FAKULTÄT FÜR !NFORMATIK

Solving Argumentation Frameworks using Answer Set Programming

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Sarah Alice Gaggl
Matrikelnummer 0026566

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer/Betreuerin: Ao.Univ.Prof. Dr. Uwe Egly
Mitwirkung: Privatdoz. Dr. Stefan Woltran

Wien, 16.02.2009

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Contents

Erklärung zur Verfassung der Arbeit

Sarah Alice Gaggl

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

Wien, 13.02.2009

Kurzfassung

Im Laufe der letzten Jahre hat sich Argumentation zu einem zentralen Gebiet in der Künstlichen Intelligenz (KI) etabliert. Dung führte 1995 *Argumentation Frameworks (AFs)* als ein formales System ein, das es einem ermöglicht, aus widersprüchlichem Wissen Schlüsse zu folgern. Desweiteren hat er mehrere Semantiken definiert, wie zum Beispiel *admissible*, *complete*, *preferred* und *grounded*, mit deren Hilfe man Mengen von zulässigen Argumenten berechnen kann.

Obwohl das Framework von Dung bereits sehr viele Möglichkeiten zur Verfügung stellt um Aussagen über Argumente zu treffen, wurde es kontinuierlich erweitert und verfeinert. Bis jetzt wurden viele weitere Semantiken eingeführt, wie zum Beispiel *s-admissible, c-admissible, semi-stable, prudent* und *ideal*. Zudem wurde auch das Framework durch Präferenzrelationen (PAFs), Werte (VAFs) und Unterstüzungsrelationen (BAFs) erweitert. All das führt dazu, dass wir mit unterschiedlichen Frameworks und Semantiken geradezu überschüttet werden.

Ein Bestandteil dieser Arbeit und Voraussetzung für eine elegante Implementierung ist das Erkennen der relevanten und vielversprechenden Frameworks und Semantiken, und das Vereinheitlichen der unterschiedlichen Definitionen. Trotz ausgiebiger Forschung auf diesem Gebiet gibt es noch immer kein System, mit dem man alle grundlegenden Semantiken von Dung berechnen kann.

Daher war der Hauptbestandteil dieser Masterarbeit, ein System zu entwickeln, mit dessen Hilfe man alle Semantiken für Dungs Framework, sowie die wichtigsten für PAFs, VAFs und BAFs berechen kann. Unser System ASPARTIX, welches für "Answer Set Programming Argumentation Reasoning Tool" steht, zeichnet sich dadurch aus, dass es aus einem fixen disjunktiven logischen Programm besteht, welches vollkommen unabhängig vom jeweiligen Argumentation Framework ist. Das bedeutet, dass der Benutzer nur das Framework und die gewünschte Semantik als Eingabedatenbank spezifizieren muss. Daher kann ASPARTIX auf einfache Weise verwendet werden, um die Unterschiede und Zusammenhänge der verschiedenen Semantiken und Frameworks zu erforschen. Die Modularität unseres Systems ermöglicht eine einfache Erweiterung und Verbesserung.

Abstract

Within the last years, argumentation has become a very important field in Artificial Intelligence (AI). In 1995, Dung introduced *Argumentation Frameworks (AFs)* as a formal system to reason over conflicting knowledge. He defined several semantics (i.e., *admissible, complete, preferred* and *grounded*) for AFs to compute acceptable sets of arguments.

Even though Dung's framework offers many possibilities for reasoning over arguments, it has been extended continuously. Till now, many more semantics (i.e., *s-admissible, c-admissible, semi-stable, prudent, ideal*) have been introduced. Also the basic framework has been extended by, for example, preference relations (PAFs), values (VAFs) and support relations (BAFs), just to name some of them. All these extensions lead to the fact that we are overwhelmed with different frameworks and semantics.

One part of this work is to filter out the most relevant/promising frameworks and semantics and to bring all those definitions into a uniform notation. Such a uniform notation is a prerequisite for an elegant implementation. Despite a constant research effort in this area, there exists neither an implementation for computing all semantics of Dung's original framework nor implementations for the extensions thereof.

The main part of this thesis is to develop a system to compute all semantics of Dung's framework, as well as the most relevant ones for PAFs, VAFs and BAFs. Our system ASPARTIX, which stands for Answer Set Programming Argumentation Reasoning Tool, is exceptional due to the fact that it is a fixed disjunctive logic program, totally independent of the concrete AF. This means that the user just needs to set up the framework and the desired semantic as an input database. It is not necessary to make any translation of the framework or to modify the system. This makes it easy for researchers to use ASPARTIX to compare the different semantics and frameworks even if the user is not an expert in answer set programming. Due to the modularity of our system, it is easy to extend and debug.

Acknowledgements

I owe a great deal to colleagues, friends and members of my family who have supported me in completing my degree, and who, through their own research, comments and questions have encouraged and enlightened me.
In particular, I want to thank my supervisor Uwe Egly. I have learned much from working with him on that project, and his notes and suggestions have been a great help in writing this thesis.
Furthermore, I greatly appreciate the support of my second adviser Stefan Woltran. His support has been a great help with the formal details. By spending such a lot of time on proofreading, he enabled me to complete this work in time.
Special gratitude is dedicated to my parents, Elisabeth and Albin Gaggl. Without their support, I would not have been able to make a full time study without any financial problems.
One of my greatest supporter has been Ülkü Keskin. She always believed in me and has been positive about the fact that I can achieve my goals.

Chapter 1

Introduction

When the mind's eye rests on objects illuminated by truth and reality, it understands and comprehends them, and functions intelligently; but when it turns to the twilight world of change and decay, it can only form opinions, its vision is confused and its beliefs shifting, and it seems to lack intelligence.
(Plato, 380BC)

Discussions and argumentation are part and parcel of human communication and interaction. Hence, argumentation regards almost everybody. The most important parts of such an argumentation are the arguments. Intuitively, one can say that the one who brings up the better arguments, comes off as winner.

Imagine a political debate on TV, politicians need to define their own position and want to derive a benefit from their discussions. Therefore, they are not only interested in bringing forward good arguments for their political programs, they also try to attack the arguments of their political opponents, and defend themselves against attacks. Most politicians, managers, economists and in general people who arouse public interest prepare themselves very good for such discussions. They check up on the subject and their opponent, they graduate special trainings to improve their skills, like for example Neuro Linguistic Programming (NLP), and they have several consultants who ensure that they do not make any mistakes. However, not only politicians and economists benefit from good argumentation skills. From time to time, everybody faces such situations, even if it is just a discussion with a friend.

It is possible to observe that the arguments in such a dialog bear a relation to each other, and also that different means of privileges can appear. Therefore, the idea of analysing, formalising, and of course solving argumentation arose very early in history. Down to the present day, argumentation has become a central issue in many research areas, i.e. philosophy, philology, mathematics, logic, artificial intelligence (AI), law research and multiagent systems, just to mention a few of them.

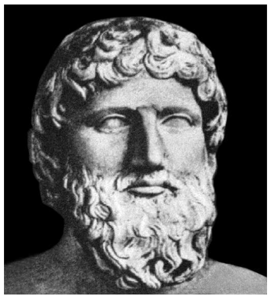


Figure 1.1: Plato (428/427 BC - 348/347 BC)

Nevertheless, till now there does not exist a computer-aided system, capable of solving all those argumentation issues. With this work, we hope to contribute to change this fact.

This chapter is structured as follows. In Section 1.1 we give some famous characters from history, who already dealt with argumentation. In Section 1.2 we analyse the status quo concerning argumentation frameworks. In Section 1.3 we point out the main contributions of this thesis. Finally, in Section 1.4 we exemplify how argumentation frameworks can be used, and the need of extending the basic framework by values.

1.1 Historical Background

The desire of dealing with argumentation in a formal manner can be traced back till antiquity. The quarrel was part and parcel of scientific research during the antiquity and the Middle Ages. The concept of dialectic was first mentioned by Plato who treated it as a question-and-answer game. In the so called Socratic Method, the arguments were represented by the questions, and the progress of the argumentation resulted from the affirmative or denial of these premises. Also Leibniz dreamed of an automatization of the philosophical dispute. Therefore he conceived of and attempted to design a *lingua characteristica* (a language in which all knowledge could be formally expressed) and a *calculus ratiocinator* (calculus of reasoning) such that when philosophers disagreed over some problem they could say 'calculemus' (let us calculate) and agree to formulate the problem in the lingua characteristica and solve it using the calculus ratiocinator. Although Leibniz believed in the realizability of his dream, he needed to face up to several problems such as:

- some philosophical weaknesses,
- the inevitable incompleteness of science,

If we had it, we should be able to reason in metaphysics and morals in much the same way as in geometry and analysis... If controversies were to arise, there would be no more need of disputation between two philosophers than between two accountants. For it would suffice to take their pencils in their hands, to sit down to their slates, and to say to each other (with a friend as witness, if they liked): let us calculate.
(Gottfried Leibniz)

Figure 1.2: Gottfried Wilhelm Leibniz (1646-1716)

- insufficient rigor in the mathematics of his day,
- the limitations of the logic known in his time, and
- the lack of adequate information technology.

Although Leibniz's dream of a complete formalization of science was destroyed in the thirties of the last century, restricted versions of Leibniz's dream survived.

Another famous philosopher who dealt with argumentation was Arthur Schopenhauer. In his book "The Art of Controversy", he gives a guideline for winning every conversation. In contrast to Plato and Leibniz, he does not concentrate on finding a logic based solution for a dispute. According to his opinion, nearly everything is allowed, cheating included, to win a conversation.

1.2 Current State of Affairs

Within the last decade, the area of argumentation has become a central issue in Artificial Intelligence (AI) (see [12] for an excellent summary). Argumentation frameworks formalize statements together with a relation, denoting conflicts between them. By special semantics for those frameworks, it is possible to built sets of acceptable arguments, and therefore to solve the inherent conflicts.

Argumentation covers a wide range of applications fields, including multiagent systems and law research. This led to the fact, that till now several different proposals for formalizations of argumentation exist. They not only vary in the way to define acceptable arguments, also the notion of rebuttal has different meanings, or even additional relationships between statements are taken into account. Finally, statements are augmented with priorities, such that the semantics yields those admissible sets which contain statements of higher priority.

The absence of an accepted standard for argumentation framework is one current problem. Furthermore, argumentation problems are in general intractable, thus developing dedicated algorithms for the different reasoning problems is non-trivial.

1.3 Main Contributions of the Thesis

Argumentation has gained increasing interest in artificial intelligence within the last years. The consequence thereof is that we face loads of different frameworks and semantics. To keep sight of the bigger picture, it is important to know which framework is convenient for a particular problem, and which possibilities for reasoning are available for that framework.

In this work, we investigate the most relevant and promising argumentation frameworks together with the underlying semantics, and we bring them into a uniform notation. Moreover, properties of acceptability, depending on the structure of argumentation graphs, are analysed. We also describe the relations between the different semantics. Therewith, we form the basis for an elegant implementation.

Furthermore, there still does not exist neither an implementation, capable of computing all semantics of Dung's basic argumentation framework, nor for the extensions thereof. With this work, we present solutions for reasoning problems in different types of argumentation frameworks, by means of computing the answer sets of an (extended) datalog program. To be more specific, with our system, many important types of extensions (i.e., admissible, preferred, stable, complete and grounded) in Dung's original argumentation framework (AF) [25], the preference based framework (PAF) [1], the value based framework (VAF) [10], and the bipolar framework (BAF) [2, 19] can be computed.

The declarative programming paradigm of *Answer Set Programming* (ASP) [41,43] under the stable-models semantics [40] (which is our target formalism) is especially well suited for our purpose. First, advanced solvers such as Smodels, DLV, GnT, Cmodels, Clasp, or ASSAT are available which are able to deal with large problem instances (see [38]). Thus, using the proposed implementation approach delegates the burden of optimizations to these systems. Second, language extensions such systems offer can be used to employ different extensions to AFs, which so far have not been studied (for instance, weak constraints or aggregates could yield interesting specially tailored problems for AFs). Finally, depending on the class of the programs one uses for a given type of extension, one can show that, in general, the complexity of evaluation within the target formalism is of the same complexity as the original problem. Thus, our approach is adequate from a complexity-theoretic point of view.

The advantage of a fixed logic program is that it is completely independent of the argumentation framework to process. Moreover, the input AF can be changed easily and dynamically which simplifies the answering of questions like “What happens if I add this new argument?”.

Our system makes use of the prominent answer set solver DLV [41]. All necessary programs to run ASPARTIX and some illustrating examples are available at the following web side:

http://www.kr.tuwien.ac.at/research/systems/argumentation/

Parts of the work have been published. The system ASPARTIX has been presented at the *24th International Conference on Logic Programming (ICLP 2008)* [32], and the encodings for the system ASPARTIX have been presented at the *1st Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2008)* [30]. Additionally a technical report [31] presents more details on the system and the encodings.

In the next section, we show the application of argumentation frameworks by means of two examples from the law research area.

The structure of the remaining thesis is as follows:
Chapter 2 provides the necessary definitions of argumentation frameworks, as well as an analysis of the different semantics and frameworks. The formal background of ASP is introduced in Chapter 3. Besides syntax and semantics of ASP, we discuss some programming techniques which we used to implement our system. In Chapter 4, we discuss complexity issues for argumentation frameworks. The ASP encodings and the proof of their correctness are introduced stepwise in Chapter 5. Chapter 6 is dedicated to the description of the system ASPARTIX and its utilization. An overview of related work is given in Chapter 7. Chapter 8 provides a conclusion of the topics addressed and delivers an outlook on further research.

1.4 Example of Use in Terms of Case Law

In this section we will exemplify how an argumentation framework can be used to formalize and solve real legal cases. In the description of case law and argumentation frameworks we follow [9,11,8]. In the USA, the judgment is based on previous decisions. Therefore, the case law system is a very good example of the usage of an argumentation framework. The lawyers compare the current case with previous ones, in order to argue in favor of their clients.

First we consider an example, concerning a case about the right to hunt wild animals. We show how an argumentation framework is set up, and how a set of acceptable arguments is obtained.

Then we consider a more complex example, concerning the possession of a baseball. It will be seen that a framework for a real case can become quite big and complicated, so that the winner is not obvious any more. With this example, we show the need of a computer-aided system to establish acceptable arguments. Furthermore, the need of an extension of the basic framework by values and preferences is demonstrated in order to get to a reasonable decision.

Wild Animals

The example in [9] is based on wild animal cases. We will consider one of those cases, namely *Keeble versus Hickergill in 1970.* This was an English case in which Keeble owned a duck pond to which he lured ducks, which he shot and sold for consumption. Hickergill scared the ducks away by firing guns. The court found for Keeble.
To represent a case as an argumentation framework, we need to decide what counts as an argument. Arguments need to have a conclusion and a reason for that conclusion. Arguments will attack one another if their conclusions are in conflict. They will also attack one another if the conclusion of the attacker indicates that the reason does not hold, or if it suggests that the reason is no reason for the conclusion.
Now we will build the argumentation framework for the case Keeble versus Hickergill. The basic contention of Keeble is that he had a right to the ducks, and that Hickergill had prevented him from exercising this right. This is argument (A).
Hickergill must attack this argument, which he can do by saying that Keeble had no right to the animals since he was not in possession of them (B). Keeble may respond in one of two ways. He could argue that he is in pursuit of his livelihood, and that he should be free to do so without interference, especially since his activity is socially useful in providing food for others (F). Here the attack says the reason advanced in B is no reason for its conclusion.
Alternatively Keeble could argue that he possesses the animals through his possession of the land on which he would, in the absence of interference, find them (C). Hickergill may attack this latter argument by pointing out that the ducks were wild animals, and were not confined to Keeble´s land (D).
To this Keeble may respond that he had made efforts to attract the ducks through the use of decoys, and his efforts promised in the absence of interference, success (E). That is, although the ducks could be scared away, it was reasonable for the owner of the pond to expect ducks to be there, and so was something he could expect to enjoy in consequence of owning the pond. The table in Figure 1.3 lists the arguments and their attacks. In the shown figure, the graph represents the corresponding attack relations.

In the resulting AF, one can see that the arguments E and F are not attacked, hence they need to be part of the acceptable set. Hence, we can conclude that also A needs to be acceptable, because his attack from B is is defended by E and F. Furthermore, the argument E attacks the only attacker of C. It follows that the acceptable set is {A,C,E,F}, a straightforward win for Keeble.

ID	Argument	Attacks
A	Pursuer had right to animal	
B	Pursuer not in possession	A
C	Owns the land so possesses animals	B
D	Animals not confined by owner	C
E	Effort promising success made to secure animal made by pursuer	B,D
F	Pursuer has right to pursue livelihood	B

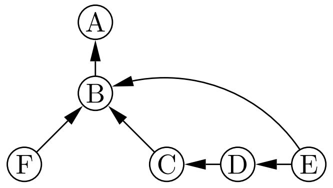


Figure 1.3: The attack relation for the case Keeble versus Hickergill.

In the next subsection, we will explain the case *Popov v. Hayashi*, based on the analysis of [51], and show how it can be formalized as an argumentation framework. We will not get into the details on which facts and rules the arguments and attacks are based on. We are more interested in the decision made by the judge and how this decision can be logically interpreted. For the more interested reader we refer to [51].

Who Owns the Baseball?

The case Popov v. Hayashi is a real case and has been decided by the honourable Kevin McCarthy in 2002. The case concerned the possession of the baseball which Barry Bonds hit for his record breaking 73rd home run in the 2001 season. Such a ball is very valuable (Mark McGwire's 1998 70th home run ball sold at auction for $3,000,000). When the ball was struck into the crowd, Popov caught it in the upper part of the webbing of his baseball glove. Such a catch, a *snowcone catch* because the ball is not fully in the mitt, does not give certainty of retaining control of the ball, particularly since Popov was stretching and may have fallen. However, Popov was not given the chance to complete his catch since, as it entered his glove, he was tackled and thrown to the ground by others trying to secure the ball, which became dislodged from his glove. Hayashi (himself innocent of the attack on Popov), then picked up the ball and put it in his pocket, so securing possession.

The Arguments

The arguments for this case arise out of testimonies of witnesses and videotape footage and the reasoning involved. The arguments are formulated as follows:
It is a general principle that the fans have the right to take home any baseball. Popov caught the baseball, it follows that he has possession of the baseball (A1). But Hayashi retrieved the ball, so it follows that he has possession of the ball too (A2). If someone catches a ball, he is the owner of it, so that no one has the right to assume possession (A3). The baseball was in the glove of Popov, which means that he caught the ball (A4). But Popov was not in control of the ball (A5), otherwise it could not have been possible to lose the ball. Popov was still in motion (A6). He lost contact with the ball, which means that he had no control of the baseball (A7).
The reason why Popov lost control of the ball was, because he has been assaulted, and so the contact was intentional on the part of other persons (A8, A9). Popov was ably and actively engaged in establishing control (A10). As the baseball was in the glove of Popov, he was ably and actively engaged in establishing control (A11).
McCarthy argued that the custom and practice of the stands creates a reasonable expectation that a person will achieve full control of a ball before claiming possession (A12). He also considered that Popov, who was ably and actively engaged in establishing control of the ball, had a pre-possessory interest in it, which precludes Hayashi from establishing control by retrieving the ball (A13).
Additional to the arguments based on video and witness testimony, we have four questions which are no observable, and so have to be inferred on the basis of what was observed. Thus whether the ball was caught (Q1), whether Popov was not in control (Q2), whether he was actively attempting to establish control (Q3) and whether the contact was intentional (Q4) are all open to question. These four arguments are intended to represent positive answers to the questions: if the argument is not defeated, the contrary has not been shown.
Table 1.1 shows the arguments and basic questions for the case, and Figure 1.4 pictures the corresponding argument graph.

Analysis

From Table 1.1 and Figure 1.4 it is obvious that the arguments A9, A11 and A12 are not attacked and therefore need to be accepted. The argument A9 eliminates the possibility that the contact was incidental, therefore A8 can defeat A7. Hence it has not been shown that Popov was not in control of the baseball. From A11 it follows that Popov was actively and ably engaged in establishing control. But A12 eliminates A10 and it follows that Popov was still in motion and were not able to complete the catch. Thus A1 and A3 are defeated and it seems that A2 survives, giving possession to Hayashi. But the fact that Popov, who was ably and actively engaged in establishing control of the ball, had a pre-possessory interest in it, defeats A2.

ID	Argument	Attacks
A1	Popov has possession of the baseball	A2
A2	Hayashi has possession of the baseball	A1
A3	Popov caught the baseball	A2
A4	The baseball was in the glove of Popov	Q1
A5	Popov has not been in complete control of the baseball	A4
A6	Popov was still in motion	A4
A7	Popov lost contact with the baseball	Q2
A8	Contact on Popov was intentional on the part of Y	A7
A9	Y assaulted Popov	Q3
A10	Popov was ably and actively engaged in establishing control	A5, A6
A11	The baseball was in the glove of Popov	Q4
A12	The custom and practice of the stands in baseball	A10
A13	Popov had a pre-possessory interest in the ball	A2
Q1	Not caught?	A1, A3
Q2	In control?	A5
Q3	Contact incidental?	A8
Q4	Not active?	A10, A13

Table 1.1: Arguments and basic question for Popov v. Hayashi.

The result of this is, that neither Popov nor Hayashi had established possession of the ball. Therefore McCarthy had to make a decision which is fair to both parties.

Argumentation Framework

In order to be able to make a fair decision, McCarthy considered the following arguments on which he also assigned different values:

PR1: Where interruption of completing the catch so establishing possession was illegal; decide for Popov; to prevent assault being rewarded; promoting the value of public order.

PR2: Where it has not been shown that Hayashi did not have possession and did nothing wrong; do not decide for Popov; which would punish Hayashi; demoting the value of fairness.

PR3: Where Hayashi had unequivocal control of the baseball; decide for Hayashi; to provide a bright line; promoting clarity of law.

PR4: Where interruption of completing the catch so establishing possession was illegal; do not insist on unequivocal control; which would reward assault; demoting the value of public order.

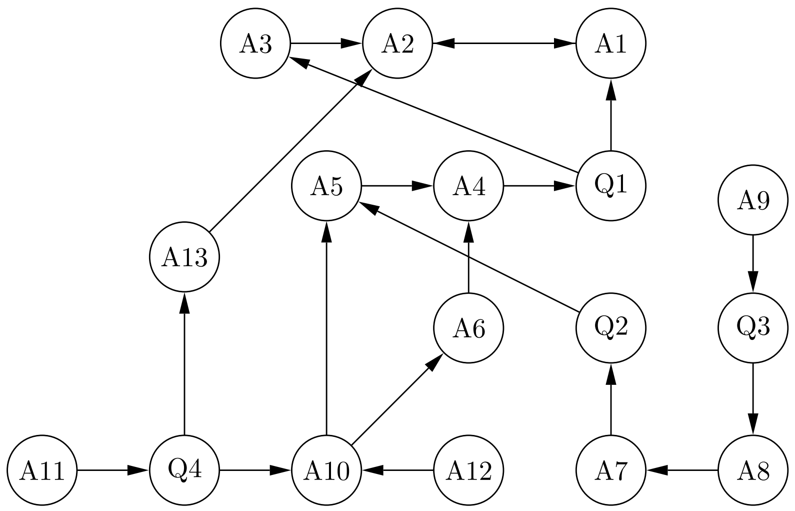


Figure 1.4: Argument Graph for Popov v. Hayashi.

PR5: Since Hayashi was not an assailant, finding for Hayashi would not reward assault.

PR6: Where it has not been shown that Popov did not have possession and did nothing wrong; do not decide for Hayashi; which would punish Popov; demoting the value of fairness.

PR7: Where interruption of completing the catch so establishing possession was illegal; Popov should sue the assailants of the assault; which would not punish Popov; promoting the value of fairness.

PR8: Since assailants cannot be identified, suing those responsible for the assault is not a viable action.

Figure 1.5 shows the corresponding Argument Graph.

In order to be able to evaluate the acceptability of the arguments, we need to specify a value ranking. McCarthy has explicitly said that fairness will be his primary value. Thus, both PR1 and PR3 are defeated. PR5 will defeat PR4, and PR8 will defeat PR7, since both must be objectively accepted given the facts. This leaves us with PR2 and PR6 which are not in conflict.

The result is that McCarthy can decide neither for Popov nor for Hayashi. Therefore, he decided that the ball should be sold and the proceeds divided between the two.

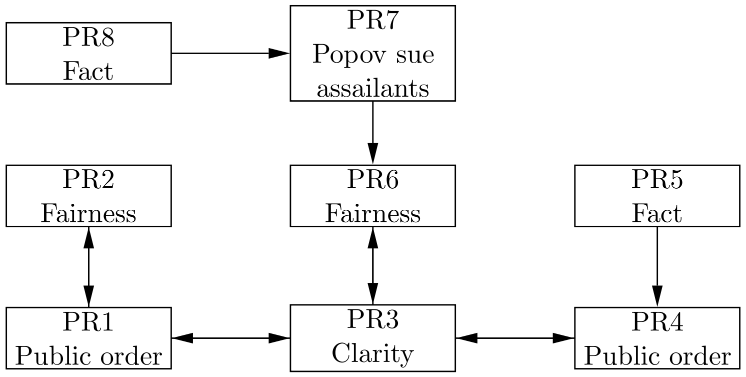


Figure 1.5: Value-based Argument Graph for Popov v. Hayashi.

Conclusion of the Example

With this example we illustrated how useful an argumentation framework can be for finding and/or analysing a decision. Therefore we used different kinds of approaches such as basic argumentation frameworks and value based argumentation frameworks, which will be introduced in the following chapters. In order to be able to reason about those different argumentation frameworks, different semantics of acceptable arguments are defined which will also be introduced and discussed. Furthermore a program has been implemented in the ASP language DLV which is able to compute all those semantics for a variety of different argumentation frameworks.

Chapter 2

Theory and Conceptual Formulation

In this chapter, we introduce different argumentation frameworks and discuss possible semantics from the literature. We present basic definitions and notations which are needed for the thesis.

2.1 Basic Argumentation Framework

We introduce basic argumentation frameworks and discuss some semantics for them. The exposition (and especially the definitions) are based on [25].

Definition 2.1.1. *An* **argumentation framework** *AF is a pair (A, R) where A is a set of arguments and R is a binary relation on A, i.e., $R \subseteq A \times A$. For two arguments a, b, the statement $(a, b) \in R$ means that a attacks b. A set $S \subseteq A$ of arguments attacks b, if b is attacked by an argument in S. A set S of arguments attacks a set S' of arguments, if there is an argument $a \in S$ which attacks an argument $b \in S'$.*

An argumentation framework $AF = (A, R)$ can be represented as a directed graph $G = (V, E)$. The set of vertices is A, and R is the set of directed edges.

Example 2.1.1. *Let (A, R) be an argumentation framework with*

$$\begin{aligned} A &= \{a, b, c, d, e\}, \\ R &= \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\}. \end{aligned}$$

This argumentation framework AF is represented by the graph shown in Figure 2.1.

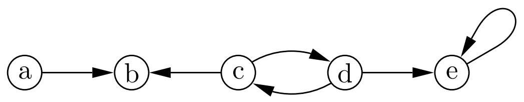


Figure 2.1: Graph of the respective $AF = (A, R)$ from Example 2.1.1.

2.1.1 Acceptability

In order to be able to perform reasoning about an argumentation framework, it is necessary to group arguments with special properties to *extensions*. One of the basic properties of an extension is that the arguments are not in conflict with each other. But avoiding conflicts alone is not very significant, therefore arguments should also be able to reject arguments that are outside the extension. This leads to the definition of stable extensions.

Definition 2.1.2. *Let $AF = (A, R)$. A set $S \subseteq A$ is said to be* **conflict-free** *(in AF), if there are no arguments $a, b \in S$ such that $(a, b) \in R$. We denote by $cf(AF)$ the set of all conflict-free sets of AF.*
A conflict-free set $S \subseteq A$ is a **stable extension** *of AF, iff for each argument $c \in A$ with $c \notin S$, there exists an argument $b \in S$ such that $(b, c) \in R$. We denote by $stable(AF)$ the set of all stable extensions of AF.*

For the framework AF of Example 2.1.1, we obtain $stable(AF) = \{\{a, d\}\}$. Indeed $\{a, d\}$ is conflict-free, since a and d are not adjacent. Moreover, each further element b, c, e is attacked by either a or d. In turn, $\{a, c\}$ for instance is not contained in $stable(AF)$, although it is clearly conflict free. Since e is not attacked by $\{a, c\}$, the latter is not an element of $stable(AF)$.

It is possible that an argumentation framework does not have a stable extension. Another possibility to achieve an acceptable set of arguments is to consider those arguments which are able to defend themselves against all external attacks. This is the basis of the notion of admissible sets.

Definition 2.1.3. *Let $AF = (A, R)$. An argument $a \in A$ is* **defended** *by a set $S \subseteq A$ of arguments (in AF), iff for each argument $b \in A$, it holds that, if $(b, a) \in R$, then b is attacked by S. A conflict-free set $S \subseteq A$ is* **admissible** *(in AF), iff each argument in S is defended by S in AF. We denote by $adm(AF)$ the set of all admissible extensions of AF.*

For the argumentation framework AF of Example 2.1.1, we obtain $adm(AF) = \{\{\}, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}\}$. By definition, the empty set is always an admissible extension, and an argumentation framework can have several admissible extensions, therefore reasoning over admissible extensions is limited. The maximal admissible sets, called *preferred extensions*, are considered to be of more interest.

Definition 2.1.4. *Let $AF = (A, R)$. A* **preferred extension** *of AF is a maximal (with respect to set inclusion) admissible set of AF. We denote by $pref(AF)$ the set of all preferred extensions of AF.*

For the argumentation framework AF of Example 2.1.1, we obtain $pref(AF) = \{\{a, c\}, \{a, d\}\}$. Beside the maximal admissible sets, one might be interested in admissible sets which just defend those arguments which belong to them. This expresses a kind of completeness.

Definition 2.1.5. *Let $AF = (A, R)$. An admissible set $S \subseteq A$ is called a* **complete extension** *of AF, iff each argument, which is defended by S (in AF), belongs to S. We denote by $comp(AF)$ the set of all complete extensions of AF.*

For the argumentation framework AF of Example 2.1.1, we obtain

$$comp(AF) = \{\{a\}, \{a, c\}, \{a, d\}\}.$$

In contrast to the preferred extensions, which are characterized by maximality, the *grounded extension* is determined by the minimal complete extension.

Definition 2.1.6. *Let $AF = (A, R)$. The least (with respect to set inclusion) complete extension of AF is the* **grounded extension**. *We denote by $ground(AF)$ the grounded extension of AF.*

The grounded extension for the argumentation framework of Example 2.1.1 is given by $ground(AF) = \{a\}$. As mentioned above, the preferred extensions are of particular interest. The preferred extensions of an argumentation framework AF can be taken as being the consistent positions that can be adopted within AF. Any argument that appears in all preferred extensions will be acceptable in every consistent position, and any argument that appears in no preferred extension cannot be held in any consistent position. This means that the notion can be related to varieties of semantics of non-monotonic reasoning: credulously acceptable arguments will be those that appear in at least one preferred extension, and sceptically acceptable arguments will be those which appear in all preferred extensions. This notion allows us to distinguish those arguments which must be accepted, those which can be defended, and those which are indefensible [27].

Since any admissible set is a subset of some preferred extension, credulous acceptance of an argument a is ensured by finding any admissible (not necessarily maximal) set containing a.

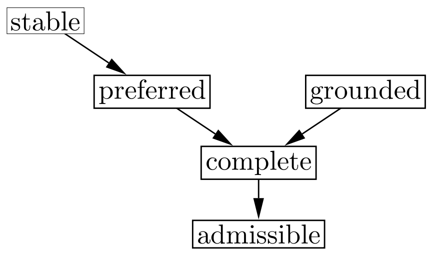


Figure 2.2: Overview of relations.

2.1.2 Properties of Acceptability

As mentioned so far, there exists a variety of different notions of acceptability, but in some cases these semantics are not totally different. It happens quite often that certain extensions from diverse semantics coincide [5]. Furthermore, there can be drawn conclusions from the structure (in particular cycles) of the argumentation framework to the acceptability. A lot of those properties have been discussed in literature. Here we give an outline of the most important ones.

Property 2.1.1. *Every stable extension is a preferred extension, but not vice versa. Moreover, every preferred extension is also a complete extension. Stable, preferred and complete semantics can have multiple extensions whereas the grounded semantics just has a single extension.*

The relations between the semantics are depicted in Figure 2.2, where an arrow from e to f indicates that each e-extension is also a f-extension.

The next property follows from the fact that the empty set is always admissible.

Property 2.1.2. *Every argumentation framework possesses at least one preferred extension.*

Property 2.1.3. *Each admissible set is included in a preferred extension. The grounded extension is included in each preferred extension. Each argument which is not attacked belongs to the grounded extension (hence to each preferred and to each stable extension).*

An argumentation framework can not have more than one grounded extension. Therefore, an alternative definition is possible which is based on a fix-point operator:

Definition 2.1.7. *The grounded extension of an argumentation framework $AF = (A, R)$ is given by the least fix-point of the operator $\Gamma_{AF} : 2^A \to 2^A$, defined as*

$$\Gamma_{AF}(S) = \{a \in A \mid a \textit{ is defended by } S \textit{ in } AF\}.$$

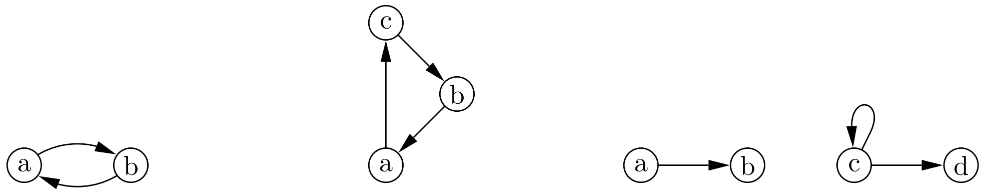


Figure 2.3: Graphs for the respective argumentation frameworks AF_1, AF_2 and AF_3 from Example 2.1.2.

In [10, 14, 25, 27] the relation between preferred extensions and cycles in the graph representation have been analysed.

Property 2.1.4. *If an argumentation framework has (at least) two different preferred extensions, then the directed graph of AF contains a directed cycle of even length.*

Property 2.1.5. *If an argumentation framework has no cycle of even length, then AF has a single preferred extension.*

As an illustration consider the following examples.

Example 2.1.2. *Consider the argumentation frameworks from Figure 2.3. The argumentation framework AF_1 from the graph on the left side as one even cycle and hence $pref(AF_1) = \{\{a\}, \{b\}\}$. For AF_2 from the graph in the middle with one odd cycle we obtain $pref(AF_2) = \{\{\}\}$, and for AF_3 from the graph on the right side we obtain $pref(AF_3) = \{\{a\}\}$.*

The argumentation framework AF_2 from Example 2.1.2 also exemplifies the following property.

Property 2.1.6. *Let $AF = (A, R)$ be an argumentation framework with no cycle of even length. If each vertex in A is the endpoint of an edge in R, then the only preferred extension of AF is the empty set.*

From cycles, we can also draw conclusions whether preferred and stable extensions coincide.

Property 2.1.7. *If an argumentation framework has no cycle of odd length, then each preferred extension of AF is a stable extension of AF.*

Example 2.1.3. *Consider the argumentation framework from Figure 2.4. We get $pref(AF) = \{\{a\}, \{b\}, \{c\}\}$, and these three extensions are also stable extensions.*

Property 2.1.8. *If an argumentation framework has no cycle and A is finite, then AF has a single extension. It is stable, preferred, complete, and grounded.*

Example 2.1.4. *Consider the argumentation framework from Figure 2.5.*
Since AF has no cycle, $\{a, c\}$ is the only extension of AF, and it is stable, preferred, complete, and grounded.

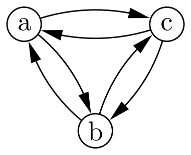


Figure 2.4: Graph of the respective *AF* from Example 2.1.3.

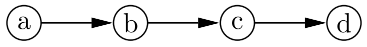
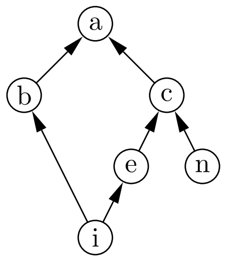


Figure 2.5: Graph of the respective *AF* from Example 2.1.4.

2.2 Handling Controversial Arguments

Sometimes it may occur that an argument a is defended by an argument b, and at the same time the argument b defends an attacker c of a. Even though there is no direct conflict between the arguments a and b, it may be incautious to accept both arguments at the same extension.

Example 2.2.1. *Let $AF = (A, R)$ with $A = \{a, b, c, e, n, i\}$ and $R = \{(b, a), (c, a), (n, c), (i, b), (e, c), (i, e)\}$.*



The argument a is defended by the argument i and at the same time i, defends the argument c which is an attacker of a. Thus it would not be cautious to accept both a and i in the same extension. But the semantics for argumentation frameworks defined so far has one preferred extension, namely $\{a, n, i\}$.

Definition 2.2.1. *Let $AF = (A, R)$. An argument $a \in A$* **indirectly attacks** *an argument $b \in A$, iff there exists a finite sequence (of odd length) $a_0, \ldots, a_{2n+1}$ such that $b = a_0$ and $a = a_{2n+1}$, and for each i, $0 \leq i \leq 2n$, a_{i+1} attacks a_i.*
An argument a **indirectly defends** *an argument b, iff there exists a sequence (of even length) $a_0, \ldots, a_{2n}$, such that $b = a_0$ and $a = a_{2n}$, and for each i, $0 \leq i \leq 2n$, a_{i+1} attacks a_i.*
An argument a is **controversial** *with respect to an argument b, iff a indirectly attacks b and indirectly defends b.*

2.2.1 Prudent Semantics for Argumentation Frameworks

In order to handle controversial arguments, the prudent semantics have been introduced in [20].

Definition 2.2.2. *Let $AF = (A, R)$. A set $S \subseteq A$ is* **p(rudend)-conflict-free** *(in AF), iff there are no arguments $a, b \in S$ such that a indirectly attacks b. We denote by $pcf(AF)$ the set of all p-conflict-free sets of AF. A p-conflict-free set $S \subseteq A$ is a* **stable p-extension** *of AF, iff for each argument $c \notin S$ an argument $b \in S$ exists that directly attacks c. We denote by $pstable(AF)$ the set of all stable p-extensions of AF.*

The argumentation framework AF of Example 2.2.1 has no stable p-extension.

Definition 2.2.3. *Let $AF = (A, R)$. A set $S \subseteq A$ is said to be* **p-admissible** *(in AF), iff every $a \in S$ is defended by S and for all $a, b \in S$, a does not indirectly attack b. We denote by $padm(AF)$ the set of all p-admissible sets of AF.*

The argumentation framework AF of Example 2.2.1 has $\{i, n\}$ and its subsets as p-admissible sets of AF.

Definition 2.2.4. *Let $AF = (A, R)$. A* **preferred p-extension** *of AF is a maximal (with respect to set inclusion) p-admissible extension of AF. We denote by $ppref(AF)$ the set of all preferred p-extensions of AF.*

In AF from Example 2.2.1, $\{i, n\}$ is the unique preferred p-extension of AF.

Definition 2.2.5. *Let $AF = (A, R)$. An admissible set $S \subseteq A$ is called a* **complete p-extension** *of AF, iff each argument which is defended by S (in AF) and without indirect conflicts with S (in AF), belongs to S. We denote by $pcomp(AF)$ the set of all complete p-extensions of AF.*

Definition 2.2.6. *Let $AF = (A, R)$. The least (with respect to set inclusion) complete p-extension of AF is the* **grounded p-extension** *of AF. We denote by $pground(AF)$ the set of all grounded p-extensions of AF.*

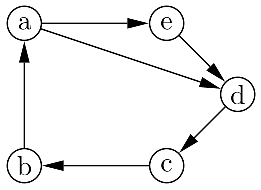
2.2.2 Properties of Prudent Semantics

From [20] we can conclude some properties of prudent semantics.

Property 2.2.1. *Let $AF = (A, R)$ and $a, b \in A$. If a is controversial with respect to b, then $\{a, b\}$ can not be included into any p-admissible set for AF.*

This means that no arguments belonging to an odd-length cycle of AF can also belong to a p-admissible set. Especially, it is not cautious to consider within a single extension the arguments of an odd-length cycle since they attack themselves indirectly. Furthermore, any argument from an odd-length cycle is controversial with respect to an argument of the cycle.

Example 2.2.2. *Consider the following $AF = (A, R)$. Let $A = \{a, b, c, d, e\}$ and $R = \{(a, e), (a, d), (b, a), (c, b), (d, c), (e, d)\}$.*



The argument a is controversial with respect to argument c because a attacks both the defender e and the attacker d of c. Hence $\{a, c\}$ can not belong to any p-admissible set of AF. Note that $\{a, c\} \in adm(AF)$.

Some of the previous mentioned properties from argumentation frameworks also hold for prudent semantics in a similar form.

Property 2.2.2. *Every stable p-extension of an argumentation framework AF also is a preferred p-extension of AF. The converse does not hold.*

Property 2.2.3. *Let $AF = (A, R)$. If AF is acyclic, then the grounded p-extension of AF is nonempty.*

Thus, every argumentation framework has at least one preferred p-extension, a unique grounded p-extension and zero, one or many stable p-extension. While the grounded extension of an argumentation framework AF is included in the intersection of all complete extensions of AF, it is in general not the case that the grounded p-extension of AF is included in every preferred p-extension of AF.

Property 2.2.4. *Let $AF = (A, R)$. If AF has a stable p-extension, then the intersection of all preferred p-extensions of AF is included into the grounded p-extension of AF.*

2.3 Extensions of Basic Argumentation Frameworks

The basic argumentation frameworks provide various semantics for acceptability, but there are still several limitations. All arguments have the same priority, but in daily-life discussions this is not always the case. There exists a requirement for preference relations or for the possibility to add values to the arguments. Furthermore, it is only possible to represent a support implicitly via defended arguments, but we may be interested in a relation which is independent of the attack relation. Therefore we present some extensions of the basic argumentation framework which cover these requirements.

2.3.1 Preference Based Argumentation Frameworks

In preference based argumentation frameworks, we have the possibility to define additionally to the attack relation preference relations between the arguments. This allows us to prefer and compare particular arguments.

Example 2.3.1. *Let $PAF = (A, R)$. Let $A = \{a, b, c\}$ and $R = \{(b, a), (c, b)\}$.*

$$\begin{tikzcd} c \arrow[r] & b \arrow[r] & a \end{tikzcd}$$

According to basic argumentation frameworks, we obtain $pref(PAF) = \{\{a, c\}\}$. But if we know that the argument b is preferred to the argument c, then b defends itself against c in some sense, and we obtain $pref(PAF) = \{\{b, c\}\}$.

In order to define preference based argumentation frameworks, we need the concept of preorders.

Definition 2.3.1. *A binary relation $\lesssim$ defined on a set S is a* **preordering** *if it is reflexive and transitive, i.e., $\forall a, b, c \in S$: $a \lesssim a$ (reflexive); if $a \lesssim b$ and $b \lesssim c$ then $a \lesssim c$ (transitive).*

The following definitions are based on [1].

Definition 2.3.2. *A* **preference based argumentation framework (PAF)** *is a triplet $(A, R, Pref)$ where A and R are as for a standard argumentation framework and $Pref$ is a (partial or complete) preordering on $A \times A$.*

Different definitions of the relations R and $Pref$ lead to different preference-based argumentation systems.

Definition 2.3.3. *Let $PAF = (A, R, Pref)$. An argument $a \in A$* **defeats** *an argument $b \in A$ iff both $(a, b) \in R$ and $(b, a) \notin Pref$.*

Acceptability in PAFs

The notions for acceptable sets of arguments for preference based argumentation frameworks remain the same as for basic argumentation frameworks, except that we consider defeat relations instead of attack relations.

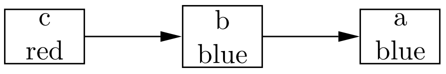
2.3.2 Value Based Argumentation Frameworks

A value based argumentation framework is similar to a preference based argumentation framework, except that we can assign values to the arguments and define an ordering over those values. Remember the introductory example *Popov v. Hayashi* from Section 1.4, where at first the judge was not able to make a decision. Only after he extended the argumentation framework to a value based argumentation framework, he was able to pronounce a judgement. In the example, we just presented a verbal reasoning for the judgement. Now we will introduce the necessary definitions, following [7].

Definition 2.3.4. *A* **value based argumentation framework (VAF)** *is a 5-tuple* $(A, R, V, val, valpref)$*, where* A *and* R *are as for a standard argumentation framework,* V *is a non-empty set of values,* val *is a function which maps from elements of* A *to elements of* V*, and* $valpref \subseteq V \times V$ *is a preference relation (transitive, irreflexive and asymmetric). We say that an argument* $a \in A$ *relates to value* $v \in V$ *if accepting* a *promotes or defends* v*: the value in question is given by* $val(a)$*. For every* $a \in A, val(a) \in V$*.*

Definition 2.3.5. *Let* $VAF = (A, R, V, val, valpref)$*. An argument* $a \in A$ **defeats** *an argument* $b \in A$ *iff both* $(a, b) \in R$ *and* $(val(b), val(a)) \notin valpref$ *hold.*

Example 2.3.2. *Consider the following* $VAF = (A, R, V, val, valpref)$*. Let* $A = \{a, b, c\}$*,* $R = \{(b, a), (c, b)\}$*,* $V = \{red, blue\}$*,* $val(a) = blue$*,* $val(b) = blue$*,* $val(c) = red$ *and* $valpref = \{(red, blue)\}$*.*



We obtain that c *defeats* b*, because* $(c, b) \in R$ *and* $(val(b), val(c)) \notin valpref$ *holds. But, for* $valpref(blue, red)$*, this does not hold any longer.*

Acceptability in VAFs

In order to exemplify the notions of acceptance in value based argumentation frameworks, let us consider the following example from [10].

Example 2.3.3. *Hal, a diabetic, loses his insulin in an accident through no fault of his own. Before collapsing into a coma, he rushes to the house of Carla, another diabetic. She is not at home, but Hal enters her house and uses some of her insulin. Was Hal justified, and does Carla have a right to compensation?*
The first argument A *is that Hal is justified, since a person has a privilege to use the property of others to save his/her life. The second argument* B *is that it is wrong to infringe the property rights of another. The argument* C *denotes that Carla's rights have*

not been infringed. Argument D stands for the fact that if Hal were to poor to compensate Carla, he should none the less be allowed to take the insulin. Argument E denotes that poverty is no defence for theft. The last argument F means that Hal is endangering Carla's life. Arguments A, D and F are based on the value that life is important (life). Whereas the arguments B, C and E are based on the value that property owners should be able to enjoy their property (property). Figure 2.3.3 shows the corresponding graph.

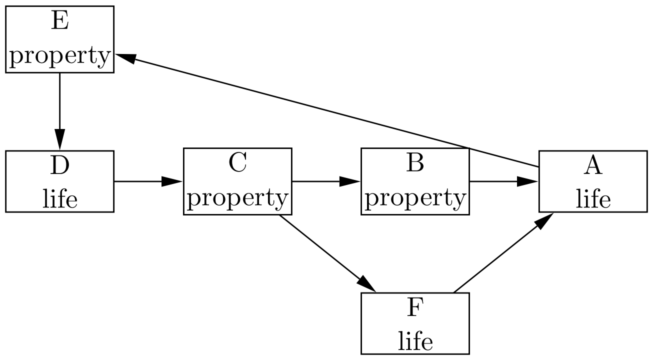


Figure 2.6: Corresponding graph for VAF from Example 2.3.3.

We assume that life has a higher value than property.

Definition 2.3.6. *Let $VAF = (A, R, V, val, valpref)$. An argument $a \in A$ is* **defended** *by a set $S \subseteq A$ of arguments, iff for each argument $b \in A$, it holds that, if b defeats a, then b is defeated by S.*

Definition 2.3.7. *Let $VAF = (A, R, V, val, valpref)$. A set $S \subseteq A$ of arguments is said to be* **conflict-free***, if there are no two arguments $a, b \in S$ such that $(a, b) \in R$ and $(val(b), val(a)) \in valpref$. We denote by $cf(VAF)$ the set of all conflict-free sets of VAF.*

Definition 2.3.8. *Let $VAF = (A, R, V, val, valpref)$. A conflict-free set of arguments $S \subseteq A$ is* **admissible***, iff each argument in S is defended by S. We denote by $adm(VAF)$ the set of all admissible sets of VAF.*

Some admissible extensions of the VAF from Example 2.3.3 are:

$$\{B, D\}, \{D, E, F\}, \{B, D, F\}, \{B, D, E, F\}.$$

Definition 2.3.9. *Let $VAF = (A, R, V, val, valpref)$. A set $S \subseteq A$ of arguments is a* **preferred extension** *of VAF if it is a maximal (with respect to set inclusion) admissible set of VAF. We denote by $pref(VAF)$ the set of all preferred extensions of VAF.*

The preferred extension from the VAF from Example 2.3.3 is $\{B, D, E, F\}$. If we value property more than life, the preferred extension is $\{A, C, E\}$.

Definition 2.3.10. *Let $VAF = (A, R, V, val, valpref)$. A conflict-free set of arguments $S \subseteq A$ is a* **stable extension** *of VAF, iff for each argument $c \in A$, with $c \notin S$, an argument $b \in S$ exists such that b defeats a. We denote by $stable(VAF)$ the set of all stable extensions of VAF.*

The stable extension for the VAF from Example 2.3.3 equals the preferred extension.

Properties of Acceptability in VAFs

By the extension of an argumentation framework with values, the acceptance of arguments may change significantly depending on the ranking of values.

Consider the VAF from Example 2.3.2. If we prefer *red* to *blue*, we obtain $\{a, c\}$ as the preferred extension. But, if we prefer *blue* to *red*, we obtain $\{b, c\}$ as the preferred extension. The argument a, which is sceptically acceptable in a value free framework, has been rejected by an alteration of priorities, whereas the argument c remains acceptable independent of the value ranking. This leads to two different notions of acceptance, the objective and the subjective acceptance [10].

Definition 2.3.11 (Objective Acceptance)**.** *Given a $VAF = \{A, R, V, val, valpref\}$, an argument $a \in A$ is* **objectively acceptable** *if and only if a is contained in every preferred extension for all possible value preferences.*

Definition 2.3.12 (Subjective Acceptance)**.** *Given a $VAF = \{A, R, V, val, valpref\}$, an argument $a \in A$ is* **subjectively acceptable** *if and only if a is contained in some preferred extensions for some value preferences.*

An argument which is neither objectively nor subjectively acceptable is said to be **indefensible**. In [7] several properties of value based argumentation frameworks with respect to cycles have been analysed. A cycle in a VAF is *monochromatic* if it contains arguments related to a single value, *dichromatic* if it contains arguments related to exactly two arguments and *polychromatic* if they have two or more values. In a VAF, monochromatic cycles act like in an AF. But $VAFs$ with no monochromatic cycles have a unique, non-empty preferred extension, given an ordering on values. This follows from the fact that an AF related to a VAF with no monochromatic cycles is cycle free.

2.3.3 Bipolar Argumentation Frameworks

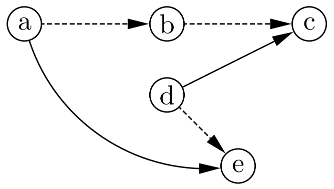
The following definitions are based on [3, 18, 19].

A bipolar argumentation framework is an extension of the basic argumentation framework introduced in [25], in which two kinds of interactions between arguments are possible: the attack relation and the support relation. These two relations are independent which leads to a bipolar representation of the interaction between arguments.

Definition 2.3.13. *An* **abstract bipolar argumentation framework (BAF)** *is a triplet $BAF = (A, R_{def}, R_{sup})$, where A is a set of arguments, R_{def} is a binary relation on A called* **defeat relation** *and R_{sup} is another binary relation on A called* **support relation**. *For two arguments a and b, $(a,b) \in R_{def}$ (resp. $(a,b) \in R_{sup}$) means that a defeats b (resp. a supports b).*

A bipolar argumentation framework can be represented by a directed graph G_b called the *bipolar interaction graph*. In order to differentiate between the two relations, two kinds of edges are used. For two arguments a and b, $(a,b) \in R_{def}$ is represented by $a \rightarrow b$ and $(a,b) \in R_{sup}$ is represented by $a \dashrightarrow b$.

Example 2.3.4. *Consider the following $BAF = (A, R_{def}, R_{sup})$. Let $A = \{a,b,c,d,e\}$, $R_{def} = \{(a,e),(d,c)\}$ and $R_{sup} = \{(a,b),(b,c),(d,e)\}$. Then the graph G_b looks as follows:*



Definition 2.3.14. *Let $BAF = (A, R_{def}, R_{sup})$. A* **supported defeat** *for an argument $b \in A$ is a sequence*

$$a_1 R_1 \dots R_{n-1} a_n,$$

$n \geq 3$, with $a_n = b$, such that $\forall i = 1, \dots, n-2$, $R_i = R_{sup}$ and $R_{n-1} = R_{def}$. A direct defeat on $b \in A$ (a sequence reduced to two arguments $a R_{def} b$) will also be called a supported defeat.
An **indirect defeat** *for an argument $b \in A$ is a sequence*

$$a_1 R_1 \dots R_{n-1} a_n,$$

$n \geq 3$, with $a_n = b$, such that $\forall i = 2, \dots, n-1$, $R_i = R_{sup}$ and $R_1 = R_{def}$.

Definition 2.3.15. *A set $S \subseteq A$* **set-defeats** *an argument $b \in A$, iff there exists a supported defeat or an indirect defeat for b from an element of S. A set $S \subseteq A$* **set-supports** *an argument $b \in A$, iff there exists a sequence*

$$a_1 R_{sup} \dots R_{sup} a_n,$$

$n \geq 2$, such that $a_n = b$ and $a_1 \in S$.

Definition 2.3.16. *A set $S \subseteq A$* **defends** *an argument $a \in A$, iff for each argument $b \in A$, if $\{b\}$ set-defeats a, then b is set-defeated by S.*

Acceptability in Bipolar Argumentation Frameworks

In the following we will define the semantics for acceptability in bipolar argumentation frameworks [19].

Definition 2.3.17. *A set $S \subseteq A$ is* **conflict-free***, iff there are no two arguments a and b in S such that $\{a\}$ set-defeats b.*

Definition 2.3.18. *A set $S \subseteq A$ is* **safe***, iff there is no argument $b \in A$ such that S set-defeats b and either S set-supports b or $b \in S$.*

Some safe sets from the BAF from Example 2.3.4 are $\{d, e\}$, $\{a, b, c\}$ and $\{b, c, e\}$.

Definition 2.3.19. *A conflict-free set $S \subseteq A$ is a* **stable extension***, iff for each argument $a \notin S$, S set-defeats a.*

The unique stable extension from the BAF from Example 2.3.4 is $\{a, b, d\}$.

We can give three different definitions for admissibility. The first one is equal to Dung's definition and is therefore called d-admissible extension.

Definition 2.3.20. *A conflict-free set $S \subseteq A$ is a* **d-admissible extension***, iff each argument in S is defended by S.*

The d-admissible extensions from the BAF from Example 2.3.4 are as follows:

$$\{\{\}, \{a\}, \{b\}, \{d\}, \{a, b\}, \{a, d\}, \{b, d\}, \{a, b, d\}\}.$$

Next we consider admissibility for safe sets.

Definition 2.3.21. *A safe set $S \subseteq A$ is a* **s-admissible extension***, iff each argument in S is defended by S.*

The s-admissible extensions from the BAF from Example 2.3.4 are as follows:

$$\{\{\}, \{a\}, \{b\}, \{d\}, \{a, b\}\}.$$

The last definition of admissibility is related to closed sets[1].

[1]A set is closed under an operator if that operator returns a member of the set when evaluated on members of the set.

Definition 2.3.22. *A conflict-free set $S \subseteq A$ is a* **c-admissible extension***, iff S is closed* [2] *for R_{sup} and each argument in S is defended by S.*

The unique c-admissible extension from the BAF from Example 2.3.4 is the empty set $\{\}$. Finally, we define preferred semantics for $BAFs$.

Definition 2.3.23. *A* **d-preferred** *(resp.* **s-preferred, c-preferred***)* **extension** *of a bipolar argumentation framework BAF is a maximal (with respect to set inclusion) d-admissible (resp. s-admissible, c-admissible) set of BAF.*

The preferred extensions from the BAF from Example 2.3.4 are as follows:

- $\{a, b, d\}$ is the unique d-preferred extension,
- $\{d\}$ and $\{a, b\}$ are the s-preferred extensions, and
- $\{\}$ is the unique c-preferred extension.

Properties of Acceptability in BAFs

The following properties show relations between the semantics from $BAFs$. The exposition here is based on [19].

Property 2.3.1. *If a set $S \subseteq A$ is safe, then S is conflict-free. If a set $S \subseteq A$ is conflict-free and closed for R_{sup}, then S is safe.*

Property 2.3.2. *Let S be a stable extension. If S is safe, then S is closed for R_{sup}.*

Consequence 1. *Let S be a stable extension of a bipolar argumentation framework $BAF = (A, R_{def}, R_{sup})$. Then S is safe iff S is closed for R_{sup}.*

Property 2.3.3. *Each c-admissible extensions is also s-admissible, and each s-admissible extension is also d-admissible.*

Property 2.3.4. *Let S be the unique stable extension of the bipolar argumentation framework $BAF = (A, R_{def}, R_{sup})$.*

1. *The s-preferred extensions and the c-preferred extensions are subsets of S.*
2. *Each s-preferred extension which is closed for R_{sup} is also a c-preferred extension.*
3. *If S is safe, then S is the unique c-preferred extension and also the unique s-preferred extension.*

[2] We remark that a set S of arguments is closed under R_{sup}, iff S is closed under the transitive closure of R_{sup}.

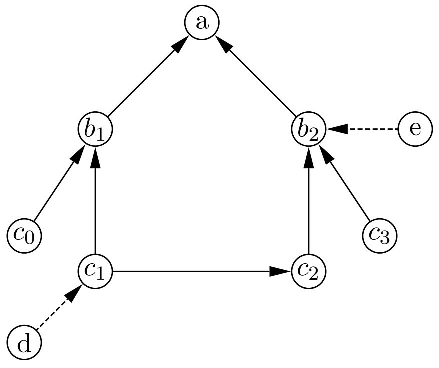


Figure 2.7: Graph of the respective $BAF = (A, R_{def}, R_{sup})$ from Example 2.3.5

4. *If A is finite, each c-preferred extension is included in a s-preferred extension.*
5. *If S is not safe, the s-preferred extensions are the subsets of S which are maximal (with respect to set inclusion) s-admissible.*
6. *If S is not safe, and A is finite, then there is only one c-preferred extension.*

Controversial Arguments in BAFs

We have already discussed controversial arguments before, but in the case of a bipolar argumentation framework, we also need to take into account the support relation. In some particular arrangements of a bipolar argumentation framework, it may happen that an argument a supports an argument b, which is controversial with respect to an argument c. In such a case, the arguments a and c are not directly controversial. Therefore *b(ipolar)-controversial* arguments have been introduced in [17].

Definition 2.3.24. *An argument $a \in A$ is* **b-controversial** *with respect to an argument $b \in A$, iff a supports (by a sequence of supports) an argument c which indirectly attacks b and a supports (by a sequence of supports) an argument d which indirectly defends b.*

Example 2.3.5. *Consider the BAF in Figure 2.7. Let $A = \{a, b_1, b_2, c_0, c_1, c_2, c_3, d, e\}$, $R_{def} = \{(b_1, a), (b_2, a), (c_0, b_1), (c_1, b_1), (c_1, c_2), (c_2, b_2), (c_3, b_2)\}$ and $R_{sup} = \{(d, c_1), (e, b_2)\}$.*

Definition 2.3.25. *A set $S \subseteq A$ is* **b(ipolar)p(prudent)-conflict-free**, *iff there are no two arguments a and b in S such that there exists a sequence*

$$a_1 R_{sup} \dots R_{sup} a_n R_{def} \dots R_{def} a_{n+m},$$

$n \geq 1$, with $a_1 = a$, $a_{n+m} = b$, and m is an odd number.

In the BAF from Example 2.3.5, the set $\{a, c_0, c_3, d\}$ does not contain indirect attacks, but it is not bp-conflict-free.

Definition 2.3.26. *A set $S \subseteq A$ is* **bp-admissible** *for a bipolar argumentation framework BAF, iff S is bp-conflict-free and each argument in S is defended by S. A* **preferred bp-extension** *of BAF is a maximal (with respect to set inclusion) bp-admissible set of BAF. A bp-conflict-free set S is a* **stable bp-extension***, iff for each argument $a \notin S$, S defeats a.*

The BAF from Example 2.3.5 has two preferred bp-extensions: $\{a, c_0, c_3\}$ and $\{c_0, c_1, c_3, d, e\}$, but no stable bp-extension.

Chapter 3

Answer Set Programming

In this chapter we introduce the basic definitions of Answer Set Programming (ASP) which are needed for the remainder of the thesis.

3.1 Syntax

Answer Set Programming has been first introduced in [40]. It is a declarative programming paradigm based on an extension of function-free first-order logic.

A *term* is constant symbol or a variable symbol.

An *atom* is an expression $A(t_1, \dots, t_n)$, where A is a predicate symbol of arity n and $t_1, \dots, t_n$ are terms.

A *literal* is an atom $A(t_1, \dots, t_n)$ or a strongly negated atom $\neg A(t_1, \dots, t_n)$, where $\neg$ stands for the classical (strong) negation. In contrast, *not* represents *negation as failure*; if an atom is not true in some model, then its negation should be considered to be true in that model.

A literal is called *ground* if it does not contain any variable symbols.

A *disjunctive logic program* Π is a set of rules r of the form

$$L_1 \vee \cdots \vee L_k \leftarrow L_{k+1}, \dots, L_m, \mathit{not}\, L_{m+1}, \dots, \mathit{not}\, L_n, \tag{3.1}$$

where $n \geq m \geq k \geq 0$ and L_i $(i = 1, \dots, n)$ is a literal. The *head* of the rule r is the set $H(r) = \{L_1, \dots, L_k\}$, and the *body* of r is $B(r) = \{L_{k+1}, \dots, L_m, \mathit{not}\, L_{m+1}, \dots, \mathit{not}\, L_n\}$. Furthermore, we denote the *positive body* of r by $B^+(r) = \{L_{k+1}, \dots, L_m\}$ and the *negative body* of r by $B^-(r) = \{L_{m+1}, \dots, L_n\}$.

A rule is *safe*[1] if each variable in the head of the rule also appears in at least one literal in the positive body of that rule. A program is safe if each of its rules is safe, and in the following we will only consider safe programs.

An *integrity constraint* is a rule with empty head, whereas a *fact* is a non-disjunctive rule with empty body.

A rule is called *ground* if it contains only ground literals, and a program is called *ground program* if it only contains ground rules.

Occasionally, we write for a set F of facts (which is considered to represent some term of input) and a program Π, $\Pi(F)$ instead of $F \cup \Pi$.

A program Π is called *stratified*, if there exists an assignment $s(\cdot)$ of integers to the predicates in Π, such that for each rule $r \in \Pi$, the following holds: If predicate p occurs in the head of r and predicate q occurs

(i) in the positive body of r, then $s(p) \geq s(q)$ holds;

(ii) in the negative body of r, then $s(p) > s(q)$ holds.

The *Herbrand universe* $\mathcal{HU}_\Pi$ of a program Π is the set of all constant symbols occurring in Π. A *ground instance* of a rule $r \in \Pi$ is a ground rule r' where each variable occurring in r is substituted by an element of $\mathcal{HU}_\Pi$. In the following, we will denote with $\mathcal{G}r(\Pi)$ the ground instance of the program Π i.e. the set of all ground instances of rules in Π.

Example 3.1.1. *Consider the program* Π*:*

$$\begin{aligned} \Pi \quad = \{ \quad & p(1,2); \\ & q(x) \leftarrow p(x,y), \neg q(y)\}. \end{aligned}$$

Its Herbrand universe $\mathcal{HU}_\Pi$ *is* $\{1,2\}$*. The ground instance of the program* Π *is as follows:*

$$\begin{aligned} \mathcal{G}r(\Pi) \quad = \{ \quad & q(1) \leftarrow p(1,1), \neg q(1); \\ & q(1) \leftarrow p(1,2), \neg q(2); \\ & q(2) \leftarrow p(2,1), \neg q(1); \\ & q(2) \leftarrow p(2,2), \neg q(2)\}. \end{aligned}$$

3.2 Semantics

The semantics of logic programs is an extension of the answer set semantics for general logic programs proposed in [39]. The stable model semantics defines, when a set S of ground atoms is a “stable model” of a given program.

[1]Note that *safe* in the context of ASP has no bearing on *safe* in the context of BAFs from Section 2.3.3

The semantics of extended programs treat a rule with variables as shorthand for the set of its ground instances. It is sufficient then to define answer sets for extended programs without variables. This will be done in two steps.

First we consider programs without *not* (i.e., $m = n$ in every rule of the form (3.1) of the program).

Definition 3.2.1. *Let Π be a ground disjunctive logic program without variables that does not contain not, and let Lit be the set of ground literals over $\mathcal{HU}_\Pi$. An* answer set *of Π is any minimal subset S of Lit such that,*

(i) for each rule $L_1 \vee \cdots \vee L_k \leftarrow L_{k+1}, \ldots, L_m$ from Π, if $L_{k+1}, \ldots, L_m \in S$, then, for some $i = 1, \ldots, k$, $L_i \in S$;

(ii) if S contains a pair of complementary literals, then $S = Lit$.

We will denote the set of answer sets of a program Π by $AS(\Pi)$.

Example 3.2.1. *Consider the following program Π:*

$$\begin{aligned}\{\quad & man(tom);\\ & single \vee husband \leftarrow man(X)\}.\end{aligned}$$

The program Π has two answer sets $\{man(tom), husband\}$ and $\{man(tom), single\}$.

Now we define a method to obtain the answer sets of a program containing negation.

Definition 3.2.2. *Let Π be a ground disjunctive logic program. The Gelfond-Lifschitz reduct of Π with respect to a set $S \subset Lit$ is the the disjunctive logic program Π^S obtained from Π by deleting*

(i) each rule that has a formula not L in its body with $L \in S$, and

(ii) all formulas of the form not L in the bodies of the remaining rules.

Definition 3.2.3. *As Π^S does not contain not, its answer sets are already defined. If S is one of them, then we say that S is an* answer set *of Π.*

Example 3.2.2. *We modify the program Π from Example 3.2.1 as follows:*

$$\begin{aligned}\Pi \quad = \{\quad & man(tom) \leftarrow not\, husband;\\ & single \vee husband \leftarrow man(X)\}.\end{aligned}$$

Now the modified program has the unique answer set $\{man(tom), single\}$*. Consider the answer set* $S = \{man(tom), husband\}$ *and its Gelfond-Lifschitz reduct*

$$\Pi^S \quad = \{ \quad single \vee husband \leftarrow man(tom)\}.$$

The program Π^S *has one answer set, namely* $\{\}$*. Hence,* S *is not an answer set of* Π^S *and therefore it is not an answer set of* Π*.*

Property 3.2.1. *[39] Let* Π *be a normal program such that* Π *is stratified. Then* Π *has at most one answer set. If* Π *is, in addition, constraint-free, then it has a unique answer set.*

3.3 Programming Techniques

Disjunctive logic programming under the answer set semantics is a very expressive formalism for knowledge representation and reasoning. In Chapter 4, the relevant complexity classes are introduced and we give the corresponding complexity results. In this section, we mention some programming techniques which have been used for our system.

3.3.1 Linear Ordering

To derive predicates for infimum, supremum, and successor, we need to define an order over the domain elements. We will exemplify this method by an example.

Example 3.3.1. *Consider a company. The employees and their salaries are represented by* empl/2*. We compute the linear ordering over the employees and derive the predicates* succ*,* inf *and* sup *for successor, infimum and supremum with the following program* Π_{ord}*.*

$$\begin{aligned}\Pi_{ord} \quad = \{ \quad & \mathrm{lt}(X,Y) \leftarrow \mathrm{empl}(X,\_), \mathrm{empl}(Y,\_), X < Y;\\ & \mathrm{nsucc}(X,Z) \leftarrow \mathrm{lt}(X,Y), \mathrm{lt}(Y,Z);\\ & \mathrm{succ}(X,Y) \leftarrow \mathrm{lt}(X,Y), \mathit{not}\,\mathrm{nsucc}(X,Y);\\ & \mathrm{ninf}(Y) \leftarrow \mathrm{lt}(X,Y);\\ & \mathrm{inf}(X) \leftarrow \mathrm{empl}(X,\_), \mathit{not}\,\mathrm{ninf}(X);\\ & \mathrm{nsup}(X) \leftarrow \mathrm{lt}(X,Y);\\ & \mathrm{sup}(X) \leftarrow \mathrm{empl}(X,\_), \mathit{not}\,\mathrm{nsup}(X)\}.\end{aligned}$$

Now we can compute the sum of salaries of the employees with the following program Π_{sum}*.*

$$\begin{aligned}\Pi_{\mathrm{sum}} \quad = \{ \quad & \mathrm{partSum}(X,S) \leftarrow \mathrm{inf}(X), \mathrm{empl}(X,S);\\ & \mathrm{partSum}(Y,S) \leftarrow \mathrm{succ}(X,Y), \mathrm{partSum}(X,S1), \mathrm{empl}(X,S2), S = S1 + S2;\\ & \mathrm{sum}(S) \leftarrow \mathrm{sup}(X), \mathrm{partSum}(X,S)\}.\end{aligned}$$

3.3.2 Guess&Check Methodology

With the **Guess&Check** methodology [34, 41], it is possible to encode problems in a declarative way, by at first guessing a search space, and then filtering out the solutions via the elimination of wrong candidates.

Given a set $\widehat{I}$ of facts that specify an instance I of some problem $\mathbf{P}$, a Guess&Check program $\mathcal{P}$ for $\mathbf{P}$ consists of the following two parts:

Guessing Part: The guessing part $\mathcal{G} \subseteq \mathcal{P}$ of the program defines the search space, such that answer sets of $\mathcal{G}(\widehat{I})$ represent solution candidates for I.

Checking part: The checking part $\mathcal{C} \subseteq \mathcal{P}$ of the program filters the solution candidates in such a way that the answer sets of $\mathcal{G} \cup \mathcal{C} \cup \widehat{I}$ represent the admissible solutions for the problem instance I.

This technique can be extended to the **Guess&Check&Optimize** paradigm by adding an optimization part. As we did not make use of the optimization part, we refer to [41] for details.

For a number of problems, the Guess&Check program has the following layered structure:

- The guessing part $\mathcal{G}$ consists of disjunctive rules that "guess" a solution candidate S.
- The checking part $\mathcal{C}$ consists of integrity constraints that check the admissibility of S.

Each layer may have further auxiliary predicates, defined by normal stratified rules. As an example let us consider the *3-Colorability* problem.

Example 3.3.2. *Given a graph $G = (V, E)$ as input, assign to each node one of three colors (red, green, or blue), such that adjacent nodes always have different colors. 3-Colorability is a classical NP-complete problem. The encoding looks as follows, where a set of nodes V and edges E are specified by the predicates node/1 and edge/2.*

$$\begin{aligned}\mathcal{G} &= \{col(X,r) \vee col(X,g) \vee col(X,b) \leftarrow node(X).\}\\ \mathcal{C} &= \{\leftarrow edge(X,Y), col(X,C), col(Y,C).\}\end{aligned}$$

The nodes and edges of the graph are represented by a set $\widehat{I}$ of facts with predicates node and edge. The "guessing" rule $\mathcal{G}$ states that every node is colored red, green, or blue. The "checking" constraint $\mathcal{C}$ assures that there are no two connected nodes with the same color. The answer sets of $\mathcal{G}(\widehat{I})$ represent all possible assignments of colors to the nodes. The minimality of answer sets guarantees that every node has a different color. All those *answer sets of $\mathcal{G}(\widehat{I})$ which satisfy $\mathcal{C}$ are solutions for the problem. There is a one-to-one correspondence between the solutions of the 3-colorable problem and the answer sets of $\widehat{I} \cup \mathcal{G} \cup \mathcal{C}$. The graph G is 3-colorable if and only if $\widehat{I} \cup \mathcal{G} \cup \mathcal{C}$ has some answer set, and each of the answer sets of $\widehat{I} \cup \mathcal{G} \cup \mathcal{C}$ represents a legal 3-coloring of G.*

3.3.3 Saturation Technique

In the previous example, we considered a program, where the checking part $\mathcal{C}$ had no influence on the guessing part $\mathcal{G}$. Now we consider problems, where the checking part $\mathcal{C}$ must have influence on the guessing part $\mathcal{G}$. This influence is defined by disjunctive rules in $\mathcal{C}$ or an interference from $\mathcal{C}$ with $\mathcal{G}$ (in particular head-cycles must be present in $\mathcal{G} \cup \mathcal{C}$).

As an example, let us consider a restricted class of *Quantified Boolean Formulas*, namely $\exists\forall$-*QBFs*. QBFs are extensions of propositional logic by the possibility to quantify over propositional atoms. Here, we consider prenex QBFs of the form

$$Q_1X_1Q_2X_2 \dots Q_nX_n\phi,$$

where $Q \in \{\forall, \exists\}$, and ϕ is a quantifier free propositional formula. The semantic is defined as follows.

- $\forall\phi(p)$ is a shorthand for $\phi(\top) \wedge \phi(\bot)$, and
- $\exists\phi(p)$ is a shorthand for $\phi(\top) \vee \phi(\bot)$.

Example 3.3.3. *[35,41] The $\exists\forall$-QBF problem is to decide whether a quantified Boolean formula (QBF) of the shape $\Phi = \exists X \forall Y \phi$, where X and Y are disjoint sets of propositional variables and $\phi = C_1 \vee \dots \vee C_k$ is a 3DNF formula over $X \cup Y$, evaluates to true. We want to have a witness assignment σ to the variables X, such that $\phi[X/\sigma(X)]$ is a tautology, where $X/\sigma(X)$ denotes the substitution of X by $\sigma(X)$. This leads to a* Guess&Check *disjunctive logic program, in which the witness assignment σ is guessed by some rules and the rest of the program checks whether $\phi[X/\sigma(X)]$ is a tautology.*
A QBF Φ is encoded as a set of facts $\widehat{\Phi}$, consisting of

- *exists(v), for each existential variable $v \in X$;*
- *forall(v), for each universal variable $v \in Y$; and*
- *term$(p_1, p_2, p_3, q_1, q_2, q_3)$, for each disjunct $l_1 \wedge l_2 \wedge l_3$ in ϕ, where*
  - *(i) if l_i is a positive atom v_i, then $p_i = v_i$, otherwise $p_i =$ "true", and*
  - *(ii) if l_i is a negated atom $\neg v_i$, then $q_i =$ "false".*

For example, $term(x_1, true, y_4, false, y_2, false)$, encodes $x_1 \wedge \neg y_2 \wedge y_4$.

The program $\mathcal{P}_{\exists\forall-QBF}$ looks as follows.

$$
\begin{array}{lll}
\mathcal{G} & = \{ & t(true); \\
 & & f(false); \\
 & & t(X) \vee f(X) \leftarrow exists(X)\}. \\
\mathcal{C} & = \{ & t(Y) \vee f(Y) \leftarrow forall(Y); \\
 & & w \leftarrow term(X, Y, Z, Na, Nb, Nc), t(X), t(Y), t(Z), f(Na), f(Nb), f(Nc); \\
 & & t(Y) \leftarrow w, forall(Y); \\
 & & f(Y) \leftarrow w, forall(Y); \\
 & & \leftarrow not\, w\}.
\end{array}
$$

The guessing part $\mathcal{G}$ "initializes" the logical constants "true" and "false" and chooses a witnessing assignment σ to the variables in X, which leads to an answer set A_G for this part. The more tricky checking part $\mathcal{C}$ then tests whether $\phi[X/\sigma(X)]$ is a tautology, using a saturation technique [35]: the constraint $\leftarrow not\, w$. enforces that w must be true in any answer set of the program; the preceding two rules imply that such an answer set A contains both $t(y)$ and $f(y)$ for every $y \in Y$. Hence, A has a unique extension with respect to w and all $t(y)$ and $f(y)$ where $y \in Y$. By the minimality of answer sets, an extension of A_G to the (uniquely determined) answer set A of the whole program exists, iff for each possible assignment μ to the variables in Y, effected by the disjunctive rule in the checking part, the atom w is derived. The latter holds iff there is some disjunct in $\phi[X/\sigma(X), Y/\mu(Y)]$ which is true. Hence, A is an answer set iff the formula $\phi[X/\sigma(X)]$ is a tautology. In summary, we obtain that Φ is a Yes-instance, that is, it evaluates to true, iff $\mathcal{P}_{\exists\forall-QBF}(\widehat{\Phi})$ has some answer set. Moreover, the answer sets of $\mathcal{P}_{\exists\forall-QBF}(\widehat{\Phi})$ are in a one-to-one correspondence with the witnesses σ for the truth of Φ.

Chapter 4

Complexity Issues

4.1 Basic Introduction

The *polynomial hierarchy* is a sequence of classes starting with

$$\Delta_0^P := \Sigma_0^P := \Pi_0^P := \mathrm{P},$$

where P is the set of decision problems solvable in polynomial time.
For $k \geq 0$, we get

$$\begin{aligned}
&\Delta_{k+1}^P := \mathrm{P}^{\Sigma_k^P}, \\
&\Sigma_{k+1}^P := \mathrm{NP}^{\Sigma_k^P}, \\
&\Pi_{k+1}^P := \mathrm{coNP}^{\Sigma_k^P},
\end{aligned}$$

where A^B is the set of decision problems solvable by a Turing machine in class A augmented by an oracle for some problem in class B. For example, $\Sigma_1^P = \mathrm{NP}$, $\Pi_1^P = \mathrm{coNP}$, and $\Delta_2^P = \mathrm{P}^{\mathrm{NP}}$ is the class of problems solvable in polynomial time with an oracle for some problem in NP.

Known relations between the classes in the polynomial hierarchy are as follows:

$$\begin{aligned}
&\Sigma_k^P \subseteq \Delta_{k+1}^P \subseteq \Sigma_{k+1}^P \\
&\Pi_k^P \subseteq \Delta_{k+1}^P \subseteq \Pi_{k+1}^P \\
&\Sigma_k^P = \mathrm{co}\Pi_k^P
\end{aligned}$$

The class *PSPACE* is the set of decision problems that can be solved by a deterministic or Turing machine using a polynomial amount of memory and an arbitrary run time. If a nondeterministic Turing machine is used, then the class *NPSPACE* obtained. In

	stable	*adm*	*prefex*	*comp*	*ground*
Cred_e	NP-c	NP-c	NP-c	NP-c	in P
Skept_e	coNP-c	(trivial)	Π_2^P-c	in P	in P

Table 4.1: Complexity for decision problems in argumentation frameworks.

Savitch's theorem [48], it is shown that *PSPACE* equals *NPSPACE*. A decision problem is *PSPACE-complete* if it is in *PSPACE* and every problem in *PSPACE* can be reduced to it in polynomial time.

4.2 Complexity of Argumentation Frameworks

In this section, we briefly review the complexity of reasoning in argumentation frameworks. To this end, we define the following decision problems for $e \in \{stable, adm, prefex, comp, ground\}$:

- Cred_e: Given an argumentation framework $AF = (A, R)$, and an argument $a \in A$. Is a contained in some $S \in e(AF)$?
- Skept_e: Given an argumentation framework $AF = (A, R)$, and an argument $a \in A$. Is a contained in each $S \in e(AF)$?

The complexity results are depicted in Table 4.1 (many of them follow implicitly from [24], for the remaining results and discussions we refer to [21, 29]).

In Table 4.1, "$\mathcal{C}$-c" refers to a problem which is complete for class $\mathcal{C}$, while "in $\mathcal{C}$" is assigned to problems for which a tight lower complexity bound is not known. Skeptical reasoning over admissible extensions is always trivially false. Moreover, we note that credulous reasoning over preferred extensions is easier than skeptical reasoning. This is due to the fact that the additional maximality criterion only comes into play for the latter task. Indeed, for credulous reasoning, the following makes clear why there is no increase in complexity compared to credulous reasoning over admissible extensions: a is contained in some $S \in adm(AF)$ iff a is contained in some $S \in prefex(AF)$. A similar argument immediately shows why skeptical reasoning over complete extensions reduces to skeptical reasoning over the grounded extension.

Complexity of Value Based Argumentation Frameworks

In value based argumentation framework, the complexity of decision problems is dependent on the value preferences. As a reminder, on Page 24 we already gave the definitions

	stratified programs	normal programs	general case
$\models_c$	P	NP	Σ_2^P
$\models_s$	P	coNP	Π_2^P

Table 4.2: Data complexity for datalog (all results are completeness results).

for objective acceptance (OBA) and subjective acceptance (SBA). In [28], it has been shown that OBA is coNP-complete and SBA is NP-complete. Basically, complexity of reasoning in VAFs has the same complexity as in basic AFs.

Complexity of Preference Based Argumentation Frameworks

In [23] it has been shown, that credulous reasoning over stable extensions is NP-hard, and sceptical reasoning over stable extensions is coNP-hard. In principle, reasoning in PAFs has the same complexity as in basic AFs.

4.3 Data Complexity

We briefly recall some complexity results for disjunctive logic programs. We focus on results for data complexity. Data complexity in our context is the complexity of checking whether $P(D) \models A$ when programs P are fixed, while input databases D and ground atoms A are an input of the decision problem. Depending on the concrete definition of $\models$, we give the complexity results in Table 4.2 (cf. [22] and the references therein).

When comparing Table 4.1 to Table 4.2, it is obvious which programs are adequate for the respective decision problems of argumentation frameworks.

Further Complexity Results

In Section 3.3, we introduced some programming techniques by some examples. Here we give the respective complexity results. Problems which are in NP, like the NP-complete 3-Colorability problem, can be encoded with a simple Guess&Check program, as used in Example 3.3.2. In this program, we just used disjunction in the guessing part and the layer is head-cycle free. Hence, an answer set A of $\Pi_{guess} \cup \mathcal{F}_I$ can be guessed in polynomial time [6, 34]. The complexity of deciding whether an atom belongs to some answer set is thus in NP.

For problems beyond NP and in particular for Σ_2^P-complete problems, this technique can not be applied anymore, because it would imply $\Sigma_2^P \subseteq \mathrm{NP}$, which is widely believed to be false. Thus, if a program solves a Σ_2^P-complete problem and has guessing and checking parts Π_{guess} and Π_{check} with complexities below Σ_2^P, then Π_{check} must either contain disjunctive rules or interfere with Π_{guess} (and in particular head-cycles must be present in $\Pi_{guess} \cup \Pi_{check}$) [34, 41]. In Example 3.3.3 we used a program where the checking part Π_{check} contains a disjunctive rule. This is adequate due to the fact that the $\exists\forall$-QBF problem is Σ_2^P-complete [46].

Chapter 5

Encodings

In this chapter, we introduce stepwise the DLV program Π, which computes the semantics for various variants of frameworks described in Chapter 2. Moreover we prove correctness of Π, by showing that the answer sets of the program are in a one-to-one correspondence to the acceptable sets of arguments. We provide a fixed program, totally independent of the concrete problem instance. The argumentation framework AF and the type of extension to compute is set up as an input database.

In Section 5.1 we describe the decomposition of the program Π into modules. The basic dependence relation between the modules forms a partial order. The decomposition does not only make it easier to describe and understand the whole program, but also allows us to prove the functioning of the program step by step. In Section 5.2 and 5.3 we will add additional modules for value based argumentation frameworks and bipolar argumentation frameworks. The entire program Π is given at a glance in Appendix A.

5.1 Basic Argumentation Framework

For a better understanding, we split the program Π into modules π_χ with $\chi \in \mathit{Mod}(\Pi)$ where, for a given argumentation framework AF, and a semantic $\varepsilon \in \Sigma = \{\mathit{stable}, \mathit{adm}, \mathit{comp}, \mathit{ground}, \mathit{prefex}\}$,

$$\begin{aligned} \mathit{Mod}(\Pi) \; = \{ \; & \widehat{AF}, \mathit{ext}(\varepsilon), \mathit{guess}, \mathit{defeat}, \mathit{cf}, \mathit{defeated}, \mathit{not\_defended}, \mathit{adm}, \\ & \mathit{comp}, \mathit{stable}, \mathit{ground}, \mathit{ord}, \mathit{prefex}\}. \end{aligned}$$

Later, we will use the above as splitting sets [42]. A description of the respective modules is given in Table 5.1. Following the notion of *potential usage* in [36] we will prove step by step the correctness of the program Π resulting in Theorem 1.

Definition 5.1.1. *Let π_1 and π_2 be programs. We say that π_2* **potentially uses** *π_1, or*

Module	Description
$\widehat{AF}$	Representation of the argumentation framework AF.
$\pi_{ext(\varepsilon)}$	Predicates for calculations control specifying the type of the argumentation framework and which extensions should be calculated.
π_{guess}	Guessing of an interpretation representing a set subset of arguments.
π_{defeat}	Derivation of successful defeat.
π_{cf}	Constraint for elimination of answer sets which represent conflicts.
$\pi_{defeated}$	Rule for derivation of defeated arguments.
$\pi_{not\_defended}$	Rule for derivation of not defended arguments.
π_{adm}	Constraint for elimination of answer sets representing non-admissible sets.
π_{comp}	Constraint for elimination of answer sets representing non-complete sets.
π_{stable}	Constraint for elimination of answer sets representing non-stable sets.
π_{ground}	Stratified rules to compute the answer set representing the grounded extension.
π_{ord}	Rules for specification of an ordering on the arguments.
π_{prefex}	Elimination of not preferred answer sets.

Table 5.1: Modules π_χ of the program Π

π_1 is **independent of** *π_2 (denoted $\pi_2 \rhd \pi_1$), iff each predicate that occurs in some head of π_2 does not occur (positively or negatively) in π_1.*

Proposition 5.1.1. *Let π_1, π_2 and π_3 be disjunctive programs. Then,*

1. *$(\pi_2 \cup \pi_3) \rhd \pi_1$ iff $\pi_2 \rhd \pi_1$ and $\pi_3 \rhd \pi_1$, and*
2. *$\pi_3 \rhd (\pi_1 \cup \pi_2)$ iff $\pi_3 \rhd \pi_1$ and $\pi_3 \rhd \pi_2$.*

For a proof of Proposition 5.1.1, we refer to [47]. The following proposition is based on [36,42].

Proposition 5.1.2. *Let $\Pi = \pi_1 \cup \pi_2$ be a disjunctive program such that $\pi_2 \rhd \pi_1$. Then, it holds that*

$$AS(\Pi) = \bigcup_{A \in AS(\pi_1)} (AS(A \cup \pi_2)).$$

Each module π_χ (except $\widehat{AF}$ and $\pi_{ext(\varepsilon)}$) depends on the output of other modules. We will denote this dependency by the transitive *partial order relation* $\prec$. If $\pi_j \prec \pi_i$, then π_i serves as input for π_j.

Figure 5.1 shows the partial order relation between the modules π_χ, where an edge from module π_i to π_j stands for $\pi_j \prec \pi_i$.

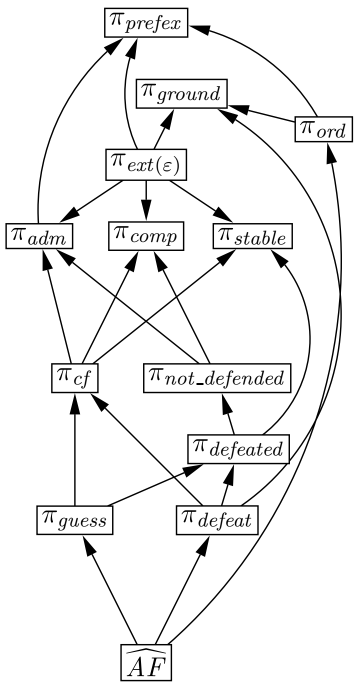


Figure 5.1: Partial order relation between modules π_χ.

Definition 5.1.2. *Let $AF = (A, R)$. We denote by $\Pi_\chi(\widehat{AF})$ the union of module π_χ and its predecessors for the argumentation framework AF:*

$$\Pi_\chi(\widehat{AF}) = \pi_\chi \cup \bigcup_{\pi_\chi \prec \pi_\varphi} \pi_\varphi,$$

where $\chi \in Mod(\Pi)$.

For example:

$$\begin{aligned}
\Pi_{cf}(\widehat{AF}) &= \pi_{cf} \cup \pi_{defeat} \cup \pi_{guess} \cup \widehat{AF}, \\
\Pi_{guess}(\widehat{AF}) &= \pi_{guess} \cup \widehat{AF}, \\
\Pi_{defeat}(\widehat{AF}) &= \pi_{defeat} \cup \widehat{AF}, \\
\Pi_{ord}(\widehat{AF}) &= \pi_{ord} \cup \widehat{AF}, \\
\Pi_{defeated}(\widehat{AF}) &= \pi_{defeated} \cup \pi_{guess} \cup \pi_{defeat} \cup \widehat{AF}, \\
\Pi_{not\_defended}(\widehat{AF}) &= \pi_{not\_defended} \cup \pi_{defeated} \cup \pi_{guess} \cup \pi_{defeat} \cup \widehat{AF}.
\end{aligned}$$

Definition 5.1.3. *Let $AF = (A, R)$ and $\varepsilon \in \Sigma = \{stable, adm, comp, ground, prefex\}$. The program $\Pi_\varepsilon(\widehat{AF})$ represents the union of module π_ε and its predecessors with respect to $\prec$.*

For example:

$$\begin{aligned}
\Pi_{stable}(\widehat{AF}) &= \pi_{stable} \cup \pi_{cf} \cup \pi_{defeated} \cup \pi_{defeat} \cup \pi_{guess} \cup \widehat{AF} \cup \pi_{ext(stable)},\\
\Pi_{adm}(\widehat{AF}) &= \pi_{adm} \cup \pi_{cf} \cup \pi_{not\_defended} \cup \pi_{defeat} \cup \pi_{guess} \cup \widehat{AF} \cup \pi_{ext(adm)},\\
\Pi_{comp}(\widehat{AF}) &= \pi_{comp} \cup \pi_{cf} \cup \pi_{not\_defended} \cup \pi_{defeat} \cup \pi_{guess} \cup \widehat{AF} \cup \pi_{ext(comp)},\\
\Pi_{ground}(\widehat{AF}) &= \pi_{ground} \cup \pi_{cf} \cup \pi_{defeat} \cup \pi_{guess} \cup \widehat{AF} \cup \pi_{ext(ground)},\\
\Pi_{prefex}(\widehat{AF}) &= \pi_{prefex} \cup \pi_{adm} \cup \pi_{ord} \cup \pi_{cf} \cup \pi_{not\_defended} \cup \pi_{defeat} \cup \pi_{guess}\\
&\quad \cup \widehat{AF} \cup \pi_{ext(prefex)}.
\end{aligned}$$

Theorem 1. *For each argumentation framework $AF = (A, R)$ and extension $\varepsilon \in \Sigma = \{stable, adm, comp, ground, prefex\}$, there is a one-to-one correspondence between $\varepsilon(AF)$ and $AS(\Pi_\varepsilon(\widehat{AF}))$.*

5.1.1 Description of the Modules π_χ of Π

In the following we introduce the modules in detail, and we prove their correctness resulting in Theorem 1.

Representation of the Argumentation Framework

The argumentation framework $AF = (A, R)$ is represented in the following way:

Definition 5.1.4. *Given an argumentation framework AF, $\widehat{AF}$ consists of the following facts:*

$$\begin{aligned}
\widehat{AF} = \ &\{\ \ \mathrm{arg}(a) \mid a \in A\} \cup\\
&\{\ \ \mathrm{attack}(a, b) \mid (a, b) \in R\}.
\end{aligned}$$

For example, the encoding $\widehat{AF}$ for the argumentation framework AF of Example 2.1.1

is as follows:

$$\begin{aligned}\widehat{AF} = \{ \; & \text{arg}(a),\\ & \text{arg}(b),\\ & \text{arg}(c),\\ & \text{arg}(d),\\ & \text{arg}(e),\\ & \text{attack}(a,b),\\ & \text{attack}(c,b),\\ & \text{attack}(c,d),\\ & \text{attack}(d,c),\\ & \text{attack}(d,e),\\ & \text{attack}(e,e)\}.\end{aligned}$$

As the module $\widehat{AF}$ does not contain any rules or constraints, the following proposition is obvious.

Proposition 5.1.3. *For every argumentation framework $AF = (A, R)$ it holds that*

$$AS(\widehat{AF}) = \{\widehat{AF}\} = \{\{\text{arg}(a) \mid a \in A\} \cup \{\text{attack}(a,b) \mid (a,b) \in R\}\}.$$

This means that the program $\widehat{AF}$ has one single answer set $\widehat{AF}$.

Selection of Extensions

The module $\pi_{ext(\varepsilon)}$ denotes which extensions $\varepsilon \in \Sigma$ should be computed. It is not possible to compute more than one extension at the same time. Except for the preferred extensions which require also the admissible ones.

Definition 5.1.5. *For an extension $\varepsilon \in \Sigma$, the module $\pi_{ext(\varepsilon)}$ consists of the following fact and rule:*

$$\begin{aligned}\pi_{ext(\varepsilon)} = \{ \; & \varepsilon;\\ & \text{adm} \leftarrow \text{prefex}\}.\end{aligned}$$

Guessing an Interpretation

In the module π_{guess} all subsets $S \subseteq A$ are guessed. The resulting answer sets represent the search space for the subsequent modules.

Definition 5.1.6. *The module π_{guess} consists of the following two rules:*

$$\begin{aligned} \pi_{guess} \quad = \{ \quad & \mathrm{in}(X) \leftarrow \mathit{not}\, \mathrm{out}(X), \mathrm{arg}(X); & (5.1) \\ & \mathrm{out}(X) \leftarrow \mathit{not}\, \mathrm{in}(X), \mathrm{arg}(X)\}. & (5.2) \end{aligned}$$

The two predicates in/1 and out/1 determine, whether an argument $a \in A$ is in or oppositely not in the set $S \subseteq A$.

Definition 5.1.7. *For a set I of facts, we define*

$$\sigma(I) = \{a \mid \mathrm{in}(a) \in I\}.$$

Proposition 5.1.4. *Let $AF = (A, R)$. For each set $S \subseteq A$, there exists an interpretation $I \in AS(\Pi_{guess}(\widehat{AF}))$ such that*

$$I = \widehat{AF} \cup \{\mathrm{in}(a) \mid a \in S\} \cup \{\mathrm{out}(a) \mid a \in A \setminus S\}.$$

For each interpretation $I \in AS(\Pi_{guess}(\widehat{AF}))$ it holds that $\widehat{AF} \subseteq I$ and there exists a set $S \subseteq A$ such that $S = \sigma(I)$.

The concept of guessing is well known, therefore we omit the proof.

Defeat

In module π_{defeat} a successful attack is derived. For the basic argumentation frameworks the defeat relation equals the attack relation. For PAFs and VAFs this module will be modified!

Definition 5.1.8. *The module π_{defeat} consists of the following rule:*

$$\pi_{defeat} \quad = \{ \quad \mathrm{defeat}(X, Y) \leftarrow \mathrm{attack}(X, Y)\}.$$

Proposition 5.1.5. *For every argumentation framework $AF = (A, R)$ it holds that*

$$\begin{aligned} AS(\Pi_{defeat}(\widehat{AF})) \quad & = \{ \quad I \cup \{\mathrm{defeat}(a, b) \mid \mathrm{attack}(a, b) \in I\} \mid I \in AS(\widehat{AF})\} \\ & = \{ \quad \widehat{AF} \cup \{\mathrm{defeat}(a, b) \mid (a, b) \in R\}\}. \end{aligned}$$

The program $\Pi_{defeat}(\widehat{AF})$ has one single answer set denoted by $\widehat{AF}_{defeat}$.

Conflict free Sets

The module π_{cf} eliminates all answer sets with conflicting arguments.

Definition 5.1.9. *The module π_{cf} consists of the following constraint:*

$$\pi_{cf} \quad = \{ \quad \leftarrow \mathrm{in}(X), \mathrm{in}(Y), \mathrm{defeat}(X,Y)\}.$$

We know that $\pi_{guess} \rhd \pi_{cf}$ and $\pi_{defeat} \rhd \pi_{cf}$ holds. From Proposition 5.1.1, we can conclude that $(\pi_{guess} \cup \pi_{defeat}) \rhd \pi_{cf}$ holds. Hence, we can conclude from Proposition 5.1.2 and Definition 5.1.2 that the answer sets of the program $\Pi_{cf}(\widehat{AF})$ are defined as follows,

$$AS(\Pi_{cf}(\widehat{AF})) \quad = \quad \bigcup_{J \in AS((\Pi_{guess} \cup\ \Pi_{defeat})(\widehat{AF}))} AS(J \cup \pi_{cf}).$$

Hence, the correspondence between the answer sets of the program $\Pi_{cf}(\widehat{AF})$ and the conflict free subsets of arguments is given in Proposition 5.1.6.

Proposition 5.1.6. *Let $AF = (A, R)$. For each conflict free set $S \subseteq A$, there exists an interpretation $I \in AS(\Pi_{cf}(\widehat{AF}))$, such that*

$$I = \widehat{AF}_{defeat} \cup \{\mathrm{in}(a) \mid a \in S\} \cup \{\mathrm{out}(a) \mid a \in A \setminus S\}.$$

For each interpretation $I \in AS(\Pi_{cf}(\widehat{AF}))$ it holds that $\widehat{AF}_{defeat} \subseteq I$, and there exists a conflict free set $S \subseteq A$, such that $S = \sigma(I)$.

Proof. We know that the program $\Pi_{defeat}(\widehat{AF})$ has one single answer set $\widehat{AF}_{defeat}$. Hence, it is easy to see that

$$AS((\Pi_{guess} \cup \Pi_{defeat})(\widehat{AF})) \quad = \quad \{\widehat{AF}_{defeat} \cup J \mid J \in AS(\Pi_{guess}(\widehat{AF}))\}. \qquad (5.3)$$

Let $S \subseteq A$ be a conflict free set. By Proposition 5.1.4, we know that there exists an interpretation $J \in AS(\Pi_{guess}(\widehat{AF}))$ such that

$$J = \widehat{AF} \cup \{\mathrm{in}(a) \mid a \in S\} \cup \{\mathrm{out}(a) \mid a \in A \setminus S\}.$$

By (5.3), we know that $\widehat{AF}_{defeat} \supseteq \widehat{AF}$. One can see that there exists an interpretation $I \in AS((\Pi_{guess} \cup \Pi_{defeat})(\widehat{AF}))$, such that

$$I = \widehat{AF}_{defeat} \cup \{\mathrm{in}(a) \mid a \in S\} \cup \{\mathrm{out}(a) \mid a \in A \setminus S\}.$$

Since S is conflict free, the constraint in π_{cf} can not fire. Hence, it follows that

$$I \in AS(\Pi_{cf}(\widehat{AF})).$$

Now we show that for each interpretation $I \in AS(\Pi_{cf}(\widehat{AF}))$ there exists a conflict free set $S \subseteq A$, such that $S = \sigma(I)$. The module π_{cf} only consists of one constraint. Hence, no new facts are derived. Therefore, $I \in AS((\Pi_{guess} \cup \Pi_{defeat})(\widehat{AF}))$ has to hold. Consider an interpretation J, such that

$$J = I \setminus \{\mathrm{defeat}(a,b) \mid (a,b) \in R\}.$$

It is obvious that $J \in AS(\Pi_{guess}(\widehat{AF}))$. Hence by Proposition 5.1.4, we know that there exists a set $S \subseteq A$ such that $\sigma(I) = \sigma(J) = S$. □

Defeated Arguments

With the module $\pi_{defeated}$ we derive defeated arguments which we need to compute stable extensions. Furthermore we use $\pi_{defeated}$ for the module $\pi_{not\_defended}$.

Definition 5.1.10. *The module $\pi_{defeated}$ consists of the following rule:*

$$\pi_{defeated} = \{\text{defeated}(X) \leftarrow \text{in}(Y), \text{defeat}(Y, X)\}.$$

We know that $\pi_{guess} \rhd \pi_{defeated}$ and $\pi_{defeat} \rhd \pi_{defeated}$ holds. From Proposition 5.1.1, we can conclude that $(\pi_{guess} \cup \pi_{defeat}) \rhd \pi_{defeated}$ holds. Hence, we can conclude from Proposition 5.1.2 and Definition 5.1.2 that the answer sets of the program $\Pi_{defeated}(\widehat{AF})$ are defined as follows,

$$AS(\Pi_{defeated}(\widehat{AF})) \quad = \bigcup_{J \in AS((\Pi_{guess} \cup\, \Pi_{defeat})(\widehat{AF}))} AS(J \cup \pi_{defeated}).$$

Hence, the correspondence between the answer sets of the program $\Pi_{defeated}(\widehat{AF})$ and the subsets of arguments is given in Proposition 5.1.7.

Proposition 5.1.7. *Let $AF = (A, R)$. For each set $S \subseteq A$, there exists an interpretation $I \in AS(\Pi_{defeated}(\widehat{AF}))$, such that*

$$\begin{aligned} I \quad = \quad & \widehat{AF}_{defeat} \cup \{\text{in}(a) \mid a \in S\} \cup \{\text{out}(a) \mid a \in A \setminus S\} \\ & \cup \{\text{defeated}(a) \mid b \in S, (b, a) \in R\}. \end{aligned}$$

For each interpretation $I \in AS(\Pi_{defeated}(\widehat{AF}))$ it holds that $\widehat{AF}_{defeat} \subseteq I$, and there exists a set $S \subseteq A$, such that $S = \sigma(I)$.

Proof. According to the proof of Proposition 5.1.6, by (5.3), we know that $\widehat{AF}_{defeat} \supseteq \widehat{AF}$. One can see that for each set $S \subseteq A$, there exists an interpretation $J \in AS((\Pi_{guess} \cup \Pi_{defeat})(\widehat{AF}))$, such that

$$J = \widehat{AF}_{defeat} \cup \{\text{in}(a) \mid a \in S\} \cup \{\text{out}(a) \mid a \in A \setminus S\}.$$

Assume that there exists an interpretation $I \supset J$, such that

$$I = J \cup \{\text{defeated}(a) \mid b \in S, (b, a) \in R\}.$$

It is easy to see that

$$I \in AS(\Pi_{defeated}(\widehat{AF})).$$

Now we show that for each interpretation $I \in AS(\Pi_{defeated}(\widehat{AF}))$ there exists a set $S \subseteq A$, such that $S = \sigma(I)$. Consider an interpretation $J \subset I$, such that

$$J = I \setminus \{\text{defeat}(a, b) \mid (a, b) \in R\} \cup \{\text{defeated}(a) \mid b \in S, (b, a) \in R\}.$$

It is obvious that $J \in AS(\Pi_{guess}(\widehat{AF}))$. Hence by Proposition 5.1.4, we know that there exists a set $S \subseteq A$ such that $\sigma(I) = \sigma(J) = S$. □

Non Defended Arguments

In the module $\pi_{not\_defended}$, we derive arguments which are not defended. We need these arguments later to compute admissible and complete extensions.

Definition 5.1.11. *The module $\pi_{not\_defended}$ consists of the following rule:*

$$\pi_{not\_defended} = \{\text{not\_defended}(X) \leftarrow \text{defeat}(Y,X), \textit{not}\ \text{defeated}(Y)\}.$$

We know that $\pi_{defeated} \triangleright \pi_{not\_defended}$. As shown in Figure 5.1, it holds that $\pi_{not\_defended} \prec \pi_{defeated}$. Hence, we can conclude from Proposition 5.1.2 and Definition 5.1.2 that the answer sets of the program $\Pi_{not\_defended}(\widehat{AF})$ are defined as follows,

$$AS(\Pi_{not\_defended}(\widehat{AF})) = \bigcup_{J \in AS(\Pi_{defeated}(\widehat{AF}))} (AS(J \cup \pi_{not\_defended}))$$

Hence, the correspondence between the answer sets of the program $\Pi_{not\_defended}(\widehat{AF})$ and the subsets of arguments is given in Proposition 5.1.8.

Proposition 5.1.8. *Let $AF = (A, R)$. For each set $S \subseteq A$, there exists an interpretation $I \in AS(\Pi_{not\_defended}(\widehat{AF}))$, such that*

$$\begin{aligned} I = \{ \ & \widehat{AF}_{defeat} \cup \{\text{in}(a) \mid a \in S\} \cup \{\text{out}(a) \mid a \in A \setminus S\} \\ & \cup \{\text{defeated}(a) \mid b \in S, (b,a) \in R\} \\ & \cup \{\text{not\_defended}(a) \mid (b,a) \in R, \nexists c \in S : (c,b) \in R\}\}. \end{aligned}$$

For each interpretation $I \in AS(\Pi_{not\_defended}(\widehat{AF}))$ it holds that $\widehat{AF}_{defeat} \subseteq I$, and there exists a set $S \subseteq A$, such that $S = \sigma(I)$.

Proof. From Proposition 5.1.7 we know that for each set $S \subseteq A$, there exists an interpretation $J \in AS(\Pi_{defeated}(\widehat{AF}))$, such that

$$\begin{aligned} J = \ & \widehat{AF}_{defeat} \cup \{\text{in}(a) \mid a \in S\} \cup \{\text{out}(a) \mid a \in A \setminus S\} \\ & \cup \{\text{defeated}(a) \mid b \in S, (b,a) \in R\}. \end{aligned}$$

Assume that there exists an interpretation $I \supset J$, such that

$$I = J \cup \{\text{not\_defended}(a) \mid (b,a) \in R, \nexists c \in S : (c,b) \in R\}.$$

It is easy to see that

$$I \in AS(\Pi_{not\_defended}(\widehat{AF})).$$

Now we show that for each interpretation $I \in AS(\Pi_{not\_defended}(\widehat{AF}))$ there exists a set $S \subseteq A$, such that $S = \sigma(I)$. Consider an interpretation $J \subset I$, such that

$$J = I \setminus \{\text{not\_defended}(a) \mid (b,a) \in R, \nexists c \in S : (c,b) \in R\}.$$

It is obvious that $J \in AS(\Pi_{defeated}(\widehat{AF}))$. Hence by Proposition 5.1.7, we know that there exists a set $S \subseteq A$ such that $\sigma(I) = \sigma(J) = S$. □

Admissible Sets

The module π_{adm} eliminates all answer sets which do not correspond to $adm(AF)$.

Definition 5.1.12. *The module π_{adm} consists of the following constraint:*

$$\pi_{adm} \quad = \{ \quad \leftarrow \mathrm{in}(X), \mathrm{not\_defended}(X), \mathrm{adm}\}.$$

We know that $\pi_{cf} \rhd \pi_{adm}$ and $\pi_{not\_defended} \rhd \pi_{adm}$ holds. From Proposition 5.1.1, we can conclude that $(\pi_{cf} \cup \pi_{not\_defended}) \rhd \pi_{adm}$ holds. Hence, we can conclude from Proposition 5.1.2 and Definition 5.1.3 that the answer sets of the program $\Pi_{adm}(\widehat{AF})$ are defined as follows,

$$AS(\Pi_{adm}(\widehat{AF})) = \bigcup_{J \in AS((\Pi_{cf} \cup \Pi_{not\_defended})(\widehat{AF}))} (AS(J \cup \pi_{adm} \cup \pi_{ext(adm)})).$$

Hence, the correspondence between the answer sets of the program $\Pi_{adm}(\widehat{AF})$ and the admissible sets of arguments is given in Proposition 5.1.9.

Proposition 5.1.9. *Let $AF = (A, R)$. For each set $S \in adm(AF)$, there exists an interpretation $I \in AS(\Pi_{adm}(\widehat{AF}))$, such that*

$$\begin{aligned} I \quad = \quad & \widehat{AF}_{defeat} \cup \{\mathrm{adm.}\} \cup \{\mathrm{in}(a) \mid a \in S\} \cup \{\mathrm{out}(a) \mid a \in A \setminus S\} \\ & \cup \{\mathrm{defeated}(a) \mid b \in S, (b, a) \in R\} \\ & \cup \{\mathrm{not\_defended}(a) \mid (b, a) \in R, \nexists c \in S : (c, b) \in R\}. \end{aligned}$$

For each interpretation $I \in AS(\Pi_{adm}(\widehat{AF}))$ there exists a set $S \in adm(AF)$, such that $S = \sigma(I)$.

In order to prove Proposition 5.1.9, we first need to define the following proposition.

Proposition 5.1.10. *Let $AF = (A, R)$. For each conflict free set $S \subseteq A$, there exists an interpretation $I \in AS((\Pi_{cf} \cup \Pi_{not\_defended})(\widehat{AF}))$, such that*

$$\begin{aligned} I \quad = \quad & \widehat{AF}_{defeat} \cup \{\mathrm{in}(a) \mid a \in S\} \cup \{\mathrm{out}(a) \mid a \in A \setminus S\} \\ & \cup \{\mathrm{defeated}(a) \mid b \in S, (b, a) \in R\} \\ & \cup \{\mathrm{not\_defended}(a) \mid (b, a) \in R, \nexists c \in S : (c, b) \in R\}. \end{aligned}$$

For each interpretation $I \in AS((\Pi_{cf} \cup \Pi_{not\_defended})(\widehat{AF}))$ there exists a conflict free set $S \subseteq A$, such that $S = \sigma(I)$.

Proof. By Proposition 5.1.8, we know that there exists an interpretation
$I \in AS(\Pi_{not\_defended}(\widehat{AF}))$, such that

$$\begin{aligned} I \quad = \quad & \widehat{AF}_{defeat} \cup \{\mathrm{in}(a) \mid a \in S\} \cup \{\mathrm{out}(a) \mid a \in A \setminus S\} \\ & \cup \{\mathrm{defeated}(a) \mid b \in S, (b, a) \in R\} \\ & \cup \{\mathrm{not\_defended}(a) \mid (b, a) \in R, \nexists c \in S : (c, b) \in R\}. \end{aligned}$$

Since S is conflict free, the constraint in π_{cf} does not fire. Hence, $I \in AS((\Pi_{cf} \cup \Pi_{not\_defended})(\widehat{AF}))$ also holds. Now, we show that for each interpretation $I \in AS((\Pi_{cf} \cup \Pi_{not\_defended})(\widehat{AF}))$, there exists a conflict free set $S \subseteq A$, such that $S = \sigma(I)$. The module π_{cf} only consists of one constraint. Hence, no new facts are derived. Therefore, $I \in AS(\Pi_{not\_defended}(\widehat{AF}))$ has to hold. By Proposition 5.1.8, we know that there exists a set $S \subseteq A$, such that $S = \sigma(I)$. □

Proof. (of Proposition 5.1.9) By Proposition 5.1.10, we know that there exists an interpretation $J \in AS((\Pi_{cf} \cup \Pi_{not\_defended})(\widehat{AF}))$, such that

$$\begin{aligned} J &= \widehat{AF}_{defeat} \cup \{\text{in}(a) \mid a \in S\} \cup \{\text{out}(a) \mid a \in A \setminus S\} \\ &\quad \cup \{\text{defeated}(a) \mid b \in S, (b,a) \in R\} \\ &\quad \cup \{\text{not\_defended}(a) \mid (b,a) \in R, \nexists c \in S : (c,b) \in R\}. \end{aligned}$$

Assume that there exists an interpretation $I \supset J$, such that

$$I = J \cup \{\text{adm.}\}$$

Since S is admissible, it does not contain arguments which are not defended. Therefore, in(a) and not_defended(a) can not hold for any $a \in S$. So, the constraint in π_{adm} does not fire. Hence, $I \in AS(\Pi_{adm}(\widehat{AF}))$ holds. Now, we show that for each interpretation $I \in AS(\Pi_{adm}(\widehat{AF}))$ there exists a set $S \in adm(AF)$, such that $S = \sigma(I)$. Consider an interpretation $J \subset I$, such that

$$J = I \setminus \{\text{adm}\}.$$

It is obvious that $J \in AS((\Pi_{cf} \cup \Pi_{not\_defended})(\widehat{AF}))$. Hence, by Proposition 5.1.9, we know that there exists a set $S \subseteq A$, such that $S = \sigma(I) = \sigma(J)$. □

Complete Sets

The module π_{comp} eliminates all answer sets which do not correspond to $comp(AF)$.

Definition 5.1.13. *The module π_{comp} consists of the following constraint:*

$$\pi_{comp} \quad = \{ \quad \leftarrow \text{out}(X), \textit{not}\ \text{not\_defended}(X), \text{comp}\}.$$

We know that $\pi_{cf} \triangleright \pi_{comp}$ and $\pi_{not\_defended} \triangleright \pi_{comp}$ holds. From Proposition 5.1.1, we can conclude that $(\pi_{cf} \cup \pi_{not\_defended}) \triangleright \pi_{comp}$ holds. Hence, we can conclude from Proposition 5.1.2 and Definition 5.1.3 that the answer sets of the program $\Pi_{comp}(\widehat{AF})$ are defined as follows,

$$AS(\Pi_{comp}(\widehat{AF})) = \bigcup_{J \in AS((\Pi_{cf} \cup \Pi_{not\_defended})(\widehat{AF}))} (AS(J \cup \pi_{comp} \cup \pi_{ext(comp)})).$$

Hence, the correspondence between the answer sets of the program $\Pi_{comp}(\widehat{AF})$ and the complete sets of arguments is given in Proposition 5.1.11.

Proposition 5.1.11. *Let $AF = (A, R)$. For each set $S \in comp(AF)$, there exists an interpretation $I \in AS(\Pi_{comp}(\widehat{AF}))$, such that*

$$\begin{aligned} I &= \widehat{AF}_{defeat} \cup \{\text{comp.}\} \cup \{\text{in}(a) \mid a \in S\} \cup \{\text{out}(a) \mid a \in A \setminus S\} \\ &\cup \{\text{defeated}(a) \mid b \in S, (b,a) \in R\} \\ &\cup \{\text{not\_defended}(a) \mid (b,a) \in R, \nexists c \in S : (c,b) \in R\}. \end{aligned}$$

For each interpretation $I \in AS(\Pi_{comp}(\widehat{AF}))$ there exists a set $S \in comp(AF)$, such that $S = \sigma(I)$.

Proof. By Proposition 5.1.10, we know that there exists an interpretation $J \in AS((\Pi_{cf} \cup \Pi_{not\_defended})(\widehat{AF}))$, such that

$$\begin{aligned} J &= \widehat{AF}_{defeat} \cup \{\text{in}(a) \mid a \in S\} \cup \{\text{out}(a) \mid a \in A \setminus S\} \\ &\cup \{\text{defeated}(a) \mid b \in S, (b,a) \in R\} \\ &\cup \{\text{not\_defended}(a) \mid (b,a) \in R, \nexists c \in S : (c,b) \in R\}. \end{aligned}$$

Assume that there exists an interpretation $I \supset J$, such that

$$I = J \cup \{\text{comp.}\}$$

Since S is a complete set, every arguments which is defended by S, belongs to S. Hence, the constraint in π_{comp} does not fire. Hence, $I \in AS(\Pi_{comp}(\widehat{AF}))$ holds. Now, we show that for each interpretation $I \in AS(\Pi_{comp}(\widehat{AF}))$ there exists a set $S \in comp(AF)$, such that $S = \sigma(I)$. Consider an interpretation $J \subset I$, such that

$$J = I \setminus \{\text{comp}\}.$$

It is obvious that $J \in AS((\Pi_{cf} \cup \Pi_{not\_defended})(\widehat{AF}))$. Hence, by Proposition 5.1.9, we know that there exists a set $S \subseteq A$, such that $S = \sigma(I) = \sigma(J)$. □

Stable Sets

The module π_{stable} eliminates all answer sets which do not correspond to $stable(AF)$.

Definition 5.1.14. *The module π_{stable} consists of the following constraint:*

$$\pi_{stable} = \{ \quad \leftarrow \text{out}(X), \textit{not}\, \text{defeated}(X), \text{stable}\}.$$

We know that $\pi_{cf} \rhd \pi_{stable}$ and $\pi_{defeated} \rhd \pi_{stable}$ holds. From Proposition 5.1.1, we can conclude that $(\pi_{cf} \cup \pi_{defeated}) \rhd \pi_{stable}$ holds. Hence, we can conclude from Proposition 5.1.2 and Definition 5.1.3 that the answer sets of the program $\Pi_{stable}(\widehat{AF})$ are defined as follows,

$$AS(\Pi_{stable}(\widehat{AF})) = \bigcup_{J \in AS((\Pi_{cf} \cup \Pi_{defeated})(\widehat{AF}))} (AS(J \cup \pi_{stable} \cup \pi_{ext(stable)})).$$

Hence, the correspondence between the answer sets of the program $\Pi_{stable}(\widehat{AF})$ and the stable sets of arguments is given in Proposition 5.1.12.

Proposition 5.1.12. *Let $AF = (A, R)$. For each set $S \in stable(AF)$, there exists an interpretation $I \in AS(\Pi_{stable}(\widehat{AF}))$, such that*

$$\begin{aligned} I &= \widehat{AF}_{defeat} \cup \{\text{stable.}\} \cup \{\text{in}(a) \mid a \in S\} \cup \{\text{out}(a) \mid a \in A \setminus S\} \\ &\quad \cup \{\text{defeated}(a) \mid b \in S, (b, a) \in R\}. \end{aligned}$$

For each interpretation $I \in AS(\Pi_{stable}(\widehat{AF}))$ there exists a set $S \in stable(AF)$, such that $S = \sigma(I)$.

Proof. By Proposition 5.1.7, we can conclude that there exists an interpretation $J \in AS(\Pi_{defeated}(\widehat{AF}))$, such that

$$\begin{aligned} J &= \widehat{AF}_{defeat}\{\text{in}(a) \mid a \in S\} \cup \{\text{out}(a) \mid a \in A \setminus S\} \\ &\quad \cup \{\text{defeated}(a) \mid b \in S, (b, a) \in R\}. \end{aligned}$$

This holds especially for all conflict free sets $S \subseteq A$. Therefore, $J \in AS((\Pi_{cf} \cup \Pi_{defeated})(\widehat{AF}))$ also holds. Assume that there exists an interpretation $I \supset J$, such that

$$I = J \cup \{\text{stable.}\}$$

Since $S \in stable(AF)$, there does not exist any argument $a \in A \setminus S$, such that a is not defeated. Therefore, the constraint in π_{stable} does not fire. Hence, $I \in AS(\Pi_{stable}(\widehat{AF}))$ holds. Now, we show that for each interpretation $I \in AS(\Pi_{stable}(\widehat{AF}))$ there exists a set $S \in stable(AF)$, such that $S = \sigma(I)$. Consider an interpretation $J \subset I$, such that

$$J = I \setminus \{\text{stable}\}.$$

It is obvious that $J \in AS(\Pi_{defeated}(\widehat{AF}))$. Hence, by Proposition 5.1.7, we know that there exists a set $S \subseteq A$, such that $S = \sigma(I) = \sigma(J)$. □

Defining an Order over the Arguments

The module π_{ord} allows us to compute an order over the arguments. It derives infimum, supremum and successor predicates from the arguments which are needed for the grounded and preferred extensions.

Definition 5.1.15. *The module π_{ord} consists of the following rules:*

$$
\begin{aligned}
\pi_{ord} \quad = \{ \quad & \mathrm{lt}(X,Y) \leftarrow \arg(X), \arg(Y), X < Y; \\
& \mathrm{nsucc}(X,Z) \leftarrow \mathrm{lt}(X,Y), \mathrm{lt}(Y,Z); \\
& \mathrm{succ}(X,Y) \leftarrow \mathrm{lt}(X,Y), \mathit{not}\, \mathrm{nsucc}(X,Y); \\
& \mathrm{ninf}(Y) \leftarrow \mathrm{lt}(X,Y); \\
& \inf(X) \leftarrow \arg(X), \mathit{not}\, \mathrm{ninf}(X); \\
& \mathrm{nsup}(X) \leftarrow \mathrm{lt}(X,Y); \\
& \sup(X) \leftarrow \arg(X), \mathit{not}\, \mathrm{nsup}(X)\}.
\end{aligned}
$$

The predicate $<$ is supplied by DLV and fixes an order over the the arguments. We get a fixed sequence (with respect to $<$) of arguments $a_1, \dots, a_n$ with the minimal argument $min_< = a_1$ denoted by $\inf(a_1)$, the maximal argument $max_< = a_n$ denoted by $\sup(a_n)$ and the successor of an argument $s_<(a_i) = a_{i+1}$ denoted by $\mathrm{succ}(a_i, a_{i+1})$. For two arguments $a_i, a_j \in a_1, \dots, a_n$ we say $a_i < a_j$ if $i < j$ denoted by $\mathrm{lt}(a_i, a_j)$.

From Proposition 5.1.2, Definition 5.1.2 and the partial order relation pictured in Figure 5.1, we can conclude that the answer sets of the program $\Pi_{ord}(\widehat{AF})$ are defined as follows,

$$AS(\Pi_{ord}(\widehat{AF})) = AS(\pi_{ord} \cup \widehat{AF}).$$

The program $\Pi_{ord}(\widehat{AF})$ contains stratified negation, and therefore it has one single answer set denoted by $\widehat{AF}_{ord}$.

Example 5.1.1. *Let $AF = (A, R)$, with $A = \{e, f, g, h, s, w\}$ and $R = \{(e,s), (f,e), (h,g), (w,h), (w,s), (e,w)\}$.*

$$
\begin{aligned}
\widehat{AF} \quad = \{ \quad & \arg(e), \arg(f), \arg(g), \arg(h), \arg(s), \arg(w), \mathrm{attack}(e,s), \\
& \mathrm{attack}(f,e), \mathrm{attack}(h,g), \mathrm{attack}(w,h), \mathrm{attack}(w,s), \mathrm{attack}(e,w)\}
\end{aligned}
$$

The unique answer set of the program $\Pi_{ord}(\widehat{AF})$ contains $\widehat{AF}$ together with all the pred-

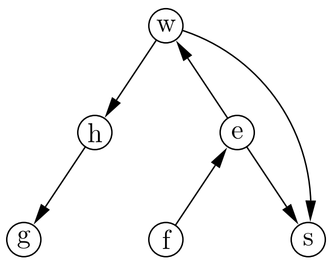


Figure 5.2: Graph representation from the example AF.

icate derived in the module π_{ord}.

$$\begin{aligned}\widehat{AF}_{ord} \;=\;& \widehat{AF} \cup \{\mathrm{inf}(e), \mathrm{sup}(w), \mathrm{lt}(e,f), \mathrm{lt}(e,g), \mathrm{lt}(f,g), \mathrm{lt}(e,h), \mathrm{lt}(f,h), \mathrm{lt}(g,h),\\ & \mathrm{lt}(e,s), \mathrm{lt}(f,s), \mathrm{lt}(g,s), \mathrm{lt}(h,s), \mathrm{lt}(e,w), \mathrm{lt}(f,w), \mathrm{lt}(g,w), \mathrm{lt}(h,w), \mathrm{lt}(s,w),\\ & \mathrm{nsup}(e), \mathrm{nsup}(f), \mathrm{nsup}(g), \mathrm{nsup}(h), \mathrm{nsup}(s), \mathrm{ninf}(f), \mathrm{ninf}(g), \mathrm{ninf}(h), \mathrm{ninf}(s),\\ & \mathrm{ninf}(w), \mathrm{nsucc}(e,g), \mathrm{nsucc}(e,h), \mathrm{nsucc}(e,s), \mathrm{nsucc}(e,w), \mathrm{nsucc}(f,h),\\ & \mathrm{nsucc}(f,s), \mathrm{nsucc}(f,w), \mathrm{nsucc}(g,s), \mathrm{nsucc}(g,w), \mathrm{nsucc}(h,w),\\ & \mathrm{succ}(e,f), \mathrm{succ}(f,g), \mathrm{succ}(g,h), \mathrm{succ}(h,s), \mathrm{succ}(s,w)\}\end{aligned}$$

Grounded Extension

To compute the grounded extension for a given argumentation framework $AF = (A, R)$, we need to encode the operator Γ_{AF} as defined in Definition 2.1.7. Therefore we will use a stratified program. Note that here we are not able to first guess a candidate for the extension and then check whether the guess satisfies certain conditions. Instead, we "fill" the in($\cdot$)-predicate according to the definition of the operator Γ_{AF}.

Definition 5.1.16. *The module π_{ground} consists of the following rules:*

$$\begin{aligned}\pi_{ground} = \{\;& \mathrm{defended\_upto}(X,Y) \leftarrow \mathrm{inf}(Y), arg(X), \mathit{not}\,\mathrm{defeat}(Y,X),\\ &\qquad \mathrm{ground}; && (5.4)\\ & \mathrm{defended\_upto}(X,Z) \leftarrow \mathrm{succ}(Y,Z), \mathrm{defended\_upto}(X,Y),\\ &\qquad \mathit{not}\,\mathrm{defeat}(Z,X), \mathrm{ground}; && (5.5)\\ & \mathrm{defended\_upto}(X,Y) \leftarrow \mathrm{inf}(Y), \mathrm{in}(Z), \mathrm{defeat}(Z,Y),\\ &\qquad \mathrm{defeat}(Y,X); && (5.6)\\ & \mathrm{defended\_upto}(X,Z) \leftarrow \mathrm{succ}(Y,Z), \mathrm{defended\_upto}(X,Y),\\ &\qquad \mathrm{in}(V), \mathrm{defeat}(V,Z), \mathrm{defeat}(Z,X); && (5.7)\\ & \mathrm{defended}(X) \leftarrow \mathrm{sup}(Y), \mathrm{defended\_upto}(X,Y), \mathrm{ground}; && (5.8)\\ & \mathrm{in}(X) \leftarrow \mathrm{defended}(X), \mathrm{ground}\}. && (5.9)\end{aligned}$$

In Definition 2.1.7 we defined the grounded extension $ground(AF)$ over the least fix-point of the operator $\Gamma_{AF}(S) = \{a \in A \mid a \text{ is defended by } S \text{ in } AF\}$. This operator works as follows:
In the first iteration the operator Γ_{AF} contains all arguments $a \in A$ which are not defeated:

$$\Gamma^0_{AF} = \Gamma_{AF}(\emptyset) = \{a \in A \mid \nexists b : (b, a) \in R\}.$$

In the following iterations all arguments are added which are defended by the arguments contained in Γ^{i-1}_{AF}:

$$\Gamma^i_{AF} = \Gamma_{AF}(\Gamma^{i-1}_{AF}) = \{a \in A \mid a \text{ is defended by } \Gamma^{i-1}_{AF}\}.$$

The iteration stops if no new arguments can be added:

$$\Gamma^i_{AF} = \Gamma^{i-1}_{AF}$$

which is the least fix-point *lfp* representing the grounded extension of AF $ground(AF)$.

From Proposition 5.1.2, Definition 5.1.3 and the partial order relation pictured in Figure 5.1, we can conclude that the answer sets of the program $\Pi_{ground}(\widehat{AF})$ are defined as follows,

$$AS(\Pi_{ground}(\widehat{AF})) = AS(\pi_{ground} \cup \widehat{AF}_{ord}\widehat{AF}_{defeat} \cup A_{ext(ground)}).$$

The program $\Pi_{ground}(\widehat{AF})$ contains stratified negation, and therefore it has one single answer set denoted by $\widehat{AF}_{ground}$.

In order to prove the correspondence between the operator Γ_{AF}, and the program $\Pi_{ground}(\widehat{AF})$, we define the operator $\Delta_{\widehat{AF}}$ representing the answer set $\widehat{AF}_{ground}$ as follows.

Definition 5.1.17. *In the first iteration, the operator* $\Delta^0_{\widehat{AF}} = \Delta_{\widehat{AF}}(\emptyset)$ *contains all facts from the answer set* $A_{ord}(\widehat{AF}) \cup \{\text{ground.}\} \cup \text{defended\_upto}(a, b)$, $\text{defended}(a)$, $\text{in}(a)$ *for all arguments a which are not defeated; formally*

$$\begin{aligned}
\Delta^0_{\widehat{AF}} = \; & \widehat{AF}_{ord} \cup \{\text{ground.}\} \\
& \cup\{\text{defended\_upto}(a, b) \mid \forall a_j < b : (a_j, a) \notin R, (b, a) \notin R\} \\
& \cup\{\text{defended}(a) \mid \forall a_j \in A : (a_j, a) \notin R\} \\
& \cup\{\text{in}(a) \mid \forall a_j \in A : (a_j, a) \notin R\}.
\end{aligned}$$

In the ith iteration, the operator $\Delta^i_{\widehat{AF}} = \Delta_{\widehat{AF}}(\Delta^{i-1}_{\widehat{AF}})$ *contains all facts from the previous iteration* $\Delta^{i-1}_{\widehat{AF}}$*, plus* $\{\text{defended\_upto}(a, b)\}$, $\{\text{defended}(a)\}$, *and* $\{\text{in}(a)\}$ *for all arguments* *a defended by those contained in* $\Delta^{i-1}_{\widehat{AF}}$.

$$\begin{aligned}
\Delta^{i}_{\widehat{AF}} &= \Delta^{i-1}_{\widehat{AF}} \\
&\cup\{\text{defended\_upto}(a,b) \mid \forall b \in A \textit{ such that } (a_j < b) \textit{ and } \text{in}(a_j) \in \Delta^{i-1}_{\widehat{AF}}, \\
&\qquad\qquad \exists\ \text{in}(c) \in \Delta^{i-1}_{\widehat{AF}} : (b,a) \in R, (c,b) \in R\} \\
&\cup\{\text{defended}(a) \mid \forall b \in A \textit{ such that } (a_j < b) \textit{ and } \text{in}(a_j) \in \Delta^{i-1}_{\widehat{AF}}, \\
&\qquad\qquad \exists\ \text{in}(c) \in \Delta^{i-1}_{\widehat{AF}} : (b,a) \in R, (c,b) \in R\} \\
&\cup\{\text{in}(a) \mid \forall b \in A \textit{ such that } (a_j < b) \textit{ and } \text{in}(a_j) \in \Delta^{i-1}_{\widehat{AF}}, \\
&\qquad \exists\ \text{in}(c) \in \Delta^{i-1}_{\widehat{AF}} : (b,a) \in R, (c,b) \in R\}
\end{aligned}$$

Proposition 5.1.13. *Let* $AF = (A, R)$*, the answer set* $\widehat{AF}_{ground}$ *for the program* $\Pi_{ground}(\widehat{AF})$ *and the operator* $\Delta_{\widehat{AF}}$ *defined as above, then it holds that*

$$\widehat{AF}_{ground} = \mathit{lfp}(\Delta_{\widehat{AF}}).$$

Proof. In the first iteration we will apply the Rules (5.4), (5.5), (5.8) and (5.9). Hence $\Delta_{\widehat{AF}}$ will consist of the union of the answer sets $\widehat{AF}_{ord}$, $\widehat{AF}_{defeat}$ and $A_{ext(ground)}$) and the predicates derived from those rules $r \in \mathcal{G}r(\Pi_{ground})$ with $\widehat{AF}_{ord} \cup \widehat{AF}_{defeat} \cup A_{ext(ground)}$ satisfies $B(r)$

$$\begin{aligned}
\Delta^{0}_{\widehat{AF}} &= AS(\widehat{AF}_{ord} \cup A_{ext(ground)}) \\
&\quad \cup\{H(r) \mid r \in \mathcal{G}r(\Pi_{ground}(\widehat{AF})), \widehat{AF}_{ord} \cup \widehat{AF}_{defeat} \cup A_{ext(ground)} \text{ satisfies } B(r)\} \\
&= \widehat{AF}_{ord} \cup \{\text{ground.}\} \\
&\quad \cup\{\text{defended\_upto}(a,b) \mid \nexists\ a_j < b : \text{defeat}(a_j,a) \in \widehat{AF}_{ord}, \\
&\qquad\qquad\qquad \text{defeat}(b,a) \in \widehat{AF}_{ord}\} \\
&\quad \cup\{\text{defended}(a) \mid \nexists\ a_j < b : \text{defeat}(a_j,a) \in \widehat{AF}_{ord}\} \\
&\quad \cup\{\text{in}(a) \mid \nexists\ a_j < b : \text{defeat}(a_j,a) \in \widehat{AF}_{ord}\}.
\end{aligned}$$

In this step in$(\cdot)$ represents those arguments which are not defeated in AF. In the next iterations the Rules (5.6),(5.7), (5.8) and (5.9) will add all arguments which are defended by arguments c, with $\text{in}(c) \in \Delta^{i-1}_{\widehat{AF}}$:

$$\begin{aligned}
\Delta^{i}_{\widehat{AF}} &= \Delta^{i-1}_{\widehat{AF}} \cup \{H(r) \mid r \in \mathcal{G}r(\Pi_{ground}), \Delta^{i-1}_{\widehat{AF}} \text{ satisfies } B(r)\} \\
&= \Delta^{i-1}_{\widehat{AF}} \\
&\quad \cup \{\text{defended\_upto}(a,b) \mid \forall b \in A \text{ such that } (a_j < b) \text{ and } \text{in}(a_j) \in \Delta^{i-1}_{\widehat{AF}}, \\
&\qquad\qquad \exists\ \text{in}(c) \in \Delta^{i-1}_{\widehat{AF}} : \text{defeat}(b,a) \in \Delta^{i-1}_{\widehat{AF}}, \\
&\qquad\qquad\qquad \text{defeat}(c,b) \in \Delta^{i-1}_{\widehat{AF}}\} \\
&\quad \cup \{\text{defended}(a) \mid \forall b \in A \text{ such that } (a_j < b) \text{ and } \text{in}(a_j) \in \Delta^{i-1}_{\widehat{AF}}, \\
&\qquad\qquad \exists\ \text{in}(c) \in \Delta^{i-1}_{\widehat{AF}} : \text{defeat}(b,a) \in \Delta^{i-1}_{\widehat{AF}}, \text{defeat}(c,b) \in \Delta^{i-1}_{\widehat{AF}}\} \\
&\quad \cup \{\text{in}(a) \mid \forall b \in A \text{ such that } (a_j < b) \text{ and } a_j \in \Delta^{i-1}_{\widehat{AF}}, \\
&\qquad\qquad \exists\ \text{in}(c) \in \Delta^{i-1}_{\widehat{AF}} : \text{defeat}(b,a) \in \Delta^{i-1}_{\widehat{AF}}, \text{defeat}(c,b) \in \Delta^{i-1}_{\widehat{AF}}\}
\end{aligned}$$

The module π_{ground} contains stratified negation, therefore the program $\Pi_{ground}(\widehat{AF})$ has one single answer set, and the iteration stops if $\Delta^{i}_{\widehat{AF}} = \Delta^{i-1}_{\widehat{AF}}$. The least fix-point $\mathit{lfp}(\Delta_{\widehat{AF}})$ contains $\widehat{AF}_{ord} \cup \{\text{ground.}\}$, plus all facts $\{\text{defended\_upto(.)}\}$, $\{\text{defended(.)}\}$, and $\{\text{in(.)}\}$ derived by $\pi_{ground} \cup \widehat{AF}_{ord} \cup \widehat{AF}_{defeat} \cup A_{ext(ground)}$. Hence

$$\mathit{lfp}(\Delta_{\widehat{AF}}) = AS(\Pi_{ground}(\widehat{AF}))$$

. $\square$

Consider the argumentation framework from Example 5.1.1.

$$\begin{aligned}
\widehat{AF} &= \{\mathrm{arg}(e), \mathrm{arg}(f), \mathrm{arg}(g), \mathrm{arg}(h), \mathrm{arg}(s), \mathrm{arg}(w), \mathrm{attack}(e,s),\\
&\quad \mathrm{attack}(f,e), \mathrm{attack}(h,g), \mathrm{attack}(w,h), \mathrm{attack}(w,s), \mathrm{attack}(e,w)\}\\
\widehat{AF}_{ord} &= \widehat{AF} \cup \{\mathrm{inf}(e), \mathrm{sup}(w), \mathrm{lt}(e,f), \ldots, \mathrm{lt}(s,w), \mathrm{succ}(e,f), \ldots, \mathrm{succ}(s,w)\}\\
A_{ext(ground)} &= \{\mathrm{ground}\}\\
\widehat{AF}_{defeat} &= \widehat{AF} \cup \{\mathrm{defeat}(e,s), \ldots, \mathrm{defeat}(e,w)\}\\
\Delta^0_{\widehat{AF}} &= \widehat{AF}_{ord} \cup A_{ext(ground)} \cup \{\mathrm{defended\_upto}(e,e), \mathrm{defended\_upto}(f,e),\\
&\quad \mathrm{defended\_upto}(g,e), \mathrm{defended\_upto}(h,e), \mathrm{defended\_upto}(f,f),\\
&\quad \mathrm{defended\_upto}(g,f), \mathrm{defended\_upto}(h,f), \mathrm{defended\_upto}(f,g),\\
&\quad \mathrm{defended\_upto}(g,g), \mathrm{defended\_upto}(h,g), \mathrm{defended\_upto}(f,h),\\
&\quad \mathrm{defended\_upto}(h,h), \mathrm{defended\_upto}(f,s), \mathrm{defended\_upto}(h,s),\\
&\quad \mathrm{defended\_upto}(f,w), \mathrm{defended}(f), \mathrm{in}(f)\}\\
\Delta^1_{\widehat{AF}} &= \Delta^0_{\widehat{AF}} \cup \{\mathrm{defended\_upto}(s,e), \mathrm{defended\_upto}(w,e),\\
&\quad \mathrm{defended\_upto}(s,f), \mathrm{defended\_upto}(w,f), \mathrm{defended\_upto}(s,g),\\
&\quad \mathrm{defended\_upto}(w,g), \mathrm{defended\_upto}(s,h), \mathrm{defended\_upto}(w,h),\\
&\quad \mathrm{defended\_upto}(s,s), \mathrm{defended\_upto}(w,s), \mathrm{defended\_upto}(w,w),\\
&\quad \mathrm{defended}(w), \mathrm{in}(w)\}\\
\Delta^2_{\widehat{AF}} &= \Delta^1_{\widehat{AF}} \cup \{\mathrm{defended\_upto}(g,h), \mathrm{defended\_upto}(g,s),\\
&\quad \mathrm{defended\_upto}(g,w), \mathrm{defended}(g), in(g)\}\\
\Delta^3_{\widehat{AF}} &= \Delta^2_{\widehat{AF}}
\end{aligned}$$

Therefore, we have $ground(AF) = \{f, g, w\}$.

According to the Propositions 5.1.13 and 5.1.14, we formulate the following lemma:

Lemma 5.1.1. *Let* $AF = (A, R)$. *Then, for every* $i \geq 0$,

$$\Gamma^i_{AF} = \{a \in A \mid \mathrm{in}(a) \in \Delta^i_{\widehat{AF}}\}.$$

Proof. **Induction Base $i = 0$:**
$\subseteq$: Let $a \in \Gamma^0_{AF}$. We show that $\mathrm{in}(a) \in \Delta^0_{\widehat{AF}}$. Since $a \in \Gamma^0_{\widehat{AF}}$, there is no $b \in A$ with $(b, a) \in R$. We show that for each $b \in A$ defended_upto(a, b) holds. This holds, since defeat$(b, a) \in A_{ord}$ iff $(b, a) \in R$. Thus, for each $b \in A$ with $(b, a) \notin R$, defended_upto$(a, b) \in \Delta^0_{\widehat{AF}}$ holds. Therefore, defended$(a) \in \Delta^0_{\widehat{AF}}$ and necessarily $\mathrm{in}(a) \in \Delta^0_{\widehat{AF}}$.
$\supseteq$: Let $a \in A$ such that $\mathrm{in}(a) \in \Delta^0_{\widehat{AF}}$. We will show that $a \in \Gamma^0_{AF}$. For any $a \in A$, if $\mathrm{in}(a) \in \Delta^0_{\widehat{AF}}$ then defended$(a) \in \Delta^0_{\widehat{AF}}$. This holds, since defended_upto(a, b) holds for each $b \in A$. Therefore, there is no $b \in A$ with $(b, a) \in R$ and $a \in \Gamma^0_{AF}$ must hold.

Induction Step: Suppose the assumption holds for all $j < i$; particularly we can assume that $\Gamma_{AF}^{i-1} = \{a \in A \mid \text{in}(a) \in \Delta_{\widehat{AF}}^{i-1}\}$.
$\subseteq$: It will do to show that for each $a \in \Gamma_{AF}^{i} \setminus \Gamma_{AF}^{i-1}$, $\text{in}(a) \in \Delta_{\widehat{AF}}^{i}$ holds.
Since $a \in \Gamma_{AF}^{i}$, it must hold that for each $b \in A$ with $(b, a) \in R$ there exists an $c \in \Gamma_{AF}^{i-1}$ with $(c, b) \in R$. Therefore $\text{in}(c) \in \Delta_{\widehat{AF}}^{i-1}$ holds by induction hypothesis. Since for all $b \in A$, $\text{defeat}(b, a)$ and $\text{defeat}(c, b) \in \Delta_{\widehat{AF}}^{i-1}$, then $\text{defended\_upto}(a, b) \in \Delta_{\widehat{AF}}^{i}$. Therefore, $\text{defended}(a) \in \Delta_{\widehat{AF}}^{i}$ and necessarily $\text{in}(a) \in \Delta_{\widehat{AF}}^{i}$.
$\supseteq$: Let $a \in A$ such that $\text{in}(a) \in \Delta_{\widehat{AF}}^{i} \setminus \Delta_{\widehat{AF}}^{i-1}$. We will show that $a \in \Gamma_{AF}^{i}$ holds. Since $a \in A$, if $\text{in}(a) \in \Delta_{\widehat{AF}}^{i}$, $\text{defended}(a) \in \Delta_{\widehat{AF}}^{i}$. This holds, since for all $b \in A$, $\text{defended\_upto}(a, b) \in \Delta_{\widehat{AF}}^{i}$. Therefore, for any $b \in A$ with $\text{defeat}(b, a)$ and $\text{defeat}(c, b) \in \Delta_{\widehat{AF}}^{i-1}$ it holds that $\text{in}(c) \in \Delta_{\widehat{AF}}^{i-1}$. Hence, by induction hypothesis $c \in \Gamma_{AF}^{i-1}$ holds. Therefore, for each $b \in A$ with $(b, a) \in R$ there exists an $c \in \Gamma_{AF}^{i-1}$ with $(c, b) \in R$. It follows that $a \in \Gamma_{AF}^{i}$ holds. □

Proposition 5.1.14. *Let $AF = (A, R)$. There is a one-to-one correspondence between $ground(AF)$ and $AS(\Pi_{ground}(\widehat{AF}))$.*

Preferred Extensions

The module π_{prefex} assures that just answer sets survive which represent the preferred extensions. We recall that the preferred extensions are the maximal admissible extensions.

Definition 5.1.18. *The module π_{prefex} consists of the following rules:*

$$
\begin{aligned}
\pi_{prefex} = \{ \quad & \text{inN}(X) \vee \text{outN}(X) \leftarrow \text{out}(X), \text{prefex}; && (5.10)\\
& \text{inN}(X) \leftarrow \text{in}(X), \text{prefex}; && (5.11)\\
& \text{eq\_upto}(Y) \leftarrow \text{inf}(Y), \text{in}(Y), \text{inN}(Y); && (5.12)\\
& \text{eq\_upto}(Y) \leftarrow \text{inf}(Y), \text{out}(Y), \text{outN}(Y); && (5.13)\\
& \text{eq\_upto}(Y) \leftarrow \text{succ}(Z, Y), \text{in}(Y), \text{inN}(Y), \text{eq\_upto}(Z); && (5.14)\\
& \text{eq\_upto}(Y) \leftarrow \text{succ}(Z, Y), \text{out}(Y), \text{outN}(Y), \text{eq\_upto}(Z); && (5.15)\\
& \text{eq} \leftarrow \text{sup}(Y), \text{eq\_upto}(Y); && (5.16)\\
& \text{undefeated\_upto}(X, Y) \leftarrow \text{inf}(Y), \text{outN}(X), \text{outN}(Y); && (5.17)\\
& \text{undefeated\_upto}(X, Y) \leftarrow \text{inf}(Y), \text{outN}(X), \textit{not}\ \text{defeat}(Y, X); && (5.18)\\
& \text{undefeated\_upto}(X, Y) \leftarrow \text{succ}(Z, Y), \text{undefeated\_upto}(X, Z), && (5.19)\\
& \qquad \text{outN}(Y); && (5.20)
\end{aligned}
$$

$$\begin{aligned}
\text{undefeated\_upto}(X,Y) &\leftarrow \text{succ}(Z,Y), \text{undefeated\_upto}(X,Z), && (5.21)\\
& \qquad \textit{not}\, \text{defeat}(Y,X); && (5.22)\\
\text{undefeated}(X) &\leftarrow \text{sup}(Y), \text{undefeated\_upto}(X,Y); && (5.23)\\
\text{spoil} &\leftarrow \text{eq}; && (5.24)\\
\text{spoil} &\leftarrow \text{inN}(X), \text{inN}(Y), \text{defeat}(X,Y); && (5.25)\\
\text{spoil} &\leftarrow \text{inN}(X), \text{outN}(X), \text{arg}(X); && (5.26)\\
\text{spoil} &\leftarrow \text{inN}(X), \text{outN}(Y), \text{defeat}(Y,X), \text{undefeated}(Y); && (5.27)\\
\text{inN}(X) &\leftarrow \text{spoil}, \text{arg}(X); && (5.28)\\
\text{outN}(X) &\leftarrow \text{spoil}, \text{arg}(X); && (5.29)\\
\bot &\leftarrow \textit{not}\, \text{spoil}\}. && (5.30)
\end{aligned}$$

We know that $\pi_{adm} \rhd \pi_{prefex}$ and $\pi_{ord} \rhd \pi_{prefex}$ holds. From Proposition 5.1.1, we can conclude that $(\pi_{adm} \cup \pi_{ord}) \rhd \pi_{prefex}$ holds. Hence, we can conclude from Proposition 5.1.2 and Definition 5.1.3 that the answer sets of the program $\Pi_{prefex}(\widehat{AF})$ are defined as follows,

$$AS(\Pi_{prefex}(\widehat{AF})) = \bigcup_{J \in AS((\Pi_{adm} \cup \Pi_{ord})(\widehat{AF}))} (AS(J \cup \pi_{prefex} \cup \pi_{ext(prefex)})).$$

Hence, the correspondence between the answer sets of the program $\Pi_{prefex}(\widehat{AF})$ and the preferred sets of arguments is given in Proposition 5.1.15.

Proposition 5.1.15. *Let $AF = (A,R)$. For each set $S \in pref(AF)$, there exists an interpretation $I \in AS(\Pi_{prefex}(\widehat{AF}))$, such that*

$$\begin{aligned}
I \;=\; & \widehat{AF}_{ord} \cup \{\text{spoil.}\} \cup \{\text{prefex.}\} \cup \{\text{defeat}(a,b) \mid (a,b) \in R\}\\
& \cup \{\text{in}(a) \mid a \in S\} \cup \{\text{out}(a) \mid a \in A \setminus S\}\\
& \cup \{\text{inN}(a) \mid a \in A\} \cup \{\text{outN}(a) \mid a \in A\}\\
& \cup\ \{\text{defeated}(a) \mid b \in S, (b,a) \in R\}\\
& \cup\ \{\text{not\_defended}(a) \mid (b,a) \in R, \nexists c \in S : (c,b) \in R\}\\
& \cup\ \{\text{eq\_upto}(a) \mid a \in A\} \cup \{eq\}\\
& \cup\ \{\text{undefeated\_upto}(a,b) \mid a,b \in A\} \cup \{\text{undefeated}(a) \mid a \in A\}.
\end{aligned}$$

For each interpretation $I \in AS(\Pi_{prefex}(\widehat{AF}))$ there exists a set $S \in pref(AF)$, such that $S = \sigma(I)$.

Proof. Since $S \in pref(AF)$, and the fact that every preferred extension is also an admissible extension, we know that $S \in adm(AF)$ also holds. By Proposition 5.1.9 we know, that there exists an interpretation $J \in AS(\Pi_{adm}(\widehat{AF}))$, such that $\sigma(J) = S$. Let I be given as above, with that particular S. We show that,

a) I is a model of $\Pi_{prefex}(\widehat{AF})$, and

b) $\forall K \subset I$, K is not a model $(\Pi_{prefex}(\widehat{AF}))^I$.

Ad a): We know that $J \subseteq I$ holds. Hence, I is a model of $\Pi_{adm}(\widehat{AF})$, since J is an answer set of $\Pi_{adm}(\widehat{AF})$, and no atom of $I \setminus J$ occurs in $\Pi_{adm}(\widehat{AF})$. Moreover, all other rules (5.10) to (5.30) are satisfied by I, since all heads of the corresponding ground rules occur in I. Hence, I is a model of $\Pi_{prefex}(\widehat{AF})$.
Ad b): Towards a contradiction, suppose such a K exists.

- First suppose that $(K \cap J) \subset J$ holds. If K is a model of $(\Pi_{adm}(\widehat{AF}))^I$ then $(K \cap J)$ is also a model of $(\Pi_{adm}(\widehat{AF}))^I$. This and the fact that $(K \cap J) \subset J$, leads to a contradiction to $J \in AS(\Pi_{adm})$.

- Now, suppose that $(K \cap J) = J$ holds. Then, at least some atoms of the form spoil, inN(.) and outN(.) from $I \setminus J$ are missing in K. Note that, whenever, for each argument $a \in A$, inN(a) and outN(a) are contained in K, then for all $a, b \in A$ also all atoms eq_upto(a), eq and undefeated_upto(a, b) are in K. According to Definition 5.1.7 we define a new operator σ' as follows

$$\sigma'(K) = \{a \mid \text{inN}(a) \in K\}.$$

  - If spoil $\in K$, then $K = I$ must hold (see Rules(5.28) and (5.29)). Hence, spoil $\notin K$.
  - By Rule (5.26), we know that for any $a \in A$, not both, inN(a) and outN(a), are contained in K.
  - By Rule (5.24), we know that $\{eq\}$ is not contained in K.
  - By Rule (5.25), we know that $\sigma'(K)$ is conflict free.
  - By Rule (5.27), we know that $\sigma'(K)$ is admissible.

  By Rule (5.10), we know that $\sigma'(K) \supseteq \sigma(K) = \sigma(I)$, and since $eq \notin K$, one can see that $\sigma'(K) \neq \sigma(K)$. Hence, $\sigma'(K) \supset \sigma(K)$ holds. We showed that there is a $S' = \sigma'(K)$, such that $S' \in adm(AF)$ and $S' \supset S$. This is a contradiction to $S \in pref(AF)$.

Now, we show that for each interpretation $I \in AS(\Pi_{prefex}(\widehat{AF}))$, there exists a set

$S \in \mathit{pref}(AF)$, such that $S = \sigma(I)$. Consider an interpretation $J \subset I$, such that

$$\begin{array}{rcl} J & = & I \setminus \{\mathrm{lt}(X,Y) \mid\ a,b \in A : a < b\} \cup \{\mathrm{nsucc}(a,c) \mid a,b,c \in A : a < b, b < c\} \\ & & \cup\ \{\mathrm{succ}(a,b) \mid a,b \in A : a < b,\ \nexists c \in A : b < c\} \\ & & \cup\ \{\mathrm{ninf}(b) \mid a,b \in A : a < b\} \\ & & \cup\ \{\mathrm{inf}(a) \mid a \in A,\ \nexists b \in A : b < a\} \\ & & \cup\ \{\mathrm{nsup}(a) \mid a,b \in A : a < b\} \\ & & \cup\ \{\mathrm{sup}(a) \mid a \in A,\ \nexists b \in A : a < b\} \\ & & \cup\ \{\mathrm{spoil.}\} \cup \{\mathrm{prefex.}\} \\ & & \cup\ \{\mathrm{inN}(a) \mid a \in A\} \cup \{\mathrm{outN}(a) \mid a \in A\} \\ & & \cup\ \{\mathrm{eq\_upto}(a) \mid a \in A\} \cup \{eq\} \\ & & \cup\ \{\mathrm{undefeated\_upto}(a,b) \mid a,b \in A\} \cup \{\mathrm{undefeated}(a) \mid a \in A\}. \end{array}$$

It is easy to see that $J \in AS(\Pi_{adm}(\widehat{AF}))$. Hence, by Proposition 5.1.9 we know that that there exists a set $S \in adm(AF)$, such that $S = \sigma(J)$. We can conclude that $\sigma(J) = \sigma(I) = S$ holds. For $S \in \mathit{pref}(AF)$, it remains to show that there is no $S' \supset S$, such that $S' \in adm(AF)$. Consider an interpretation $K \subset I$, such that

$$K \ = \ J \cup \{\mathrm{prefex.}\} \cup\ \{\mathrm{inN}(a) \mid a \in S'\} \cup \{\mathrm{outN}(a) \mid a \notin S'\} \cup\ K'$$

The set K' contains all further atoms over the grounding of the rules (5.12) to (5.23) according to the set S'. However spoil $\notin K$. We show that K is a model of $\Pi_{\mathit{prefex}}(\widehat{AF})^I$. First, K is a model of $\Pi_{adm}(\widehat{AF})^I$, since $J \in AS(\Pi_{adm}(\widehat{AF}))$. By definition of K', we only have to show that spoil is not derived by K. This holds since,

- $S' \neq S$ and thus Rule (5.24) cannot fire,
- S' is conflict free and thus Rule (5.25) cannot fire,
- either inN(a) or outN(a) are contained in K for each $a \in A$. Thus Rule (5.26) cannot fire,
- S' is admissible and thus Rule (5.27) cannot fire.

Hence, one can see that $K \subset I$ and K is a model of $\Pi_{\mathit{prefex}}(\widehat{AF})^I$. This is a contradiction to $I \in AS(\Pi_{\mathit{prefex}}(\widehat{AF}))$. □

5.2 Value Based Argumentation Framework

In order to compute value based argumentation frameworks, the encoding of the basic argumentation framework needs to be modified in the following way. We need to add some additional facts to the input database. Furthermore, we must add two modules to the preference relation. This new modules have the following partial order

$$\pi_{defeat} \prec \pi_{pref} \prec \pi_{valpref} \prec \widehat{AF}.$$

Representation of a Value Based Argumentation Framework

To represent a value based argumentation framework $AF = (A, R, V, val, valpref)$, we need to extend the module $\widehat{AF}$ from the basic argumentation framework, by facts for the values of arguments and the preference relation of the values.

Definition 5.2.1. *Given a value based argumentation framework AF, $\widehat{AF}$ consists of the following facts:*

$$\begin{aligned} \widehat{AF} \quad = \{ \quad & \text{arg}(a) \mid a \in A\} \cup \\ \{ \quad & \text{attack}(a, b) \mid (a, b) \in R\} \cup \\ \{ \quad & \text{val}(a, val\_a) \mid val\_a \in V, val(a) = val\_a\} \cup \\ \{ \quad & \text{valpref}(v1, v2) \mid (v1, v2) \in valpref\}. \end{aligned}$$

In the following, we use the modified version of $\widehat{AF}$ to represent a value based argumentation framework AF. As in the case of a basic argumentation framework, the program $\widehat{AF}$ of a value based argumentation framework, also has one single answer set $\widehat{AF}$.

Preference on Values

The preference relation is transitive. This will be assured by the module $\pi_{valpref}$.

Definition 5.2.2. *The module $\pi_{valpref}$ consists of the following rule:*

$$\pi_{valpref} \quad = \{ \quad \text{valpref}(X, Z) \leftarrow \text{valpref}(X, Y), \text{valpref}(Y, Z)\}.$$

It holds that $\widehat{AF} \rhd \pi_{valpref}$, and the partial order relation is

$$\pi_{valpref} \prec \widehat{AF}.$$

Therefore the answer sets of the program $\Pi_{valpref}(\widehat{AF})$ are defined as follows,

$$\begin{aligned} AS(\Pi_{valpref}(\widehat{AF})) &= AS(\widehat{AF} \cup \pi_{valpref}) \\ &= \widehat{AF} \cup \{\text{valpref}(a, c) \mid \exists b_1 \dots b_n \text{ such that } (a, b_1) \in valpref, \\ &\qquad (b_i, b_{i+1}) \in valpref, (b_n, c) \in valpref, \\ &\qquad \text{for } i \in \{1, \dots, n-1\}\}. \end{aligned}$$

The program $\Pi_{valpref}(\widehat{AF})$ has one single answer set denoted by $\widehat{AF}_{valpref}$.

Preference on Arguments

The preference relation of a value based argumentation framework is defined over values. In order to conclude which arguments are preferred over other ones, we derive the predicate *prefex*/2 for arguments as follows.

Definition 5.2.3. *The module π_{pref} consists of the following rules:*

$$\begin{aligned}\pi_{pref} \quad = \{ \quad & \mathrm{pref}(X,Y) \leftarrow \mathrm{valpref}(\mathit{Val\_X}, \mathit{Val\_Y}), \mathrm{val}(X, \mathit{Val\_X}), \mathrm{val}(Y, \mathit{Val\_Y});\\ & \mathrm{pref}(X,Z) \leftarrow \mathrm{pref}(X,Y), \mathrm{pref}(Y,Z)\}.\end{aligned}$$

Defeat

In a value based argumentation framework a defeat is successful, if for two arguments $a, b \in A$ it holds that $(a,b) \in R$ and the argument b is not preferred over the argument a. Therefore the module π_{defeat} from the basic argumentation framework needs to be modified as follows.

Definition 5.2.4. *The module π_{defeat} consists of the following rule:*

$$\pi_{defeat} \quad = \{ \quad \mathrm{defeat}(X,Y) \leftarrow \mathrm{attack}(X,Y), \mathit{not}\, \mathrm{pref}(Y,X)\}.$$

Note that all extensions from basic AFs can now be computed for VAFs accordingly. We just need to add the modules $\pi_{valpref}$ and π_{pref} to each $\Pi_\varepsilon(\widehat{AF})$, and modify the modules $\widehat{AF}$ and π_{defeat} as described above.

To compute preference based argumentation frameworks we do not need any new modules. We just need to add in the module $\widehat{AF}$, the facts for the preference relation. This means that for a given preference based argumentation framework $AF = (A, R, \mathit{Pref})$, the module $\widehat{AF}$ contains facts $\mathrm{pref}(a,b)$ for $(a,b) \in \mathit{Pref}$, in addition to the facts for the basic framework. The partial order for preference based argumentation frameworks is as follows

$$\pi_{defeat} \prec \pi_{pref} \prec \widehat{AF}.$$

5.3 Bipolar Argumentation Framework

In order to compute Bipolar Argumentation Frameworks, the encoding of the basic argumentation framework needs to be extended by the following modules. The partial order relations for bipolar argumentation frameworks are as follows.

$$\pi_{defeat} \prec \pi_{supported} \quad \prec \quad \pi_{support} \prec \widehat{AF}$$

In this section we provide the definitions of the new modules, but only sketch their formal functioning.

Representation of a Bipolar Argumentation Framework

To represent a bipolar argumentation framework $AF = (A, R_{def}, R_{sup})$, we need to extend the module $\widehat{AF}$ from the basic argumentation framework by the support relation.

Definition 5.3.1. *Given a bipolar argumentation framework AF, $\widehat{AF}$ consists of the following facts:*

$$\begin{aligned}\widehat{AF} \quad = \{ \quad & \text{arg}(a) \mid a \in A\} \cup \\ \{ \quad & \text{attack}(a,b) \mid (a,b) \in R_{def}\} \cup \\ \{ \quad & \text{support}(a,b) \mid (a,b) \in R_{sup}\}.\end{aligned}$$

Support

In bipolar argumentation frameworks supports are denoted by the support relation. The following module assures the transitivity of this relation.

Definition 5.3.2. *The module $\pi_{support}$ is defined as follows,*

$$\pi_{support} \quad = \{ \quad \text{support}(X,Z) \leftarrow \text{support}(X,Y), \text{support}(Y,Z)\}.$$

Supported Arguments

The module $\pi_{supported}$ derives arguments which are supported by a set $S \subseteq A$.

Definition 5.3.3. *The module $\pi_{supported}$ is defined as follows,*

$$\pi_{supported} \quad = \{ \quad \text{supported}(X) \leftarrow \text{in}(Y), \text{support}(Y,X)\}.$$

Defeat

The module π_{defeat} needs to be modified as follows.

Definition 5.3.4. *The module π_{defeat} is defined as follows,*

$$\begin{aligned}\pi_{defeat} \quad = \{ \quad & \text{defeat}(X,Y) \leftarrow \text{attack}(X,Y); \\ & \text{defeat}(X,Y) \leftarrow \text{attack}(Z,Y), \text{support}(X,Z); \\ & \text{defeat}(X,Y) \leftarrow \text{attack}(X,Z), \text{support}(Z,Y)\}.\end{aligned}$$

Note that the d-admissible extensions for BAFs are computed in a similar way as for AFs. We just need to add the modules $\pi_{support}$ and $\pi_{supported}$ to the program $\Pi_{adm}(\widehat{AF})$, and modify the modules $\widehat{AF}$ and π_{defeat} as described above.

Safe Sets

According to the definition of safe sets for bipolar argumentation frameworks the module π_{safe} eliminates all sets which are not safe.

Definition 5.3.5. *The module π_{safe} is defined as follows,*

$$\pi_{safe} = \{\leftarrow \text{supported}(a), \text{defeated}(a)\}.$$

The partial order relations for the module π_{safe} are as follows

$$\begin{aligned} \pi_{safe} &\prec \pi_{\text{defeated}} \\ \pi_{safe} &\prec \pi_{supported}. \end{aligned}$$

Closed

The module π_{closed} eliminates all answer sets which are not closed under R_{sup}.

Definition 5.3.6. *The module π_{closed} contains the following two constraints.*

$$\begin{aligned} \pi_{closed} = \{ &\leftarrow \text{support}(X, Y), \text{in}(X), \text{out}(Y); \\ &\leftarrow \text{support}(X, Y), \text{out}(X), \text{in}(Y)\}. \end{aligned}$$

The partial order relation for the module π_{closed} is as follows

$$\pi_{closed} \prec \pi_{support}.$$

s-admissible Sets

With the module π_{sadm}, we eliminate all answer sets which are not s-admissible.

Definition 5.3.7. *The module π_{sadm} is defined as follows,*

$$\pi_{sadm} = \{ \leftarrow \text{in}(x), \text{not\_defended}(X), \text{s\_adm}\}.$$

The program $\Pi_{sadm}(\widehat{AF})$ is defined as follows.

Definition 5.3.8. *Let $BAF = (A, R_{def}, R_{sup})$. The program $\Pi_{sadm}(\widehat{AF})$ represents the union of module π_{sadm} and its predecessors.*

$$\begin{aligned}\Pi_{sadm}(\widehat{AF}) \quad &= \quad \pi_{sadm} \cup \pi_{safe} \cup \pi_{supported} \cup \pi_{support} \cup \pi_{not\_defended} \cup \pi_{defeat} \\ &\cup \; \pi_{guess} \cup \widehat{AF} \cup \pi_{ext(sadm)}.\end{aligned}$$

Note that we use the modified modules $\widehat{AF}$ and π_{defeat}.

c-admissible Sets

With the module π_{cadm}, we eliminate all answer sets which are not c-admissible.

Definition 5.3.9. *The module π_{cadm} contains the following constraint,*

$$\pi_{cadm} \quad = \{ \quad \leftarrow \mathrm{support}(X, Y), \mathrm{in}(X), \mathrm{out}(Y), \mathrm{c\_adm}\}.$$

The program $\Pi_{cadm}(\widehat{AF})$ is defined as follows.

Definition 5.3.10. *Let $BAF = (A, R_{def}, R_{sup})$. The program $\Pi_{cadm}(\widehat{AF})$ represents the union of module π_{cadm} and its predecessors.*

$$\begin{aligned}\Pi_{cadm}(\widehat{AF}) \quad &= \quad \pi_{cadm} \cup \pi_{closed} \cup \pi_{support} \cup \pi_{not\_defended} \cup \pi_{defeat} \\ &\cup \; \pi_{guess} \cup \widehat{AF} \cup \pi_{ext(cadm)}.\end{aligned}$$

Note that here we also use the modified modules $\widehat{AF}$ and π_{defeat}.

The one-to-one correspondence between the d-admissible, s-admissible and c-admissible extensions, and the answer sets of the respective programs, can be shown in a similar way as done in the previous sections. Moreover, the programs for d-preferred, s-preferred and c-preferred extensions can be defined, by replacing the respective modules for admissible extensions (and their predecessors) to the program $\Pi_{prefex}(\widehat{AF})$.

5.4 Discussion

We briefly argue that our encodings are adequate in a complexity point of view. However, we restrict ourselves to the encodings for basic AFs. By the one-to-one correspondence between the answer sets and the extensions (see Theorem 1, and in particular Propositions 5.1.9, 5.1.11, 5.1.12, 5.1.13 and 5.1.15), we reduce the reasoning problems Cred_e and Skept_e, for $e \in \{stable,\ adm,\ prefex,\ comp,\ ground\}$ (see Section 4.2) to the respective problems $\models_c$ and $\models_s$ defined for answer set programming in Section 4.3. Recall

	stable	*adm*	*prefex*	*comp*	*ground*
Cred_e	$\pi_{stable}(\widehat{AF}) \models_c a$	$\pi_{adm}(\widehat{AF}) \models_c a$	$\pi_{adm}(\widehat{AF}) \models_c a$	$\pi_{comp}(\widehat{AF}) \models_c a$	$\pi_{ground}(\widehat{AF}) \models a$
Skept_e	$\pi_{stable}(\widehat{AF}) \models_s a$	(trivial)	$\pi_{prefex}(\widehat{AF}) \models_s a$	$\pi_{ground}(\widehat{AF}) \models a$	$\pi_{ground}(\widehat{AF}) \models a$

Table 5.2: Overview of the encodings of the reasoning tasks for $AF = (A, R)$ and $a \in A$.

that our program is fixed. The adequate encodings, depending on the chosen reasoning task, are depicted in Table 5.2 (see also [31]). Hence, the complexity of evaluating the respective encodings always coincides with the complexity of the encoded reasoning task (see Tables 4.1 and 4.2). For this reason, we understand our encodings as adequate.

Chapter 6

System Description

In this chapter, we describe the system ASPARTIX which has been implemented on top of the answer set programming system DLV. ASPARTIX is capable to compute extensions of several kinds of argumentation frameworks such as the basic AFs, as well as PAFs, VAFs and BAFs.

The program and some examples are available at

http://www.kr.tuwien.ac.at/research/systems/argumentation/

Table 6.1 gives an overview which extensions are possible for which argumentation frameworks. Note that the empty fields follow from the fact that not all semantics have been defined for all frameworks, i.e., (s-)admissible and (c-)admissible extensions are only meaningful for BAFs, whereas the complete and grounded extensions have not been introduced for VAFs and BAFs.

Extension	AF	PAF	VAF	BAF
stable	×	×	×	×
(d-)admissible	×	×	×	×
(s-)admissible				×
(c-)admissible				×
complete	×	×		
grounded	×	×		
(d-)preferred	×	×	×	×
(s-)preferred				×
(c-)preferred				×

Table 6.1: Extensions and argumentation frameworks.

6.1 Manual

Requirements

As ASPARTIX is a DLV implementation, it is required to have installed DLV. For further information about DLV and its usage we refer to the DLV-project page

http://www.dbai.tuwien.ac.at/research/project/dlv/

ASPARTIX consists of an "interpreter" program executed by DLV. The interpreter gets the argumentation framework by reading the input file.

6.1.1 Input File

In the input file, the user specifies the input database. It should contain the specification of the argumentation framework $AF = (A, R)$ and the desired semantics. Furthermore we recall that the ending of the input file should be `.dl` as this is common usage for DLV. For example `af1.dl` is a legal file name.

Representation of Arguments

For a given argumentation framework $AF = (A, R)$, the arguments $a \in A$ are defined by the facts arg/1. For example,

$$\text{arg}(a).$$

stands for the argument a. Due to the syntax of DLV, it is necessary that the name of the argument starts with a lower-case letter.

Representation of Attack Relations

For a given argumentation framework $AF = (A, R)$, the attack relation $R \subseteq A \times A$ is defined by the facts attack/2. For example,

$$\text{attack}(a, b).$$

denotes that the argument a attacks the argument b.

Figure 6.1 shows how the input file for Example 2.1.1 looks like. The fact prefex. denotes that the preferred extension should be computed.

```
prefex.

arg(a).
arg(b).
arg(c).
arg(d).
arg(e).

attack(a, b).
attack(c, b).
attack(c, d).
attack(d, c).
attack(d, e).
attack(e, e).
```

Figure 6.1: The file example2.1.1.dl for Example 2.1.1 and preferred extensions.

PAFs

In a given preference based argumentation framework $PAF = (A, R, \mathit{Pref})$, preferences on arguments are expressed via the preference relation $\mathit{Pref} \subseteq A \times A$. To specify this preordering on the arguments, the facts pref/2. are added to the input file. For example,

$$\mathrm{pref}(a, b).$$

denotes that the argument a is preferred over the argument b. Additionally it is required to add the fact paf. to the input to specify that we are dealing with a PAF.

Figure 6.2 shows how the input file for Example 2.3.1 looks like.

VAFs

For a given value based argumentation framework $VAF = (A, R, V, val, valpref)$, the following facts need to be added to the input file. To specify the values val for the arguments, the facts val/2. are used. For example,

$$\mathrm{val}(a, value\_a).$$

```
paf.
prefex.

arg(a).
arg(b).
arg(c).

attack(c, b).
attack(b, a).

pref(b, c).
```

Figure 6.2: The file example2.3.1.dl for Example 2.3.1 and preferred extensions.

denotes that the argument a has the value *value_a*. The preference relation is defined via the fact valpref/2. For example,

$$\text{valpref}(\mathit{value\_a}, \mathit{value\_b}).$$

denotes that the value *value_a* is preferred to the value *value_b*. Additionally it is required to add the fact vaf. to the input to specify, that we are dealing with a VAF.

Figure 6.3 shows how the input file for Example 2.3.3 looks like.

BAFs

For a given bipolar argumentation framework $BAF = (A, R_{def}, R_{sup})$, the following facts need to be added to the input file. The support relation $R_{sup} \subseteq A \times A$ is defined via the fact support/2. For example,

$$\text{support}(a, b).$$

stands for the argument a supporting the argument b. We recall that the attack relation R of the basic argumentation framework equals the defeat relation R_{def} of a BAF. Therefore R_{def} is defined via the fact attack/2. Additionally it is required to add the fact baf. to the input to specify that we are dealing with a BAF.

Figure 6.4 shows how the input file for Example 2.3.4 and s-admissible extensions looks like.

```
vaf.
prefex.

arg(a).arg(b).arg(c).
arg(d).arg(e).arg(f).

attack(a, e).attack(b, a).
attack(c, b).attack(c, f).
attack(d, c).attack(e, d).
attack(f, a).

val(a, life).
val(b, property).
val(c, property).
val(d, life).
val(e, property).
val(f, life).

valpref(life, property).
```

Figure 6.3: The file example2.3.3.dl for Example 2.3.3 and preferred extensions.

Extensions

To specify which extension should be computed, the respective fact should be added to the input file. If no extension is defined in the input file, no valid answer set can be computed and the program returns an `input_error`. It is only possible to compute one extension at a time. The possible extensions for the basic AFs are:

```
adm.        % for admissible
compl.      % for complete
ground.     % for ground
prefex.     % for preferred
stable.     % for stable
```

```
baf.
s_adm.
arg(a).
arg(b).
arg(c).
arg(d).
arg(e).

attack(a, e).
attack(d, c).

support(a, b).
support(b, c).
support(d, e).
```

Figure 6.4: The file example2.3.4.dl for Example 2.3.4 and s-admissible extensions.

Special semantics for BAFs are the following:

```
c_adm.        % for c-admissible (complete)
d_adm.        % for d-admissible (Dung)
s_adm.        % for s-admissible (stable)
c_prefex.     % for c-preferred
d_prefex.     % for d-preferred
s_prefex.     % for s-preferred
```

In the input files from Figures 6.1, 6.2 and 6.3, the preferred extensions have been selected, whereas in Figure 6.4, the s-admissible extension has been selected.

6.1.2 Execution

To execute the program correctly, it is just necessary to run DLV together with the input file and ASPARTIX on the command line. The output are answer sets specifying the required semantics. For better readability, it is useful to use the filter option of DLV. For example,

```
$ ./DLV input.dl aspartix.dl -filter=in,input_error
```

executes ASPARTIX with the input file input.dl. With `-filter=in,input_error` only the predicates in/1 and input_error are printed out.

Output

In the resulting answer sets, the fact in/1 denotes the arguments which are in the required extension. If the answer set contains the fact input_error, then the input has not been declared correctly.

The output for Example 2.1.1 as in Figure 6.1 with the input options

```
-filter=in,input_error
```

is shown in Figure 6.5. As expected, we have two preferred extensions, namely $\{a, c\}$ and $\{a, d\}$.

```
DLV [build BEN/Oct 11 2007 gcc 4.0.1 (Apple Computer, Inc.
build 5367)]

{in(a), in(c)}

{in(a), in(d)}
```

Figure 6.5: Output for Example 2.1.1 and preferred extensions.

The output for Example 2.3.4 as in Figure 6.4 is shown in Figure 6.6. We have five s-admissible extensions namely $\{\}$, $\{a\}$, $\{b\}$, $\{d\}$ and $\{a, b\}$.

```
DLV [build BEN/Oct 11 2007 gcc 4.0.1 (Apple Computer, Inc.
build 5367)]

{in(d)}

{in(a)}

{in(a), in(b)}

{in(b)}

{}
```

Figure 6.6: Output for Example 2.3.4 and s-admissible extensions.

Chapter 7

Related Work

A lot of research has been performed in the field of argumentation over the last ten years, resulting in a variety of approaches to compute acceptable semantics of an argumentation framework. In this chapter we mention the most relevant of them compared to our approach.

7.1 Model Checking

The approach to compute acceptable sets of arguments with model checking was introduced by Besnard and Doutre in [13].
For a given argumentation framework $AF = (A, R)$ and semantics $\varepsilon \in \Sigma = \{$*stable*, *adm*, *comp*, *ground*, *prefex*$\}$, a propositional formula is associated whose models correspond to the acceptable sets under the semantics. One disadvantage of this approach is that for the preferred semantic the maximal (with respect to set inclusion) models of the formula of the admissible extensions are required. Furthermore, to obtain the grounded extension, the minimal (with respect to set inclusion) model of the formula of the complete extensions is required. Unfortunately, computing the maximal or minimal models of a formula is usually not supported by current SAT-solvers.

This problem has been resolved by Egly and Woltran in [33] via the use of Quantified Boolean Formulas (QBFs).

7.1.1 Encodings Based on QBF

Egly and Woltran showed in [33] how to extend formulas by some conditions which express maximality respectively minimality of the models. They used QBFs to evaluate the preferred and grounded semantics.

7.2 Logic Programming

The idea of computing acceptable sets of arguments in terms of logic programming has already been mentioned by Dung [25].

7.2.1 ASP

The work which is closest related to ours is the approach from Nieves et al. [44] and Osorio et al. [45]. In [45] they gave an ASP encoding for admissible extensions which is very similar to ours. On the other hand, their method to compute the preferred extensions in [44] differs from ours substantially. They introduced a mapping function that constructs a disjunctive logic program P, such that the preferred extensions of an argumentation framework correspond to the stable models of P. This means that they need a translation for every new instance, whereas we designed a fixed program which is independent of the concrete argumentation framework. From the complexity point of view, their approach is on a par with our program because, the reductions are polynomial-time computable. From the application point of view, our system has some useful features. It is easier to extend and debug. The user can easily add or delete arguments and relations and thereby compare different argumentation frameworks.

7.3 Implementations

To the best of our knowledge, so far no system is available which supports such a broad range of different semantics, although nowadays a number of implementations exists[3].

- Dungine [49] is a Java reasoner capable of reasoning with grounded and credulous preferred extensions.
- An epistemic and practical reasoner is available which also supports preferred credulous semantics for practical arguments [50] (see http://www.wietskevisser.nl/research/epr/).
- The program PARMENIDES is a system for e-democracy based on value-based AFs [16, 4].
- CASAPI is a Prolog implementation that combines abstract and assumption-based argumentation [37] (see http://www.doc.ic.ac.uk/~dg00/casapi.html).

[3]See also http://www.csc.liv.ac.uk/~azwyner/software.html for an overview.

Chapter 8

Conclusion

In this work, we dealt with different types of argumentation frameworks and related reasoning problems.
We analysed the respective definitions and properties and brought them into a uniform notation. The system ASPARTIX has been implemented on top of DLV, capable to compute different types of extensions in Dung's argumentation framework as well as in some extensions of it. To the best of our knowledge, so far no system is available which supports such a broad range of different semantics for all those frameworks. Beside the encodings and their proof of correctness, we also discussed argumentation frameworks in terms of complexity. Thereby we showed that our system is also adequate from the complexity point of view.

The work which is closest related to ours is by Nieves *et al.* [44] who also suggest to use answer-set programming for computing extensions of argumentation frameworks. The most important difference is that in their work the program has to be re-computed for each new instance, while our system relies on a *single fixed* program which just requires the actual instance as an input database. We believe that our approach thus is more reliable and easier extendible to further formalisms.

Future work includes a comparison of the efficiency of different implementations and an extension of our system by incorporating further recent notions of semantics, for instance, the semi-stable semantics [15] or the ideal semantics [26].

Bibliography

[1] Leila Amgoud and Claudette Cayrol. A reasoning model based on the production of acceptable arguments. *Ann. Math. Artif. Intell.*, 34(1-3):197–215, 2002.

[2] Leila Amgoud, Claudette Cayrol, Marie-Christine Lagasquie, and Pierre Livet. On bipolarity in argumentation frameworks. *International Journal of Intelligent Systems*, 23:1–32, 2008.

[3] Leila Amgoud, Claudette Cayrol, and Marie-Christine Lagasquie-Schiex. On the bipolarity in argumentation frameworks. In *10th International Workshop on Non-Monotonic Reasoning (NMR 2004), Whistler, Canada, June 6-8, 2004, Proceedings*, pages 1–9, 2004.

[4] Katie Atkinson, Trevor J. M. Bench-Capon, and Peter McBurney. Parmenides: Facilitating democratic debate. In *Electronic Government: Third International Conference, EGOV 2004, Zaragoza, Spain, August 30 - September 3, 2004, Proceedings*, pages 313–316, 2004.

[5] Pietro Baroni and Massimiliano Giacomin. A systematic classification of argumentation frameworks where semantics agree. In *Proceedings of the 2nd Conference on Computational Models of Argument* (*COMMA'08*), pages 37–48. IOS Press, 2008.

[6] Rachel Ben-Eliyahu and Rina Dechter. Propositional semantics for disjunctive logic programs. *Ann. Math. Artif. Intell.*, 12(1-2):53–87, 1994.

[7] Trevor J. M. Bench-Capon. Value-based argumentation frameworks. In *9th International Workshop on Non-Monotonic Reasoning (NMR 2002), April 19-21, Toulouse, France, Proceedings*, pages 443–454, 2002.

[8] Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in AI and law: Editors' introduction. *Artif. Intell. Law*, 13(1):1–8, 2005.

[9] Trevor J.M. Bench-Capon. Representation of case law as an argumentation framework. *Bench-Capon, T., Daskalopoulu, A. and Winkels, R. (eds.), Proceedings of the Fifteenth Annual Conference on Legal Knowledge and Information Systems (2002)*, pages 103–112, 2002.

[10] Trevor J.M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *J. Log. Comput.*, 13(3):429–448, 2003.

[11] Trevor J.M. Bench-Capon, Katie Atkinson, and Alison Chorley. Persuasion and value in legal argument. *J. Log. Comput.*, 15(6):1075–1097, 2005.

[12] Trevor J.M. Bench-Capon and Paul E. Dunne. Argumentation in artificial intelligence. *Artif. Intell.*, 171(10-15):619–641, 2007.

[13] Philippe Besnard and Sylvie Doutre. Checking the acceptability of a set of arguments. In *10th International Workshop on Non-Monotonic Reasoning (NMR 2004), Whistler, Canada, June 6-8, 2004, Proceedings*, pages 59–64, 2004.

[14] Philippe Besnard and Anthony Hunter. Elements of argumentation. Cambridge, MA, MIT Press, 2008.

[15] Martin Caminada. Semi-stable semantics. In *Proceedings of the 1st Conference on Computational Models of Argument* (*COMMA'06*), pages 121–130. IOS Press, 2006.

[16] Dan Cartwright and Katie Atkinson. Political engagement through tools for argumentation. In *Proceedings of the 2nd Conference on Computational Models of Argument* (*COMMA'08*), pages 116–127. IOS Press, 2008.

[17] Claudette Cayrol, Caroline Devred, and Marie-Christine Lagasquie-Schiex. Handling controversial arguments in bipolar argumentation systems. In *Proceedings of the 1st Conference on Computational Models of Argument (COMMA'06)*, pages 261–272, Liverpool, United Kingdom, 2006. IOS Press.

[18] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. Gradual valuation for bipolar argumentation frameworks. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 8th European Conference, ECSQARU 2005, Barcelona, Spain, July 6-8, 2005, Proceedings*, volume 3571 of *LNCS*, pages 366–377. Springer, 2005.

[19] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. On the acceptability of arguments in bipolar argumentation frameworks. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 8th European Conference, ECSQARU 2005, Barcelona, Spain, July 6-8, 2005, Proceedings*, pages 378–389, 2005.

[20] Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Prudent semantics for argumentation frameworks. In *Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence(ICTAI'05)*, pages 368–372, Hong-Kong, 2005.

[21] Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Symmetric argumentation frameworks. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 8th European Conference, ECSQARU 2005, Barcelona, Spain, July 6-8, 2005, Proceedings*, volume 3571 of *LNCS*, pages 317–328. Springer, 2005.

[22] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[23] Yannis Dimopoulos, Pavlos Moraitis, and Leila Amgoud. Theoretical and computational properties of preference-based argumentation. In *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*, pages 463–467, 2008.

[24] Yannis Dimopoulos and Alberto Torres. Graph theoretical structures in logic programs and default theories. *Theor. Comput. Sci.*, 170(1-2):209–244, 1996.

[25] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.

[26] Phan Minh Dung, Paolo Mancarella, and Francesca Toni. Computing ideal sceptical argumentation. *Artif. Intell.*, 171(10-15):642–674, 2007.

[27] Paul E. Dunne. Complexity and combinatorial properties of argument systems. Technical report, 2001.

[28] Paul E. Dunne and Trevor Bench-capon. Complexity in value-based argument systems. In *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004, Proceedings*, volume 3229 of *Lecture Notes in Computer Science*, pages 360–371. Springer-Verlag, 2004.

[29] Paul E. Dunne and Trevor J.M. Bench-Capon. Coherence in finite argument systems. *Artif. Intell.*, 141(1/2):187–203, 2002.

[30] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. Answer-set programming encodings for argumentation frameworks. In *1st Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2008), Udine, Italy*, 2008.

[31] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. Answer-set programming encodings for argumentation frameworks. Technical Report DBAI-TR-2008-62, Technische Universität Wien, 2008.

[32] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. ASPARTIX: Implementing argumentation frameworks using answer-set programming. In *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, volume 5366 of *Lecture Notes in Computer Science*, pages 734–738. Springer, 2008.

[33] Uwe Egly and Stefan Woltran. Reasoning in argumentation frameworks using quantified boolean formulas. In *Proceedings of the 1st Conference on Computational Models of Argument (COMMA'06)*, pages 133–144. IOS Press, 2006.

[34] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Declarative problem-solving using the dlv system. In *Logic-Based Artificial Intelligence*, pages 79–103, Norwell, MA, USA, 2000. Kluwer Academic Publishers.

[35] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.*, 15(3-4):289–323, 1995.

[36] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.

[37] Dorian Gaertner and Francesca Toni. Hybrid argumentation and its properties. In *Proceedings of the 2nd Conference on Computational Models of Argument* (*COMMA'08*), pages 183–195. IOS Press, 2008.

[38] M. Gebser, L. Liu, G. Namasivayam, A. Neumann, T. Schaub, and M. Truszczyński. The first answer set programming system competition. In *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'07*), volume 4483 of *LNCS*, pages 3–17. Springer, 2007.

[39] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP'88), Seattle, Washington, August 15-19, 1988*, pages 1070–1080, 1988.

[40] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[41] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7(3):499–562, 2006.

[42] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In *Proceedings of the 8th International Conference on Logic Programming (ICLP'94)*, pages 23–37, Cambridge, MA, USA, 1994. MIT Press.

[43] Ilkka Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3–4):241–273, 1999.

[44] Juan Carlos Nieves, Ulises Cortés, and Mauricio Osorio. Preferred extensions as stable models. *Theory and Practice of Logic Programming (TPLP)*, 8(4):527–543, 2008.

[45] Mauricio Osorio, Claudia Zepeda, Juan Carlos Nieves, and Ulises Cortés. Inferring acceptable arguments with answer set programming. In *Proceedings of the 6th Mexican International Conference on Computer Science (ENC 2005), 26-30 September 2005, Puebla, Mexico*, pages 198–205, 2005.

[46] Christos M. Papadimitriou. *Computational complexity.* Addison-Wesley, Reading, Massachusetts, 1994.

[47] Jörg Pührer. On debugging of propositional answer-set programming. Master's thesis, Technical University Vienna, 2008.

[48] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[49] Matthew South, Gerard Vreeswijk, and John Fox. Dungine: A Java Dung Reasoner. In *Proceedings of the 2nd Conference on Computational Models of Argument (COMMA'08)*, pages 360–368. IOS Press, 2008.

[50] Wietske Visser. Implementation of argument-based practical reasoning. Master's thesis, Utrecht University, The Netherlands, 2008.

[51] Adam Wyner, Trevor J.M. Bench-Capon, and Katie Atkinson. Arguments, values and baseballs: Representation of Popov v. Hayashi. *JURIX 2007: The Twentieth Annual Conference on Legal Knowledge and Information Systems*, pages 151–160, 2007.

Appendix A

ASPARTIX

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% for some input-facts new facts are derived, for example
% for every preferred extension, from any argumentation
% framework (AF) except BAF's also the admissible
% extensions need to be computed.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

adm :- d_adm, baf, not input_error.            % d-admissible extensions for
                                               % BAF's are the same as the
                                               % standard admissible extensions
adm :- comp.                                   % every complete ext. is also
                                               % an admissible ext.
adm :- prefex, not baf, not input_error.
comp :- ground, not input_error.               % every grounded ext. is also
                                               % a complete ext.
prefex :- d_prefex, baf, not input_error.      % d-preferred ext. for BAF's
                                               % are the same as
                                               % standard pref. ext.
d_adm :- d_prefex, baf, not input_error.
closed :- c_adm, baf, not input_error.         % c-adm. ext. for BAF's need
                                               % to be closed
safe :- s_adm, not input_error.                % s-adm. ext. for BAF's need
                                               % to be safe
s_adm :- s_prefex, baf, not input_error.
c_adm :- c_prefex, baf, not input_error.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% check the possible extensions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

extension :- adm.
extension :- c_adm, baf.
extension :- d_adm, baf.
extension :- s_adm, baf.
extension :- closed, baf.
extension :- comp.
extension :- ground.
extension :- safe, baf.
extension :- stable.
extension :- prefex.
extension :- c_prefex, baf.
extension :- d_prefex, baf.
extension :- s_prefex, baf.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% error-handling for wrong input combinations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% support relation is only allowed for BAFs
input_error :- support(X,Y), not baf.

%% if no extension is specified in the input file, we do not
%% compute anything.
input_error :- not extension.

%% valpref preference relation is only specified for VAFs
input_error :- valpref(X,Y), not vaf.

%% preordering of arguments is only specified for VAFs and PAFs
vafpref :- pref(X,Y), not paf, vaf.
input_error :- not vafpref, pref(X,Y), not paf.

%% complete and grounded extensions are not supported for BAFs
input_error :- baf, comp.
input_error :- baf, ground.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% preferences for PAF and VAF
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% valpref preference relation; transitiv
valpref(X,Z) :- valpref(X,Y), valpref(Y,Z).

%% pref computes preference of arguments depending on the
%% preference relation valpref
pref(X,Y) :- valpref(U,V),
             val(X,U),
             val(Y,V).

%% transitivity of pref
pref(X,Z) :- pref(X,Y), pref(Y,Z).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       support and defeat for BAF
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% argument x is supported by argument y
support(X,Z) :- support(X,Y), support(Y,Z).

%% set-supports: argument x is supported by the set S
supported(X) :- in(Y), support(Y,X).

%% defeats (BAF)
defeat(X,Y) :- attack(Z,Y), support(X,Z), baf. %supported defeat
defeat(X,Y) :- attack(X,Y), baf.              %supported defeat
defeat(X,Y) :- attack(X,Z), support(Z,Y), baf. %indirekt defeat
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% argument a defeats argument b for classical AF as well as
%% for PAF and VAF
```

```
defeat(A,B) :- attack(A,B),
               not pref(B,A).

%% Guess S \subseteq A (only if no input_error has been derived).
in(X) v out(X) :- arg(X), not ground, not input_error.

%% S has to be conflictfree for all extensions
:- in(X), in(Y), defeat(X,Y).

%% S defeats X
defeated(X):- in(Y), defeat(Y,X).

%% X \in S is not defended by S
not_defended(X):- defeat(Y,X), not defeated(Y).

%% admissible
:- in(X), not_defended(X), adm.

%% complete
:- out(X), not not_defended(X), comp.

%% stable
:- out(X), not defeated(X), stable.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% speciall semantics for BAF
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% safe
:- supported(B), defeated(B), safe.

%% s-admissible
:- in(X), not_defended(X), s_adm.

%% closed
:- support(X,Y), out(Y),in(X), closed.
:- support(X,Y), in(Y), out(X), closed.

%% c_admissible
:- in(X), not_defended(X), c_adm.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% For the remaining part we need to put an order on the domain.
% Therefore, we define a successor-relation with infinum and supremum
% as follows
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

lt(X,Y) :- arg(X),arg(Y), X<Y, not input_error.
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).
sup(X) :- not nsup(X), arg(X).

%% grounded
%% the grounded extension is the minimal complete extension

defended_upto(X,Y) :- inf(Y), arg(X), not defeat(Y,X), ground.
defended_upto(X,Y) :- inf(Y), in(Z), defeat(Z,Y), defeat(Y,X),
                      ground.
```

```
defended_upto(X,Y) :- succ(Z,Y), defended_upto(X,Z),
                      not defeat(Y,X), ground.
defended_upto(X,Y) :- succ(Z,Y), defended_upto(X,Z), in(V),
                      defeat(V,Y), defeat(Y,X), ground.

defended(X) :- sup(Y), defended_upto(X,Y).
in(X) :- defended(X), ground, not input_error.

%% Guess S' \supseteq S for classical pref. ext.

inN(X) v outN(X) :- out(X), prefex, not input_error.
inN(X) :- in(X), prefex, not input_error.

%% Guess S' \supseteq S for s-preferred
inN(X) v outN(X) :- out(X), s_prefex, not input_error.
inN(X) :- in(X), s_prefex.

%% Guess S' \supseteq S for c-preferred
inN(X) v outN(X) :- out(X), c_prefex, not input_error.
inN(X) :- in(X), c_prefex.

%% If S' = S then spoil.
%% Use the sucessor function and check starting from supremum whether
%% elements in S' is also in S. If this is not the case we "stop"
%% If we reach the supremum we spoil up.

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 :- sup(Y), eq_upto(Y).

%% get those X \notin S' which are not defeated by S'
%% using successor again...

undefeated_upto(X,Y) :- inf(Y), outN(X), outN(Y).%, prefex.
undefeated_upto(X,Y) :- inf(Y), outN(X),  not defeat(Y,X).%, prefex.

undefeated_upto(X,Y) :- succ(Z,Y), undefeated_upto(X,Z), outN(Y).
undefeated_upto(X,Y) :- succ(Z,Y), undefeated_upto(X,Z), not defeat(Y,X).

undefeated(X) :- sup(Y), undefeated_upto(X,Y).

spoil :- eq.

%% S' has to be conflictfree - otherwise spoil
spoil :- inN(X), inN(Y), defeat(X,Y), c_prefex.
spoil :- inN(X), inN(Y), defeat(X,Y), prefex.

%% set-supports
supportedN(X) :- inN(Y), support(Y,X).

%% S' has to be safe for s-preferred
spoil :- supportedN(B), defeat(X,B), inN(X), s_prefex.
spoil :- defeat(X,B), inN(X), inN(B), s_prefex.

%% S' has to be closed for c-preferred
spoil :- support(X,Y), outN(Y), inN(X), c_prefex.
spoil :- support(X,Y), inN(Y), outN(X), c_prefex.
```

```
%% S' has to be admissible - otherwise spoil
spoil :- inN(X), outN(Y), defeat(Y,X), undefeated(Y).

inN(X) :- spoil, arg(X), not input_error.
outN(X) :- spoil, arg(X), not input_error.

%% do the final spoil-thing ...
:- not spoil, prefex, not input_error.
:- not spoil, s_prefex, not input_error.
:- not spoil, c_prefex, not input_error.
```

Index