Answer-Set Programming Encodings for Argumentation Frameworks

Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran

Institut für Informationssysteme, Technische Universität Wien,
Favoritenstraße 9–11, A–1040 Vienna, Austria

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 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 [1] 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 [2, 3] 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 a datalog program. To be more specific, the system is capable to compute the most important types of extensions (i.e., admissible, preferred, stable, complete, and grounded) in Dung's original framework [4], the preference-based argumentation framework [5], the value-based argumentation framework [6], and the bipolar argumentation framework [7, 8]. Hence our system can be used by researchers to compare different argumentation semantics on concrete examples within a uniform setting. In fact, investigations on the relationship between different argumentation semantics has received increasing interest lately [9].

The declarative programming paradigm of *Answer-Set Programming* (ASP) [10, 11] under the stable-models semantics [12] (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 [13]) are available. Thus, using the proposed reduction method delegates the burden of optimizations to these systems. Second, language extensions 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 [2, 3], 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 [10]. 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 [12]; for further background, see [10, 14].

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 ground rule without disjunction and empty body. An *(input) database* is a 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 (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 $r \in \mathcal{P}$, 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 $U_{\mathcal{P}}$). $Gr(\mathcal{P})$ 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 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 $\mathcal{AS}(\mathcal{P})$.

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 datalog 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. [15] and the references therein).

	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).

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 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 1. 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.

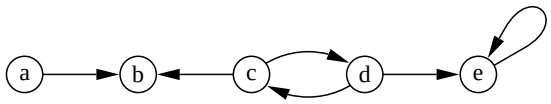

Fig. 1. Graph of Example 1.

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 of such an extension is that the arguments are not in conflict with each other.

Definition 2. *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 $cf(F)$.*

The first concept of extension, we present are 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. *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 1 has a single stable extension $\{a, d\}$. Indeed $\{a, d\}$ is conflict-free, since a and d are not adjacent. Moreover, 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 clearly conflict free. The obvious reason is that e is not defeated by $\{a, c\}$.

Stable semantics in terms of argumentation are considered as quite restricted. It is often sufficient to consider those arguments which are able to defend themselves from external attacks, like the admissible semantics proposed by Dung [4]:

Definition 4. *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 1, we obtain, $adm(F) = \{\emptyset, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}\}$. 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 5. *Let $F = (A, R)$ be an AF. A set S is a* preferred extension *of F, if $S \in adm(F)$ and for each $S' \in adm(F)$, $S \not\subset S'$. We denote the collection of all preferred extensions of F by $pref(F)$.*

Obviously, the preferred extensions of framework F from Example 1 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.

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

Definition 6. *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 1 are $\{a, c\}$, $\{a, d\}$, and $\{a\}$, with the last being also the grounded extensions of F.

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

	$stable$	adm	$pref$	$comp$	$ground$
Cred_e	NP	NP	NP	NP	P
Skept_e	coNP	(trivial)	Π_2^P	P	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 [16], for the remaining results and discussions see [17, 18]). All NP-entries as well as the coNP-entry and the Π_2^P-entry refer to completeness results. 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 observation 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 observation 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, since 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 $\Gamma_F(S) = \{a \in A \mid a \text{ is defended by } S \text{ in } F\}$ (see [4]).

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. Note that having the fixed program π_e at hand, the only translation required for a given AF F is thus its reformulation as input $\widehat{F}$, which is very simple (see below). 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.

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 make 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 7. *Let $\mathcal{S} \subseteq 2^{\mathcal{U}}$ be a collection of sets of domain elements and $\mathcal{I} \subseteq 2^{B_\mathcal{U}}$ 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 (i) for each $I \in \mathcal{I}$, there exists an $S \in \mathcal{S}$, such that $\{a \mid \mathrm{in}(a) \in I\} = S$; and (ii) for each $S \in \mathcal{S}$, there exists an $I \in \mathcal{I}$, such that $\{a \mid \mathrm{in}(a) \in I\} = S$.*

Let us first determine how an AF is presented to our programs as input. In fact, we encode a given AF $F = (A, R)$ as follows

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

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

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

Proposition 1. *For any AF F, $cf(F) \cong \mathcal{AS}(\pi_{cf}(\widehat{F}))$.*

The additional rules for the stability test are as follows:

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

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. This brings us to an encoding for stable extensions, which satisfies the following correspondence result.

Proposition 2. *For any AF F, $stable(F) \cong \mathcal{AS}(\pi_{stable}(\widehat{F}))$.*

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

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

The first rule is the same as in π_{stable}. The second rule derives those arguments which are not defended by the current guess, i.e., those arguments which are defeated by some other argument, which itself is not defeated by the current guess. If such a non-defended argument is contained in the guess, we have to eliminate that guess.

Proposition 3. *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

$$\pi_{comp} = \pi_{adm} \cup \{\text{ :- } \mathrm{out}(X), \mathit{not}\ \mathrm{not\_defended}(X)\}.$$

Proposition 4. *For any AF F, $comp(F) \cong \mathcal{AS}(\pi_{comp}(\widehat{F}))$.*

We now turn to the grounded extension. Suitably encoding the operator Γ_F, we can come up with a stratified program which computes this extension. 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 $\mathrm{in}(\cdot)$-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 we 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

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

Note that π_{ground} is indeed stratified.

Proposition 5. *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, especially 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 now continue with the more involved encoding for preferred extensions. Compared to the one for admissible extensions, this encoding requires 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, we make a second guess using new predicates, say $\mathrm{inN}(\cdot)$ and $\mathrm{outN}(\cdot)$, such that they represent a guess $S' \supset S$. For that guess, we will use disjunction (rather than default negation), which allows that *both* $\mathrm{inN}(a)$ and $\mathrm{outN}(a)$ are contained in a possible answer set (under certain conditions), for each a. In fact, exactly such answer sets will correspond to the preferred extension. The saturation is therefore performed in such a way that all predicates $\mathrm{inN}(a)$ and $\mathrm{outN}(a)$ are derived, for those S' which do *not* characterize an admissible extension. If this saturation succeeds for each $S' \supset S$, 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 $:\!\text{-}\ not\,\mathrm{spoil}$. This ensures that only saturated guesses survive.

Such saturation techniques always requires a restricted use of negation. The predicates defined in $\pi_<$ will serve for this purpose. Two new predicates are needed: predicate $\mathrm{eq}(\cdot)$ which indicates whether a guess S' 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 S'. Both predicates are computed in $\pi_{helpers}$ 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. To this end let

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

$$
\begin{aligned}
\pi_{spoil} = \{\ & \mathrm{inN}(X) \vee \mathrm{outN}(X) :\!\text{-}\ \mathrm{out}(X); && (1)\\
& \mathrm{inN}(X) :\!\text{-}\ \mathrm{in}(X); && (2)\\
& \mathrm{spoil} :\!\text{-}\ \mathrm{eq}; && (3)\\
& \mathrm{spoil} :\!\text{-}\ \mathrm{inN}(X), \mathrm{inN}(Y), \mathrm{defeat}(X,Y); && (4)\\
& \mathrm{spoil} :\!\text{-}\ \mathrm{inN}(X), \mathrm{outN}(Y), \mathrm{defeat}(Y,X), \mathrm{undefeated}(Y); && (5)\\
& \mathrm{inN}(X) :\!\text{-}\ \mathrm{spoil}, \mathrm{arg}(X); && (6)\\
& \mathrm{outN}(X) :\!\text{-}\ \mathrm{spoil}, \mathrm{arg}(X); && (7)\\
& :\!\text{-}\ \mathit{not}\,\mathrm{spoil}\}. && (8)
\end{aligned}
$$

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

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

We define

$$\pi_{pref} = \pi_{adm} \cup \pi_{helpers} \cup \pi_{spoil}.$$

When joined with $\widehat{F}$ for some AF $F = (A, R)$, the rules of π_{spoil} work as follows: (1) and (2) guess a new set $S' \subseteq A$, which compares to the guess $S \subseteq A$ (characterized by predicates $\mathrm{in}(\cdot)$ and $\mathrm{out}(\cdot)$ as used in π_{adm}) as $S \subseteq S'$. In case $S' = S$, we obtain predicate eq and derive predicate spoil (rule (3)). The remaining guesses S' are now handled as follows. First, rule (4) derives predicate spoil if the new guess S' contains a conflict. Second, rule (5) derives spoil if the new guess S' contains an element which is attacked by an argument outside S' which itself is undefeated (by S'). Hence, we derived spoil for those $S \subseteq S'$ where either $S = S'$ or S' 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 (6) and (7). Recall that due to constraint (8) such spoiled interpretation are the only candidates for answer sets. To turn them into an answer set, it is however necessary that we spoiled for *each* S', such that $S \subseteq S'$; but by definition this is exactly the case if S is a preferred extension.

Proposition 6. *For any AF F, $pref(F) \cong \mathcal{AS}(\pi_{pref}(\widehat{F}))$.*

We summarize the results from this section.

Theorem 1. *For any AF F and $e \in \{stable, adm, comp, ground, pref\}$, it holds that $e(F) \cong \mathcal{AS}(\pi_e(\widehat{F}))$.*

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 program involved is π_{pref}, all other are 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 an example consider the Π_2^P-complete problem of *coherence* [17], 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 1. *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.

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) [6] which themselves generalize the preference-based argumentation frameworks [5]. Again we give the definition wrt the universe $\mathcal{U}$.

Definition 8. *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 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 [6] for conflict-free sets, admissible extensions, and preferred extensions are exactly along the lines of Definition 2, 4, and 5, 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 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 [20], 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 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 2. *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 [7] augment basic AFs by a second relation between arguments which indicates supports independent from defeats.

Definition 9. *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.

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

Following [7], we can use this notion of defeat to define conflict-free sets, stable extensions, admissible extensions and preferred extensions[1] exactly along the lines of Definition 2, 3, 4, and 5, respectively.

Theorem 4. *For any BAF F and $e \in \{stable, adm, pref\}$, $e(F) \cong \mathcal{AS}(\pi_{baf} \cup \pi_e(\widehat{F}))$.*

More specific variants of admissible extensions from [7] are obtained by replacing the notion a conflict-free set by other concepts.

Definition 10. *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.

Definition 11. *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

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

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

Definition 12. *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 $S' \in sadm(F)$, $S \not\subseteq S'$. Likewise, a set $S \subseteq A$ is called a* c-preferred extension *of F if $S \in cadm(F)$ and for each $S' \in cadm(F)$, $S \not\subseteq S'$. 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

[1] These extensions are called d-admissible and resp. d-preferred in [7].

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

Theorem 5. *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 [8], 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 9. 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[2]. The encoding (together with some examples) is available on the web and can be run with the answer-set solver DLV [10]. 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.* [21] 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-normal semantics [22] or the ideal semantics [23].

Acknowledgments 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.

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

References

1. Bench-Capon, T.J.M., Dunne, P.E.: Argumentation in artificial intelligence. Artif. Intell. **171** (2007) 619–641
2. Besnard, P., Doutre, S.: Checking the acceptability of a set of arguments. In: Proceedings NMR'04. (2004) 59–64
3. Egly, U., Woltran, S.: Reasoning in argumentation frameworks using quantified boolean formulas. In: Proceedings COMMA'06, IOS Press (2006) 133–144
4. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. **77** (1995) 321–358
5. Amgoud, L., Cayrol, C.: A reasoning model based on the production of acceptable arguments. Ann. Math. Artif. Intell. **34** (2002) 197–215
6. Bench-Capon, T.J.M.: Persuasion in practical argument using value-based argumentation frameworks. J. Log. Comput. **13** (2003) 429–448
7. Cayrol, C., Lagasquie-Schiex, M.C.: On the acceptability of arguments in bipolar argumentation frameworks. In: Proceedings ECSQARU'05. Volume 3571 of LNCS., Springer (2005) 378–389
8. Amgoud, L., Cayrol, C., Lagasquie, M.C., Livet, P.: On bipolarity in argumentation frameworks. International Journal of Intelligent Systems **23** (2008) 1–32
9. Baroni, P., Giacomin, M.: A systematic classification of argumentation frameworks where semantics agree. In: Proceedings COMMA'08, IOS Press (2008) 37–48
10. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM Trans. Comput. Log. **7** (2006) 499–562
11. Niemelä, I.: Logic programming with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. **25** (1999) 241–273
12. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. **9** (1991) 365–386
13. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In: Proceedings LPNMR'07. Volume 4483 of LNCS., Springer (2007) 3–17
14. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive datalog. ACM Trans. Database Syst. **22** (1997) 364–418
15. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Computing Surveys **33** (2001) 374–425
16. Dimopoulos, Y., Torres, A.: Graph theoretical structures in logic programs and default theories. Theor. Comput. Sci. **170** (1996) 209–244
17. Dunne, P.E., Bench-Capon, T.J.M.: Coherence in finite argument systems. Artif. Intell. **141** (2002) 187–203
18. Coste-Marquis, S., Devred, C., Marquis, P.: Symmetric argumentation frameworks. In: Proceedings ECSQARU'05. Volume 3571 of LNCS., Springer (2005) 317–328
19. Eiter, T., Polleres, A.: Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. Theory and Practice of Logic Programming **6** (2006) 23–60
20. Bench-Capon, T.J.M.: Value-based argumentation frameworks. In: Proceedings NMR'02. (2002) 443–454
21. Nieves, J.C., Osorio, M., Cortés, U.: Preferred extensions as stable models. Theory and Practice of Logic Programming **8** (2008) 527–543
22. Caminada, M.: Semi-stable semantics. In: Proceedings COMMA'06, IOS Press (2006) 121–130
23. Dung, P.M., Mancarella, P., Toni, F.: Computing ideal sceptical argumentation. Artif. Intell. **171** (2007) 642–674