DBAI

TECHNICAL REPORT

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG DATENBANKEN UND ARTIFICIAL INTELLIGENCE

Answer-Set Programming Encodings for Argumentation Frameworks

DBAI-TR-2008-62

Uwe Egly **Sarah Alice Gaggl** **Stefan Woltran**

Institut für Informationssysteme
Abteilung Datenbanken und
Artificial Intelligence
Technische Universität Wien
Favoritenstr. 9
A-1040 Vienna, Austria
Tel: +43-1-58801-18403
Fax: +43-1-58801-18492
sekret@dbai.tuwien.ac.at
www.dbai.tuwien.ac.at

DBAI TECHNICAL REPORT
2008

TU
TECHNISCHE UNIVERSITÄT WIEN

DBAI TECHNICAL REPORT
DBAI TECHNICAL REPORT DBAI-TR-2008-62, 2008

Answer-Set Programming Encodings for Argumentation Frameworks

Uwe Egly[1] **Sarah Alice Gaggl**[2] **Stefan Woltran**[3]

Abstract. We present reductions from Dung's argumentation framework (AF) and generalizations thereof to logic programs under the answer-set semantics. The reduction is based on a fixed disjunctive datalog program (the interpreter) and its input which is the only part depending on the AF to process. We discuss the reductions, which are the basis for the system ASPARTIX in detail and show their adequacy in terms of computational complexity.

[1]Institute for Information Systems 184/3, Technische Universität Wien, Favoritenstrasse 9-11, 1040 Vienna, Austria. E-mail: uwe@kr.tuwien.ac.at

[2]Technische Universität Wien, E-mail: e0026566@student.tuwien.ac.at

[3]Institute for Information Systems 184/2, Technische Universität Wien, Favoritenstrasse 9-11, 1040 Vienna, Austria. E-mail: woltran@dbai.tuwien.ac.at

Acknowledgements: The authors would like to thank Wolfgang Faber for comments on an earlier draft of this paper. This work was partially supported by the Austrian Science Fund (FWF) under grant P20704-N18.

This is an extended version of a paper published in the Proceedings of the ICLP'08 Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'08).

1 Motivation

Dealing with arguments and counter-arguments in discussions is a daily life process. We usually employ this process to convince our opponent to our point of view. As everybody knows, this is sometimes a cumbersome activity because we miss a formal reasoning procedure for argumentation.

This problem is not new. Leibniz (1646–1716) was the first who tried to deal with arguments and their processing by reasoning in a more formal way. He proposed to use a *lingua characteristica* (a knowledge representation (KR) language) to formalize arguments and a *calculus ratiocinator* (a deduction system) to reason about them. 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.

In Artificial Intelligence (AI), the area of argumentation (see [6] for an excellent summary) has become one of the central issues within the last decade, providing a formal treatment for reasoning problems arising in a number of interesting applications fields, including Multi-Agent Systems and Law Research. In a nutshell, argumentation frameworks formalize statements together with a relation denoting rebuttals between them, such that the semantics gives an abstract handle to solve the inherent conflicts between statements by selecting admissible subsets of them. The reasoning underlying such argumentation frameworks turned out to be a very general principle capturing many other important formalisms from the areas of AI and Knowledge Representations.

The increasing interest in argumentation led to numerous proposals for formalizations of argumentation. These approaches differ in many aspects. First, there are several ways how "admissibility" of a subset of statements can be defined; second, 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.

Argumentation problems are in general intractable, thus developing dedicated algorithms for the different reasoning problems is non-trivial. A promising approach to implement such systems is to use a reduction method, where the given problem is translated into another language, for which sophisticated systems already exist. Earlier work [7, 17] proposed reductions for basic argumentation frameworks to (quantified) propositional logic. In 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, the system is capable to compute the many important types of extensions (i.e., admissible, preferred, stable, semi-stable, complete, and grounded) in Dung's original framework [13], the preference-based argumentation framework [1], the value-based argumentation framework [5], and the bipolar argumentation framework [2, 9]. Hence our system can be used by researchers to compare different argumentation semantics on concrete examples within a uniform setting. In fact, investigating the relationship between different argumentation semantics has received increasing interest lately [3].

The declarative programming paradigm of *Answer-Set Programming* (ASP) [22, 24] under the stable-models semantics [21] (which is our target formalism) is especially well suited for our purpose. First, advanced solvers such as Smodels, DLV, GnT, Cmodels, Clasp, or ASSAT which

are able to deal with large problem instances (see [20]) are available. Thus, using the proposed reduction method 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 program 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.

With the fixed logic program (independent from the concrete AF to process), we are more in the tradition of a classical implementation, because we construct an interpreter in ASP which processes the AF given as input. This is in contrast to, e.g., the reductions to (quantified) propositional logic [7, 17], where one obtains a formula which completely depends on the AF to process. Although there is no advantage of the interpreter approach from a theoretical point of view (as long as the reductions are polynomial-time computable), there are several practical ones. The interpreter is easier to understand, easier to debug, and easier to extend. Additionally, proving properties like correspondence between answer sets and extensions is simpler. Moreover, the input AF can be changed easily and dynamically without translating the whole formula 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 [22]. All necessary programs to run ASPARTIX and some illustrating examples are available at

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

2 Preliminaries

In this section, we first give a brief overview of the syntax and semantics of disjunctive datalog under the answer-sets semantics [21]; for further background, see [18, 22].

We fix a countable set $\mathcal{U}$ of *(domain) elements*, also called *constants*; and suppose a total order $<$ over the domain elements. An *atom* is an expression $p(t_1, \ldots, t_n)$, where p is a *predicate* of arity $n \geq 0$ and each t_i is either a variable or an element from $\mathcal{U}$. An atom is *ground* if it is free of variables. By $B_{\mathcal{U}}$ we denote the set of all ground atoms over $\mathcal{U}$.

A *(disjunctive) rule* r is of the form

$$a_1 \vee \cdots \vee a_n \text{ :- } b_1, \ldots, b_k, \ not\, b_{k+1}, \ldots, \ not\, b_m,$$

with $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$, and where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms, and "not" stands for *default negation*. The *head* of r is the set $H(r) = \{a_1, \ldots, a_n\}$ and the *body* of r is $B(r) = \{b_1, \ldots, b_k, \ not\, b_{k+1}, \ldots, \ not\, b_m\}$. Furthermore, $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$. A rule r is *normal* if $n \leq 1$ and a *constraint* if $n = 0$. A rule r is *safe* if each variable in r occurs in $B^+(r)$. A rule r is *ground* if no variable occurs in r. A *fact* is a disjunctive-free ground rule with an empty body. An *input (database)* is a finite set of facts. A program is a finite set of disjunctive rules. For a program $\mathcal{P}$ and an input database D, we often write $\mathcal{P}(D)$ instead of $D \cup \mathcal{P}$. If each rule in a program is normal (resp. ground), we call the program normal

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

Table 1: Data Complexity for datalog (all results are completeness results).

(resp. ground). A program $\mathcal{P}$ is called *stratified* if there exists an assignment $a(\cdot)$ of integers to the predicates in $\mathcal{P}$, such that for each rule $r \in \mathcal{P}$ the following holds: If predicate p occurs in the head of r and predicate q occurs

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

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

For any program $\mathcal{P}$, let $U_\mathcal{P}$ be the set of all constants appearing in $\mathcal{P}$ (if no constant appears in $\mathcal{P}$, an arbitrary constant is added to $U_\mathcal{P}$), and let $B_\mathcal{P}$ be the set of all ground atoms constructible from the predicate symbols appearing in $\mathcal{P}$ and the constants of $U_\mathcal{P}$. Moreover, $Gr(\mathcal{P})$ is the set of rules $r\sigma$ obtained by applying, to each rule $r \in \mathcal{P}$, all possible substitutions σ from the variables in $\mathcal{P}$ to elements of $U_\mathcal{P}$.

An *interpretation* $I \subseteq B_\mathcal{U}$ *satisfies* a ground rule r iff $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. I satisfies a ground program $\mathcal{P}$, if each $r \in \mathcal{P}$ is satisfied by I. A non-ground rule r (resp., a program $\mathcal{P}$) is satisfied by an interpretation $I \subseteq B_\mathcal{U}$ iff I satisfies all groundings of r (resp., $Gr(\mathcal{P})$). $I \subseteq B_\mathcal{U}$ is an *answer set* of $\mathcal{P}$ iff it is a subset-minimal set satisfying the *Gelfond-Lifschitz reduct*

$$\mathcal{P}^I = \{H(r) \text{ :- } B^+(r) \mid I \cap B^-(r) = \emptyset, r \in Gr(\mathcal{P})\}.$$

For a program $\mathcal{P}$, we denote the set of its answer sets by $\mathcal{AS}(\mathcal{P})$. We note that for each $I \in \mathcal{AS}(\mathcal{P})$, $I \subseteq B_\mathcal{P}$ holds.

Credulous and skeptical reasoning in terms of programs is defined as follows. Given a program $\mathcal{P}$ and a set of ground atoms A. Then, we write $\mathcal{P} \models_c A$ (credulous reasoning), if A is contained in some answer set of $\mathcal{P}$; we write $\mathcal{P} \models_s A$ (skeptical reasoning), if A is contained in each answer set of $\mathcal{P}$.

We briefly recall some complexity results for disjunctive logic programs. In fact, since we will deal with fixed programs, we focus on results for data complexity. Recall that 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 1 (cf. [11] and the references therein).

Finally, we recall the concepts of splitting sets [23]. Given a program P, a set U of predicates is a *splitting set* for P, if and only if, for every rule $r \in P$, it holds if some predicate from U occurs in the head of r, then each predicate in r is from U as well. Any splitting set U for program P

divides P in two parts. The *top* P^t_U of P contains all rules of P which have an occurrence of a predicate *not* contained in U, while the *bottom* P^b_U of P is defined as $P \setminus P^t_U$. Splitting sets allow to compute the answer sets of a program P step-by-step due to the following result (the *splitting theorem*): Let U be a splitting set of a program P, $I \subseteq B_{\mathcal{U}}$. Then, $I \in \mathcal{AS}(P)$ if and only if $I \in \mathcal{AS}(P^t_U(J))$, where $J = I \cap B_{P^b_U}$ is contained in $\mathcal{AS}(P^b_U)$.

3 Encodings of Basic Argumentation Frameworks

In this section, we first introduce the most important semantics for basic argumentation frameworks in some detail. In a distinguished section, we then provide encodings for these semantics in terms of datalog programs.

3.1 Basic Argumentation Frameworks

In order to relate frameworks to programs, we use the universe $\mathcal{U}$ of domain elements also in the following basic definition.

Definition 3.1 *An* argumentation framework (AF) *is a pair $F = (A, R)$ where $A \subseteq \mathcal{U}$ is a set of arguments and $R \subseteq A \times A$. The pair $(a, b) \in R$ means that a attacks (or defeats) b. A set $S \subseteq A$ of arguments* defeats *b (in F), if there is an $a \in S$, such that $(a, b) \in R$. An argument $a \in A$ is* defended *by $S \subseteq A$ (in F) iff, for each $b \in A$, it holds that, if $(b, a) \in R$, then S defeats b (in F).*

An argumentation framework can be naturally represented as a directed graph.

Example 3.2 *Let $F = (A, R)$ be an AF with $A = \{a, b, c, d, e\}$ and $R = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\}$. The graph representation of F is the following.*

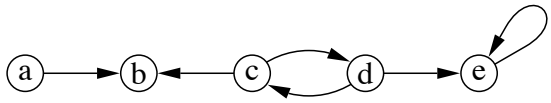

Figure 1: Graph of Example 3.2.

In order to be able to reason about such frameworks, it is necessary to group arguments with special properties to *extensions*. One of the basic properties is the absence of conflicts between arguments contained in an extension.

Definition 3.3 *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is said to be* conflict-free (in F)*, if there are no $a, b \in S$, such that $(a, b) \in R$. We denote the collection of sets which are conflict-free (in F) by $cf(F)$.*

For our example framework $F = (A, R)$ from Example 3.2, we have

$$cf(F) = \big\{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}, \{b, d\}\big\}.$$

As a first concept of extension, we present the *stable extensions* which are based on the idea that an extension should not only be internally consistent but also able to reject the arguments that are outside the extension.

Definition 3.4 *Let $F = (A, R)$ be an AF. A set S is a* stable extension *of F, if $S \in cf(F)$ and each $a \in A \setminus S$ is defeated by S in F. We denote the collection all of stable extensions of F by $stable(F)$.*

The framework F from Example 3.2 has a single stable extension $\{a, d\}$. Indeed $\{a, d\}$ is conflict-free, and each further element b, c, e is defeated by either a or d. In turn, $\{a, c\}$ for instance is not contained in $stable(F)$, although it is conflict-free as well. The obvious reason is that e is not defeated by $\{a, c\}$.

Stable semantics in terms of argumentation are considered to be quite restricted. Moreover, it is not guaranteed that a framework possesses at least one stable extension (consider, e.g., the simple cyclic framework $(\{a\}, \{(a, a)\})$). Therefore it is also reasonable to consider those arguments which are able to defend themselves from external attacks, like the *admissible semantics* proposed by Dung [13].

Definition 3.5 *Let $F = (A, R)$ be an AF. A set S is an* admissible extension *of F, if $S \in cf(F)$ and each $a \in S$ is defended by S in F. We denote the collection of all admissible extensions of F by $adm(F)$.*

For the framework F from Example 3.2, we obtain,

$$adm(F) = \big\{\emptyset, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}\big\}.$$

By definition, the empty set is always an admissible extension, therefore reasoning over admissible extensions is also limited. In fact, some reasoning (for instance, given an AF $F = (A, R)$, and $a \in A$, is a contained in any extension of F) becomes trivial wrt admissible extensions. Thus, many researchers consider maximal (wrt set-inclusion) admissible sets, called *preferred extensions*, as more important.

Definition 3.6 *Let $F = (A, R)$ be an AF. A set S is a* preferred extension *of F, if $S \in adm(F)$ and for each $T \in adm(F)$, $S \not\subset T$. We denote the collection of all preferred extensions of F by $pref(F)$.*

Obviously, the preferred extensions of framework F from Example 3.2 are $\{a, c\}$ and $\{a, d\}$. We note that each stable extension is also preferred, but the converse does not hold, as witnessed by this example.

The next semantics we consider is the *semi-stable semantics*, recently introduced by Caminada [8] and investigated also in [16]. Semi-stable semantics are located in-between stable and

preferred semantics, in the sense that each stable extension of an argumentation framework F is also a semi-stable extension of F, and each semi-stable extension of F is a preferred extension of F. However, in general both inclusions do not hold in the opposite direction. In contrast to the stable semantics, semi-stability guarantees that there exists at least one extension. We use the definition given in [16].

Definition 3.7 *Let $F = (A, R)$ be an AF, and for a set $S \subseteq A$, let S_R^+ be defined as $S \cup \{b \mid \exists a \in S$, such that $(a, b) \in R\}$. A set S is a* semi-stable extension *of F, if $S \in adm(F)$ and for each $T \in adm(F)$, $S_R^+ \not\subset T_R^+$, We denote the collection of all semi-stable extensions of F by $semi(F)$.*

For our example framework (A, R), the only semi-stable extension coincides with the stable extension $T = \{a, d\}$. In contrast, $S = \{a, c\}$ is not semi-stable, because $S_R^+ = \{a, b, c, d\} \subset \{a, b, c, d, e\} = T_R^+$.

Finally, we introduce complete and grounded extensions which Dung considered as skeptical counterparts of admissible and preferred extensions, respectively.

Definition 3.8 *Let $F = (A, R)$ be an AF. A set S is a* complete extension *of F, if $S \in adm(F)$ and, for each $a \in A$ defended by S (in F), $a \in S$ holds. The least (wrt set inclusion) complete extension of F is called the* grounded extension *of F. We denote the collection of all complete (resp., grounded) extensions of F by $comp(F)$ (resp., $ground(F)$).*

The complete extensions of framework F from Example 3.2 are $\{a, c\}$, $\{a, d\}$, and $\{a\}$, with the last being also the grounded extensions of F.

This concludes our collection of argumentation semantics, we consider in this paper. The relations between the semantics are depicted in Figure 2, where an arrow from e to f indicates that each e-extension is also a f-extension.

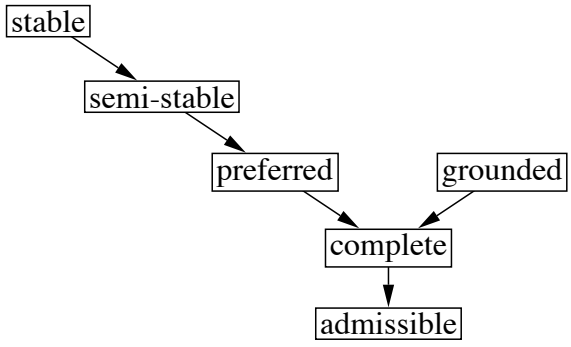

Figure 2: Overview of argumentation semantics and their relations.

We briefly review the complexity of reasoning in AFs. To this end, we define the following decision problems for $e \in \{stable, adm, pref, semi, comp, ground\}$:

	stable	*adm*	*pref*	*semi*	*comp*	*ground*
Cred_e	NP-c	NP-c	NP-c	in Σ_2^P	NP-c	in P
Skept_e	coNP-c	(trivial)	Π_2^P-c	in Π_2^P	in P	in P

Table 2: Complexity for decision problems in argumentation frameworks.

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

The complexity results are depicted in Table 2 (many of them follow implicitly from [12], for the remaining results and discussions see [10, 15, 16]). In the table, "$\mathcal{C}$-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. A few further comments are in order. We already mentioned that skeptical reasoning over admissible extensions always is 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 simple makes clear why there is no increase in complexity compared to credulous reasoning over admissible extensions: a is contained in some $S \in adm(F)$ iff a is contained in some $S \in pref(F)$. A similar argument immediately shows why skeptical reasoning over complete extensions reduces to skeptical reasoning over the grounded extension. Finally, we recall that reasoning over the grounded extension is tractable [13]:

Proposition 3.9 *The grounded extension of an AF $F = (A, R)$ is given by the least fix-point of the operator $\Gamma_F : 2^A \to 2^A$, defined as $\Gamma_F(S) = \{a \in A \mid a \text{ is defended by } S \text{ in } F\}$.*

3.2 Encodings

We now provide a fixed encoding π_e for each extension of type e introduced so far, in such a way that the AF F is given as an input database $\widehat{F}$ and the answer sets of the combined program $\pi_e(\widehat{F})$ are in a certain one-to-one correspondence with the respective extensions (with some additions, we can of course combine the different encodings into a single program, where the user just has to specify which type of extensions she wants to compute). Note that having established the fixed program π_e, the only translation required is to provide a given AF F as input database $\widehat{F}$ to π_e. In fact, for an AF F, we define $\widehat{F}$ as

$$\widehat{F} = \{\mathrm{arg}(a) \mid a \in A\} \cup \{\mathrm{defeat}(a, b) \mid (a, b) \in R\}.$$

In most cases, we have to guess candidates for the selected type of extensions and then check whether a guessed candidate satisfies the corresponding conditions. We use unary predicates $\mathrm{in}(\cdot)$ and $\mathrm{out}(\cdot)$ to perform such a guess for a set $S \subseteq A$, where $\mathrm{in}(a)$ represents that $a \in S$. Thus the following notion of correspondence is relevant for our purposes.

Definition 3.10 *Let $\mathcal{S} \subseteq 2^{\mathcal{U}}$ be a collection of sets of domain elements and let $\mathcal{I} \subseteq 2^{B_{\mathcal{U}}}$ be a collection of sets of ground atoms. We say that $\mathcal{S}$ and $\mathcal{I}$ correspond to each other, in symbols $\mathcal{S} \cong \mathcal{I}$, iff $|\mathcal{S}| = |\mathcal{I}|$ and for each $S \in \mathcal{S}$, there exists an $I \in \mathcal{I}$, such that $\{a \mid \mathrm{in}(a) \in I\} = S$.*

Let $F = (A, R)$ an argumentation framework. The following program fragment guesses, when augmented by $\widehat{F}$, any subset $S \subseteq A$ and then checks whether the guess is conflict-free in F:

$$\begin{aligned}\pi_{cf} \quad = \{ \quad & \mathrm{in}(X) \text{ :- } \mathit{not}\, \mathrm{out}(X), \mathrm{arg}(X); \\ & \mathrm{out}(X) \text{ :- } \mathit{not}\, \mathrm{in}(X), \mathrm{arg}(X); \\ & \text{ :- } \mathrm{in}(X), \mathrm{in}(Y), \mathrm{defeat}(X, Y)\}.\end{aligned}$$

Proposition 3.11 *For any AF F, $cf(F) \cong \mathcal{AS}(\pi_{cf}(\widehat{F}))$.*

For our example framework F from Example 3.2, we have as input

$$\begin{aligned}\widehat{F} \quad = \{ \quad & \mathrm{arg}(a), \mathrm{arg}(b), \mathrm{arg}(c), \mathrm{arg}(d), \mathrm{arg}(e), \\ & \mathrm{defeat}(a, b), \mathrm{defeat}(c, b), \mathrm{defeat}(c, d), \mathrm{defeat}(d, c), \mathrm{defeat}(d, e), \mathrm{defeat}(e, e) \ \}.\end{aligned}$$

Moreover, using $\widehat{F}$ together with π_{cf}, we obtain:

$$\mathcal{AS}(\pi_{cf}(\widehat{F})) = \{S_\emptyset, S_a, S_b, S_c, S_d, S_{ac}, S_{ad}, S_{bd}\},$$

where we denote by S_α the following sets:

$$\begin{aligned}S_\emptyset &= \widehat{F} \cup \{\mathrm{out}(a), \mathrm{out}(b), \mathrm{out}(c), \mathrm{out}(d), \mathrm{out}(e)\}, \\ S_a &= \widehat{F} \cup \{\mathrm{in}(a), \mathrm{out}(b), \mathrm{out}(c), \mathrm{out}(d), \mathrm{out}(e)\}, \\ S_b &= \widehat{F} \cup \{\mathrm{out}(a), \mathrm{in}(b), \mathrm{out}(c), \mathrm{out}(d), \mathrm{out}(e)\}, \\ S_c &= \widehat{F} \cup \{\mathrm{out}(a), \mathrm{out}(b), \mathrm{in}(c), \mathrm{out}(d), \mathrm{out}(e)\}, \\ S_d &= \widehat{F} \cup \{\mathrm{out}(a), \mathrm{out}(b), \mathrm{out}(c), \mathrm{in}(d), \mathrm{out}(e)\}, \\ S_{ac} &= \widehat{F} \cup \{\mathrm{in}(a), \mathrm{out}(b), \mathrm{in}(c), \mathrm{out}(d), \mathrm{out}(e)\}, \\ S_{ad} &= \widehat{F} \cup \{\mathrm{in}(a), \mathrm{out}(b), \mathrm{out}(c), \mathrm{in}(d), \mathrm{out}(e)\}, \\ S_{bd} &= \widehat{F} \cup \{\mathrm{out}(a), \mathrm{in}(b), \mathrm{out}(c), \mathrm{in}(d), \mathrm{out}(e)\}.\end{aligned}$$

We are now already well prepared to present the first encoding which is concerned with stable extensions. The additional rules for the stability test are as follows:

$$\begin{aligned}\pi_{stable} \quad = \pi_{cf} \cup \{ \quad & \mathrm{defeated}(X) \text{ :- } \mathrm{in}(Y), \mathrm{defeat}(Y, X); \\ & \text{ :- } \mathrm{out}(X), \mathit{not}\, \mathrm{defeated}(X)\}.\end{aligned}$$

The first rule computes those arguments attacked by the current guess, while the constraint eliminates those guesses where some argument not contained in the guess remains undefeated.

For our example, let us first consider the collection C of answer sets of $\pi_{cf}(\widehat{F}) \cup \{\mathrm{defeated}(X) \text{ :- } \mathrm{in}(Y), \mathrm{defeat}(Y,X)\}$. Note that we can use the splitting theorem and, therefore, make direct use of the answer sets of $\pi_{cf}(\widehat{F})$. In fact, using our calculations from above we obtain

$$\begin{aligned} C \;= \{\;& S_\emptyset, \\ & S_a \cup \{\mathrm{defeated}(b)\}, \\ & S_b, \\ & S_c \cup \{\mathrm{defeated}(b), \mathrm{defeated}(d)\}, \\ & S_d \cup \{\mathrm{defeated}(c), \mathrm{defeated}(e)\}, \\ & S_{ac} \cup \{\mathrm{defeated}(b), \mathrm{defeated}(d)\}, \\ & S_{ad} \cup \{\mathrm{defeated}(b), \mathrm{defeated}(c), \mathrm{defeated}(e)\}, \\ & S_{bd} \cup \{\mathrm{defeated}(c), \mathrm{defeated}(e)\}\;\}. \end{aligned}$$

If we now apply the constraint $\text{:- } \mathrm{out}(X), \mathit{not}\, \mathrm{defeated}(X)$ to each element in C, we observe that any set from C except $S_{ad} \cup \{\mathrm{defeated}(b), \mathrm{defeated}(c), \mathrm{defeated}(e)\}$ is violated by that constraint. In fact, each other set contains at least one atom $\mathrm{out}(y)$ without the matching $\mathrm{defeated}(y)$.

In general, our encoding for stable extensions satisfies the following correspondence result.

Proposition 3.12 *For any AF F, $stable(F) \cong \mathcal{AS}(\pi_{stable}(\widehat{F}))$.*

Next, we give the additional rules for the admissibility test:

$$\begin{aligned} \pi_{adm} \;= \pi_{cf} \cup \{\;& \mathrm{defeated}(X) \text{ :- } \mathrm{in}(Y), \mathrm{defeat}(Y,X); \\ & \text{:- } \mathrm{in}(X), \mathrm{defeat}(Y,X), \mathit{not}\, \mathrm{defeated}(Y)\}. \end{aligned}$$

The first rule is the same as in π_{stable}. The new constraint rules out sets containing a non-defended argument. Indeed, we can identify non-defended arguments as those, which are defeated by an argument, which itself is undefeated.

For our example framework, we thus can start from set C as above but now we check which sets violate the new constraint $\text{:- } \mathrm{in}(X), \mathrm{defeat}(Y,X), \mathit{not}\, \mathrm{defeated}(Y)$. This is the case for two of the candidates. (1) S_b contains $\mathrm{in}(b)$ and $\mathrm{defeat}(a,b)$ but since $\mathrm{defeated}(a)$ is not contained, the constraint applies; (2) for $S_{bd} \cup \{\mathrm{defeated}(c), \mathrm{defeated}(e)\}$ the argumentation is analogously. Hence, we obtain

$$\begin{aligned} \mathcal{AS}(\pi_{adm}(\widehat{F})) \;= \{\;& S_\emptyset, \\ & S_a \cup \{\mathrm{defeated}(b)\}, \\ & S_c \cup \{\mathrm{defeated}(b), \mathrm{defeated}(d)\}, \\ & S_d \cup \{\mathrm{defeated}(c), \mathrm{defeated}(e)\}, \\ & S_{ac} \cup \{\mathrm{defeated}(b), \mathrm{defeated}(d)\}, \\ & S_{ad} \cup \{\mathrm{defeated}(b), \mathrm{defeated}(c), \mathrm{defeated}(e)\}\}. \end{aligned}$$

Again, we observe the one-to-one correspondence to the admissible extensions of F. The general result is as follows.

Proposition 3.13 *For any AF F, $adm(F) \cong \mathcal{AS}(\pi_{adm}(\widehat{F}))$.*

We proceed with the encoding for complete extensions, which is also quite straightforward. We define

$$\begin{aligned}\pi_{comp} \;=\; \pi_{adm} \cup \{&\mathrm{not\_defended}(X) \text{ :- } \mathrm{defeat}(Y,X), \mathit{not}\ \mathrm{defeated}(Y);\\ &\text{ :- } \mathrm{out}(X), \mathit{not}\ \mathrm{not\_defended}(X)\}.\end{aligned}$$

Once more, we use our running example to illustrate the functioning of π_{comp}. Again, we proceed in two steps and first compute the answer sets of the program without the constraint $\text{ :- } \mathrm{out}(X), \mathit{not}\ \mathrm{not\_defended}(X)$. Here, we can directly use the sets from $\mathcal{AS}(\pi_{adm}(\widehat{F}))$ and check which predicates $\mathrm{not\_defended}(\cdot)$ can be derived. The answer sets of $\pi_{adm}(\widehat{F}) \cup \{\mathrm{not\_defended}(X) \text{ :- } \mathrm{defeat}(Y,X), \mathit{not}\ \mathrm{defeated}(Y)\}$ are

$$\begin{aligned}
S_\emptyset \;\cup\; &\{\mathrm{not_defended}(b), \mathrm{not_defended}(c), \mathrm{not_defended}(d), \mathrm{not_defended}(e)\},\\
S_a \;\cup\; &\{\mathrm{defeated}(b), \mathrm{not_defended}(b), \mathrm{not_defended}(c), \mathrm{not_defended}(d), \mathrm{not_defended}(e)\},\\
S_c \;\cup\; &\{\mathrm{defeated}(b), \mathrm{defeated}(d), \mathrm{not_defended}(b), \mathrm{not_defended}(d), \mathrm{not_defended}(e)\},\\
S_d \;\cup\; &\{\mathrm{defeated}(c), \mathrm{defeated}(e), \mathrm{not_defended}(b), \mathrm{not_defended}(c), \mathrm{not_defended}(e)\},\\
S_{ac} \;\cup\; &\{\mathrm{defeated}(b), \mathrm{defeated}(d), \mathrm{not_defended}(b), \mathrm{not_defended}(d), \mathrm{not_defended}(e)\},\\
S_{ad} \;\cup\; &\{\mathrm{defeated}(b), \mathrm{defeated}(c), \mathrm{defeated}(e), \mathrm{not_defended}(b), \mathrm{not_defended}(c), \mathrm{not_defended}(e)\}.
\end{aligned}$$

Obviously, each candidate which contains $\mathrm{out}(a)$ is ruled out by the constraint $\text{ :- } \mathrm{out}(X), \mathit{not}\ \mathrm{not\_defended}(X)$, since no candidate set contains $\mathrm{not\_defended}(a)$. One can check that all other sets do not violate the constraint, and thus are answer sets of $\pi_{comp}(\widehat{F})$. Again, these remaining three sets characterize the complete extensions of F, as desired.

Proposition 3.14 *For any AF F, $comp(F) \cong \mathcal{AS}(\pi_{comp}(\widehat{F}))$.*

We now turn to the grounded extension. For suitably encoding the operator Γ_F, we can come up with a *stratified* program for this task. Note that in a stratified program it is not possible to first guess a candidate for the extension and then check whether the guess satisfies certain conditions. Instead, we "fill" the $\mathrm{in}(\cdot)$-predicate according to the definition of the operator Γ_F. To compute (without unstratified negation) the required predicate for being defended, we now make use of the order $<$ over the domain elements and derive corresponding predicates for infimum, supremum, and successor.

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

We now define the desired predicate $\mathrm{defended}(X)$ which itself is obtained via a predicate $\mathrm{defended\_upto}(X, Y)$ with the intended meaning that argument X is defended by the current assignment with respect to all arguments $U \leq Y$. In other words, we let range Y starting from the infimum and then using the defined successor predicate to derive $\mathrm{defended\_upto}(X, Y)$ for "increasing" Y. If we arrive at the supremum element in this way, we finally derive $\mathrm{defended}(X)$. We define

$$\begin{aligned}
\pi_{defended} = \{ \quad & \mathrm{defended_upto}(X, Y) \text{ :- } \mathrm{inf}(Y), \mathrm{arg}(X), \mathit{not}\, \mathrm{defeat}(Y, X); \\
& \mathrm{defended_upto}(X, Y) \text{ :- } \mathrm{inf}(Y), \mathrm{in}(Z), \mathrm{defeat}(Z, Y), \mathrm{defeat}(Y, X); \\
& \mathrm{defended_upto}(X, Y) \text{ :- } \mathrm{succ}(Z, Y), \mathrm{defended_upto}(X, Z), \\
& \qquad\qquad \mathit{not}\, \mathrm{defeat}(Y, X); \\
& \mathrm{defended_upto}(X, Y) \text{ :- } \mathrm{succ}(Z, Y), \mathrm{defended_upto}(X, Z), \\
& \qquad\qquad \mathrm{in}(V), \mathrm{defeat}(V, Y), \mathrm{defeat}(Y, X); \\
& \mathrm{defended}(X) \text{ :- } \mathrm{sup}(Y), \mathrm{defended_upto}(X, Y)\}, \text{ and} \\
\pi_{ground} = \quad & \pi_{<} \cup \pi_{defended} \cup \{\mathrm{in}(X) \text{ :- } \mathrm{defended}(X)\}.
\end{aligned}$$

Note that π_{ground} is indeed stratified.

We illustrate the building blocks for π_{ground} using our example framework F. Moreover, we assume as order $a < b < c < d < e$. For this order, $\pi_{<}$ yields a single answer set S_0 which contains (among other atoms, which will not be used in later calculations):

$$\{\mathrm{inf}(a), \mathrm{succ}(a, b), \mathrm{succ}(b, c), \mathrm{succ}(c, d), \mathrm{succ}(d, e), \mathrm{sup}(e)\} \subseteq S_0$$

We now compute the answer set for $\widehat{F} \cup \pi_{<} \cup \pi_{defended}$ step by step. In the "first round" we have no $\mathrm{in}(\cdot)$ predicate derived so far, hence only the first and third rule in $\pi_{defended}$ are of interest. In fact, for $\mathrm{inf}(a)$, the first rule in $\pi_{defended}$ yields:

$$\mathrm{defended\_upto}(a, a), \mathrm{defended\_upto}(c, a), \mathrm{defended\_upto}(d, a), \mathrm{defended\_upto}(e, a);$$

note that $\mathrm{defended\_upto}(b, a)$ is missing, since we have $\mathrm{defeat}(a, b) \in \widehat{F}$. Now we use $\mathrm{succ}(a, b)$ and obtain

$$\mathrm{defended\_upto}(a, b), \mathrm{defended\_upto}(c, b), \mathrm{defended\_upto}(d, b), \mathrm{defended\_upto}(e, b).$$

The remaining atoms we derive are

$$\mathrm{defended\_upto}(a, c), \mathrm{defended\_upto}(c, c), \mathrm{defended\_upto}(e, c);$$

(since d is attacked by c, $\mathrm{defended\_upto}(d, c)$ cannot be derived) and finally,

$$\mathrm{defended\_upto}(a, d), \mathrm{defended\_upto}(a, e).$$

Hence, we obtain $\mathrm{defended}(a)$ via $\mathrm{sup}(e)$ and $\mathrm{defended\_upto}(a, e)$. Moreover, the rule $\mathrm{in}(X)$:- $\mathrm{defended}(X)$ derives $\mathrm{in}(a)$. We now can use the additional fact $\mathrm{in}(a)$ for a second round

of evaluating $\pi_{defended}$, in particular, by using the second and fourth rule in $\pi_{defended}$. However, as a does not defend any argument, it can be checked that no further atoms can be derived. Thus we obtain that in the single answer set of $\pi_{ground}(\widehat{F})$ the only $\mathrm{in}(\cdot)$ predicate is $\mathrm{in}(a)$. However, this corresponds to the grounded extensions of F.

Proposition 3.15 *For any AF F, $ground(F) \cong \mathcal{AS}(\pi_{ground}(\widehat{F}))$.*

Obviously, we could have used the $\mathrm{defended}(\cdot)$ predicate in previous programs. Indeed, π_{comp} could be defined as

$$\pi_{cf} \cup \pi_{defended} \cup \{ \text{:-}\, \mathrm{in}(X), \mathit{not}\, \mathrm{defended}(X);\ \ \text{:-}\, \mathrm{out}(X), \mathrm{defended}(X)\}.$$

We continue with the more involved encodings for preferred and semi-stable extensions. Compared to the one for admissible extensions, these encodings require an additional maximality test. However, this is sometimes quite complicate to encode (see also [19] for a thorough discussion on this issue).

In fact, to compute the preferred extensions, we will use a saturation technique as follows: Having computed an admissible extension S (characterized via predicates $\mathrm{in}(\cdot)$ and $\mathrm{out}(\cdot)$) , we perform a second guess using new predicates, say $\mathrm{inN}(\cdot)$ and $\mathrm{outN}(\cdot)$, such that they represent a guess $T \supset S$. For that guess, we will use disjunction (rather than default negation), which allows that for each a *both* $\mathrm{inN}(a)$ and $\mathrm{outN}(a)$ are contained in a possible answer set (under certain conditions). In fact, exactly such answer sets will correspond to the preferred extensions. The saturation is therefore performed in such a way that all predicates $\mathrm{inN}(a)$ and $\mathrm{outN}(a)$ are derived for those T, which do *not* characterize an admissible extension. If this saturation succeeds for each $T \supset S$, we want that saturated interpretation to become an answer set. This can be done by using a saturation predicate spoil, which is handled via a constraint $\text{:-}\, \mathit{not}\, \mathrm{spoil}$. This ensures that only saturated guesses survive.

Such saturation techniques always require a restricted use of negation. The predicates defined in $\pi_<$ will serve for this purpose. Two new predicates are needed: predicate eq which indicates whether a guess T represented by atoms $\mathrm{inN}(\cdot)$ and $\mathrm{outN}(\cdot)$ is equal to the guess for S (represented by atoms $\mathrm{in}(\cdot)$ and $\mathrm{out}(\cdot)$). The second predicate we define is $\mathrm{undefeated}(X)$ which indicates that X is not defeated by any element from T. Both predicates are computed via predicates $\mathrm{eq\_upto}(\cdot)$ (resp. $\mathrm{undefeated\_upto}(\cdot,\cdot)$) in the same manner as we used $\mathrm{defended\_upto}(\cdot,\cdot)$ for $\mathrm{defended}(\cdot)$ in the module $\pi_{defended}$ above:

$$\begin{aligned}
\pi_{eq} \ = \{\ & \mathrm{eq_upto}(Y) \text{:-}\, \mathrm{inf}(Y), \mathrm{in}(Y), \mathrm{inN}(Y);\\
& \mathrm{eq_upto}(Y) \text{:-}\, \mathrm{inf}(Y), \mathrm{out}(Y), \mathrm{outN}(Y);\\
& \mathrm{eq_upto}(Y) \text{:-}\, \mathrm{succ}(Z,Y), \mathrm{in}(Y), \mathrm{inN}(Y), \mathrm{eq_upto}(Z);\\
& \mathrm{eq_upto}(Y) \text{:-}\, \mathrm{succ}(Z,Y), \mathrm{out}(Y), \mathrm{outN}(Y), \mathrm{eq_upto}(Z);\\
& \mathrm{eq} \text{:-}\, \mathrm{sup}(Y), \mathrm{eq_upto}(Y)\};
\end{aligned}$$

$$
\begin{aligned}
\pi_{undefeated} = \{\ & \mathrm{undefeated\_upto}(X,Y) \text{ :- } \mathrm{inf}(Y), \mathrm{outN}(X), \mathrm{outN}(Y);\\
& \mathrm{undefeated\_upto}(X,Y) \text{ :- } \mathrm{inf}(Y), \mathrm{outN}(X), \mathit{not}\ \mathrm{defeat}(Y,X);\\
& \mathrm{undefeated\_upto}(X,Y) \text{ :- } \mathrm{succ}(Z,Y), \mathrm{undefeated\_upto}(X,Z),\\
& \qquad\qquad \mathrm{outN}(Y);\\
& \mathrm{undefeated\_upto}(X,Y) \text{ :- } \mathrm{succ}(Z,Y), \mathrm{undefeated\_upto}(X,Z),\\
& \qquad\qquad \mathit{not}\ \mathrm{defeat}(Y,X);\\
& \mathrm{undefeated}(X) \text{ :- } \mathrm{sup}(Y), \mathrm{undefeated\_upto}(X,Y)\}.
\end{aligned}
$$

With these predicates at hand, we next define the spoiling module for preferred extensions:

$$
\begin{aligned}
\pi_{spoilpref} = \{\ & \mathrm{inN}(X) \vee \mathrm{outN}(X) \text{ :- } \mathrm{out}(X);\ \mathrm{inN}(X) \text{ :- } \mathrm{in}(X); && (1)\\
& \mathrm{spoil} \text{ :- } \mathrm{eq}; && (2)\\
& \mathrm{spoil} \text{ :- } \mathrm{inN}(X), \mathrm{inN}(Y), \mathrm{defeat}(X,Y); && (3)\\
& \mathrm{spoil} \text{ :- } \mathrm{inN}(X), \mathrm{outN}(Y), \mathrm{defeat}(Y,X), \mathrm{undefeated}(Y); && (4)\\
& \mathrm{inN}(X) \text{ :- } \mathrm{spoil}, \mathrm{arg}(X);\ \mathrm{outN}(X) \text{ :- } \mathrm{spoil}, \mathrm{arg}(X); && (5)\\
& \text{ :- } \mathit{not}\ \mathrm{spoil}\}. && (6)
\end{aligned}
$$

We define

$$\pi_{pref} = \pi_{adm} \cup \pi_{<} \cup \pi_{eq} \cup \pi_{undefeated} \cup \pi_{spoilpref}\,.$$

When joined with $\widehat{F}$ for some AF $F = (A, R)$, the rules of $\pi_{spoilpref}$ work as follows: Rules (1) guess a new set $T \subseteq A$ (via predicates $\mathrm{inN}(\cdot)$ and $\mathrm{outN}(\cdot)$), which compares to the guess $S \subseteq A$ (S is characterized by predicates $\mathrm{in}(\cdot)$ and $\mathrm{out}(\cdot)$ as used in π_{adm}) as $S \subseteq T$. In case $T = S$, we obtain predicate eq and derive predicate spoil (by rule (2)). The remaining guesses for T are now handled as follows. First, rule (3) derives predicate spoil if the new guess T contains a conflict. Second, rule (4) derives spoil if the new guess T contains an element which is attacked by an argument outside T which itself is undefeated (by T). Hence, we derived spoil for those $S \subseteq T$ where either $S = T$ or T did not correspond to an admissible extension of F. We now finally spoil up the current guess and derive all $\mathrm{inN}(a)$ and $\mathrm{outN}(a)$ in rules (5). Recall that due to constraint (6) such spoiled interpretations are the only candidates for answer sets. To turn them into an answer set, it is however necessary that we spoiled for *each* T, such that $S \subseteq T$; but by definition this is exactly the case if S is a preferred extension.

To illustrate how π_{pref} applies to our example framework, note that a step-by-step evaluation as used before is no longer possible. In particular, the sub-program $\Pi = \pi_{\mathrm{eq}} \cup \pi_{undefeated} \cup \pi_{\mathrm{spoil}}$ has to be treated as once, due to the cyclic dependencies among the atoms (in other words, we only obtain trivial splitting sets for Π). However, we can still split π_{pref} into $\pi_{adm} \cup \pi_{<}$ and Π. We already know the single answer set S_0 of $\pi_{<}(\widehat{F})$ and the collection $\mathcal{AS}(\pi_{adm}(\widehat{F}))$ of answer sets of $\pi_{adm}(\widehat{F})$. As is easily checked, we thus get $\mathcal{AS}(\widehat{F} \cup \pi_{adm} \cup \pi_{<}) = \{S_0 \cup S \mid S \in \pi_{adm}(\widehat{F})\}$. Hence, let us illustrate the functioning of Π for the two inputs[1] $S_1 = S_0 \cup S_a$ and $S_2 = S_0 \cup S_{ac}$. Indeed, we expect that S_1 does not lead to an answer set of $\pi_{pref}(\widehat{F})$, while the second set S_2

[1]We omit the further atoms from the corresponding answer sets in $\widehat{F} \cup \pi_{adm} \cup \pi_{<}$, since they play no role in Π.

corresponds to a preferred extension of F, and thus should be part of an answer set of $\pi_{pref}(\widehat{F})$. As discussed above, the only potential answer set I_1 of $\Pi(S_1)$ contains contains S_1 as well as atoms

$$\mathrm{inN}(a), \mathrm{outN}(a), \mathrm{inN}(b), \mathrm{outN}(b), \mathrm{inN}(c), \mathrm{outN}(c), \mathrm{inN}(d), \mathrm{outN}(d), \mathrm{inN}(d), \mathrm{outN}(e), \mathrm{spoil}. \quad (7)$$

We next check whether some $J_1 \subset I_1$ satisfies $\Pi(S_1)^{I_1} = \Pi(S_1) \setminus \{ :\!\!- \mathit{not}\, \mathrm{spoil}\}$. If this is *not* the case, I_1 becomes an answer set. Indeed, one can check that

$$S_a \cup \{\mathrm{inN}(a), \mathrm{outN}(b), \mathrm{inN}(c), \mathrm{outN}(d), \mathrm{outN}(e)\}$$

satisfies $\Pi(S_1)^{I_1}$. This can be seen as follows: this set does not contain spoil, thus the bodies of rules (2–4) must not be satisfied. For the first rule this is the case since eq is not derived (we leave it to the reader to check this), for the second rule this is the case as well, since the vertices for which $\mathrm{inN}(\cdot)$ holds are not adjacent. Finally, for (4), we first mention that $\pi_{\mathit{undefeated}}$ is derived for the following instantiations $\mathit{undefeated}(a)$, $\mathit{undefeated}(c)$, $\mathit{undefeated}(e)$. One can now check that the bodies of (4) are not satisfied. As well, rules (5) are not applied (since spoil has not been derived). Thus, we found a proper subset J_1 of I_1, such that $J_1 \models \Pi(S_1)^{I_1}$. Consequently, I_1 cannot be an answer set of $\Pi(S_1)$ and thus not of $\pi_{pref}(\widehat{F})$.

The situation is different for set $S_2 = S_0 \cup S_{ac}$. As before the only potential answer set I_2 of $\Pi(S_2)$ contains S_2 as well as atoms

$$\mathrm{inN}(a), \mathrm{outN}(a), \mathrm{inN}(b), \mathrm{outN}(b), \mathrm{inN}(c), \mathrm{outN}(c), \mathrm{inN}(d), \mathrm{outN}(d), \mathrm{inN}(d), \mathrm{outN}(e), \mathrm{spoil}. \quad (8)$$

Moreover, $\Pi(S_2)^{I_2} = \Pi(S_2) \setminus \{ :\!\!- \mathit{not}\, \mathrm{spoil}\}$ as before, and we thus seek for sets $J_2 \subset I_2$, such that $J_2 \models \Pi(S_2)^{I_2}$. Note that rule (1) guarantees that J_2 contains at least $\mathrm{inN}(a), \mathrm{inN}(c)$ but further $\mathrm{inN}(\cdot)$ predicates could be contained in J_2. However, if the only $\mathrm{inN}(\cdot)$ predicates in J_2 are $\mathrm{inN}(a), \mathrm{inN}(c)$, predicate eq is derived and we spoil. As well, if a further $\mathrm{inN}(\cdot)$ predicate is contained in J_2 then we already know that such a set characterizes a subset $S' \subseteq A$ which cannot be conflict free. Indeed, rule (3) applies in this case, and we obtain spoil. As soon as spoil is derived, rules (5) "turn J_2 into I_2". From this observation it is clear that we cannot find a $J_2 \subset I_2$, such that $J_2 \models \Pi(S_2)^{I_2}$. Thus I_2 becomes an answer set of $\Pi(S_2)$ and therefore also of $\pi_{pref}(\widehat{F})$. This meets our expectation, since S_{ac} relates to the preferred extension $\{a, c\}$ of F.

Proposition 3.16 *For any AF F, $\mathit{pref}(F) \cong \mathcal{AS}(\pi_{pref}(\widehat{F}))$.*

We conclude our encodings for the different types of extensions with the program for the semi-stable semantics. The basic intuition for the forthcoming encoding is as for the preferred semantics. The main difference lies in the fact that, given an admissible extension S for an AF $F = (A, R)$, we now have to test whether no $T \in \mathit{adm}(F)$ with $S_R^+ \subset T_R^+$ exists, while for preferred extensions it was sufficient to test whether no such T of the form $S \subset T$ exists. This requires the following changes. First, we have to guess an arbitrary set T (for preferred extensions we could restrict ourselves to supersets of S). Then we spoil (as before) in case T is not admissible. Finally, we explicitly get rid off the cases where $S_R^+ \not\subset T_R^+$ (for preferred extensions, we only had to exclude

the case $S = T$ via the predicate eq). Hence, we need a new predicate eqplus which tests for $S_R^+ = T_R^+$, and we spoil if eqplus is derived, or in case there exists an $a \in S_R^+$ not contained in T_R^+.

We can reuse the modules π_{adm}, $\pi_<$, as well as $\pi_{undefeated}$ and define the following additional rules

$$
\begin{aligned}
\pi_{eq}^+ \;= \{\;& \mathrm{eqplus\_upto}(Y) \text{ :- } \mathrm{inf}(Y), \mathrm{in}(Y), \mathrm{inN}(Y);\\
& \mathrm{eqplus\_upto}(Y) \text{ :- } \mathrm{inf}(Y), \mathrm{in}(Y), \mathrm{inN}(X), \mathrm{defeat}(X,Y);\\
& \mathrm{eqplus\_upto}(Y) \text{ :- } \mathrm{inf}(Y), \mathrm{in}(X), \mathrm{inN}(Y), \mathrm{defeat}(X,Y);\\
& \mathrm{eqplus\_upto}(Y) \text{ :- } \mathrm{inf}(Y), \mathrm{in}(X), \mathrm{inN}(Z), \mathrm{defeat}(X,Y), \mathrm{defeat}(Z,Y);\\
& \mathrm{eqplus\_upto}(Y) \text{ :- } \mathrm{inf}(Y), \mathrm{out}(Y), \mathrm{outN}(Y), \mathit{not}\ \mathrm{defeated}(Y), \mathrm{undefeated}(Y);\\
& \mathrm{eqplus\_upto}(Y) \text{ :- } \mathrm{succ}(Z,Y), \mathrm{in}(Y), \mathrm{inN}(Y), \mathrm{eqplus\_upto}(Z);\\
& \mathrm{eqplus\_upto}(Y) \text{ :- } \mathrm{succ}(Z,Y), \mathrm{in}(Y), \mathrm{inN}(X), \mathrm{defeat}(X,Y), \mathrm{eqplus\_upto}(Z);\\
& \mathrm{eqplus\_upto}(Y) \text{ :- } \mathrm{succ}(Z,Y), \mathrm{in}(X), \mathrm{inN}(Y), \mathrm{defeat}(X,Y), \mathrm{eqplus\_upto}(Z);\\
& \mathrm{eqplus\_upto}(Y) \text{ :- } \mathrm{succ}(Z,Y), \mathrm{in}(X), \mathrm{inN}(U), \mathrm{defeat}(X,Y), \mathrm{defeat}(U,Y),\\
& \qquad\qquad \mathrm{eqplus\_upto}(Z);\\
& \mathrm{eqplus\_upto}(Y) \text{ :- } \mathrm{succ}(Z,Y), \mathrm{out}(Y), \mathrm{outN}(Y), \mathit{not}\ \mathrm{defeated}(Y), \mathrm{undefeated}(Y),\\
& \qquad\qquad \mathrm{eqplus\_upto}(Z);\\
& \mathrm{eqplus} \text{ :- } \mathrm{sup}(Y), \mathrm{eqplus\_upto}(Y)\};
\end{aligned}
$$

$$
\begin{aligned}
\pi_{spoilsemi} \;= \{\;& \mathrm{inN}(X) \vee \mathrm{outN}(X) \text{ :- } \mathrm{arg}(X);\\
& \mathrm{spoil} \text{ :- } \mathrm{eqplus};\\
& \mathrm{spoil} \text{ :- } \mathrm{inN}(X), \mathrm{inN}(Y), \mathrm{defeat}(X,Y);\\
& \mathrm{spoil} \text{ :- } \mathrm{inN}(X), \mathrm{outN}(Y), \mathrm{defeat}(Y,X), \mathrm{undefeated}(Y);\\
& \mathrm{spoil} \text{ :- } \mathrm{in}(X), \mathrm{outN}(X), \mathrm{undefeated}(X);\\
& \mathrm{spoil} \text{ :- } \mathrm{in}(Y), \mathrm{defeat}(Y,X), \mathrm{outN}(X), \mathrm{undefeated}(X);\\
& \mathrm{inN}(X) \text{ :- } \mathrm{spoil}, \mathrm{arg}(X);\ \mathrm{outN}(X) \text{ :- } \mathrm{spoil}, \mathrm{arg}(X);\\
& \text{ :- } \mathit{not}\ \mathrm{spoil}\}.
\end{aligned}
$$

We define

$$\pi_{semi} = \pi_{adm} \cup \pi_< \cup \pi_{eq}^+ \cup \pi_{undefeated} \cup \pi_{spoilsemi}$$

and obtain the following result.

Proposition 3.17 *For any AF F, $semi(F) \cong \mathcal{AS}(\pi_{semi}(\widehat{F}))$.*

We summarize the results from this section.

Theorem 3.18 *For any AF F and $e \in \{stable, adm, pref, semi, comp, ground\}$, it holds that $e(F) \cong \mathcal{AS}(\pi_e(\widehat{F}))$.*

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

Table 3: Overview of the encodings of the reasoning tasks for AF $F = (A, R)$ and $a \in A$.

We note that our encodings are *adequate* in the sense that the data complexity of the encodings mirrors the complexity of the encoded task. In fact, depending on the chosen reasoning task, the adequate encodings are depicted in Table 3. Recall that credulous reasoning over preferred extensions reduces to credulous reasoning over admissible extensions and skeptical reasoning over complete extensions reduces to reasoning over the single grounded extension. The only proper disjunctive programs involved are π_{pref} and π_{semi}, all other encodings are disjunction-free. Moreover, π_{ground} is stratified. Stratified programs have at most one answer set, hence there is no need to distinguish between $\models_c$ and $\models_s$. If one now assigns the complexity entries from Table 1 to the encodings as depicted in Table 3, one obtains Table 2.

However, we also can encode more involved decision problems using our programs. As a first example consider the Π_2^P-complete problem of *coherence* [15], which decides whether for a given AF F, $pref(F) \subseteq stable(F)$ (recall that $pref(F) \supseteq stable(F)$ always holds). We can decide this problem by extending π_{pref} in such a way that an answer-set of π_{pref} survives only if it does not correspond to a stable extension. By definition, the only possibility to do so is if some undefeated argument is not contained in the extension.

Corollary 3.19 *The coherence problem for an AF F holds iff the program*

$$\pi_{pref}(\widehat{F}) \cup \{v \text{ :- } \mathrm{out}(X), \mathit{not}\, \mathrm{defeated}(X);\ \text{ :- } \mathit{not}\, v\}$$

has no answer set.

As a second example, we give a program which decides, for a given AF F, whether the semi-stable and the preferred extension of F coincide. This problem has been shown to be Π_2^P-complete in [16].

Again, we can decide this problem by reusing some of the modules from previous encodings. In this particular case, however, we need to separate some of the atoms which are used in common by π_{pref} and π_{semi}. For this reason, we require new atoms $\mathrm{inNN}(\cdot), \mathrm{outNN}(\cdot), \mathrm{undefeatedN}(\cdot)$ and $\mathrm{undefeatedN\_upto}(\cdot,\cdot)$, and denote by $\pi_{\mathrm{undefeatedN}}$ the program resulting from $\pi_{\mathrm{undefeated}}$ by using the new atoms instead of $\mathrm{inN}(\cdot), \mathrm{outN}(\cdot), \mathrm{undefeated}(\cdot)$ and $\mathrm{undefeated\_upto}(\cdot,\cdot)$, respectively.

Similarly, we obtain π^+_{eqN} from π^+_{eq}. Consider now the following program

$$\begin{aligned}\pi_{coinicde} \;=\; & \pi_{pref} \cup \pi_{\mathrm{undefeatedN}} \cup \pi^+_{eqN} \cup \{ \\ & \mathrm{inNN}(X) \vee \mathrm{outNN}(X) \text{ :- } \mathrm{arg}(X); \\ & \text{ :- } \mathrm{eqplus}; \\ & \text{ :- } \mathrm{inNN}(X), \mathrm{inNN}(Y), \mathrm{defeat}(X,Y); \\ & \text{ :- } \mathrm{inNN}(X), \mathrm{outNN}(Y), \mathrm{defeat}(Y,X), \mathrm{undefeatedN}(Y); \\ & \text{ :- } \mathrm{in}(X), \mathrm{outNN}(X), \mathrm{undefeated}(X); \\ & \text{ :- } \mathrm{in}(Y), \mathrm{defeat}(Y,X), \mathrm{outNN}(X), \mathrm{undefeatedN}(X)\}.\end{aligned}$$

Corollary 3.20 *Given an AF F, it holds that $semi(F) = pref(F)$ iff $\pi_{coinicde}(\widehat{F})$ has no answer set.*

Roughly speaking we combine here the program which computes the preferred extensions with a program which checks whether the input is *not* semi-stable. The latter test can be accomplished via constraints (instead of the spoiling technique used above), since it is sufficient here to just get rid off candidates which already have been checked to be preferred but are not semi-stable.

4 Encodings for Generalizations of Argumentation Frameworks

4.1 Value-Based Argumentation Frameworks

As a first example for generalizing basic AFs, we deal with value-based argumentation frameworks (VAFs) [5] which themselves generalize the preference-based argumentation frameworks [1]. Again we give the definition wrt the universe $\mathcal{U}$.

Definition 4.1 *A* value-based argumentation framework (VAF) *is a 5-tuple $F = (A, R, \Sigma, \sigma, <)$ where $A \subseteq \mathcal{U}$ are arguments, $R \subseteq A \times A$, $\Sigma \subseteq \mathcal{U}$ is a non-empty set of values disjoint from A, $\sigma : A \to \Sigma$ assigns a value to each argument from A , and $<$ is a preference relation (irreflexive, asymmetric) between values.*

Let $\ll$ be the transitive closure of $<$. An argument $a \in A$ defeats *an argument $b \in A$ in F if and only if $(a, b) \in R$ and $(b, a) \notin \ll$.*

Using this notion of defeat, we say in accordance to Definition 3.1 that a set $S \subseteq A$ of arguments *defeats* b (in F), if there is an $a \in S$ which defeats b. An argument $a \in A$ is *defended* by $S \subseteq A$ (in F) iff, for each $b \in A$, it holds that, if b defeats a in F, then S defeats b in F. Using these notions of defeat and defense, the definitions in [5] for conflict-free sets, admissible extensions, and preferred extensions are exactly along the lines of Definition 3.3, 3.5, and 3.6, respectively.

In order to compute these extensions for VAFs, we thus only need to slightly adapt the modules introduced in Section 3.2. In fact, we just overwrite $\widehat{F}$ for a VAF F as

$$\begin{aligned}\widehat{F} \;=\; & \{\mathrm{arg}(a) \mid a \in A\} \cup \{\mathrm{attack}(a,b) \mid (a,b) \in R\} \cup \\ & \{\mathrm{val}(a,\sigma(a)) \mid a \in A\} \cup \{\mathrm{valpref}(w,v) \mid v < w\}\end{aligned}$$

and we require one further module, which now obtains the $\mathrm{defeat}(\cdot,\cdot)$ relation accordingly:

$$\begin{aligned}\pi_{vaf} \;=\; \{\; & \mathrm{valpref}(X,Z) \text{ :- } \mathrm{valpref}(X,Y), \mathrm{valpref}(Y,Z); \\ & \mathrm{pref}(X,Y) \text{ :- } \mathrm{valpref}(U,V), \mathrm{val}(X,U), \mathrm{val}(Y,V); \\ & \mathrm{defeat}(X,Y) \text{ :- } \mathrm{attack}(X,Y), \mathit{not}\, \mathrm{pref}(Y,X)\}.\end{aligned}$$

We obtain the following theorem using the new concepts for $\widehat{F}$ and π_{vaf}, as well as re-using π_{adm} and π_{pref} from Section 3.2.

Theorem 4.2 *For any VAF F and $e \in \{adm, pref\}$, $e(F) \cong \mathcal{AS}(\pi_{vaf} \cup \pi_e(\widehat{F}))$.*

For the other notions of extensions, we can employ our encodings from Section 3.2 in a similar way. The concrete composition of the modules however depends on the exact definitions, and whether they make use of the notion of a defeat in a uniform way. In [4], for instance, stable extensions for a VAF F are defined as those conflict-free subsets S of arguments, such that each argument not in S is attacked (rather than defeated) by S. Still, we can obtain a suitable encoding quite easily using the following redefined module:

$$\begin{aligned}\pi_{stable} \;=\; \pi_{cf} \cup \{\; & \mathrm{attacked}(X) \text{ :- } \mathrm{in}(Y), \mathrm{attack}(Y,X); \\ & \text{ :- } \mathrm{out}(X), \mathit{not}\, \mathrm{attacked}(X)\}.\end{aligned}$$

Theorem 4.3 *For any VAF F, $stable(F) \cong \mathcal{AS}(\pi_{vaf} \cup \pi_{stable}(\widehat{F}))$.*

The coherence problem for VAFs thus can be decided as follows.

Corollary 4.4 *The coherence problem for a VAF F holds iff the program*

$$\begin{aligned}& \pi_{pref}(\widehat{F}) \cup \{\mathrm{attacked}(X) \text{ :- } \mathrm{in}(Y), \mathrm{attack}(Y,X); \\ & \quad v \text{ :- } \mathrm{out}(X), \mathit{not}\, \mathrm{attacked}(X);\;\; \text{ :- } \mathit{not}\, v\}\end{aligned}$$

has no answer set.

4.2 Bipolar Argumentation Frameworks

Bipolar argumentation frameworks [9] augment basic AFs by a second relation between arguments which indicates supports independent from defeats.

Definition 4.5 *A bipolar argumentation framework (BAF) is a tuple $F = (A, R_d, R_s)$ where $A \subseteq \mathcal{U}$ is a set of arguments, and $R_d \subseteq A \times A$ and $R_s \subseteq A \times A$ are the defeat (resp., support) relation of F.*

An argument a defeats *an argument b in F if there exists a sequence $a_1, \ldots, a_{n+1}$ of arguments from A (for $n \geq 1$), such that $a_1 = a$, and $a_{n+1} = b$, and either*

- *$(a_i, a_{i+1}) \in R_s$ for each $1 \leq i \leq n-1$ and $(a_n, a_{n+1}) \in R_d$; or*
- *$(a_1, a_2) \in R_d$ and $(a_i, a_{i+1}) \in R_s$ for each $2 \leq i \leq n$.*

As before, we say that a set $S \subseteq A$ *defeats* an argument b in F if some $a \in S$ defeats b; an argument $a \in A$ is *defended* by $S \subseteq A$ (in F) iff, for each $b \in A$, it holds that, if b defeats a in F, then S defeats b in F.

Again, we just need to adapt the input database $\widehat{F}$ and incorporate the new defeat-relation. Other modules from Section 3.2 can then be reused. In fact, we define for a given BAF $F = (A, R_d, R_s)$,

$$\widehat{F} \;=\; \{\mathrm{arg}(a) \mid a \in A\} \cup \{\mathrm{attack}(a,b) \mid (a,b) \in R_d\} \cup \{\mathrm{support}(a,b) \mid (a,b) \in R_s\},$$

and for the defeat relation we first compute the transitive closure of the $\mathrm{support}(\cdot,\cdot)$-predicate and then define $\mathrm{defeat}(\cdot,\cdot)$ accordingly.

$$\begin{aligned}\pi_{\mathit{baf}} \;= \{\;& \mathrm{support}(X,Z) \text{ :- } \mathrm{support}(X,Y), \mathrm{support}(Y,Z);\\ & \mathrm{defeat}(X,Y) \text{ :- } \mathrm{attack}(X,Y);\\ & \mathrm{defeat}(X,Y) \text{ :- } \mathrm{attack}(Z,Y), \mathrm{support}(X,Z);\\ & \mathrm{defeat}(X,Y) \text{ :- } \mathrm{attack}(X,Z), \mathrm{support}(Z,Y)\}.\end{aligned}$$

Following [9], we can use this notion of defeat to define conflict-free sets, stable extensions, admissible extensions and preferred extensions[2] exactly along the lines of Definition 3.3, 3.4, 3.5, and 3.6, respectively.

Theorem 4.6 *For any BAF F and $e \in \{stable, adm, pref\}$, $e(F) \cong \mathcal{AS}(\pi_{\mathit{baf}} \cup \pi_e(\widehat{F}))$.*

More specific variants of admissible extensions from [9] are obtained by replacing the notion a conflict-free set by other concepts.

Definition 4.7 *Let $F = (A, R_d, R_s)$ be a BAF and $S \subseteq A$. Then S is called* safe *in F if for each $a \in A$, such that S defeats a, $a \notin S$ and there is no sequence $a_1, \ldots, a_n$ ($n \geq 2$), such that $a_1 \in S$, $a_n = a$, and $(a_i, a_{i+1}) \in R_s$, for each $1 \leq i \leq n-1$. A set S is* closed under R_s *if, for each $(a,b) \in R_s$, it holds that $a \in S$ if and only if $b \in S$.*

Note that for a BAF F, each safe set (in F) is conflict-free (in F). We also remark that a set S of arguments is closed under R_s iff S is closed under the transitive closure of R_s.

[2]These extensions are called d-admissible and resp. d-preferred in [9].

Definition 4.8 *Let $F = (A, R_d, R_s)$ be a BAF. A set $S \subseteq A$ is called an* s-admissible extension *of F if S is safe (in F) and each $a \in S$ is defended by S (in F). A set $S \subseteq A$ is called a* c-admissible extension *of F if S is* closed under R_s*, conflict-free (in F), and each $a \in S$ is defended by S (in F). We denote the collection of all s-admissible extensions (resp. of all c-admissible extensions) of F by $sadm(F)$ (resp. by $cadm(F)$).*

We define now further programs as follows

$$\begin{aligned}
\pi_{sadm} &= \pi_{adm} \cup \{ \quad \mathrm{supported}(X) \text{ :- } \mathrm{in}(Y), \mathrm{support}(Y, X); \\
&\qquad\qquad\quad \text{ :- } \mathrm{supported}(X), \mathrm{defeated}(X) \ \} \\
\pi_{cadm} &= \pi_{adm} \cup \{ \quad \text{ :- } \mathrm{support}(X, Y), \mathrm{in}(X), \mathrm{out}(Y); \\
&\qquad\qquad\quad \text{ :- } \mathrm{support}(X, Y), \mathrm{out}(X), \mathrm{in}(Y) \ \}.
\end{aligned}$$

Finally, one defines s-preferred (resp. c-preferred) extensions as maximal (wrt set-inclusion) s-admissible (resp. c-admissible) extensions.

Definition 4.9 *Let $F = (A, R_d, R_s)$ be a BAF. A set $S \subseteq A$ is called an* s-preferred extension *of F if $S \in sadm(F)$ and for each $T \in sadm(F)$, $S \not\subseteq T$. Likewise, a set $S \subseteq A$ is called a* c-preferred extension *of F if $S \in cadm(F)$ and for each $T \in cadm(F)$, $S \not\subseteq T$. By $spref(F)$ (resp. $cpref(F)$) we denote the collection of all s-preferred extensions (resp. of all c-preferred extensions) of F.*

Again, we can reuse parts of the π_{pref}-program from Section 3.2. The only additions necessary are to spoil in case the additional requirements are violated. We define

$$\begin{aligned}
\pi_{spref} &= \pi_{sadm} \cup \pi_{helpers} \cup \pi_{\mathrm{spoil}} \cup \\
&\quad \{ \quad \mathrm{supported}(X) \text{ :- } \mathrm{inN}(Y), \mathrm{support}(Y, X); \\
&\qquad \mathrm{spoil} \text{ :- } \mathrm{supported}(X), \mathrm{defeated}(X) \ \} \\
\pi_{cpref} &= \pi_{cadm} \cup \pi_{helpers} \cup \pi_{\mathrm{spoil}} \cup \\
&\quad \{ \quad \mathrm{spoil} \text{ :- } \mathrm{support}(X, Y), \mathrm{inN}(X), \mathrm{outN}(Y); \\
&\qquad \mathrm{spoil} \text{ :- } \mathrm{support}(X, Y), \mathrm{outN}(X), \mathrm{inN}(Y) \ \}.
\end{aligned}$$

Theorem 4.10 *For any BAF F and $e \in \{sadm, cadm, spref, cpref\}$, we have $e(F) \cong \mathcal{AS}(\pi_{baf} \cup \pi_e(\widehat{F}))$.*

Slightly different semantics for BAFs occur in [2], where the notion of defense is based on R_d, while the notion of conflict remains evaluated with respect to the more general concept of defeat as given in Definition 4.5. However, also such variants can be encoded within our system by a suitable composition of the concepts introduced so far.

Again, we note that we can put together encodings for complete and grounded extensions for BAFs, which have not been studied in the literature.

5 Discussion

In this work we provided logic-program encodings for computing different types of extensions in Dung's argumentation framework as well as in some recent extensions of it. 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]. The encoding (together with some examples) is available on the web and can be run with the answer-set solver DLV [22]. We note that DLV also supplies the built-in predicate $<$ which we used in some of our encodings. Moreover, DLV provides further language-extensions which might lead to alternative encodings; for instance weak constraints could be employed to select the grounded extension from the admissible, or prioritization techniques could be used to compute the preferred extensions.

The work which is closest related to ours is by Nieves *et al*. [25] 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 ideal semantics [14].

References

[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] 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.

[4] Trevor J. M. Bench-Capon. Value-based argumentation frameworks. In *Proceedings of the 9th International Workshop on Non-Monotonic Reasoning* (*NMR'02*), pages 443–454, 2002.

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

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

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

[7] Philippe Besnard and Sylvie Doutre. Checking the acceptability of a set of arguments. In *Proceedings of the 10th International Workshop on Non-Monotonic Reasoning* (*NMR'02*), pages 59–64, 2004.

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

[9] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. On the acceptability of arguments in bipolar argumentation frameworks. In *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty* (*ECSQARU'05*), volume 3571 of *LNCS*, pages 378–389. Springer, 2005.

[10] Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Symmetric argumentation frameworks. In *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty* (*ECSQARU'05*), volume 3571 of *LNCS*, pages 317–328. Springer, 2005.

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

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

[13] 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.

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

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

[16] Paul E. Dunne and Martin Caminada. Computational complexity of semi-stable semantics in abstract argumentation frameworks. In *Proceedings of the 11th European Conference on Logics in Artificial Intelligence* (*JELIA 2008*), volume 5293 of *LNCS*, pages 153–165. Springer, 2008.

[17] 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.

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

[19] Thomas Eiter and Axel Polleres. Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. *Theory and Practice of Logic Programming*, 6(1-2):23–60, 2006.

[20] 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.

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

[22] 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. Log.*, 7(3):499–562, 2006.

[23] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In *Proceedings of the 11th International Conference on Logic Programming* (*ICLP'94*), pages 23–37. MIT Press, 1994.

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

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