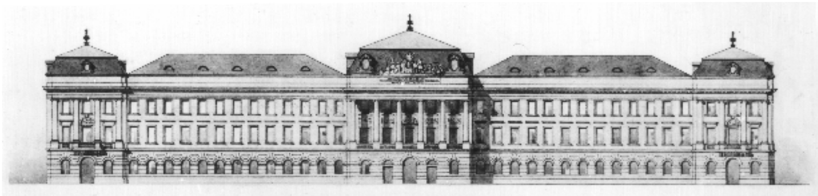
dbai

TECHNICAL REPORT



INSTITUT FÜR LOGIC AND COMPUTATION

ABTEILUNG DATENBANKEN UND ARTIFICIAL INTELLIGENCE

# Evaluating SETAFs via Answer-Set Programming

**DBAI-TR-2018-112**

**Wolfgang Dvořák** **Alexander Greßler**
**Stefan Woltran**

Institut für Logic and Computation
Abteilung Datenbanken und
Artificial Intelligence
Technische Universität Wien
Favoritenstr. 9
A-1040 Vienna, Austria
Tel: +43-1-58801-18403
Fax: +43-1-58801-918403
sek@dbai.tuwien.ac.at
www.dbai.tuwien.ac.at

DBAI TECHNICAL REPORT
2018

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DBAI TECHNICAL REPORT
DBAI TECHNICAL REPORT DBAI-TR-2018-112, 2018

# On the Expressive Power of Collective Attacks

**Wolfgang Dvořák** [1] **Alexander Greßler** [2] **Stefan Woltran** [3]

**Abstract.** Following the tradition of the ASPARTIX system, we present answer-set programming encodings for the main semantics of argumentation frameworks with collective attacks (also known as SETAFs). By that, we provide the first system dedicated to reasoning in SETAFs. Since to date, no known polynomial-time (with respect to the number of arguments) translation from SETAFs to Dung-style frameworks is known, genuine implementations for SETAFs appear necessary towards practically efficient systems for this particular formalism. As a by-product, we introduce semi-stable and stage semantics for SETAFs and pinpoint the complexity of all considered semantics.

[1]Institute of Logic and Computation, TU Wien, Austria. E-mail: dvorak@dbai.tuwien.ac.at
[2]Institute of Logic and Computation, TU Wien, Austria.
[3]Institute of Logic and Computation, TU Wien, Austria.

# 1 Introduction

Abstract argumentation frameworks (AFs) as introduced by Dung in his seminal paper [7] are a core formalism in formal argumentation and have been extensively studied in the literature. In Dung AFs, conflicts and attacks are restricted to be binary in the sense that a single attack only concerns two arguments.

SETAFs as introduced by Nielsen and Parsons [23] are a generalization of Dung AFs which generalize the binary attacks in Dung AFs to collective attacks. This enables the formalization of the fact that a set $B$ of arguments may jointly attack another argument $a$ but no proper subset of $B$ attacks $a$. The semantics as proposed in [23], make SETAFs a conservative generalization of Dung AFs; in other words, a SETAF where all attacks are binary is evaluated in the same way as the corresponding Dung AF. As illustrated in [23], there are several scenarios where arguments interact and can constitute an attack on another argument only if these arguments are jointly taken into account. Representing such a situation in Dung AFs often requires additional artificial arguments to "encode" the conjunction of arguments, potentially causing an exponential blow-up in the number of arguments [24]. In [24] it is also shown that SETAFs allow for more straightforward and compact encodings of support between arguments than AFs do. Moreover, recent work [11] shows that the collective attacks of SETAFs increase the expressiveness when compared to Dung AFs.

Computational properties of SETAFs have been neglected so far. One notable exception is by Nielsen and Parsons [22] who adapted the algorithms by Doutre and Mengin [6] for enumerating preferred extensions in Dung AFs to SETAFs. Besides that, SETAFs have been identified as special case of the more general Abstract Dialectical Frameworks (ADFs) [2] and thus systems for ADFs, e.g. the DIAMOND system [18], can in principle be used to evaluate SETAFs as well. However, these systems have a significant drawback since SETAFs are encoded in a more complex formalism leading to a potential computational overhead. Another way to evaluate SETAFs is to translate them to Dung AFs and use existing systems. However, since to date, no known polynomial-time (with respect to the number of arguments) translation from SETAFs to AFs is known, this again might cause a non-negligible computational overhead. Genuine implementations for SETAFs thus appear necessary towards practically efficient systems.

The main aim of this work is to provide a flexible system dedicated to reasoning in SETAFs with a broad range of semantics. Answer-Set Programming (ASP) [1] proved useful for rapid prototyping of systems for Dung AFs (see e.g. [13, 19, 25]) and generalization thereof (see e.g. [18, 12]). We follow the approach of the ASPARTIX system, where for each semantics a fixed encoding is provided which when combined with an AF as input returns the corresponding extensions. We provide ASP-encodings for all the SETAF-semantics defined in [23] as well as semi-stable and stage semantics. These encodings are also provided in executable format [1] and can be used to enumerate extensions or to decide credulous and skeptical acceptance for SETAFs.

Our contributions and the organization of the paper are as follows. We first recall the definitions and fundamental results from [23] (see Section 2) and then generalize the definitions of semi-stable and stage semantics to SETAFs (see Section 2.1). Next, we clarify the complexity landscape of

[1] See `www.dbai.tuwien.ac.at/research/argumentation/aspartix/setaf.html`.

SETAFs for the standard reasoning problems (see Section 2.2). In the main part of our paper we provide ASP-encodings for the different semantics of SETAFs (see Section 3). Finally, we discuss our results in a conclusion section. Some technical details are omitted in the main part of the paper but are provided in the two appendices.

# 2 Argumentation Frameworks with Collective Attacks

We first introduce formal definitions of argumentation frameworks with collective attacks following [23].

**Definition 1.** A *SETAF* is a pair $F = (A, R)$ where $A$ is finite, and $R \subseteq (2^A \setminus \emptyset) \times A$ is the attack relation. We write $S \mapsto_R b$ if there is a set $S' \subseteq S$ with $(S', b) \in R$. Moreover, we write $S' \mapsto_R S$ if $S' \mapsto_R b$ for some $b \in S$. For $S \subseteq A$, the *range* of $S$ (w.r.t. $R$), denoted $S_R^\oplus$, is the set $S \cup \{b \mid S \mapsto_R b\}$.

We call a SETAF with binary attacks only, i.e. $|S| = 1$ for each $(S, a) \in R$, Dung argumentation framework (AF) as such SETAFs are equivalent to the AFs introduced in [7].

**Example 1.** Consider an argumentation framework with arguments $a, b, c$ where each pair of arguments attacks the third argument. This is modeled by the SETAF $F = (A, R)$ with arguments $A = \{a, b, c\}$ and attacks $R = \{(\{a, b\}, c), (\{a, c\}, b), (\{b, c\}, a)\}$. $\Diamond$

The notion of defense naturally generalizes to SETAFs.

**Definition 2.** Given a SETAF $F = (A, R)$, an argument $a \in A$ is *defended* (in $F$) by a set $S \subseteq A$ if for each $B \subseteq A$, such that $B \mapsto_R a$, also $S \mapsto_R B$. A set $T$ of arguments is defended (in $F$) by $S$ if each $a \in T$ is defended by $S$ (in $F$).

Based on the concept of defense also the definition of the characteristic function of an argumentation framework can be extended to SETAFs.

**Definition 3.** The characteristic function of an SETAF $F = (A, R)$ is the function $\mathcal{F}_F : 2^A \to 2^A$ with $\mathcal{F}_F(S) = \{a \in A : a \text{ is defended by } S\}$.

## 2.1 Semantics

Next, we introduce the semantics we study in this work. These are the naive, stable, preferred, complete, grounded, stage, and semi-stable semantics, which we will abbreviate by *naive*, *stb*, *pref*, *com*, *grd*, *stage*, and *sem*, respectively. All semantics except semi-stable and stage are defined according to [23], while semi-stable and stage are straight forward generalizations of the according semantics for Dung AFs [26, 3]. For a given semantics $\sigma$, $\sigma(F)$ denotes the set of extensions of $F$ under $\sigma$.

**Definition 4.** Given a SETAF $F = (A, R)$, a set $S \subseteq A$ is *conflict-free* (in $F$), if $S' \cup \{a\} \not\subseteq S$ for each $(S', a) \in R$. We denote the set of all conflict-free sets in $F$ as $cf(F)$. $S \in cf(F)$ is called *admissible* (in $F$) if $S$ defends itself. We denote the set of admissible sets in $F$ as $adm(F)$. For a conflict-free set $S \in cf(F)$, we say that

- $S \in naive(F)$, if there is no $T \in cf(F)$ such that $T \supset S$,
- $S \in stb(F)$, if $S \mapsto_R a$ for all $a \in A \setminus S$,
- $S \in pref(F)$, if $S \in adm(F)$ and there is no $T \in adm(F)$ such that $T \supset S$,
- $S \in com(F)$, if $S \in adm(F)$ and $a \in S$ for all $a \in A$ defended by $S$,
- $S \in grd(F)$, if $S = \bigcap_{T \in com(F)} T$,
- $S \in stage(F)$, if there is no $T \in cf(F)$ such that $T_R^{\oplus} \supset S_R^{\oplus}$, and
- $S \in sem(F)$, if $S \in adm(F)$ and there is no $T \in adm(F)$ such that $T_R^{\oplus} \supset S_R^{\oplus}$.

As shown in [23], most of the fundamental properties of Dung AFs extend to SETAFs. We have the same relations between the semantics as in Dung AFs, i.e. $stb(F) \subseteq sem(F) \subseteq pref(F) \subseteq com(F) \subseteq adm(F) \subseteq cf(F)$ and $stb(F) \subseteq stage(F) \subseteq naive(F)$. The grounded extension is the unique minimal complete extension for any SETAF $F$. If there is at least one stable extension then stable, semi-stable, and stage semantics coincide. The properties for semi-stable and stage semantics follow from straightforward adaptations of the proofs for Dung AFs (see Appendix A). Moreover, Dung's fundamental lemma generalizes to SETAFs.

**Lemma 1** ([23]). *Given a SETAF $F = (A, R)$, a set $B \subset A$, and arguments $a, b \in A$ that are defended by $B$. Then (a) $B \cup \{a\}$ is admissible in $F$ and (b) $B \cup \{a\}$ defends $b$ in $F$.*

## 2.2 Complexity

In this section we assume the reader to have basic knowledge in computational complexity theory[2], in particular we make use of the complexity classes L (logarithmic space), P (polynomial time), NP (non-deterministic polynomial time), coNP, $\Sigma_2^{\mathsf{P}}$ and $\Pi_2^{\mathsf{P}}$.

For a given SETAF we consider the standard reasoning problems in formal argumentation: Credulous acceptance $Cred_\sigma$: Is a given argument contained in at least one $\sigma$ extension?; Skeptical acceptance $Skept_\sigma$: Is a given argument contained in all $\sigma$ extensions?; Verification $Ver_\sigma$: Is a given set a $\sigma$ extensions of the SETAF?; Existence of a Extension $Exists_\sigma$: Does the SETAF have a $\sigma$ extension?; and Existence of a nonempty Extension $Exists_\sigma^{\neg\emptyset}$: Does the SETAF have a non-empty $\sigma$ extension?

The complexity landscape of SETAFs coincides with that of Dung AFs and is depicted in Table 1. As SETAFs generalize Dung AFs the hardness results for Dung AFs [4, 5, 8, 14, 15, 9] (for a survey see [10]) carry over to SETAFs. Interestingly also the same upper bounds hold

[2]For a gentle introduction to complexity theory in the context of formal argumentation, see [10].

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

| $\sigma$ | $Cred_\sigma$ | $Skept_\sigma$ | $Ver_\sigma$ | $Exists_\sigma$ | $Exists_\sigma^{\neg\emptyset}$ |
|---|---|---|---|---|---|
| *cf* | in L | trivial | in L | trivial | in L |
| *naive* | in L | in L | in L | trivial | in L |
| *grd* | P-c | P-c | P-c | trivial | in L |
| *stb* | NP-c | coNP-c | in L | NP-c | NP-c |
| *adm* | NP-c | trivial | in L | trivial | NP-c |
| *com* | NP-c | P-c | in L | trivial | NP-c |
| *pref* | NP-c | $\Pi_2^P$-c | coNP-c | trivial | NP-c |
| *sem* | $\Sigma_2^P$-c | $\Pi_2^P$-c | coNP-c | trivial | NP-c |
| *stage* | $\Sigma_2^P$-c | $\Pi_2^P$-c | coNP-c | trivial | in L |

for SETAFs. In SETAFs checking whether a set is conflict-free and evaluating the characteristic function is more evolved than in Dung AFs but still can be done in L. As this is at the basis of the complexity upper bounds for Dung AFs, these upper bounds also apply to SETAFs with slight adaptations in the algorithms (details are provided in Appendix B). Notice that also our ASP-encodings implicitly show matching upper bounds for many of the decision problems (by the complexity of the corresponding fragments of the ASP language).

However, we want to highlight a subtle difference between the complexity results for Dung AFs and SETAFs. In both cases the complexity is w.r.t. the size of the input framework, which in case of Dung AFs is often interpreted as w.r.t. the number of arguments $|A|$ in the input framework. However, this interpretation is not valid for SETAFs where the number of attacks $|R|$ can be exponentially larger than the number of arguments $|A|$ (this even holds for normal forms where redundant attacks are removed). Thus, the results for SETAFs should be understood as complexity w.r.t. $|A| + |R|$. This also reflects the fact that when translating SETAFs to AFs there can be an exponential blow up in the number of arguments [24].

# 3 ASP-Encodings

In this part of the paper we provide ASP-encodings for the different SETAF semantics. We do this by following the same approach as in the ASPARTIX system [16, 19] for Dung AFs. That is, for each of the semantics we provide a fixed encoding (a.k.a. query) that, when combined with the encoding of a SETAF as input, returns the corresponding extensions as answer-sets.

In what follows, we first give an overview on disjunctive logic programs under the answer-sets semantics (Section 3.1), then we state the input format for SETAFs (Section 3.2), and finally we present our encodings for the different semantics (Section 3.3).

### 3.1 Background on ASP

We give an overview of the syntax and semantics of disjunctive logic programs under the answer-sets semantics [21].

We fix a countable set $\mathcal{U}$ of *(domain) elements*, also called *constants*. 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* $r$ is of the form

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

with $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$, where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are literals, 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. If each rule in a program is normal (resp. ground), we call the program normal (resp. ground).

For any program $\pi$, let $U_\pi$ be the set of all constants appearing in $\pi$. $Gr(\pi)$ is the set of rules $r\sigma$ obtained by applying, to each rule $r \in \pi$, all possible substitutions $\sigma$ from the variables in $r$ to elements of $U_\pi$. 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 $\pi$, if each $r \in \pi$ is satisfied by $I$. A non-ground rule $r$ (resp., a program $\pi$) 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 $\pi$ 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 $\pi$, we denote the set of its answer-sets by $\mathrm{AS}(\pi)$.

Modern ASP solvers offer additional language features. Among them we make use of the *conditional literal* [20]. In the head of a disjunctive rule literals may have conditions, e.g. consider the head of rule "$\mathbf{p}(X) : \mathbf{q}(X) \leftarrow$". Intuitively, this represents a head of disjunctions of atoms $\mathbf{p}(a)$ where also $\mathbf{q}(a)$ is true. As well rules might have conditions in their body, e.g. consider the body of rule "$\leftarrow \mathbf{p}(X) : \mathbf{q}(X)$", which intuitively represents a conjunction of atoms $\mathbf{p}(a)$ where also $\mathbf{q}(a)$ is true.

### 3.2 Encoding SETAFs

Before specifying the encodings for the specific semantics, we need to fix an encoding $\pi_{setaf}(A, R)$ of SETAFs as input databases. A SETAF $F = (A, R)$ is encoded by predicates $\mathbf{arg}$, $\mathbf{att}$, and $\mathbf{mem}$. The former is used to encode arguments, the latter two to encode the set attacks. Notice that the encoding uses a unique identifier for each attack in $R$.

$$\begin{aligned} \pi_{setaf}(A, R):\quad & \{\mathbf{arg}(a).\mid \text{for } a \in A\} \cup \\ & \{\mathbf{att}(r, x).\mid \text{for } r \in R \text{ and } r = (S, x)\} \cup \\ & \{\mathbf{mem}(r, y).\mid \text{for } r \in R,\ r = (S, x) \text{ and } y \in S\} \end{aligned}$$

We next exemplify this encoding on the SETAF from Example 1.

**Example 2.** Consider the SETAF $F$ from Example 1. The encoding $\pi_{setaf}(F)$ of $F$ is given by

$$\begin{aligned}
&\mathbf{arg}(a).\ \mathbf{arg}(b).\ \mathbf{arg}(c).\\
&\mathbf{att}(r1, c).\ \mathbf{mem}(r1, a).\ \mathbf{mem}(r1, b).\\
&\mathbf{att}(r2, b).\ \mathbf{mem}(r2, a).\ \mathbf{mem}(r2, c).\\
&\mathbf{att}(r3, a).\ \mathbf{mem}(r3, b).\ \mathbf{mem}(r3, c).
\end{aligned}$$

$\Diamond$

### 3.3 Encoding Semantics

In this section we provide encodings $\pi_\sigma$ for the semantics under our considerations, such that the answer-sets of $\pi_{setaf}(F) \cup \pi_\sigma$ are in certain correspondence to the $\sigma$-extensions of $F = (A, R)$. Intuitively, in an answer-set we are interested in the set of atoms for which the predicate **in** holds true and we require these sets to be in a one to one correspondence to the extensions of the SETAF $F$. We make our notion of correspondence precise by the next definition.

**Definition 5.** Let $\mathbf{S} \subseteq 2^{\mathcal{U}}$ 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 $\mathbf{S}$ and $\mathcal{I}$ correspond to each other, in symbols $\mathbf{S} \cong \mathcal{I}$, iff (i) for each $S \in \mathbf{S}$, there exists an $I \in \mathcal{I}$, such that $\{a \mid \mathbf{in}(a) \in I\} = S$; (ii) for each $I \in \mathcal{I}$, it holds that $\{a \mid \mathbf{in}(a) \in I\} \in \mathbf{S}$; and (iii) $|\mathbf{S}| = |I|$.

Notice that by the above definition we want to avoid situations where several answer-sets correspond to the same extension of the SETAF.

#### 3.3.1 Conflict-free Semantics

We start with a program fragment that, when augmented by $\pi_{setaf}(F)$, generates any subset $S \subseteq A$ and can then be augmented by further program fragments to filter out extensions of specific semantics.

$$\begin{aligned}
\pi_{guess}:\quad &\mathbf{in}(Y) \leftarrow \mathbf{arg}(Y), not\ \mathbf{out}(Y).\\
&\mathbf{out}(Y) \leftarrow \mathbf{arg}(Y), not\ \mathbf{in}(Y).
\end{aligned}$$

We call an attack $(B, a) \in R$ blocked w.r.t. a set $S \subseteq A$ if $B \not\subseteq S$. In our encoding of the conflict-freeness test we first compute the blocked attacks and then use a constraint that checks whether there is a non-blocked attack that attacks an argument in $S$.

$$\begin{aligned}
\pi_{cf'}:\quad &\mathbf{blocked}(R) \leftarrow \mathbf{mem}(R, X), \mathbf{out}(X).\\
&\leftarrow \mathbf{in}(X), \mathbf{att}(R, X), not\ \mathbf{blocked}(R).
\end{aligned}$$

To enumerate the conflict-free sets of a SETAF $F$ we combine the above code fragments, i.e. we define $\pi_{cf} = \pi_{guess} \cup \pi_{cf'}$ and obtain that $cf(F) \cong \mathrm{AS}(\pi_{setaf}(F) \cup \pi_{cf})$.

### 3.3.2 Admissible & Complete Semantics

For admissible semantics we start from the encoding of conflict-free semantics and add constraints to make sure that all arguments in the set are defended. To this end we introduce the concept of defeated attacks. An attack $(S, a) \in R$ is considered to be defeated by a set $E$ iff $E$ attacks at least one $x \in S$. Given the blocked (resp. the unblocked) attacks we can easily compute the defeated attacks. Now the definition of defense can be restated as, an argument $x$ is defended by as set $E$ iff all attacks $(S, x)$ are defeated. The code fragment $\pi_{def}$ consists of a rule that computes the defeated attacks and a constraint that checks that each argument in the extension is defended by the extension.

$$\begin{aligned} \pi_{def}: \quad \mathbf{defeated}(R) &\leftarrow \mathbf{att}(R, X), \mathbf{mem}(R, Y), \mathbf{att}(R2, Y), \mathit{not}\ \mathbf{blocked}(R2). \\ &\leftarrow \mathbf{in}(X), \mathbf{att}(R, X), \mathit{not}\ \mathbf{defeated}(R). \end{aligned}$$

Now we obtain the encoding $\pi_{adm}$ for admissible semantics by adding the above code fragment to the encoding of conflict-free semantics, i.e, $\pi_{adm} = \pi_{cf} \cup \pi_{def}$ and we have *adm*$(F) \cong \text{AS}(\pi_{setaf}(F) \cup \pi_{adm})$.

For complete semantics we, additionally to admissible semantics, have to make sure that no argument outside the set is defended. We do so by computing a predicate $\mathbf{notDefended}$ of arguments not defended by the extension and then add a constraint that checks whether there is an argument outside the extension for which the predicate does not hold true.

$$\begin{aligned} \pi_{cl}: \quad \mathbf{notDefended}(X) &\leftarrow \mathbf{att}(R, X), \mathit{not}\ \mathbf{defeated}(R). \\ &\leftarrow \mathbf{out}(X), \mathit{not}\ \mathbf{notDefended}(X). \end{aligned}$$

Now we obtain $\pi_{comp} = \pi_{adm} \cup \pi_{cl}$ as encoding for complete semantics, i.e. *com*$(F) \cong \text{AS}(\pi_{setaf}(F) \cup \pi_{comp})$.

### 3.3.3 Stable Semantics

To test whether a conflict-free set is stable we first compute all arguments that are attacked, and store them in the predicate $\mathbf{attArg}(Y)$. We then apply a constraint to test whether there is an argument that is neither in nor attacked by the extension.

$$\begin{aligned} \pi_{stb'}: \quad \mathbf{attArg}(X) &\leftarrow \mathbf{att}(R, X), \mathit{not}\ \mathbf{blocked}(R). \\ &\leftarrow \mathbf{out}(X), \mathit{not}\ \mathbf{attArg}(X). \end{aligned}$$

Combining the above fragment with the encoding for conflict-free semantics results the encoding $\pi_{stb} = \pi_{cf} \cup \pi_{stb'}$ of stable semantics with *stb*$(F) \cong \text{AS}(\pi_{setaf}(F) \cup \pi_{stb})$.

### 3.3.4 Naive Semantics

Given a conflict-free set $E$, it is either a naive extension or there is an argument $a \in A \setminus E$ such that $E \cup \{a\}$ is conflict-free. Thus, we first compute a binary predicate $\mathbf{blocked}(r, a)$ encoding that an attack $r$ is blocked for the set $E \cup \{a\}$. We then use this predicate to check whether $E \cup \{a\}$ has a conflict and if so, we set $\mathbf{conflArg}(a)$ true. Finally, we check whether there is an argument $a \in A \setminus E$ such that $E \cup \{a\}$ is conflict-free.

$$\begin{aligned} \pi_{cfmax}: \quad \mathbf{blocked}(R, A) &\leftarrow \mathbf{out}(A), \mathbf{mem}(R, X), \mathbf{out}(X), X \neq A. \\ \mathbf{conflArg}(A) &\leftarrow \mathbf{out}(A), \mathbf{in}(X), \mathbf{att}(R, X), \mathit{not}\ \mathbf{blocked}(R, A). \\ \mathbf{conflArg}(A) &\leftarrow \mathbf{out}(A), \mathbf{att}(R, A), \mathit{not}\ \mathbf{blocked}(R, A). \\ &\leftarrow \mathbf{out}(A), \mathit{not}\ \mathbf{conflArg}(A). \end{aligned}$$

Now the full encoding for naive semantics is $\pi_{naive} = \pi_{cf} \cup \pi_{cfmax}$, i.e. we have $\mathit{naive}(F) \cong \mathrm{AS}(\pi_{setaf}(F) \cup \pi_{naive})$.

### 3.3.5 Preferred Semantics

For the encoding of preferred semantics we start from the encoding of admissible sets and add a maximality check using the so-called saturation technique [17] (see also [16]). The idea is to use disjunctive rules to construct a superset (stored in the $\mathbf{sIn}$ predicate) of the admissible set stored in the $\mathbf{in}$ predicate. We first use a disjunctive rule to guess an argument that can be added to the set and then add further arguments in order to defend all arguments in the set. We then perform certain tests to check whether the set is actually admissible. If one of the tests fails we derive an atom $\mathbf{fail}$ and force all arguments to be in the $\mathbf{sIn}$ predicate and all attacks to be in the $\mathbf{necAtt}$ predicate (which we introduce later on), to make sure there is at most one answer-set for each extension. In case all tests succeed, i.e. the set is admissible, there is a constraint ensuring that the guess does not produce an answer-set.

We first consider the construction of the superset. In a first step we test whether the admissible set already contains all arguments of the AF. In that case it is the only preferred extension and we can skip the saturation part. Otherwise, we (a) require all arguments in $\mathbf{in}$ to be also contained in $\mathbf{sIn}$ and (b) use the conditional disjunction to force that at least one of the arguments not in $\mathbf{in}$ (thus in $\mathbf{out}$) is in $\mathbf{sIn}$. [3]

$$\begin{aligned} \pi_{prf-guess}: \quad \mathbf{notTrivial} &\leftarrow \mathbf{out}(X). \\ \mathbf{sIn}(X) &\leftarrow \mathbf{in}(X), \mathbf{notTrivial}. \\ \mathbf{sIn}(X) : \mathbf{out}(X) &\leftarrow \mathbf{notTrivial}. \end{aligned}$$

For the saturation technique to work we are not allowed to use $\mathit{not}$ with any predicate that appears in the head of any rule in the saturation block, except for the $\mathit{not}$ $\mathbf{fail}$ at the very end. In particular we cannot compute the unblocked attacks w.r.t. the set stored $\mathbf{sIn}$ via a predicate for the blocked attacks (as we did for the blocked attacks w.r.t. the arguments in $\mathbf{in}$). To overcome this we compute

[3] Conditional disjunction allows for more compact saturation based encodings and investigations on AF also show computational benefits [19].

the unblocked attacks, i.e. the predicate $\mathbf{unBlocked}$, directly using conditionals in the body of the rule.

$$\pi_{unblocked}: \qquad \mathbf{unBlocked}(R) \leftarrow \mathbf{att}(R, Y), \mathbf{sIn}(X) : \mathbf{mem}(R, X).$$

Next, we make sure that the constructed set either defends all its arguments or derives **fail**. To this end we introduce a new predicate $\mathbf{necAtt}$ that holds attacks that must be unblocked in order to defend all the arguments of the set. First, we have a rule stating that if an argument $x$ of the set is attacked by $r = (S, x) \in R$ then there must be a $r' \in R$ that attacks one argument $y \in S$. If such attacks $r'$ exist, one of them is added to $\mathbf{necAtt}$, otherwise the set is obviously not admissible and we derive **fail**. Our second rule states that if a rule $r = (S, x) \in R$ is unblocked then all arguments in $S$ must be in the extension.

$$\begin{aligned}\pi_{prf-adm}: \quad \mathbf{fail}|\mathbf{necAtt}(R2) : \mathbf{att}(R2, Y), \mathbf{mem}(R1, Y) &\leftarrow \mathbf{sIn}(X), \mathbf{att}(R1, X).\\ \mathbf{sIn}(X) &\leftarrow \mathbf{necAtt}(R), \mathbf{mem}(R, X).\end{aligned}$$

Next, we have a rule that derives **fail** if the constructed set of arguments is not conflict-free, i.e. if we have a unblocked attack whose target argument is in the set.

$$\pi_{prf-cf}: \qquad \mathbf{fail} \leftarrow \mathbf{sIn}(Y), \mathbf{att}(R, Y), \mathbf{unBlocked}(R).$$

Finally, we have a fragment that completes the saturation. Whenever we derive **fail** we make sure that (a) all arguments are in $\mathbf{sIn}$ and (b) all attacks are in $\mathbf{necAtt}$. Otherwise, if we can derive *not* **fail** then we have found a larger admissible set and thus we have a constraint excluding such answer-sets.

$$\begin{aligned}\pi_{prf-spoil}: \qquad \mathbf{sIn}(X) &\leftarrow \mathbf{fail}, \mathbf{arg}(X).\\ \mathbf{necAtt}(R) &\leftarrow \mathbf{fail}, \mathbf{att}(R, X).\\ &\leftarrow \mathit{not}\ \mathbf{fail}, \mathbf{notTrivial}.\end{aligned}$$

If the predicate $\mathbf{in}$ already stores a preferred extensions, all guesses from $\pi_{prf-guess}$ will eventually derive **fail** and thus all end up with the same answer-set where all arguments are in $\mathbf{sIn}$ and all attacks are in $\mathbf{necAtt}$. Otherwise, if the set of predicate $\mathbf{in}$ is admissible but not preferred, then at least one guess from $\pi_{prf-guess}$ will not derive **fail**. For this guess, because of the constraint in $\pi_{prf-spoil}$, no answer-set will be returned. Moreover, also all the guesses which derive **fail** and thus have all arguments in $\mathbf{sIn}$ and all attacks in $\mathbf{necAtt}$, do not return an answer-set because of the subset-minimal model criteria of answer-set semantics.

Finally, we obtain the encoding $\pi_{pref}$ for preferred semantics by augmenting the encoding for admissible (or alternatively complete) semantics by all the above code fragments, i.e. $\pi_{pref} = \pi_{adm} \cup \pi_{prf-guess} \cup \pi_{unblocked} \cup \pi_{prf-adm} \cup \pi_{prf-cf} \cup \pi_{prf-spoil}$ and we have *pref*$(F) \cong \mathrm{AS}(\pi_{setaf}(F) \cup \pi_{pref})$.

### 3.3.6 Semi-Stable and Stage Semantics

We next introduce the encodings for semi-stable and stage semantics. The former starts from the encoding of admissible semantics while the latter starts from the encoding of conflict-free sets. We will again make use of the saturation technique with the additional challenge that we have to encode the range and maximize along the range.

We first define the predicate $\mathbf{sPlus}$ that holds the range of the admissible/conflict-free set stored $\mathbf{in}$. That is, we have two rules, one stating that each argument in the set is also in the range and one stating that each target of an unblocked rule is in the range.

$$\begin{aligned}\pi_{range}: \quad \mathbf{sPlus}(Y) &\leftarrow \mathbf{in}(Y).\\ \mathbf{sPlus}(Y) &\leftarrow \mathbf{att}(X,Y), not\ \mathbf{blocked}(X).\\ \mathbf{notSPlus}(Y) &\leftarrow not\ \mathbf{sPlus}(Y), \mathbf{arg}(Y).\end{aligned}$$

Before starting with the saturation technique, we test whether the admissible/conflict-free set is already stable and thus semi-stable/stage. If not we again make a guess for saturation technique, but this time, as we are maximizing the range, we guess an argument that can be added to the range.

$$\begin{aligned}\pi_{extRange}: \quad \mathbf{notStable} &\leftarrow \mathbf{arg}(Y), not\ \mathbf{sPlus}(Y).\\ \mathbf{extRange}(Y) : \mathbf{notSPlus}(Y) &\leftarrow \mathbf{notStable}.\\ \mathbf{extRange}(Y) &\leftarrow \mathbf{sPlus}(Y), \mathbf{notStable}.\end{aligned}$$

Now we have to make sure that each argument is either in the constructed extensions, i.e. in $\mathbf{sIn}$, or the target of an unblocked attack, i.e. in $\mathbf{necAtt}$. We then have an additional rule that makes sure that each attack $(S,x) \in \mathbf{necAtt}$ is unblocked by adding all arguments of $S$ to $\mathbf{sIn}$.

$$\begin{aligned}\pi_{justRange}: \quad \mathbf{sIn}(X)|\mathbf{necAtt}(R) : \mathbf{att}(R,X) &\leftarrow \mathbf{extRange}(X).\\ \mathbf{sIn}(X) &\leftarrow \mathbf{att}(R,Y), \mathbf{necAtt}(R), \mathbf{mem}(R,X).\end{aligned}$$

We next add code fragments to test whether the constructed set is actually an extension. The next fragment tests whether the constructed set is conflict-free and if not derives **fail**.

$$\begin{aligned}\pi_{satCf}: \quad \mathbf{unBlocked}(R) &\leftarrow \mathbf{att}(R,Y), \mathbf{sIn}(X) : \mathbf{mem}(R,X).\\ \mathbf{fail} &\leftarrow \mathbf{sIn}(Y), \mathbf{att}(R,Y), \mathbf{unBlocked}(R).\end{aligned}$$

The following rule test whether the constructed set is admissible and is thus only used for semi-stable but not for stage semantics.

$$\pi_{satAdm}: \quad \mathbf{fail}|\mathbf{necAtt}(R2) : \mathbf{att}(R2,Y), \mathbf{mem}(R1,Y) \leftarrow \mathbf{sIn}(X), \mathbf{att}(R1,X).$$

Finally, we complete the saturation with a code fragment that again spoils the answer-set that can derive **fail** and avoids answer-sets for guesses where one cannot derive **fail**. In comparison to preferred semantics we additionally require that whenever we derive $\mathbf{fail}$ then all arguments are added to $\mathbf{extRange}$.

$$\begin{aligned}\pi_{spoil}: \qquad \mathbf{sIn}(X) &\leftarrow \mathbf{fail}, \mathbf{arg}(X).\\ \mathbf{extRange}(X) &\leftarrow \mathbf{fail}, \mathbf{arg}(X).\\ \mathbf{necAtt}(R) &\leftarrow \mathbf{fail}, \mathbf{att}(R, X).\\ &\leftarrow \mathit{not}\ \mathbf{fail}, \mathbf{notStable}.\end{aligned}$$

Now we get our encoding $\pi_{semi}$ for semi-stable semantics by combing all the above code fragments with the encoding of admissible semantics, i.e. $\pi_{semi} = \pi_{adm} \cup \pi_{range} \cup \pi_{extRange} \cup \pi_{justRange} \cup \pi_{satCf} \cup \pi_{satAdm} \cup \pi_{spoil}$ and we have *sem*$(F) \cong \mathrm{AS}(\pi_{setaf}(F) \cup \pi_{semi})$. Moreover, to obtain the encoding $\pi_{stage}$ for stage semantics we combine all the above code fragments, except $\pi_{satAdm}$ with the encoding of conflict-free sets, i.e. $\pi_{stage} = \pi_{cf} \cup \pi_{range} \cup \pi_{extRange} \cup \pi_{justRange} \cup \pi_{satCf} \cup \pi_{spoil}$ and we have *stage*$(F) \cong \mathrm{AS}(\pi_{setaf}(F) \cup \pi_{stage})$.

# 4 Conclusion

In this work we first clarified the complexity landscape of SETAFs and provided definitions for semi-stable and stage semantics that generalize their counterparts in Dung AFs. We then provided ASP-encodings for the standard SETAF semantics as introduced in [23] as well as for the semi-stable and stage semantics. Our ASP-encodings can be executed with the clingo [20] solver and are available at `www.dbai.tuwien.ac.at/research/argumentation/aspartix/setaf.html`. Beside enumerating all extensions, the solver features brave (a.k.a. credulous) and cautious (a.k.a. skeptical) reasoning. In particular, in order to compute the grounded extension, one can perform cautious reasoning on complete semantics.

**Acknowledgments.** This research has been supported by the Austrian Science Fund (FWF): I2854, Y698.

# References

[1] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[2] Gerhard Brewka, Stefan Ellmauthaler, Hannes Strass, Johannes P. Wallner, and Stefan Woltran. Abstract dialectical frameworks. An overview. *IfCoLog Journal of Logics and their Applications*, 4(8):2263–2318, 2017.

[3] Martin Caminada, Walter A. Carnielli, and Paul E. Dunne. Semi-stable semantics. *J. Log. Comput.*, 22:1207–1254, 2012.

[4] Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Symmetric argumentation frameworks. In Lluis Godo, editor, *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2005)*, volume 3571 of *Lecture Notes in Computer Science*, pages 317–328. Springer, 2005.

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

[6] Sylvie Doutre and Jérôme Mengin. Preferred extensions of argumentation frameworks: Query answering and computation. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, volume 2083 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2001.

[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] Wolfgang Dvořák. *Computational Aspects of Abstract Argumentation*. PhD thesis, Vienna University of Technology, Institute of Information Systems, 2012.

[10] Wolfgang Dvořák and Paul E. Dunne. Computational problems in formal argumentation and their complexity. *IfCoLog Journal of Logic and its Applications*, 4(8):2557–2622, 2017.

[11] Wolfgang Dvořák, Jorge Fandinno, and Stefan Woltran. On the expressive power of collective attacks. In Sanjay Modgil, editor, *to appear in Proceedings of COMMA 20018*, 2018.

[12] Wolfgang Dvořák, Sarah Alice Gaggl, Thomas Linsbichler, and Johannes Peter Wallner. Reduction-based approaches to implement Modgil's extended argumentation frameworks. In Thomas Eiter, Hannes Strass, Miroslaw Truszczynski, and Stefan Woltran, editors, *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, volume 9060 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2015.

[13] Wolfgang Dvořák, Sarah Alice Gaggl, Johannes Peter Wallner, and Stefan Woltran. Making use of advances in answer-set programming for abstract argumentation systems. *CoRR*, abs/1108.4942, 2011. In Proceedings of the 19th International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2011).

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

[15] Wolfgang Dvořák and Stefan Woltran. On the intertranslatability of argumentation semantics. *J. Artif. Intell. Res. (JAIR)*, 41:445–475, 2011.

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

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

[18] Stefan Ellmauthaler and Hannes Strass. DIAMOND 3.0 - A native C++ implementation of DIAMOND. In Pietro Baroni, Thomas F. Gordon, Tatjana Scheffler, and Manfred Stede, editors, *Computational Models of Argument - Proceedings of COMMA 2016, Potsdam, Germany, 12-16 September, 2016.*, volume 287 of *Frontiers in Artificial Intelligence and Applications*, pages 471–472. IOS Press, 2016.

[19] Sarah Alice Gaggl, Norbert Manthey, Alessandro Ronca, Johannes Peter Wallner, and Stefan Woltran. Improved answer-set programming encodings for abstract argumentation. *TPLP*, 15(4-5):434–448, 2015.

[20] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Marius Lindauer, Max Ostrowski, Javier Romero, Torsten Schaub, and Sven Thiele. *Potassco User Guide*. Univ. Potsdam, second edition edition, 2015.

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

[22] Søren Holbech Nielsen and Simon Parsons. Computing preferred extensions for argumentation systems with sets of attacking arguments. In Paul E. Dunne and Trevor J. M. Bench-Capon, editors, *Computational Models of Argument: Proceedings of COMMA 2006, September 11-12, 2006, Liverpool, UK*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 97–108. IOS Press, 2006.

[23] Søren Holbech Nielsen and Simon Parsons. A generalization of Dung's abstract framework for argumentation: Arguing with sets of attacking arguments. In *Proc. ArgMAS*, volume 4766 of *Lecture Notes in Computer Science*, pages 54–73. Springer, 2006.

[24] Sylwia Polberg. *Developing the abstract dialectical framework*. PhD thesis, TU Wien, Institute of Information Systems, 2017.

[25] Francesca Toni and Marek Sergot. Argumentation and answer set programming. In Marcello Balduccini and Tran C. Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays in Honor of Michael Gelfond*, volume 6565 of *Lecture Notes in Computer Science*, pages 164–180. Springer, 2011.

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

## A Properties of Semi-stable and Stage Semantics.

Here we provide proofs for some basic properties of semi-stable and stage semantics in SETAFs. All the proofs are straightforward adaptations of the corresponding proofs in Dung AFs.

**Lemma 2.** *stb*$(F) \subseteq$ *sem*$(F)$*, for any SETAF* $F$.

*Proof.* Consider a SETAF $F = (A, R)$. By [23] we have that each stable extension $E$ is also preferred and thus admissible. As $E$ is stable we have that $E_R^{\oplus} = A$ and thus there cannot be another admissible set $D$ with $E_R^{\oplus} \subset D_R^{\oplus}$. Hence $E$ is semi-stable. $\square$

**Lemma 3.** *stb*$(F) \subseteq$ *stage*$(F)$*, for any SETAF* $F$

*Proof.* Consider a SETAF $F = (A, R)$. By definition, we have that each stable extension $E$ is conflict-free. As $E$ is stable we have that $E_R^{\oplus} = A$ and thus there cannot be another conflict-free set $D$ with $E_R^{\oplus} \subset D_R^{\oplus}$. Hence $E$ is semi-stable. $\square$

**Lemma 4.** *sem*$(F) \subseteq$ *pref*$(F)$*, for any SETAF* $F$

*Proof.* Consider a SETAF $F = (A, R)$. Towards a contradiction assume there is a semi-stable extension $E$ that is not preferred. Then there is a preferred extension $D$ with $E \subset D$. and an argument $x \in D$ such that $x \notin E$ and $E \not\mapsto_R x$ (otherwise there would be a conflict in $D$). As the range operator $^{\oplus}$ is monotone by definition, we have $E_R^{\oplus} \subseteq D_R^{\oplus}$ and as $x \notin E_R^{\oplus}$ we obtain that $E_R^{\oplus} \subset D_R^{\oplus}$. Hence, $E$ is not semi-stable, a contradiction to our initial assumption. $\square$

**Lemma 5.** *stage*$(F) \subseteq$ *naive*$(F)$*, for any SETAF* $F$

*Proof.* Consider a SETAF $F = (A, R)$. Towards a contradiction assume there is a stage extension $E$ that is not naive. Then there is a naive extension $D$ with $E \subset D$. and an argument $x \in D$ such that $x \notin E$ and $E \not\mapsto_R x$. As the range operator $^{\oplus}$ is monotone by definition we have $E_R^{\oplus} \subseteq D_R^{\oplus}$ and as $x \notin E_R^{\oplus}$ we obtain that $E_R^{\oplus} \subset D_R^{\oplus}$. Hence, $E$ is not stage, a contradiction to our initial assumption. $\square$

**Lemma 6.** *For any SETAF* $F = (A, R)$*, if stb*$(F) \neq \emptyset$ *then stb*$(F) =$ *sem*$(F) =$ *stage*$(F)$.

*Proof.* Consider a SETAF $F = (A, R)$ with *stb*$(F) \neq \emptyset$. First consider stage semantics. As each stable extension is stage as well, we have that the range of each stage extension must be $A$ (by the range maximality condition) and thus each stage extensions is a stable extension. Hence, *stb*$(F) =$ *stage*$(F)$. Now consider semi-stable semantics. As each stable extension is semi-stable as well, we have that the range of each semi-stable extension must be $A$ (by the range maximality condition) and thus each semi-stable extensions is a stable extension. Hence, *stb*$(F) =$ *sem*$(F)$. $\square$

## B Complexity Results

In this appendix we discuss the complexity results depicted in Table 1. We start with the complexity of evaluating the characteristic function and then discuss the results for each semantics separately.

**Proposition 1.** *When given an SETAF $F$, a set of Arguments $S$, and an argument $a$ deciding whether $a \in \mathcal{F}_F(S)$ is in* $\mathsf{L}$.

*Proof.* We simple iterate over all attacks $(B, x) \in R$ and whenever $x = a$ we again iterate over all attacks $(B', x') \in R$ and test whether $x' \in B$ and $B' \subseteq S$. If there is an attack $(B, a)$ such that there is no defending attack $(B', x')$ we have $a \notin \mathcal{F}_F(S)$ otherwise $a \in \mathcal{F}_F(S)$. Clearly the above can be implemented with a constant number of pointers and thus in $\mathsf{L}$. □

**Lemma 7.** *The results for cf semantics depicted in Table 1 hold.*

*Proof.* Consider an arbitrary SETAF $F = (A, R)$. We next give $\mathsf{L}$ upper bounds for the complexity of the non-trivial decision problems for *cf* semantics:

- $Cred_{cf}$: In order to decide credulous acceptance for an argument $a$ we simply check whether $(\{a\}, a) \notin R$. If yes then $\{a\} \in cf(F)$ and thus $a$ is credulously accepted, otherwise $a$ cannot be in any conflict-free set and thus is not credulously accepted. Thus $Cred_{cf} \in \mathsf{L}$.

- $Skept_{cf}$: No argument can be skeptically accepted w.r.t. conflict-free semantics as the empty set is always conflict-free.

- $Ver_{cf}$: To verify that a set $S$ is conflict free we simple iterate over all $(B, x) \in R$ and test whether $B \cup \{x\} \subseteq S$. If yes there is a conflict in $S$ and we terminate. If none of the attacks is contained in $S$, the set is conflict-free. This, is clearly in $\mathsf{L}$.

- $Exists_{cf}$: This is always true as the empty set is always conflict-free.

- $Exists_{cf}^{\neg\emptyset}$: To decide whether there is a non-empty conflict-free set we test each argument for being credulously accepted. If one of them is there is a non-empty extension and vice versa. Again, this is clearly in $\mathsf{L}$.

□

**Lemma 8.** *The results for naive semantics depicted in Table 1 hold.*

*Proof.* Consider an arbitrary SETAF $F = (A, R)$. We next give $\mathsf{L}$ upper bounds for the complexity of the non-trivial decision problems for *naive* semantics:

- $Cred_{naive}$: Here we can exploit that an argument is contained in at least one naive extension iff it is contained in at least one conflict-free set. Thus, the credulous acceptance problem is the same for conflict-free and naive semantics, i.e. $Cred_{naive} = Cred_{cf}$, and thus it can be solved in $\mathsf{L}$.

- *$Skept_{naive}$*: An argument $a$ is skeptically accepted w.r.t. naive semantics iff $a$ is credulously accepted and there is no attack $(B, x) \in R$ such that $a \in B \cup \{x\}$ and $(B \cup \{x\}) \setminus \{a\} \in cf(F)$. We already discussed how to check the former in L and the latter can be easily checked by iterating over all attacks, which is in L. and as we already know that testing whether a set is conflict-free is in L we can test skeptical acceptance in L.

- *$Ver_{naive}$*: We have that $S \in naive(F)$ iff $S \in cf(F)$ and for each $a \in A \setminus S$ there is an attack $(B, x) \in R$ such that $a \in B \cup \{x\}$ and $(B \cup \{x\}) \setminus \{a\} \subseteq S$. We already discussed how the check the former in L. The latter can be easily checked by iterating over all edges and thus is also in L.

- $Exists_{naive}$: As the empty set is always conflict-free there also exists a subset maximal conflict-free set, i.e. a naive extension. This question can thus be always answered yes.

- $Exists^{\neg\emptyset}_{naive}$: Again, this problem is the same as $Exists^{\neg\emptyset}_{cf}$, as there is a nonempty conflict-free set iff there is a nonempty naive extension.

□

**Lemma 9.** *The results for grd semantics depicted in Table 1 hold.*

*Proof.* The hardness results are by the corresponding results for AFs [15, 9]. For the upper bounds consider an arbitrary SETAF $F = (A, R)$.

- *$Cred_{grd}, Skept_{grd}, Ver_{grd}$*: All three problems can be solved by first computing the unique grounded extension and then perform a simple test. We can compute the grounded extension by iteratively applying the characteristic function in order to compute the least fixed-point. By Proposition 1 this can be done in polynomial time. Now for credulous and skeptical acceptance we simple check whether the argument under question is in the set, while for the verification we test whether the set under question coincides with the computed grounded extension. Thus all the problems can be decided in polynomial time.

- $Exists_{grd}$: As there is always a grounded extension, this problem can be always answered with yes.

- $Exists^{\neg\emptyset}_{grd}$: Here we exploit that the grounded extension is non-empty iff there is an argument which is not attacked at all. We can check this for each argument by iterating over all attacks, which can be done in L.

□

**Lemma 10.** *The results for stb semantics depicted in Table 1 hold.*

*Proof.* The hardness results are by the corresponding results for AFs [5]. For the upper bounds consider an arbitrary SETAF $F = (A, R)$. First consider the verification problem *$Ver_{stb}$*. We can verify that a given set $S$ is stable by first checking that it is *cf* and then for each $a \in A \setminus S$ that there is an attack $(B, a) \in R$ with $B \subseteq S$. As both can be done in $\mathsf{L}$, we obtain the $\mathsf{L}$ membership of *Ver*$_{stb}$.

The other reasoning problems for stable semantics can now be solved by the standard guess & check algorithms. That is, we first use the non-determinism to guess a set and then use a deterministic part to verify that the set is a stable extension and satisfied the desired property. For credulous acceptance we check whether the argument under question is in the set, for skeptical acceptance, in order to find a counter example, we check whether the argument under question is not in the set and for $Exists_{stb}$ and $Exists_{stb}^{\neg\emptyset}$ no further tests are required. This gives $\mathsf{NP}$ procedures for *Cred*$_{stb}$, $Exists_{stb}$, and $Exists_{stb}^{\neg\emptyset}$ and a $\mathsf{coNP}$ procedures for *Skept*$_{stb}$. □

**Lemma 11.** *The results for adm semantics depicted in Table 1 hold.*

*Proof.* The hardness results are by the corresponding results for AFs [5]. For the upper bounds consider the SETAF $F = (A, R)$. First consider *Ver*$_{adm}$. We can verify that a given set $S$ is admissible by first checking that it is *cf* and then for each $a \in S$ that $a \in \mathcal{F}_F(S)$. Both can be done in $\mathsf{L}$ (cf. Proposition 1). Now *Cred*$_{adm}$ and $Exists_{adm}^{\neg\emptyset}$ can be solved by standard guess & check algorithms (as discussed for stable semantics). Moreover, as the empty set is always admissible *Skept*$_{adm}$ is trivially false while $Exists_{adm}$ is trivially true. □

**Lemma 12.** *The results for com semantics depicted in Table 1 hold.*

*Proof.* The hardness results are by the corresponding results for AFs [4]. For the upper bounds consider an arbitrary SETAF $F = (A, R)$. First consider *Ver*$_{com}$. We can verify that a given set $S$ is complete by (a) checking that it is *cf*, (b) for each $a \in S$ check that $a \in \mathcal{F}_F(S)$, and (c) for each $a \in A \setminus S$ check that $a \notin \mathcal{F}_F(S)$. All three can be done in $\mathsf{L}$ (cf. Proposition 1). Now *Cred*$_{com}$ and $Exists_{com}^{\neg\emptyset}$ can be solved by standard guess & check algorithms. Moreover, as the grounded extension is the unique minimal complete set we have that a argument is skeptically accepted w.r.t. complete semantics iff it is contained in the grounded extension, i.e. $Skept_{com} = Skept_{grd}$. Finally, we have that $Exists_{com}$ is always true as each SETAF has a grounded extension. □

**Lemma 13.** *The results for pref semantics depicted in Table 1 hold.*

*Proof.* The hardness results are by the corresponding results for AFs [5, 8]. For the upper bounds consider an arbitrary SETAF $F = (A, R)$. First consider *Ver*$_{pref}$. We can verify that a given set $S$ is preferred by (a) checking that it is in *adm*$(F)$ and (b) checking that each $S' \supset S$ is not admissible. The former is in $\mathsf{L}$ while the latter can be solved in $\mathsf{coNP}$ by a standard guess & check algorithm (guess a set and verify that it is admissible and superset of the original set). Now *Skept*$_{pref}$ can be solved by a standard guess & check algorithm, which because of the $\mathsf{coNP}$ algorithm for verifying the extension results a $\Pi_2^{\mathsf{P}}$ algorithm. For credulous acceptance we can exploit that an argument is contained in a preferred extension iff it is contained in a admissible set, i.e. we have $Cred_{pref} = Cred_{adm}$. Similarly, there is a non-empty preferred extension iff there is a non-empty admissible set, i.e. $Exists_{pref}^{\neg\emptyset} = Exists_{adm}^{\neg\emptyset}$. Thus we have $\mathsf{NP}$ procedures for *Cred*$_{pref}$ and $Exists_{pref}^{\neg\emptyset}$. Finally, as each SETAF has a preferred extension we have that $Exists_{pref}$ is trivially true. □

**Lemma 14.** *The results for sem semantics depicted in Table 1 hold.*

*Proof.* The hardness results are by the corresponding results for AFs [15]. For the upper bounds consider an arbitrary SETAF $F = (A, R)$. First consider *Ver*$_{sem}$. We can verify that a given set $S$ is semi-stable by (a) checking that it is in $adm(F)$ and (b) checking that each $S' \subseteq A$ with $S'^{+}_{R} \supset S^{+}_{R}$ is not admissible. The former is in $\mathsf{L}$ while the latter can be solved in $\mathsf{coNP}$ by a standard guess & check algorithm (guess a set and verify that it is admissible and its range is a superset of the range of the original set). Now *Cred*$_{sem}$ and *Skept*$_{sem}$ can be solved by standard guess & check algorithms, yielding $\Sigma_2^{\mathsf{P}}$ resp. $\Pi_2^{\mathsf{P}}$ algorithms. Moreover, we have that there is a non-empty semi-stable extension iff there is a non-empty admissible set, i.e. $Exists^{\neg\emptyset}_{sem} = Exists^{\neg\emptyset}_{adm}$. Finally, as each SETAF has a semi-stable extension we have that $Exists_{sem}$ is trivially true. □

**Lemma 15.** *The results for stage semantics depicted in Table 1 hold.*

*Proof.* The hardness results are by the corresponding results for AFs [15]. For the upper bounds consider an arbitrary SETAF $F = (A, R)$. First consider *Ver*$_{stage}$. We can verify that a given set $S$ is stage by (a) checking that it is in $cf(F)$ and (b) checking that each $S' \subseteq A$ with $S'^{+}_{R} \supset S^{+}_{R}$ is not conflict-free. The former is in $\mathsf{L}$ while the latter can be solved in $\mathsf{coNP}$ by a standard guess & check algorithm (guess a set and verify that it is conflict-free and its range is a superset of the range of the original set). Now *Cred*$_{stage}$ and *Skept*$_{stage}$ can be solved by standard guess & check algorithms, which results in a $\Sigma_2^{\mathsf{P}}$, resp. in a $\Pi_2^{\mathsf{P}}$, algorithm. Moreover, we have that there is a non-empty stage extension iff there is a non-empty conflict-free set, i.e. $Exists^{\neg\emptyset}_{stage} = Exists^{\neg\emptyset}_{cf}$. Finally, as each SETAF has a stage extension we have that $Exists_{stage}$ is trivially true. □