cf2 Semantics Revisited [1]

Sarah Alice GAGGL and Stefan WOLTRAN
Institute of Information Systems 184, Vienna University of Technology, A-1040 Vienna, Austria

Abstract. Abstract argumentation frameworks nowadays provide the most popular formalization of argumentation on a conceptual level. Numerous semantics for this paradigm have been proposed, whereby *cf2* semantics has shown to nicely solve particular problems concernend with odd-length cycles in such frameworks. In order to compare different semantics not only on a theoretical basis, it is necessary to provide systems which implement them within a uniform platform. Answer-Set Programming (ASP) turned out to be a promising direction for this aim, since it not only allows for a concise representation of concepts inherent to argumentation semantics, but also offers sophisticated off-the-shelves solvers which can be used as core computation engines. In fact, many argumentation semantics have meanwhile been encoded within the ASP paradigm, but not all relevant semantics, among them *cf2* semantics, have yet been considered. The contributions of this work are thus twofold. Due to the particular nature of *cf2* semantics, we first provide an alternative characterization which, roughly speaking, avoids the recursive computation of sub-frameworks. Then, we provide the concrete ASP-encodings, which are incorporated within the ASPARTIX system, a platform which already implements a wide range of semantics for abstract argumentation.

Keywords. Abstract Argumentation. Implementation.

1. Introduction

Abstract argumentation frameworks (AFs), introduced by Dung [4], represent the most popular approach for formalizing and reasoning over argumentation problems on a conceptual level. Dung already introduced different extension-based semantics (preferred, complete, stable, grounded) for such frameworks. In addition, recent proposals tried to overcome several shortcomings observed for those original semantics. For instance, the semi-stable semantics [2] handles the problem of the possible non-existence of stable extensions, while the ideal semantics [5] is proposed as a unique-status approach (each AF possesses exactly one extension) less skeptical than the grounded extension.

Another family of semantics, the so-called SCC-recursive semantics [1], has been introduced in order to solve particular problems arising for AFs with odd-length cycles. Hereby, a recursive decomposition of the given AF along strongly connected components (SCCs) is necessary to obtain the extensions. A particular instance of the SCC-recursive semantics, the *cf2* semantics, satisfies many requirements such as the symmetric treatment of odd- and even-length cycles, and ensures that attacks from self-defeating

[1]This work was supported by the Vienna Science and Technology Fund (WWTF) under grant ICT08-028.

arguments have no influence on the selection of other arguments to be included in an extension.

This leads us to the fact that abstract argumentation actually offers an ever growing number of different semantics, and thus a uniform implementation is necessary to compare them not only on a theoretical level. Answer-Set Programming (ASP, for short) is a promising approach towards this direction, since this paradigm [9,10] allows a concise representation of concepts as Guess and Check (guess a set of arguments and check whether this set satisfies the semantics' properties) and transitive closure (important to formulate reachability). Moreover, sophisticated ASP-systems such as Smodels, DLV, Cmodels, Clasp, or ASSAT are able to deal with large problem instances [3]. Finally, the data complexity of evaluating ASP programs ranges (depending from different syntactical classes) from complexity classes P, NP, coNP up to Σ_2^P and to Π_2^P. It is thus possible to provide ASP queries which are on the same complexity level as the encoded argumentation problem (see [6] for such complexity results). Previous work [7,11,13,14] already addressed this issue and gave ASP-encodings for several argumentation semantics. In particular, the system ASPARTIX [7] provides queries for the most important types of extensions including preferred, stable, semi-stable, complete, grounded and ideal.

In this paper, we focus on the theoretical foundations towards an ASP-encoding for the $cf2$ semantics, which has been neglected in the literature so far. In particular, it turns out to be rather cumbersome to represent $cf2$ semantics directly within ASP. This is due the fact that the original definition involves a recursive computation of different subframeworks. Our aim here is, roughly speaking, to shift the need of recursion from generating subframeworks to the concept of recursively component defeated arguments. Having computed this set $\mathcal{RD}_F(S)$ for a given AF F and a set S of arguments, we construct from F an *instance* of F with respect to $\mathcal{RD}_F(S)$ such that the $cf2$ extensions of F are given by the sets S which are maximal conflict-free in their instance with respect to $\mathcal{RD}_F(S)$. As a second result, we show that the set $\mathcal{RD}_F(S)$ can be captured via a fixed-point operator; in other words, this allows to characterize $cf2$ semantics using linear recursion only. This novel characterization is then captured by a corresponding ASP-encoding, where we now are able to directly (i) guess a set S and then (ii) check whether S is maximal conflict-free in the respective instance of the given AF F. Our encodings are incorporated to the ASPARTIX system and are available on the web[2].

The remainder of the paper is organized as follows. In the next section we recall the necessary basics of argumentation frameworks and give the definition of $cf2$ semantics. In Section 3 we introduce our alternative characterization for $cf2$ semantics and in Section 4 we put this characterization to work and sketch our ASP-encodings for the $cf2$ semantics. Finally, in Section 5 we conclude with a brief discussion of related and future work.

2. Preliminaries

We first recall some basic definitions for abstract argumentation frameworks and introduce some further notations which are relevant for the rest of the paper.

[2]`www.dbai.tuwien.ac.at/research/project/argumentation/systempage/`

Definition 1 *An* argumentation framework *(AF) is a pair $F = (A, R)$, where A is a finite set of arguments and $R \subseteq A \times A$. The pair $(a, b) \in R$ means that a attacks b. A set $S \subseteq A$ of arguments* **defeats** *b (in F), if there is an $a \in S$, such that $(a, b) \in R$. An argument $a \in A$ is* **defended** *by $S \subseteq A$ (in F) iff, for each $b \in A$, it holds that, if $(b, a) \in R$, then S defeats b (in F).*

A minimal criterion for an acceptable set of arguments is to not contain an argument attacking another argument in the set. Such acceptable sets are called conflict-free, and maximal (wrt. set-inclusion) such sets will play an important role for *cf2* semantics.

Definition 2 *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is said to be* **conflict-free** *(in F), if there are no $a, b \in S$, such that $(a, b) \in R$. We denote the collection of sets which are conflict-free (in F) by $cf(F)$. $S \subseteq A$ is* **maximal conflict-free**, *if $S \in cf(F)$ and for each $T \in cf(F)$, $S \not\subset T$. We denote the collection of all maximal conflict-free sets of F by $mcf(F)$. For the empty AF $F_0 = (\emptyset, \emptyset)$, we set $mcf(F_0) = \{\emptyset\}$.*

For our purposes, we require some further formal machinery. By $SCCs(F)$, we denote the set of strongly connected components of an AF $F = (A, R)$ which identify the maximal strongly connected[3] subgraphs of F; $SCCs(F)$ is thus a partition of A. Moreover, for an argument $a \in A$, we denote by $C_F(a)$ the component of F where a occurs in, i.e. the (unique) set $C \in SCCs(F)$, such that $a \in C$. AFs $F_1 = (A_1, R_1)$ and $F_2 = (A_2, R_2)$ are called *disjoint* if $A_1 \cap A_2 = \emptyset$. Moreover, the union between (not necessarily disjoint) AFs is defined as $F_1 \cup F_2 = (A_1 \cup A_2, R_1 \cup R_2)$.

It turns out to be convenient to use two different concepts to obtain sub-frameworks of AFs. Let $F = (A, R)$ be an AF and S a set of arguments. Then, $F|_S = ((A \cap S), R \cap (S \times S))$ is the *sub-framework* of F wrt S and we also use $F - S = F|_{A \setminus S}$. We note the following relation (which we use implicitly later on), for an AF F and sets S, S': $F|_{S \setminus S'} = F|_S - S' = (F - S')|_S$. In particular, for an AF F, a component $C \in SCCs(F)$ of F and a set S we thus have $F|_{C \setminus S} = F|_C - S$.

We now give the definition of *cf2* semantics. Our definition slightly differs from (but is equivalent to) the original definition in [1].[4]

Definition 3 *Let $F = (A, R)$ be an AF and $S \subseteq A$. An argument $b \in A$ is* **component-defeated** *by S (in F), if there exists an $a \in S$, such that $(a, b) \in R$ and $a \notin C_F(b)$. The set of arguments component-defeated by S in F is denoted by $D_F(S)$.*

Definition 4 *Let $F = (A, R)$ be an argumentation framework and S a set of arguments. Then, S is a cf2 extension of F, i.e. $S \in cf2(F)$, iff*

- *in case $|SCCs(F)| = 1$, then $S \in mcf(F)$,*
- *otherwise, $\forall C \in SCCs(F)$, $(S \cap C) \in cf2(F|_C - D_F(S))$.*

In words, the recursive definition *cf2*(F) is based on a decomposition of the AF F into its $SCCs$ depending on a given set S of arguments. We illustrate the behavior of this procedure in the following example.

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

[4]$D_F(S)$, as introduced next, replaces the set "$D_F(S, E)$" and $F|_C - D_F(S)$ replaces "$F_{\downarrow UP_F(S,E)}$"; moreover, the set of undefeated arguments "$U_F(S, E)$" as used in the general schema from [1], is not required here, because the base function for *cf2* semantics does make use of this set.

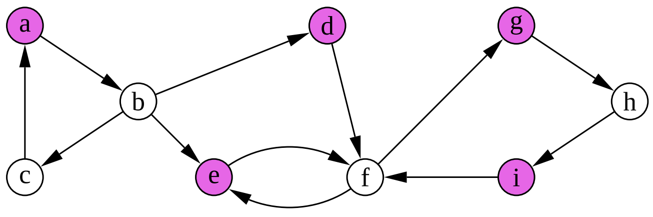

Figure 1. The argumentation framework F from Example 1.

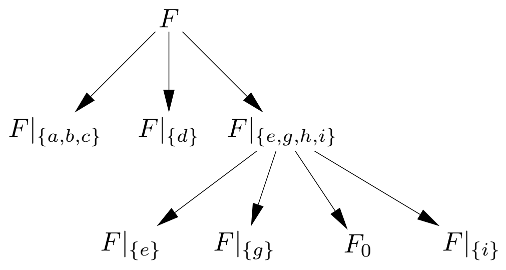

Figure 2. Tree of recursive calls for computing *cf2*(F).

Example 1 *Consider the AF $F = (A, R)$ with $A = \{a, b, c, d, e, f, g, h, i\}$ and $R = \{(a, b), (b, c), (c, a), (b, d), (b, e), (d, f), (e, f), (f, e), (f, g), (g, h), (h, i), (i, f)\}$ as illustrated in Figure 1. We want to check whether $S = \{a, d, e, g, i\}$ is a cf2 extension of F (the arguments of the set S are highlighted in Figure 1). Following Definition 4, we first identify the SCCs of F, namely $C_1 = \{a, b, c\}$, $C_2 = \{d\}$ and $C_3 = \{e, f, g, h, i\}$. Moreover, we have $D_F(S) = \{f\}$. This leads us to the following checks (see also Figure 2 which shows the involved subframeworks).*

1. *$(S \cap C_1) \in cf2(F|_{C_1})$: $F|_{C_1}$ consists of a single SCC; hence, we have to check whether $(S \cap C_1) = \{a\} \in mcf(F|_{C_1})$, which indeed holds.*
2. *$(S \cap C_2) \in cf2(F|_{C_2})$: $F|_{C_2}$ consists of a single argument d (and thus of a single SCC); $(S \cap C_2) = \{d\} \in mcf(F|_{C_2})$ thus holds.*
3. *$(S \cap C_3) \in cf2(F|_{C_3} - \{f\})$: $F|_{C_3} - \{f\} = F|_{\{e,g,h,i\}}$ consists of four SCCs, namely $C_4 = \{e\}$, $C_5 = \{g\}$, $C_6 = \{h\}$ and $C_7 = \{i\}$. Hence, we need a second level of recursion for $F' = F|_{\{e,g,h,i\}}$ and $S' = S \cap C_3$. Note that we have $D_{F'}(S') = \{h\}$. The single-argument AFs $F'|_{C_4} = F|_{\{e\}}$, $F'|_{C_5} = F|_{\{g\}}$, $F'|_{C_7} = F|_{\{i\}}$ all satisfy $(S' \cap C_i) \in mcf(F'|_{C_i})$; while $F'|_{C_6 \setminus \{h\}}$ yields the empty AF. Therfore, $(S' \cap C_6) = \emptyset \in cf2(F|_{C_6 \setminus \{h\}})$ holds as well.*

We thus conclude that S is a cf2 extension of F. Further cf2 extensions of F are $\{b, f, h\}$, $\{b, g, i\}$ and $\{c, d, e, g, i\}$.

3. An Alternative Characterization for the *cf2* Semantics

In this section, we provide an alternative characterization for the *cf2* semantics. In particular, our aim is to avoid the recursive computation of sub-frameworks (as, for instance, depicted in Figure 2) and instead collect the different sets of component-defeated arguments by a recursively defined set of arguments.

To avoid splitting an AF into sub-frameworks, we introduce the following concept.

Definition 5 *An AF $F = (A, R)$ is called* **separated** *if for each $(a, b) \in R$, $C_F(a) = C_F(b)$. We define $[[F]] = \bigcup_{C \in SCCs(F)} F|_C$ and call $[[F]]$ the* **separation** *of F.*

In words, an AF is separated if there are no attacks between different strongly connected components. Thus, the separation of an AF always yields a separated AF. The following technical lemma will be useful later.

Lemma 1 *For any AF F and set S of arguments, $\bigcup_{C \in SCCs(F)} [[F|_C - S]] = [[F - S]]$.*

Proof. We first note that for disjoint AFs F and G, $[[F]] \cup [[G]] = [[F \cup G]]$ holds. Moreover, for a set S of arguments and arbitrary AFs F and G, $(F - S) \cup (G - S) = (F \cup G) - S$ is clear. Using these observations, we obtain

$$\bigcup_{C \in SCCs(F)} [[F|_C - S]] = [[\bigcup_{C \in SCCs(F)} (F|_C - S)]] = [[(\bigcup_{C \in SCCs(F)} F|_C) - S]] = [[[[F]] - S]].$$

It remains to show that $[[[[F]] - S]] = [[F - S]]$. Obviously, both AFs possess the same arguments A. Thus let R be the attacks of $[[[[F]] - S]]$ and R' the attacks of $[[F - S]]$. $R \subseteq R'$ holds by the fact that each attack in $[[F]]$ is also contained in F. To show $R' \subseteq R$, let $(a, b) \in R'$. Then $a, b \notin S$, and $C_{F-S}(a) = C_{F-S}(b)$. From the latter, $C_F(a) = C_F(b)$ and thus (a, b) is an attack in $[[F]]$ and also in $[[F]] - S$. Again using $C_{F-S}(a) = C_{F-S}(b)$, shows $(a, b) \in R$. $\square$

Next, we define the level of recursiveness a framework shows with respect to a set S of arguments and then the aforementioned set of recursively component defeated arguments (by S) in an AF.

Definition 6 *For an AF $F = (A, R)$ and a set S of arguments, we recursively define the* **level** *$\ell_F(S)$ of F wrt S as follows:*

- *if $|SCCs(F)| = 1$ then $\ell_F(S) = 1$;*
- *otherwise, $\ell_F(S) = 1 + max(\{\ell_{F|_C - D_F(S)}(S \cap C) \mid C \in SCCs(F)\})$.*

Definition 7 *Let $F = (A, R)$ be an AF and S a set of arguments. We define the set of arguments* **recursively component defeated** *by S (in F) as follows:*

- *if $|SCCs(F)| = 1$ then $\mathcal{RD}_F(S) = \emptyset$;*
- *otherwise, $\mathcal{RD}_F(S) = D_F(S) \cup \bigcup_{C \in SCCs(F)} \mathcal{RD}_{F|_C - D_F(S)}(S \cap C)$.*

We are now prepared to give our first alternative characterization, which establishes a *cf2* extension S of a given AF F by checking whether S is maximal conflict-free in a certain separated framework constructed from F using S.

Lemma 2 *Let $F = (A, R)$ be an AF and S be a set of arguments. Then,*

$$S \in \mathit{cf2}(F) \text{ iff } S \in \mathit{mcf}([[F - \mathcal{RD}_F(S)]]).$$

Proof. We show the claim by induction over $\ell_F(S)$.

Induction base. For $\ell_F(S) = 1$, we have $|SCCs(F)| = 1$. By definition $\mathcal{RD}_F(S) = \emptyset$ and we have $[[F - \mathcal{RD}_F(S)]] = [[F]] = F$. Thus, the assertion states that $S \in \mathit{cf2}(F)$ iff $S \in \mathit{mcf}(F)$ which matches the original definition for the $\mathit{cf2}$ semantics in case the AF has a single strongly connected component.

Induction step. Let $\ell_F(S) = n$ and assume the assertion holds for all AFs F' and sets S' with $\ell_{F'}(S') < n$. In particular, we have by definition that, for each $C \in SCCs(F)$, $\ell_{F|_C - D_F(S)}(S \cap C) < n$. By the induction hypothesis, we thus obtain that, for each $C \in SCCs(F)$, the following holds:

$$(S \cap C) \in \mathit{cf2}(F|_C - D_F(S)) \text{ iff } (S \cap C) \in \mathit{mcf}\Big([[(F|_C - D_F(S)) - \mathcal{R}'_{F,C,S}]]\Big) \quad (1)$$

where $\mathcal{R}'_{F,C,S} = \mathcal{RD}_{F|_C - D_F(S)}(S \cap C)$. Let us fix now a $C \in SCCs(F)$. Since for each further $C' \in SCCs(F)$ (i.e. $C \neq C'$), no argument from $\mathcal{RD}_{F|_{C'} - D_F(S)}(S \cap C')$ occurs in $F|_C$, we have

$$\begin{aligned}
&(F|_C - D_F(S)) - \mathcal{R}'_{F,C,S} = \\
&\Big((F|_C - D_F(S)) - \mathcal{R}'_{F,C,S}\Big) - \bigcup_{C' \in SCCs(F); C \neq C'} \mathcal{RD}_{F|_{C'} - D_F(S)}(S \cap C') = \\
&\Big(F|_C - D_F(S)\Big) - \bigcup_{C \in SCCs(F)} \mathcal{RD}_{F|_C - D_F(S)}(S \cap C) = \\
&F|_C - \Big(D_F(S) \cup \bigcup_{C \in SCCs(F)} \mathcal{RD}_{F|_C - D_F(S)}(S \cap C)\Big) = F|_C - \mathcal{RD}_F(S).
\end{aligned}$$

Thus, for any $C \in SCCs(F)$, relation (1) amounts to

$$(S \cap C) \in \mathit{cf2}(F|_C - D_F(S)) \text{ iff } (S \cap C) \in \mathit{mcf}\big([[F|_C - \mathcal{RD}_F(S)]]\big). \quad (2)$$

We now prove the assertion. Let $S \in \mathit{cf2}(F)$. By definition, for each $C \in SCCs(F)$, $(S \cap C) \in \mathit{cf2}(F|_C - D_F(S))$. Using (2), we get that for each $C \in SCCs(F)$, $(S \cap C) \in \mathit{mcf}([[F|_C - \mathcal{RD}_F(S)]])$. By the definition of components and the semantics of being maximal conflict-free, the following relation thus follows:

$$\bigcup_{C \in SCCs(F)} (S \cap C) \in \mathit{mcf}\Big(\bigcup_{C \in SCCs(F)} [[F|_C - \mathcal{RD}_F(S)]]\Big).$$

Since $S = \bigcup_{C \in SCCs(F)} (S \cap C)$ and, by Lemma 1, $\bigcup_{C \in SCCs(F)} [[F|_C - \mathcal{RD}_F(S)]] = [[F - \mathcal{RD}_F(S)]]$, we arrive at $S \in \mathit{mcf}([[F - \mathcal{RD}_F(S)]])$ as desired. The other direction is by essentially the same arguments. □

Next, we provide an alternative characterization for $\mathcal{RD}_F(S)$ via a fixed-point operator. In other words, this yields a linearization in the recursive computation of this set. To this end, we require a parameterized notion of reachability.

Definition 8 *Let $F = (A, R)$ be an AF, B a set of arguments, and $a, b \in A$. We say that b is* **reachable** *in F from a* **modulo** *B, in symbols $a \Rightarrow_F^B b$, if there exists a path from a to b in $F|_B$, i.e. there exists a sequence $c_1, \ldots, c_n$ ($n > 1$) of arguments such that $c_1 = a$, $c_n = b$, and $(c_i, c_{i+1}) \in R \cap (B \times B)$, for all i with $1 \leq i < n$.*

Definition 9 *For an AF $F = (A, R)$, $D \subseteq A$, and a set S of arguments,*

$$\Delta_{F,S}(D) = \{a \in A \mid \exists b \in S : b \neq a, (b, a) \in R, a \not\Rightarrow_F^{A \setminus D} b\}.$$

The operator is clearly monotonic, i.e. $\Delta_{F,S}(D) \subseteq \Delta_{F,S}(D')$ holds for $D \subseteq D'$. As usual, we let $\Delta_{F,S}^0 = \Delta_{F,S}(\emptyset)$ and, for $i > 0$, $\Delta_{F,S}^i = \Delta(\Delta_{F,S}^{i-1})$. Furthermore, $\Delta_{F,S}$ is used to denote the lfp of $\Delta_{F,S}(\emptyset)$, which exists due to the monotonicity. We need two more lemmata before showing that $\Delta_{F,S}$ captures $\mathcal{RD}_F(S)$.

Lemma 3 *For any AF $F = (A, R)$ and any set $S \subseteq A$, $\Delta_{F,S}^0 = D_F(S)$.*

Proof. We have $\Delta_{F,S}^0 = \Delta_{F,S}(\emptyset) = \{a \in A \mid \exists b \in S : b \neq a, (b, a) \in R, a \not\Rightarrow_F^A b\}$. Hence, $a \in \Delta_{F,S}^0$, if there exists a $b \in S$, such that $(b, a) \in R$ and a does not reach b in F, i.e. $b \notin C_F(a)$. This meets exactly the definition of $D_F(S)$. □

Lemma 4 *For any AF $F = (A, R)$ and any set $S \in cf(F)$,*

$$\Delta_{F,S} = D_F(S) \cup \bigcup_{C \in SCCs(F)} \Delta_{F|_C - D_F(S), (S \cap C)}.$$

Proof. Let $F = (A, R)$. For the $\subseteq$-direction, we show by induction over $i \geq 0$ that $\Delta_{F,S}^i \subseteq D_F(S) \cup \bigcup_{C \in SCCs(F)} \Delta_{F|_C - D_F(S), (S \cap C)}$. To ease notation, we write $\bar{\Delta}_{F,S,C}$ as a shorthand for $\Delta_{F|_C - D_F(S), (S \cap C)}$, where $C \in SCCs(F)$.

Induction base. $\Delta_{F,S}^0 \subseteq D_F(S) \cup \bigcup_{C \in SCCs(F)} \bar{\Delta}_{F,S,C}$ follows from Lemma 3.

Induction step. Let $i > 0$ and assume $\Delta_{F,S}^j \subseteq D_F(S) \cup \bigcup_{C \in SCCs(F)} \bar{\Delta}_{F,S,C}$ holds for all $j < i$. Let $a \in \Delta_{F,S}^i$. Then, there exists a $b \in S$, such that $(b, a) \in R$ and $a \not\Rightarrow_F^D b$, where $D = A \setminus \Delta_{F,S}^{i-1}$. If $b \notin C_F(a)$, we have also $a \not\Rightarrow_F^A b$ and thus $a \in D_F(S)$. Hence, suppose $b \in C_F(a)$. Then, $a \notin D_F(S)$ and, since $S \in cf(F)$ and $b \in S$, also $b \notin D_F(S)$. Thus, both a and b are contained in the framework $F|_C - D_F(S)$ (and so is the attack (b, a)) for $C = C_F(a)$. Moreover, $b \in (S \cap C)$. Towards a contradiction, assume now $a \notin \bar{\Delta}_{F,S,C}$. This yields that $a \Rightarrow_{F|_C - D_F(S)}^{D'} b$ for $D' = A \setminus \bar{\Delta}_{F,S,C}$, i.e. there exist arguments $c_1, \ldots, c_n$ ($n > 1$) in $F|_C - D_F(S)$ but not contained in $\bar{\Delta}_{F,S,C}$, such that $c_1 = a$, $c_n = b$, and $(c_i, c_{i+1}) \in R$, for all i with $1 \leq i < n$. Obviously all the c_i's are contained in F as well, but since $a \not\Rightarrow_F^D b$ (recall that $D = A \setminus \Delta_{F,S}^{i-1}$), it must hold that at least one of the c_i's, say c, has to be contained in $\Delta_{F,S}^{i-1}$. By the induction hypothesis, we get $c \in \bar{\Delta}_{F,S,C}$, a contradiction.

For the $\supseteq$-direction of the claim we proceed as follows. By Lemma 3, $D_F(S) = \Delta^0_{F,S}$ and thus $D_F(S) \subseteq \Delta_{F,S}$. It remains to show $\bigcup_{C \in SCCs(F)} \Delta_{F|_C - D_F(S),(S\cap C)} \subseteq \Delta_{F,S}$. We show by induction over i that $\Delta^i_{F|_C - D_F(S),(S\cap C)} \subseteq \Delta_{F,S}$ holds for each $C \in SCCs(F)$. Thus, let us fix a $C \in SCCs(F)$ and use $\bar{\Delta}^i_{F,S,C}$ as a shorthand for $\Delta^i_{F|_C - D_F(S),(S\cap C)}$.

Induction base. Let $a \in \bar{\Delta}^0_{F,S,C}$. Then, there is a $b \in (S \cap C)$, such that b attacks a in $F' = F|_C - D_F(S)$ and $a \not\Rightarrow^{A'}_{F'} b$, where A' denotes the arguments of F', i.e. $A' = C \setminus D_F(S)$. Since $F|_C$ is built from a SCC C of F, it follows that $a \not\Rightarrow^{A \setminus D_F(S)}_F b$. Since $b \in S$, $(b, a) \in R$, and $D_F(S) = \Delta^0_{F,S}$ (Lemma 3), we get $a \in \Delta^1_{F,S} \subseteq \Delta_{F,S}$.

Induction step. Let $i > 0$ and assume $\bar{\Delta}^j_{F,S,C} \subseteq \Delta_{F,S}$ for all $j < i$. Let $a \in \bar{\Delta}^i_{F,S,C}$. Then, there is a $b \in (S \cap C)$, such that b attacks a in F' and $a \not\Rightarrow^{D'}_{F'} b$, where $D' = A' \setminus \bar{\Delta}^{i-1}_{F,S,C}$. Towards a contradiction, suppose $a \notin \Delta_{F,S}$. Since $b \in S$ and $(b, a) \in R$, it follows that there exist arguments $c_1, \ldots, c_n$ ($n > 1$) in $F \setminus \Delta_{F,S}$, such that $c_1 = a$, $c_n = b$, and $(c_i, c_{i+1}) \in R$, for all i with $1 \leq i < n$. All these c_i's are thus contained in the same component as a, and moreover these c_i's cannot be contained in $D_F(S)$, since $D_F(S) \subseteq \Delta_{F,S}$. Thus, they are contained in $F|_C - D_F(S)$, but since $a \not\Rightarrow^{D'}_{F'} b$, there is at least one such c_i, say c, contained in $\bar{\Delta}^{i-1}_{F,S,C}$. By the induction hypothesis, $c \in \Delta_{F,S}$, a contradiction. □

We now are able to obtain the desired relation.

Lemma 5 *For any AF $F = (A, R)$ and any set $S \in cf(F)$, $\Delta_{F,S} = \mathcal{RD}_F(S)$.*

Proof. The proof is by induction over $\ell_F(S)$.

Induction base. For $\ell_F(S) = 1$, $|SCCs(F)| = 1$ by Definition 6. From this and Definition 7, we obtain $\mathcal{RD}_F(S) = D_F(S) = \emptyset$. By Lemma 3, $\Delta^0_{F,S} = D_F(S) = \emptyset$. By definition, $\Delta_{F,S} = \emptyset$ follows from $\Delta^0_{F,S} = \emptyset$.

Induction step. Let $\ell_F(S) = n$ and assume the claim holds for all pairs F', $S' \in cf(F')$, such that $\ell_{F'}(S') < n$. In particular, this holds for $F' = F|_C - D_F(S)$ and $S' = (S \cap C)$, with $C \in SCCs(F)$. Note that $(S \cap C)$ is indeed conflict-free in $F|_C - D_F(S)$. By definition, $\mathcal{RD}_F(S) = D_F(S) \cup \bigcup_{C \in SCCs(F)} \mathcal{RD}_{F|_C - D_F(S)}(S \cap C)$ and by Lemma 4, $\Delta_{F,S} = D_F(S) \cup \bigcup_{C \in SCCs(F)} \Delta_{F|_C - D_F(S), S\cap C}$. Using the induction hypothesis, i.e. $\Delta_{F|_C - D_F(S), S\cap C} = \mathcal{RD}_{F|_C - D_F(S)}(S \cap C)$, the assertion follows. □

We finally reached our main result in this section, i.e. an alternative characterization for *cf2* semantics, where the need for recursion is delegated to a fixed-point operator.

Theorem 1 *For any AF F, $cf2(F) = \{S \mid S \in cf(F) \cap mcf([[F - \Delta_{F,S}]])\}$.*

Proof. The result holds by the following observations. By Lemma 2, $S \in cf2(F)$ iff $S \in mcf([[F - \mathcal{RD}_F(S)]])$. Moreover, from Lemma 5, for any $S \in cf(F)$, $\Delta_{F,S} = \mathcal{RD}_F(S)$. Finally, $S \in cf2(F)$ implies $S \in cf(F)$ (see [1], Proposition 47). □

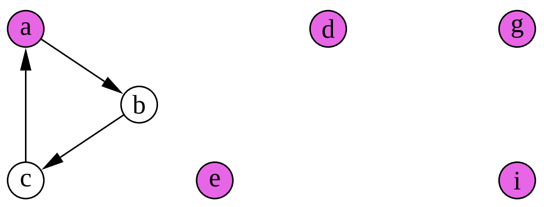

Figure 3. Graph of instance $[[F - \Delta_{F,S}]]$ of Example 2.

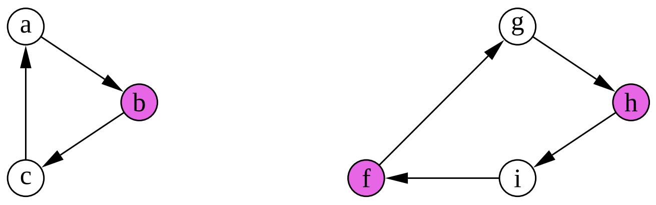

Figure 4. Graph of instance $[[F - \Delta_{F,S'}]]$ of Example 2.

Example 2 *To exemplify the behavior of $\Delta_{F,S}$ and $[[F - \Delta_{F,S}]]$, we consider the AF F and $S = \{a, d, e, g, i\}$ from Example 1. In the first iteration of computing the lfp of $\Delta_{F,S}$, we have $\Delta_{F,S}(\emptyset) = \{f\}$ because the argument f is the only one which is attacked by S but its attacker d is not reachable by f in F. In the second iteration, we obtain $\Delta_{F,S}(\{f\}) = \{f, h\}$, and in the third iteration we reach the lfp with $\Delta_{F,S}(\{f, h\}) = \{f, h\}$. Hence, $[[F - \Delta_{F,S}]]$ of the AF F wrt S is given by*

$$[[F - \Delta_{F,S}]] = \big(\{a, b, c, d, e, g, i\}, \{(a, b), (b, c), (c, a)\}\big).$$

Figure 3 shows the graph of $[[F - \Delta_{F,S}]]$. As is easily checked $S \in mcf([[F - \Delta_{F,S}]])$ as expected, since $S \in cf2(F)$. For comparison, Figure 4 shows the graph of $[[F - \Delta_{F,S'}]]$ wrt the $cf2$ extension $S' = \{b, f, h\}$ consisting of two SCCs.

4. ASP-Encodings

In this section, we first give a brief overview of ASP (to be more precise, logic programming under the answer-set semantics [8]). Then, we use our novel characterization to implement the $cf2$ semantics under this paradigm. To this end, we provide a fixed program π_{cf2} which, augmented with an input database representing a given AF F, has its answer sets in a one-to-one correspondence to the $cf2$ extensions of F. For more background on ASP, we refer to [9].

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 a constant from a domain $\mathcal{U}$. We suppose that a total order $<$ over the domain elements is available.[5] 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 *rule* r is of the form

[5]ASP-solvers as DLV [9], which is underlying our system ASPARTIX, usually provide such an order for the domain elements of the currently given program.

$$a\,\text{:-}\,b_1,\ldots,b_k,\ \mathit{not}\,b_{k+1},\ldots,\ \mathit{not}\,b_m,$$

with $m \geq k \geq 0$, and where $a, b_1, \ldots, b_m$ are atoms, and "not" stands for *default negation*. We identify the *head* of such a rule r as $H(r) = a$ and also use $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$ to denote the positive, and resp., negative body of r. A rule r is *ground* if no variable occurs in r. An *(input) database* is a set of ground rules with empty body. A program is a finite set of rules. For a program $\mathcal{P}$ and an input database D, we write $\mathcal{P}(D)$ instead of $D \cup \mathcal{P}$. $Gr(\mathcal{P})$ is the set of rules $r\sigma$ obtained by applying, to each rule $r \in \mathcal{P}$, all possible substitutions σ from the variables in $\mathcal{P}$ to the constants in $\mathcal{P}$.

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

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

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

We now turn to our encoding π_{cf2} which computes $\mathit{cf2}$ extension along the lines of Theorem 1. For a better understanding, we split π_{cf2} into several modules which we explain in an informal manner. These modules implement the following steps, given an AF $F = (A, R)$:

1. *Guess* the conflict-free sets $S \subseteq A$ of F.
2. For each S, compute the set $\Delta_{F,S}$.
3. For each S, derive the *instance* $[[F - \Delta_{F,S}]]$.
4. *Check* whether S is maximal conflict-free in $[[F - \Delta_{F,S}]]$.

To start with, let us first fix that a given AF $F = (A, R)$ is presented to π_{cf2} as a database

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

1. The guessing module. The following rules guess, when augmented by $\widehat{F}$ for an AF $F = (A, R)$, any subset $S \subseteq A$ (to be precise, for an argument $a \in A$, atom $\mathrm{in}(a)$ indicates that $a \in S$; while atom $\mathrm{out}(a)$ indicates that $a \notin S$) and then check whether the represented guess S is conflict-free in F:

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

2. The fixed-point module. Here we use the auxiliary predicates $\mathrm{inf}(\cdot)$, $\mathrm{succ}(\cdot,\cdot)$ and $\mathrm{sup}(\cdot)$ which identify an infimum, a successor function and a supremum for arguments with respect to the previously mentioned order $<$.[6] We exploit this order to iterate over the operator $\Delta_{F,S}(\cdot)$. Given $F = (A, R)$, by definition of $\Delta_{F,S}$ it is sufficient to compute at most $|A|$ such iterations to reach the fixed-point. Let us now present the module and then explain its behavior in more detail.

[6]For more details, we refer to [7], where a module $\pi_<$ is given which defines these predicates.

$$\begin{aligned}\pi_{\text{reach}} = \{ \; & \text{arg\_set}(N, X) \,\text{:-}\, \text{arg}(X), \text{inf}(N); & (3)\\ & \text{reach}(N, X, Y) \,\text{:-}\, \text{arg\_set}(N, X), \text{arg\_set}(N, Y), \text{att}(X, Y); & (4)\\ & \text{reach}(N, X, Y) \,\text{:-}\, \text{arg\_set}(N, X), \text{att}(X, Z), \text{reach}(N, Z, Y); & (5)\\ & \text{d}(N, X) \,\text{:-}\, \text{arg\_set}(N, Y), \text{arg\_set}(N, X), \text{in}(Y), \text{att}(Y, X),\\ & \qquad\qquad \textit{not}\, \text{reach}(N, X, Y); & (6)\\ & \text{arg\_set}(M, X) \,\text{:-}\, \text{arg\_set}(N, X), \textit{not}\, \text{d}(N, X), \text{succ}(N, M) \;\}. & (7)\end{aligned}$$

Rule (3) first copies all arguments into a set indexed by the infimum which initiates the computation. The remaining rules are applicable to arbitrary indices, whereby rule (7) copies (a subset of the) arguments from the currently computed set into the "next" set using the successor function $\text{succ}(\cdot, \cdot)$. This guarantees a step-by-step computation of $\text{arg\_set}(i, \cdot)$ by incrementing the index i. The functioning of rules (4)–(7) is as follows. Rules (4) and (5) compute a predicate $\text{reach}(n, x, y)$ indicating that there is a path from argument x to argument y in the given framework *restricted* to the arguments of the current set n. In rule (6), $\text{d}(n, x)$ is obtained for all arguments x which are component-defeated by S in this restricted framework. In other words, if n is the i-th argument in the order $<$, $\text{d}(n, x)$ carries exactly those arguments x which are contained in $\Delta^i_{F,S}$. Finally, rule (7) copies arguments from the current set which are *not* component-defeated to the successor set.

3. The instance module. As already outlined above, if the supremum m is reached in π_{reach}, we are guaranteed that the derived atoms $\text{arg\_set}(m, x)$ characterize exactly those arguments x from the given AF which are not contained in $\Delta_{F,S}$. It is thus now relatively easy to obtain the instance $[[F - \Delta_{F,S}]]$ which is done below via predicates $\text{arg\_new}(\cdot)$ and $\text{att\_new}(\cdot, \cdot)$.

$$\begin{aligned}\pi_{inst} = \{ \; & \text{arg\_new}(X) \,\text{:-}\, \text{arg\_set}(M, X), \text{sup}(M);\\ & \text{att\_new}(X, Y) \,\text{:-}\, \text{arg\_new}(X), \text{arg\_new}(Y), \text{att}(X, Y),\\ & \qquad\qquad \text{reach}(M, Y, X), \text{sup}(M) \;\}.\end{aligned}$$

4. The checking module. It remains to verify whether the initially guessed set S is a *cf2* extension. To do so, we need to check whether S is maximal conflict-free in the instance $[[F - \Delta_{F,S}]]$. The following module does this job by checking whether only those arguments are not contained in S, for which an addition to S would yield a conflict.

$$\begin{aligned}\pi_{mcf} = \{ \; & \text{conflicting}(X) \,\text{:-}\, \text{att\_new}(Y, X), \text{out}(X), \text{in}(Y);\\ & \text{conflicting}(X) \,\text{:-}\, \text{att\_new}(X, Y), \text{out}(X), \text{in}(Y);\\ & \text{conflicting}(X) \,\text{:-}\, \text{att\_new}(X, X);\\ & \text{:-}\, \textit{not}\, \text{conflicting}(X), \text{out}(X), \text{arg\_new}(X) \;\}.\end{aligned}$$

We now have our entire encoding $\pi_{cf2} = \pi_{cf} \cup \pi_{<} \cup \pi_{\text{reach}} \cup \pi_{inst} \cup \pi_{mcf}$ available (recall that we have not given here the definition of $\pi_{<}$; see [7] for the details). The desired correspondence between answer-sets and *cf2* extensions is as follows.

Theorem 2 *Let F be an AF. Then, (i) for each $S \in cf2(F)$, there is an $I \in \mathcal{AS}(\pi_{cf2}(\widehat{F}))$ with $S = \{a \mid \mathrm{in}(a) \in I\}$; (ii) for each $I \in \mathcal{AS}(\pi_{cf2}(\widehat{F}))$, $\{a \mid \mathrm{in}(a) \in I\} \in cf2(F)$.*

5. Discussion and Conclusions

In this paper, we introduced an alternative characterization for the $cf2$ semantics which is based on a certain fixed-point operator in order to avoid the more involved recursions from the original definition [1]. This new characterization allowed us to provide a relatively succinct ASP-encoding for computing $cf2$ extensions which has been incorporated to the ASP-based argumentation system ASPARTIX. Extending our techniques to other SCC-recursive semantics [1] is ongoing work.

Previous work [12] has shown that $cf2$ extensions can be characterized using a different (however, not implemented) semantics for logic programs. In the same paper, complexity results for $cf2$ semantics have been reported, in particular that the verification problem (i.e. checking whether a given set is a $cf2$ extension) can be decided in polynomial time. We note that this result is reflected in our encodings by the fact that (unstratified) negation is only used for guessing a candidate set, while the verification part does not contain any costly programming concepts (in particular, we could avoid the use of disjunction which is necessary to capture more involved semantics; see [7] for details).

References

[1] P. Baroni, M. Giacomin, and G. Guida. SCC-Recursiveness: A General Schema for Argumentation Semantics. *Artif. Intell.*, 168(1-2):162–210, 2005.

[2] M. Caminada. Semi-Stable Semantics. *Proc. COMMA'06*, volume 144 of *FAIA*, pages 121–130. IOS Press, 2006.

[3] M. Denecker, J. Vennekens, S. Bond, M. Gebser, and M. Truszczynski. The Second Answer Set Programming Competition. *Proc. LPNMR'09*, volume 5753 of *LNCS*, pages 637–654. Springer, 2009.

[4] P. M. Dung. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artif. Intell.*, 77(2):321–358, 1995.

[5] P. M. Dung, P. Mancarella, and F. Toni. Computing Ideal Sceptical Argumentation. *Artif. Intell.*, 171(10-15):642–674, 2007.

[6] P. E. Dunne and M. Wooldridge. Complexity of Abstract Argumentation. *Argumentation in Artificial Intelligence*, pages 85–104. Springer, 2009.

[7] U. Egly, S. A. Gaggl, and S. Woltran. Answer-Set Programming Encodings for Argumentation Frameworks. Accepted for publication in *Argument and Computation*. Available as Technical Report DBAI-TR-2008-62, Technische Universität Wien, 2008.

[8] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[9] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, Simona Perri, and Francesco Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[10] I. Niemelä. Logic Programming with Stable Model Semantics as a Constraint Programming Paradigm. *Ann. Math. Artif. Intell.*, 25(3–4):241–273, 1999.

[11] J. C. Nieves, M. Osorio, and U. Cortés. Preferred Extensions as Stable Models. *Theory and Practice of Logic Programming*, 8(4):527–543, 2008.

[12] J. C. Nieves, M. Osorio, and C. Zepeda. Expressing Extension-Based Semantics Based on Stratified Minimal Models. *Proc. WoLLIC'09*, volume 5514 of *LNCS*, pages 305–319. Springer, 2009.

[13] M. Osorio, C. Zepeda, J. C. Nieves, and U. Cortés. Inferring Acceptable Arguments with Answer Set Programming. *Proc. ENC'05*, pages 198–205. IEEE Computer Society, 2005.

[14] T. Wakaki and K. Nitta. Computing Argumentation Semantics in Answer Set Programming. *Proc. JSAI'08*, volume 5447 of *LNCS*, pages 254–269. Springer, 2008.