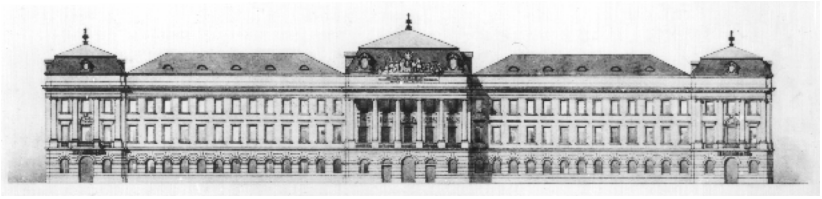
dbai

TECHNICAL REPORT



INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG DATENBANKEN UND ARTIFICIAL INTELLIGENCE

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

**DBAI-TR-2011-70**

**Wolfgang Dvořák** **Sarah Alice Gaggl**
**Johannes Wallner** **Stefan Woltran**

Institut für Informationssysteme
Abteilung Datenbanken und
Artificial Intelligence
Technische Universität Wien
Favoritenstr. 9
A-1040 Vienna, Austria
Tel: +43-1-58801-18403
Fax: +43-1-58801-18493
sekret@dbai.tuwien.ac.at
www.dbai.tuwien.ac.at

DBAI TECHNICAL REPORT
2011

TU WIEN
TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DBAI TECHNICAL REPORT
DBAI TECHNICAL REPORT DBAI-TR-2011-70, 2011

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

**Wolfgang Dvořák** [1] **Sarah Alice Gaggl** [1]
**Johannes Wallner** [1] **Stefan Woltran** [1]

**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 frontend for the ASP-package `gringo/claspD` provides 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, thus evaluating the efficiency of the novel `metasp` technique. 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.

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

**Acknowledgements**: This work was Supported by the Vienna Science and Technology Fund (WWTF) under grant ICT08-028.

# 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 have been made over the last years. In this paper we shall focus on the preferred [7], semi-stable [4], stage [19], and the resolution-based grounded semantics [1].

Each of these semantics is based on some kind of subset-maximality (resp. -minimality) and thus it is well amenable for the novel `metasp` concepts which we describe below. Among these semantics we distinguish two classes of semantics: On the one hand, the first three semantics mentioned are among the hardest for abstract argumentation, i.e. decision problems (as credulous or skeptical acceptance) are located on the this makes them suitable candidates to employ the novel `metasp` concepts. On the other hand, for the resolution-based grounded semantics we are not aware of any existing implementation; we will present (three alternative) novel ASP encodings for this semantics in the course of this paper.

Let us thus now turn to the main context of the paper, which is the realization of abstract argumentation within the paradigm of Answer-Set Programming (see [18] for a recent overview on this topic). More specifically, we follow here the ASPARTIX approach [11], where a single program (or query) is used to encode a particular argumentation semantics, while the instance of an argumentation framework is given just as an input database. Many semantics have already been encoded in this way[1], most of them discussed in [11]. 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 (which dates back to the original $\Sigma_2^P$-hardness proof for disjunctive ASP [12]; see [11] for its application to argumentation encodings). Recent advances in ASP solvers, in particular, the `metasp` optimization frontend for the ASP-system `gringo/clasp` allows for much simpler encodings for such tests. More precisely, `metasp` allows to use the traditional $\#minimize$ statement (which in its standard variant minimizes by cardinality or weights, but not by subset inclusion) also for selection among answer-sets which are minimal (or maximal) wrt. subset-inclusion in certain predicates. Details about `metasp` can be found in [15].

Our first main contribution will be the practical comparison between handcraft encodings (i.e. encodings in the standard ASP language without the new semantics for the $\#minimize$ statement)

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

and the much simpler `metasp` encodings for argumentation semantics. The experiments show that the simpler `metasp` encodings do not necessarily result in longer runtimes. In turn, for semantics located on the second level of the polynomial hierarchy, the `metasp` encodings outperform the handcraft saturation-based encodings. We thus can give additional evidence to the observations in [15], 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 quite interesting semantics has been released so far. In this paper, we present a rather involved handcraft encoding (basically following the algorithm presented in [1]) but also two much simpler encodings (using `metasp`) which closely follow the original definition of the semantics. However, there is an important difference to the former three semantics: while preferred, stage, and semi-stable semantics are located on the second level of the polynomial hierarchy, this is not true for the resolution-based grounded semantics (which remains on the NP level). However, using `metasp` internally results in a disjunctive ASP program (and thus falls into a class of second-level complexity), while our handcraft encoding here is free of disjunction. This jump of complexity is mirrored in our experiments which are more opaque than for the other three semantics.

These two contributions hence suggest 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. We believe that the illustrated application of abstract argumentation demonstrates the practical usefulness of this `metasp` approach, but further experiments from other areas still have to be undertaken.

The remainder of the paper is organized as follows: Section 2 provides the necessary background; first on abstract argumentation, then on Answer-Set Programming. Section 3 then contains the ASP encodings for the semantics we are interested in here. We first discuss the handcraft saturation-based encoding for stage semantics (the ones for preferred and semi-stable are similar and already published; so we decided to omit them here). Then, in Section 3.2 we provide the novel `metasp` encodings for all considered semantics (including two variants for the resolution-based grounded semantics). Afterwards, in Section 3.3 we finally present an alternative 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.

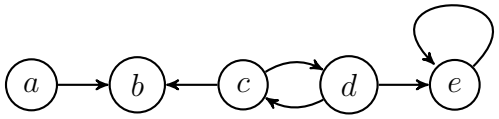
# 2 Background

## 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 2.1** *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 2.2** *Consider the AF $F = (A, R)$, 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 given as follows:*




Semantics for argumentation frameworks are given via a function $\sigma$ which assigns to each AF $F = (A, R)$ a set $\sigma(F) \subseteq 2^A$ of extensions. We shall consider here for $\sigma$ the functions $stb$, $adm$, $prf$, $com$, $grd$, $grd^*$, $stg$, and $sem$ which stand for stable, admissible, preferred, complete, 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 2.3** *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 $S_R^{\oplus} = \{x \mid \exists y \in S \text{ such that } (y, x) \in R\}$, and define the* range of $S$ *as $S_R^+ = S \cup S_R^{\oplus}$.*

We are now ready to define semantics for AFs:

**Definition 2.4** *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is* conflict-free (in $F$)*, iff 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 such a conflict-free set $S$, 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 $T_R^+ \supset S_R^+$;*

*For all semantics $\sigma$, the sets defined above are the only ones in $\sigma(F)$.*

We recall that for each AF $F$, $stb(F) \subseteq sem(F) \subseteq prf(F) \subseteq com(F) \subseteq adm(F)$ holds, and that for each of the considered semantics $\sigma$ except stable semantics, $\sigma(F) \neq \emptyset$ holds. Moreover the grounded semantics always proposes an unique extension, the grounded extension, which is the least fix-point of the characteristic function $\mathcal{F}_F$.

**Example 2.5** *Recall the AF $F$ from example 2.2. We have that $\{a, d\}$ is a stable extension and thus that $stb(F) = stg(F) = sem(F) = \{\{a, d\}\}$. The admissible sets of $F$ are $\{\}$, $\{a\}$, $\{c\}$, $\{d\}$, $\{a, c\}$, $\{a, d\}$, and therefore $prf(F) = \{\{a, c\},\{a, d\}\}$. Finally we have $com(F) = \{\{a\},\{a, c\},\{a, d\}\}$, with $\{a\}$ being the grounded extension.*

On the base of these semantics one can define the family of resolution based semantics (see [1]), with the resolution-based grounded semantics being the most popular instance. Towards a definition of resolution-based grounded semantics we briefly introduce the concept of resolutions in AFs.

**Definition 2.6** *Given AF $F = (A, R)$. A (full) resolution $\beta \subset R$ of $F$ is a set of attacks such that $(a, b) \in \beta$ iff $(b, a) \notin \beta$ and $\{(a, b), (b, a)\} \subseteq R$.*

We are now ready to define resolution-based grounded extensions.

**Definition 2.7** *A set $S \subseteq A$ is a resolution-based grounded extension of $F$ if (i) there exists a resolution $\beta$ such that $grd((A, R\backslash\beta)) = S$ [2]; and (ii) there is no resolution $\beta'$ such that $grd((A, R\backslash\beta')) \subset S$.*

**Example 2.8** *Recall the AF from example 2.2. There is one mutual attack and thus we have two resolutions $\beta_1 = \{(c, d)\}$ and $\beta_2 = \{(d, c)\}$. Condition (i) of Definition 2.7 gives us two candidates for resolution-based grounded extensions, namely $\{a, d\}$ and $\{a, c\}$, and as they are not in $\subset$-relation they are indeed resolution-based grounded extensions.*

We now turn to the complexity of reasoning in AFs. To this end, we define the following decision problems for the semantics $\sigma$ introduced in Definitions 2.4 and 2.7.

- *Credulous Acceptance* $\mathsf{Cred}_\sigma$: Given AF $F = (A, R)$ and an argument $a \in A$. Is $a$ contained in some $S \in \sigma(F)$?
- *Skeptical Acceptance* $\mathsf{Skept}_\sigma$: Given AF $F = (A, R)$ and an argument $a \in A$. Is $a$ contained in each $S \in \sigma(F)$?
- *Verification of an extension* $\mathsf{Ver}_\sigma$: Given AF $F = (A, R)$ and a set of arguments $S \subseteq A$. Is $S \in \sigma(F)$?

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

We assume the reader has knowledge about standard complexity classes, i.e. P, NP and LOGSPACE (L), but we briefly recapitulate the concept of oracle machines and the complexity classes $\Sigma_2^P, \Pi_2^P$. By a NP-oracle machine we mean a Turing machine which can access an oracle, that decides a given (sub)-problem in the class NP within one step. We define $\Sigma_2^P$, as 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 $\Pi_2^P$ is defined as the complementary class of $\Sigma_2^P$, i.e. $\Pi_2^P = \text{co}\Sigma_2^P$.

In Table 1 we summarize complexity results relevant for our work [1, 6, 8, 9, 10].

| | *prf* | *sem* | *stg* | $grd^*$ |
|---|---|---|---|---|
| $\mathsf{Cred}_\sigma$ | NP-c | $\Sigma_2^P$-c | $\Sigma_2^P$-c | NP-c |
| $\mathsf{Skept}_\sigma$ | $\Pi_2^P$-c | $\Pi_2^P$-c | $\Pi_2^P$-c | coNP-c |
| $\mathsf{Ver}_\sigma$ | in L | in L | in L | in P |

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

## 2.2 Answer-Set Programming

In this section, we first give a brief overview of the syntax and semantics of disjunctive logic programs under the answer-sets semantics [16]; for further background, see [13, 17].

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

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

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

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

$$\#minimize[l_1 = w_1@J_1, \ldots, l_k = w_k@J_k], \tag{1}$$

| $e$ | normal programs | disjunctive program | optimization programs |
|---|---|---|---|
| $\models_c$ | NP | $\Sigma_2^P$ | $\Sigma_2^P$ |
| $\models_s$ | coNP | $\Pi_2^P$ | $\Pi_2^P$ |

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

where $l_i$ is a default literal, $w_i$ an integer weight and $J_i$ provides an integer priority level for $1 \leq i \leq k$.

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

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

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

For a program $\mathcal{P}$, we denote the set of its answer sets by $\mathcal{AS}(\mathcal{P})$.

The semantics of optimization programs, we require here, follows the novel `metasp` technique, and we consider here the interpretation of the $\#minimize$ 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 (1), $Y \models l$ implies $X \models l$. Then, $M$ is a collection of relations of the form $\subseteq_J^w$ for priority levels $J$ and weights $w$. An answer set $Y$ of $\mathcal{P}$ *dominates* an answer set $X$ of $\mathcal{P}$ 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 we obtain that $X$ is an answer set of the optimization program $\mathcal{P}$ wrt. $M$ if there is no answer set $Y$ of $\mathcal{P}$ that dominates $X$ wrt. $M$.

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

We briefly recall some complexity results for disjunctive logic programs. In fact, since we will deal with fixed programs we focus on results for data complexity. Recall that data complexity in our context is the complexity of checking whether $\mathcal{P}(D) \models A$ when disjunctive logic programs $\mathcal{P}$ are fixed, while input databases $D$ and ground atoms $A$ are an input of the decision problem. Depending on the concrete definition of $\models$, we give the complexity results in Table 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 $\Sigma_2^P$ (resp. $\Pi_2^P$).

Inspecting now Table 1 one can directly see which encoding is appropriate for which argumentation semantics.

# 3 Encodings of AF Semantics

In this section we first show how to represent AFs in ASP and we give three programs which we need later on in this section [3]. The first one $\pi_{cf}$ opens the search space for our solutions via two guessing rules and eliminates all guesses which are not conflict-free. The second program $\pi_{<}$ defines an order over the domain elements, and the third one $\pi_{range}$ computes the range of a set $S$. Then, in Subsection 3.1 we exemplify on the stage semantics the saturation technique for encodings which 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 to encode the preferred, semi-stable, stage and resolution-based grounded semantics without (explicitly) using the quite complicated saturation technique. As the resolution-based grounded semantics is on a lower complexity level than the other semantics used here, we give in Subsection 3.3 an alternative encoding based on the algorithm of Baroni *et al.* in [1].

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 $\pi_\sigma$ for a semantics $\sigma$. In fact, for an AF $F = (A, R)$, we define $\hat{F}$ as

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

In what follows, we use unary predicates $\mathrm{in}/1$ and $\mathrm{out}/1$ to perform a guess for a set $S \subseteq A$, where $\mathrm{in}(a)$ 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 $\{a \mid \mathrm{in}(a) \in I\} = S$; (ii) for each $I \in \mathcal{I}$, it holds that $\{a \mid \mathrm{in}(a) \in I\} \in \mathcal{S}$; and (iii) $|\mathcal{S}| = |\mathcal{I}|$.*

Let $F = (A, R)$ be an argumentation framework. 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 $F$:

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

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

For ASP encodings, it is sometimes required or desired 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 the computation of the grounded extension in Subsection 3.2.

[3] We make use of some program modules already defined in [11], for a more detailed explanation we refer to the mentioned paper.

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 program $\pi_<$ below; in fact, predicates $\inf/1, \mathrm{succ}/2$ and $\sup/1$ denote infinum, successor and supremum of the order $<$.

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

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

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

## 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 very similar to the one of semi-stable extensions. The main difference is that for semi-stable extensions the set $S \subseteq A$ needs to be admissible, whereas for stage extensions the set $S$ is only required to be conflict-free. Therefore we obtain the encoding for stage extensions by a slight modification of the encoding for semi-stable extensions from [11].

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, and (ii) $S_R^+ \not\subset T_R^+$. Together with $\pi_{cf}$ 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 $\mathrm{fail}$ 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 this two conditions holds exactly if (ii) holds.

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

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 $\pi_<$ above.

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

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

We define $\pi_{stg} = \pi_{cf} \cup \pi_< \cup \pi_{range} \cup \pi_{rangeN} \cup \pi^{+}_{eq} \cup \pi_{satstage}$ and obtain the following result.

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

## 3.2 Meta ASP Encodings

The following encodings for preferred, semi-stable and stage semantics are written using the $\#minimize[\cdot]$ statement when evaluated with the subset minimization semantics provided by `metasp`. For our encodings we do not need priorization 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 original answer-set program to solve is given to the grounder `gringo`, which reifies the program and outputs its ground version in the form of facts. The grounder is then again executed on this output with the meta programs, which encode the optimization. Lastly `claspD` handles

the solving part. Note that here we use the version of `clasp` which supports disjunctive rules. Therefore for a program $\pi$ 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(1,1,incl)` 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.

$$\begin{array}{rcl} \pi_{adm} & = & \pi_{cf} \cup \{\mathrm{defeated}(X) \leftarrow \mathrm{in}(Y), \mathrm{defeat}(Y,X); \\ & & \qquad \leftarrow \mathrm{in}(X), \mathrm{defeat}(Y,X), \mathit{not}\, \mathrm{defeated}(Y)\}. \end{array}$$

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 admissible module and use the minimization of the $\mathrm{out}$ 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 $\mathrm{not\_in\_range}$ from the $\pi_{range}$ module. The only difference between the semi-stable and stage encoding is that the former uses the admissible module, whereas the latter needs only conflict-freeness.

$$\begin{array}{rcl} \pi_{prf\_metasp} & = & \pi_{adm} \cup \{\#\mathit{minimize}[\mathrm{out}]\}. \\ \pi_{sem\_metasp} & = & \pi_{adm} \cup \pi_{range} \cup \{\#\mathit{minimize}[\mathrm{not\_in\_range}]\}. \\ \pi_{stg\_metasp} & = & \pi_{cf} \cup \pi_{range} \cup \{\#\mathit{minimize}[\mathrm{not\_in\_range}]\}. \end{array}$$

The following results follow now quite directly.

**Proposition 3.4** *For any AF $F$, the following relations hold.*

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

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 additionally 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 $\pi_{grd}$ below for computing the grounded extension is taken from [11] with

a small modification: instead of the $\mathrm{defeat}$ predicate we use $\mathrm{defeat\_minus\_beta}$, since we need the grounded extensions of a restricted $\mathrm{defeat}$ relation. The $\pi_{res}$ module guesses this restricted $\mathrm{defeat}$ relation, i.e. $\{R \setminus \beta\}$ for a resolution $\beta$.

$$\begin{aligned}\pi_{res} = \{\ & \mathrm{defeat\_minus\_beta}(X,Y) \leftarrow \mathrm{defeat}(X,Y), \mathit{not}\ \mathrm{defeat\_minus\_beta}(Y,X), \\ & \qquad\qquad X \neq Y; \\ & \mathrm{defeat\_minus\_beta}(X,Y) \leftarrow \mathrm{defeat}(X,Y), \mathit{not}\ \mathrm{defeat}(Y,X); \\ & \mathrm{defeat\_minus\_beta}(X,X) \leftarrow \mathrm{defeat}(X,X)\}.\end{aligned}$$

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

$$\begin{aligned}\pi_{defended} = \{\ & \mathrm{defended\_upto}(X,Y) \leftarrow \mathrm{inf}(Y), \mathrm{in}(X), \mathit{not}\ \mathrm{defeat\_minus\_beta}(Y,X); \\ & \mathrm{defended\_upto}(X,Y) \leftarrow \mathrm{inf}(Y), \mathrm{in}(Z), \mathrm{defeat\_minus\_beta}(Z,Y), \\ & \qquad\qquad \mathrm{defeat\_minus\_beta}(Y,X); \\ & \mathrm{defended\_upto}(X,Y) \leftarrow \mathrm{succ}(Z,Y), \mathrm{defended\_upto}(X,Z), \\ & \qquad\qquad \mathit{not}\ \mathrm{defeat\_minus\_beta}(Y,X); \\ & \mathrm{defended\_upto}(X,Y) \leftarrow \mathrm{succ}(Z,Y), \mathrm{in}(V), \mathrm{defeat\_minus\_beta}(V,Y), \\ & \qquad\qquad \mathrm{defeat\_minus\_beta}(Y,X); \\ & \mathrm{defended}(X) \leftarrow \mathrm{sup}(Y), \mathrm{defended\_upto}(X,Y)\}.\end{aligned}$$

$$\pi_{grd} = \pi_{<} \cup \pi_{defended} \cup \{\mathrm{in}(X) \leftarrow \mathrm{defended}(X)\}$$

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

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

The second version computes the grounded extensions simply by subset minimization from the complete extensions. We compute the complete extensions again wrt. the restricted $\mathrm{defeat}$ relation.

$$\begin{aligned}\pi_{com} &= \pi_{adm} \cup \{\ \mathrm{undefended}(X) \leftarrow \mathrm{defeat\_minus\_beta}(Y,X), \mathit{not}\ \mathrm{defeated}(Y); \\ &\qquad\qquad \leftarrow \mathrm{out}(X), \mathit{not}\ \mathrm{undefended}(X)\ \}. \\ \pi'_{grd^*\_metasp} &= \pi_{com} \cup \pi_{res} \cup \{\#\mathit{minimize}[\mathrm{in}]\}.\end{aligned}$$

**Proposition 3.5** *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 3.1, but without property (iii).*

As the proposition suggests there is a caveat for these two encodings of the resolution-based grounded semantics. Define for projection of an extension $Pr(X) = \{a \mid \mathrm{in}(a) \in X\}$. For $\pi \in \{\pi_{grd^*\_metasp}, \pi'_{grd^*\_metasp}\}$ in general we have that several $I \in \mathcal{AS}(\pi(\hat{F}))$ map to the same $Pr(I)$, i.e. we have duplicated extensions if we project out only the $\mathrm{in}$ predicate from the answer sets. 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`. The duplicated extensions might also influence computation times, especially for outputting the extensions. The reason for this behavior lies in the guessing of a resolution. Whereas

the other encodings guess basically the $\mathtt{in}/1$ predicate, the two `metasp` encodings guess the resolution. Therefore the result might include the same extension with different resolutions guessed.

## 3.3 Alternative Encodings for Resolution-based Grounded Semantics

In the previous section, 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 thus expressed an 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]. Our new 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] subgraphs of $F$; $SCCs(F)$ is thus a partition of $A$. A partial order $\prec_F$ over $SCCs(F) = \{C_1, \ldots, C_n\}$, 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 direct path from $x$ to $y$ in $F$.

**Definition 3.6** *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 three conditions:*

*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 $MR(F)$.*

**Definition 3.7 ([1])** *Given an AF $F = (A, R)$ such that $(F - S_R^+) \neq (\emptyset, \emptyset)$ and $MR(F - S_R^+) \neq \emptyset$, where $S \in grd(F)$, a set of arguments $U \subseteq A$ is* resolution-based grounded *in $F$, i.e. $U \in grd^*(F)$ iff the following three conditions are satisfied.*

*(i) $U \cap S_R^+ = S$;*

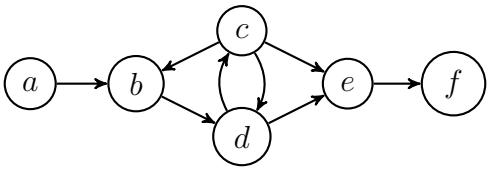
*(ii) $(T \cap \Pi_F) \in stb(F|_{\Pi_F})$, where $T = U \setminus S_R^+$, and $\Pi_F = \bigcup_{V \in MR(F - S_R^+)} V$;*

*(iii) $(T \cap \Pi_F^C) \in grd^*(F|_{\Pi_F^C} - (S_R^+ \cup (T \cap \Pi_F)_R^{\oplus}))$, where $T$ and $\Pi_F$ are as in (ii) and $\Pi_F^C$ stands for $A \setminus \Pi_F$.*

[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.

To illustrate the conditions of Definition 3.7, let us have a look at the following example.

**Example 3.8** *Consider the following AF $F$ over $A = \{a, b, c, d, e, f\}$*




*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 first 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: $\Pi_F^C = \{a, b, e, f\}$, $T \cap \Pi_F^C = \{f\}$ and $(T \cap \Pi_F)_R^{\oplus} = \{c, e\}$. It remains to check $\{f\} \in grd^*(\{f\}, \emptyset)$ which is easy to see. Hence, $U \in grd^*(F)$.*

*Concerning the original defintion of resolution-based grounded semantics (cf. Definition 2.7), observe that we have two possible resolutions for $F$, namely either selecting $(c, d)$ or $(d, c)$. For the former selection, the grounded extension of the resulting framework, is $\{a, c, f\}$, for the latter selection we actually obtain $U$ as the grounded extension. Since $\{a, c, f\} \not\subset U$, $U$ is indeed resolution-based grounded in $F$.*

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 $\pi_{cf}$, we check whether the conditions of Definitions 3.6 and 3.7 hold. Therefore the program $\pi_{arg\_set}$ 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 Definition 3.7). 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 $\mathrm{inf}/1$.

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

In the program $\pi_{defendedN}$ together with the program $\pi_{groundN}$ we perform a fixed-point computation of the predicate $\mathrm{defendedN}/2$ via $\mathrm{def\_uN}/3$, as in the definition of the characteristic function $\mathcal{F}_F$ in Definition 2.3. This is very similar to the rules used in module $\pi_{defended}$ but now we use an additional argument $N$ for the iteration step; on the other hand we do not incorporate resolutions directly here.

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

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 $N$.

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

The next module $\pi_{F\_minus\_range}$ computes the arguments in $(F - S_R^+)$, represented by the predicate $\mathrm{not\_in\_SplusN}/2$, via predicates $\mathrm{in\_SplusN}/2$ and $\mathrm{u\_cap\_Splus}/2$ (for $S_R^+$ and $U \cap S_R^+$). The two constraints in $\pi_{F\_minus\_range}$ check condition (i) of Definition 3.7.

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

The module $\pi_{MR}$ computes $\Pi_F = \bigcup_{V \in MR(F - S_R^+)} V$, 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 3.6. 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} \quad = \{ \quad & \mathrm{reach}(N,X,Y) \leftarrow \mathrm{not\_in\_SplusN}(N,X), \mathrm{not\_in\_SplusN}(N,Y), \mathrm{defeatN}(N,X,Y);\\
& \mathrm{reach}(N,X,Y) \leftarrow \mathrm{not\_in\_SplusN}(N,X), \mathrm{defeatN}(N,X,Z), \mathrm{reach}(N,Z,Y),\\
& \qquad\qquad X! = Y;\\
& \mathrm{self\_defeat}(N,X) \leftarrow \mathrm{not\_in\_SplusN}(N,X), \mathrm{defeatN}(N,X,X);\\
& \mathrm{nsym}(N,X) \leftarrow \mathrm{not\_in\_SplusN}(N,X), \mathrm{not\_in\_SplusN}(N,Y), \mathrm{defeatN}(N,X,Y),\\
& \qquad\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{not\_in\_SplusN}(N,X), \mathrm{not\_in\_SplusN}(N,Y), \mathrm{defeatN}(N,X,Y),\\
& \qquad\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{not\_in\_SplusN}(N,V),\\
& \qquad\qquad\qquad\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\qquad\qquad\qquad \mathrm{reachnotvia}(N,Z,V,Y), \mathrm{reach}(N,Y,X),\\
& \qquad\qquad\qquad\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\qquad\qquad \mathrm{defeatN}(N,Y,Z), \mathrm{defeatN}(N,Z,Y),\\
& \qquad\qquad\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{pos\_mr}(N,X) \leftarrow \mathrm{not\_in\_SplusN}(N,X), \mathit{not}\ \mathrm{bad}(N,X), \mathit{not}\ \mathrm{self\_defeat}(N,X),\\
& \qquad\qquad\qquad \mathit{not}\ \mathrm{nsym}(N,X);\\
& \mathrm{notminimal}(N,Z) \leftarrow \mathrm{reach}(N,X,Y), \mathrm{reach}(N,Y,X),\\
& \qquad\qquad\qquad\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 Definition 3.7, 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 frameworks 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 $(T \cap \Pi_F) \notin \mathit{stb}(F|_{\Pi_F})$.

$$
\begin{aligned}
\pi_{stableN} \quad = \{ \quad & \mathrm{t}(N,X) \leftarrow \mathrm{inU}(N,X), \mathit{not}\ \mathrm{inS}(N,X);\\
& \mathrm{nemptyT}(N) \leftarrow \mathrm{t}(N,X);\\
& \mathrm{emptyT}(N) \leftarrow \mathit{not}\ \mathrm{nemptyT}(N), \mathrm{arg\_set}(N,X);\\
& \mathrm{existsMR}(N) \leftarrow \mathrm{mr}(N,X), \mathrm{not\_in\_SplusN}(N,X);\\
& \mathrm{not\_exists\_mr}(N) \leftarrow \mathit{not}\ \mathrm{existsMR}(N), \mathrm{not\_in\_SplusN}(N,X);
\end{aligned}
$$

$$\begin{aligned}
&\mathrm{true}(N) \leftarrow \mathrm{emptyT}(N), \mathit{not}\,\mathrm{existsMR}(N);\\
&\leftarrow \mathrm{not_exists_mr}(N), \mathrm{nemptyT}(N);\\
&\mathrm{defeated}(N,X) \leftarrow \mathrm{mr}(N,X), \mathrm{mr}(N,Y), \mathrm{t}(N,Y), \mathrm{defeatN}(N,Y,X);\\
&\leftarrow \mathit{not}\,\mathrm{t}(N,X), \mathit{not}\,\mathrm{defeated}(N,X), \mathrm{mr}(N,X)\ \}.
\end{aligned}$$

With the last module $\pi_{iterate}$ we perform step (iii) of Definition 3.7. The predicate $\mathrm{t\_mrOplus}/2$ computes the set $(T \cap \Pi_F)^{\oplus}_R$ and with the second rule we start the next iteration for the framework $(F|_{\Pi_F^C} - (S_R^+ \cup (T \cap \Pi_F)^{\oplus}_R))$ and the set $(T \cap \Pi_F^C)$.

$$\begin{aligned}
\pi_{iterate} \quad = \{ \quad & \mathrm{t_mrOplus}(N,Y) \leftarrow \mathrm{t}(N,X), \mathrm{mr}(N,X), \mathrm{defeatN}(N,X,Y);\\
& \mathrm{arg_set}(M,X) \leftarrow \mathrm{not_in_SplusN}(N,X), \mathit{not}\,\mathrm{mr}(N,X), \mathit{not}\,\mathrm{t_mrOplus}(N,X),\\
& \qquad\qquad \mathrm{succ}(N,M), \mathit{not}\,\mathrm{true}(N);\\
& \mathrm{inU}(M,X) \leftarrow \mathrm{t}(N,X), \mathit{not}\,\mathrm{mr}(N,X), \mathrm{succ}(N,M), \mathit{not}\,\mathrm{true}(N)\ \}.
\end{aligned}$$

Finally we put everything together and obtain the program $\pi_{grd^*}$.

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

**Proposition 3.9** *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 handcrafted saturation-based and the alternative `metasp` encodings.

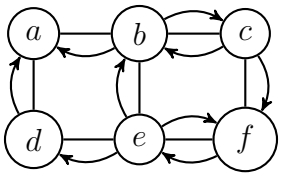
The tests were executed on an openSUSE based machine with eight Intel Xeon processors (2.33 GHz) and 49 GB memory. For generating the answer sets, we used `gringo` version 3.0.3 for grounding and the solver `claspD` (version 1.1.1). The latter being the variant of `clasp` for disjunctive answer-set programs.

We randomly generated AFs (i.e. graphs) ranging from 20 to 110 arguments. We used two parameterized methods for generating the attack relation in these AFs.

1. Arbitrary AF $F$: For any pair $a, b$ of arguments in $F$ the attack $(a, b)$ is inserted into $F$ with a given probability $p$.

2. Grid-structured AF $F$: Every argument is arranged in an $(n \times m)$ grid, i.e. is connected to its neighbors via attacks. The actual values for $n$ and $m$ depend on the total number of arguments; more precisely we let $n$ be either 5, 15, or 25 and $m$ is then chosen accordingly. For every neighbor $b$ of $a$ either the mutual attack {(a,b),(b,a)} is added with a probability $p$, or otherwise a single attack is inserted into $F$. The direction of the single attack is chosen randomly with equal chance. For the grids we consider two neighborhoods, namely 4-neighborhood and 8-neighborhood. The former connects all arguments horizontally and vertically, the latter connecting them also diagonally.

As an illustrative example for the grid structure, consider the following grid.




The undirected edges represent the neighborhood (4-neighborhood in this case). With a probability $p$ the mutual attack is used (as for the arguments $b, c$) and otherwise only one direction is inserted. The probability $p$ for generating the instances was chosen from $0.1$ to $0.4$.

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 for the semantics during the test runs.

The Figures 1 - 6 show the comparison results for the preferred, semi-stable and stage semantics respectively. Each of the figures has several subfigures. The subfigures show first a comparison of average computation time wrt. to the number of arguments and then a percentage of timeouts. The remaining subfigures show detailed behavior of each encoding separately, namely the time consumption of each test run individually and a box plot. In these detailed subfigures the instances are ordered from the left to right wrt. number of arguments. The box plot shows the mean, lower and higher quartile as a box of the computation times. Note that the whiskers of the box plot extend to the minimum and maximum time consumption. The box plot is shown for the arbitrary and the two different grid-structured instances.

One can see that the `metasp` encodings have a better performance, compared to the handcrafted encodings, on the preferred, semi-stable and stage semantics. In particular, for the stage semantics the performance difference between the handcrafted and the `metasp` variant is noticeable. The `metasp` encodings performed better wrt. to arbitrary AFs than on the structured ones, whereas for the handcrafted encodings this behavior differs. The handcrafted stage encoding performed similar on both the arbitrary and grid-structured instances. Recall that the average computation time includes the timeouts, which strongly influence the diagrams.

The overall picture changes wrt. the resolution-based grounded semantics. The graphical representations of the results can be seen in the Figures 7 - 10. We distinguish between the three encoding alternatives, namely the handcrafted, $\pi_{grd^*}$, and the two `metasp` variants, $\pi_{grd^*\_metasp}$ and $\pi'_{grd^*\_metasp}$. The two `metasp` encodings use the minimization for the resolution part, but the latter also uses it for computing the grounded extension. If we compare the resolution-based grounded semantics with the other three semantics, then all three encodings presented have a relatively high average computation time even for a low number of arguments.

If we look more closely at the structure of the AFs, then we see that the computation time quickly rises for arbitrary generated AFs. Noticeable is the statistics for AFs with argument size of at least $40$. The handcrafted encoding could not solve most of these instance due to memory allocation faults. These are indicated by the missing data points. The `metasp` encodings could only

solve a small subset of the AFs with at least $40$ arguments without timing out, namely those which were generated with a small number of attacks, i.e. the parameter for the inserting probability $p$ was low.

For the grid-structured AFs we see that the $4$-neighborhood grids were easier to solve for all three encodings, the $\pi'_{grd^*\_metasp}$ encoding outperforming the other two. Interestingly computing the grounded extension explicitly did not result in a performance gain in this case. The $8$-neighborhood grids were more difficult for the encodings to solve, but we have a similar overall picture as we had with the $4$-neighborhood grids, except that the timeout rate is much higher.

Again, the timeouts strongly influence the average computation time. In particular the timeout rate for the resolution-based grounded encodings is much higher than for the other semantics.

# 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 handcraft encodings) with novel alternative encodings which are based on the recently developed `metasp` approach [15], where subset minimization can be directly specified (and a frontend, i.e. a meta-interpreter) compiles such statements back into the core ASP language. (2) For the stage semantics, we presented here both a handcraft and a `metasp` encoding. Finally, (3) for the resolution-based grounded semantics we provided three encodings, two of them using the `metasp` techniques.

Although the `metasp` encodings are much simpler to design (since saturation techniques are delegated to the meta-interpreter), 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 has to focus on further experiments which hopefully will strengthen our observations. To this end, typical benchmarks for abstract argumentation have to be collected and provided; we believe that this is a necessary next step for the argumentation community.

# References


[1] Pietro Baroni, Paul E. Dunne, and Massimiliano Giacomin. On the resolution-based family of abstract argumentation semantics and its grounded instance. *Artif. Intell.*, 175(3-4):791–813, 2011.

[2] Pietro Baroni and Massimiliano Giacomin. Semantics of abstract argument systems. In I. Rahwan and G. Simari, editors, *Argumentation in Artificial Intelligence*, pages 25–44. Springer, 2009.

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

[4] Martin Caminada. Semi-stable semantics. In Paul E. Dunne and Trevor J. M. Bench-Capon, editors, *Proceedings of the 1st Conference on Computational Models of Argument* (*COMMA 2006*), volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 121–130. IOS Press, 2006.

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

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

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

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

[9] Paul E. Dunne and Martin Caminada. Computational complexity of semi-stable semantics in abstract argumentation frameworks. In Steffen Hölldobler, Carsten Lutz, and Heinrich Wansing, editors, *Proceedings of the 11th European Conference on Logics in Artificial Intelligence JELIA 2008*, volume 5293 of *Lecture Notes in Computer Science*, pages 153–165. Springer, 2008.

[10] Wolfgang Dvořák and Stefan Woltran. Complexity of semi-stable and stage semantics in argumentation frameworks. *Inf. Process. Lett.*, 110(11):425–430, 2010.

[11] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. Answer-set programming encodings for argumentation frameworks. *Argument and Computation*, 1(2):147–177, 2010.

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

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

[14] M. Gebser, B. Kaufmann, and T. Schaub. Solution enumeration for projected Boolean search problems. In W. van Hoeve and J. Hooker, editors, *Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09)*, volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2009.

[15] Martin Gebser, Roland Kaminski, and Torsten Schaub. Complex optimization in answer set programming. *TPLP*, 2011. Accepted for publication.

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

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

[18] Francesca Toni and Marek Sergot. Argumentation and answer set programming. In M. Balduccini and T.C. Son, editors, *Gelfond Festschrift*, volume 6565 of *LNAI*, pages 164–180. Springer, 2011.

[19] Bart Verheij. Two approaches to dialectical argumentation: admissible sets and argumentation stages. In J. Meyer and L. van der Gaag, editors, *Proceedings of the 8th Dutch Conference on Artificial Intelligence (NAIC'96)*, pages 357–368, 1996.

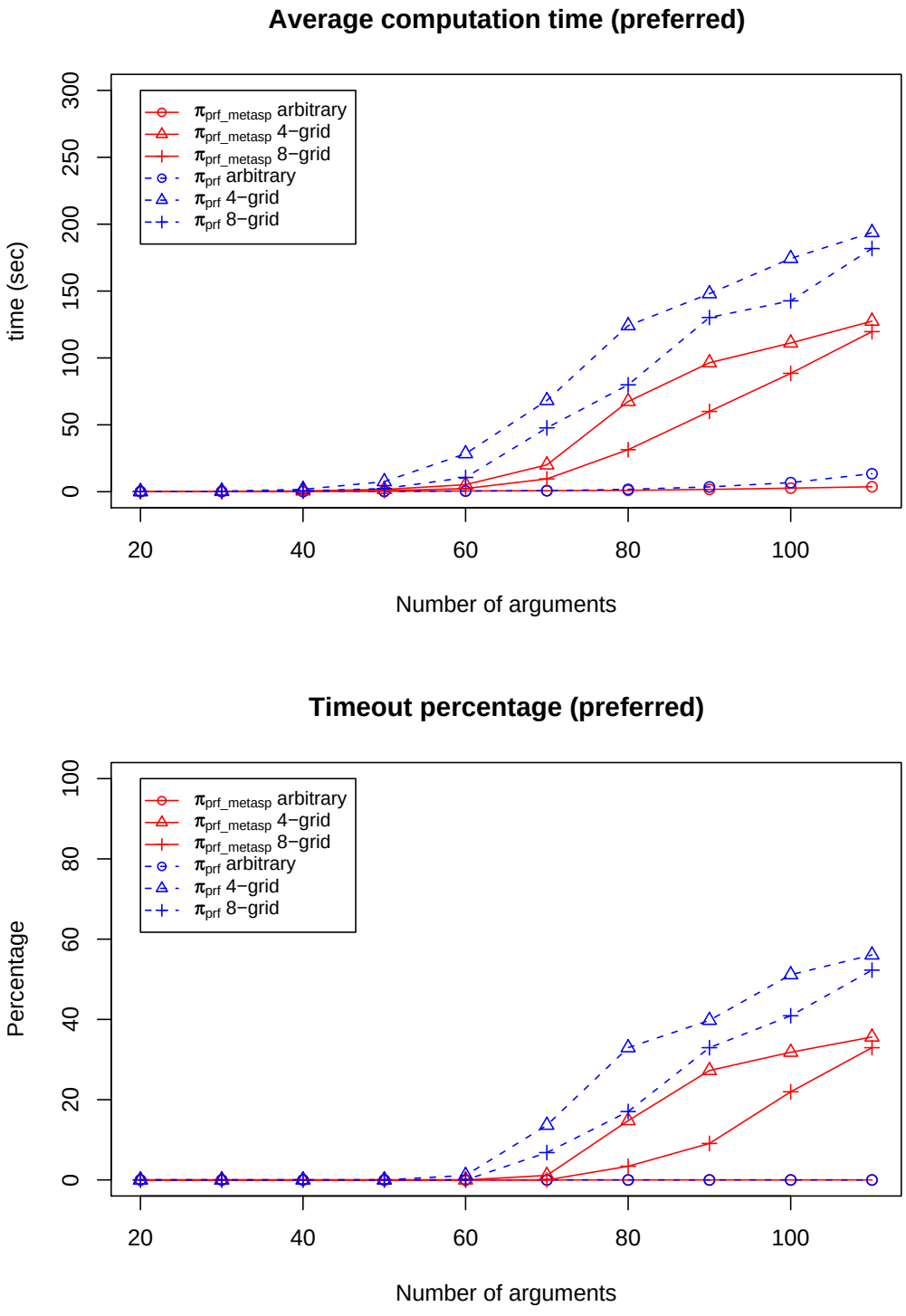



Figure 1: Average and timeout statistics for preferred semantics.

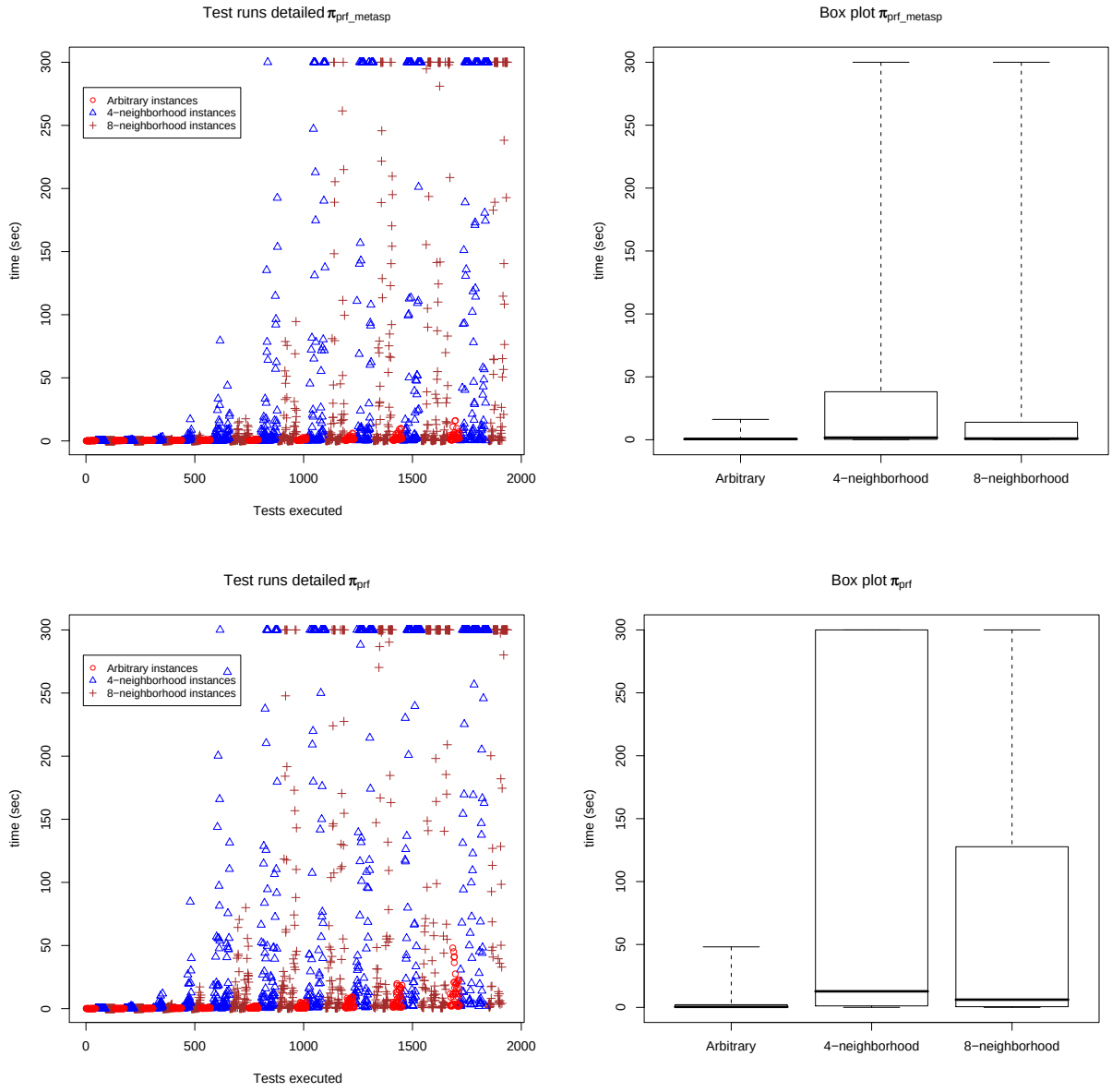



Figure 2: Statistics for preferred semantics.

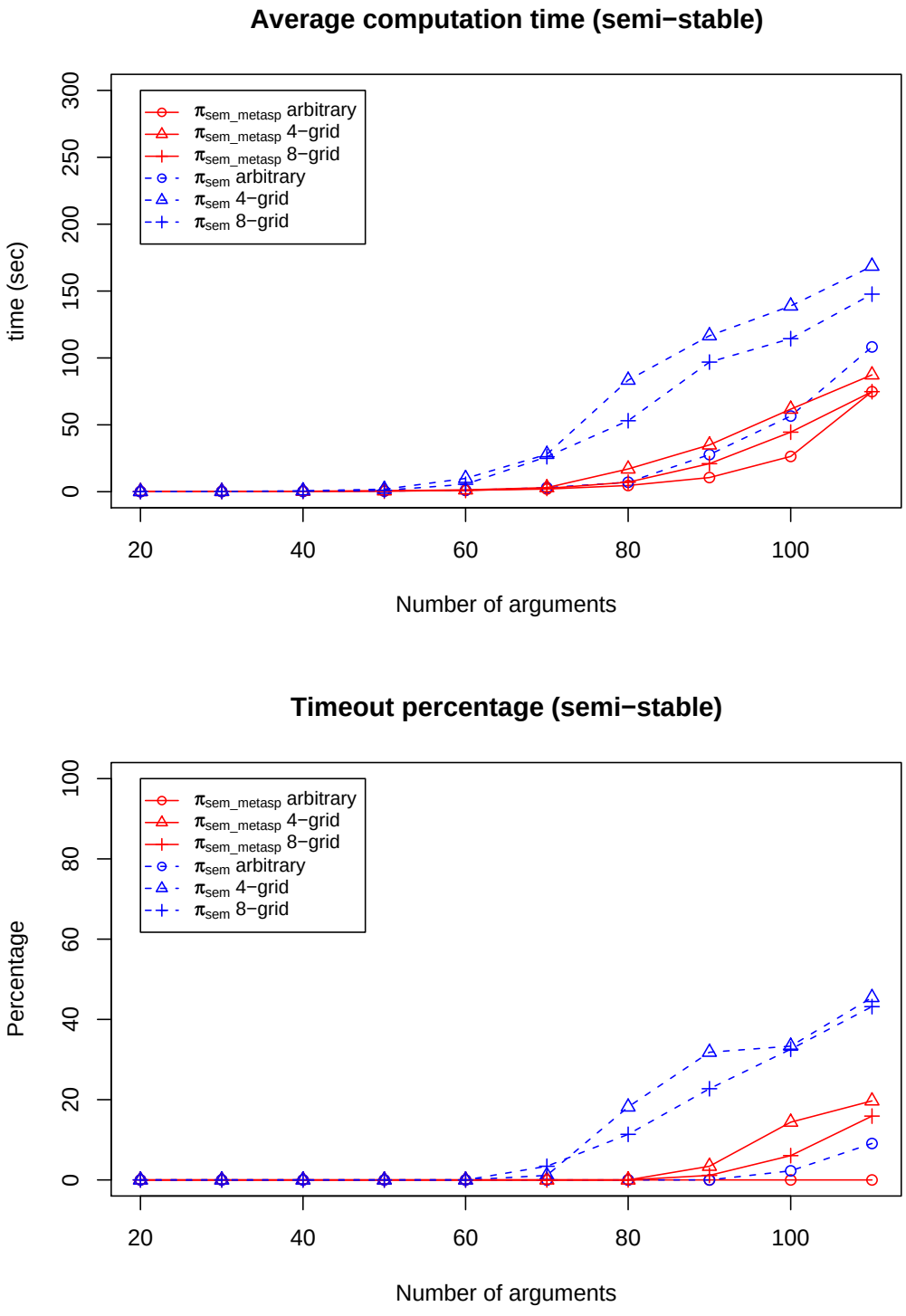



Figure 3: Average and timeout statistics for semi-stable semantics.

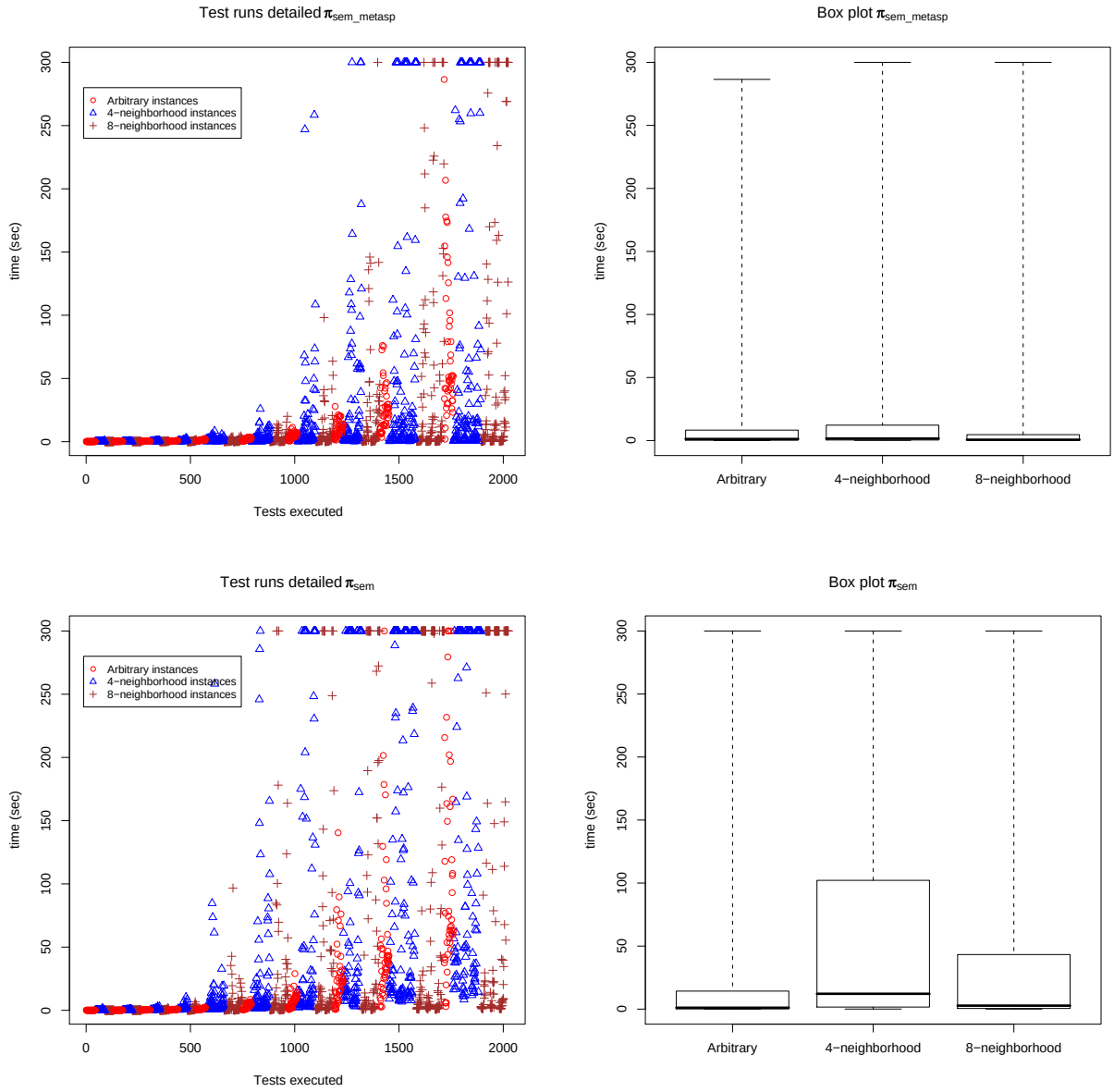



Figure 4: Statistics for semi-stable semantics.

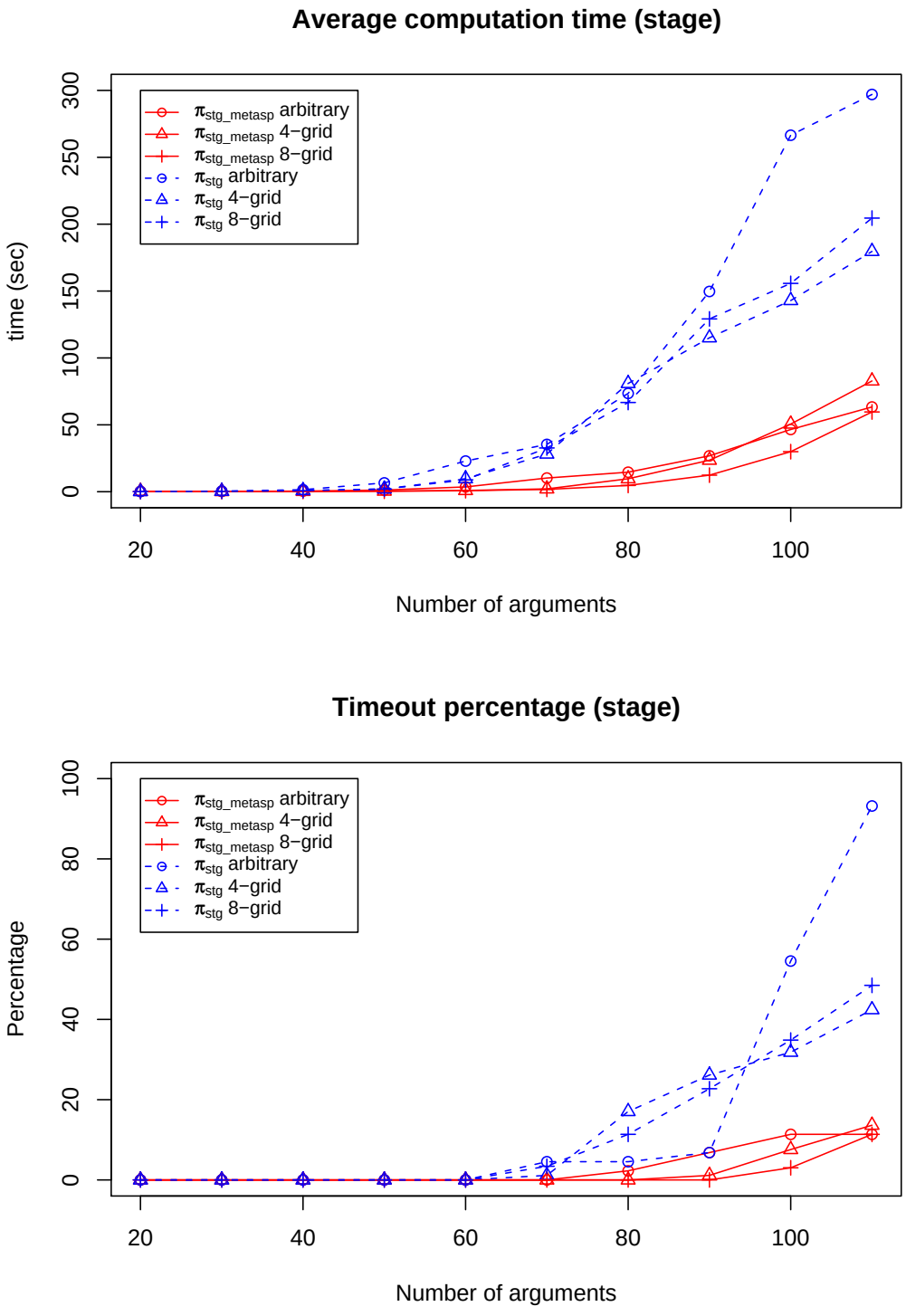



Figure 5: Average and timeout statistics for stage semantics.

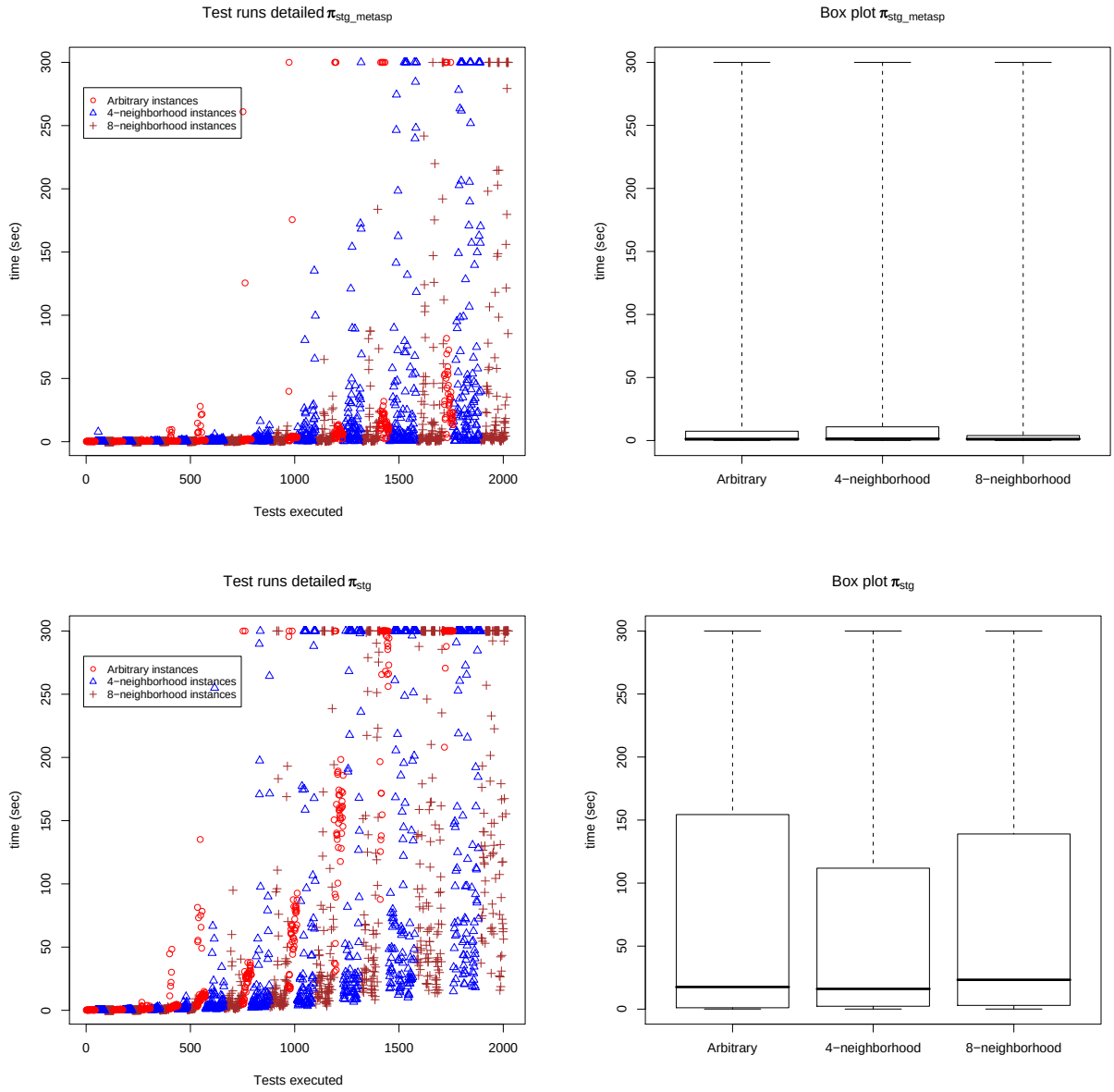



Figure 6: Statistics for stage semantics.

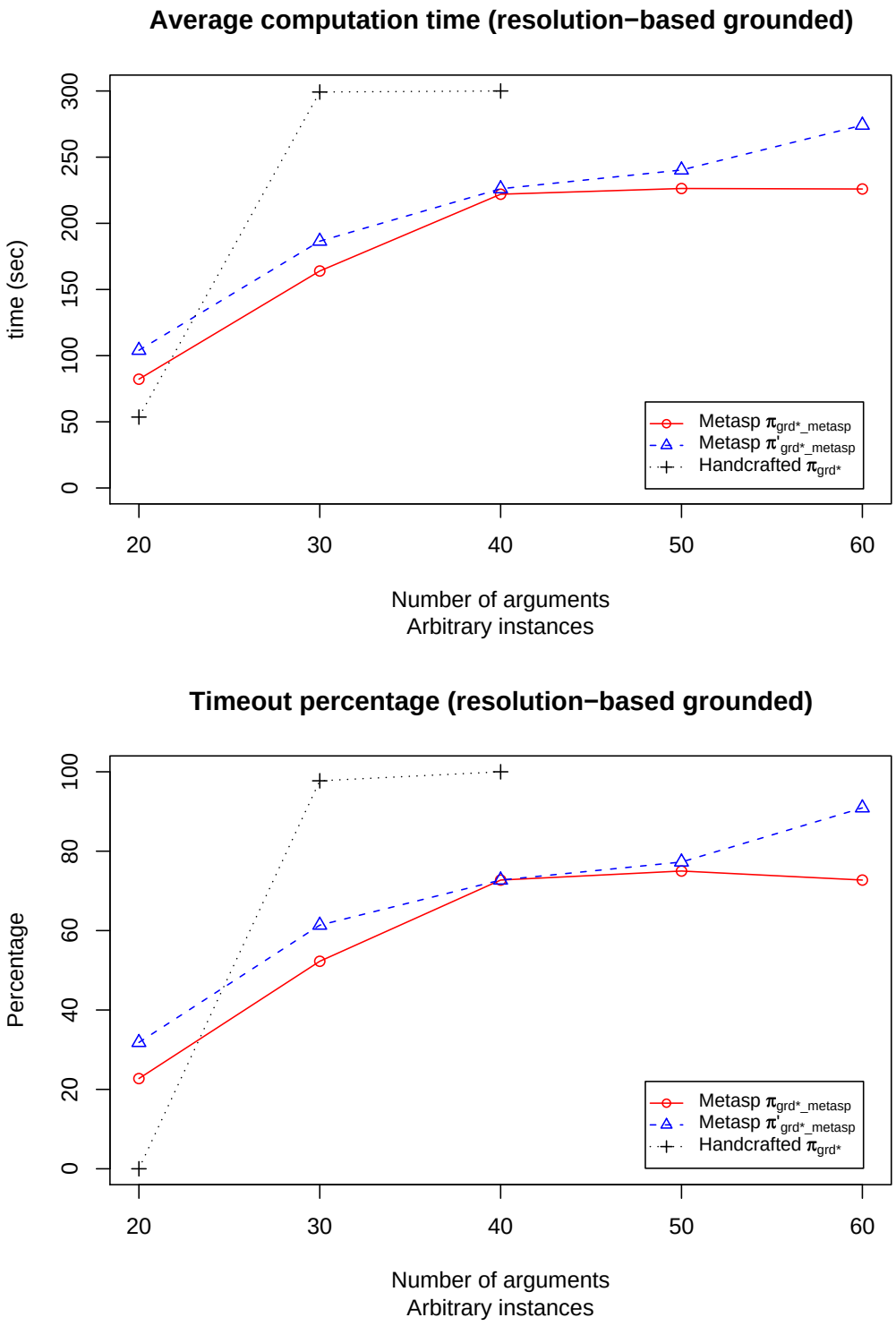
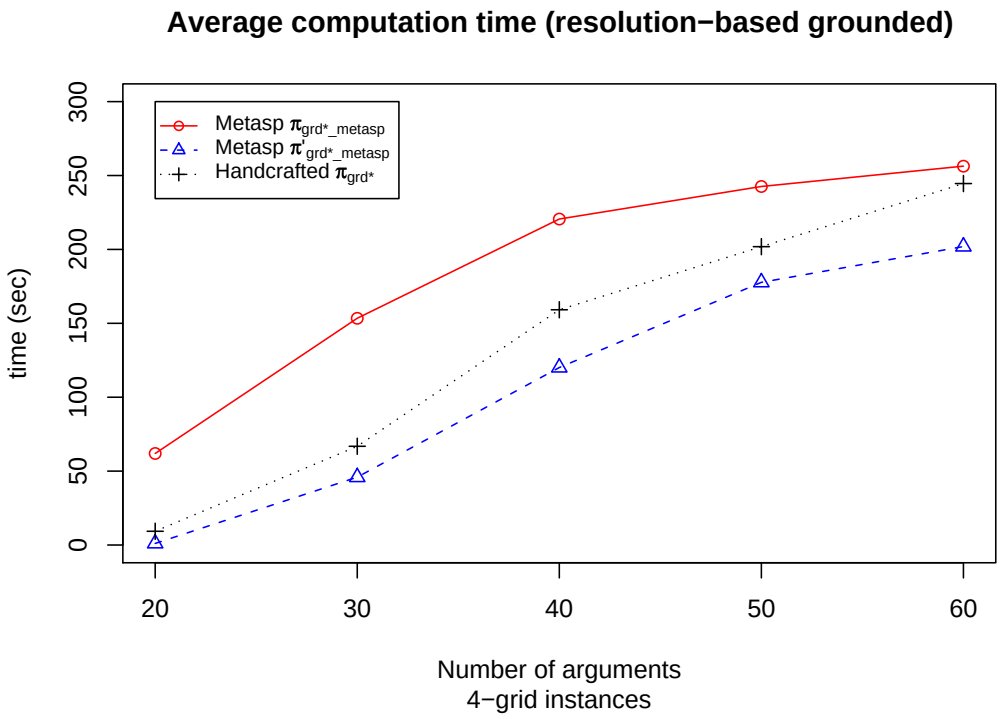



Figure 7: Average computation time and timeouts for resolution-based grounded semantics and arbitrary AFs.



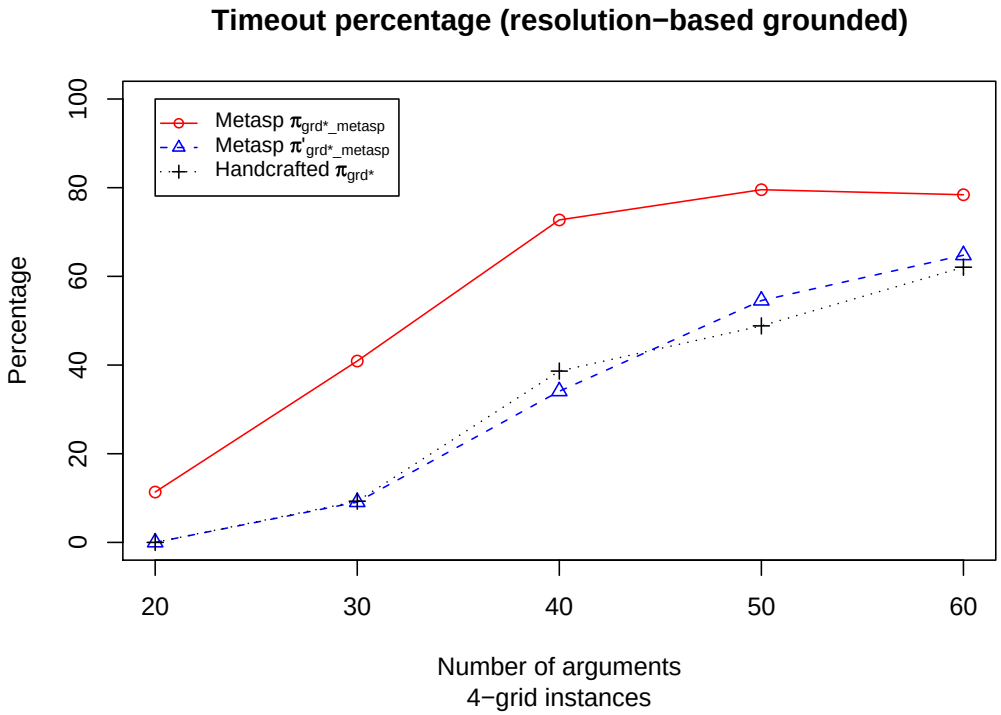
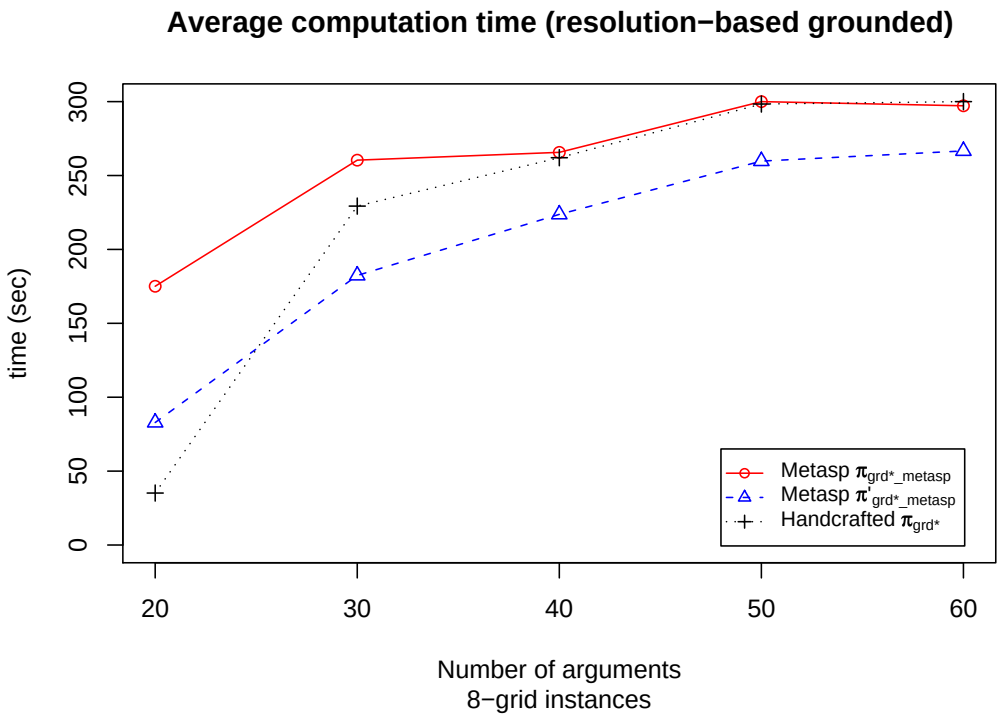




Figure 8: Average computation time and timeouts for resolution-based grounded semantics and 4-neighborhood AFs.



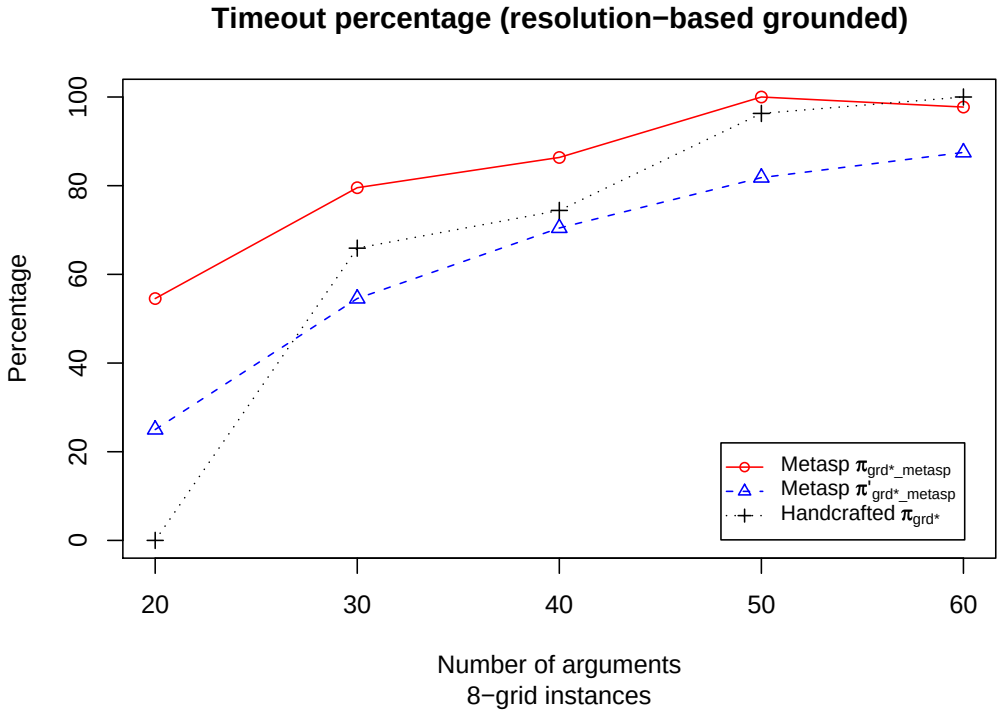




Figure 9: Average computation time and timeouts for resolution-based grounded semantics and 8-neighborhood AFs.

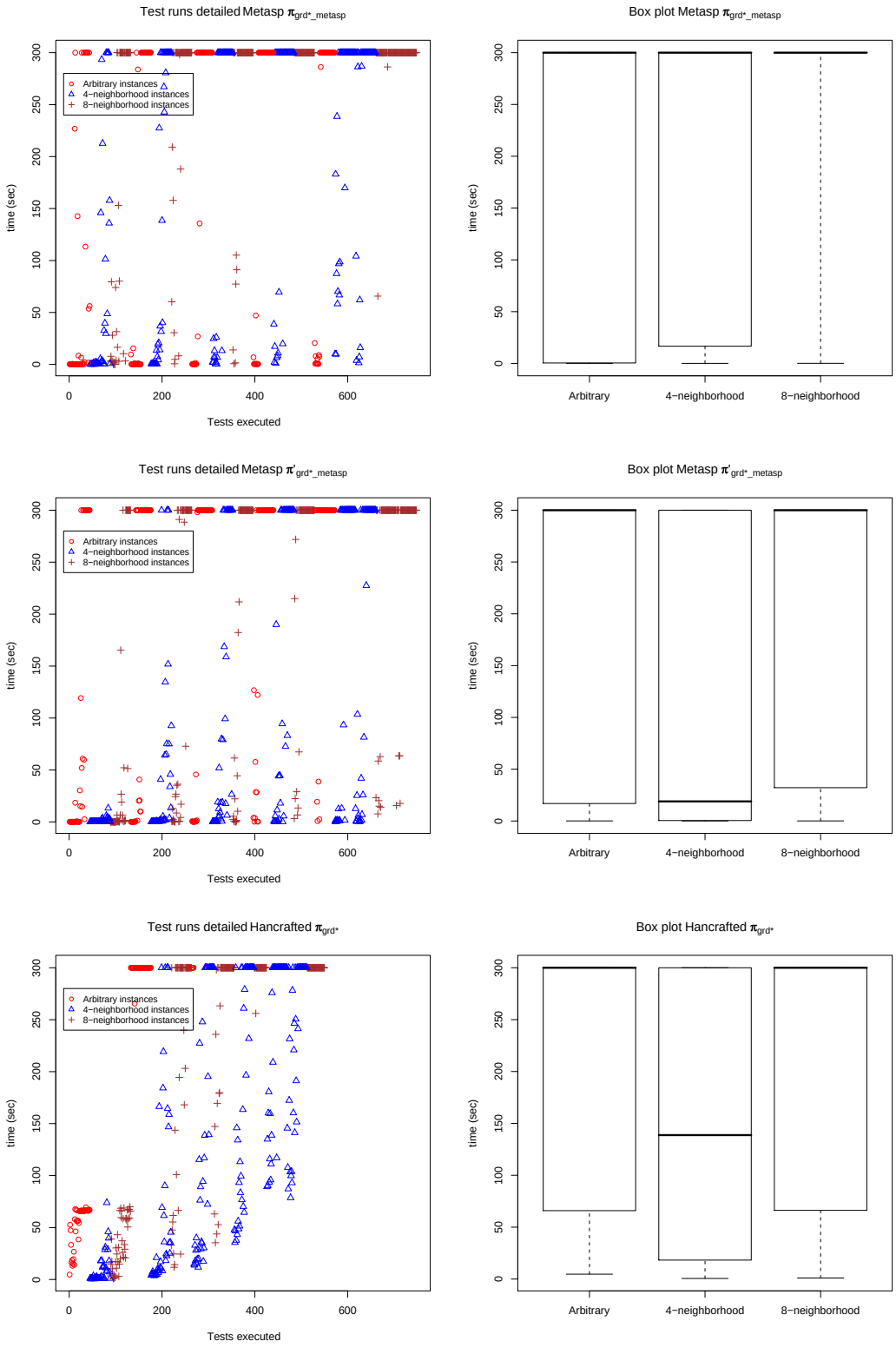



Figure 10: Detailed statistics for resolution-based grounded semantics.