Making Use of Advances in Answer-Set Programming for Abstract Argumentation Systems*

Wolfgang Dvořák, Sarah Alice Gaggl, Johannes Wallner, and Stefan Woltran

Institute of Information Systems, Database and Artificial Intelligence Group,
Vienna University of Technology, Favoritenstraße 9-11, 1040 Wien, Austria
EMail: {dvorak, gaggl, wallner, woltran}@dbai.tuwien.ac.at

Abstract. Dung's famous abstract argumentation frameworks represent the core formalism for many problems and applications in the field of argumentation which significantly evolved within the last decade. Recent work in the field has thus focused on implementations for these frameworks, whereby one of the main approaches is to use Answer-Set Programming (ASP). While some of the argumentation semantics can be nicely expressed within the ASP language, others required rather cumbersome encoding techniques. Recent advances in ASP systems, in particular, the `metasp` optimization front-end for the ASP-package `gringo/claspD` provide direct commands to filter answer sets satisfying certain subset-minimality (or -maximality) constraints. This allows for much simpler encodings compared to the ones in standard ASP language. In this paper, we experimentally compare the original encodings (for the argumentation semantics based on preferred, semi-stable, and respectively, stage extensions) with new `metasp` encodings. Moreover, we provide novel encodings for the recently introduced resolution-based grounded semantics. Our experimental results indicate that the `metasp` approach works well in those cases where the complexity of the encoded problem is adequately mirrored within the `metasp` approach.

Keywords: Abstract Argumentation, Answer-Set Programming, Meta Programming

1 Introduction

In Artificial Intelligence (AI), the area of argumentation (the survey by Bench-Capon and Dunne [3] gives an excellent overview) has become one of the central issues during the last decade. Although there are now several branches within this area, there is a certain agreement that Dung's famous abstract argumentation frameworks (AFs) [7] still represent the core formalism for many of the problems and applications in the field. In a nutshell, AFs formalize statements together with a relation denoting rebuttals between them, such that the semantics gives a handle to solve the inherent conflicts between statements by selecting admissible subsets of them, but without taking the concrete contents of the statements into account. Several semantical principles how to select those subsets have already been proposed by Dung [7] but numerous other proposals

* This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT08-028.

have been made over the last years. In this paper we shall focus on the preferred [7], semi-stable [4], stage [18], and the resolution-based grounded semantics [1]. Each of these semantics is based on some kind of $\subseteq$-maximality (resp. -minimality) and thus is well amenable for the novel `metasp` concepts which we describe below.

Let us first talk about the general context of the paper, which is the realization of abstract argumentation within the paradigm of Answer-Set Programming (see [17] for an overview). We follow here the ASPARTIX[1] approach [11], where a single program is used to encode a particular argumentation semantics, while the instance of an argumentation framework is given as an input database. For problems located on the second level of the polynomial hierarchy (i.e. for preferred, stage, and semi-stable semantics) ASP encodings turned out to be quite complicated and hardly accessible for non-experts in ASP (we will sketch here the encoding for the stage semantics in some detail, since it has not been presented in [11]). This is due to the fact that tests for subset-maximality have to be done "by hand" in ASP requiring a certain saturation technique. However, recent advances in ASP solvers, in particular, the `metasp` optimization front-end for the ASP-system `gringo/claspD` allows for much simpler encodings for such tests. More precisely, `metasp` allows to use the traditional $\#minimize$ statement (which in its standard variant minimizes wrt. cardinality or weights, but not wrt. subset inclusion) also for selection among answer sets which are minimal wrt. subset inclusion in certain predicates. Details about `metasp` can be found in [13].

Our first main contribution will be the practical comparison between handcrafted encodings (i.e. encodings in the standard ASP language without the new semantics for the $\#minimize$ statement) and the much simpler `metasp` encodings for argumentation semantics. The experiments show that the `metasp` encodings do not necessarily result in longer runtimes. In fact, the `metasp` encodings for the semantics located on the second level of the polynomial hierarchy outperform the handcrafted saturation-based encodings. We thus can give additional evidence to the observations in [13], where such a speed-up was reported for encodings in a completely different application area.

Our second contribution is the presentation of ASP encodings for the resolution-based grounded semantics [1]. To the best of our knowledge, no implementation for this recently proposed semantics has been released so far. In this paper, we present a rather involved handcrafted encoding (basically following the NP-algorithm presented in [1]) but also two much simpler encodings (using `metasp`) which rely on the original definition of the semantics.

Our results indicate that `metasp` is a very useful tool for problems known to be hard for the second-level, but one might loose performance in case `metasp` is used for "easier" problems just for the sake of comfortability. Nonetheless, we believe that the concept of the advanced $\#minimize$ statement is vital for ASP, since it allows for rapid prototyping of second-level encodings without being an ASP guru.

The remainder of the paper is organized as follows: Section 2 provides the necessary background. Section 3 then contains the ASP encodings for the argumentation semantics we are interested in this work. We begin with the handcrafted saturation-based encoding for stage semantics. Then, in Section 3.2 we provide the novel `metasp` encodings for all considered semantics and afterwards, in Section 3.3, we present an alter-

---
[1] See http://rull.dbai.tuwien.ac.at:8080/ASPARTIX for a web front-end of ASPARTIX.

native encoding for the resolution-based grounded semantics which better mirrors the complexity of this semantics. Section 4 then presents our experimental evaluation. We conclude the paper with a brief summary and discussion for future research directions.

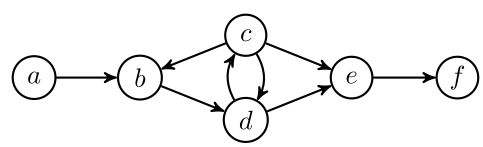
2 Background

2.1 Abstract Argumentation

In this section we introduce (abstract) argumentation frameworks [7] and recall the semantics we study in this paper (see also [1, 2]). Moreover, we highlight complexity results for typical decision problems associated to such frameworks.

Definition 1. *An* argumentation framework (AF) *is a pair $F = (A, R)$ where A is a set of arguments and $R \subseteq A \times A$ is the attack relation. The pair $(a, b) \in R$ means that a attacks b. An argument $a \in A$ is* defended *by a set $S \subseteq A$ if, for each $b \in A$ such that $(b, a) \in R$, there exists a $c \in S$ such that $(c, b) \in R$.*

Example 1. Consider the AF $F = (A, R)$ with $A = \{a, b, c, d, e, f\}$ and $R = \{(a, b), (b, d), (c, b), (c, d), (c, e), (d, c), (d, e), (e, f)\}$, and the graph representation of F:



Semantics for argumentation frameworks are given via a function σ which assigns to each AF $F = (A, R)$ a set $\sigma(F) \subseteq 2^A$ of extensions. We shall consider here for σ the functions stb, adm, com, prf, grd, grd^*, stg, and sem which stand for stable, admissible, complete, preferred, grounded, resolution-based grounded, stage, and semi-stable semantics respectively. Towards the definition of these semantics we have to introduce two more formal concepts.

Definition 2. *Given an AF $F = (A, R)$. The characteristic function $\mathcal{F}_F : 2^A \Rightarrow 2^A$ of F is defined as $\mathcal{F}_F(S) = \{x \in A \mid x \text{ is defended by } S\}$. Moreover, for a set $S \subseteq A$, we denote the set of arguments attacked by S as $S^{\oplus}_R = \{x \mid \exists y \in S \text{ such that } (y, x) \in R\}$, and define the* range *of S as $S^+_R = S \cup S^{\oplus}_R$.*

Definition 3. *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is* conflict-free (in F)*, if there are no $a, b \in S$, such that $(a, b) \in R$. $cf(F)$ denotes the collection of conflict-free sets of F. For a conflict-free set $S \in cf(F)$, it holds that*

- *$S \in stb(F)$, if $S^+_R = A$;*
- *$S \in adm(F)$, if $S \subseteq \mathcal{F}_F(S)$;*
- *$S \in com(F)$, if $S = \mathcal{F}_F(S)$;*
- *$S \in grd(F)$, if $S \in com(F)$ and there is no $T \in com(F)$ with $T \subset S$;*
- *$S \in prf(F)$, if $S \in adm(F)$ and there is no $T \in adm(F)$ with $T \supset S$;*
- *$S \in sem(F)$, if $S \in adm(F)$ and there is no $T \in adm(F)$ with $T^+_R \supset S^+_R$;*

– $S \in stg(F)$, *if there is no* $T \in cf(F)$ *in* F*, such that* $T_R^+ \supset S_R^+$.

We recall that for each AF F, the grounded semantics yields a unique extension, the grounded extension, which is the least fixed point of the characteristic function $\mathcal{F}_F$.

Example 2. Consider the AF F from Example 1. We have $\{a, d, f\}$ and $\{a, c, f\}$ as the stable extensions and thus $stb(F) = stg(F) = sem(F) = \{\{a, d, f\}, \{a, c, f\}\}$. The admissible sets of F are $\{\}$, $\{a\}$, $\{c\}$, $\{a, c\}$, $\{a, d\}$, $\{c, f\}$, $\{a, c, f\}$, $\{a, d, f\}$ and therefore $prf(F) = \{\{a, c, f\}$,$\{a, d, f\}\}$. Finally we have $com(F) = \{\{a\}$, $\{a, c, f\}$, $\{a, d, f\}\}$, with $\{a\}$ being the grounded extension.

On the base of these semantics one can define the family of resolution-based semantics [1], with the resolution-based grounded semantics being the most popular instance.

Definition 4. *A resolution* $\beta \subset R$ *of an AF* $F = (A, R)$ *contains exactly one attack from each bidirectional attack in* F*, i.e.* $\forall a, b \in A$*, if* $(a, b), (b, a) \in R$ *then* $\mid \{(a, b), (b, a)\} \cap \beta \mid = 1$ *and* $\{(c, d) \mid (c, d) \in R, (d, c) \notin R\} \cap \beta = \emptyset$*. A set* $S \subseteq A$ *is a* resolution-based grounded extension *of* F*, denoted by* $S \in grd^*(F)$*, if (i) there exists a resolution* β *such that* $S = grd((A, R \setminus \beta))$*;*[2] *and (ii) there is no resolution* β' *such that* $grd((A, R \setminus \beta')) \subset S$.

Example 3. Recall the AF $F = (A, F)$ from Example 1. There is one mutual attack and thus we have two resolutions $\beta_1 = \{(c, d)\}$ and $\beta_2 = \{(d, c)\}$. Definition 4 gives us two candidates, namely $grd((A, R \setminus \beta_1)) = \{a, d, f\}$ and $grd((A, R \setminus \beta_2)) = \{a, c, f\}$; as they are not in $\subset$-relation they are the resolution-based grounded extensions of F.

We now turn to the complexity of reasoning in AFs. To this end, we define the following decision problems for the semantics σ introduced in Definitions 3 and 4:

– *Credulous Acceptance* Cred_σ: Given AF $F = (A, R)$ and an argument $a \in A$. Is a contained in some $S \in \sigma(F)$?
– *Skeptical Acceptance* Skept_σ: Given AF $F = (A, R)$ and an argument $a \in A$. Is a contained in each $S \in \sigma(F)$?
– *Verification of an extension* Ver_σ: Given AF $F = (A, R)$ and a set of arguments $S \subseteq A$. Is $S \in \sigma(F)$?

We assume the reader has knowledge about standard complexity classes like P and NP and recall that Σ_2^P is the class of decision problems that can be decided in polynomial time using a nondeterministic Turing machine with access to an NP-oracle. The class Π_2^P is defined as the complementary class of Σ_2^P, i.e. $\Pi_2^P = \mathrm{co}\Sigma_2^P$.

In Table 1 we summarize complexity results relevant for our work [1, 6, 8–10].

2.2 Answer-Set Programming

We give a brief overview of the syntax and semantics of disjunctive logic programs under the answer-sets semantics [15]; for further background, see [16].

[2] Abusing notation slightly, we use $grd(F)$ for denoting the unique grounded extension of F.

Table 1. Complexity of abstract argumentation ($\mathcal{C}$-c denotes completeness for class $\mathcal{C}$)

	prf	sem	stg	grd^*
Cred_σ	NP-c	Σ_2^P-c	Σ_2^P-c	NP-c
Skept_σ	Π_2^P-c	Π_2^P-c	Π_2^P-c	coNP-c
Ver_σ	coNP-c	coNP-c	coNP-c	in P

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

A *(disjunctive) rule* r with $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$ is of the form

$$a_1 \ \vee \ \cdots \ \vee \ a_n \ \leftarrow b_1, \ldots, b_k, \ \mathit{not}\, b_{k+1}, \ldots, \ \mathit{not}\, b_m$$

where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms, and "not" stands for *default negation*. An atom a is a positive literal, while $\mathit{not}\, a$ is a default negated literal. The *head* of r is the set $H(r)$ = $\{a_1, \ldots, a_n\}$ and the *body* of r is $B(r) = B^+(r) \cup B^-(r)$ with $B^+(r)$ = $\{b_1, \ldots, b_k\}$ and $B^-(r)$ = $\{b_{k+1}, \ldots, b_m\}$. A rule r is *normal* if $n \leq 1$ and a *constraint* if $n = 0$. A rule r is *safe* if each variable in r occurs in $B^+(r)$. A rule r is *ground* if no variable occurs in r. A *fact* is a ground rule without disjunction and with an empty body. An *(input) database* is a set of facts. A program is a finite set of disjunctive rules. For a program π and an input database D, we often write $\pi(D)$ instead of $D \cup \pi$. If each rule in a program is normal (resp. ground), we call the program normal (resp. ground). Besides disjunctive and normal program, we consider here the class of optimization programs, i.e. normal programs which additionally contain $\#\mathit{minimize}$ statements

$$\#\mathit{minimize}[l_1 = w_1@J_1, \ldots, l_k = w_k@J_k]$$

where l_i is a literal, w_i an integer weight and J_i an integer priority level.

For any program π, let U_π be the set of all constants appearing in π. $Gr(\pi)$ is the set of rules $r\tau$ obtained by applying, to each rule $r \in \pi$, all possible substitutions τ from the variables in r to elements of U_π. An *interpretation* $I \subseteq B_\mathcal{U}$ *satisfies* a ground rule r iff $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. I satisfies a ground program π, if each $r \in \pi$ is satisfied by I. A non-ground rule r (resp., a program π) is satisfied by an interpretation I iff I satisfies all groundings of r (resp., $Gr(\pi)$). $I \subseteq B_\mathcal{U}$ is an *answer set* of π iff it is a subset-minimal set satisfying the *Gelfond-Lifschitz reduct* $\pi^I = \{H(r) \leftarrow B^+(r) \mid I \cap B^-(r) = \emptyset, r \in Gr(\pi)\}$. For a program π, we denote the set of its answer sets by $\mathcal{AS}(\pi)$.

For semantics of optimization programs, we interpret the $\#\mathit{minimize}$ statement wrt. subset-inclusion: For any sets X and Y of atoms, we have $Y \subseteq_J^w X$, if for any weighted literal $l = w@J$ occurring in (2.2), $Y \models l$ implies $X \models l$. Then, M is a collection of relations of the form $\subseteq_J^w$ for priority levels J and weights w. A standard answer set (i.e. not taking the minimize statements into account) Y of π *dominates* a

Table 2. Data Complexity for logic programs (all results are completeness results).

e	normal programs	disjunctive program	optimization programs
$\models_c$	NP	Σ_2^P	Σ_2^P
$\models_s$	coNP	Π_2^P	Π_2^P

standard answer set X of π wrt. M if there are a priority level J and a weight w such that $X \subseteq_J^w Y$ does not hold for $\subseteq_J^w \in M$, while $Y \subseteq_{J'}^{w'} X$ holds for all $\subseteq_{J'}^{w'} \in M$ where $J' \geq J$. Finally a standard answer set X is an answer set of an optimization program π wrt. M if there is no standard answer set Y of π that dominates X wrt. M.

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

We briefly recall some complexity results for disjunctive logic programs. In fact, since we will deal with fixed programs we focus on results for data complexity. Depending on the concrete definition of $\models$, we give the complexity results in Table 2 (cf. [5] and the references therein). We note here, that even normal programs together with the optimization technique have a worst case complexity of Σ_2^P (resp. Π_2^P). Inspecting Table 1 one can see which kind of encoding is appropriate for an argumentation semantics.

3 Encodings of AF Semantics

In this section we first show how to represent AFs in ASP and we discuss three programs which we need later on in this section[3]. Then, in Subsection 3.1 we exemplify on the stage semantics the saturation technique for encodings that solve associated problems which are on the second level of the polynomial hierarchy. In Subsection 3.2 we will make use of the newly developed `metasp` optimization technique. In Subsection 3.3 we give an alternative encoding based on the algorithm by Baroni *et al.* in [1], which respects the lower complexity of resolution-based grounded semantics.

All our programs are fixed which means that the only translation required, is to give an AF F as input database $\hat{F}$ to the program π_σ for a semantics σ. In fact, for an AF $F = (A, R)$, we define $\hat{F}$ as

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

In what follows, we use unary predicates $\operatorname{in}/1$ and $\operatorname{out}/1$ to perform a guess for a set $S \subseteq A$, where $\operatorname{in}(a)$ represents that $a \in S$. The following notion of correspondence is relevant for our purposes.

Definition 5. *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 $\{a \mid \operatorname{in}(a) \in I\} = S$; (ii) for each $I \in \mathcal{I}$, it holds that $\{a \mid \operatorname{in}(a) \in I\} \in \mathcal{S}$; and (iii) $|\mathcal{S}| = |\mathcal{I}|$.*

[3] We make use of some program modules already defined in [11].

Consider an AF F. 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:

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

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

Sometimes we have to avoid the use of negation. This might either be the case for the saturation technique or if a simple program can be solved without a Guess&Check approach. Then, encodings typically rely on a form of loops where all domain elements are visited and it is checked whether a desired property holds for all elements visited so far. We will use this technique in our saturation-based encoding in the upcoming subsection, but also for computing the grounded extension in Subsection 3.2. For this purpose, an order $<$ over the domain elements (usually provided by common ASP solvers) is used together with a few helper predicates defined in the program $\pi_<$ below; in fact, predicates $\mathrm{inf}/1, \mathrm{succ}/2$ and $\mathrm{sup}/1$ denote infimum, successor and supremum of the order $<$.

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

Finally, the following module computes for a guessed subset $S \subseteq A$ the range S_R^+ (see Def. 2) of S in an AF (A, R).

$$\begin{aligned}\pi_{range} = \{ \; & \mathrm{in\_range}(X) \leftarrow \mathrm{in}(X); \\ & \mathrm{in\_range}(X) \leftarrow \mathrm{in}(Y), \mathrm{defeat}(Y, X); \\ & \mathrm{not\_in\_range}(X) \leftarrow \mathrm{arg}(X), \mathit{not}\, \mathrm{in\_range}(X) \; \}.\end{aligned}$$

3.1 Saturation Encodings

In this subsection we make use of the saturation technique introduced by Eiter and Gottlob in [12]. In [11], this technique was already used to encode the preferred and semi-stable semantics. Here we give the encodings for the stage semantics, which is similar to the one of semi-stable semantics, to exemplify the use of the saturation technique.

In fact, for an AF $F = (A, R)$ and $S \in cf(F)$ we need to check whether no $T \in cf(F)$ with $S_R^+ \subset T_R^+$ exists. Therefore we have to guess an arbitrary set T and saturate in case (i) T is not conflict-free, or (ii) $S_R^+ \not\subset T_R^+$. Together with π_{cf} this is done with the following module, where $\mathrm{in}/1$ holds the current guess for S and $\mathrm{inN}/1$ holds the current guess for T. More specifically, rule $\mathrm{fail} \leftarrow \mathrm{inN}(X), \mathrm{inN}(Y), \mathrm{defeat}(X, Y)$ checks for (i) and the remaining two rules with fail in the head fire in case $S_R^+ = T_R^+$

(indicated by predicate $\mathrm{eqplus}/0$ described below), or there exists an $a \in S_R^+$ such that $a \notin T_R^+$ (here we use predicate $\mathrm{in\_range}/1$ from above and predicate $\mathrm{not\_in\_rangeN}/1$ which we also present below). As is easily checked one of these two conditions holds exactly if (ii) holds.

$$
\begin{aligned}
\pi_{satstage} = \{\ & \mathrm{inN}(X) \vee \mathrm{outN}(X) \leftarrow \mathrm{arg}(X);\\
& \mathrm{fail} \leftarrow \mathrm{inN}(X), \mathrm{inN}(Y), \mathrm{defeat}(X,Y);\\
& \mathrm{fail} \leftarrow \mathrm{eqplus};\\
& \mathrm{fail} \leftarrow \mathrm{in\_range}(X), \mathrm{not\_in\_rangeN}(X);\\
& \mathrm{inN}(X) \leftarrow \mathrm{fail}, \mathrm{arg}(X);\\
& \mathrm{outN}(X) \leftarrow \mathrm{fail}, \mathrm{arg}(X);\\
& \leftarrow \mathit{not}\, \mathrm{fail}\ \}.
\end{aligned}
$$

For the definition of predicates $\mathrm{not\_in\_rangeN}/1$ and $\mathrm{eqplus}/0$ we make use of the aforementioned loop technique and predicates from program $\pi_<$.

$$
\begin{aligned}
\pi_{rangeN} = \{\ & \mathrm{undefeated\_upto}(X,Y) \leftarrow \mathrm{inf}(Y), \mathrm{outN}(X), \mathrm{outN}(Y);\\
& \mathrm{undefeated\_upto}(X,Y) \leftarrow \mathrm{inf}(Y), \mathrm{outN}(X), \mathit{not}\, \mathrm{defeat}(Y,X);\\
& \mathrm{undefeated\_upto}(X,Y) \leftarrow \mathrm{succ}(Z,Y), \mathrm{undefeated\_upto}(X,Z),\\
& \qquad \mathrm{outN}(Y);\\
& \mathrm{undefeated\_upto}(X,Y) \leftarrow \mathrm{succ}(Z,Y), \mathrm{undefeated\_upto}(X,Z),\\
& \qquad \mathit{not}\, \mathrm{defeat}(Y,X);\\
& \mathrm{not\_in\_rangeN}(X) \leftarrow \mathrm{sup}(Y), \mathrm{outN}(X), \mathrm{undefeated\_upto}(X,Y);\\
& \mathrm{in\_rangeN}(X) \leftarrow \mathrm{inN}(X);\\
& \mathrm{in\_rangeN}(X) \leftarrow \mathrm{outN}(X), \mathrm{inN}(Y), \mathrm{defeat}(Y,X)\ \}.
\end{aligned}
$$

$$
\begin{aligned}
\pi_{eq}^+ = \{\ & \mathrm{eqp\_upto}(X) \leftarrow \mathrm{inf}(X), \mathrm{in\_range}(X), \mathrm{in\_rangeN}(X);\\
& \mathrm{eqp\_upto}(X) \leftarrow \mathrm{inf}(X), \mathrm{not\_in\_range}(X), \mathrm{not\_in\_rangeN}(X);\\
& \mathrm{eqp\_upto}(X) \leftarrow \mathrm{succ}(Z,X), \mathrm{in\_range}(X), \mathrm{in\_rangeN}(X), \mathrm{eqp\_upto}(Z);\\
& \mathrm{eqp\_upto}(X) \leftarrow \mathrm{succ}(Y,X), \mathrm{not\_in\_range}(X), \mathrm{not\_in\_rangeN}(X),\\
& \qquad \mathrm{eqp\_upto}(Y);\\
& \mathrm{eqplus} \leftarrow \mathrm{sup}(X), \mathrm{eqp\_upto}(X)\ \}.
\end{aligned}
$$

Proposition 2. *For any AF F, $stg(F) \cong \mathcal{AS}(\pi_{stg}(\hat{F}))$, where $\pi_{stg} = \pi_{cf} \cup \pi_< \cup \pi_{range} \cup \pi_{rangeN} \cup \pi_{eq}^+ \cup \pi_{satstage}$.*

3.2 Meta ASP Encodings

The following encodings for preferred, semi-stable and stage semantics are written using the $\#minimize$ statement when evaluated with the subset-minimization semantics provided by `metasp`. For our encodings we do not need prioritization and weights, therefore these are omitted (i.e. set to default) in the minimization statements. The minimization technique is realized through meta programming techniques, which themselves are answer-set programs. This works as follows: The ASP encoding to solve is given to the grounder `gringo` which reifies the program, i.e. outputs a ground program consisting of facts, which represent the rules and facts of the original input encoding. The grounder is then again executed on this output with the meta programs which encode the optimization. Finally, claspD computes the answer sets. Note that here we use the version of clasp which supports disjunctive rules. Therefore for a program π and an AF F we have the following execution.

```
gringo --reify π(F̂) | \
    gringo - {meta.lp,metaO.lp,metaD.lp} \
    <(echo "optimize(1,1,incl).") | claspD 0
```

Here, meta.lp, metaO.lp and metaD.lp are the encodings for the minimization statement. The statement optimize(incl,1,1) indicates that we use subset inclusion for the optimization technique using priority and weight 1.

We now look at the encodings for the preferred, semi-stable and stage semantics using this minimization technique. First, we need one auxiliary module for admissible extensions.

$$\pi_{adm} = \pi_{cf} \cup \{\text{defeated}(X) \leftarrow \text{in}(Y), \text{defeat}(Y, X);\\ \leftarrow \text{in}(X), \text{defeat}(Y, X), \textit{not}\ \text{defeated}(Y)\}.$$

Now the modules for preferred, semi-stable and stage semantics are easy to encode using the minimization statement of metasp. For the preferred semantics we take the module π_{adm} and minimize the $\text{out}/1$ predicate. This in turn gives us the subset-maximal admissible extensions which captures the definition of preferred semantics. The encodings for the semi-stable and stage semantics are similar. Here we minimize the predicate $\text{not\_in\_range}/1$ from the π_{range} module.

$$\pi_{prf\_metasp} = \pi_{adm} \cup \{\#\textit{minimize}[\text{out}]\}.$$
$$\pi_{sem\_metasp} = \pi_{adm} \cup \pi_{range} \cup \{\#\textit{minimize}[\text{not\_in\_range}]\}.$$
$$\pi_{stg\_metasp} = \pi_{cf} \cup \pi_{range} \cup \{\#\textit{minimize}[\text{not\_in\_range}]\}.$$

The following results follow now directly.

Proposition 3. *For any AF F, we have*

1. $\textit{prf}(F) \cong \mathcal{AS}(\pi_{prf\_metasp}(\hat{F}))$,
2. $\textit{sem}(F) \cong \mathcal{AS}(\pi_{sem\_metasp}(\hat{F}))$, *and*
3. $\textit{stg}(F) \cong \mathcal{AS}(\pi_{stg\_metasp}(\hat{F}))$.

Next we give two different encodings for computing resolution-based grounded extensions. Both encodings use subset-minimization for the resolution part, i.e. the resulting extension is subset-minimal with respect to all possible resolutions. The difference between the two encodings is that the first one computes the grounded extension for the guessed resolution explicitly (making use of looping concepts presented already in [11]). The second encoding uses the metasp subset-minimization also to get the grounded extension from the complete extensions of the current resolution (recall that the grounded extension is in fact the unique subset-minimal complete extension). The module π_{grd} below for computing the grounded extension is taken from [11] with a small modification: instead of the defeat predicate we use defeat_minus_beta, since we need the grounded extensions of a restricted defeat relation. In fact, the π_{res} module guesses this restricted defeat relation $\{R \setminus \beta\}$ for a resolution β.

$$\pi_{res} = \{ \text{defeat\_minus\_beta}(X,Y) \leftarrow \text{defeat}(X,Y), \textit{not}\ \text{defeat\_minus\_beta}(Y,X), X \neq Y;$$
$$\text{defeat\_minus\_beta}(X,Y) \leftarrow \text{defeat}(X,Y), \textit{not}\ \text{defeat}(Y,X);$$
$$\text{defeat\_minus\_beta}(X,X) \leftarrow \text{defeat}(X,X)\}.$$

We repeat the definition of π_{grd} here, which includes the module $\pi_{defended}$.

$$\pi_{defended} = \{ \text{defended\_upto}(X,Y) \leftarrow \text{inf}(Y), \text{in}(X), \textit{not}\ \text{defeat\_minus\_beta}(Y,X);$$
$$\text{defended\_upto}(X,Y) \leftarrow \text{inf}(Y), \text{in}(Z), \text{defeat\_minus\_beta}(Z,Y), \text{defeat\_minus\_beta}(Y,X);$$
$$\text{defended\_upto}(X,Y) \leftarrow \text{succ}(Z,Y), \text{defended\_upto}(X,Z), \textit{not}\ \text{defeat\_minus\_beta}(Y,X);$$
$$\text{defended\_upto}(X,Y) \leftarrow \text{succ}(Z,Y), \text{in}(V), \text{defeat\_minus\_beta}(V,Y), \text{defeat\_minus\_beta}(Y,X);$$
$$\text{defended}(X) \leftarrow \text{sup}(Y), \text{defended\_upto}(X,Y)\}.$$
$$\pi_{grd} = \pi_{<} \cup \pi_{defended} \cup \{\text{in}(X) \leftarrow \text{defended}(X)\}.$$

Now we can give the first encoding for resolution-based grounded semantics.

$$\pi_{grd^*\_metasp} = \pi_{grd} \cup \pi_{res} \cup \{\#\textit{minimize}[\text{in}]\}.$$

The second encoding for resolution-based grounded semantics performs the `metasp` subset-minimization from the complete extensions of the current resolution to compute the grounded extension (recall that the grounded extension is in fact the unique subset-minimal complete extension). We again use the restricted defeat relation.

$$\pi_{com} = \pi_{adm} \cup \{ \text{undefended}(X) \leftarrow \text{defeat\_minus\_beta}(Y,X), \textit{not}\ \text{defeated}(Y);$$
$$\leftarrow \text{out}(X), \textit{not}\ \text{undefended}(X) \}.$$

We obtain the following `metasp` encoding:

$$\pi'_{grd^*\_metasp} = \pi_{com} \cup \pi_{res} \cup \{\#\textit{minimize}[\text{in}]\}.$$

Proposition 4. *For any AF F and $\pi \in \{\pi_{grd^*\_metasp}, \pi'_{grd^*\_metasp}\}$, $grd^*(F)$ corresponds to $\mathcal{AS}(\pi(\hat{F}))$ in the sense of Definition 5, but without property (iii).*

As the proposition suggests there is a caveat for these two encodings of the resolution-based grounded semantics. In general we have that several answer sets map to the same extension, i.e. there is no one-to-one correspondence between answer sets and extensions. The reason for this behavior lies in the guessing of a resolution. Whereas the other encodings guess basically the in/1 predicate, these two `metasp` encodings guess the resolution. Therefore the result might include the same extension with different resolutions guessed. While this does not harm credulous or skeptical reasoning, some measures have to be taken to remove these duplicates when enumerating or counting extensions. The solver `clasp` already features such a technique which is presented in [14]. This feature is not yet implemented in `claspD`. Furthermore the meta encodings for `metasp` use disjunctive ASP, which increases the computational complexity to the second level of the polynomial hierarchy, whereas the problem of resolution based grounded semantics is situated on the first level.

3.3 Alternative Encodings for Resolution-based Grounded Semantics

So far, we have shown two encodings for the resolution-based grounded semantics via optimization programs, i.e. we made use of the $\#minimize$ statement under the subset-inclusion semantics. From the complexity point of view this is not adequate, since we expressed a problem on the NP-layer (see Table 1) via an encoding which implicitly makes use of disjunction (see Table 2 for the actual complexity of optimization programs). Hence, we provide here an alternative encoding for the resolution-based grounded semantics based on the verification algorithm proposed by Baroni *et al.* in [1]. This encoding is just a normal program and thus located at the right level of complexity.

We need some further notation. For an AF $F = (A, R)$ and a set $S \subseteq A$ we define $F|_S = ((A \cap S), R \cap (S \times S))$ as the *sub-framework* of F wrt. S; furthermore we also use $F - S$ as a shorthand for $F|_{A \setminus S}$. By $SCCs(F)$, we denote the set of strongly connected components of an AF $F = (A, R)$ which identify the vertices of a maximal strongly connected[4] subgraph of F; $SCCs(F)$ is thus a partition of A. A partial order $\prec_F$ over $SCCs(F) = \{C_1, \dots, C_n\}$, denoted as $(C_i \prec_F C_j)$ for $i \neq j$, is defined, if $\exists x \in C_i, y \in C_j$ such that there is a directed path from x to y in F.

Definition 6. *A $C \in SCCs(F)$ is* minimal relevant *(in an AF F) iff C is a minimal element of $\prec_F$ and $F|_C$ satisfies the following:*

(a) the attack relation $R(F|_C)$ of F is irreflexive, i.e. $(x, x) \notin R(F|_C)$ for all arguments x;
(b) $R(F|_C)$ is symmetric, i.e. $(x, y) \in R(F|_C) \Leftrightarrow (y, x) \in R(F|_C)$;
(c) the undirected graph *obtained by replacing each (directed) pair $\{(x, y), (y, x)\}$ in $F|_C$ with a single undirected edge $\{x, y\}$ is acyclic.*

The set of minimal relevant SCCs in F is denoted by $MR(F)$.

Proposition 5 ([1]). *Given an AF $F = (A, R)$ such that $(F - S^+_R) \neq (\emptyset, \emptyset)$ and $MR(F - S^+_R) \neq \emptyset$, where $S = grd(F)$, a set $U \subseteq A$ of arguments is* resolution-based grounded *in F, i.e. $U \in grd^*(F)$ iff the following conditions hold:*

(i) $U \cap S^+_R = S$;
(ii) $(T \cap \Pi_F) \in stb(F|_{\Pi_F})$, where $T = U \setminus S^+_R$, and $\Pi_F = \bigcup_{V \in MR(F - S^+_R)} V$;
(iii) $(T \cap \Pi^C_F) \in grd^*(F|_{\Pi^C_F} - (S^+_R \cup (T \cap \Pi_F)^{\oplus}_R))$, where T and Π_F are as in (ii) and $\Pi^C_F = A \setminus \Pi_F$.

To illustrate the conditions of Proposition 5, let us have a look at our example.

Example 4. Consider the AF F of Example 1. Let us check whether $U = \{a, d, f\}$ is resolution-based grounded in F, i.e. whether $U \in grd^*(F)$. $S = \{a\}$ is the grounded extension of F and $S^+_R = \{a, b\}$, hence the Condition (i) is satisfied. We obtain $T = \{d, f\}$ and $\Pi_F = \{c, d\}$. We observe that $T \cap \Pi_F = \{d\}$ is a stable extension of the AF $F|_{\Pi_F}$; that satisfies Condition (ii). Now we need to check Condition (iii); we first identify the necessary sets: $\Pi^C_F = \{a, b, e, f\}$, $T \cap \Pi^C_F = \{f\}$ and $(T \cap \Pi_F)^{\oplus}_R = \{c, e\}$. It remains to check $\{f\} \in grd^*(\{f\}, \emptyset)$ which is easy to see. Hence, $U \in grd^*(F)$.

[4] A directed graph is called *strongly connected* if there is a directed path from each vertex in the graph to every other vertex of the graph.

The following encoding is based on the Guess&Check procedure which was also used for the encodings in [11]. After guessing all conflict-free sets with the program π_{cf}, we check whether the conditions of Definition 6 and Proposition 5 hold. Therefore the program π_{arg_set} makes a copy of the actual arguments, defeats and the guessed set to the predicates $\mathrm{arg\_set}/2, \mathrm{defeatN}/3$ and $\mathrm{inU}/2$. The first variable in these three predicates serves as an identifier for the iteration of the algorithm (this is necessary to handle the recursive nature of Proposition 5). In all following predicates we will use the first variable of each predicate like this. As in some previous encodings in this paper, we use the program $\pi_<$ to obtain an order over the arguments, and we start our computation with the infimum represented by the predicate $\mathrm{inf}/1$.

$$
\begin{aligned}
\pi_{arg\_set} = \{\; & \mathrm{arg\_set}(N,X) \leftarrow \mathrm{arg}(X), \mathrm{inf}(N);\\
& \mathrm{inU}(N,X) \leftarrow \mathrm{in}(X), \mathrm{inf}(N);\\
& \mathrm{defeatN}(N,Y,X) \leftarrow \mathrm{arg\_set}(N,X), \mathrm{arg\_set}(N,Y), \mathrm{defeat}(Y,X) \;\}.
\end{aligned}
$$

We use here the program $\pi_{defendedN}$ (which is a slight variant of the program $\pi_{defended}$) together with the program $\pi_{groundN}$ where we perform a fixed-point computation of the predicate $\mathrm{defendedN}/2$, as in the definition of the characteristic function $\mathcal{F}_F$ in Definition 2. The basic difference here is that now, we use an additional argument N for the iteration step where predicates $\mathrm{arg\_set}/2$, $\mathrm{defeatN}/3$ and $\mathrm{inS}/2$ replace $\mathrm{arg}\,/1$, $\mathrm{defeat}/2$ and $\mathrm{in}/1$.

$$
\begin{aligned}
\pi_{defendedN} = \{\; & \mathrm{def\_uN}(N,X,Y) \leftarrow \mathrm{inf}(Y), \mathrm{arg\_set}(N,X), \mathit{not}\; \mathrm{defeatN}(N,Y,X);\\
& \mathrm{def\_uN}(N,X,Y) \leftarrow \mathrm{inf}(Y), \mathrm{inS}(N,Z), \mathrm{defeatN}(N,Z,Y),\\
& \qquad\qquad \mathrm{defeatN}(N,Y,X);\\
& \mathrm{def\_uN}(N,X,Y) \leftarrow \mathrm{succ}(Z,Y), \mathit{not}\; \mathrm{defeatN}(N,Y,X),\\
& \qquad\qquad \mathrm{def\_uN}(N,X,Z);\\
& \mathrm{def\_uN}(N,X,Y) \leftarrow \mathrm{succ}(Z,Y), \mathrm{def\_uN}(N,X,Z), \mathrm{inS}(N,V),\\
& \qquad\qquad \mathrm{defeatN}(N,V,Y), \mathrm{defeatN}(N,Y,X)\\
& \mathrm{defendedN}(N,X) \leftarrow \mathrm{sup}(Y), \mathrm{def\_uN}(N,X,Y) \;\}.
\end{aligned}
$$

In $\pi_{groundN}$ we then obtain the predicate $\mathrm{inS}(N,X)$ which identifies argument X to be in the grounded extension of the iteration N.

$$
\pi_{groundN} = \pi_{cf} \cup \pi_< \cup \pi_{arg\_set} \cup \pi_{defendedN} \cup \{\; \mathrm{inS}(N,X) \leftarrow \mathrm{defendedN}(N,X) \;\}.
$$

The next module $\pi_{F\_minus\_range}$ computes the arguments in $(F - S_R^+)$, represented by the predicate $\mathrm{notInSplusN}/2$, via predicates $\mathrm{in\_SplusN}/2$ and $\mathrm{u\_cap\_Splus}/2$ (for S_R^+ and $U \cap S_R^+$). The two constraints check condition (i) of Proposition 5.

$$
\begin{aligned}
\pi_{F\_minus\_range} = \{\; & \mathrm{in\_SplusN}(N,X) \leftarrow \mathrm{inS}(N,X);\\
& \mathrm{in\_SplusN}(N,X) \leftarrow \mathrm{inS}(N,Y), \mathrm{defeatN}(N,Y,X);\\
& \mathrm{u\_cap\_Splus}(N,X) \leftarrow \mathrm{inU}(N,X), \mathrm{in\_SplusN}(N,X);\\
& \leftarrow \mathrm{u\_cap\_Splus}(N,X), \mathit{not}\; \mathrm{inS}(N,X);\\
& \leftarrow \mathit{not}\; \mathrm{u\_cap\_Splus}(N,X), \mathrm{inS}(N,X);\\
& \mathrm{notInSplusN}(N,X) \leftarrow \mathrm{arg\_set}(N,X), \mathit{not}\; \mathrm{in\_SplusN}(N,X) \;\}.
\end{aligned}
$$

The module π_{MR} computes $\mathit{\Pi}_F = \bigcup_{V \in MR(F - S_R^+)} V$, where $\mathrm{mr}(N,X)$ denotes that an argument is contained in a set $V \in MR$. Therefore we need to check all three conditions of Definition 6. The first two rules compute the predicate $\mathrm{reach}(N,X,Y)$ if there is a path between the arguments $X,Y \in (F - S_R^+)$. With this predicate we will identify the SCCs. The third rule computes $\mathrm{self\_defeat}/2$ for all arguments violating Condition (a). Next we need to check Condition (b). With $\mathrm{nsym}/2$ we obtain those arguments which do not have a symmetric attack to any other argument from the same component. Condition (c) is a bit more tricky. With predicate $\mathrm{reachnotvia}/4$ we say that there is a path from X to Y not going over argument V in the framework $(F - S_R^+)$. With this predicate at hand we can check for cycles with $\mathrm{cyc}/4$. Then, to complete Condition (c) we derive $\mathrm{bad}/2$ for all arguments which are connected to a cycle (or a self-defeating argument). In the predicate $\mathrm{pos\_mr}/2$, we put all the three conditions together and say that an argument x is possibly in a set $V \in MR$ if (i) $x \in (F - S_R^+)$, (ii) x is neither connected to a cycle nor self-defeating, and (iii) for all y it holds that $(x,y) \in (F - S_R^+) \Leftrightarrow (y,x) \in (F - S_R^+)$. Finally we only need to check if the SCC obtained with $\mathrm{pos\_mr}/2$ is a minimal element of $\prec_F$. Hence we get with $\mathrm{notminimal}/2$ all arguments not fulfilling this, and in the last rule we obtain with $\mathrm{mr}/2$ the arguments contained in a minimal relevant SCC.

$$
\begin{aligned}
\pi_{MR} = \{\ & \mathrm{reach}(N,X,Y) \leftarrow \mathrm{notInSplusN}(N,X), \mathrm{notInSplusN}(N,Y), \mathrm{defeatN}(N,X,Y);\\
& \mathrm{reach}(N,X,Y) \leftarrow \mathrm{notInSplusN}(N,X), \mathrm{defeatN}(N,X,Z), \mathrm{reach}(N,Z,Y),\\
& \qquad X! = Y;\\
& \mathrm{self\_defeat}(N,X) \leftarrow \mathrm{notInSplusN}(N,X), \mathrm{defeatN}(N,X,X);\\
& \mathrm{nsym}(N,X) \leftarrow \mathrm{notInSplusN}(N,X), \mathrm{notInSplusN}(N,Y), \mathrm{defeatN}(N,X,Y),\\
& \qquad \mathit{not}\ \mathrm{defeatN}(N,Y,X), \mathrm{reach}(N,X,Y), \mathrm{reach}(N,Y,X), X! = Y;\\
& \mathrm{nsym}(N,Y) \leftarrow \mathrm{notInSplusN}(N,X), \mathrm{notInSplusN}(N,Y), \mathrm{defeatN}(N,X,Y),\\
& \qquad \mathit{not}\ \mathrm{defeatN}(N,Y,X), \mathrm{reach}(N,X,Y), \mathrm{reach}(N,Y,X), X! = Y;\\
& \mathrm{reachnotvia}(N,X,V,Y) \leftarrow \mathrm{defeatN}(N,X,Y), \mathrm{notInSplusN}(N,V),\\
& \qquad \mathrm{reach}(N,X,Y), \mathrm{reach}(N,Y,X), X! = V, Y! = V;\\
& \mathrm{reachnotvia}(N,X,V,Y) \leftarrow \mathrm{reachnotvia}(N,X,V,Z), \mathrm{reach}(N,X,Y),\\
& \qquad \mathrm{reachnotvia}(N,Z,V,Y), \mathrm{reach}(N,Y,X),\\
& \qquad Z! = V, X! = V, Y! = V;\\
& \mathrm{cyc}(N,X,Y,Z) \leftarrow \mathrm{defeatN}(N,X,Y), \mathrm{defeatN}(N,Y,X),\\
& \qquad \mathrm{defeatN}(N,Y,Z), \mathrm{defeatN}(N,Z,Y),\\
& \qquad \mathrm{reachnotvia}(N,X,Y,Z), X! = Y, Y! = Z, X! = Z;\\
& \mathrm{bad}(N,Y) \leftarrow \mathrm{cyc}(N,X,U,V), \mathrm{reach}(N,X,Y), \mathrm{reach}(N,Y,X);\\
& \mathrm{bad}(N,Y) \leftarrow \mathrm{self\_defeat}(N,X), \mathrm{reach}(N,X,Y), \mathrm{reach}(N,Y,X);\\
& \mathrm{bad}(N,Y) \leftarrow \mathrm{nsym}(N,X), \mathrm{reach}(N,X,Y), \mathrm{reach}(N,Y,X);\\
& \mathrm{pos\_mr}(N,X) \leftarrow \mathrm{notInSplusN}(N,X), \mathit{not}\ \mathrm{bad}(N,X), \mathit{not}\ \mathrm{self\_defeat}(N,X),\\
& \qquad \mathit{not}\ \mathrm{nsym}(N,X);\\
& \mathrm{notminimal}(N,Z) \leftarrow \mathrm{reach}(N,X,Y), \mathrm{reach}(N,Y,X),\\
& \qquad \mathrm{reach}(N,X,Z), \mathit{not}\ \mathrm{reach}(N,Z,X);\\
& \mathrm{mr}(N,X) \leftarrow \mathrm{pos\_mr}(N,X), \mathit{not}\ \mathrm{notminimal}(N,X)\ \}.
\end{aligned}
$$

We now turn to Condition (ii) of Proposition 5, where the first rule in $\pi_{stableN}$ computes the set $T = U \setminus S_R^+$. Then we check whether $T = \emptyset$ and $MR(F - S_R^+) = \emptyset$ via predicates $\mathrm{emptyT}/1$ and $\mathrm{not\_exists\_mr}/1$. If this is so, we terminate the iteration in the last module $\pi_{iterate}$. The first constraint eliminates those guesses where $MR(F - S_R^+) = \emptyset$ but $T \neq \emptyset$, because the algorithm is only defined for AFs fulfilling this. Finally we derive the arguments which are defeated by the set T in the MR denoted by $\mathrm{defeated}/2$, and with the last constraint we eliminate those guesses where there is an argument not contained in T and not defeated by T in MR and hence $(T \cap \Pi_F) \notin stb(F|_{\Pi_F})$.

$$
\begin{aligned}
\pi_{stableN} = \{\ & \mathrm{t}(N,X) \leftarrow \mathrm{inU}(N,X), \mathit{not}\,\mathrm{inS}(N,X);\\
& \mathrm{nemptyT}(N) \leftarrow \mathrm{t}(N,X);\\
& \mathrm{emptyT}(N) \leftarrow \mathit{not}\,\mathrm{nemptyT}(N), \mathrm{arg\_set}(N,X);\\
& \mathrm{existsMR}(N) \leftarrow \mathrm{mr}(N,X), \mathrm{notInSplusN}(N,X);\\
& \mathrm{not\_exists\_mr}(N) \leftarrow \mathit{not}\,\mathrm{existsMR}(N), \mathrm{notInSplusN}(N,X);\\
& \mathrm{true}(N) \leftarrow \mathrm{emptyT}(N), \mathit{not}\,\mathrm{existsMR}(N);\\
& \leftarrow \mathrm{not\_exists\_mr}(N), \mathrm{nemptyT}(N);\\
& \mathrm{defeated}(N,X) \leftarrow \mathrm{mr}(N,X), \mathrm{mr}(N,Y), \mathrm{t}(N,Y), \mathrm{defeatN}(N,Y,X);\\
& \leftarrow \mathit{not}\,\mathrm{t}(N,X), \mathit{not}\,\mathrm{defeated}(N,X), \mathrm{mr}(N,X)\ \}.
\end{aligned}
$$

With the last module $\pi_{iterate}$ we perform Step (iii) of Proposition 5. The predicate $\mathrm{t\_mrOplus}/2$ computes the set $(T \cap \Pi_F)_R^{\oplus}$ and with the second rule we start the next iteration for the AF $(F|_{\Pi_F^C} - (S_R^+ \cup (T \cap \Pi_F)_R^{\oplus}))$ and the set $(T \cap \Pi_F^C)$.

$$
\begin{aligned}
\pi_{iterate} = \{\ & \mathrm{t\_mrOplus}(N,Y) \leftarrow \mathrm{t}(N,X), \mathrm{mr}(N,X), \mathrm{defeatN}(N,X,Y);\\
& \mathrm{arg\_set}(M,X) \leftarrow \mathrm{notInSplusN}(N,X), \mathit{not}\,\mathrm{mr}(N,X),\\
& \qquad \mathit{not}\,\mathrm{t\_mrOplus}(N,X), \mathrm{succ}(N,M), \mathit{not}\,\mathrm{true}(N);\\
& \mathrm{inU}(M,X) \leftarrow \mathrm{t}(N,X), \mathit{not}\,\mathrm{mr}(N,X), \mathrm{succ}(N,M), \mathit{not}\,\mathrm{true}(N)\ \}.
\end{aligned}
$$

Finally we put everything together and obtain the program π_{grd^*}.

$$\pi_{grd^*} = \pi_{groundN} \cup \pi_{F\_minus\_range} \cup \pi_{MR} \cup \pi_{stableN} \cup \pi_{iterate}.$$

Proposition 6. *For any AF F, $grd^*(F) \cong \mathcal{AS}(\pi_{grd^*}(\hat{F}))$.*

4 Experimental Evaluation

In this section we present our results of the performance evaluation. We compared the time needed for computing all extensions for the semantics described earlier, using both the handcraft saturation-based and the alternative `metasp` encodings.

The tests were executed on an openSUSE based machine with eight Intel Xeon processors (2.33 GHz) and 49 GB memory. For computing the answer sets, we used `gringo` (version 3.0.3) for grounding and the solver `claspD` (version 1.1.1). The latter being the variant for disjunctive answer-set programs.

We randomly generated AFs (i.e. graphs) ranging from 20 to 110 arguments. We used two parametrized methods for generating the attack relation. The first generates arbitrary AFs and inserts for any pair (a, b) the attack from a to b with a given probability p. The other method generates AFs with an $n \times m$ grid-structure. We consider two different neighborhoods, one connecting arguments vertically and horizontally and one that additionally connects the arguments diagonally. Such a connection is a mutual

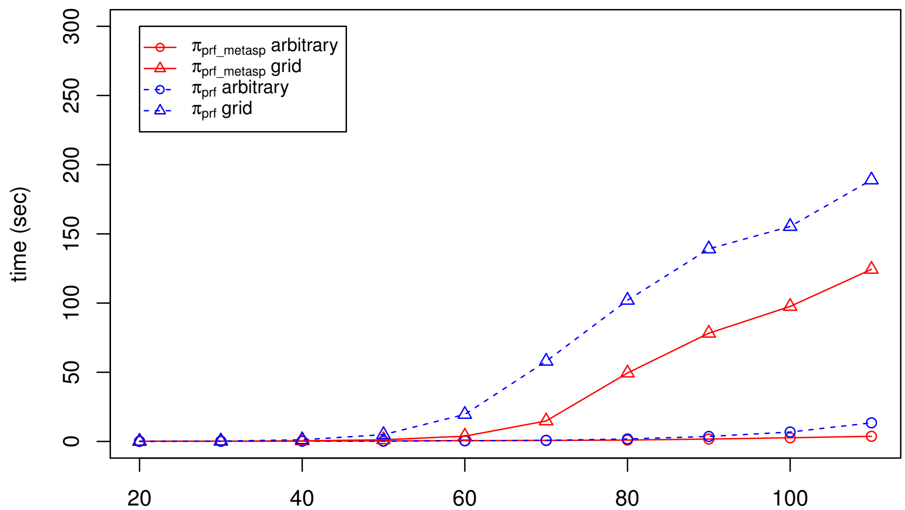


Fig. 1. Average computation time for preferred semantics.

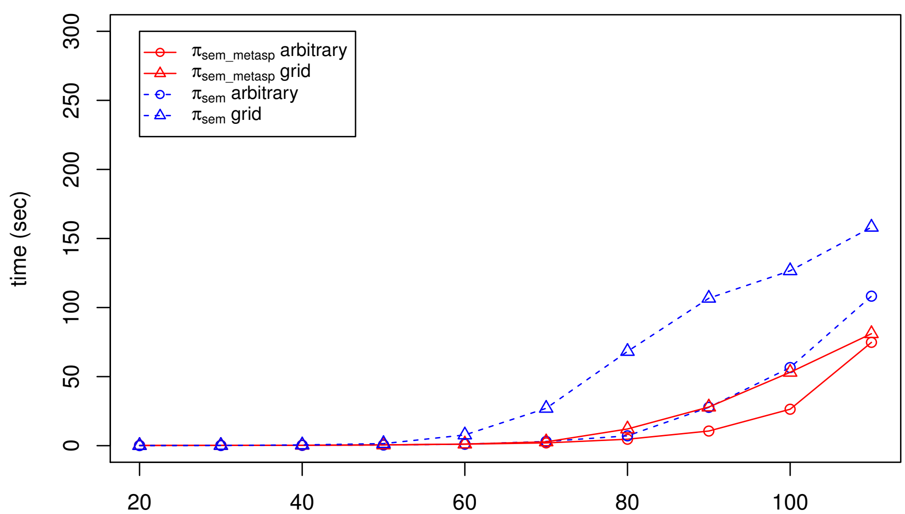


Fig. 2. Average computation time for semi-stable semantics.

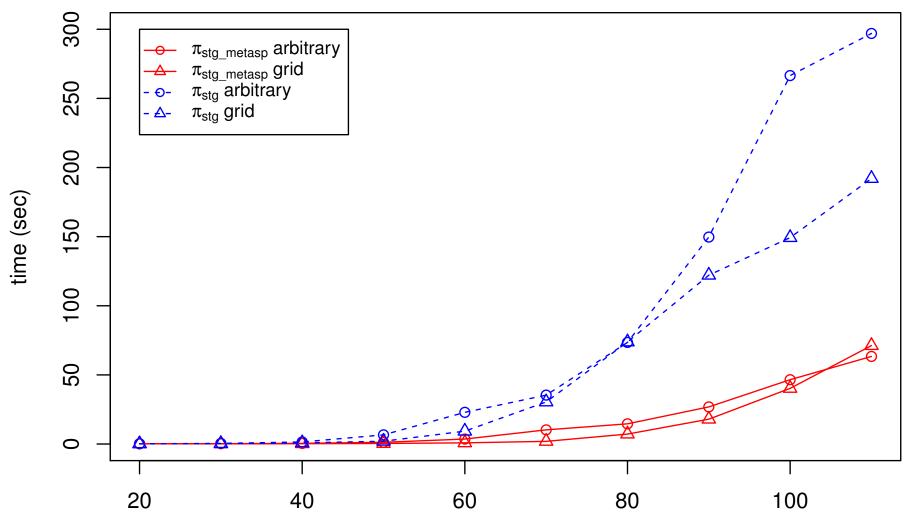


Fig. 3. Average computation time for stage semantics.

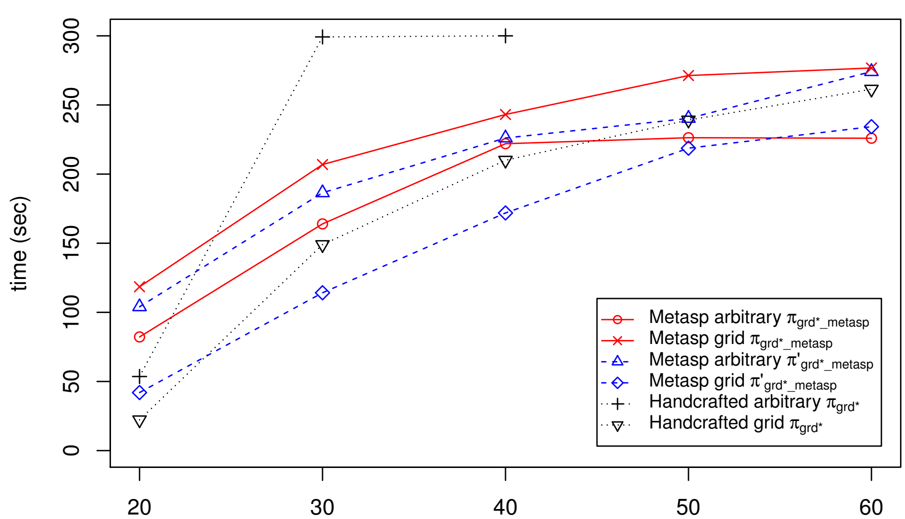


Fig. 4. Average computation time for resolution-based grounded semantics

attack with a given probability p and in only one direction otherwise. The probability p was chosen between 0.1 and 0.4.

Overall 14388 tests were executed, with a timeout of five minutes for each execution. Timed out instances are considered as solved in 300 seconds. The time consumption was measured using the Linux `time` command. For all the tests we let the solver generate all answer sets, but only outputting the number of models. To minimize external influences on the test runs, we alternated the different encodings during the tests.

Figures 1 – 3 depict the results for the preferred, semi-stable and stage semantics respectively. The figures show the average computation time for both the handcraft and the `metasp` encoding for a certain number of arguments. We distinguish here between arbitrary, i.e. completely random AFs and grid structured ones. One can see that the `metasp` encodings have a better performance, compared to the handcraft encodings. In particular, for the stage semantics the performance difference is noticeable. Recall that the average computation time includes the timeouts, which strongly influence the diagrams.

For the resolution-based grounded semantics, Figure 4 shows again the average computation time needed for a certain number of arguments. Let us first consider the case of arbitrary AFs. The handcraft encoding struggled with AFs of size 40 or larger. Many of those instances could not be solved due to memory faults. This is indicated by the missing data points. Both `metasp` encodings performed better overall, but still many timeouts were encountered. If we look more closely at the structured AFs then we see that $\pi'_{grd^*\_metasp}$ performs better overall than the other `metasp` variant. Interestingly, computing the grounded part with a handcraft encoding without a Guess&Check part did not result in a lower computation time on average. The handcraft encoding performed better than $\pi_{grd^*\_metasp}$ on grids.

One reason for the performance problems of the handcraft encoding lies in the relatively high arity of some predicates. The encoding uses predicates with up to four variables, in contrast to the encoding for e.g. the stage semantics which needs only predicates with up to three variables. This can increase the time needed for grounding drastically. On the other side, the `metasp` encodings, as mentioned in Proposition 4, suffer from the fact that the answer sets are not in a one-to-one correspondence to the solutions, i. e. several answer sets may represent the same extension.

Overall the `metasp` encodings outperform the direct encodings. This is partially due to the fact that the former utilize encodings tailored to the `gringo/claspD` package.

5 Conclusion

In this paper, we inspected various ASP encodings for four prominent semantics in the area of abstract argumentation. (1) For the preferred and the semi-stable semantics, we compared existing saturation-based encodings [11] (here we called them handcrafted encodings) with novel alternative encodings which are based on the recently developed `metasp` approach [13], where subset-minimization can be directly specified and a front-end (i.e. a meta-interpreter) compiles such statements back into the core ASP language. (2) For the stage semantics, we presented here both a handcrafted and a `metasp` encoding. Finally, (3) for the resolution-based grounded semantics we provided three novel encodings, two of them using the `metasp` techniques.

While with some performance optimization techniques for ASP the readability of the encodings change for the worse, the `metasp` encodings are shorter than the handcrafted saturation encodings. Furthermore, they are much simpler to design (since saturation techniques are delegated to the meta-interpreter), and they perform surprisingly well when compared with the handcrafted encodings which are directly given to the ASP solver. This shows the practical relevance of the `metasp` technique also in the area of abstract argumentation. Future work will be to investigate performance improvements of other optimization features like aggregates, which are provided by most of the prominent ASP solvers.

References

1. P. Baroni, P. E. Dunne, and M. Giacomin. On the resolution-based family of abstract argumentation semantics and its grounded instance. *Artif. Intell.*, 175(3-4):791–813, 2011.
2. P. Baroni and M. Giacomin. Semantics of abstract argument systems. *Argumentation in Artif. Intell.*, pages 25–44. Springer, 2009.
3. T. J. M. Bench-Capon and P. E. Dunne. Argumentation in artificial intelligence. *Artif. Intell.*, 171(10-15):619–641, 2007.
4. M. Caminada. Semi-stable semantics. In Proc. *COMMA 2006*, pages 121–130, 2006.
5. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.
6. Y. Dimopoulos and A. Torres. Graph theoretical structures in logic programs and default theories. *Theor. Comput. Sci.*, 170(1-2):209–244, 1996.
7. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.
8. P. E. Dunne and T. J. M. Bench-Capon. Coherence in finite argument systems. *Artif. Intell.*, 141(1/2):187–203, 2002.
9. P. E. Dunne and M. Caminada. Computational complexity of semi-stable semantics in abstract argumentation frameworks. In Proc. *JELIA 2008*, pages 153–165, 2008.
10. W. Dvořák and S. Woltran. Complexity of semi-stable and stage semantics in argumentation frameworks. *Inf. Process. Lett.*, 110(11):425–430, 2010.
11. U. Egly, S. A. Gaggl, and S. Woltran. Answer-set programming encodings for argumentation frameworks. *Argument and Computation*, 1(2):147–177, 2010.
12. T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.*, 15(3-4):289–323, 1995.
13. M. Gebser, R. Kaminski, and T. Schaub. Complex optimization in answer set programming. *Theory and Practice of Logic Programming* 11(4-5): 821–839 (2011).
14. M. Gebser, B. Kaufmann and T. Schaub. Solution Enumeration for Projected Boolean Search Problems. Proc. *CPAIOR 2009*, pages 71–86, 2009.
15. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.
16. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.
17. F. Toni and M. Sergot. Argumentation and answer set programming. *Logic Programming, Knowledge Representation, and Nonmon. Reasoning*, *LNAI* 6565:164–180. Springer, 2011.
18. B. Verheij. Two approaches to dialectical argumentation: admissible sets and argumentation stages. In Proc. *NAIC'96*, pages 357–368, 1996.