Argument and Computation
Vol. 1, No. 2, June 2010, 147–177

Answer-set programming encodings for argumentation frameworks

Uwe Egly, Sarah Alice Gaggl and Stefan Woltran*

Institute of Information Systems 184, Vienna University of Technology, Favoritenstrasse 9-11, A-1040 Vienna, Austria

(*Received 6 August 2009; final version received 8 April 2010*)

Answer-set programming (ASP) has emerged as a declarative programming paradigm where problems are encoded as logic programs, such that the so-called answer sets of theses programs represent the solutions of the encoded problem. The efficiency of the latest ASP solvers reached a state that makes them applicable for problems of practical importance. Consequently, problems from many different areas, including diagnosis, data integration, and graph theory, have been successfully tackled via ASP. In this work, we present such ASP-encodings for problems associated to abstract argumentation frameworks (AFs) and generalisations thereof. Our encodings are formulated as fixed queries, such that the input is the only part depending on the actual AF to process. We illustrate the functioning of this approach, which is underlying a new argumentation system called ASPARTIX in detail and show its adequacy in terms of computational complexity.

Keywords: abstract argumentation frameworks; answer-set programming; implementation

1. Motivation

In Artificial Intelligence (AI), the area of argumentation (the survey by Bench-Capon and Dunne (2007) gives an excellent overview) has become one of the central issues during the last decade. Argumentation provides a formal treatment for reasoning problems arising in a number of applications fields, including Multi-Agent Systems and Law Research. In a nutshell, the so-called abstract argumentation frameworks (AFs) formalise 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 AFs turned out to be a very general principle capturing many other important formalisms from the areas of AI and knowledge representation.

The increasing interest in argumentation led to numerous proposals for formalisations of argumentation. These approaches differ in many aspects. First, there are several ways as to 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. Thus, in order to compare these different proposals, it is desirable to have a system at hand, which is capable of dealing with a large number of argumentation semantics.

Argumentation problems are, in general, intractable; for instance, deciding if an argument is contained in some preferred extension is known to be NP-complete. Therefore, developing dedicated algorithms for the different reasoning problems is non-trivial. A promising way 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.

*Corresponding author. Email: woltran@dbai.tuwien.ac.at

ISSN 1946-2166 print/ISSN 1946-2174 online
© 2010 Taylor & Francis
DOI: 10.1080/19462166.2010.486479
http://www.informaworld.com

In this work, we present an approach which meets these challenges, i.e. we describe a system which, on the one hand, implements numerous approaches for abstract argumentation, and, on the other hand, gains its efficiency by exploiting highly sophisticated solvers to which we can map the problems in question. The declarative programming paradigm of *Answer-Set Programming* (ASP) (Niemelä 1999; Leone et al. 2006) is especially well suited for this purpose due to the following three characteristics.

- The prototypical language of ASP (i.e. logic programming under the answer-set semantics (Gelfond and Lifschitz 1991), also known as stable logic programming or A-Prolog) is very expressible and allows to formulate queries (in an extended datalog fashion) over databases, such that multiple results can be obtained. In our context, queries thus can be employed to obtain multiple extensions for AFs, where the actual AF to process is just given as an input database.
- Advanced ASP-solvers such as Smodels, DLV, GnT, Cmodels, Clasp, or ASSAT are nowadays able to deal with large problem instances, see, e.g. (Gebser et al. 2007). Thus, using our proposed reduction method delegates the burden of optimisation to these systems.
- Depending on the syntactical structure of a given ASP query, the complexity of evaluating that query on input databases (i.e. the data complexity of ASP) varies from classes P, NP, coNP up to Σ_2^P and to Π_2^P. Hence, for the different types of problems in abstract argumentation, we are able to formulate queries which are "well suited" from a complexity point of view. In other words, the complexity of evaluating ASP queries representing some argumentation problem lies in the same complexity class as the original problem.

The main aim of this paper is to present ASP queries for reasoning problems within different types of AFs. To be more specific, we give queries for the most important types of extensions (i.e. admissible, preferred, stable, semi-stable, complete, and grounded (Dung 1995; Caminada 2006)) in terms of Dung's original abstract framework, the preference-based AF by Amgoud and Cayrol (2002), the value-based argumentation framework (VAF) by Bench-Capon (2003), and the bipolar argumentation framework (BAF) (Cayrol and Lagasquie-Schiex 2005; Amgoud, Cayrol, Lagasquie-Schiex, and Livet 2008).

We have implemented these queries in a system called ASPARTIX, which makes use of the prominent answer set solver DLV (Leone et al. 2006). All necessary programs to run ASPARTIX and some illustrating examples are available at

www.dbai.tuwien.ac.at/research/project/argumentation/systempage/

We believe that our system is a useful tool for researchers in the argumentation community 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 (Baroni and Giacomin 2008b).

Earlier work already proposed reductions from argumentation problems to certain target formalisms. Most notably are encodings in terms of (quantified) propositional logic (Besnard and Doutre 2004; Egly and Woltran 2006; Besnard, Hunter, and Woltran 2009) and logic programs (Nieves, Osorio, and Cortés 2008; Nieves, Osorio, and Zepeda 2009; Osorio, Zepeda, Nieves, and Cortés 2005; Wakaki and Nitta 2008; we will come back to the earlier work on reductions to logic programs, which is indeed most closely related to ours, in the discussion section at the end of this article).

The main difference of this earlier work compared with our approach is the necessity of compiling (at least, for some of the semantics) each problem instance into a different instance of the target formalism (e.g. into a different logic program).

In our approach, *all* semantics are encoded within a *fixed* query (independent from the concrete AF to process). Thus, we are more in the tradition of a classical implementation because we construct an interpreter in ASP which takes an AF given as input. 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. This indeed simplifies the answering of questions like "What happens if I add this new argument?".

The remainder of the article is organised as follows. In the next section, we recall the necessary concepts of ASP and AFs. Furthermore we give some illustrative examples and recall the respective complexity results. In Section 3, we gradually introduce the encodings for the particular semantics and in Section 4, we adapt these encodings for some generalisations of AFs. Finally, in Section 5, we give an overview of related approaches and discuss future work.

2. Preliminaries

2.1. *Answer-set programming*

We first give a brief overview of the syntax and semantics of the ASP-formalism we consider, i.e. disjunctive datalog under the answer-set semantics (Gelfond and Lifschitz 1991). Then, we illustrate useful programming techniques on some simple examples. Finally, we briefly recall some important complexity results for disjunctive datalog. We refer to Eiter, Gottlob, and Mannila (1997) and Leone et al. (2006) for a broader exposition on all of these topics.

In what follows, we fix a countable set $\mathcal{U}$ of *(domain) elements*, also called *constants* and suppose a total order $<$ over these elements. An *atom* is an expression $p(t_1, \ldots, t_n)$, where p is a *predicate* symbol 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.

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

$$a_1 \lor \cdots \lor a_n \coloneq b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m$$

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* (or *disjunction-free*) 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. If each rule in a program is normal (resp., ground), we call the program normal (resp., ground). A *fact* is a disjunction-free ground rule with an empty body.

A program is a finite set of safe (disjunctive) rules. Employing database notation, we call a finite set of facts also an *input database* and a set of non-ground rules a *query*. For a query Π and an input database D, we often write $\Pi(D)$, instead of the program $D \cup \Pi$, in order to indicate that D serves as input for query Π.

A normal program Π is called *stratified* if no atom a depends by recursion through negation on itself (Apt, Blair, and Walker 1988). More formally, Π is stratified if there exists an assignment $\alpha(\cdot)$ of integers to the predicates in Π, such that for each rule $r \in \Pi$, the following holds: If predicate p occurs in the head of r and predicate q occurs

(i) in the positive body of r, then $\alpha(p) \geq \alpha(q)$ holds;
(ii) in the negative body of r, then $\alpha(p) > \alpha(q)$ holds.

As an example, consider the following program Π:

$$\Pi = \{\, a(X) :\!-\, \mathrm{not}\, b(X), d(X);\;\; b(X) :\!-\, a(X) \,\}.$$

In order to find an assignment $\alpha(\cdot)$ satisfying the above conditions for Π, observe that the first rule of Π requires $\alpha(a) > \alpha(b)$, but the second rule, in turn, forces $\alpha(b) \geq \alpha(a)$. In other words, each assignment $\alpha(\cdot)$ violates at least one of the conditions, and hence, Π is not stratified.

For the program

$$\Pi' = \{\, a(X) :\!-\, \mathrm{not}\, b(X), d(X);\;\; b(X) :\!-\, c(X);\;\; c(X) :\!-\, b(X) \,\},$$

we can use the assignment $\alpha(a) = 2, \alpha(b) = \alpha(c) = \alpha(d) = 1$ to show that Π' is stratified. The concept of stratified programs is very important in logic programming, since it allows for a restricted form of negation, but does not lead to an increase in the complexity (see also the complexity results below, which show that stratified programs still can be evaluated efficiently, while this is not the case for normal or disjunctive programs).

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

Let the set of all ground atoms over $\mathcal{U}$ be denoted by $B_\mathcal{U}$. 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 $B^-(r) \cap I = \emptyset$. A ground program Π is satisfied by I, if I satisfies each $r \in \Pi$. A non-ground rule r (resp., a non-ground program Π) is satisfied by I, if I satisfies all groundings of r (resp., $Gr(\Pi)$). An interpretation $I \subseteq B_\mathcal{U}$ is an *answer set* of Π iff it is a subset-minimal set satisfying the *Gelfond–Lifschitz reduct*

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

For a program Π, we denote the set of its answer sets by $\mathcal{AS}(\Pi)$. We note that for each $I \in \mathcal{AS}(\Pi)$, $I \subseteq \mathcal{B}_\Pi$ holds. Moreover, a program can have multiple answer sets. A stratified program has at most one answer set, and a constraint-free stratified program has exactly one answer set.

Let us discuss some typical concepts of ASP which are important for the implementation later on by providing queries for some graph problems (see Leone et al. (2006) for a more detailed discussion of similar examples). First, consider the problem of reachability in directed graphs. We assume that an input graph G is given as a set of facts $\mathrm{edge}(a, b)$, indicating that there is an edge from vertex a to vertex b in G. Moreover, we assume that each such input database D also contains a fact $\mathrm{start}(a)$ for a designated vertex a. The goal is now to compute all vertices reachable from a via the edges in G. Therefore, we use the predicate $\mathrm{reached}(\cdot)$ in the following program:

$$\begin{aligned}\pi_{\mathrm{reach}} = \{&\mathrm{reached}(X) :\!-\, \mathrm{start}(X);\\ &\quad \mathrm{reached}(Y) :\!-\, \mathrm{reached}(X), \mathrm{edge}(X, Y)\}\end{aligned}$$

Given an input database D as specified above, we get that predicate $\mathrm{reached}(b)$ is in the answer set of $\pi_{\mathrm{reach}}(D)$ iff vertex b is reachable in G from the designated start vertex a. Note that π_{reach} does neither contain disjunction nor negation. Moreover, there is no constraint in π_{reach}, hence $\pi_{\mathrm{reach}}(D)$ has exactly one answer set, for each input database D. Minimality of the answer-set semantics guarantees that the answer set of $\pi_{\mathrm{reach}}(D)$ does not contain predicates $\mathrm{reached}(c)$, if vertex c is not reachable from a in G.

Before we give some further examples, let us introduce the concept of splitting sets (Lifschitz and Turner 1994). Given a program Π, a set S of predicate symbols is a *splitting set* for Π, iff, for every rule $r \in \Pi$, it holds that if some atom with predicate symbol from S occurs in the head of r, then each atom in r has its predicate symbol from S as well. Any splitting set S for program Π divides Π in two parts. The *top* of Π (with respect to S), in symbols Π^t_S, contains all rules of Π which have an occurrence of a predicate symbol *not* contained in S, while the *bottom* of Π (with respect to S) is defined as $\Pi^b_S = \Pi \setminus \Pi^t_S$. To have an example, recall the program $\Pi' = \{a(X) :- \mathrm{not}\, b(X), d(X);\ b(X) :- c(X);\ c(X) :- b(X)\}$. The set $S = \{b, c\}$ is a splitting set for Π', and we obtain $(\Pi')^t_S = \{a(X) :- \mathrm{not}\, b(X), d(X)\}$ and $(\Pi')^b_S = \{b(X) :- c(X);\ c(X) :- b(X)\}$. Splitting sets allow to compute the answer sets of a program step-by-step due to the following result, known as the Splitting Theorem.

Proposition 2.1 *Let S be a splitting set of a program Π, and let $I \subseteq B_{\mathcal{U}}$. Then $I \in \mathcal{AS}(\Pi)$ iff $I \in \mathcal{AS}(\Pi^t_S(J))$, where $J = I \cap B_{\Pi^b_S}$ and $J \in \mathcal{AS}(\Pi^b_S)$.*

We next illustrate a typical use of default negation. We consider here the problem of 3-colourability of an (undirected) graph. Suppose a graph's vertices are defined via the predicate vertex$(\cdot)$ and its edges via the predicate edge$(\cdot,\cdot)$. We employ default negation to guess a colour for each node in the graph, and then check whether adjacent vertices have indeed different colours.

$$\pi_{\mathrm{col}} = \pi_{\mathrm{guess}} \cup \pi_{\mathrm{check}} \quad \text{where}$$

$$\pi_{\mathrm{guess}} = \{\mathrm{red}(X) :- \mathrm{vertex}(X), \mathrm{not\,green}(X), \mathrm{not\,blue}(X); \tag{1}$$

$$\mathrm{green}(X) :- \mathrm{vertex}(X), \mathrm{not\,red}(X), \mathrm{not\,blue}(X); \tag{2}$$

$$\mathrm{blue}(X) :- \mathrm{vertex}(X), \mathrm{not\,red}(X), \mathrm{not\,green}(X)\} \tag{3}$$

$$\pi_{\mathrm{check}} = \{\mathrm{fail} :- \mathrm{red}(X), \mathrm{red}(Y), \mathrm{edge}(X,Y); \tag{4}$$

$$\mathrm{fail} :- \mathrm{green}(X), \mathrm{green}(Y), \mathrm{edge}(X,Y); \tag{5}$$

$$\mathrm{fail} :- \mathrm{blue}(X), \mathrm{blue}(Y), \mathrm{edge}(X,Y); \tag{6}$$

$$:- \mathrm{fail}\} \tag{7}$$

The three rules of π_{guess} are typical "guessing rules" which assign to each vertex exactly one colour. To illustrate this, let D be an input database over predicates vertex and edge, and assume D contains three vertices {vertex(a), vertex(b), vertex(c)}. Then, the program $\pi_{\mathrm{guess}}(D)$ possesses 27 answer sets, namely

$$D \cup \{\mathrm{red}(a), \mathrm{red}(b), \mathrm{red}(c)\}, D \cup \{\mathrm{red}(a), \mathrm{red}(b), \mathrm{green}(c)\}, \ldots$$

We note that the fact that no vertex gets assigned to more than one colour is ensured by the interplay between minimality and the notion of a reduct. Also observe that π_{guess} is not stratified. Indeed, we would require a mapping $\alpha(\cdot)$, such that $\alpha(\mathrm{red}) > \alpha(\mathrm{green})$ (because of rule (1)) and $\alpha(\mathrm{green}) > \alpha(\mathrm{red})$ (because of rule (2)). Such a mapping obviously does not exist.

Now let use consider π_{check}. The role of this program is to exclude answer set candidates provided by π_{guess}. In our case, we want to rule out candidates having adjacent vertices of the same colour. Thanks to the Splitting Theorem, we indeed are allowed to first compute the answer sets of $\pi_{\mathrm{guess}}(D)$ and then compute the answer sets of $\pi_{\mathrm{check}}(J)$ for each $J \in \mathcal{AS}(\pi_{\mathrm{guess}}(D))$, in order to obtain the answer sets of the entire program $\pi_{\mathrm{col}}(D)$. In fact, the set $S = \{\mathrm{vertex}, \mathrm{edge}, \mathrm{red}, \mathrm{green}, \mathrm{blue}\}$ is a splitting set for $\pi_{\mathrm{col}}(D)$, for any input database D over atoms with predicate symbols from S. The top of $\pi_{\mathrm{col}}(D)$ with respect to S is thus given by $(\pi_{\mathrm{col}}(D))^t_S = \pi_{\mathrm{check}}$, and the bottom of $\pi_{\mathrm{col}}(D)$ with respect to S is given by $(\pi_{\mathrm{col}}(D))^b_S = \pi_{\mathrm{guess}}(D)$. For illustration, assume D contains $\mathrm{edge}(a,b)$, $\mathrm{edge}(b,c)$, and $\mathrm{edge}(c,a)$. Assume the answer set $J = D \cup \{\mathrm{red}(a), \mathrm{red}(b), \mathrm{green}(c)\}$ of $\pi_{\mathrm{guess}}(D)$ is used as input now to π_{check}. Then, the rule fail $:-$ $\mathrm{red}(X), \mathrm{red}(Y), \mathrm{edge}(X,Y)$ derives fail, but then rule (7) cannot be satisfied. Thus, J does not satisfy $\pi_{\mathrm{col}}(D)$, and consequently, J cannot become an answer set of $\pi_{\mathrm{col}}(D)$. On the other hand, $D \cup \{\mathrm{red}(a), \mathrm{green}(b), \mathrm{blue}(c)\}$ (which is also an answer set of $\pi_{\mathrm{guess}}(D)$) satisfies all rules of π_{check} and thus becomes an answer set of $\pi_{\mathrm{col}}(D)$. In fact, for the example graph represented in D, $\pi_{\mathrm{col}}(D)$ possesses six answer sets.

Actually, instead of the rules (1)–(3), we could have used disjunction to guess a colour for each vertex. As we shall see next, disjunction is a more expressive concept than default negation. While with default negation, one is able to formulate an exclusive guess (as we did in the above example), disjunction can be additionally employed for a certain saturation technique, which allows for representing even more complex problems. The term “saturation” indicates that all atoms which are subject to a guess can also be jointly contained in an interpretation. To saturate a guess, it is however necessary that the checking part of a program interacts with the guessing part.

As a very simple example, let us compare the following two programs:

$$\begin{array}{ll} \Pi = \{p \vee q :- ; & \Pi' = \{p :- \mathrm{not}\, q;\;\; q :- \mathrm{not}\, p; \\ \qquad p :- q;\;\; q :- p\} & \qquad p :- q;\;\; q :- p\} \end{array}$$

In Π we are guessing p or q via disjunction and in Π' we have formulated the same guess via two normal rules. Note that the remaining rules $p :- q$ and $q :- p$ however saturate any such guess between p and q by adding the respective other predicate. In fact, this saturation of the guess shows the difference between programs Π and Π' in the following sense: We have that $\{p,q\}$ is the unique answer set of Π (which can be seen by the fact that $\{p,q\}$ is the minimal interpretation satisfying Π; note that Π does not contain default negation, so there is no need to consider the reduct.). On the other hand, for Π' we obtain as the reduct $(\Pi')^{\{p,q\}} = \{p :- q;\;\; q :- p\}$. But now, $\{p,q\}$ is not the minimal interpretation satisfying $(\Pi')^{\{p,q\}}$, since we could also take the empty interpretation (i.e., both p and q are set to false). Thus, $\{p,q\}$ is not an answer set of Π'. In fact, Π' has no answer set at all.

To have a slightly more meaningful application of this saturation technique, let us now reduce also the non-3-colourability problem to the problem of deciding whether an answer set exists. Therefore, we adapt the encoding π_{col} from above as follows. First, we use disjunction in rule (8) to guess the colour of vertices. Second, we have the same rules (9)–(11) to derive fail as we had in π_{col}. Finally, instead of the constraint $:-$ fail to exclude invalid colourings, we “saturate” the guess via rules (12)–(14) and add as constraint $:-$ not fail.

$$\begin{aligned} \pi_{\mathrm{ncol}} = \{&\mathrm{red}(X) \vee \mathrm{green}(X) \vee \mathrm{blue}(X) :- \mathrm{vertex}(X); && (8)\\ &\mathrm{fail} :- \mathrm{red}(X), \mathrm{red}(Y), \mathrm{edge}(X,Y); && (9)\\ &\mathrm{fail} :- \mathrm{green}(X), \mathrm{green}(Y), \mathrm{edge}(X,Y); && (10)\\ &\mathrm{fail} :- \mathrm{blue}(X), \mathrm{blue}(Y), \mathrm{edge}(X,Y); && (11)\\ &\mathrm{red}(X) :- \mathrm{fail}, \mathrm{vertex}(X); && (12)\\ &\mathrm{green}(X) :- \mathrm{fail}, \mathrm{vertex}(X); && (13)\\ &\mathrm{blue}(X) :- \mathrm{fail}, \mathrm{vertex}(X); && (14)\\ &:- \mathrm{not\ fail}\} && (15) \end{aligned}$$

Suppose graphs are represented by inputs D as above. The only possible answer set J of $\pi_{\mathrm{ncol}}(D)$ (if one exists) contains fail because of the last rule $:-$ not fail. But if fail is contained in J, J has to contain red(X), green(X), and blue(X) for each vertex X. Thus J has to be of the form $D \cup \{\text{fail}\} \cup \{\text{red}(a), \text{green}(a), \text{blue}(a) \mid \text{vertex}(a) \in D\}$. We observe that the reduct $(\pi_{\text{ncol}}(D))^J$ is given by $\pi_{\text{ncol}}(D) \setminus \{:- \text{not fail}\}$. Note that interpretations satisfying $(\pi_{\text{ncol}}(D))^J$ do not necessarily contain fail. Indeed, we can come up with such an interpretation (not containing fail), exactly in the case where the input D represents a 3-colourable graph. But this means that D represents a graph which is *not* 3-colourable iff J is (the unique) answer set of $\pi_{\text{ncol}}(D)$. We refer to the work by Eiter and Polleres (2006) for a general discussion on how this saturation technique can be further exploited in ASP queries.

We conclude this section by recalling some central complexity results for ASP. Credulous and skeptical reasoning in terms of programs are defined as follows. Given a program Π and a set A of ground atoms, we denote by $\Pi \models_c A$ that A is contained in some answer sets of Π. Likewise, we denote by $\Pi \models_s A$ that A is contained in all answer sets of Π. In the former case, we reason *credulously*; in the latter case, we reason *skeptically*.

Since we will deal with fixed programs, we focus on results for data complexity. Recall that data complexity in our context addresses the problem $\Pi(D) \models A$ where the query Π is fixed, while the input database D and ground atoms A are inputs of the decision problem. Depending on the concrete definition of $\models$, we get the complexity results in Table 1, compiled from (Dantsin, Eiter, Gottlob, and Voronkov 2001) and the references therein.

2.2. *Basic argumentation frameworks*

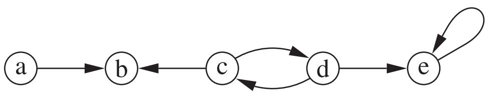
In this section, we recall the most important semantics for the basic version of abstract AFs and highlight complexity results for typical decision problems. Later, in Section 4, we will introduce some generalisations of AFs.

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

Definition 2.2 *An AF is a pair $F = (A, R)$ where $A \subseteq \mathcal{U}$ is a finite 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 AF can be naturally represented as a directed graph.

Example 2.3 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.



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 the same extension.

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

	Stratified programs	Normal programs	General case
$\models_c$	P	NP	Σ_2^P
$\models_s$	P	coNP	Π_2^P

Definition 2.4 *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* $\mathrm{cf}(F)$.

For our framework $F = (A, R)$ from Example 2.3, we have

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

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

Definition 2.5 *Let $F = (A, R)$ be an AF. A set S is a stable extension of F, if $S \in \mathrm{cf}(F)$ and each $a \in A \setminus S$ is defeated by S in F. We denote the collection of all stable extensions of F by* $\mathrm{stable}(F)$.

The framework F from Example 2.3 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 $\mathrm{stable}(F)$, 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 (1995).

Definition 2.6 *Let $F = (A, R)$ be an AF. A set S is an admissible extension of F (or S is admissible in F), if $S \in \mathrm{cf}(F)$ and each $a \in S$ is defended by S in F. We denote the collection of all admissible extensions of F by* $\mathrm{adm}(F)$.

For the framework F from Example 2.3, 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, skeptical acceptance (i.e. given an AF $F = (A, R)$, and $a \in A$, is a contained in all extensions of F) becomes trivial w.r.t. admissible extensions. Thus, many researchers consider maximal (w.r.t. set-inclusion) admissible sets, called *preferred extensions*, as more important.

Definition 2.7 *Let $F = (A, R)$ be an AF. A set S is a preferred extension of F, if $S \in \mathrm{adm}(F)$ and for each $T \in \mathrm{adm}(F)$, $S \not\subset T$. We denote the collection of all preferred extensions of F by* $\mathrm{pref}(F)$.

Obviously, the preferred extensions of the framework F from Example 2.3 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 (2006) and investigated also in (Dunne and Caminada 2008). Semi-stable semantics are located in-between stable and preferred semantics, in the sense that each stable extension of an AF 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 (in case of finite AFs). We use the definition given by Dunne and Caminada (2008).

DEFINITION 2.8 *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 $S \in \mathrm{adm}(F)$ and for each $T \in \mathrm{adm}(F)$, $S_R^+ \not\subset T_R^+$. We denote the collection of all semi-stable extensions of F by* $\mathrm{semi}(F)$.

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 2.9 *Let $F = (A, R)$ be an AF. A set S is a complete extension of F, if $S \in \mathrm{adm}(F)$ and, for each $a \in A$ defended by S in F, $a \in S$ holds. The least (w.r.t. 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* $\mathrm{comp}(F)$ *(resp.,* $\mathrm{ground}(F)$*).*

The complete extensions of framework F from Example 2.3 are $\{a, c\}$, $\{a, d\}$, and $\{a\}$, with the last being also the grounded extension of F.

This concludes our collection of argumentation semantics that we consider in this paper. The relations between the semantics are depicted in Figure 1, where an arrow from e to f indicates that each e-extension is also an f-extension.

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

- 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 all $S \in e(F)$?

The complexity results are depicted in Table 2 (many of them follow implicitly from Dimopoulos and Torres (1996); for the remaining results and discussions, see (Dunne and Bench-Capon 2002, 2004; Coste-Marquis, Devred, and Marquis 2005; Dunne and Caminada 2008; Dvořák and Woltran 2009)). 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

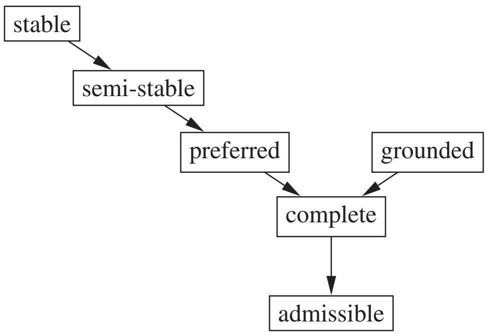


Figure 1. Overview of argumentation semantics and their relations.

Table 2. Complexity for decision problems in AFs.

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

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 with credulous reasoning over admissible extensions: a is contained in some $S \in adm(F)$ iff a is contained in some $S \in \mathrm{pref}(F)$. 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 (Dung 1995), since the least fixed point of the following operator captures (for finite AFs) the grounded extension and can be computed in polynomial time.

PROPOSITION 2.10 *The grounded extension of an AF $F = (A, R)$ is given by the least fixed point of Γ_F, where the operator $\Gamma_F : 2^A \to 2^A$ is defined as*

$$\Gamma_F(S) = \{a \in A \mid a \text{ is defended by } S \text{ in } F\}.$$

3. ASP-encodings for abstract argumentation frameworks

We now provide fixed queries π_e for each extension of type e introduced in the previous section, in such a way that the AF F is given as an input database $\hat{F}$ and the answer sets of the combined program $\pi_e(\hat{F})$ are in a certain one-to-one correspondence with the respective extensions. Note that having established the fixed query π_e, the only translation required is to provide a given AF F as input database $\hat{F}$ to π_e. For an AF $F = (A, R)$, we define

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

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. As we have outlined in Section 2.1, default negation is an appropriate concept to formulate such a guess within a query. In what follows, we use unary predicates $\text{in}(\cdot)$ and $\text{out}(\cdot)$ to perform a guess for a set $S \subseteq A$, where $\text{in}(a)$ represents that $a \in S$. The following notion of correspondence is relevant for our purposes.

DEFINITION 3.1 *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 (i) for each $S \in \mathcal{S}$, there exists an $I \in \mathcal{I}$, such that $\{a \mid \text{in}(a) \in I\} = S$; and (ii) for each $I \in \mathcal{I}$, it holds that $\{a \mid \text{in}(a) \in I\} \in \mathcal{S}$.*

Note that $\mathcal{S} \cong \mathcal{I}$ thus implies $|\mathcal{S}| = |\mathcal{I}|$. In what follows, we will stepwise introduce the rules from which our queries will be built.

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

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

For our framework F from Example 2.3, we have as input

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

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

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

where

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

PROPOSITION 3.2 *For any AF F,* $\mathrm{cf}(F) \cong \mathcal{AS}(\pi_{\mathrm{cf}}(\hat{F}))$.

Proof Let $F = (A, R)$ and $S \in cf(F)$. We show that the interpretation

$$I = \hat{F} \cup \{\mathrm{in}(a) \mid a \in S\} \cup \{\mathrm{out}(a) \mid a \in A \setminus S\}$$

(corresponding to S) is an answer set of $\pi_{\mathrm{cf}}(\hat{F})$. We have to show that I satisfies $\pi_{\mathrm{cf}}(\hat{F})$ (we recall that an interpretation I satisfies a program Π iff I satisfies Π^I), and that no $J \subset I$ satisfies $(\pi_{\mathrm{cf}}(\hat{F}))^I$. To show that I satisfies $\pi_{\mathrm{cf}}(\hat{F})$, we have to show that I satisfies each $r \in \mathrm{Gr}(\pi_{\mathrm{cf}}(\hat{F}))$. Note that

$$\begin{aligned}\mathrm{Gr}(\pi_{\mathrm{cf}}(\hat{F})) = \hat{F} \cup&\\ &\{\mathrm{in}(a) \text{ :– } \mathrm{not}\,\mathrm{out}(a), \mathrm{arg}(a) \mid a \in A\} \cup &(16)\\ &\{\mathrm{out}(a) \text{ :– } \mathrm{not}\,\mathrm{in}(a), \mathrm{arg}(a) \mid a \in A\} \cup &(17)\\ &\{\text{:– } \mathrm{in}(a), \mathrm{in}(b), \mathrm{defeat}(a, b) \mid a, b \in A\}. &(18)\end{aligned}$$

Clearly, I satisfies $\hat{F}$, since $\hat{F} \subseteq I$. Moreover, I satisfies each instance of rule (16) (and likewise of rule (17)), since either $\mathrm{in}(a)$ or $\mathrm{out}(a)$ is contained in I, for each $a \in A$. Finally, each instance of constraint (18) is satisfied by I, since $B^+(r) \not\subseteq I$. This can be seen as follows: Suppose r is of the form $:\!-\ \mathrm{in}(a), \mathrm{in}(b), \mathrm{defeat}(a,b)$. In case, $\{\mathrm{in}(a), \mathrm{in}(b)\} \not\subseteq I$, $B^+(r) \not\subseteq I$ is clear. Otherwise, then both a and b are contained in S, by definition of I. Since $S \in \mathrm{cf}(F)$ by assumption, $(a,b) \notin R$ has to hold. Thus, by definition, $\mathrm{defeat}(a,b) \notin \hat{F}$ (and thus $\mathrm{defeat}(a,b) \notin I$). Hence $B^+(r) \not\subseteq I$.

This shows that I satisfies $\pi_{\mathrm{cf}}(\hat{F})$. Now let $J \subseteq I$, such that J satisfies $(\pi_{\mathrm{cf}}(\hat{F}))^I$. We show that then $J = I$. Note that the reduct is always grounded by definition. We have

$$
\begin{aligned}
(\pi_{\mathrm{cf}}(\hat{F}))^I = \hat{F} \cup & \\
& \{\mathrm{in}(a) :\!-\ \mathrm{arg}(a) \mid a \in A, \mathrm{out}(a) \notin I\} \cup & (19)\\
& \{\mathrm{out}(a) :\!-\ \mathrm{arg}(a) \mid a \in A, \mathrm{in}(a) \notin I\} \cup & (20)\\
& \{:\!-\ \mathrm{in}(a), \mathrm{in}(b), \mathrm{defeat}(a,b) \mid a, b \in A\}. & (21)
\end{aligned}
$$

Since J satisfies $(\pi_{\mathrm{cf}}(\hat{F}))^I$, $\hat{F} \subseteq J$. Moreover, since J satisfies each rule in Equation (19), for each $a \in A$, such that $\mathrm{out}(a) \notin I$, $\mathrm{in}(a)$ has to be in J. But since, for each $a \in A$, we have that $\mathrm{out}(a) \notin I$ iff $\mathrm{in}(a) \in I$, we get that $\mathrm{in}(a) \in J$ iff $\mathrm{in}(a) \in I$. Similarly, using the rules from Equation (20), we obtain that $\mathrm{out}(a) \in J$ iff $\mathrm{out}(a) \in I$. $J = I$ follows. Hence, there is no $J \subset I$, such that J satisfies $(\pi_{\mathrm{cf}}(\hat{F}))^I$. This concludes the proof for $I \in \mathcal{AS}(\pi_{\mathrm{cf}}(\hat{F}))$.

Now suppose $I \in \mathcal{AS}(\pi_{\mathrm{cf}}(\hat{F}))$. Consequently, $\hat{F} \subseteq I$, and no further atom of the form $\mathrm{defeat}(\cdot,\cdot)$ or $\mathrm{arg}(\cdot)$ is contained in I (by the minimality of answer sets, and since such predicates do not occur in heads of rules apart from $\hat{F}$). We show that $S \in \mathrm{cf}(F)$ for $S = \{a \mid \mathrm{in}(a) \in I\}$. Since $I \in \mathcal{AS}(\pi_{\mathrm{cf}}(\hat{F}))$, I satisfies $\pi_{\mathrm{cf}}(\hat{F})$, i.e. I satisfies the grounding of $\pi_{\mathrm{cf}}(\hat{F})$, $\mathrm{Gr}(\pi_{\mathrm{cf}}(\hat{F}))$, as given above. In particular, I satisfies each constraint in Equation (18), thus jointly $\mathrm{in}(a) \in I$ and $\mathrm{in}(b) \in I$, only if $\mathrm{defeat}(a,b) \notin I$. By our observation from above, $\mathrm{defeat}(a,b) \notin \hat{F}$, and thus $(a,b) \notin R$. Thus, for each a, b jointly occurring in S, a and b do not attack each other in F. Thus S is conflict-free for F. ■

3.1. *Stable extensions*

We are now prepared to present our first encoding. Two additional rules for the stability test are required and we define

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

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

Note that we can use the Splitting Theorem (Section 2.1) as follows. The splitting set $C = \{\mathrm{in}, \mathrm{out}, \mathrm{arg}, \mathrm{defeat}\}$ divides the program $\pi_{\mathrm{stable}}(\hat{F})$ into $(\pi_{\mathrm{stable}}(\hat{F}))^t_C = \pi_{\mathrm{srules}}$ and $(\pi_{\mathrm{stable}}(\hat{F}))^b_C = \pi_{\mathrm{cf}}(\hat{F})$. Therefore, we make direct use of the answer sets of $\pi_{\mathrm{cf}}(\hat{F})$. For our example, let us first consider the collection $\mathcal{C}$ of answer sets of

$$\pi_{\mathrm{cf}}(\hat{F}) \cup \{\mathrm{defeated}(X) :\!-\ \mathrm{in}(Y), \mathrm{defeat}(Y,X)\}.$$

In fact, using our calculations from the previous subsection, we obtain

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

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

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

Proposition 3.3 *For any AF F, stable$(F) \cong \mathcal{AS}(\pi_{\text{stable}}(\hat{F}))$.*

Proof First, let us formally describe the application of the Splitting Theorem (Proposition 2.1) as sketched above with the splitting set $C = \{\text{in}, \text{out}, \text{arg}, \text{defeat}\}$. Then, we obtain

$$\mathcal{AS}(\pi_{\text{stable}}(\hat{F})) = \bigcup_{J \in \mathcal{AS}(\pi_{\text{cf}}(\hat{F}))} \mathcal{AS}(J \cup \pi_{\text{srules}}). \tag{22}$$

Let $F = (A, R)$ and $S \in \text{stable}(F)$. Since $S \in \text{stable}(F)$, $S \in \text{cf}(F)$, and by Proposition 3.2, there exists an interpretation

$$J = \big(\hat{F} \cup \{\text{in}(a) \mid a \in S\} \cup \{\ \text{out}(a) \mid a \in A \setminus S\}\big) \in \mathcal{AS}(\pi_{\text{cf}}(\hat{F})).$$

By Equation (22), we know that each answer set of $J \cup \pi_{\text{srules}}$ is also an answer set of $\pi_{\text{stable}}(\hat{F})$. We now show that I given as

$$\hat{F} \cup \{\ \text{in}(a) \mid a \in S\ \} \cup \{\ \text{out}(a) \mid a \in A \setminus S\} \cup \{\text{defeated}(a) \mid b \in S, (b, a) \in R\ \}$$

is indeed an answer set of $\pi_{\text{stable}}(\hat{F})$. We first show that I satisfies the grounding of $J \cup \pi_{\text{srules}}$. Clearly, I satisfies J, since $J \subseteq I$. The groundings of the remaining rules are given by

$$\{\text{defeated}(a) :\!-\ \text{in}(b), \text{defeat}(b, a) \mid a, b \in A\} \tag{23}$$

$$\{:\!-\ \text{out}(a), \text{not defeated}(a) \mid a \in A\}. \tag{24}$$

Note that, for each argument b, we have $\text{in}(b) \in I$ iff $b \in S$. Moreover, we already know $\text{defeat}(a, b) \in J$ iff $(a, b) \in R$. Thus, I satisfies each instance of rule (23). Further, since S is stable, we know that each $a \in A \setminus S$ is defeated by S in F. By definition, if $a \in A \setminus S$ then $\text{out}(a) \in I$. But this shows that either $\text{out}(a) \notin I$ or $\text{defeated}(a) \in I$ holds for each $a \in A$. Thus I satisfies each instance of constraint (24). This shows that I satisfies $\pi_{\text{stable}}(\hat{F})$. In order to show that $I \in \mathcal{AS}(\pi_{\text{stable}}(\hat{F}))$, let $K \subseteq I$ be an interpretation satisfying $(\pi_{\text{stable}}(\hat{F}))^I$. First, we know $K \cap B_{\pi_{\text{cf}}(\hat{F})} = I \cap B_{\pi_{\text{cf}}(\hat{F})}$, otherwise $J \notin \mathcal{AS}(\pi_{\text{cf}}(\hat{F}))$ (in that case we would have

$K \cap B_{\pi_{\mathrm{cf}}(\hat{F})} \subset I \cap B_{\pi_{\mathrm{cf}}(\hat{F})}$, but then also $K \cap B_{\pi_{\mathrm{cf}}(\hat{F})} \subset J$ holds since $J = I \cap B_{\pi_{\mathrm{cf}}(\hat{F})}$; further, by assumption, K (and thus also $K \cap B_{\pi_{\mathrm{cf}}(\hat{F})}$) satisfies $(\pi_{\mathrm{cf}}(\hat{F}))^J = (\pi_{\mathrm{cf}}(\hat{F}))^I \subseteq (\pi_{\mathrm{stable}}(\hat{F}))^I)$. Second, since K satisfies the rules in Equation (23) – which are all without default negation and thus contained in the reduct – $\mathrm{defeated}(a) \in K$ iff $\mathrm{defeated}(a) \in I$. Hence $K = I$. This shows $I \in \mathcal{AS}(\pi_{\mathrm{stable}}(\hat{F}))$.

Let now $I \in \mathcal{AS}(\pi_{\mathrm{stable}}(\hat{F}))$. We show that $S = \{a \mid \mathrm{in}(a) \in I\}$ is a stable extension of F. Since $I \in \mathcal{AS}(\pi_{\mathrm{stable}}(\hat{F}))$ there exists (because of relation (22)) a $J \in \mathcal{AS}(\pi_{\mathrm{cf}}(\hat{F}))$ such that $I \in \mathcal{AS}(J \cup \pi_{\mathrm{srules}})$. It is not hard to see that $S = \{a \mid \mathrm{in}(a) \in J\}$. By Proposition 3.2, S is conflict-free in F. Since $I \in \mathcal{AS}(J \cup \pi_{\mathrm{srules}})$, we get (by analogous observations as above) that $\mathrm{defeated}(a) \in I$ iff a is defeated by S in F. Now, since I satisfies $\pi_{\mathrm{stable}}(\hat{F}))$, I has to satisfy each instance of constraint (24). In particular, we obtain that for each a such that $\mathrm{out}(a) \in I$, also $\mathrm{defeated}(a) \in I$. But this yields that each $a \notin S$ is defeated by S in F. Thus $S \in \mathrm{stable}(F)$. ■

3.2. *Admissible extensions*

We next give the rules for the admissibility test:

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

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 start from a collections $\mathcal{C}$ of sets as above but now we check which sets violate the new constraint $:- \mathrm{in}(X), \mathrm{defeat}(Y,X), \mathrm{not\ defeated}(Y)$. 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 analogous.

Hence, we obtain

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

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

PROPOSITION 3.4 *For any AF F,* $\mathrm{adm}(F) \cong \mathcal{AS}(\pi_{\mathrm{adm}}(\hat{F}))$.

The proof is similar to the one of Proposition 3.3.

3.3. *Complete extensions*

We proceed with the encoding for complete extensions. We define

$$\pi_{\text{comp}} = \pi_{\text{adm}} \cup \{\text{undefended}(X) :- \text{defeat}(Y, X), \text{not defeated}(Y);$$
$$:- \text{out}(X), \text{not undefended}(X)\}.$$

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

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

Obviously, each candidate which contains out(a) (i.e. the sets $S_\emptyset$, S_c, and S_d) is ruled out by the constraint $:- \text{out}(X), \text{not undefended}(X)$, since no such candidate set contains undefended(a). One can check that all other sets, i.e., S_a, S_{ac}, and S_{ad}, do not violate the constraint, and thus are answer sets of $\pi_{\text{comp}}(\hat{F})$. Again, these three sets characterise the complete extensions of F, as desired.

Proposition 3.5 *For any AF F,* $\text{comp}(F) \cong \mathcal{AS}(\pi_{\text{comp}}(\hat{F}))$.

The proof is similar to the one of Proposition 3.3.

3.4. *Grounded extension*

Computing the grounded extension of a given AF could also be done by the “Guess & Check” method we used in the previous encodings. But since reasoning via the grounded extension is tractable (Table 2), we aim to find an encoding in a tractable subclass of datalog (the encodings we used so far are not contained in such a tractable subclass). Inspecting Table 1 shows that *stratified* programs are the appropriate candidate. However, as discussed in Section 2.1, stratified negation is too weak to formalise guesses as we did in the encodings above. Instead, a suitable encoding of the operator Γ_F as introduced in Proposition 2.10 is possible. Therefore, we recursively derive in$(\cdot)$ predicates according to the definition of the operator Γ_F (following the same idea as we used to derive the reached$(\cdot)$ predicates in the example program for reachability in graphs, discussed in Section 2.1). To compute (in a stratified program) the required predicate for being defended, we have to use the order $<$ over the domain elements (such an order is provided by all ASP-solvers) and derive corresponding predicates for infimum, supremum, and successor w.r.t. $<$. In fact, we shall use these predicates to perform “loops” over all arguments given by the input database.

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

Note that $\pi_{<}$ is indeed stratified and constraint-free. Hence, $\pi_{<}(\hat{F})$ yields exactly one answer set for each AF F. To illustrate the purpose of $\pi_{<}$, recall our example framework F, and assume the arguments are ordered as follows: $a < b < c < d < e$. For this particular order, the single answer set S_0 of $\pi_{<}(\hat{F})$ contains

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

The other atoms (which however will not be used in later calculations) contained in S_0 are $\mathrm{lt}(a, b)$, $\mathrm{lt}(a, c)$, $\mathrm{lt}(a, d)$, $\mathrm{lt}(a, e)$, $\mathrm{lt}(b, c)$, $\mathrm{lt}(b, d)$, $\mathrm{lt}(b, e)$, $\mathrm{lt}(c, d)$, $\mathrm{lt}(c, e)$, $\mathrm{lt}(d, e)$, $\mathrm{nsucc}(a, c)$, $\mathrm{nsucc}(a, d)$, $\mathrm{nsucc}(a, e)$, $\mathrm{nsucc}(b, d)$, $\mathrm{nsucc}(b, e)$, $\mathrm{nsucc}(c, e)$, $\mathrm{ninf}(b)$, $\mathrm{ninf}(c)$, $\mathrm{ninf}(d)$, $\mathrm{ninf}(e)$, $\mathrm{nsup}(a)$, $\mathrm{nsup}(b)$, $\mathrm{nsup}(c)$, and $\mathrm{nsup}(d)$.

We now define the required 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 $U \le Y$. In other words, we perform a loop starting with the infimum Y and then use the successor predicate to derive $\mathrm{defended\_upto}(X, Y)$ for all further Y. If we arrive at the supremum element in this way, i.e. $\mathrm{defended\_upto}(X, Y)$ is derived for the supremum Y, we finally obtain $\mathrm{defended}(X)$. We define

$$\begin{aligned}\pi_{\mathrm{defended}} = \{&\mathrm{defended\_upto}(X, Y) :- \mathrm{inf}(Y), \mathrm{arg}(X), \mathrm{not}\,\mathrm{defeat}(Y, X); && (25)\\ &\mathrm{defended\_upto}(X, Y) :- \mathrm{inf}(Y), \mathrm{in}(Z), \mathrm{defeat}(Z, Y),\\ &\qquad \mathrm{defeat}(Y, X); && (26)\\ &\mathrm{defended\_upto}(X, Y) :- \mathrm{succ}(Z, Y), \mathrm{defended\_upto}(X, Z),\\ &\qquad \mathrm{not}\,\mathrm{defeat}(Y, X); && (27)\\ &\mathrm{defended\_upto}(X, Y) :- \mathrm{succ}(Z, Y), \mathrm{defended\_upto}(X, Z),\\ &\qquad \mathrm{in}(V), \mathrm{defeat}(V, Y), \mathrm{defeat}(Y, X); && (28)\\ &\mathrm{defended}(X) :- \mathrm{sup}(Y), \mathrm{defended\_upto}(X, Y)\}, \text{ and} && (29)\\ \pi_{\mathrm{ground}} = \pi_{<} &\cup \pi_{\mathrm{defended}} \cup \{\mathrm{in}(X) :- \mathrm{defended}(X)\}.\end{aligned}$$

Note that π_{ground} is also stratified. We also mention that the relevant predicate $\mathrm{in}(a)$ is derived for some argument a if $\mathrm{defended\_upto}(a, b)$ holds for each b. However, if there is an unattacked argument c which attacks a, $\mathrm{defended\_upto}(a, c)$ is not derived. It is thus not relevant in which order we derive the predicates $\mathrm{defended\_upto}(a, b)$. Consequently, the particular definition of the order $<$, from which we obtained the $\mathrm{inf}(\cdot)$, $\mathrm{succ}(\cdot, \cdot)$, and $\mathrm{sup}(\cdot)$ predicates used in π_{defended}, plays no role. Any total order over the constants can be used.

For illustration, we compute the answer set for $\hat{F} \cup \pi_< \cup \pi_{\text{defended}}$ step by step, for our example AF F. Recall that we already have given S_0, the answer set of $\hat{F} \cup \pi_<$. Since π_{ground} only derives distinguished predicates defended_upto$(\cdot, \cdot)$ and defended$(\cdot)$, we can view S_0 as input to π_{ground}.

In the "first round" of computation, we have no in$(\cdot)$ predicate derived so far; hence only the rules (25) and (27) in π_{defended} are of interest. In fact, for inf(a), the rule (25) in π_{defended} yields

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

Note that defended_upto(b, a) is missing, since we have defeat$(a, b) \in \hat{F}$. Now we use rule (27) and succ(a, b) to obtain

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

The remaining atoms we derive are

$$\text{defended\_upto}(a, c), \text{defended\_upto}(c, c), \text{defended\_upto}(e, c).$$

Note that, since d is attacked by c, defended_upto(d, c) cannot be derived. Finally, we get

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

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 $\pi_{\text{ground}}(\hat{F})$ the only in$(\cdot)$ predicate is in(a). This corresponds to the grounded extension of F, as desired.

PROPOSITION 3.6 *For any AF F*, ground$(F) \cong \mathcal{AS}(\pi_{\text{ground}}(\hat{F}))$.

Proof Let $F = (A, R)$ and let $\hat{F}_<$ be the answer set of $\pi_<(\hat{F})$. Recall that each F possesses a unique grounded extension which is given by the least fixed point of Γ_F. In order to prove the correspondence between the operator Γ_F, and the unique answer set of the program $\pi_{\text{ground}}(\hat{F})$, we first define the operator $\Delta_{\hat{F}}$ representing the derivation of the answer set $\hat{F}_{\text{ground}}$ of $\pi_{\text{ground}}(\hat{F})$ as follows.

$$\begin{aligned}\Delta_{\hat{F}}(I) = I \cup \{&\text{defended\_upto}(a, b) \mid \text{arg}(a), \text{arg}(b) \in I, \forall c \text{ s.t. } \text{arg}(c) \in I \text{ and } c \leq b :\\ &\quad \text{defeat}(c, a) \notin I \text{ or } \exists d \text{ s.t. } \text{defeat}(d, c) \in I \text{ and } \text{in}(d) \in I\} \cup\\ &\{\text{defended}(a), \text{in}(a) \mid \text{arg}(a) \in I, \forall c \text{ s.t. } \text{arg}(c) \in I :\\ &\quad \text{defeat}(c, a) \notin I \text{ or } \exists d \text{ s.t. } \text{defeat}(d, c) \in I \text{ and } \text{in}(d) \in I\}.\end{aligned}$$

Using the one-step provability operator for logic programs (see e.g. van Emden and Kowalski (1976) for details on that operator), it can be shown that for every AF $F = (A, R)$, the answer set of the program $\pi_{\text{ground}}(\hat{F})$ is given by the least fixed point of $\Delta_{\hat{F}}$. It remains to show that, in turn, the least fixed point of $\Delta_{\hat{F}}$ equals the grounded extension of F.

Therefore, let $\Delta^0_{\hat{F}} = \Delta_{\hat{F}}(\hat{F}_<)$ and, for $i > 0$, $\Delta^i_{\hat{F}} = \Delta_{\hat{F}}(\Delta^{i-1}_{\hat{F}})$. Likewise, for the operator Γ defined in Proposition 2.10, let $\Gamma^0_F = \Gamma_F(\emptyset)$, and $\Gamma^i_F = \Gamma_F(\Gamma^{i-1}_F)$.

We now show, given an AF $F = (A, R)$, that

$$\Gamma^i_F = \{a \in A \mid \text{in}(a) \in \Delta^i_{\hat{F}}\} \tag{30}$$

holds for every $i \geq 0$. The proof is by induction on i.

Induction base Suppose $i = 0$ and let $a \in \Gamma_F^0$. We show that in$(a) \in \Delta_{\hat{F}}^0$. Since $a \in \Gamma_F^0$ and $\Gamma_F^0 = \Gamma_F(\emptyset)$, a is not attacked in F, i.e. there is no $c \in A$ with $(c, a) \in R$. By definition, arg$(a) \in \hat{F}_<$, and there is no c such that defeat$(c, a) \in \hat{F}_<$. Hence, in$(a) \in \Delta_{\hat{F}}^0$ by construction. To show that, for each $a \in A$ with in$(a) \in \Delta_{\hat{F}}^0$, also $a \in \Gamma_F^0$ holds, can be done in a similar manner.

Induction step Let $i > 0$, and suppose Relation (30) holds for all $0 \leq j < i$; in particular, we can assume that $\Gamma_F^{i-1} = \{a \in A \mid \text{in}(a) \in \Delta_{\hat{F}}^{i-1}\}$ holds. Let $a \in \Gamma_F^i \setminus \Gamma_F^{i-1}$. We show in$(a) \in \Delta_{\hat{F}}^i$. Since $a \in \Gamma_F^i$, it must hold that for each $c \in A$ with $(c, a) \in R$, there exists an argument $d \in \Gamma_F^{i-1}$ with $(d, c) \in R$. Therefore, in$(d) \in \Delta_{\hat{F}}^{i-1}$ holds by induction hypothesis. Moreover, we clearly have arg$(e) \in \Delta_{\hat{F}}^{i-1}$ for all $e \in A$, and defeat$(e, f) \in \Delta_{\hat{F}}^{i-1}$ for all $(e, f) \in R$ (recall that $\hat{F} \subseteq \Delta_{\hat{F}}^{i-1}$). Hence, we know that for each c with arg$(c) \in \Delta_{\hat{F}}^{i-1}$, such that defeat$(c, a) \in \Delta_{\hat{F}}^{i-1}$, there exists an argument d, such that defeat$(d, c) \in \Delta_{\hat{F}}^{i-1}$ and in$(d) \in \Delta_{\hat{F}}^{i-1}$. By definition, in$(a) \in \Delta_{\hat{F}}^i$. Again, the other direction is by similar arguments. ■

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

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

We continue with the more involved encodings for preferred and semi-stable extensions. Compared with the one for admissible extensions, these encodings require an additional maximality test. However, this is quite complicate to encode and requires the use of disjunction, together with the saturation technique introduced in Section 2.1 for the encoding of the complement of the 3-colourability problem.

3.5. *Preferred extensions*

To compute the preferred extensions of an AF, we will use the saturation technique described in Section 2.1 as follows: Having computed an admissible extension S (characterised via predicates in$(\cdot)$ and out$(\cdot)$ using our encoding $\pi_{\text{adm}}(\hat{F})$), we perform a second guess using new predicates, say inN$(\cdot)$ and outN$(\cdot)$, to represent a further guess $T \supset S$. In order to check whether the first guess (via in$(\cdot)$ and out$(\cdot)$) characterises a preferred extension, we have to ensure that *no* guess of the second form (i.e., via inN$(\cdot)$ and outN$(\cdot)$) characterises an admissible extension. Recall that in Section 2.1, we already have seen how to encode (using the program π_{ncol}) such a complementary problem when we considered the problem of non-3-colourability. Here the saturation module looks as follows:

$$\pi_{\text{satpref}} = \{\text{inN}(X) \vee \text{outN}(X) :- \text{out}(X); \tag{31}$$

$$\text{inN}(X) :- \text{in}(X); \tag{32}$$

$$\text{fail} :- \text{eq}; \tag{33}$$

$$\text{fail} :- \text{inN}(X), \text{inN}(Y), \text{defeat}(X, Y); \tag{34}$$

$$\text{fail} :- \text{inN}(X), \text{outN}(Y), \text{defeat}(Y, X), \text{undefeated}(Y); \tag{35}$$

$$\text{inN}(X) :- \text{fail}, \text{arg}(X); \tag{36}$$

$$\text{outN}(X) \;{:}{-}\; \text{fail}, \text{arg}(X); \tag{37}$$

$$ {:}{-}\; \text{not fail}\}. \tag{38}$$

Let us for the moment also assume that predicates eq (rule (33)) and undefeated$(\cdot)$ (rule (35)) are defined (we will give the additional modules for those predicates below) and provide the following information:

- eq is derived if the guess S via in$(\cdot)$ and out$(\cdot)$ equals the second guess T via inN$(\cdot)$ and outN$(\cdot)$; in other words, eq is derived if $S = T$;
- undefeated(a) is derived if argument a is not defeated in F by the second guess T.

In what follows, we discuss the functioning of π_{satpref} when conjoined with the program $\pi_{\text{adm}}(\hat{F})$ from Section 3.2, for a given AF F. First, rule (31) guesses a set $T \subseteq A$ as already discussed above. Rule (32) ensures that the new guess satisfies $S \subseteq T$ (recall that S is characterised by predicates in$(\cdot)$ and out$(\cdot)$ from π_{adm}).

The task of the rules (33)–(35) is to check whether the new guess T is a proper superset of S and characterises an admissible extension of the given AF F. If this is not the case, we derive the predicate fail. More specifically, rule (33) checks whether $S = T$, and in case this holds we derive fail; rule (34) checks whether T is not conflict-free in F, and in case this holds we derive fail; rule (35) checks whether T contains an argument not defended by T in F, and in case this holds we derive fail. In other words, we have not derived fail if $T \supset S$ and T is admissible in F. By definition, S then cannot be a preferred extension of F.

The remaining rules (36)–(38) saturate the guess in case fail was derived, and finally ensure that fail has to be in an answer set. We already discussed the underlying idea of such rules for the program π_{ncol} in terms of the non-3-colourability encoding in Section 2.1 (see rules (12)–(15) of program π_{ncol}).

Let us illustrate now the behaviour of π_{satpref} for two scenarios. First, suppose the first guess S (via predicates in$(\cdot)$ and out$(\cdot)$) is a preferred extension of the given AF $F = (A, R)$. Hence, for each $T \supset S$, T must not be admissible. But then, we have that every new guess T (via predicates inN$(\cdot)$ and outN$(\cdot)$) derives fail. Due to the spoiling rules (36) and (37), we thus have no interpretation (without predicate fail) which satisfies π_{satpref}. However, the saturated interpretation (which contains fail and both inN(a) and outN(a) for each $a \in A$) does satisfy the program and also becomes an answer set of the program.

Now suppose the first guess S (via predicates in$(\cdot)$ and out$(\cdot)$) is an admissible but not a preferred extension of the given AF F. Then there exists a set $T \supset S$, such that T is admissible for F. If we consider the interpretation I characterising T (i.e. we have inN$(a) \in I$, for each $a \in T$, and outN$(a) \in I$, for each $a \in A \setminus T$), then I does not contain fail and satisfies the rules (31)–(37). But this shows that we cannot have an answer set J which characterises S. Due to rule (38) such an answer set J has to contain fail and by rules (36) and (37), J contains both inN(a) and outN(a) for each $a \in A$. Note that we thus have $I \subset J$ (if I and J characterise the same initial guess S). Moreover, I satisfies the reduct of our program with respect to J. This can be seen by the fact that the only occurrence of default negation is in rule (38). In other words, there is an $I \subset J$ satisfying the reduct and thus J cannot be an answer set. This, however, is as desired, since the initial guess S characterised by J is not a preferred extension.

We still have to define the rules for the predicates eq and undefeated$(\cdot)$. Basically, these predicates would be easy to define, but as we have seen in the discussion above, default negation plays a central rule in the saturation technique (recall the functioning of :– not fail). We therefore have to find encodings which suitably define the required predicates only with a limited use of negation. In fact, we are only allowed to have stratified negation in these modules. Therefore, we employ similar techniques as we have used for the predicate defended$(\cdot)$ as defined in Section 3.4 where we also required a stratified program. In fact, both predicates eq and undefeated$(\cdot)$ are computed via predicates eq_upto$(\cdot)$ (resp., undefeated_upto$(\cdot,\cdot)$) in the same manner as we used defended_upto$(\cdot,\cdot)$ for defended$(\cdot)$ in the module π_{defended} in Section 3.4. We thus make use of the helper predicates from $\pi_<$ as defined in the previous subsection and define

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

With these predicates at hand, we can now formally define the module for preferred extensions, $\pi_{\text{pref}} = \pi_{\text{adm}} \cup \pi_< \cup \pi_{\text{eq}} \cup \pi_{\text{undefeated}} \cup \pi_{\text{satpref}}$. The general result is as follows.

PROPOSITION 3.7 *For any AF F,* $\text{pref}(F) \cong \mathcal{AS}(\pi_{\text{pref}}(\hat{F}))$.

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 $\Pi = \pi_{\text{eq}} \cup \pi_{\text{undefeated}} \cup \pi_{\text{satpref}}$ has to be treated as a whole, due to the cyclic dependencies among the atoms (in other words, we obtain only trivial splitting sets for Π). However, we can still split π_{pref} into $\pi_{\text{adm}} \cup \pi_<$ and Π. We already know the single answer set S_0 of $\pi_<(\hat{F})$ (see Section 3.4) and the collection $\{S_\emptyset, S_a, S_c, S_d, S_{ac}, S_{ad}\} = \mathcal{AS}(\pi_{\text{adm}}(\hat{F}))$ of answer sets of $\pi_{\text{adm}}(\hat{F})$ (see Section 3.2). As is easily checked, we get

$$\mathcal{AS}(\hat{F} \cup \pi_{\text{adm}} \cup \pi_<) = \{S_0 \cup S \mid S \in \mathcal{AS}(\pi_{\text{adm}}(\hat{F}))\}.$$

Hence, let us illustrate the program Π for the two inputs $S_1 = S_0 \cup S_a$ and $S_2 = S_0 \cup S_{ac}$ (We omit the further atoms from the corresponding answer sets in $\hat{F} \cup \pi_{\text{adm}} \cup \pi_<$, since they play no role in Π.). Indeed, we expect that S_1 does not lead to an answer set of $\pi_{\text{pref}}(\hat{F})$, while the second set S_2 corresponds to a preferred extension of F, and thus should be part of an answer set of $\pi_{\text{pref}}(\hat{F})$. As discussed above, the only potential answer set I_1 of $\Pi(S_1)$ contains

$$\begin{aligned}
S_1 \cup \{&\text{inN}(a), \text{inN}(b), \text{inN}(c), \text{inN}(d), \text{inN}(e),\\
&\text{outN}(a), \text{outN}(b), \text{outN}(c), \text{outN}(d), \text{outN}(e), \text{fail}\}.
\end{aligned}$$

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

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

satisfies $\Pi(S_1)^{I_1}$. This can be seen as follows: The above set J_1 does not contain fail, thus the bodies of rules (33–35) 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 inN$(\cdot)$ holds are not adjacent. Finally, for rule (35), we first mention that the predicate undefeated$(\cdot)$ is derived for the following instantiations undefeated(a), undefeated(c), undefeated(e). One can now check that the bodies of rule (35) are not satisfied. As well, rules (36) and (37) are not applied (since fail 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 $\pi_{\text{pref}}(\hat{F})$.

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

$$S_2 \cup \{\text{inN}(a), \text{inN}(b), \text{inN}(c), \text{inN}(d), \text{inN}(e),$$
$$\text{outN}(a), \text{outN}(b), \text{outN}(c), \text{outN}(d), \text{outN}(e), \text{fail}\}.$$

Moreover, $\Pi(S_2)^{I_2} = \Pi(S_2) \setminus \{:- \text{not fail}\}$ 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 (32) guarantees that J_2 contains at least inN(a), inN(c) but further inN$(\cdot)$ predicates could be contained in J_2. However, if the only inN$(\cdot)$ predicates in J_2 are inN(a), inN(c), predicate eq is derived and we saturate. As well, if a further inN$(\cdot)$ predicate is contained in J_2 then we already know that such a set characterises a subset $S' \subseteq A$ which cannot be conflict-free. Indeed, rule (34) applies in this case, and we obtain fail. As soon as fail is derived, rules (36) and (37) "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 $\pi_{\text{pref}}(\hat{F})$. This meets our expectation, since S_{ac} relates to the preferred extension $\{a, c\}$ of F.

3.6. *Semi-stable extensions*

We conclude our encodings for the different types of extensions with a query for the semi-stable semantics. The basic intuition for the forthcoming encoding is the same 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 $T \in adm(F)$ with $S_R^+ \subset T_R^+$ exists, while for preferred extensions, it was sufficient to test whether no such T exists for which $S \subset T$ holds. 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 saturate (as before) in case T is not admissible. Finally, we explicitly get rid of 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 saturate if eqplus is derived, or in case there exists an $a \in S_R^+$ not contained in T_R^+.

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

$$\pi_{\text{eq}}^{+} = \{\text{eqplus\_upto}(X) :- \text{inf}(X), \text{in}(X), \text{inN}(X);$$
$$\text{eqplus\_upto}(X) :- \text{inf}(X), \text{in}(X), \text{inN}(Y), \text{defeat}(Y, X);$$
$$\text{eqplus\_upto}(X) :- \text{inf}(X), \text{in}(Y), \text{inN}(X), \text{defeat}(Y, X);$$
$$\text{eqplus\_upto}(X) :- \text{inf}(X), \text{in}(Y), \text{inN}(Z), \text{defeat}(Y, X), \text{defeat}(Z, X);$$
$$\text{eqplus\_upto}(X) :- \text{inf}(X), \text{out}(X), \text{outN}(X), \text{not defeated}(X),$$
$$\text{undefeated}(X);$$
$$\text{eqplus\_upto}(X) :- \text{succ}(Z, X), \text{in}(X), \text{inN}(X), \text{eqplus\_upto}(Z);$$
$$\text{eqplus\_upto}(X) :- \text{succ}(Z, X), \text{in}(X), \text{inN}(Y), \text{defeat}(Y, X), \text{eqplus\_upto}(Z);$$

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

and

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

We define $\pi_{\text{semi}} = \pi_{\text{adm}} \cup \pi_{<} \cup \pi^{+}_{\text{eq}} \cup \pi_{\text{undefeated}} \cup \pi_{\text{satsemi}}$ and obtain the following result.

PROPOSITION 3.8 *For any AF F,* $\text{semi}(F) \cong \mathcal{AS}(\pi_{\text{semi}}(\hat{F}))$.

3.7. *Summary and combinations of encodings*

We summarise the results from this section.

THEOREM 3.9 *For any AF F and $e \in$* {stable, adm, pref, semi, comp, ground}*, it holds that $e(F) \cong \mathcal{AS}(\pi_e(\hat{F}))$.*

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 queries which have to be used 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 and constraint-free. Recall that such programs have exactly 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 type of queries used in Table 3, one obtains Table 2. This shows that the complexity of the encoded decision problem of AFs meets the corresponding complexity of the employed datalog fragment.

We are now able to encode more involved decision problems using our queries. As a first example, consider the Π_2^P-complete problem of coherence (Dunne and Bench-Capon 2002), which decides whether for a given AF F, $\text{pref}(F) \subseteq \text{stable}(F)$ (recall that $\text{pref}(F) \supseteq \text{stable}(F)$

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

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

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

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

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. Dunne and Caminada (2008) have shown Π_2^P-completeness for this problem.

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 inNN$(\cdot)$, outNN$(\cdot)$, undefeatedN$(\cdot)$ and undefeatedN_upto$(\cdot,\cdot)$, and denote by $\pi_{\text{undefeatedN}}$ the program resulting from $\pi_{\text{undefeated}}$ by using the new atoms instead of inN$(\cdot)$, outN$(\cdot)$, undefeated$(\cdot)$ and undefeated_upto$(\cdot,\cdot)$, respectively. Similarly, we obtain π^+_{eqN} from π^+_{eq}. Consider now the following program.

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

Corollary 3.11 *Given an AF F, it holds that* semi(F) = pref(F) *iff* $\pi_{\text{coincide}}(\hat{F})$ *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 saturation 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 generalisations of argumentation frameworks

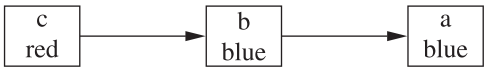
In this section, we again show the flexibility of our approach by reusing the queries from the previous section in the context of generalisations of abstract AFs.

4.1. *Value-based argumentation frameworks*

As a first example for generalising basic AFs, we deal with value-based AFs (VAFs) (Bench-Capon 2003) which themselves generalise the preference-based AFs (Amgoud and Cayrol 2002). Again we give the definition w.r.t. the universe $\mathcal{U}$.

Definition 4.1 *A VAF is a 5-tuple* $F = (A, R, \Sigma, \sigma, <)$ *where* $A \subseteq \mathcal{U}$ *are arguments,* $R \subseteq A \times A$ *denotes the attack relation,* $\Sigma \subseteq \mathcal{U}$ *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 iff* $(a, b) \in R$ *and* $(\sigma(b), \sigma(a)) \notin \ll$.

Example 4.2 Consider the following VAF $F = (A,\ R,\ \Sigma,\ \sigma,\ <)$. Let $A = \{a, b, c\}$, $R = \{(b, a), (c, b)\}$, $\Sigma = \{\text{red}, \text{blue}\}$, $\sigma(a) = \text{blue}$, $\sigma(b) = \text{blue}$, $\sigma(c) = \text{red}$, and $<= \{(\text{red}, \text{blue})\}$. The graph representation of F is as follows.



We obtain that c defeats b, because $(c, b) \in R$ and $(\sigma(b), \sigma(c)) \notin \ll$ holds. But, if $(\text{blue}, \text{red}) \in <$, this does not hold any longer.

Using this notion of defeat, we say in accordance with Definition 2.2 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 defence, the definitions in Bench-Capon (2003) for conflict-free sets, admissible extensions, and preferred extensions are exactly along the lines of Definitions 2.4, 2.6, and 2.7, respectively.

In order to compute these extensions for VAFs, we thus only need to slightly adapt the modules introduced in Section 3.

In fact, we now use, for a VAF F,

$$\begin{aligned}\tilde{F} = {} & \{\text{arg}(a) \mid a \in A\} \cup \{\text{att}(a, b) \mid (a, b) \in R\} \cup \\ & \{\text{val}(a, \sigma(a)) \mid a \in A\} \cup \{\text{valpref}(v, w) \mid v < w\}\end{aligned}$$

and we require a further module, which obtains the defeat$(\cdot, \cdot)$ relation accordingly:

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

We obtain the following theorem using the new concepts for $\tilde{F}$ and π_{vaf}, as well as reusing π_{adm} and π_{pref} from Section 3.

Theorem 4.3 *For any VAF F and* $e \in \{\text{adm}, \text{pref}\}$, $e(F) \cong \mathcal{AS}(\pi_{\text{vaf}} \cup \pi_e(\tilde{F}))$.

For the other notions of extensions, we can employ our encodings from Section 3 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 (Bench-Capon 2002), 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:

$$\begin{aligned}\pi_{\text{vaf\_stable}} = \pi_{\text{cf}} \cup \{&\text{attacked}(X) :\!- \text{in}(Y), \text{att}(Y, X);\\ &:\!- \text{out}(X), \text{not attacked}(X)\}.\end{aligned}$$

THEOREM 4.4 *For any VAF F*, $\text{stable}(F) \cong \mathcal{AS}(\pi_{\text{vaf}} \cup \pi_{\text{vaf\_stable}}(\tilde{F}))$.

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

COROLLARY 4.5 *The coherence problem for a VAF F holds iff the program*

$$\begin{aligned}\pi_{\text{pref}}(\tilde{F}) \cup \ &\{\text{attacked}(X) :\!- \text{in}(Y), \text{att}(Y, X);\\ &v :\!- \text{out}(X), \text{not attacked}(X);\ :\!- \text{not } v\}\end{aligned}$$

has no answer set.

4.2. *Bipolar argumentation frameworks*

Bipolar AFs (BAFs) (Cayrol and Lagasquie-Schiex 2005) augment basic AFs by a second relation between arguments which indicates supports independent from defeats.

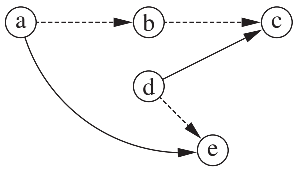
DEFINITION 4.6 *A BAF is a tuple $F = (A, R_d, R_s)$ where $A \subseteq \mathcal{U}$ is a set of arguments, and $R_d \subseteq A \times A$ and $R_s \subseteq A \times A$ are the attack, and resp., support relation of F.*

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 $2 \leq i \leq n$.

A BAF can be represented by a directed graph called the *bipolar interaction graph*. In order to distinguish between the two relations, two kinds of edges are used. For two arguments a and b, $(a, b) \in R_d$ is represented by $a \to b$ and $(a, b) \in R_s$ is represented by $a \dashrightarrow b$.

Example 4.7 Consider the following $F = (A, R_d, R_s)$. Let $A = \{a, b, c, d, e\}$, $R_d = \{(a, e), (d, c)\}$ and $R_s = \{(a, b), (b, c), (d, e)\}$. Then the bipolar interaction graph looks as follows.



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 and incorporate the new defeat-relation. Other modules from Section 3 can then be reused. In fact, we define for a given BAF $F = (A, R_d, R_s)$,

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

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

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

Following Cayrol and Lagasquie-Schiex (2005), we can use this notion of defeat to define conflict-free sets, stable extensions, admissible extensions, and preferred extensions (these extensions are respectively called d-admissible and d-preferred in Cayrol and Lagasquie-Schiex (2005)) exactly along the lines of Definitions 2.4–2.7, respectively.

THEOREM 4.8 *For any BAF F and $e \in \{\mathrm{stable}, \mathrm{adm}, \mathrm{pref}\}$, $e(F) \cong \mathcal{AS}(\pi_{\mathrm{baf}} \cup \pi_e(\bar{F}))$.*

More specific variants of admissible extensions from Cayrol and Lagasquie-Schiex (2005) are obtained by replacing the notion of a conflict-free set by other concepts.

DEFINITION 4.9 *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 4.10 *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:

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

The two constraints in π_{cadm} rule out, according to the definition of closed sets, all answer sets where $(a,b) \in R_s$ but either $a \in S$ and $b \notin S$ or $a \notin S$ and $b \in S$ (note that the relation support is not symmetric).

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

DEFINITION 4.11 *Let $F = (A, R_d, R_s)$ be a BAF. A set $S \subseteq A$ is called an s-preferred extension of F if $S \in \mathrm{sadm}(F)$ and for each $T \in \mathrm{sadm}(F)$, $S \not\subset T$. Likewise, a set $S \subseteq A$ is called a c-preferred extension of F if $S \in \mathrm{cadm}(F)$ and for each $T \in \mathrm{cadm}(F)$, $S \not\subset T$. By $\mathrm{spref}(F)$ (resp., $\mathrm{cpref}(F)$) we denote the collection of all s-preferred extensions (resp., of all c-preferred extensions) of F.*

For the framework F from Example 4.7, we obtain $\mathrm{pref}(F) = \{\{a, b, d\}\}$, $\mathrm{spref}(F) = \{\{d\}, \{a, b\}\}$, and $\mathrm{cpref}(F) = \{\emptyset\}$.

Again, we can reuse parts of the program π_{pref} from Section 3. The only addition necessary is to saturate in case the additional requirements are violated.

We define

$$\begin{aligned}
\pi_{\mathrm{spref}} &= \pi_{\mathrm{sadm}} \cup \pi_{<} \cup \pi_{\mathrm{eq}} \cup \pi_{\mathrm{undefeated}} \cup \pi_{\mathrm{satpref}} \cup \\
&\qquad \{\mathrm{supported}(X) :- \mathrm{inN}(Y), \mathrm{support}(Y, X); \\
&\qquad \ \mathrm{fail} :- \mathrm{supported}(X), \mathrm{defeated}(X)\}; \\
\pi_{\mathrm{cpref}} &= \pi_{\mathrm{cadm}} \cup \pi_{<} \cup \pi_{\mathrm{eq}} \cup \pi_{\mathrm{undefeated}} \cup \pi_{\mathrm{satpref}} \cup \\
&\qquad \{\mathrm{fail} :- \mathrm{support}(X, Y), \mathrm{inN}(X), \mathrm{outN}(Y); \\
&\qquad \ \mathrm{fail} :- \mathrm{support}(X, Y), \mathrm{outN}(X), \mathrm{inN}(Y)\}.
\end{aligned}$$

THEOREM 4.12 *For any BAF F and $e \in \{\mathrm{sadm}, \mathrm{cadm}, \mathrm{spref}, \mathrm{cpref}\}$, we have $e(F) \cong \mathcal{AS}(\pi_{baf} \cup \pi_e(\hat{F}))$.*

Slightly different semantics for BAFs occur in Amgoud et al. (2008), where the notion of defence 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 4.6. 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, to the best of our knowledge, have not been studied in the literature yet.

5. Discussion

In this work we provided ASP-encodings for computing different types of extensions in Dung's AF as well as in some recent extensions of it. Our system ASPARTIX, together with some illustrating examples, is available at www.dbai.tuwien.ac.at/research/project/argumentation/systempage/

Besides the types of extensions discussed in this paper, the current version of ASPARTIX also implements the recently proposed concept of ideal semantics (Dung, Mancarella, and Toni 2007). For the respective encoding for ideal semantics, we refer to the paper by Faber and Woltran (2009).

To the best of our knowledge, so far no system is available which supports such a broad range of different semantics, although nowadays a reasonable number of different implementations exists. (See http://wyner.info/LanguageLogicLawSoftware/index.php/software/ for an overview.) In short, one can divide these systems into two categories: graphical representation of AFs and computation of acceptable arguments. The first category contains systems like Argunet (http://www.argunet.org/debates/) and Araucaria (Reed and Rowe 2004). To the second category belong systems like:

- ArgKit (South, Vreeswijk, and Fox 2008), a Java reasoner capable of reasoning with grounded and preferred extensions which can be integrated in Araucaria.

- A system by Verheij (2007) which computes grounded, preferred, stable and semi-stable semantics via labelings.
- The epistemic and practical reasoner by Visser (2008) which is an implementation of an argument-based practical reasoning system. From a given belief base of formulas (in a propositional modal logic with a single modality D standing for desire) and a query formula, one selects between different argument games including one for grounded semantics and one for credulous reasoning over admissible extensions.
- PARMENIDES (Cartwright and Atkinson 2008), a system for e-democracy that makes use of argumentation tools including VAFs.
- CASAPI (Gaertner and Toni 2007), a Prolog implementation for credulous and sceptical argumentation based upon the computation of dispute derivations for grounded, admissible and ideal beliefs for the generalised assumption-based frameworks.
- An implementation of a query-answering algorithm due to Vreeswijk (2006) for grounded and admissible semantics. Compared with the other systems, this algorithm computes the so-called defence sets around the queried argument rather than the entire collection of extensions.
- An implementation (Efstathiou and Hunter 2008) for logic-based formalisations of argumentation (Besnard and Hunter 2001), which constructs arguments using connection graphs.

The work which is related closest to ours is by Nieves et al. (2008, 2009) who also suggested to use ASP for computing extensions of AFs. One aspect in their work is to use a fixed encoding *schema* to represent AFs as logic programs, and then show how different semantics for logic programs can be used to compute different forms of extensions using this particular schema. Most notably, they showed that in their setting the stable semantics (for logic programs) captures stable extensions of AFs, the well-founded semantics captures the grounded extension of AFs, and a novel stratification semantics (Nieves et al. 2009) captures the CF2 semantics due to Baroni, Giacomin and Guida (2005). Osorio et al. (2005) present an algorithm for computing preferred extensions (based on abductive logic programming) using a fixed logic program to characterise the admissible sets in the same manner we have used here. In Nieves et al. (2008), a different approach to compute preferred extensions by means of logic programs has been proposed. However, this work requires a recompilation of the encoding for each particular AF. Similarly, Wakaki and Nitta (2008) also provide ASP encodings for different semantics. In contrast to our work, their encodings for complete and stable semantics are based on labellings, whereas for grounded, preferred and semi-stable semantics they use a meta-programming technique by applying additional translations for each AF into normal logic programs. We recall that in our system, *fixed* disjunctive queries which require the actual instance just as an input database are used for all these semantics. We believe that our approach is thus more reliable and easily extendible to further formalisms.

Future work includes a comparison of the efficiency of different implementations. Preliminary tests show that our approach is able to deal with more than 100 arguments for all considered semantics in Dung's abstract framework. Another direction of future work is to extend our system by incorporating further recent notions of semantics. In particular, it is planned to implement the resolution-based semantics due to Baroni and Giacomin (2008a), semantics based on decomposition into strongly connected components (like, for instance, the CF2 semantics) due to Baroni et al. (2005), novel approaches to preferential semantics due to Amgoud and Vesic (2009), and semantics based on meta-attacks as introduced by Modgil (2009).

Acknowledgements

The authors thank the anonymous referees for valuable comments which helped in improving the paper. This work was supported by the Austrian Science Fund (FWF) under the grant P20704-N18 and the Vienna Science and Technology Fund (WWTF) under the grant ICT08-028. A short version of this article was presented at the Proceedings of the ICLP'08 Workshop *Answer-Set Programming and Other Computing Paradigms (ASPOCP)*.

References

Amgoud, L., and Cayrol, C. (2002), 'A Reasoning Model Based on the Production of Acceptable Arguments', *Annals of Mathematics and Artificial Intelligence*, 34, 197–215.

Amgoud, L., and Vesic, S. (2009), 'Repairing Preference-Based Argumentation Frameworks', in *Proceedings of the 21st International Joint Conference on Artificial Intelligence* (*IJCAI 2009*), pp. 665–670.

Amgoud, L., Cayrol, C., Lagasquie-Schiex, M.C., and Livet, P. (2008), 'On Bipolarity in Argumentation Frameworks', *International Journal of Intelligent Systems*, 23, 1–32.

Apt, K.R., Blair, H.A., and Walker, A. (1988), 'Towards a Theory of Declarative Knowledge', in *Foundations of Deductive Databases and Logic Programming*, ed. J. Minker, Morgan Kaufmann Publishers, Inc., pp. 89–148.

Baroni, P., and Giacomin, M. (2008a), 'Resolution-based Argumentation Semantics', in *Proceedings of the 2nd Conference on Computational Models of Argument* (*COMMA 2008*), eds. P. Besnard, S. Doutre, and A. Hunter, Vol. 172 of *Frontiers in Artificial Intelligence and Applications*, Toulouse, France: IOS Press, pp. 25–36.

Baroni, P., and Giacomin, M. (2008b), 'A Systematic Classification of Argumentation Frameworks Where Semantics Agree', in *Proceedings of the 2nd Conference on Computational Models of Argument* (*COMMA 2008*), eds. P. Besnard, S. Doutre, and A. Hunter, Vol. 172 of *Frontiers in Artificial Intelligence and Applications*, Toulouse, France: IOS Press, pp. 37–48.

Baroni, P., Giacomin, M., and Guida, G. (2005), 'SCC-Recursiveness: A General Schema for Argumentation Semantics', *Artificial Intelligence*, 168, 162–210.

Bench-Capon, T.J.M. (2002), 'Value-based Argumentation Frameworks', in *Proceedings of the 9th International Workshop on Non-Monotonic Reasoning* (*NMR 2002*), eds. S. Benferhat and E. Giunchiglia, Toulouse, France, pp. 443–454.

Bench-Capon, T.J.M. (2003), 'Persuasion in Practical Argument Using Value-based Argumentation Frameworks', *Journal of Logic and Computation*, 13, 429–448.

Bench-Capon, T.J.M., and Dunne, P.E. (2007), 'Argumentation in Artificial Intelligence', *Artificial Intelligence*, 171, 619–641.

Besnard, P., and Doutre, S. (2004), 'Checking the Acceptability of a Set of Arguments', in *Proceedings of the 10th International Workshop on Non-Monotonic Reasoning* (*NMR 2004*), eds. J.P. Delgrande and T. Schaub, Whistler, Canada, pp. 59–64.

Besnard, P., and Hunter, A. (2001), 'A Logic-based Theory of Deductive Arguments', *Artificial Intelligence*, 128, 203–235.

Besnard, P., Hunter, A., and Woltran, S. (2009), 'Encoding Deductive Argumentation in Quantified Boolean Formulae', *Artificial Intelligence*, 173, 1406–1423.

Caminada, M. (2006), 'Semi-Stable Semantics', in *Proceedings of the 1st Conference on Computational Models of Argument* (*COMMA 2006*), eds. P.E. Dunne and T.J.M. Bench-Capon, Vol. 144 of *Frontiers in Artificial Intelligence and Applications*, Liverpool, UK: IOS Press, pp. 121–130.

Cartwright, D., and Atkinson, K. (2008), 'Political Engagement Through Tools for Argumentation', in *Proceedings of the 2nd Conference on Computational Models of Argument* (*COMMA 2008*), eds. P. Besnard, S. Doutre, and A. Hunter, Vol. 172 of *Frontiers in Artificial Intelligence and Applications*, Toulouse, France: IOS Press, pp. 116–127.

Cayrol, C., and Lagasquie-Schiex, M.C. (2005), '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 2005*), ed. L. Godo, Vol. 3571 of *LNCS*, Barcelona, Spain: Springer, pp. 378–389.

Coste-Marquis, S., Devred, C., and Marquis, P. (2005), ‘Symmetric Argumentation Frameworks’, in *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty* (*ECSQARU 2005*), ed. L. Godo, Vol. 3571 of *LNCS*, Barcelona, Spain: Springer, pp. 317–328.

Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (2001), ‘Complexity and Expressive Power of Logic Programming’, *ACM Computing Surveys*, 33, 374–425.

Dimopoulos, Y., and Torres, A. (1996), ‘Graph Theoretical Structures in Logic Programs and Default Theories’, *Theoretical Computer Science*, 170, 209–244.

Dung, P.M. (1995), ‘On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games’, *Artificial Intelligence*, 77, 321–358.

Dung, P.M., Mancarella, P., and Toni, F. (2007), ‘Computing Ideal Sceptical Argumentation’, *Artificial Intelligence*, 171, 642–674.

Dunne, P.E., and Bench-Capon, T.J.M. (2002), ‘Coherence in Finite Argument Systems’, *Artificial Intelligence*, 141, 187–203.

Dunne, P.E., and Bench-Capon, T.J.M. (2004), ‘Complexity in Value-Based Argument Systems’, in *Proceedings of the 9th European Conference on Logics in Artificial Intelligence* (*JELIA 2004*), eds. J.J. Alferes and J.A. Leite, Vol. 3229 of *LNCS*, Lisbon, Portugal: Springer, pp. 360–371.

Dunne, P.E., and Caminada, M. (2008), ‘Computational Complexity of Semi-Stable Semantics in Abstract Argumentation Frameworks’, in *Proceedings of the 11th European Conference on Logics in Artificial Intelligence* (*JELIA 2008*), eds. S. Hölldobler, C. Lutz, and H. Wansing, Vol. 5293 of *LNCS*, Dresden, Germany: Springer, pp. 153–165.

Dvořák, W., and Woltran, S. (2009), ‘Technical Note: Complexity of Stage Semantics in Argumentation Frameworks’, Technical Report DBAI-TR-2009-66, Technische Universität Wien, Database and Artificial Intelligence Group.

Efstathiou, V., and Hunter, A. (2008), ‘Algorithms for Effective Argumentation in Classical Propositional Logic: A Connection Graph Approach’, in *Proceedings of the 5th International Symposium on Foundations of Information and Knowledge Systems*, (*FoIKS 2008*), eds. S. Hartmann and G. Kern-Isberner, February, Vol. 4932 of *Lecture Notes in Computer Science*, Pisa, Italy: Springer, pp. 272–290.

Egly, U., and Woltran, S. (2006), ‘Reasoning in Argumentation Frameworks Using Quantified Boolean Formulas’, in *Proceedings of the 1st Conference on Computational Models of Argument* (*COMMA 2006*), eds. P.E. Dunne and T.J.M. Bench-Capon, Vol. 144 of *Frontiers in Artificial Intelligence and Applications*, Liverpool, UK: IOS Press, pp. 133–144.

Eiter, T., and Polleres, A. (2006), ‘Towards Automated Integration of Guess and Check Programs in Answer Set Programming: A Meta-Interpreter and Applications’, *Theory and Practice of Logic Programming*, 6, 23–60.

Eiter, T., Gottlob, G., and Mannila, H. (1997), ‘Disjunctive Datalog’, *ACM Transactions on Database Systems*, 22, 364–418.

Faber, W., and Woltran, S. (2009), ‘Manifold Answer-Set Programs for Meta-Reasoning’, in *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 2009*), eds. E. Erdem, F. Lin, and T. Schaub, Vol. 5753 of *LNCS*, Potsdam, Germany: Springer, pp. 115–128.

Gaertner, D., and Toni, F. (2007), ‘CaSAPI – A System for Credulous and Sceptical Argumentation’, in *Proceedings of the First International Workshop on Argumentation and Nonmonotonic Reasoning* (*ArgNMR 2007*), eds. G.R. Simari and P. Torroni, May, Arizona, pp. 80–95.

Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., and Truszczyński, M. (2007), ‘The First Answer Set Programming System Competition’, in *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 2007*), eds. C. Baral, G. Brewka, and J.S. Schlipf, Vol. 4483 of *LNCS*, Tempe, AZ, USA: Springer, pp. 3–17.

Gelfond, M., and Lifschitz, V. (1991), ‘Classical Negation in Logic Programs and Disjunctive Databases’, *New Generation Computing*, 9, 365–386.

Leone, N., Pfeifer, G., Faber,W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. (2006), ‘The DLV System for Knowledge Representation and Reasoning’, *ACM Transactions on Computational Logic*, 7, 499–562.

Lifschitz, V., and Turner, H. (1994), 'Splitting a Logic Program', in *Proceedings of the 11th International Conference on Logic Programming* (*ICLP'94*), Santa Margherita Ligure, Italy: MIT Press, pp. 23–37.

Modgil, S. (2009), 'Reasoning about Preferences in Argumentation Frameworks', *Artificial Intelligence*, 173, 901–934.

Niemelä, I. (1999), 'Logic Programming with Stable Model Semantics as a Constraint Programming Paradigm', *Annals of Mathematics and Artificial Intelligence*, 25, 241–273.

Nieves, J.C., Osorio, M., and Cortés, U. (2008), 'Preferred Extensions as Stable Models', *Theory and Practice of Logic Programming*, 8, 527–543.

Nieves, J.C., Osorio, M., and Zepeda, C. (2009), 'Expressing Extension-Based Semantics Based on Stratified Minimal Models', in *Proceedings of the 16th International Workshop on Logic, Language, Information and Computation*, (*WoLLIC 2009*), eds. H. Ono, M. Kanazawa, and R.J.G.B. de Queiroz, Vol. 5514 of *LNCS*, Tokyo, Japan: Springer, pp. 305–319.

Osorio, M., Zepeda, C., Nieves, J.C., and Cortés, U. (2005), 'Inferring Acceptable Arguments with Answer Set Programming', in *Proceedings of the 6th Mexican International Conference on Computer Science* (*ENC 2005*), Puebla, Mexico: IEEE Computer Society, pp. 198–205.

Reed, C., and Rowe, G. (2004), 'Araucaria: Software for Argument Analysis, Diagramming and Representation', *International Journal on Artificial Intelligence Tools* (*IJAIT*), 13, 961–979.

South, M., Vreeswijk, G., and Fox, J. (2008), 'Dungine: A Java Dung Reasoner', in *Proceedings of the 2nd Conference on Computational Models of Argument* (*COMMA 2008*), eds. P. Besnard, S. Doutre, and A. Hunter, Vol. 172 of *Frontiers in Artificial Intelligence and Applications*, Toulouse, France: IOS Press, pp. 360–368.

van Emden, M., and Kowalski, R. (1976), 'The Semantics of Predicate Logic as a Programming Language', *Journal of the ACM*, 23, 733–742.

Verheij, B. (2007), 'A Labeling Approach to the Computation of Credulous Acceptance in Argumentation', in *Proceedings of the 20th International Joint Conference on Artificial Intelligence* (*IJCAI 2007*), ed. M.M. Veloso, Hyderabad, India: AAAI Press, pp. 623–628.

Visser, W. (2008), 'Implementation of Argument-Based Practical Reasoning', Master's thesis, Utrecht University.

Vreeswijk, G. (2006), 'An Algorithm to Compute Minimally Grounded and Admissible Defence Sets in Argument Systems', in *Proceedings of the 1st Conference on Computational Models of Argument* (*COMMA 2006*), eds. P.E. Dunne and T.J.M. Bench-Capon, Vol. 144 of *Frontiers in Artificial Intelligence and Applications*, Liverpool, UK: IOS Press, pp. 109–120.

Wakaki, T., and Nitta, K. (2008), 'Computing Argumentation Semantics in Answer Set Programming', in *New Frontiers in Artificial Intelligence* (*JSAI 2008*), *Conference and Workshops*, eds. H. Hattori, T. Kawamura, T. Idé, M. Yokoo, and Y. Murakami, Vol. 5447 of *LNCS*, Asahikawa, Japan: Springer, pp. 254–269.