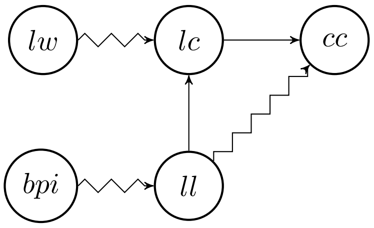

Figure 5: Bipolar argumentation framework from Example 4.

of lc implies the non-acceptance of lw. In addition to that the arguments ll is attacking the argument lc. Following that, the acceptance of ll implies the non-acceptance of lc and the non-acceptance of lw. This leads to the definition of a new type of complex attack, the ***mediated attack***.

Definition 12. *There is a **mediated attack** from x to y if and only if there is a sequence $x_1 R_{sup}$ ... $R_{sup} x_{n-1}$, and $x_n R_{att} x_{n-1}$, $n \geq 3$, with $x_1 = y$, $x_n = x$.*

In addition to mediated attacks, the deductive interpretation of support justifies the introduction of supported attacks from Definition 10. This can be observed in Example 4 in the following sense: The acceptance of lw implies the acceptance of lc and the acceptance of lc implies the non-acceptance of cc. This results in a supported attack from lw to cc. With a detailed look to Example 4, it can be seen that argument *bpi* supports the argument ll, so the acceptance of *bpi* implies the acceptance of ll. The acceptance of ll implies the non-acceptance of lc and the acceptance of lw implies the acceptance of lc. It follows, that the acceptance of *bpi* must imply the non-acceptance of lw. This condition is not included in the definition of supported or mediated attacks. For solving this problem Carol and Lagasquie-Schiex included supported attacks in addition to direct attacks, to the definition of a mediated attack. Following that, Definition 12 will be replaced by the definition of a new type of "mediated" attack, called the ***super-mediated attack***.

Definition 13. *There is a **super-mediated attack** from x to y if and only if there is a direct attack or a supported attack from x to z, and a support from y to z.*

Necessary Interpretation of support:

Example 5. *(Necessary support): This example was extracted from [5].*
The following dialogue between three agents can be observed:

- *Agent 1: The room is dark (rd), so I will light up the lamp (ll).*
- *Agent 2: The electric meter does work (ew).*
- *Agent 3: The electrician has detected a failure ($fail$).*
- *Agent 4: The electric meter does not work (enw).*

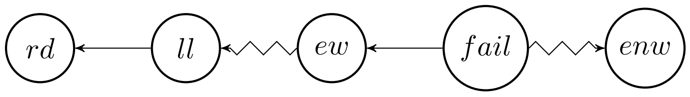

Figure 6: Bipolar argumentation framework from Example 5.

Regarding to this example the argument ew supports the argument ll, so for the acceptance of ll the acceptance of ew is necessary, following that the acceptance of ll implies the acceptance of ew. In Example 5, this relationship is obvious, because a working electric meter is necessary to light up a lamp. In the other way, if lighting up a lamp is successful it can be concluded, that the electric meter works. In the example $fail$ attacks ew, so the acceptance of $fail$ implies the non-acceptance of ew. Also $fail$ supports enw, so the acceptance of enw implies the acceptance of $fail$ and therefore the non-acceptance of ew. This situation leads to a new kind of complex attack, the ***extended attack***.

Definition 14. *There is an* ***extended attack*** *from x to y if and only if there is a sequence $x_1 R_{att} x_2 R_{sup}$ ... $R_{sup} x_n$, $n \geq 3$, with $x_1 = x$, $x_n = y$.*

As mentioned above, the acceptance of ew is necessary for the acceptance of ll. The acceptance of $fail$ implies the non-acceptance of ew and therefore the non-acceptance of ll. This constraint is equal to the secondary attack that was defined in Definition 11.

Evidential Interpretation of support: In [5], Cayrol and Lagasquie-Schiex presented a detailed analysis of complex attacks for the deductive and necessary interpretation of support. In contrast to that, a detailed introduction to the complex attacks for the evidential interpretation of support can be found in [1]. In BAFs with an evidential interpretation of support, it can be distinguished between standard arguments and *prima-facie arguments*. Only the attacks from *prima-facie* arguments or standard arguments that are supported by at least one *prima-facie* argument can be taken into account.

Example 6. *(Evidential support): The BAF from Figure 7 with the only prima-facie argument n, was found in [1].*

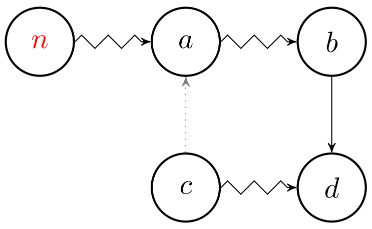

Figure 7: Bipolar argumentation framework with *prima-facie* argument n.

In this BAF the *prima-facie* argument n, supports the argument a and a supports argument b, so both arguments a and b are supported by evidence. The direct attack from c to a cannot be taken into account, which is illustrated by a gray dotted arrow, because c is not supported by evidence. The acceptance of a implies the acceptance of b, which implies the non-acceptance of d. This constraint is equal to a supported attack from a to d. When n would support c and not a, then the arguments c and d would be supported by evidence, and the attack from b to d could not be taken into account. In this case the acceptance of c implies the non-acceptance of a, which implies the non-acceptance of b and this constraint is equal to a secondary attack from c to b.

Conclusively, Table 1 that was extracted from [5], shows the specific complex attacks of each of the three different interpretations of support.

Deductive	Necessary	Evidential
Supported attack	Extended attack	Supported attack
Mediated attack	Secondary attack	Secondary attack

Table 1: Complex attacks for the deductive, necessary and evidential interpretation of support.

3.3 Acceptability

Another major issue with problems in BAFs is that the definitions for acceptability, presented by Dung in [8], has to be adapted. This work was done by Cayrol and Lagasquie-Schiex in [6] and the adapted definitions will be presented in the following. In the previous examples only single arguments that attack or support each others, were considered. In contrast to that the following definitions of ***set-attacks*** and ***set-supports*** consider sequences of supports and attacks.

Definition 15. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF, let $A \in \mathcal{A}$. A subset $S \subseteq \mathcal{A}$* ***set-attacks*** *x if and only if there exists a complex attack for x from at least one element of S.*

Definition 16. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF, let $A \in \mathcal{A}$. A subset $S \subseteq \mathcal{A}$* ***set-supports*** *x if and only if there exists a sequence of the form $x_1 R_1 ... R_{n-1} x_n, n \geq 2$, such that $\forall i = 1...n-1, R_i = R_{sup}$ with $x_n = x$ and $x_1 \in S$.*

The properties of ***conflict-freeness*** and ***collective defense*** are also important for acceptable sets of arguments in BAFs but were redefined by Cayrol and Lagasquie-Schiex:

Definition 17. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF. A subset $S \subseteq \mathcal{A}$ is* ***conflict-free*** *if and only if $\nexists x, y \in S$ such that $\{x\}$ set-attacks y.*

Definition 18. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF. A subset $S \subseteq \mathcal{A}$* ***defends collectively*** *x if $\forall y \in \mathcal{A}$, if and only if $\{y\}$ set-attacks x then $\exists z \in S$ such that $\{z\}$ set-attacks y.*

With these properties several semantics for acceptable sets were defined in [6]. For the model of AFs the property of conflict-freeness is a very important requirement for to the acceptability of sets of arguments. For BAFs this requirement is also very important but in contrast to AFs it is not strong enough for defining acceptable sets of arguments. A second requirement is necessary due to the addition of the support relation, that ensures that only sets are accepted that do not set-attack and set-support the same argument. This requirement results in the definition of ***safe*** sets:

Definition 19. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF. A subset $S \subseteq \mathcal{A}$ is **safe** if and only if $\nexists x \in \mathcal{A}$ such that S set-attacks x and either S set-supports x, or $x \in S$.*

For AFs the definition of ***stable*** extensions was presented in the previous section. For BAFs stable extensions also represent desirable sets, but the definition was adjusted in the following way:

Definition 20. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF. A subset $S \subseteq \mathcal{A}$ is a **stable** extension if and only if S is conflict-free and $\forall x \in \mathcal{A} \setminus S$, S set-attacks x.*

The next definition regards the ***closure*** of a set. Arguments that attack each others cannot be included in the same acceptable set of arguments, therefore acceptable sets cannot be closed for R_{att}. In contrast to that sets can be closed for R_{sup} with the following definition:

Definition 21. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF. A subset $S \subseteq \mathcal{A}$ is **closed** for R_{sup} if and only if $\forall x, y \in \mathcal{A}$ such that $x R_{sup} y$ and $x \in S$ then also $y \in S$.*

For AFs one definition was presented for admissible sets, by Dung in [8]. In contrast to that Cayrol and Lagasquie-Schiex presented three different definitions for admissibility in [6]. These three definitions differ regarding to their specificity. The most general type of admissibility is equal to Dung's definition and is therefore notated as ***d-admissible***, where "d" stands for in the sense of Dung.

Definition 22. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF. A subset $S \subseteq \mathcal{A}$ is **d-admissible** if and only if S is conflict-free and defends all its elements.*

This direct translation of Dung's definition does not include the requirement of safeness. By respecting this constraint the following definition of ***s-admissible***, where "s" stands for safe, was formulated.

Definition 23. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF. A subset $S \subseteq \mathcal{A}$ is **s-admissible** if and only if S is safe and defends all its elements.*

For the most specific type of admissibility, the property of closeness for R_{sup} can be required. This leads to the definition of ***c-admissible***, where "c" stands for closed.

Definition 24. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF. A subset $S \subseteq \mathcal{A}$ is **c-admissible** if and only if S is conflict-free, closed for R_{sup} and defends all its elements.*

With the following property Cayrol and Lagasquie-Schiex described the relation between the three different definitions of admissible sets:

Property 1. *Each c-admissible set is s-admissible, and each s-admissible set is d-admissible.*

With these three different types of admissibility the following definition of the ***preferred*** extensions, was presented in [6]:

Definition 25. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF. A subset $S \subseteq \mathcal{A}$ is a **d-preferred** (resp. **s-preferred, c-preferred**) extension if and only if S is maximal (w.r.t. set-inclusion) among the d-admissible (resp. s-admissible, c-admissible) subsets of $\mathcal{A}$.*

When considering Definition 7 regarding to complete extensions it can be found, that the property of admissibility is included. This leads to the fact, that regarding to the three different definitions of admissibility for BAFs, three different types of complete extensions can be differed. The most general type, the ***d-complete*** extensions, is using the direct translation of Dung's definition of admissibility.

Definition 26. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF. A subset $S \subseteq \mathcal{A}$ is a **d-complete** extension if and only if S is d-admissible and each argument that is defended by S, belongs to S.*

Taking into account the property of safeness for admissibility leads to the definition of ***s-complete*** extensions.

Definition 27. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF. A subset $S \subseteq \mathcal{A}$ is a **s-complete** extension if and only if S is s-admissible and each argument that is defended by S, belongs to S.*

The most specific definition, that belongs to ***c-complete*** extensions, is using the property of closeness for R_{sup}.

Definition 28. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF. A subset $S \subseteq \mathcal{A}$ is a **c-complete** extension if and only if S is c-admissible and each argument that is defended by S, belongs to S.*

Regarding to Definition 8 the minimal subset, with respect to set-inclusion, among all complete extensions, is the grounded extension. This results to the following definition of the ***grounded*** extension for BAFs.

Definition 29. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF. A subset $S \subseteq \mathcal{A}$ is the **d-grounded** (resp. **s-grounded, c-grounded**) extension if and only if S is minimal (w.r.t. set-inclusion) among the d-complete (resp. s-complete, c-complete) extensions of $(\mathcal{A}, R_{att}, R_{sup})$.*

3.3.1 Illustration of Acceptability

After the introducing of the semantics regarding the acceptability of BAFs, this subsection will illustrate the computation of these semantics and present the differences in determining the acceptable sets of arguments regarding the different interpretations of support.

Deductive Interpretation of support: For illustrating the computation of the semantics for BAFs, with an deductive interpretation of support, Figure 8 can be considered, that is based on a BAF shown in [6].

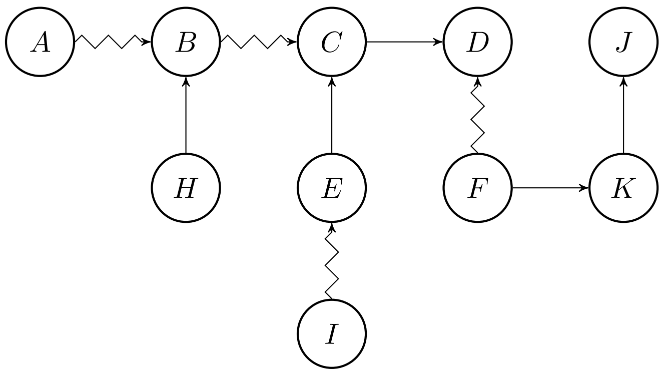

Figure 8: Bipolar argumentation framework.

The path $I - E - C$ corresponds to a supported attack from I to C. In contrast to this the path $I - E - C - B$ corresponds to a mediated attack from I to B.

Instead of listing all extensions of specific semantics, a more detailed look on a few examples will be presented. The sets $\{A, H\}$ and $\{A, D\}$ are not conflict-free, because A suffers under a mediated attack from H and D suffers under a supported attack from A. In contrast to that the sets $\{C, H\}$ and $\{D, E, H, I, K\}$ are conflict-free sets.

The notion of safe sets prevents the conflict, that an argument is set-supported and set-attacked at the same time by a subset of arguments. Following this property the set $\{A, E\}$ is not safe, because A supports B and B suffers under a mediated attack from E. In proceeding the set $\{B, F\}$ is also not safe, because D suffers a supported attack from B and is supported by F. On the contrary the sets $\{D, E, F, H, I\}$ and $\{D, F, H, I\}$ are safe sets. The set $\{D, F, H, I\}$ is not closed for R_{sup}, because I supports the argument E, which is not part of the set, in contrast to that the set $\{D, E, F, H, I\}$ is closed. But this set is not a stable extension, because it does not attack every argument, that is not part of the set, e.g. the argument J is not attacked. To make the set $\{D, E, F, H, I\}$ a stable extension the argument J have to be included. This results to the set $\{D, E, F, H, I, J\}$, which is the unique c-preferred extension and also the unique s-preferred and d-preferred extension. In addition to that the set $\{D, E, F, H, I, J\}$ is also the unique d-complete, s-complete and c-complete extensions and therefore the d-grounded, s-grounded and c-grounded extension.

Necessary Interpretation of support: After this detailed introduction of computing the semantics of BAFs with an deductive interpretation of support, this paragraph considers the necessary interpretation of support. The same BAF from Figure

8 will be considered with the difference that the support relation is interpreted necessarily.

The path $D - F - K$ corresponds to an extended attack from D to K. In contrast to that the path $H - B - C$ corresponds to a secondary attack from H to C. This leads to the fact, that the set $\{H, C\}$, which was conflict-free for the BAF with the deductive interpretation of support is not conflict-free in the sense of the necessary interpretation of support. In contrast to that the set $\{A, H\}$, that was not conflict-free with the deductive interpretation of support, is a conflict-free set with the necessary one.

The set of $\{A, D, E, F, H, I, J\}$ represents the unique stable extension of the BAF, this set is also the unique d-preferred extension, unique d-complete and d-grounded extension. In contrast to the presented unique stable extension for the BAF with the deductive interpretation of support, the unique stable extension for the necessary interpretation of support is not safe. This is due to the fact, that A supports the argument B, which is attacked by the arguments H, E and I. With $S = \{A, D, E, F, H, I, J\}$ as the unique stable extension, which is not safe, the s-preferred extensions are received by the subsets of S that are the maximal (w.r.t set-inclusion) s-admissible sets. This results to the sets $\{A, F, I, J\}$ and $\{D, E, F, H, I, J\}$ as the s-preferred extensions. The set $\{D, E, F, H, I, J\}$ is also the unique c-preferred extension. As mentioned above the set $\{A, D, E, F, H, I, J\}$ is the unique d-complete extension. This set is not safe and not close. This leads to the fact that the BAF from Figure 8 has no s-complete and no c-complete extension. This is due to the fact, that for a complete extension S, each argument that is defended by the S, has to be included in S and S has to be admissible. Such a set cannot be generated regarding to the properties of safeness and closure. Therefore no s-grounded and c-grounded extension can be received from the BAF from Figure 8 with the necessary interpretation of support.

Evidential Interpretation of support: For BAFs with the evidential interpretation of support, the computation of the semantics depends highly on the designation of the *prima-facie* arguments. The BAF from Figure 8 will be considered with the *prima-facie* arguments A, F, I. Following to the fact that the arguments H, K and J are not supported by evidence which leads to the consequence of not taking into account their attacks. This results to the BAF from Figure 9.

In this case the $\{A, B, C, H\}$ is a conflict-free set, which is neither a conflict-free set for the BAF in Figure 8 with the deductive, resp. necessary interpretation of support. The set $\{A, B, E, F, H, I, J\}$ is the unique stable extension, which is also the unique d-preferred, unique d-complete extension and therefore the d-grounded extension. Similar to the computation of the BAF with the necessary interpretation of support, no s- or c-complete extension and therefore no s- or c-grounded extension can be build. In contrast to that the sets $\{A, B, H, J\}$ and $\{E, F, H, I, J\}$ represent the s-preferred extensions. And conclusively the set $\{E, H, I, J\}$ is the unique c-preferred extension of the BAF from Figure 9.

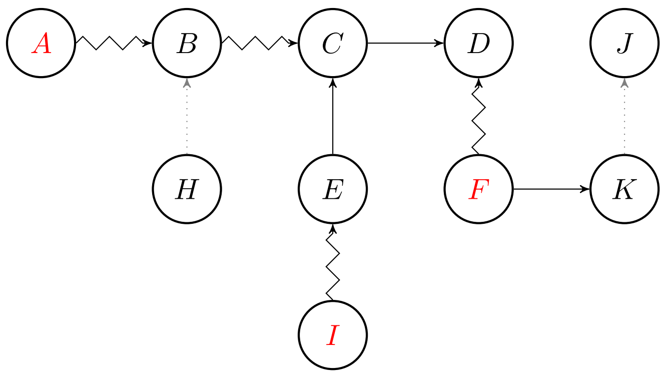

Figure 9: Bipolar argumentation framework with evidential interpretation of support and *prima-facie* arguments A, F, I.

In this section, the concept of BAFs was briefly introduced. This framework was received by adding a second possible kind of interaction between the arguments, the support relation, to the model of AFs. By adding the support relation, the strongest weakness of AFs could be overcome. In contrast to this, the addition is associated to the occurrence of new issues, regarding to the interpretation of the support relation and the acceptability. These issues were negotiated by presenting three different interpretation of support and by adjusting the definitions for acceptability, presented for AFs, by Cayrol and Lagasquie-Schiex in [5] respectively [6]. The adjusted definitions were illustrated by computing the semantics for the same argumentation problem with three different interpretations of support. The computation "by hand" is very complex and error-prone, this leads to the fact that determining the semantics computationally is of special interest. A very promising approach to do so, is the usage of the declarative programming paradigm of answer set programming, that will be introduced in the upcoming section.

4 Answer Set Programming

Argumentation systems can become very complex, with a high amount of arguments and an even higher number of connections between them. The preferred approach to compute the extensions of semantics for an AF or BAF is the usage of Boolean satisfiability problem [7] based solvers. Recent studies ([9, 13]) show that the usage of the declarative programming paradigm of Answer Set Programming (ASP), is a very promising approach. In this section the basic elements of the syntax and the semantics of ASP will be presented. In the last subsection the functionality of ASP is illustrated, by presenting an ASP encoding that can be used for solving the Traveling Salesperson Problem.

4.1 Syntax

The declarative programming paradigm of ASP is a special kind of logic programming and was first introduced, by Gelfond and Lifschitz, in [18]. Programs that are written in a logical programming language consists of the knowledge about a specific problem, formulated by using elements of formal logic. This knowledge will be used for solving the problem.

In formal logic two kinds of reasoning, ***monotonic*** and ***non-monotonic*** reasoning, can be differed. ASP is based on ***default logic*** that was introduced, by Reiter, in [24]. Default logic belongs to non-monotonic reasoning, that allows ***defeasible reasoning***. In contrast to non-monotonic reasoning, monotonic reasoning does not allow defeasible reasoning. In monotonic reasoning once proven information cannot be disproved by new information. In contrast to that in non-monotonic reasoning, new information can lead to a refutation of previous proven information. For non-monotonic reasoning, the knowledge about the problem is formulated by using ***default assumptions***. For ASP this default assumptions mean that initially something is false, until it is explicitly proven. This characteristic is very similar to human reasoning, because when humans don't have any information for the evidence of an argument, it is reasonable to assume that this argument is false.

In [13] the basic elements of the syntax of ASP were defined in the following way:

A ***term*** is a constant, a variable or a function. An ***atom*** is an expression $A(t_1, ..., t_n)$, where A is a predicate symbol of arity n and $t_1, ..., t_n$ are terms. A ***literal*** is an atom $A(t_1, ..., t_n)$ or a negated atom $\neg A(t_1, ..., t_n)$ where $\neg$ stands for the classic logical not. A ***disjunctive logic program***, denoted as π, is a set of rules r of the form:

$$L_1 \mid ... \mid L_k :- L_{k+1}, ..., L_m, not\ L_{m+1}, ..., not\ L_n. \qquad (1)$$

where $n \geq m \geq k \geq 0$ and $L_i (i = 1, ..., n)$ is a literal. The *head* of the rule r is the set $H(r) = \{L_1, ..., L_k\}$, and the ***body*** of r is the set $B(r) = \{L_{k+1}, ..., L_m, not\ L_{m+1}, ..., not\ L_n\}$. The ***positive body*** of r is defined as $B^+(r) = \{L_{k+1}, ..., L_m\}$ and the ***negative body*** is defined as $B^-(r) = \{L_{m+1}, ..., L_n\}$. The symbol *:-* stands for "if" and is dropped whenever $B(r) = \emptyset$, such a rule is called ***fact***. The symbol | stands for the logical or. A term, atom, rule or program is called ***ground*** if it does not contain any variables.

4.2 Semantics

Gelfond and Lifschitz has presented the ***stable model semantics*** in [22], which is the basis of the semantics for logic programs. The semantics can be used to determine, if a set S of ground atoms is a ***stable model***, also notated as an ***answer set***, of a given logic program.

For defining answer sets for grounded logic programs, the authors first considers programs that do not contain *not* and present the following definition:

Definition 30. *Let π be a grounded logic program that does not contain not and let Lit be the set of ground literals in the language of π. The* **answer set** *of π is the smallest subset S of Lit such that,*

1. *for any rule $L_0 \leftarrow L_1, ..., L_m$ from π, if $L_1, ..., L_m \in S$, then $L_0 \in S$.*
2. *if S contains a pair of complementary literals, then $S = Lit$.*

If the logic program π does not contain not, the second property is obvious, because without using not no complementary literals can be derived. The following examples illustrate the definition of answer sets by using a classical problem of formal logic.

Example 7. *The following programs should be considered:*

Program 1

```
1: bird(Tweety).
2: fly(X) :- bird(X).
```

Program 2

```
1: bird(Tweety).
2: fly(X) :- bird(X), healthy(X).
```

Program π_1 consists of the rules displayed in Program 1. The fact in line 1 expresses that Tweety is a bird. The rule in line 2 describes the intuitive assumption, that a bird is able to fly. Regarding to Definition 30 the program π_1 has the answer set $\{bird(Tweety), fly(Tweety)\}$. In contrast to π_1, the program π_2 consists of the rules shown in Program 2. The rule in line 2 expresses, that a healthy bird is able to fly. The program includes the fact, that Tweety is a bird, but no information about the healthiness of Tweety is provided. Therefore it cannot be derived that Tweety is able to fly, resulting to the answer set of $\{bird(Tweety)\}$ for π_2.

After presenting the definition of an answer set for a program without not, Gelfond and Lifschitz presented the following definition:

Definition 31. *Let π be a grounded logic program and let Lit be the set of ground literals in the language of π. For any set $S \subset Lit$, let π^S be the grounded logic program obtained from π by deleting*

1. *each rule that has a formula $not\ L$ in its body with $L \in S$, and*
2. *all formulas of the form $not\ L$ in the bodies of the remaining rules.*

The so derived program π^S is also notated as the ***Gelfond-Lifschitz reduct*** of π. This program does not contain not anymore so Definition 30 can be applied. For illustrating the construction of π^S, the following example should be considered.

Example 8. *By modifying Program 2 from the Example 7 the following program is received:*

Program 3

```
1: penguin(Tweety).
2: bird(X) :- penguin(X).
3: fly(X) :- bird(X), not penguin(X).
```

In this program the fact in line 1 specifies that Tweety is a penguin. Regarding to the rule in line 2, Tweety is also a bird. The rule in line 3 specifies the intuitive assumption, from Programs 1 and 2, by the requirement, that a bird can fly when it is not a penguin. Regarding to Definitions 30 and 31 π has the answer set $\{penguin(Tweety), bird(Tweety)\}$. This answer set can be derived when considering the following program that shows the Gelfond-Lifschitz reduct of Program 3:

Program 4

```
1: penguin(Tweety).
2: bird(X) :- penguin(X).
```

The rule from line 3 had to be removed regarding to the Definition 31 and this leads to the fact that $fly(Tweety)$ cannot be included in an answer set of π.

4.3 Functionality

After defining the basic elements of the syntax and the semantics of ASP, this subsection presents an overview of the functionality of programs in ASP. Figure 10, that was extracted from [19], visualizes the general workflow of solving a problem by using ASP.

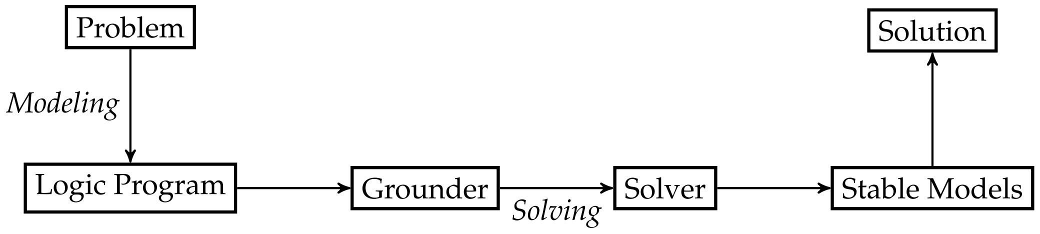

Figure 10: The workflow of ASP.

The first step of solving a problem in ASP is to express the knowledge about the problem and the problem setting as a logic program, this is notated as the ***modeling*** step. In Definition 30 an important requirement was that the logic program is grounded, following that it should not contain any variables. For the generation of a grounded program, a so-called ***grounder*** can be used, which is a program that replaces all variables in the logic program by variable-free terms. An example for a grounder is the system ***gringo***, that was described in [15]. In a next step a solver, like *clasp* [17], computes the stable models (answer sets), by using the stable model semantics, of the grounded logic program. One popular system for grounding and solving logic programs in ASP is ***clingo***, that was introduced in [16]. In clingo the grounder gringo and the solver clasp are included.

For a better understanding of each step from the ASP workflow, displayed in Figure 10, an example will be provided. This example was extracted from [14], where the authors present a method to solve the ***Traveling Salesperson Problem*** [2] (TSP) using ASP. The basis of the TSP is a list of cities with costs between them. In detail the TSP is an optimization problem, because the solution is the minimal tour (w.r.t. total costs), in which every city is exactly visited once. In other words the TSP is the problem of determining the shortest round-trip that starts and ends in the same city.

Gebser and Schaub presented the TSP instance that is shown in Figure 11.

In this figure each node belongs to one of the six german cities Berlin (B), Dresden (D), Hamburg (H), Leipzig (L), Potsdam (P) and Wolfsburg (W). City X is connected to city Y through a directed edge if there exists a direct train connection from X to Y. These edges are weighted by the duration in hours of the respective train trip. Considering Figure 10, the first step for computing the shortest round-trip using ASP, is to model the problem as a logic program. For this step a suitable encoding of the problem instance from Figure 11 has to be determined. The authors are presenting the encoding shown in Program 5.

They decided to use the two predicates *place* and *link* to model the cities, respectively the train connections between them. Whereas the *place* predicate contains a single argument that specifies the city, the *link* predicate contains three arguments. The first argument belongs to the city that is the source of the directed edge, the second argument represents the destination and the third argument is the duration

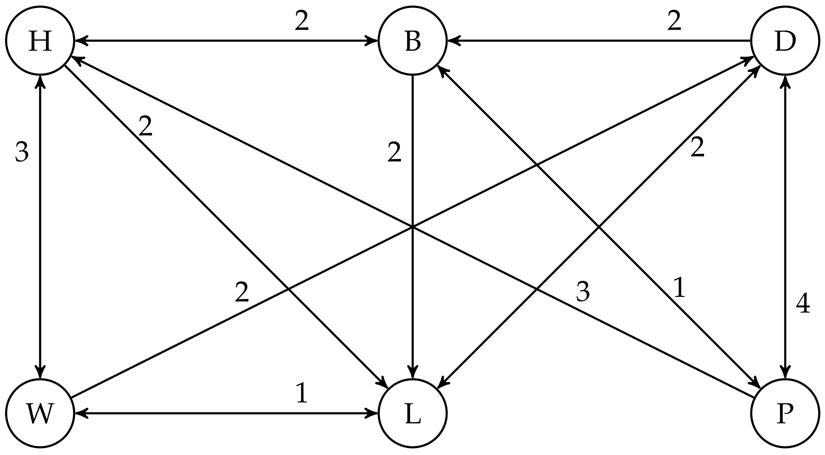

Figure 11: TSP of six German cities with distances between them.

Program 5

```
place(b).
place(d).
place(h).
place(l).
place(p).
place(w).
link(b,h,2). link(b,l,2). link(b,p,1).
link(d,b,2). link(d,l,2). link(d,p,4).
link(h,b,2). link(h,l,2). link(h,w,3).
link(l,d,2). link(l,w,1).
link(p,b,1). link(p,d,4). link(p,h,3).
link(w,d,2). link(w,h,3). link(w,l,1).
```

in hours of that train trip.

The more complex part of the modeling step is to express the knowledge about the problem in a logic program. By using the rules of the resulting encoding the shortest round-trip should be received. Gebser and Schaub decided to use an approach that is based on ***conceptual Generate-and-Test pattern*** ([11], [21], [23]). Regarding to this pattern the encoding should be divided into several parts. In [14] the authors have presented the encoding shown in Program 6 as the modeled problem knowledge.

This encoding is divided in six different sections, where the first section is denoted as *DOMAIN*. This section includes a single rule that assigns the starting point X, for the round-trip, in the predicate *start(X)*. Following to this rule the alphabetic smallest city is chosen, which is Berlin. The *DOMAIN* section is followed by the *GENERATE* section. When considering Program 5 it can be seen that every city is linked to at least two other cities by a train connection. The requirement that a city

Program 6

```
 1: %DOMAIN
 2: start(X) :- X = #min{Y : place(Y)}.
 3: %GENERATE
 4: {travel(X,Y) : link(X,Y,C)} = 1 :- place(X).
 5: %DEFINE
 6: visit(X) :- start(X).
 7: visit(Y) :- visit(X), travel(X,Y).
 8: %TEST
 9: :- place(Y), not visit(Y).
10: :- start(Y), #count{X : travel(X,Y)} < 1.
11: :- place(Y), #count{X : travel(X,Y)} > 1.
12: %OPTIMIZE
13: :~ travel(X, Y), link(X,Y,C). [C,X]
14: %DISPLAY
15: #show travel/2.
```

should be visited exactly once leads to the fact, that for each city X only one *link*, where the first argument is X, can be included in the round-trip. This constraint is expressed by the rule in line 4. This type of rule is also called a ***choice rule***, regarding to [25]. This rule contains the new predicate *travel(X,Y)*, that stands for a trip from city X to city Y. To assign this predicate the respective *link(X,Y,C)* is used to ensure, that only trips can be received that are based on an existing train connection. The third section is notated as the *DEFINE* section, that contains two rules. Both rules include the predicate *visit(X)* that is derived when city X was visited in the round-trip. These two rules provide the opportunity to check whether all cities are visited exactly ones. The rule in line 6 ensures that the starting point Berlin is visited. By using the predicate *travel(X,Y)*, where X is the last visited city and Y the destination of the next travel, all visited cities are collected during the trip by the rule in line 7.

By using the first three sections possible round-trips will be build. These possible solutions will be checked in the *TEST* section for validity. This section includes three special rules, where the respective head is empty, that are also called ***integrity constraints***. Leaving the head of a rule empty leads to a deletion of possible answer sets that do fulfill the constraint of the body. For example the integrity constraint in line 9 deletes all round-trips that did not visit all cities. One requirement for a round-trip is that it starts and ends in the same city. This requirement is covered by the integrity constraint in line 10. Therefore the round-trip should include a *travel(X,Y)* predicate, where the destination Y is the starting point Berlin. The number of cities that provides a returning to Berlin is collected by the #count aggregate. If this number is less than one, for a possible answer set, this set is deleted. The #count aggregate is also used for the integrity constraint in line 11, where it provides the number of visits for a single city. If this number is greater than one, this city has been visited multiple times and therefore this candidate has to be deleted.

After applying the integrity constraints from the *TEST* section only valid round-trips are remaining. The solution of the TSP is the round-trip with the minimal total duration among all valid candidates. This determination is located in the *OPTIMIZE* section. This section contains a so-called ***weak constraint*** that includes an empty head similar to integrity constraints. In contrast to integrity constraints, weak constraints are not eliminating candidates that fulfill the body of the rule. Weak constraints are assigning a weight, which is defined in square brackets, as a penalty to those candidates that fulfill the body. Because these weights are considered as penalties, one is interested in determining the candidate with the less weight. For receiving the round-trip with the total minimal duration, the duration C for each trip is used as the weight. The last section is notated as the *DISPLAY* section which includes the line #*show travel/2*. This line specifies that the *travel* predicate, with its two atoms, should be used as the output of the program.

The system clingo can be used to solve the respective TSP by using the Programs 5 and 6, that were developed by Gebser and Schauber in [14]. In Program 7 the resulting output can be found with the shortest round-trip of Berlin → Potsdam → Hamburg → Leipzip → Wolfsburg → Dresden → Berlin with a minimal duration of 11 hours.

Program 7

```
Answer: 1
travel(b,l) travel(l,w) travel(w,d)
travel(d,p) travel(p,h) travel(h,b)
Optimization: 14
Answer: 2
travel(b,p) travel(p,h) travel(h,w)
travel(w,l) travel(l,d) travel(d,b)
Optimization: 12
Answer: 3
travel(b,p) travel(p,h) travel(h,l)
travel(l,w) travel(w,d) travel(d,b)
Optimization: 11
OPTIMUM FOUND
```

This section provided a brief introduction to the concept and functionality of ASP. In literature different examples of systems for computing semantics of AFs and expansions of it (e.g. BAFs) can be found. In the upcoming section such an ASP based system and a Java based system will be presented.

5 Related Work

The development of implementations for computational models of argumentation is a wide research field. A platform for testing different implementations against each others is the International Competition on Computational Models of Argumentation (ICCMA), that was started in 2015 [27]. The competition focuses on computing different semantics for AFs. One participating system is the ASP based system ***ASPARTIX***. This system will be introduced briefly in the upcoming subsection. In addition to that the ***TweetyProject*** will be presented in the second subsection. This project is a collection of Java libraries, that provide an interface for working with different knowledge representations, such as argumentation.

5.1 ASPARTIX

In [10] the system ASPARTIX was introduced. The name ASPARTIX stands for Answer Set Programming Argumentation Reasoning Tool and was developed by the team of Uwe Egly from the technical University of Vienna. It is a tool for computing semantics of AFs and extensions of it (e.g. BAFs). A detailed introduction to the system and the development of the ASP based encodings can be found in [13]. The author Sarah Gaggl presented, in addition to ASP based encodings for AFs, encodings for computing the following semantics of BAFs:

- d-admissible
- s-admissible
- c-admissible
- d-preferred
- s-preferred
- c-preferred

For the development of these encodings the definitions regarding BAFs, presented by Cayrol and Lagasquie-Schiex in [6], served as the basis. Gaggl has published her work in 2009 and four years later Cayrol and Lagasquie-Schiex presented the definition of the three different interpretations of support for BAFs in [5]. In addition to the definition of the possible interpretations they have analysed the specific complex attacks. This leads to the fact that Gaggl has used a so-called ***abstract*** interpretation of support, with supported and secondary attacks as complex attacks. This leads to the fact, that semantics of BAFs with an deductive, necessary or evidential interpretation of support cannot be determined using the presented encodings.

In addition to that Gaggl has used a different definition of closed sets than Cayrol and Lagasquie-Schiex in [6]. In detail the following definition of closed sets can be found in [13]: "A set is closed under an operator if that operator returns a member of the set when evaluated on members of the set." In contrast to this, Definition 21 is equal to the presented one by Cayrol and Lagasquie-Schiex. To illustrate the differences between these two definitions the BAF from Figure 12 should be considered.

In this BAF the argument a supports the argument b, which supports the argument c. Regarding to the definition presented by Gaggl the sets $\{a, b, c\}$ and $\{\emptyset\}$ are the closed sets for R_{sup}. In contrast to that the sets $\{b, c\}$ and $\{c\}$ are no closed

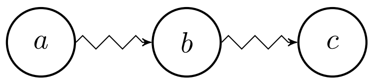

Figure 12: Bipolar argumentation framework.

sets for R_{sup}. From the definition that was used by Gaggl, it follows that whenever there is a support relation between two arguments, none of these arguments can be included without the other argument in a closed set.

In contrast to this, Definition 21 has the only requirement that whenever the supporter argument is included the supported argument has to be also included in a closed set, but not vice versa. This results to the closed sets $\{a, b, c\}, \{b, c\}, \{c\}$ and $\{\emptyset\}$ for the BAF from Figure 12, following the definition of Cayrol and Lagasquie-Schiex. The different definitions of closed sets can lead to different extensions of the c-admissible, c-preferred, c-complete and c-grounded semantics for the same BAFs.

In this subsection the system ASPARTIX was introduced, which can be used for computing semantics for BAFs, as defined in [6], with the only difference regarding the definition of closure. In addition to that the system is not able to compute desirable semantics for BAFs with an deductive, necessary or evidential interpretation of support. In contrast to that the TweetyProject, that will be presented in the next subsection, includes Java based implementations that can be used for BAFs with these properties.

5.2 TweetyProject

As mentioned in the begin the TweetyProject [26] is a project that contains a collection of Java libraries for implementing approaches to different areas of AI, such as computational models of argumentation. In detail the collection contains reasoners for problems in BAFs with an deductive, necessary or evidential interpretation of support. Therefore these reasoners can be used for testing the efficiency of developed ASP based encodings for the same tasks. The usage of ASP for computing semantics of BAFs should lead to a reduction of running time compared to a Java based system. This is due to the fact, that in ASP elements of formal logic can be used directly.

In this section two important reasoning tools, for this work, were presented. The system ASPARTIX shows the potential of using ASP for computing the semantics of BAFs. Unfortunately this system cannot be used for BAFs that are based on latest research. In contrast to that the reasoners that are included in the TweetyProject are able to do so. These reasoners are based on the object-orientated programming language of Java, which leads to a high running time for complex BAFs. The aim of this work is to link the advantages of both systems. On the one hand the advantages of using ASP, instead of Java, for computing semantics of BAFs. On the other hand the developed system should be able to compute the semantics for BAFs with an deductive, necessary or evidential interpretation of support. The developed system will be tested against the reasoners from the TweetyProject for efficiency and performance. In the next section the developed ASP based encodings will be presented and introduced in detail.

6 ASP-Encodings for Bipolar Argumentation Frameworks

This section is the main contribution of this work, in which the ASP based encodings will be introduced. The developed encodings should be able to compute the following semantics of BAFs with the deductive, necessary or evidential interpretation of support:

- d-admissible
- d-preferred
- d-complete
- d-grounded
- stable
- s-admissible
- s-preferred
- s-complete
- s-grounded
- c-admissible
- c-preferred
- c-complete
- c-grounded

The encodings presented by Gaggl in [13], are able to compute the semantics of BAFs that where defined in [6]. In addition to that, Gaggl also presented ASP based encodings for computing the grounded extension, complete and stable extensions for AFs. All presented encodings in this section are based on the encodings that were presented by Gaggl in [13]. Those encodings were adapted to obtain the possibility of computing the semantics of BAFs with an deductive, necessary or evidential interpretation of support.

6.1 Modelling Bipolar Argumentation Frameworks

Before the semantics of a BAF can be computed, the respective BAF has to be modeled as a logic program. BAFs with an deductive or necessary interpretation of support can be modeled in the same way. To be able to use existing test cases the developed system uses the same way of modeling than ASPARTIX, which is defined as follows:

Definition 32. *Let* $(\mathcal{A}, R_{att}, R_{sup})$ *be a BAF with the deductive or necessary interpretation of support and let* π_{BAF} *be the modeled BAF.* π_{BAF} *consists of the following facts:* $\pi_{BAF} = \{arg(a)|a \in \mathcal{A}\} \cup \{att(a,b) \in R_{att}\} \cup \{sup(a,b) \in R_{sup}\}$.

For BAFs with the evidential interpretation of support, a slightly different way of modeling is necessary, because two possible types of arguments can occur. *Prima-facie* arguments will be notated as *p_arg* and for standard arguments the syntax from Definition 32 can be reused.

Definition 33. *Let* $(\mathcal{A}, R_{att}, R_{sup})$ *be a BAF with the evidential interpretation of support and let* π_{BAF} *be the modeled BAF.* π_{BAF} *consists of the following facts:* $\pi_{BAF} = \{p\_arg(a)|a \in \mathcal{A}\} \cup \{arg(a)|a \in \mathcal{A}\} \cup \{att(a,b) \in R_{att}\} \cup \{sup(a,b) \in R_{sup}\}$.

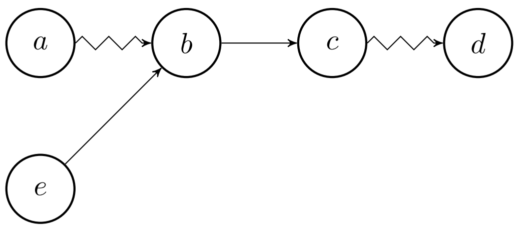

Figure 13: Bipolar argumentation framework.

To illustrate the way of modeling, the BAF in Figure 13 should be considered. Regarding to Definition 32 Program 8 shows the modeled BAF with an deductive or necessary interpretation of support.

Program 8

```
1: arg(a).
2: arg(b).
3: arg(c).
4: arg(d).
5: arg(e).
6: att(b, c).
7: att(e, b).
8: sup(a, b).
9: sup(c, d).
```

When the BAF from Figure 13 is considered with an evidential interpretation of support, using a and c as the *prima-facie* arguments, Program 9 shows the respective logic program.

Program 9

```
1: p_arg(a).
2: arg(b).
3: p_arg(c).
4: arg(d).
5: arg(e).
6: att(b, c).
7: att(e, b).
8: sup(a, b).
9: sup(c, d).
```

After defining the method of modeling BAFs as a logic program the following subsection will present some basic encodings that are used for computing different semantics.

6.2 Basic Concepts

The following steps and encodings are used for mainly all semantics:

- Generate all subsets of $\mathcal{A}$,
- Specify the output,
- Support and attack relation,
- Property of defense.

Generate all subsets of $\mathcal{A}$: For computing the extensions of specific semantics, the following method will be used: In a first step, all possible subsets of $\mathcal{A}$ will be build and serve as a possible candidate. In a second step all candidates are checked regarding to the specific requirements of the desired semantics. With the following encoding all possible subsets of arguments will be received. Program 10 shows the respective encoding, please note, that lines 4 and 5 are missing in the encoding for BAFs with an deductive or necessary interpretation of support.

Program 10

```
1: %Generate all subsets S of $\mathcal{A}$
2: in(X) :- arg(X), not out(X).
3: out(X) :- arg(X), not in(X).
4: in(X) :- p_arg(X), not out(X). %evidential
5: out(X) :- p_arg(X), not in(X). %evidential
```

Output: Following the method for computing a specific semantic, that was introduced in the previous paragraph, one is interested in the subsets that remain after the deletion step. With the encoding from Program 11 all subsets that fulfill the requirement for the desired semantic will be displayed.

Program 11

```
%Output
#show in/1.
```

Support Relation: One possible interaction between the arguments in a BAF is the support relation. This relation was defined as a binary relation between two arguments. In a BAF these binary relations often form a sequence of supports. The aim of the support encoding is that each sequence with n arguments $xR_{sup}yR_{sup}z...R_{sup}n$, is transformed to all binary support relations $sup(x,y)$, $sup(x,z)$, ..., $sup(x,n)$, $sup(y,z)$, ..., $sup(n-1,n)$. In addition to that the Definition 16 of set-support has to be implemented. The complete encoding for the support relation can be found in Program 12, that can be used for all interpretations of support.

Program 12

```
%Support
sup(X,Z) :- sup(X,Y), sup(Y,Z).
support(X,Y) :- sup(X,Y).
supported(X) :- support(Y,X), in(Y). %set-support
```

Attack Relation for Deductive Support: The second possible interaction between arguments in a BAF is the attack relation. By the interaction of supports and attacks complex attacks can occur in BAFs, that are specific for the interpretation of support. The following encoding computes all binary attacks in a BAF with the deductive interpretation of support. Recalling Table 1, the complex attacks for the deductive interpretation of support consist of supported attacks (Definition 10) and mediated attacks (Definition 13).

Program 13

```
%Attacks for deductive
attack(X,Y) :- att(X,Y). %direct attack
attack(X,Y) :- att(Z,Y), support(X,Z). %supported attack
attack(X,Y) :- att(X,Z), support(Y,Z). %mediated attack
attack(X,Y) :- support(X,Z), attack(Z,Y). %super-mediated attack
attacked(X) :- attack(Y,X), in(Y). %set-attack
```

Attack Relation for Necessary Support: In contrast to the complex attacks in a BAF with an deductive interpretation of support, the complex attacks for the necessary interpretation of support consist of extended attacks (Definition 14) and secondary attacks (Definition 11). The appropriate encoding can be found in Program 14.

Program 14

```
%Attacks for necessary
attack(X,Y) :- att(X,Y). %direct attack
attack(X,Y) :- att(Z,Y), support(Z,X). %extended attack
attack(X,Y) :- att(X,Z), support(Z,Y). %secondary attack
attacked(X) :- attack(Y,X), in(Y). %set-attack
```

Attack Relation for Evidential Support: Regarding to Table 1 the complex attacks of BAFs with the evidential interpretation of support consist of supported and secondary attacks. The encoding of these attacks can be found in Program 15.

In contrast to the attacks in BAFs with an deductive or necessary interpretation of support, not all attacks in a BAF with the evidential interpretation of support can be

Program 15

```
1: %Attacks for evidential
2: attack(X,Y) :- att(X,Y), e_supported(X). %direct attack
3: attack(X,Y) :- att(Z,Y), support(X,Z), e_supported(X). %supported attack
4: attack(X,Y) :- att(X,Z), support(Z,Y), e_supported(X). %secondary attack
5: attacked(X) :- attack(Y,X), in(Y). %set-attack
6:
7: %E-supported arguments
8: e_supported(X) :- p_arg(X).
9: e_supported(X) :- arg(X), p_arg(Y), support(Y,X).
```

taken into account. Only attacks from arguments that are supported by evidence can be considered. The property of evidential support for a single argument is encoded by the rules in line 8 and 9 of Program 15. Line 8 contains the rule, that *prima-facie* arguments are supported by evidence straightaway. The rule in line 9 ensures that standard arguments which are supported by a *prima-facie* argument, directly or transitively, also fulfill the property of supported by evidence. The requirement that only attacks from arguments can be taken into account, that are supported by evidence, is implemented by the rules in lines 2, 3 and 4. By adding the predicate $e\_supported(X)$ to these rules it is ensured that only attacks from x to y are taken into account, when x is an argument that is supported by evidence.

Collective Defense: The property of collective defense is very important for different semantics. Considering the definition of collective defense presented by Dung in [8] and from Definition 18, a set of arguments S defends an argument if and only if S attacks all attackers of this argument. Instead of implementing the property of defense in a direct way, the following encoding can be used to check if an argument is not defended. This leads to the advantage of using the predicate $not\_defended(X)$ directly for deleting answer sets with undefended arguments. The encoding can be found in Program 16 and can be used for all types of interpretation of support.

Program 16

```
%Not defended Arguments
not_defended(X) :- attack(Y,X), not attacked(Y).
```

Regarding to this encoding the predicate $not\_defended(X)$ is derived if x is attacked by the argument y and y is not set-attacked by the current assignment S.

In this subsection basic encodings were presented. These basic encodings are part of the specific encodings for mainly all desired semantics that will be introduced in the next subsection. Please note that these encodings will not be showing completely. The included basic encodings will just be mentioned and not shown again.

6.3 Admissible Sets

Regarding to Definitions 22 - 24, all three types of admissible sets have in common, that they have to defend all their elements collectively (Definition 18). In addition to that d-admissible and c-admissible sets need to be conflict-free in the sense of Definition 17. All encodings for computing admissible sets include the Programs 10 - 12, 16 and one of Programs 13 - 15 for the specific complex attack. The encoding for computing d-admissible sets, that is based on the most general type of admissibility, can be found in Program 17.

Program 17 d_adm.lp

```
1: %d-admissible
2: :- in(X), in(Y), attack(X,Y). %conflict-free
3: :- in(X), not_defended(X). %collectively defense
```

With the integrity constraint in line 2, all subset are deleted, that violate the requirement of conflict-freeness. In other words, all subsets are deleted, that contain two arguments x and y, where x attacks y. The integrity constraint in line 3, underlines the advantage of defining the predicate $not\_defended(X)$ from Program 16. This property can be used directly to delete of subsets that contain an argument x, that is not defended by this subset.

Regarding to Defintion 23 s-admissible sets do not have to fulfill the requirement of conflict-freeness. In contrast to that s-admissible sets have to fulfill the requirement of safe in the sense of Definition 19. A subset S is safe, if and only if there is no argument x in the BAF, that is set-supported and set-attacked at the same time from S or that is set-attacked from S and belongs to S. Obviously the second constraint also leads to the requirement of conflict-freeness. In Program 18 the encoding for computing the s-admissible sets of a given BAF can be found.

Program 18 s_adm.lp

```
1: %s-admissible
2: :- in(X), not_defended(X). %collectively defense
3: :- attacked(X), supported(X). %safe
4: :- attacked(X), in(X). %safe
```

The integrity constraint in line 2, corresponds to the requirement of collective defense and was also used for the encoding of d-admissible sets. The integrity constraint in line 3 regards the constraint, that a subset S should not set-attack and set-support the same argument, when considering safe sets. The integrity constraint in line 4 deletes all subsets that set-attack an argument x and includes it at the same time.

The most specific type of admissibility corresponds to c-admissibility, where "c" stands for close. Regarding to Definition 21, a subset S is a closed set, if and only if S contains an argument x, that supports argument y, then y has to be also included. As mentioned in the previous section, the encodings presented by Gaggl in [13], are based on a different definition of closure. Regarding to the definition, used by Gaggl, a closed set has to include the supported argument, when the supporter argument is included and also vice versa. If the supported argument is included than also the supporter argument has to be included. The developed encodings of this work should be based on the work of Cayrol and Lagasquie-Schiex from [5]. Therefore the property of closure will be implemented regarding to Definition 21. The fact that a different, less specific, definition of closed sets is used, can lead to different c-admissible sets in comparison to ASPARTIX. In detail, with the encoding from Program 19 more c-admissible sets are received compared to the original encoding provided by Gaggl in [13].

Program 19 c_adm.lp

```
1: %c-admissible
2: :- in(X), in(Y), attack(X,Y). %conflict-free
3: :- in(X), not_defended(X). %collectively defense
4: :- in(X), out(Y), support(X,Y). %closed
```

It is obvious that lines 2 and 3 are equal to the same lines of the encoding for d-admissible sets from Program 17. The integrity constraint in line 4, deletes all subsets that violate the constraint for closure. In detail all subsets S are deleted, that contain the argument x and do not contain the argument y, when there is a binary support relation $support(x, y)$, that means x supports y.

6.4 Preferred Extensions

As mentioned before one is not only interested in receiving all admissible sets of a given BAF but especially in computing the maximal (w.r.t. set-inclusion) sets among all admissible sets. Regarding to Definition 25 these sets where notated as the preferred extensions. In this subsection the encoding for computing the d-, s- or c-preferred extensions of a given BAF will be presented and introduced. Please note that the respective encodings include the encodings from Programs 10 - 12, 16 and one of Programs 13 - 15 for the specific complex attack.

For computing the preferred extensions, Gaggl has used the so-called ***saturation method***, that was briefly described in [12]. The saturation method can be used to check if all solution candidates fulfill a certain requirement. This technique can be used to determine valid or non-valid candidates regarding a specific property. The determination of the preferred extensions of a given BAF can be formulated as follows: Given all admissible sets of a BAF, find the sets that are maximal with respect to set-inclusion.

The saturation method can be used in the following sense to do so: Let S be the set that contains all admissible sets of a given BAF. Each admissible set $s \in S$ is a possible solution candidate for a preferred extension of the BAF. The set s is a preferred extension, if no set $t \subseteq \mathcal{A}$ can be formed, so that $s \subset t$ and t is admissible. This leads to the following procedure: For each candidate s a proper superset $t \subseteq \mathcal{A}$, so that t contains at least one element that is not in s, will be searched and tested regarding the requirements of admissibility. If such a set can be found, s is not a preferred extension and therefore no answer set of the encoding. This procedure will be repeated for all $s \in S$, so that the remaining answer sets are the preferred extensions of the BAF.

The encoding to compute the d-preferred extensions is displayed in Program 20. Please note that lines 15 and 17 are only present in the encoding file for a BAF with an evidential interpretation of support.

Program 20 d_pref.lp

```
 1: %d-admissible
 2: :- in(X), in(Y), attack(X,Y). %conflict-free
 3: :- in(X), not_defended(X). %collectively defense
 4: %Guess all subsets T.
 5: inN(X) | outN(X) :- out(X).
 6: inN(X) :- in(X).
 7: %Spoil if t = s.
 8: spoil :- eq.
 9: %Spoil if t is not conflict-free.
10: spoil :- inN(X), inN(Y), attack(X,Y). %conflict-free
11: %Spoil if t does not defend all its elements
12: spoil :- inN(X), outN(Y), attack(Y,X), unattacked(Y). %collective defense
13: %Ensure that spoil is an answer set of T.
14: inN(X) :- spoil, arg(X).
15: inN(X) :- spoil, p_arg(X). %evidential
16: outN(X) :- spoil, arg(X).
17: outN(X) :- spoil, p_arg(X). %evidential
18: %If at least one t ∈ T not derived spoil, s is not a preferred extension
19: :- not spoil.
```

This encoding will be introduced briefly in the following. For a better understanding of the encoding, the BAF from Figure 14 will be serving as an example, that has the only d-preferred extension $\{a, c\}$. This BAF does only contain direct attacks, so it can also be interpreted as an AF.

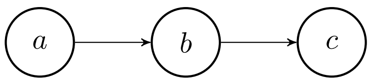

Figure 14: Bipolar argumentation framework.

For the introduction of the encoding the two predicates *eq* (line 8) and *unattacked(X)* (line 12) remain undefined until the respective lines will be reached.

The integrity constraints in lines 2 and 3 belong to Program 17 and ensure that S, as the first generation of all subsets of $\mathcal{A}$, only include d-admissible sets. Considering the BAF from Figure 14, S does consists of the d-admissible sets $\{\emptyset\}, \{a\}, \{a, c\}$. Each of these sets is a potential candidate for a preferred extension of the BAF. For each candidate $s \in S$ the rules in line 5 and 6 are building new sets t, that will be added to T, so that the relation $s \subseteq T$ holds. By rule in line 6 a new set t that is equal to s will be formed. By using the rule in line 5 all possible sets t out of the respective *in(X)* and *out(X)* predicates from s will be created. Therefore the rules in lines 5 and 6 would lead to the set $T = \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$ when $s = \{a\}$. The rule in line 8 contains the undefined predicate *eq*. The respective encoding, presented by Gaggl in [13], of this predicate can be found in Program 21.

Program 21

```
eq_upto(Y) :- inf(Y), in(Y), inN(Y).
eq_upto(Y) :- inf(Y), out(Y), outN(Y).
eq_upto(Y) :- succ(Z,Y), in(Y), inN(Y), eq_upto(Z).
eq_upto(Y) :- succ(Z,Y), out(Y), outN(Y), eq_upto(Z).
eq :- supremum(Y), eq_upto(Y).
```

This predicate should be used to compare two sets regarding equality. To test the elements of two sets regarding equality, they have to be ordered in the same way. For this reason an order on the arguments of a BAF has to be defined. Gaggl presented the encoding from Program 22 that is able to order the arguments by the "<"-relationship. Please note that the lines with an *%evidential* comment are only present in the encoding file for a BAF with the evidential interpretation of support and are missing in the encoding file for a BAF with the deductive or necessary interpretation of support.

Program 22

```
lt(X,Y) :- arg(X),arg(Y), X<Y.
lt(X,Y) :- p_arg(X),arg(Y), X<Y. %evidential
lt(X,Y) :- arg(X),p_arg(Y), X<Y. %evidential
lt(X,Y) :- p_arg(X),p_arg(Y), X<Y. %evidential
nsucc(X,Z) :- lt(X,Y), lt(Y,Z).
succ(X,Y) :- lt(X,Y), not nsucc(X,Y).
ninf(X) :- lt(Y,X).
nsup(X) :- lt(X,Y).
inf(X) :- not ninf(X), arg(X).
inf(X) :- not ninf(X), p_arg(X). %evidential
supremum(X) :- not nsup(X), arg(X).
supremum(X) :- not nsup(X), p_arg(X). %evidential
```

Regarding to the BAF from Figure 14 the arguments are ordered alphabetical as follows: $a < b < c$. For this BAF the single answer set $\{inf(a),\ succ(a, b),\ succ(b, c),\ supremum(c)\}$ will be received, without the other atoms regarding to the predicates *nsucc, ninf* and *nsup*. This order will be used to determine if two sets are equal. By the predicate *eq_upto(Y)* the elements, starting from the respective infimum of the set to its supremum, will be compared element-wise. If two sets are equal the *eq* predicate will be derived. By the rule in line 8 of Program 20 the candidate s will be checked against all sets $t \in T$ regarding equality. When a set $t \in T$ is equal to the set s, the predicate *spoil* will be derived. This predicate will also be derived, when a set $t \in T$ is not conflict-free, by the rule in line 10.

The rule in line 12 consists of the undefined predicate *unattacked(X)*. The respective encoding, presented by Gaggl in [13], of this predicate can be found in Program 23.

Program 23

```
unattacked_upto(X,Y) :- inf(Y), outN(X), outN(Y).
unattacked_upto(X,Y) :- inf(Y), outN(X), not attack(Y,X).
unattacked_upto(X,Y) :- succ(Z,Y), unattacked_upto(X,Z), outN(Y).
unattacked_upto(X,Y) :- succ(Z,Y), unattacked_upto(X,Z), not attack(Y,X).
unattacked(X) :- supremum(Y), unattacked_upto(X,Y).
```

The predicate checks if the not included arguments of a given set are unattacked by the set. Therefore each argument starting from the infimum to the supremum of the not included arguments, will be checked by using the order of the arguments and the predicate *unattacked_upto(X,Y)*. If the argument x is not attacked by any argument of a given set, than the predicate *unattacked(X)* will be derived. This predicate is used in the rule in line 12 of Program 20, to tests if a given set defends collectively all its elements.

This leads to the fact that the rules in lines 8, 10 and 12 will derive the predicate *spoil*, if a given set $t \in T$ is equal to s or violates at least one of the requirements regarding d-admissibility. Regarding to the set $s = \{a\}$ the respective set $T = \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$, the set $t_1 = \{a\}$ will be derive the predicate *spoil*, because it is equal to s. In addition to that the sets $t_2 = \{a, b\}$ and $t_4 = \{a, b, c\}$ will derive spoil, because they violate the requirements regarding d-admissibility. Only the set $t_3 = \{a, c\}$ will not derive the predicate *spoil*. The rules in lines 14 to 16 ensure that each $t \in T$, that has derived the predicate *spoil* is an answer set of T. If not all $t \in T$ derived the predicate *spoil*, this means, that there exists at least one $t \in T$, so that $s \subset t$ and t is d-admissible. In other words, s is not the d-preferred extension, because another set exists, that is greater (w.r.t. set-inlcusion) and fulfill the requirements regarding d-admissibility. For the sets $s = \{a\}$ the respective set $T = \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$, the set $t_3 = \{a, c\}$ would not derive the predicate *spoil*, which leads to the fact that the set s is not an d-preferred extension of the BAF from Figure 14.

When considering the set $s = \{a, c\}$ of this BAF, the set $T = \{\{a, c\}, \{a, b, c\}\}$ will be constructed by the rules in line 5 and 6 of Program 20. The set $t_1 = \{a, c\}$ will derive *spoil* because of the rule in line 8. The set $t_2 = \{a, b, c\}$ will derive the predicate *spoil* regarding to the rule in line 10. This leads to the fact, that all sets $t \in T$ derived spoil and s is a d-preferred extension of the BAF.

For computing s-preferred extensions (resp. c-preferred extensions) the encodings from Programs 21, 22 and 23 can be reused. Only Program 20 has to be adapted to the specific requirement regarding safeness (resp. closure). Program 24 shows the encoding for computing s-preferred extensions.

Program 24 s_pref.lp

```
 1: %s-admissible
 2: :- in(X), not_defended(X). %collectively defense
 3: :- attacked(X), supported(X). %safe
 4: :- attacked(X), in(X). %safe
 5: %Guess all subsets T.
 6: inN(X) | outN(X) :- out(X).
 7: inN(X) :- in(X).
 8: %Spoil if T = S.
 9: spoil :- eq.
10: %Set-support.
11: supportedN(X) :- support(Y,X), inN(Y). %set-support
12: %Set-attack.
13: attackedN(X) :- attack(Y,X), inN(Y). %set-attack
14: %Spoil if t is not safe.
15: spoil :- supportedN(X), attackedN(X). %safe
16: spoil :- attackedN(X), inN(X). %safe
17: %Spoil if t does not defend all its elements
18: spoil :- inN(X), outN(Y), attack(Y,X), unattacked(Y). %collective defense
19: %Ensure that spoil is an answer set of T.
20: inN(X) :- spoil, arg(X).
21: inN(X) :- spoil, p_arg(X). %evidential
22: outN(X) :- spoil, arg(X).
23: outN(X) :- spoil, p_arg(X). %evidential
24: %If at least one t ∈ T not derived spoil, s is not a preferred extension
25: :- not spoil.
```

Please note that lines 21 and 23 are only present in the encoding file for BAFs with the evidential interpretation of support. Commonly to the encoding for computing d-preferred extensions from Program 20, all s-admissible sets will be determined in a first step. For spoiling the subsets $t \in T$ that are not safe, it has to be checked if a given t supports and attacks the same argument. The predicates *supported(X)* and *attacked(X)* that were implemented in the encodings for the specific attacks and in the encoding of the support relation in Program 12, cannot be reused. This is due to the fact, that they are based on S (via in(X) and out(X)). So in lines 11 and 13 the respective predicates *supportedN(X)* and *attackedN(X)* will be defined for T (via inN(X) and outN(X)). These predicates will be used to derive *spoil* for the subsets $t \in T$, that do not fulfill the requirement of safe, by the rules in lines 15 and 16.

Instead of showing the encoding for computing c-preferred extensions, this encoding will be describe formally. The encoding is very similar to the encoding for the d-preferred extensions shown in Program 20. In contrast to that after the two integrity constraints in lines 2 and 3, a third integrity constraint with

$$: -in(X), out(Y), support(X, Y).\%closed \tag{2}$$

is added, to delete all subsets from S, that violate the requirement of closure. In addition to that the rule

$$spoil : -support(X, Y), outN(Y), inN(X).\%closed \tag{3}$$

has to be added, for spoiling all subsets $t \in T$ that do not fulfill the requirement of closure.

In this subsection the encodings, that were developed by Gaggl in [13], for computing the preferred extensions, were introduced. In the upcoming subsection the encoding for the complete extensions will be presented.

6.5 Complete Extensions

All encodings for computing the respective complete extensions include the encodings from Programs 10 - 12, 16 and one of Programs 13 - 15 for the specific complex attack. Regarding to Definitions 26, 27 and 28 a subset S is a complete extension, if it fulfill the requirements regarding the specific type of admissibility and if each argument that is defended by S belongs to S. Gaggl has presented an encoding for computing d-complete extensions in [13]. For this encodings she has used the predicate *not_defended(X)* from Program 16. By using the concept of ***double negation***, the defended arguments under a current assignment can be received. The presented encoding for computing the d-complete extensions by Gaggl, was extended by encodings regarding s- respectively c-complete extensions. Program 25 shows all three encodings for the desired complete extensions.

The respective complete extensions will be computed in the following way: In a first step all subsets $s \in S$ will be deleted that violate the requirements for the respective type of admissibility. By the integrity constraint in the last line of each encoding all subsets that do defend argument x but do not contain it will be deleted.

To sum up this subsection, all three types of complete extensions can be computed by using the encodings shown in Program 25. In the upcoming subsection the encodings for computing the respective grounded extension will be introduced.

Program 25 d_com.lp & s_com.lp & c_com.lp

```
%d-complete
:- in(X), in(Y), attack(X,Y). %conflict-free
:- in(X), not_defended(X). %collectively defense
:- out(X), not not_defended(X). %complete

%s-complete
:- in(X), not_defended(X). %collectively defense
:- attacked(X), supported(X). %safe
:- attacked(X), in(X). %safe
:- out(X), not not_defended(X). %complete

%c-complete
:- in(X), in(Y), attack(X,Y). %conflict-free
:- in(X), not_defended(X). %collectively defense
:- in(X), out(Y), support(X,Y). %closed
:- out(X), not not_defended(X). %complete
```

6.6 Grounded Extension

Regarding to the Definition 29 the d-grounded (resp. s- or c-grounded) extension of a BAF is the minimal subset (w.r.t set-inclusion) among the d-complete (resp. s- or c-complete) extensions. For computing the respective grounded extension, the saturation method could be used. But instead of using this method, Gaggl has used a more intuitive method to compute the d-grounded extension. Similar to the last subsection, the encoding for computing the d-grounded extension was extended by encodings regarding the s- respectively c-grounded extension. For the illustration of this method the BAF in Figure 15 can be considered, with an deductive interpretation of support.

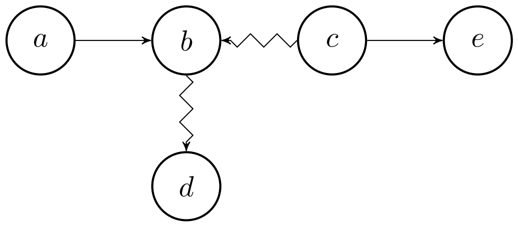

Figure 15: Bipolar argumentation framework.

To generate the complete extensions of this BAF, all attacks should be determined. The following attack relations will be received: $attack(a, b)$, $attack(c, e)$ and $attack(a, c)$. This shows, that the BAF from Figure 15 does contain the two arguments a and d that are not attacked by any argument of the BAF. Therefore these arguments has to be included in every complete extension of the BAF. This constraint leads to the fact, that these two arguments also have to be included in the grounded extension. In a next step it can be checked if one of these arguments defend another argument. Following that it can be found that a defends e from the attack of c. Therefore, argument e has to be also included in every complete extension and also in the grounded extension. The arguments e and d are not defending any other argument, so no argument has to be included further. By using this strategy the grounded extension will be received automatically, because no other complete extension that is less, w.r.t. set-inclusion, can be build. The resulting set is indeed conflict-free and defends all its elements collectively.

For computing the grounded extension, Gaggl decided to proceed loops over the arguments of the BAF. In a first loop all arguments that are not attacked by any other argument of the BAF will be determined and added to the answer set. In a next loop it will be checked if these arguments are defending other arguments. If one argument is defending another argument this defended argument has to be added to the answer set. This will be proceed until no more defended arguments can be found. For proceeding this loops the order encoding from Program 22 can be reused. In addition to that all respective encodings for the grounded extension include the encodings from Programs 11, 12 and one of Programs 13 - 15 for the specific complex attack. In Program 26 the encoding for computing the d-grounded extension can be found, please note that the rule in line 3 is only present in the encoding file for BAFs with an evidential interpretation of support.

Program 26 d_ground.lp

```
1: %Proceeding loops to get all defended arguments.
2: defended_upto(X,Y) :- inf(Y), arg(X), not attack(Y,X).
3: defended_upto(X,Y) :- inf(Y), p_arg(X), not attack(Y,X). %evidential
4: defended_upto(X,Y) :- inf(Y), in(Z), attack(Z,Y), attack(Y,X).
5: defended_upto(X,Y) :- succ(Z,Y), defended_upto(X,Z), not attack(Y,X).
6: defended_upto(X,Y) :- succ(Z,Y), defended_upto(X,Z), in(V), attack(V,Y), attack(Y,X).
7: defended(X) :- supremum(Y), defended_upto(X,Y).
8: %Writing the defended arguments to the output file.
9: in(X) :- defended(X).
```

For computing the s-grounded (resp. c-grounded) extension the derived subset has to be checked regarding the requirement of safeness (resp. closure). If the received set does not fulfill the respective requirement, no s-grounded (resp. c-grounded) extension can be generated. In this case it is also impossible to generate a s-complete (resp. c-complete) extension of the given BAF.

To illustrate this, the BAF from Figure 15 should be considered using a necessary interpretation of support. The following attacks can be received: $attack(a, b)$, $attack(c, e)$, $attack(b, e)$, $attack(d, e)$, $attack(a, d)$. The argument a and c are not attacked by any other argument, so they have to be added to answer set. None of these arguments defend another argument. The argument a attacks argument b, which attacks e. But e is also attacked by c and d, and these attacks cannot be defended by a. Therefore the unique d-complete extension $\{a, c\}$, which is also the d-grounded extension, can be received. But this set is not safe, because a attacks b and c supports b, so this set set-supports and set-attacks b at the same time, which violates the requirement of safeness. The set $\{a, c\}$ is also not closed, because c is supporting b and d following that they have to be included regarding the requirement of closure, but neither b nor d are defended. So no s- or c-grounded extension can be found.

In this subsection the encoding for computing the d-grounded (resp. s- or c-grounded) extension of a given BAF were presented. In the next subsection the remaining encoding for computing stable extensions will be introduced.

6.7 Stable Extensions

Regarding to Definition 20 a subset S is a stable extension, if and only if S is conflict-free and S set-attacks all arguments that are not in S. The respective encoding can be found in Program 27. Please note that the complete encoding also includes the Programs 10 - 12 and one of the Programs 13 - 15 for the specific complex attack.

Program 27 stable.lp

```
%stable
:- in(X), in(Y), attack(X,Y).%conflict-free
:- out(Y), not attacked(Y). %stable
```

In this encoding the subsets $s \in S$ will be deleted, that do violate the requirements regarding conflict-free and stable sets.

In this section the necessary adaptions of the encodings that were presented by Gaggl in [13], were presented. In detail, the specific complex attacks, that were introduced in [5], for the deductive, necessary and evidential interpretation of support were implemented with ASP. Also the definition regarding closed sets presented by Cayrol and Lagasquie-Schiex in [6] was implemented. All other encodings that were developed by Gaggl, were adapted to BAFs with an deductive, necessary or evidential interpretation of support. By this adaptions the encodings are able to compute desirable semantics for BAF with the three different interpretation of support. To the best of my knowledge these encodings represent the first published ASP based encodings with these characteristics. By using the declarative programming paradigm of ASP a significant reduction regarding running time, compared to the Java-based encodings from the TweetyProject, will be expected. This assumption and a detailed analysis of the developed encodings will be presented in the upcoming section.

7 Evaluation of ASP-Encodings

After the introduction of ASP based encodings for computing desirable semantics of BAFs with deductive, necessary or evidential interpretation of support, this section includes the evaluating by using tasks from the ICCMA 2019 [3]. These tasks were also computed by the Java-based encodings from the TweetyProject, to investigate the differences regarding running time. In the upcoming subsection the method that was used for generating test cases will be introduced. After this subsection the design of the experiments will be presented and the section will be concluded by presenting the results.

7.1 Generation of Test cases

As mentioned before the instances from ICCMA 2019 [3], served as the basis to evaluate the encodings. Due to the fact, that this competition focuses on reasoning tasks in AFs, the instances had to be transformed to BAFs. The file that contains all instances is available on the homepage of the competition and contains 326 different AFs.

The instances are available in different encoding files, regarding to their modeling. The solver ASPARTIX participated in the ICCMA 2019, so *apx-files* are available. This file format corresponds to Definition 32, therefore these files were chosen for testing.

The different AFs can be ranked by the amount of arguments per instance, starting from small instances with 5 arguments in total, up to instances with 10000 arguments in total. To transform these AFs to BAFs, support relations between the arguments had to be added. This led to the question of how many support relations should be added per instance. The addition of support relations does make an instance more complex and may have an impact on the running time. Therefore the impact of different ratios of support relations, with respect to the amount of arguments of an instance, on the running time should be determined. For this investigation the three different ratios of 0.25, 0.5 and 0.75 (w.r.t. amount of total arguments) were used. The support relations were randomly added to each instance with the requirement of non adding self-supports.

For the evaluation of the encodings for BAFs with an evidential interpretation of support, *prima-facie* arguments had to be added. In a ratio of 0.25 (w.r.t. the amount of arguments per instance) standard arguments were randomly chosen and transformed to *prima-facie* arguments.

7.2 Experimental Design

After the introduction of the method for generating the test cases, the different tasks that were used to investigate the differences regarding running time, will be presented in this subsection. In the ICCMA 2019 [3], four different problems for each semantic were considered, that are defined as follows:

Definition 34. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF, let $x \in \mathcal{A}$ and let σ be a desirable semantics. The problem of deciding whether x is contained in some extension of σ, is called* ***credulous acceptability (DC)****.*

For the problem of credulous acceptability of semantics, one is interested in deciding if a certain argument is contained in at least one of the extensions of the semantics. In contrast to this the definition of *skeptical acceptability* will be presented.

Definition 35. *Let $(\mathcal{A}, R_{att}, R_{sup})$ be a BAF, let $x \in \mathcal{A}$ and let σ be a desirable semantics. The problem of deciding whether x is contained in all extensions of σ, is called* ***skeptical acceptability (DS)****.*

Instead of reviewing if a certain argument is contained in at least one extension of the semantics, one is interested in deciding if a certain argument is contained in all extensions of the semantics. Regarding to these two definitions, the output of the problems DC and DS is not a sequence of extensions, it is a Boolean value. This Boolean value can be "YES" or "NO", regarding to the result of the decision.

With the presented encodings from the previous section, all extensions of a desired semantics will be returned, this corresponds to the problem of ***enumerate all extensions (EE)***. A related problem is the problem of ***determine some extension (SE)*** of specific semantics.

For the evaluation of the ASP-encodings the problems DC, DS and EE were used. With the experiments the development of running time of the encodings when dealing with different instances should be investigated. For that the problem of SE was not used, because a more detailed analysis of the behavior of running time was expected by using the problem of EE. In addition to that the following semantics were used:

- d-preferred
- s-preferred
- c-preferred
- d-complete
- d-grounded
- stable

This list does not include all presented semantics from the last section. The semantics for computing admissible sets were not evaluated. This is due to the fact, that the encoding for computing the preferred extensions includes a determination of the respective admissible sets. The preferred extensions were evaluated completely, to compare the differences between the different types of admissibility (d, s and c). In addition to that the stable semantic were also evaluated completely. In addition to that the most general type of the complete and grounded semantic were evaluated completely. This is due to the fact, that the definitions of s-complete (resp. c-complete) extensions and the s-grounded (resp. c-grounded) extension is not proper defined and investigated in literature. In addition to that, the encoding of the s-grounded (resp. c-grounded) extension computes the d-grounded extension and than reviews if this extension is safe (resp. close), so no significant difference regarding running time was expected.

Because of time restrictions only instances with a maximal amount of 1000 arguments, which corresponds to a total number of 276 instances, with each three different ratios of supports, were used. This leads to a total number of 828 different BAFs, for that the support relation can be interpreted as deductive, necessary or evidential. For each of the three possible interpretations of support six different semantics, with each three different problems, could be computed. Which would lead to 14904 single runs for each interpretation of support, where a single run corresponds to test one of three problems for one of six semantics for a specific instance. Due to the limitation of time this full cycle of single runs was only done for the deductive and evidential interpretation of support. When the run consists the problem of DC (resp. DS), the argument that should be included in some (resp. all) extensions was chosen randomly from all arguments of the instance.

The main aim of this evaluation is to investigate the assumption of a reduced running time for the ASP based encodings compared to the Java-based encodings from the TweetyProject. To do so the solver of the TweetyProject only computed the problem of EE for all six semantics. Because the most impact on the running time was expected from this problem.

The ICCMA 2019 used a timeout of 600 seconds for each single run, this timeout was also used for the evaluation. When a single run passed the running time of 600 seconds, these runs were automatically cancelled. For testing the python tool *Probo2* [20] was used, that bundles functionalities for benchmarking argumentation solvers. The tests were proceeded on a server of the FernUniversität in Hagen, with the operating system of Ubuntu. The server includes an octacore Intel Xeon E5 processor with 3.4 GHz and a 64GB DDR4 RAM.

After the introduction of the design of the experiments the results will be presented in the upcoming subsection.

7.3 Results

In this subsection the results from the previously described experiments will be presented. In the following list all proceeded experiments are presented together with an assumption of the expected results:

1. **Impact of the ratio of support relations to the running times:** An intuitive assumption is, that when the amount of support relations in an instance rises, than the complexity will also rise. This would lead to higher running times for instances with a higher ratio of supports, compared to instances with lower ratios of support.

2. **Impact of the amount of arguments per instance:** Closely related to the previous assumption is the prediction of higher running times for instances with a higher amount of arguments, compared to instances with a lower amount of arguments.

3. **Impact of the semantics:** The developed encodings do have different extents, this is due to the fact that the semantics differ in their complexity. Some semantics are easier to compute than others and this reflects in the amount of lines for each encoding. Encodings with a minor amount of lines should correspond to lower running times compared to those of encodings with a higher amount of lines.

4. **Impact of the type of admissibility:** In Definitions 22, 23 and 24 three different types of admissibility for BAFs were presented. Considering the encodings of the preferred semantics, it is obvious that the requirements for the d-preferred and c-preferred semantics are very similar. The only difference is the additional requirement of closure for the c-preferred semantics. The semantics for s-preferred includes the additional predicate of *supportedN(X)*. This could lead to a slightly higher running time in comparison to the running time of the semantics for d- respectively c-preferred.

5. **Impact of the problem type:** Three different problem types were used for the evaluation. The output of the problems DC or DS is a single Boolean value of "YES" or "NO", depending on the inclusion of a given argument. One would assume that the running times of the problems DC and DS are very similar. The problem of DC is decided as "YES", when a first extension was found, that includes the given argument. In contrast to that, the problem of DS is decided as "NO", when a first extension was found, that do not include the given argument. Only in the case that the problem of DC returns "NO" and the problem of DS returns "YES", all extensions had to be reviewed. Only the running time of the problem EE, should be higher, compared to the running times of DC and DS, because for each instance all extensions have to be computed.

6. **Impact of the support interpretation:** The evidential interpretation of support differs by distinguishing between standard arguments and *prima-facie* arguments from the deductive and necessary interpretation of support. This distinction can lead to a reduction of the amount of attacks that are taken into account for an instance. Such a reduction can reduce the complexity of an instance and reduce the running time for computing the semantics.

7. **Impact of the implementation:** The usage of the declarative programming paradigm of ASP for computing acceptable sets of arguments of BAFs is a very promising approach, because it is based on non-monotonic logic, which represents a major property of human reasoning. The solver of the TweetyProject is also able to compute the desired semantics for BAFs with the deductive, necessary or evidential interpretation of support. The usage of formal logic elements for the implementation should lead to a precise reduction of running time for the presented ASP-based encodings, compared to the running time of the Java-based encodings from the TweetyProject.

The results can be found in the following, starting with the investigation of the impact of the ratio of supports in an instance on the running time.

7.3.1 Impact of the ratio of support relations:

To investigate the previous assumption of higher running times for higher ratios of supports, Figure 16 can be considered. This figure shows the running time of the problem *EE* for the d-grounded semantics, using three different ratios of supports.

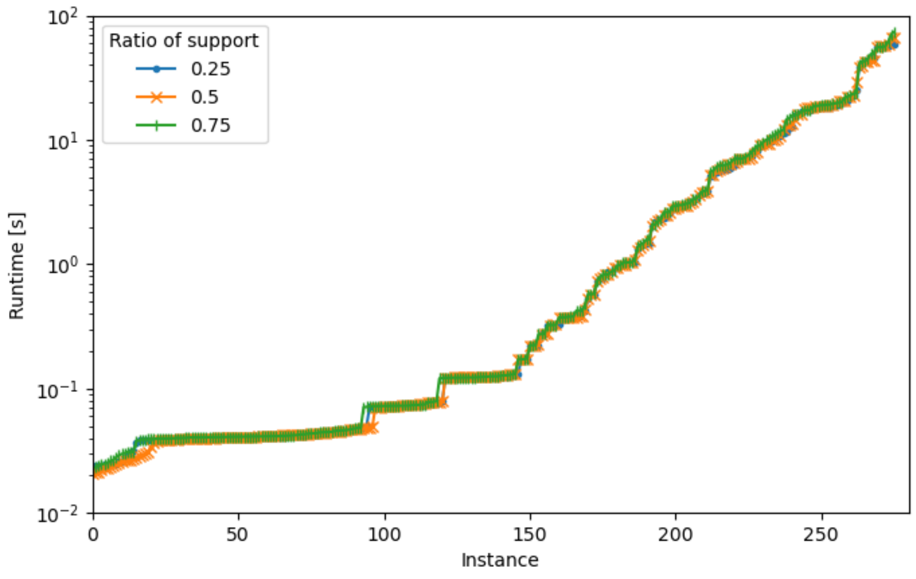

Figure 16: Running time for problem *EE* of the d-grounded semantics using three different ratios of supports.

This result is presented as a so-called ***Cactus-Plot***. For such a plot the instances of each data set are sorted ascending by their running times, resulting to a ranking of the single data points. Each data point is plotted by using the rank as the respective x-value and the running time in seconds as the y-value. Please note that a logarithmic scale was used for the y-axis.

The graphic in Figure 16 shows a cascading increase of running time for the 150 fastest solved instances that transforms to a almost constant rise. However the results do not support the assumption, because the running times do not to differ significantly by their ratio of supports. In comparison to that, Figure 17 shows the running time of the problem *EE* for the d-preferred semantics, using three different ratios of support.

This graphic shows also a cascading increase of running time for the 140 fastest solved instances. In contrast to the previous graphic this area consists of more steps,

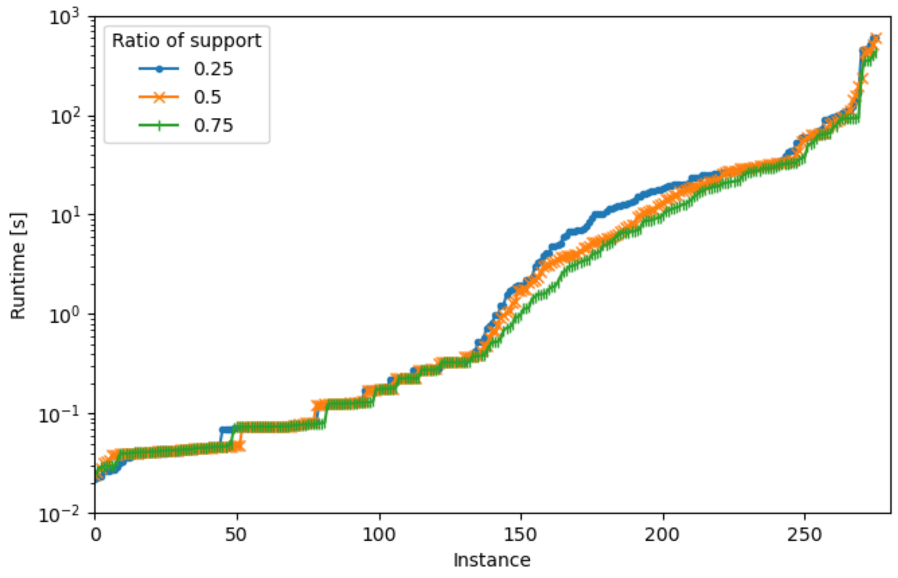

Figure 17: Running time for problem *EE* of the d-preferred semantics using three different ratios of supports.

resulting to a faster increase of running time. Surprisingly a wide area where the running time is lowest for the highest ratio of supports can be observed. Whereas the running time for the rest of the graph do not differ significantly between the different ratios of support. In comparison to that Figure 18 shows the running time for the problem of *EE* for the stable semantics.

In this graph the cascading increase of running time can be observed for the 200 fastest solved instances, where the highest ratio of support seems to enter the new levels slightly before the other ratios. However also an area can be observed where the lowest ratio of support result the highest values of running time for all three ratios.

Conclusively these results do not confirm the assumption that higher ratios of support lead to higher running times. Contrary to the expectations areas were found with the highest running times for the lowest ratio of support. However the impact of the amount of support relations could not be investigated in detail, because each addition of a support relation corresponds to a new BAF, leading to a restriction of comparability between the different data sets. For the following investigations a constant ratio of support of 0.25 (w.r.t. the amount of arguments per instance) will be used.

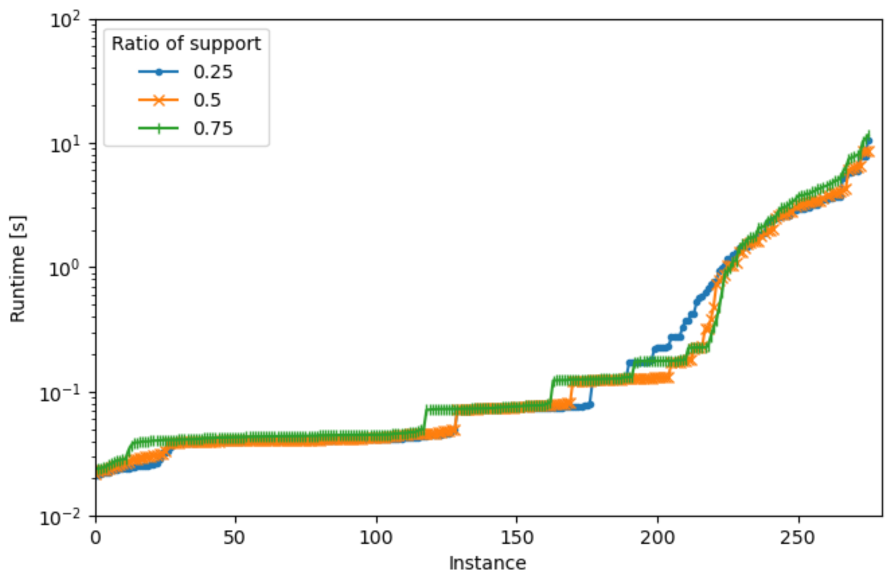

Figure 18: Running time for problem *EE* of the stable semantics using three different ratios of supports.

7.3.2 Impact of the amount of arguments:

To investigate the assumption that a higher amount of arguments in an instance lead to higher running times, Figure 19 can be considered. This figure shows the running time of the problem *EE* for the d-grounded semantics, using a ratio of 0.25 of supports.

In this graph the running time in seconds is plotted on the y-axis dependent to the number of arguments of an instance on the x-axis. Please note, that there are different instances with the same amount of total arguments, which leads to multiple marks for the same x-value in the graph. This graphic seems to confirm the intuitive assumption, in general. Except for a few single data points, a positive correlation between the amount of arguments per instance and the resulting running times can be observed.

In contrast to that, Figure 20 can be considerd, that shows the running time for the problem of *EE* for the d-preferred semantics.

In this graphic an increasing trend of the running times dependently on the amount of arguments per instance can be observed generally. However this correlation is not that strong compared to the correlation that was found in Figure 19. In comparison to the clear trend in this graph, the graph for the d-preferred semantics is interrupted by a lot of single peaks.

In addition to that Figure 21 shows the running time for the problem of *EE* for the

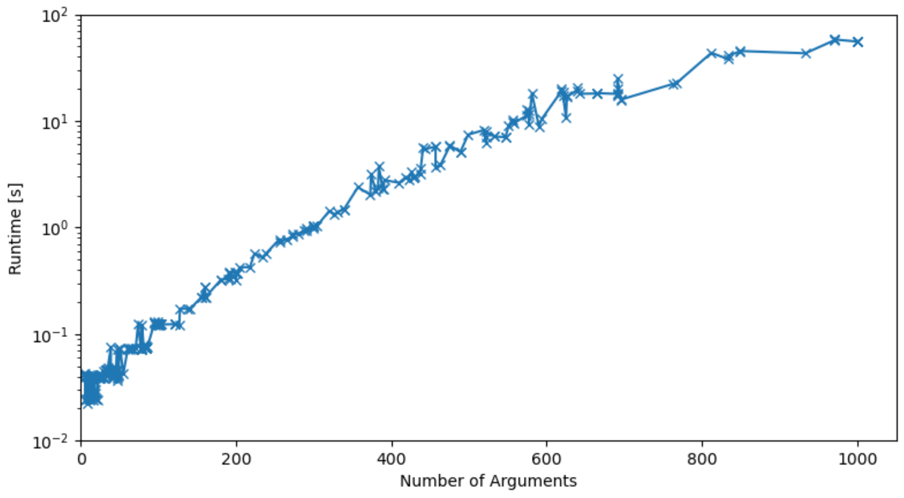

Figure 19: Running time for problem *EE* of the d-grounded semantics.

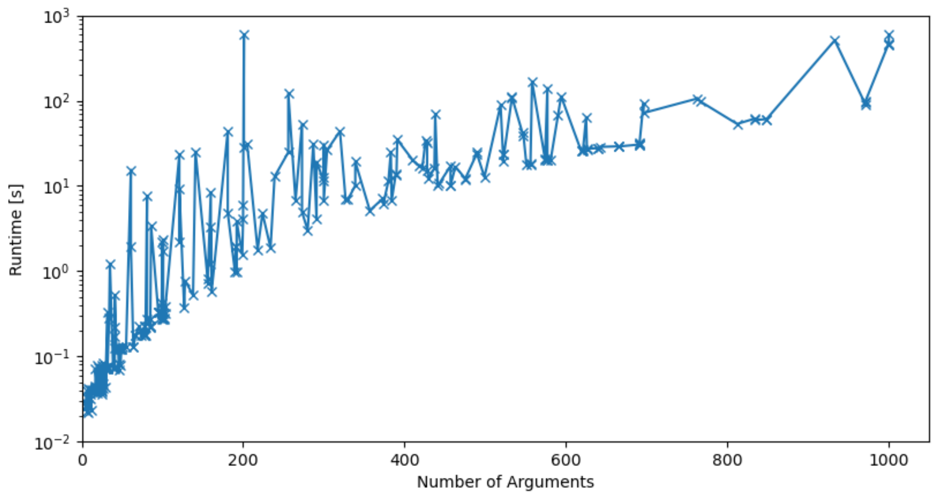

Figure 20: Running time for problem *EE* of the d-preferred semantics.

stable semantics.

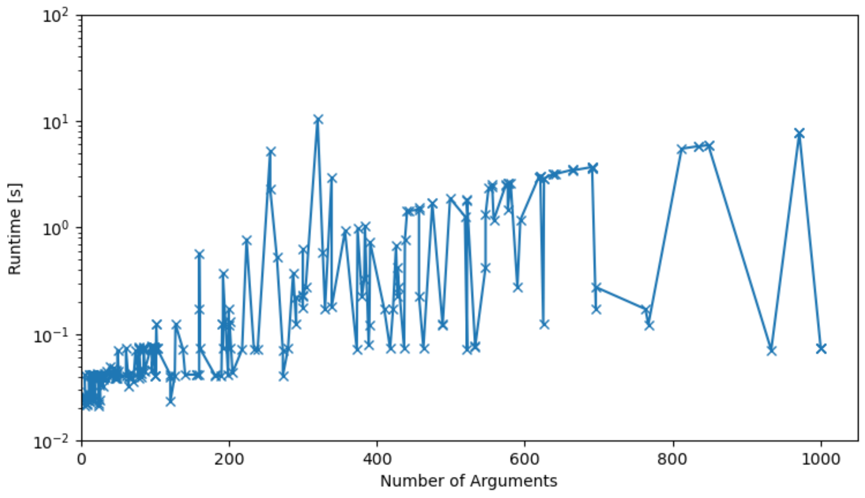

Figure 21: Running time for problem *EE* of the stable semantics.

Compared to the previous graphs, in this graph no trend can be observed. The running times are irregularly distributed over the whole graph.

These results lead to the conclusion that the total number of arguments in an instance does have an influence on the running time. However this correlation is not linear for all cases, instances with less arguments can also be more complex than instances with a higher amount of arguments.

7.3.3 Impact of the semantics:

Figure 22 shows the running times for the problem of *EE* for the semantics of d-preferred (D_PR), d-grounded (D_GR), d-complete (D_CO) and stable (ST).

This graph confirms the assumption of a positive correlation between the extent of an encoding and the running time. The encodings of the stable and d-complete semantics do have the lowest amounts of lines and show the lowest values regarding running time. There is only one exception for the d-complete semantics, where one instance passed the timeout of 600 seconds. The second highest amount of lines corresponds to the d-grounded semantics, that achieve the second highest values regarding running time. The most extensive encoding belongs to the d-preferred semantics, that shows the highest running times in the graph.

7.3.4 Impact of the type of admissibility:

Figure 23 shows the running times for the problem of *EE* for the preferred semantics.

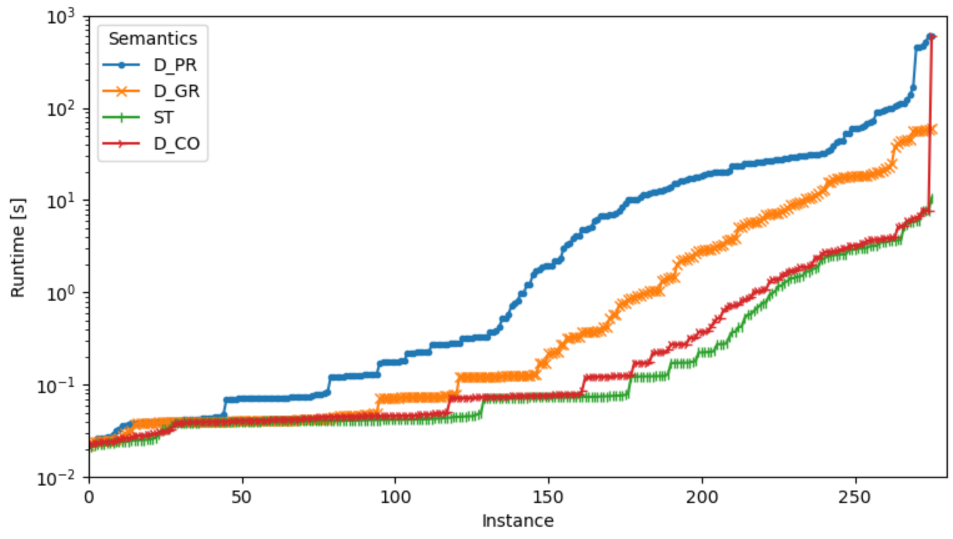

Figure 22: Running time for problem *EE* of different semantics.

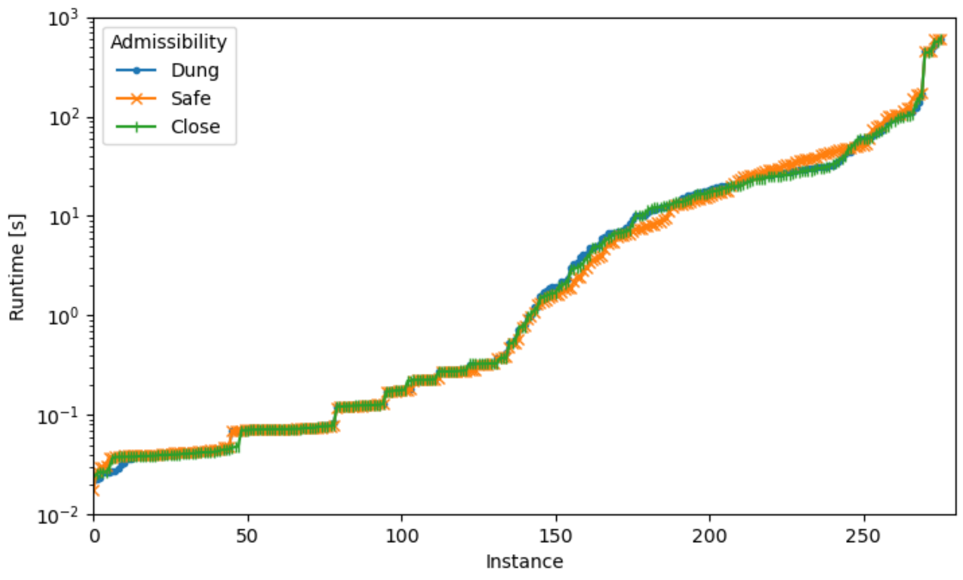

Figure 23: Running time for problem *EE* of d-, s- and c-preferred semantics.

This graph confirms the assumption of a very similar running time for all three different types of admissibility. These results support the conclusion, that the type of admissibility does have an inferior impact on the running time.

7.3.5 Impact of the problem type:

To investigate the assumption of a higher running time for the problem of *EE*, compared to the running times of the problems *DS* and *DC* Figure 24 can be considered. This figure shows the running time of the three different problems for the d-grounded semantics.

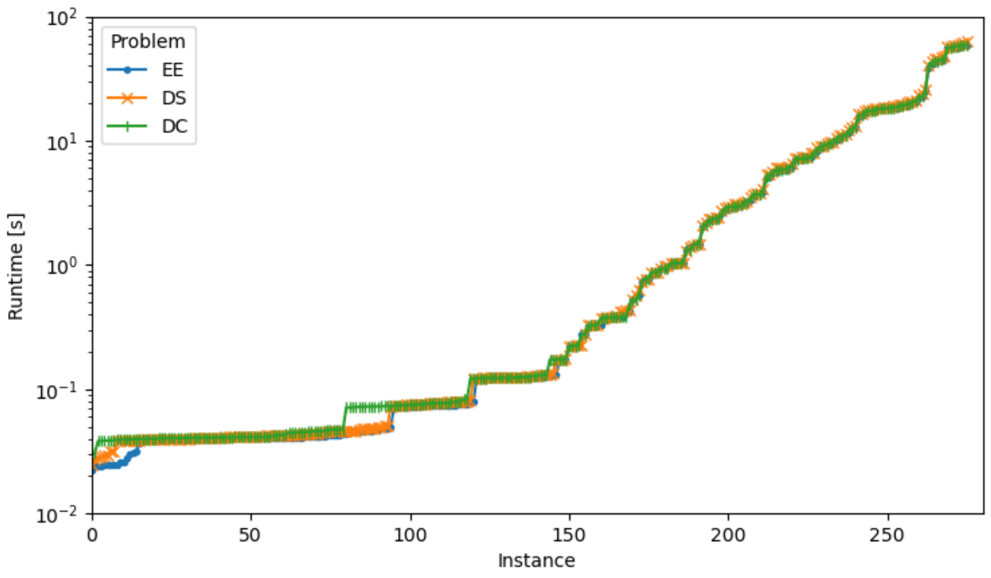

Figure 24: Running time for different problems of the d-grounded semantics.

This graph confirms the assumption of a similar running time for the problems of *DS* and *DC*, but it does not support the assumption of a higher running time for the problem of *EE*. In contrast to that, all three problem types show a similar running time. In comparison to that Figure 25 shows the running time of the three different problems for the d-preferred semantics.

This graph supports the assumption of a higher running time for the problem of *EE*, in addition to that an area can be found where the problem of *DS* shows a higher running compared to the problem of *DC*.

Conclusively Figure 26 shows the running time of the three different problems for the stable semantics.

This graph is very similar to Figure 24, in which a very similar running time for

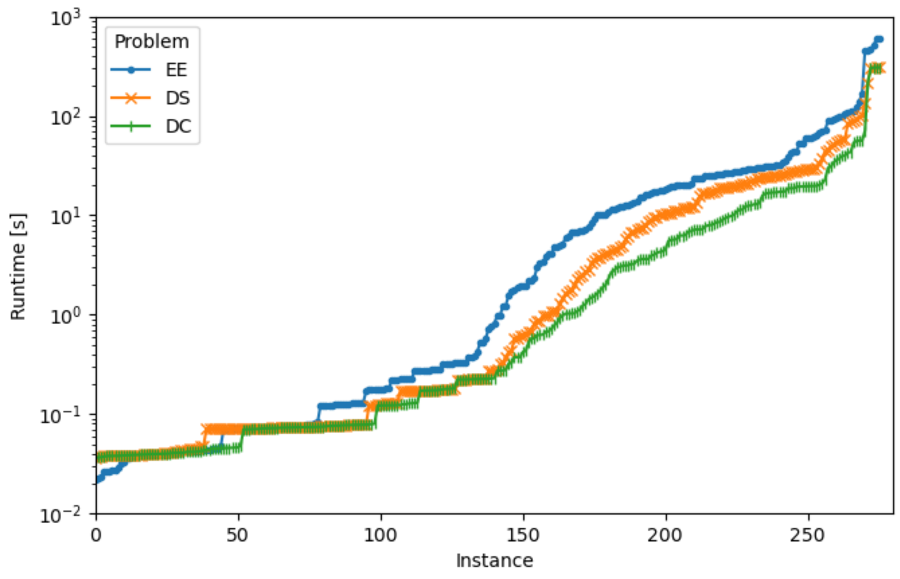

Figure 25: Running time for different problems of the d-preferred semantics.

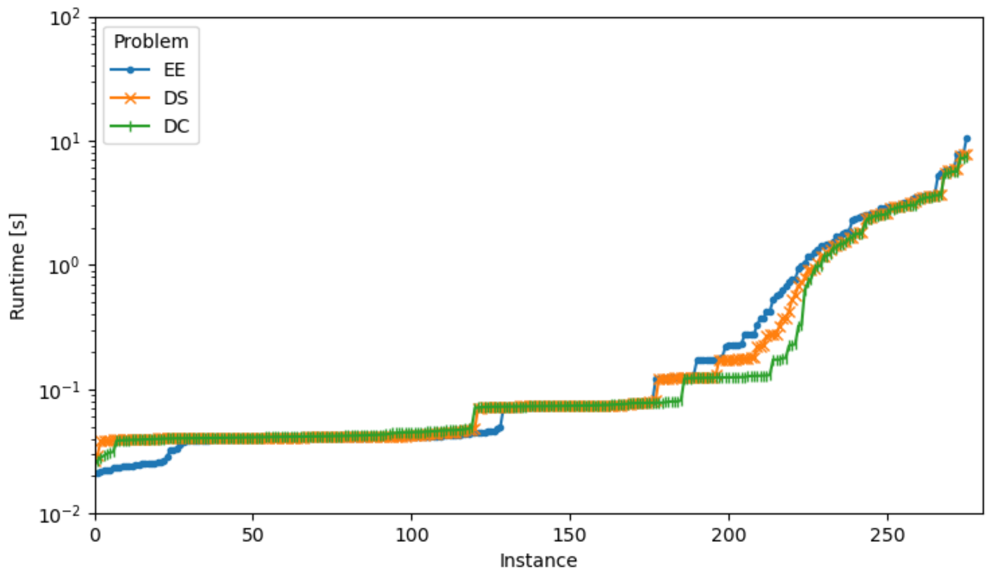

Figure 26: Running time for different problems of the stable semantics.

the three different problems could be observed. In contrast to that an area with the same ranking, that was found in Figure 25, can be found.

Conclusively a very similar running time for all three problems could be shown in general. Only for the d-preferred semantics differences regarding the running time could be shown for the three different problem types.

7.3.6 Impact of the support interpretation:

To investigate the assumption that the usage of the evidential interpretation of support can lead to a reduction of running time compared to the deductive interpretation of support, Figure 27 can be considered. This figure shows the running time for the problem of *EE* for the d-grounded semantics.

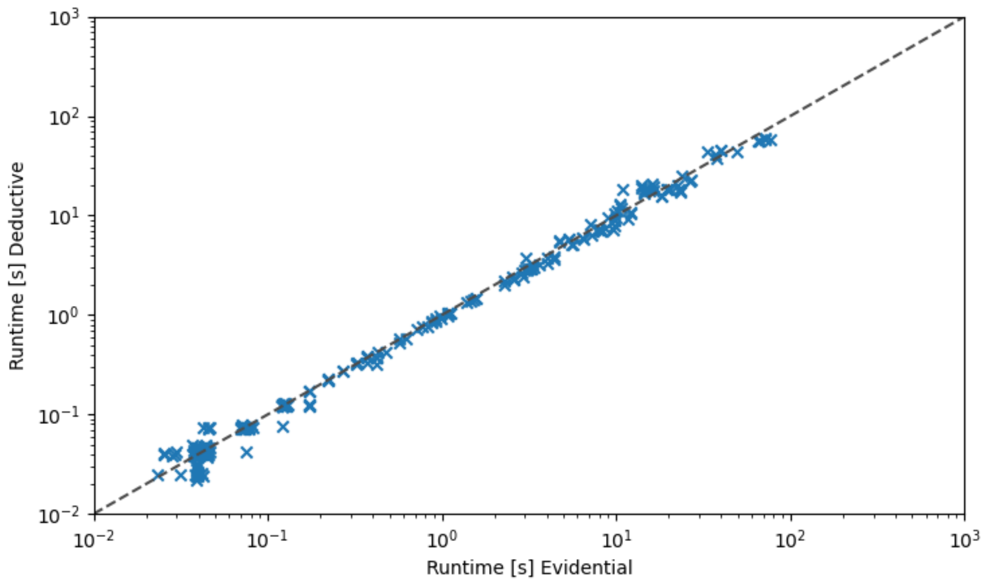

Figure 27: Running time for problem *EE* of the d-grounded semantics.

The results are presented as an ***x-y-Plot***. In comparison to the previously used Cactus-Plots the instances are not sorted by their running times. Each data point in the graph corresponds to the same instance and is plotted by using the running time for the deductive interpretation of support as the y-value and the running time for the evidential interpretation of support as the x-value. Both axis are using a logarithmic scale and in addition to the data points a dotted line that represents the diagonal is plotted. When a data point is located on this dotted line this means, that both interpretations of support achieved the same running time for this instance. When a data point is located under the dotted line this means, that the running time of the deductive interpretation of support was lower for this instance, compared to the evidential one.

The results for the d-grounded semantics do not support the assumption, both interpretations of support show a similar running time for all instances. In comparison to that Figure 28 shows the running time for the problem of *EE* for the d-preferred semantics.

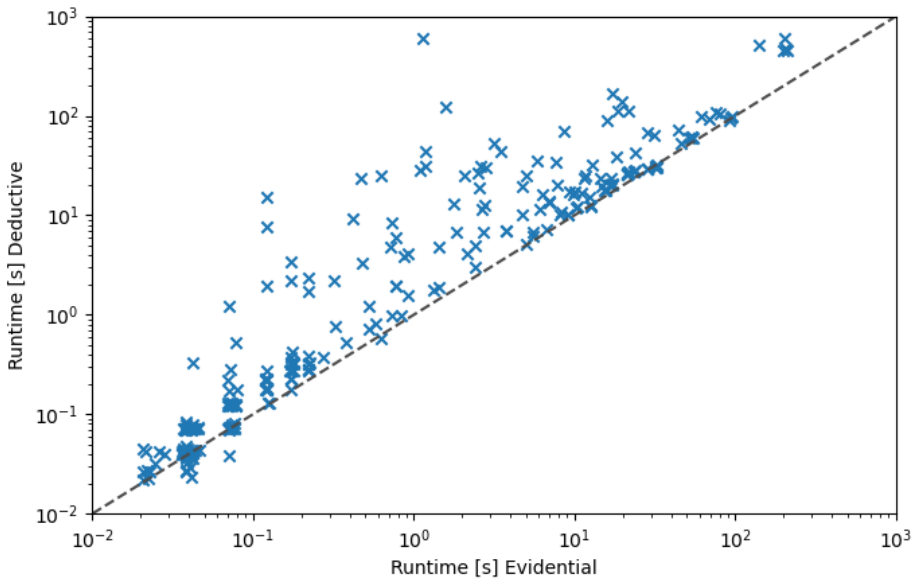

Figure 28: Running time for problem *EE* of the d-preferred semantics.

In contrast to the results for the d-grounded semantics, these results show a precise reduction of running time for the evidential interpretation of support. All data points, except a few single data points, are located over the dotted line. In addition to that Figure 29 shows the running time for the problem of *EE* for the stable semantics.

In this graph all data points, except a few single data points, are located over the diagonal.

Conclusively the results support the assumption of a reduced running time for the evidential interpretation of support compared to the deductive one. The distinguish between standard and *prima-facie* arguments for the evidential interpretation of support can lead to a reduction of the cardinality of R_{att} and this reduction is associated with a lower running time. However this correlation is impacted by the type of the semantics. For the d-grounded semantics this correlation could not be shown.

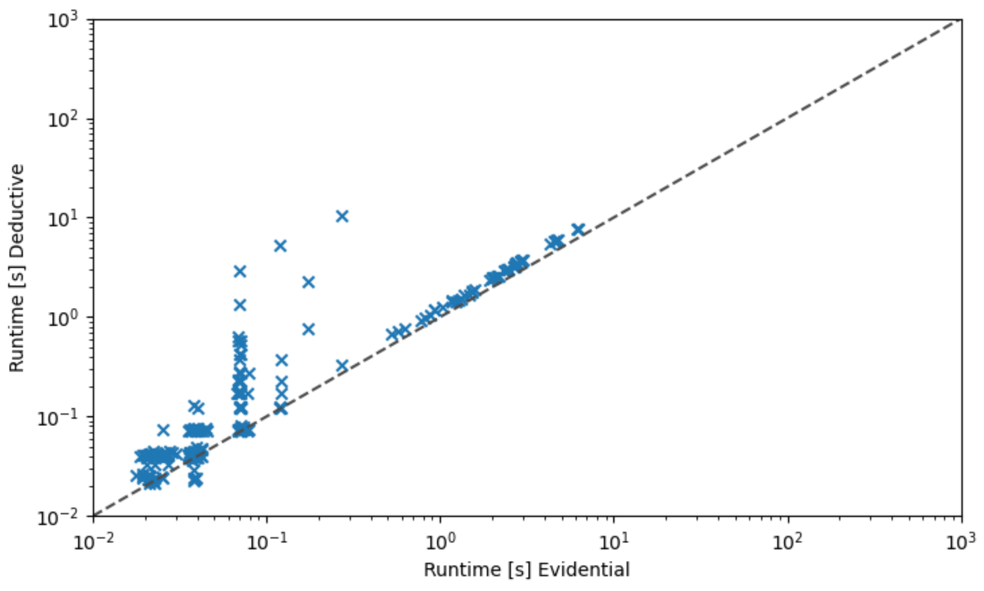

Figure 29: Running time for problem *EE* of the stable semantics.

7.3.7 Impact of the implementation:

To investigate the assumption of a precise reduction of running time for the ASP-based encodings compared to the Java based encodings from the TweetyProject, Figure 30 can be considered. This figure shows the running time for the problem of *EE* for the d-preferred semantics.

Unfortunately only the d-preferred semantics could be computed without errors by the solver of the TweetyProject. The solver was able to determine the extensions for instances with a maximal amount of 23 arguments. For instances with an amount of 24 or more arguments a memory overflow occurred, that lead to a completely empty output for these instances. The respective running times where manually changed to 600 seconds. In contrast to that the extensions of all instances, except of a single one, could be computed completely by using the ASP-based encodings. The graph from Figure 30 confirms the assumption of a precise reduction of running time for the ASP-based encodings. For all instances the ASP-based encodings were faster or equal, regarding running time, to the Java-based encodings of the TweetyProject.

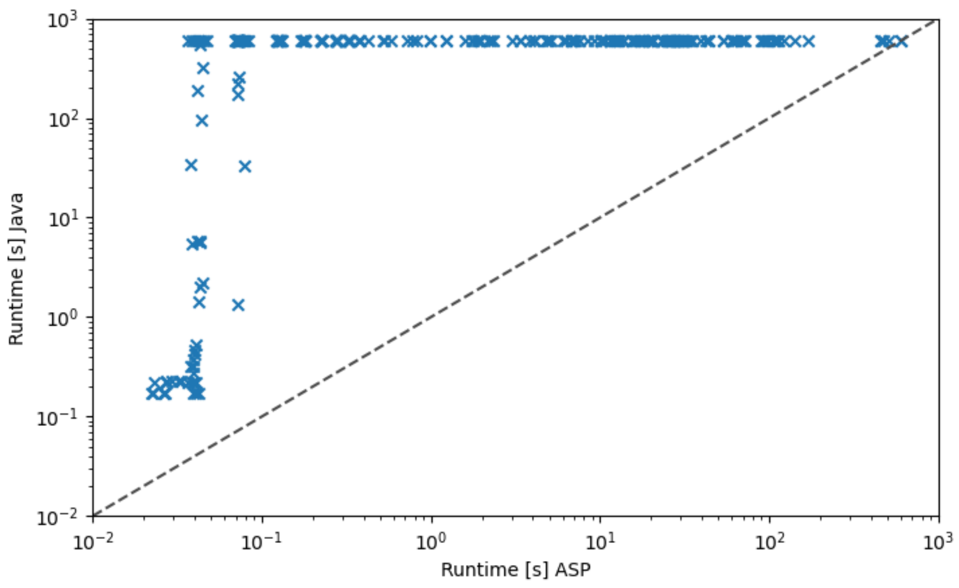

Figure 30: Running time of problem *EE* for the d-preferred semantics.

8 Conclusion

In this work the concepts of two very important models for formulating human argumentation: the ***abstract argumentation frameworks*** and the ***bipolar argumentation frameworks*** were introduced. By using BAFs, realistic frameworks of human argumentation can be build. This is due to the fact, that in addition to the possibility of attack relations, support relations between arguments can be used.

The interpretation of the support relation is not uniformly. Three different interpretation of support: ***deductive, necessary*** and ***evidential*** were presented. The interpretation of support is highly related to the context of the argumentation framework. In addition to that each type of support interpretation has its own complex attacks. This results to different acceptable sets of arguments for the same framework, when the support relation is interpreted in different ways.

The existing argumentation reasoning tool ***ASPARTIX***, that proofs the possibility of using the declarative programming paradigm of ***answer set programming***, for computing the acceptable sets for AFs and BAFs, was extended. The presented encodings are able to compute the extensions of all common desired semantics of BAFs with the deductive, necessary or evidential interpretation of support. To the best of my knowledge these encodings are the first published ASP-based encodings with these properties.

The presented encodings were compared to the Java-based solver of the Tweety-Project, which is also able to compute the extensions of the desired semantics of BAFs with the deductive, necessary or evidential interpretation of support. The developed ASP-based encodings showed a precise reduction of running time compared to the Java-based solver, which underlies the potential of using ASP for reasoning tasks.

In addition to that analysis of the factors that influence the running time of the developed ASP-based encodings was performed. The results showed that the influence of the type of the semantic do have an impact on the running time. Whereas no impact or no linear correlation could be shown for the amount of arguments per instance, the type of admissibility, the ratio of support relations (w.r.t. the amount of arguments) and the type of problem that is considered for the semantic.

Future work has to be done, to investigate the impact of the different factors to the complexity of a BAF. With a better understanding of how these factors influence the complexity of a BAF, the differences in running time can be estimated and explained in a more detailed way.

References

[1] Ph Besnard et al. „Semantics for evidence-based argumentation“. In: *Computational models of argument: proceedings of COMMA*. Vol. 172. 2008, p. 276.

[2] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1986.

[3] Stefano Bistarelli et al. „Summary report for the third international competition on computational models of argumentation“. In: *AI Magazine* 42.3 (2021), pp. 70–73.

[4] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. „Bipolar abstract argumentation systems“. In: *Argumentation in Artificial Intelligence*. Springer, 2009, pp. 65–84.

[5] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. „Bipolarity in argumentation graphs: Towards a better understanding“. In: *International Journal of Approximate Reasoning* 54.7 (2013), pp. 876–899.

[6] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. „On the acceptability of arguments in bipolar argumentation frameworks“. In: *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*. Springer. 2005, pp. 378–389.

[7] Stephen A Cook. „The complexity of theorem-proving procedures“. In: *Proceedings of the third annual ACM symposium on Theory of computing*. 1971, pp. 151–158.

[8] Phan Minh Dung. „On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games“. In: *Artificial intelligence* 77.2 (1995), pp. 321–357.

[9] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. „Answer-set programming encodings for argumentation frameworks“. In: *Argument & Computation* 1.2 (2010), pp. 147–177.

[10] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. „Aspartix: Implementing argumentation frameworks using answer-set programming“. In: *International Conference on Logic Programming*. Springer. 2008, pp. 734–738.

[11] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. „Answer set programming: A primer“. In: *Reasoning Web International Summer School*. Springer. 2009, pp. 40–110.

[12] Thomas Eiter and Axel Polleres. „Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications“. In: *Theory and Practice of Logic Programming* 6.1-2 (2006), pp. 23–60.

[13] Sarah Alice Gaggl. *Solving argumentation frameworks using answer set programming*. na, 2009.

[14] Martin Gebser and Torsten Schaub. „Modeling and language extensions“. In: *AI magazine* 37.3 (2016), pp. 33–44.

[15] Martin Gebser, Torsten Schaub, and Sven Thiele. „Gringo: A new grounder for answer set programming“. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer. 2007, pp. 266–271.

[16] Martin Gebser et al. „Advances in gringo series 3“. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer. 2011, pp. 345–351.

[17] Martin Gebser et al. „clasp: A conflict-driven answer set solver“. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer. 2007, pp. 260–265.

[18] Michael Gelfond and Vladimir Lifschitz. „Classical negation in logic programs and disjunctive databases“. In: *New generation computing* 9.3 (1991), pp. 365–385.

[19] Benjamin Kaufmann et al. „Grounding and solving in answer set programming“. In: *AI magazine* 37.3 (2016), pp. 25–32.

[20] Jonas KLEIN and Matthias THIMM. „probo2: A Benchmark Framework for Argumentation Solvers“. In: *Computational Models of Argument: Proceedings of COMMA 2022* 353 (2022), p. 363.

[21] Nicola Leone et al. „The DLV system for knowledge representation and reasoning“. In: *ACM Transactions on Computational Logic (TOCL)* 7.3 (2006), pp. 499–562.

[22] V Lifschitz. *The stable model semantics for logic programming*. 1988.

[23] Vladimir Lifschitz. „Answer set programming and plan generation“. In: *Artificial Intelligence* 138.1-2 (2002), pp. 39–54.

[24] Raymond Reiter. „A logic for default reasoning“. In: *Artificial intelligence* 13.1-2 (1980), pp. 81–132.

[25] Patrik Simons, Ilkka Niemelä, and Timo Soininen. „Extending and implementing the stable model semantics“. In: *Artificial Intelligence* 138.1-2 (2002), pp. 181–234.

[26] Matthias Thimm. „Tweety: A comprehensive collection of java libraries for logical aspects of artificial intelligence and knowledge representation“. In: *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*. 2014.

[27] Matthias Thimm et al. „Summary report of the first international competition on computational models of argumentation“. In: *AI magazine* 37.1 (2016), pp. 102–102.

[28] Frans H Van Eemeren and Rob Grootendorst. *A systematic theory of argumentation: The pragma-dialectical approach*. Cambridge University Press, 2004.