Decomposing Abstract Dialectical Frameworks

Sarah Alice GAGGL
Technische Universität Dresden, Computational Logic Group, Germany

Hannes STRASS
Computer Science Institute, Leipzig University, Germany

Abstract We introduce a decomposition scheme for abstract dialectical frameworks (ADFs). The decomposition proceeds along the ADF's strongly connected components. For several semantics, the decomposition-based version coincides with the original semantics. For others, the scheme defines new semantics. These new semantics allow us to deal with pertinent problems such as odd-length negative cycles in a more general setting, that for instance also encompasses logic programs.

Keywords. abstract dialectical frameworks, decomposition, strongly connected components, abstract argumentation

1. Introduction

The abstract dialectical framework (ADF) as introduced by Brewka and Woltran [3] is a generalization of the well studied abstract argumentation framework (AF) [5]. ADFs capture everything that is possible with AFs and allow for more general interactions between arguments, e.g. support, joint attack, joint support and mixed combinations.

This generality is achieved by using acceptance conditions for the statements, i.e. Boolean functions determining the acceptance of a statement s depending on the acceptance of its parents. These acceptance functions can also be represented as propositional formulas φ_s, thus the status of a statement s can be obtained by the evaluation of the propositional formula φ_s. For example, the AF-like relationship where statements a and b individually attack c can then be expressed by $\varphi_c = \neg a \wedge \neg b$. That is, c is accepted (true) if neither of its attackers is accepted (true). A set attack from a and b to c is written as $\varphi_c = \neg a \vee \neg b$ where c is only rejected if both a and b are accepted. The same works for support: $\varphi_c = a \wedge b$ means that c needs support from both a and b, and $\varphi_c = a \vee b$ says that c can be accepted if at least one of a or b is accepted. Most of the semantics of ADFs are defined over the acceptance conditions, however additionally the links between the statements are explicitly represented in the same way as it is done in AFs. This does not only have the advantage of the handy representation as a directed graph, it also provides information about the structure of an ADF, like cycles and strongly connected components (SCCs).

As usual the greater expressiveness of a formalism comes with a price. In our case the computational complexity of semantics for ADFs is in general higher than for

AFs [9]. A successful way of dealing with big or complex problems is to split them into smaller sub-problems where it is easier to find a solution. The overall solution then consists of a combination of the solutions of all sub-problems. Here, we propose an approach to decompose ADFs along their SCCs. While our approach is inspired by similar work on AFs by Baroni et al. [1], there are important differences. First, the SCC-recursive schema for AFs is based on a recursive decomposition of an AF along its SCCs, where in each step the semantics are computed for sub-frameworks consisting of single SCCs. The SCCs of an AF can change during the computation, depending on the outcome of the semantics from the previous SCCs. In particular, arguments which are attacked from outside their SCCs from an accepted argument are eliminated, which can change the remaining SCCs of the framework. As ADFs allow complex acceptance conditions for statements one needs a way to pass on the outcome of preceding components to all acceptance conditions of statements depending on them and additionally handle the change of SCCs. Second, as acceptance conditions can be formulated as propositional formulas, there might be redundancies in the representation. For instance, the formula $\varphi_s = a \vee \mathbf{t}$ always evaluates to true, hence a is redundant in φ_s and can be removed. However this redundant information would also be given in the links of the ADF and may lead to dependencies in the graph which are actually not present. Hence, a pure decomposition along SCCs would not work correctly. Third, in the AF case, some semantics are defined in a simplified version of the general SCC-recursive schema, namely for stable, $cf2$ and $stage2$ the notion of defense is somehow weakened [6].

The main contributions of this work are the following. We propose a recursive procedure to compute semantics for ADFs along SCCs which allows to propagate already obtained information on the acceptance state of statements to others which depend on them. Within these propagation steps, redundant information is identified and eliminated. It turns out that our approach is indeed a generalization of the SCC-recursive schema for AFs, as it allows to compute all standard admissible-based semantics and the naive-based ones within the same procedure. Hence, it can also be seen as an alternative characterization of the general SCC-recursive schema like the one for $cf2$ and $stage2$ semantics presented in [7, 6]. Finally, we briefly consider the converse problem of *composing* ADFs. Here, it turns out that there seems to be no single composition operator that works as desired for all possible ADFs.

2. Background

We will make use of many standard concepts of classical propositional logic in this paper, including the usual notions of formulas, interpretations and models, as well as satisfiability and refutability. Our analysis in this paper will be based on three-valued interpretations, mappings $v : S \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ that assign one of the truth values true ($\mathbf{t}$), false ($\mathbf{f}$) or unknown ($\mathbf{u}$) to each statement. A comparable treatment for AFs was given by the three-valued argumentation stages of Verheij [10]. For uniformity among logic-based and argumentation-based formalisms, in this paper we use standard notation and terminology from mathematical logic.

The three truth values are partially ordered by $\leq_i$ according to their information content: we have $\mathbf{u} <_i \mathbf{t}$ and $\mathbf{u} <_i \mathbf{f}$ and no other pair in $<_i$, which intuitively means that the classical truth values contain more information than the truth value unknown.

The information ordering $\leq_i$ extends in a straightforward way to valuations v_1, v_2 over S in that $v_1 \leq_i v_2$ iff $v_1(s) \leq_i v_2(s)$ for all $s \in S$. The $\leq_i$ least element of the set of all valuations is the valuation mapping all statements to unknown – the least informative interpretation. Obviously, a three-valued interpretation v is two-valued if all statements are mapped to either true or false. Such two-valued interpretations are $\leq_i$-maximal.

A particular non-standard notion we use is that of the partial evaluation of a formula. Given a three-valued interpretation v and a formula φ, the partial evaluation of φ with v takes the two-valued part of v and replaces the evaluated variables by their truth values.

Definition 1. Let φ be a propositional formula over vocabulary S and for an $M \subseteq S$ let $v : M \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ be a three-valued interpretation. The *partial valuation of φ by v* is $\varphi^v = \varphi[p/\mathbf{t} : v(p) = \mathbf{t}][p/\mathbf{f} : v(p) = \mathbf{f}]$.

For example, consider the propositional formula $\varphi = a \vee (b \wedge c)$ and the interpretation $v_1 = \{a \mapsto \mathbf{f}, b \mapsto \mathbf{t}, c \mapsto \mathbf{u}\}$. Statement c with $v_1(c) = \mathbf{u}$ will remain in φ_s, while a and b are replaced, and we get $\varphi^{v_1} = \mathbf{f} \vee (\mathbf{t} \wedge c)$. This formula is equivalent to c and thus both satisfiable (by $\{c \mapsto \mathbf{t}\}$) and refutable (by $\{c \mapsto \mathbf{f}\}$). In contrast, for $v_2 = \{a \mapsto \mathbf{t}, b \mapsto \mathbf{u}, c \mapsto \mathbf{u}\}$ the formula $\varphi^{v_2} = \mathbf{t} \vee (b \wedge c)$ is irrefutable; for $v_3 = \{a \mapsto \mathbf{f}, b \mapsto \mathbf{f}, c \mapsto \mathbf{u}\}$ the formula $\varphi^{v_3} = \mathbf{f} \vee (\mathbf{f} \wedge c)$ is unsatisfiable.

2.1. Abstract Argumentation Frameworks

In this section we introduce the basics of abstract argumentation and the semantics we need for further investigations. We first give the formal definition of abstract argumentation frameworks as introduced by Dung [5].

Definition 2. An *argumentation framework (AF)* is a pair $F = (A, R)$, where A is a finite set of arguments and $R \subseteq A \times A$ is the attack relation. The pair $(a, b) \in R$ means that a attacks b. A set $S \subseteq A$ of arguments attacks b (in F), if there is an $a \in S$ such that $(a, b) \in R$.

The conflicts between the arguments are solved on a semantical level. An argument can either be accepted, rejected or it is undecided whether to accept or reject the argument. Here we will use the notion of labelings, as they directly correspond to three-valued interpretations of ADFs. For an overview about labelings for most argumentation semantics we refer to [2]. Thus, accepted arguments are labeled with $\mathbf{t}$ (true), rejected ones with $\mathbf{f}$ (false) and undecided ones with $\mathbf{u}$.

For an AF $F = (A, R)$, a *labeling* is a total function $v : A \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$. Then, a labeling can be denoted as a triple $v = (v_\mathbf{t}, v_\mathbf{f}, v_\mathbf{u})$, where $v_l = \{a \in A \mid v(a) = l\}$. Following [2] conflict-free and naive labelings are given as follows. v is a *conflict-free labeling* of F, i.e. $v \in \mathit{cfi}(F)$, iff (i) for all $a \in v_\mathbf{t}$ there is no $b \in v_\mathbf{t}$ such that $(a, b) \in R$, (ii) for all $a \in v_\mathbf{f}$ there exists a $b \in v_\mathbf{t}$ such that $(b, a) \in R$. Then, v is a *naive labeling* of F, i.e. $v \in \mathit{nai}(F)$, iff $v \in \mathit{cfi}(F)$ and there is no $v' \in \mathit{cfi}(F)$ with either $v_\mathbf{t} \subset v'_\mathbf{t}$ or $v_\mathbf{f} \subset v'_\mathbf{f}$.

The *cf2* semantics is based on a decomposition along the SCCs of an AF. Hence, we require some further formal machinery. By $\mathit{SCCs}(F)$, we denote the set of *strongly connected components* of an AF $F = (A, R)$, i.e. sets of vertices of the maximal strongly

connected[1] sub-graphs of F; Moreover, for an $a \in A$, 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 $a \in C$. 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 $S \subseteq A$. Then, $F|_S = ((A \cap S), R \cap (S \times S))$ is the *sub-framework* of F w.r.t. 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': $F|_{S \setminus S'} = F|_S - S' = (F - S')|_S$. We now give the definition of the *cf2* semantics in form of labelings [2].

Definition 3. Let $F = (A, R)$ be an AF and v be a labeling of F. A $b \in A$ is *component-defeated* by $v_{\mathbf{t}}$ (in F), if $\exists a \in v_{\mathbf{t}}$, s.t. $(a, b) \in R$ and $a \notin C_F(b)$. The set of arguments component-defeated by v in F is denoted by $D_F(v_{\mathbf{t}})$.

Then, v is a *cf2 labeling* of F, i.e. $v \in cf2(F)$, iff

- $v \in nai(F)$, in case $|SCCs(F) = 1|$;
- otherwise, $\forall C \in SCCs(F), v|_{C \setminus D_F(v_{\mathbf{t}})} \in cf2(F|_C - D_F(v_{\mathbf{t}}))$, $D_F(v_{\mathbf{t}}) \subseteq v_{\mathbf{f}}$.

Further AF semantics exist; to save space we define them implicitly via ADFs.

2.2. Abstract Dialectical Frameworks

An abstract dialectical framework (ADF) is a directed graph whose nodes represent statements or positions which can be accepted or not. The links represent dependencies: the status of a node s only depends on the status of its parents $par(s)$, that is, the nodes with a direct link to s. Each node s has an associated acceptance condition C_s specifying the exact conditions under which s is accepted. C_s is a function assigning to each subset of $par(s)$ one of the truth values $\mathbf{t}$, $\mathbf{f}$. Intuitively, if for some $R \subseteq par(s)$ we have $C_s(R) = \mathbf{t}$, then s will be accepted provided the nodes in R are accepted and those in $par(s) \setminus R$ are not accepted.

Definition 4. An *abstract dialectical framework* is a tuple $D = (S, L, C)$ where

- S is a set of statements (positions, nodes),
- $L \subseteq S \times S$ is a set of links,
- $C = \{C_s\}_{s \in S}$ is a collection of total functions $C_s : 2^{par(s)} \to \{\mathbf{t}, \mathbf{f}\}$, one for each statement s. The function C_s is called *acceptance condition of* s.

It is often convenient to represent acceptance conditions as propositional formulas; we will do so in this paper. There, each C_s is represented by a propositional formula φ_s over $par(s)$. Then, clearly, for $M \subseteq par(s)$ we have $C_s(M) = \mathbf{t}$ iff $M \models \varphi_s$. In this way, AFs are recast as ADFs thus: For an AF $F = (A, R)$, the ADF *associated to* F is $D_F = (A, R, C)$ with $C = \{\varphi_a\}_{a \in A}$ and $\varphi_a = \bigwedge_{(b,a) \in R} \neg b$ for $a \in A$. Intuitively, an AF argument is accepted if and only if none of its attackers is accepted.

It may be the case that a link $(r, s) \in L$ in an ADF bears no actual significance. Formally, r *is redundant in* φ_s if and only if there is no two-valued interpretation $v : par(s) \setminus \{r\} \to \{\mathbf{t}, \mathbf{f}\}$ such that $v(\varphi_s^{\{r \mapsto \mathbf{t}\}}) \neq v(\varphi_s^{\{r \mapsto \mathbf{f}\}})$. That is, if (r, s) is redundant then r has no influence on the truth value of φ_s whatsoever.

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

Several semantics can be defined by using three-valued interpretations v to partially evaluate acceptance formulas φ_s. While this style of definition is novel, the resulting semantics have mostly appeared in the literature before [4]. Some others are new, but straightforward to define [8]; these are the (three-valued) conflict-free, naive and stage semantics.

Definition 5. Let $D = (S, L, C)$ be an ADF. A three-valued interpretation v is

- *admissible* iff for each $s \in S$ we have:
  - $v(s) = \mathbf{t}$ implies that φ_s^v is irrefutable,
  - $v(s) = \mathbf{f}$ implies that φ_s^v is unsatisfiable;
- *preferred* iff it is $\leq_i$-maximal with respect to being admissible;
- *complete* iff for each $s \in S$ we have:
  - $v(s) = \mathbf{t}$ if and only if φ_s^v is irrefutable,
  - $v(s) = \mathbf{f}$ if and only if φ_s^v is unsatisfiable;
- *grounded* iff v is the $\leq_i$-least complete interpretation;
- *conflict-free* iff for all $s \in S$ we have:
  - $v(s) = \mathbf{t}$ implies that φ_s^v is satisfiable,
  - $v(s) = \mathbf{f}$ implies that φ_s^v is unsatisfiable;
- *naive* iff it is $\leq_i$-maximal with respect to being conflict-free;
- *stage* iff the set $v_{\mathbf{u}}$ is $\subseteq$-minimal with respect to being conflict-free.

A two-valued interpretation v is a *model of* D iff for all $s \in S$ we find $v(s) = v(\varphi_s)$.

Intuitively, an interpretation v is admissible if it can justify the definite stances it takes: for example, whenever v judges a statement s to be true, then this must be justified by the statement's acceptance formula. This justification can take into consideration the definite assignments of v, but must be valid no matter how the undecided statements of v are interpreted. This is elegantly achieved by checking the refutability of the partial evaluation φ_s^v of the acceptance formula of s. Complete interpretations are then the ones whose recommendations are exactly in accordance with the refutability/satisfiability status of v's assignments. The grounded semantics can consequently be seen as the greatest possible consensus between all acceptable ways of interpreting the ADF at hand. The three-valued notion of conflict-freeness is clearly a weaker version of admissibility, where truth of a statement has to be justified not by irrefutability, but only by satisfiability. (The justification standard for rejected statements is the same.) As usual, naive and stage are then those conflict-free interpretations which are information-maximal or undecided-minimal, respectively. A model of an ADF is simply a two-valued complete interpretation. All of these semantics are proper generalizations of the same semantics for AFs [4, 8].

Example 1. Let $D = (S, L, C)$ be an ADF with $S = \{a, b, c\}$, $L = \{(a, b), (b, c), (c, a)\}$ and the acceptance conditions $\varphi_a = \neg c$, $\varphi_b = \neg a$ and $\varphi_c = \neg b$. (Note that this is an AF-based ADF with an attack cycle of length three.) Some conflict-free interpretations of D are $v_1 = \{a \mapsto \mathbf{u}, b \mapsto \mathbf{u}, c \mapsto \mathbf{u}\}$, $v_2 = \{a \mapsto \mathbf{u}, b \mapsto \mathbf{t}, c \mapsto \mathbf{f}\}$, $v_2 = \{a \mapsto \mathbf{t}, b \mapsto \mathbf{f}, c \mapsto \mathbf{u}\}$, and $v_4 = \{a \mapsto \mathbf{f}, b \mapsto \mathbf{u}, c \mapsto \mathbf{t}\}$. (There are six further conflict-free interpretations.) We have a closer look at interpretation v_4.

- As $v_4(a) = \mathbf{f}$, according to the definition of conflict-free interpretations, $\varphi_a^{v_4}$ needs to be unsatisfiable. Thus we construct the partial valuation of $\varphi_a = \neg c$ by v_4 and obtain $\varphi_a^{v_4} = \neg\mathbf{t}$, which indeed is unsatisfiable.
- As $v_4(c) = \mathbf{t}$, the formula $\varphi_c^{v_4} = \neg b$ needs to be satisfiable, which holds.

On the other hand, consider $v_5 = \{a \mapsto \mathbf{t}, b \mapsto \mathbf{u}, c \mapsto \mathbf{f}\}$, which is not conflict-free, as $v_5(c) = \mathbf{f}$ but $\varphi_c^{v_5} = \neg b$ is satisfiable. The naive interpretations of D are v_2, v_3 and v_4 because they are $\leq_i$-maximal with respect to being conflict-free.

3. Decomposing ADFs

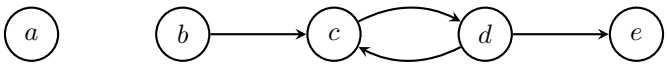
The decomposition along the SCCs of an ADF D can not be performed in the same way as it is done for AFs. If one looks at a set $M \subseteq S$ of statements, the acceptance conditions of the statements in M might still depend on statements which are not contained in M, even if M forms an SCC. To be able to decompose and evaluate an ADF, it is necessary to modify the acceptance conditions of the statements, in a way that they only depend on statements also contained in the same component. This modification will be performed depending on the decided truth values of the parents of statements.

We propose a procedure which propagates truth values from independent parts of an ADF to the rest of the ADF. We need to take several facts into account. First, we might choose to use three-valued interpretations (labellings) to represent the current acceptance status of statements. As the acceptance conditions of ADFs are defined as two-valued functions, we can not pass on the truth value $\mathbf{u}$, but we make a statement s forcibly undecided by changing its acceptance formula to $\neg s$. Second, by fixing the truth value of some statements, we might produce *redundancies* in the acceptance conditions of other statements. Eliminating these redundancies from the links and the acceptance formulas is one of the crucial points in the procedure, because by doing so, the dependencies of the statements can change, which has an important influence on the subsequent calls.

Definition 6. Let $D = (S, L, C)$ be an ADF and $p, s \in S$. We say that s *depends on* p if there is a path from p to s in L but no path from s to p in L. Now let $M \subseteq S$. A statement $s \in S$ is *independent modulo* M iff for each $p \in S$, if s depends on p then $p \in M$. A set $M \subseteq S$ is *independent* iff there is no $s \in M$ that depends on a $p \in S \setminus M$. Lastly, define $ind_D(M) = \{s \in S \mid s \text{ is independent modulo } M \text{ in } D\}$.

Note that dependence here implicitly speaks about strongly connected components (SCCs). Intuitively speaking, statements do not depend on statements in their own SCC, but on all statements in previous SCCs. The function ind_D returns the set of all statements which are independent modulo the input set. This function is $\subseteq$-monotone, that is, for $M \subseteq N \subseteq S$, we find $ind_D(M) \subseteq ind_D(N)$. Note furthermore that independence is not concerned with acceptance conditions at all, but purely relies on the topology of the ADF.

Example 2. Consider an ADF D with the statements and links given graphically:



We initially have $ind_D(\emptyset) = \{a, b\} = M_0$. Then $ind_D(M_0) = \{a, b, c, d\} = M_1$ and finally $ind_D(M_1) = \{a, b, c, d, e\}$.

Given an independent subset M of statements of an ADF, ignoring all other statements again yields an ADF.

Definition 7. Let $D = (S, L, C)$ be an ADF and $M \subseteq S$ be an independent set. The ADF D *restricted to* M is given by $D|_M = (M, L \cap (M \times M), \{\varphi_s\}_{s \in M})$.

Note that $D|_M$ really is an ADF since its acceptance formulas by presumption do not mention statements not in M.

We next define how to reduce an ADF given a subset M of its statements and an interpretation of this subset. The intuition is that the truth values of statements in M are fixed and can be propagated into the rest of the ADF. For this definition recall from Definition 1 that for a propositional formula φ and a three-valued interpretation v of parts of its signature, φ^v denotes the formula φ where atoms that v maps into $\{\mathbf{t}, \mathbf{f}\}$ have been replaced by their truth values. Through such replacements it may happen that links become redundant. For example, consider the acceptance formula $\varphi_s = a \vee (b \wedge c)$ and the interpretation $v = \{a \mapsto \mathbf{u}, b \mapsto \mathbf{f}, c \mapsto \mathbf{u}\}$. The reduced formula is $\varphi_s^v = a \vee (\mathbf{f} \wedge c)$. This formula is equivalent to a and thus c is redundant in φ_s^v. The identification and removal of such redundant parents is an important ingredient of the following definition.

Definition 8. Let $D = (S, L, C)$ be an ADF, $M \subseteq S$ and $v : M \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$. The ADF D *reduced with* v *on* M is given by $[\![D]\!]_M^v = (S, [\![L]\!]_M^v, \{[\![\varphi_s]\!]_M^v\}_{s \in S})$ with

$$[\![\varphi_s]\!]_M^v = \begin{cases} \mathbf{t} & \text{if } s \in M \text{ and } v(s) = \mathbf{t} \\ \mathbf{f} & \text{if } s \in M \text{ and } v(s) = \mathbf{f} \\ \neg s & \text{if } s \in M \text{ and } v(s) = \mathbf{u} \\ \varphi_s^v[r/\mathbf{t} : r \text{ is redundant in } \varphi_s^v] & \text{otherwise} \end{cases}$$

$$[\![L]\!]_M^v = (L \setminus \{(r, s) \in L \mid r \text{ is redundant in } [\![\varphi_s]\!]_M^v\}) \cup \{(s, s) \mid v(s) = \mathbf{u}\}.$$

That is, $[\![L]\!]_M^v$ is L without redundant links. The new acceptance formulas in ADF $[\![D]\!]_M^v$ fix the truth values of statements in M as v assigns them. Furthermore, the classical ones among these truth values are fixed in acceptance formulas that mention statements in M. Should such replacements make other statements redundant, then these are replaced by a fixed truth value to make the redundancy explicit. In the example above, the partially evaluated formula $a \vee (\mathbf{f} \wedge c)$ is further transformed into $a \vee (\mathbf{f} \wedge \mathbf{t})$, that is, former parent c is replaced by $\mathbf{t}$. (Since the parent is redundant, it is immaterial which truth value is actually used.) Whenever $par(s) \cap M = \emptyset$, that is, the parents of s are not affected by v, then $[\![\varphi_s]\!]_M^v = \varphi_s$, that is, the acceptance formula of s does not change.

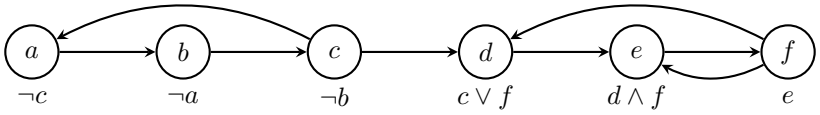
Now we present the final ingredient of our decomposition-based scheme, the most important definition of the paper. It describes the actual recursion that is used to assign to a given ADF semantics σ a new semantics σ_2.

Definition 9. Let $D = (S, L, C)$ be an ADF and σ a semantics. Define a set of interpretations as follows: $\sigma_2(D) = \sigma_2(ind_D(\emptyset), D)$, where

$$\sigma_2(M,D) = \begin{cases} \sigma(D) & \text{if } M = S \\ \bigcup\limits_{w \in \sigma(D|_M)} \sigma_2\left(ind_{[\![D]\!]^w_M}(M), [\![D]\!]^w_M\right) & \text{otherwise.} \end{cases}$$

The basic underlying intuition of this definition is to recursively decompose a given ADF along its independent statements. We start out with all statements which are independent modulo the empty set, $M_0 = ind_D(\emptyset)$. We now look only at the sub-ADF $D|_{M_0}$ that consists of D restricted to M_0 and consider all its σ-interpretations. For each σ-interpretation w, we use the information it contains (that is, the truth values it assigns) to simplify the rest of the ADF. Simplification means that we propagate the truth values of the interpretation as far as possible and at the same time remove redundant links. We then recursively invoke the definition on the ADF resulting from simplifying D by w. Note that at this point, the statements in M_0 are already dealt with, they have fixed truth values. The main task of the recursive call is to take care of all statements that have newly become independent (modulo M_0). Since the operator ind_D is $\subseteq$-monotone, the sequence $ind_D(\emptyset) \subseteq ind_D(ind_D(\emptyset)) \subseteq \ldots$ is monotonically increasing and eventually reaches the fixed-point $ind_D(S) = S$. Then the first case of the definition applies and the recursion stops. An obvious special case are ADFs D with only one strongly connected component. In this case, $ind_D(\emptyset) = S$ and thus $\sigma_2(D) = \sigma(D)$.

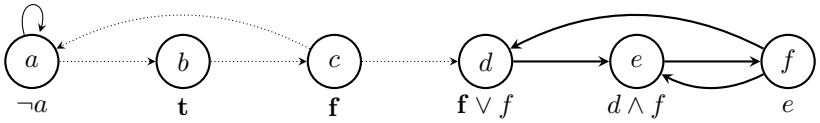
Example 3. Let the ADF $D = (S, L, C)$ be graphically given as follows:



We want to compute $nai_2(D) = nai_2(ind_D(\emptyset), D)$ and thus construct the set $ind_D(\emptyset) = \{a, b, c\} = M_0$. Then we obtain $nai(D|_{M_0}) = \{v_0, v_1, v_2\}$:

$$v_0 = \{a \mapsto \mathbf{u}, b \mapsto \mathbf{t}, c \mapsto \mathbf{f}\}, v_1 = \{a \mapsto \mathbf{f}, b \mapsto \mathbf{u}, c \mapsto \mathbf{t}\}, v_2 = \{a \mapsto \mathbf{t}, b \mapsto \mathbf{f}, c \mapsto \mathbf{u}\}.$$

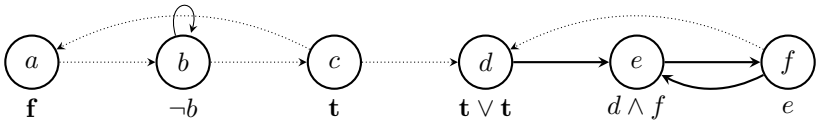
According to Definition 9, for each of these interpretations w we construct the respective reduced ADF $[\![D]\!]^w_{M_0}$ and recursively determine its nai_2 semantics. We begin with $w = v_0 \in nai(D|_{M_0})$ and compute $nai_2(M_1, D_1)$ with $D_1 = [\![D]\!]^{v_0}_{M_0}$. The ADF D_1 is graphically depicted below; links that have newly become redundant are dotted, links originating in independent statements are thin.



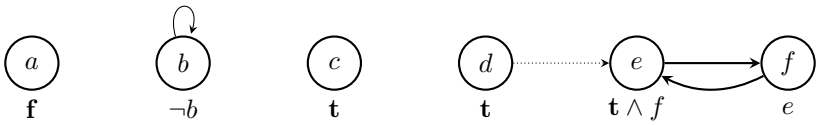
Thus $M_1 = ind_{D_1}(M_0) = S$, and we only need to consider $nai(D_1) = \{v_3, v_4\}$:

$$v_3 = v_0 \cup \{d \mapsto \mathbf{t}, e \mapsto \mathbf{t}, f \mapsto \mathbf{t}\}, \qquad v_4 = v_0 \cup \{d \mapsto \mathbf{f}, e \mapsto \mathbf{f}, f \mapsto \mathbf{f}\}.$$

We next consider $v_1 \in nai(D|_{M_0})$ and call $nai_2(M_2, D_2)$ with $D_2 = [\![D]\!]^{v_1}_{M_0}$:



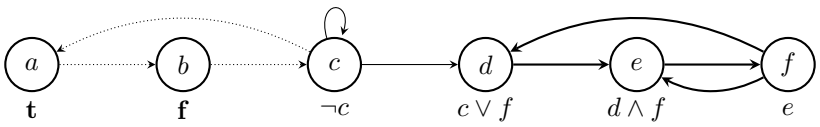
Note that $\varphi_d^{v_1} = \mathbf{t} \vee f$ where f is redundant and thus $[\![\varphi_d]\!]_{M_0}^{v_1} = \mathbf{t} \vee \mathbf{t}$. For the next step we get $M_2 = ind_{D_2}(M_0) = \{a, b, c, d\}$, and $D_2|_{M_2}$ has the single naive interpretation $v_5 = v_1 \cup \{d \mapsto \mathbf{t}\}$. We compute $nai_2(M_3, D_3)$ with $D_3 = [\![D_2]\!]_{M_2}^{v_5}$:



We obtain $M_3 = ind_{D_3}(M_2) = S$ and two naive interpretations for D_3:

$$v_6 = v_5 \cup \{e \mapsto \mathbf{t}, f \mapsto \mathbf{t}\}, \qquad v_7 = v_5 \cup \{e \mapsto \mathbf{f}, f \mapsto \mathbf{f}\}.$$

Finally, for $v_2 \in nai(D|_{M_0})$ the call $nai_2(M_4, D_4)$ is performed with $D_4 = [\![D]\!]_{M_0}^{v_2}$:



Thus $M_4 = ind_{D_4}(M_0) = S$ and the two naive interpretations of D_4 are

$$v_8 = v_2 \cup \{d \mapsto \mathbf{t}, e \mapsto \mathbf{t}, f \mapsto \mathbf{t}\}, \qquad v_9 = v_2 \cup \{d \mapsto \mathbf{t}, e \mapsto \mathbf{f}, f \mapsto \mathbf{f}\}.$$

Thus overall, we obtain the set

$$nai_2(D) = nai_2(M_1, D_1) \cup nai_2(M_2, D_2) \cup nai_2(M_4, D_4) = \{v_3, v_4, v_6, v_7, v_8, v_9\}.$$

In contrast, the naive interpretations of D contain another interpretation v_{10} with

$$v_{10} = \{d \mapsto \mathbf{t}, a \mapsto \mathbf{t}, e \mapsto \mathbf{f}, f \mapsto \mathbf{f}, b \mapsto \mathbf{u}, c \mapsto \mathbf{u}\} \in nai(D) = nai_2(D) \cup \{v_{10}\}.$$

So at least for the case of naive semantics, $nai \neq nai_2$. But what about other semantics? The following fundamental result clarifies this question. For semantics σ, τ, the expression $\sigma \leq \tau$ means that for all ADFs D we have $\sigma(D) \subseteq \tau(D)$. Due to a lack of space, we cannot present the (long and tedious) proof of our main result. We note however that the counterexamples which are needed to prove items 2 and 4 are AF-based ADFs which can be found in [6].

Theorem 1.

1.	*Let* $\sigma \in \{cfi, adm, pre, com, mod\}$.	*Then* $\sigma \leq \sigma_2$.
2.	*Let* $\sigma \in \{nai, stg\}$.	*Then* $\sigma \not\leq \sigma_2$.
3.	*Let* $\sigma \in \{cfi, nai, adm, pre, com, mod\}$.	*Then* $\sigma_2 \leq \sigma$.
4.	*Let* $\sigma \in \{stg\}$.	*Then* $\sigma_2 \not\leq \sigma$.

As an easy consequence, we get a number of semantics for which the decomposition-based scheme does not lead to new semantics, but rather new ways to compute the semantics. For the grounded semantics, the equality $grd = grd_2$ follows from the same equality for complete semantics.

Corollary 2. *For $\sigma \in \{cfi, adm, pre, com, grd, mod\}$ we find $\sigma = \sigma_2$.*

As another result, we can show that for the special case of AFs, our nai_2 semantics coincides with AFs' *cf2* semantics.

Proposition 3. *Let F be an argumentation framework and D_F its associated ADF. The cf2 labelings of F coincide with the nai_2 interpretations of D_F.*

4. Composing ADFs

While we have looked at decomposing ADFs into their strongly connected components up to here, we now ask the converse question: Is it possible to *compose* two ADFs into a single one such that the semantics of the composed ADF is readily computable from the semantics of the two constituents?

We will look at two special classes of ADFs first, ADFs based on AFs, and ADFs based on normal logic programs. While it is very natural to compose two AFs and equally natural to compose two LPs, we will show that the two composition methods do not align on the ADF level, which explains the need for two different ADF composition functions.

Composing AF-based ADFs. Recall that Brewka and Woltran [3] showed that ADFs capture AFs: For an AF $F = (A, R)$, the ADF *associated to* F is $D_F = (A, R, C)$ with $C = \{\varphi_a\}_{a \in A}$ and $\varphi_a = \bigwedge_{(b,a) \in R} \neg b$ for $a \in A$. For two AFs $F_1 = (A_1, R_1)$ and $F_2 = (A_2, R_2)$, it is natural to define their union as $F_1 \cup F_2 = (A_1 \cup A_2, R_1 \cup R_2)$. Let us look how this translates to AF-based ADFs.

Example 4. Let $F_1 = (A_1, R_1)$ with $A_1 = \{a, b\}$ and $R_1 = \{(a, b)\}$; $F_2 = (A_2, R_2)$ with $A_2 = \{b, c\}$ and $R_2 = \{(c, b)\}$. Clearly $F = F_1 \cup F_2 = (\{a, b, c\}, \{(a, b), (c, b)\}$. Now let us look at their ADF translations (or rather their acceptance formulas, where φ^1 represent the formulas for D_{F_1}, likewise φ^2 for D_{F_2} and the φ represent those for D_F). Argument a is unattacked in F_1 whence $\varphi^1_a = \mathbf{t}$, AF F_2 does not even mention a, thus in the ADF D_F corresponding to the composed AF F we have $\varphi_a = \mathbf{t}$. For argument b, we get $\varphi^1_b = \neg a$, $\varphi^2_b = \neg c$ and $\varphi_b = \neg a \wedge \neg c$. Argument c is not mentioned in F_1 and unattacked in F_2 whence $\varphi_c = \mathbf{t}$. So in general it seems that to compose two AF-based ADFs with partly overlapping signatures, we have to interpret non-existent arguments as always true, and join acceptance conditions conjunctively.

Composing LP-based ADFs. Brewka and Woltran [3] also showed how to translate logic programs to ADFs: For P a normal logic program over a set A of atoms, define an ADF $D_P = (A, L, C)$ as follows:

- $L = \{(b, a) \mid a \leftarrow M \in P, b \in M^+ \cup M^-\}$
- For $a \in A$, set $\varphi_a = \bigvee_{a \leftarrow M \in P} \left(\bigwedge_{m \in M^+} m \wedge \bigwedge_{m \in M^-} \neg m\right)$.

Since a logic program is just a set of rules, for two LPs P_1 and P_2, it is natural to define their union as $P_1 \cup P_2$.

Example 5. Let $P_1 = \{b \leftarrow a\}$ and $P_2 = \{b \leftarrow c\}$ be logic programs. Clearly their union is $P = P_1 \cup P_2 = \{b \leftarrow a, b \leftarrow c\}$. The acceptance formulas of the respective ADFs are these: For statement a, we have $\varphi_a^1 = \mathbf{f}$ since there is no rule for a in P_1. P_2 does not use a, thus in D_P we have $\varphi_a = \mathbf{f}$. Statement b is true if a is true according to P_1 and true if c is true according to P_2, thus $\varphi_b = a \vee c$. Finally, for statement c we get $\varphi_c = \mathbf{f}$ as for a. So in general it seems that to compose two LP-based ADFs with partly overlapping signatures, we have to interpret non-existent arguments as always false, and join acceptance conditions disjunctively.

Composing general ADFs. The observed duality between composing AF-based ADFs and LP-based ADFs motivates the following definition.

Definition 10. Let $D_1 = (S_1, L_1, C_1)$ and $D_2 = (S_2, L_2, C_2)$ be ADFs. Define

$$S = S_1 \cup S_2, \quad L = L_1 \cup L_2, \quad D_1 \otimes D_2 = (S, L, C^{\otimes}), \quad D_1 \oplus D_2 = (S, L, C^{\oplus})$$

$$C^{\otimes} = \{C_1 \otimes_s C_2\}_{s \in S} \text{ where } C_1 \otimes_s C_2 = \begin{cases} \varphi_s^1 \wedge \varphi_s^2 & \text{if } s \in S_1 \cap S_2 \\ \varphi_s^1 & \text{if } s \in S_1 \setminus S_2 \\ \varphi_s^2 & \text{otherwise} \end{cases}$$

$$C^{\oplus} = \{C_1 \oplus_s C_2\}_{s \in S} \text{ where } C_1 \oplus_s C_2 = \begin{cases} \varphi_s^1 \vee \varphi_s^2 & \text{if } s \in S_1 \cap S_2 \\ \varphi_s^1 & \text{if } s \in S_1 \setminus S_2 \\ \varphi_s^2 & \text{otherwise} \end{cases}$$

Clearly if $S_1 \cap S_2 = \emptyset$ then $D_1 \otimes D_2 = D_1 \oplus D_2$. With this definition, the following is easy to prove.

Proposition 4. *Let F_1, F_2 be argumentation frameworks and P_1, P_2 be normal logic programs. Then (1) $D_{F_1 \cup F_2} = D_{F_1} \otimes D_{F_2}$ and (2) $D_{P_1 \cup P_2} = D_{P_1} \oplus D_{P_2}$.*

These results nicely illustrate the different granularity of AFs and LPs: From previous work, we know that – quite independently of specific semantics – AFs can be seen as LPs of a special form [8]. (Roughly, the ADF associated to an AF is easily written as a logic program, where for each argument there is one rule with all attackers as negative body literals.) In an AF-based LP, adding an argument to the AF results in adding a rule to the LP. Adding an attack between existing arguments to the AF, on the other hand, results in adding a negative body literal to an existing rule in the AF-based LP. This modification of an existing rule cannot be easily expressed (i.e. via set union) at the granularity level of LPs.

5. Discussion

We introduced and studied a scheme to decompose abstract dialectical frameworks along their strongly connected components. For several semantics, our scheme leads to a new way to compute interpretations, among them admissible, complete, preferred, grounded and model semantics. For others, our scheme leads to new semantics which arguably remedy some of the original semantics' shortcomings, such as naive and stage semantics.

Due to the generality of ADFs, this paper – as a byproduct – defines the nai_2 and stg_2 semantics also for logic programs. That is, when a normal logic program fails to have models due to odd-length negative cycles, our decomposition-based scheme can straightforwardly be applied to the logic program's associated ADF to compute nai_2 and stg_2 interpretations. Computationally, this is quite economic since the increase in size from logic program to ADF is at most linear.

For future work, we plan to consider further semantics. For example, in this paper we have not considered the ADF stable model semantics for clarity, as it also uses notions of reduct and partial evaluation that are subtly different from the ones employed in this paper. Naturally, besides analyzing the complexity of our decomposition-based scheme, we also want to implement it to verify whether there is a performance gain in comparison to conventional evaluation methods. Furthermore, this work has laid the groundwork for considering (arbitrary) splittings of ADFs, thus also paving the way for studying their strong equivalence.

Acknowledgements. This research has been partially supported by DFG under project BR-1817/7-1.

References

[1] P. Baroni, M. Giacomin, and G. Guida. SCC-recursiveness: A general schema for argumentation semantics. *Artificial Intelligence*, 168(1–2):162–210, 2005.

[2] P. Baroni, M. Caminada, and M. Giacomin. An introduction to argumentation semantics. *Knowledge Eng. Review*, 26(4):365–410, 2011.

[3] G. Brewka and S. Woltran. Abstract Dialectical Frameworks. In *Proc. KR 2010*, pages 102–111, 2010.

[4] G. Brewka, S. Ellmauthaler, H. Strass, J. P. Wallner, and S. Woltran. Abstract Dialectical Frameworks Revisited. In *Proc. IJCAI 2013*, pages 803–809. AAAI Press, August 2013.

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

[6] W. Dvořák and S. A. Gaggl. Stage semantics and the SCC-recursive schema for argumentation semantics. *Journal of Logic and Computation*, 2014.

[7] S. A. Gaggl and S. Woltran. The cf2 argumentation semantics revisited. *Journal of Logic and Computation*, 23(5):925–949, 2013.

[8] H. Strass. Approximating operators and semantics for abstract dialectical frameworks. *Artificial Intelligence*, 205:39–70, December 2013.

[9] H. Strass and J. P. Wallner. Analyzing the Computational Complexity of Abstract Dialectical Frameworks via Approximation Fixpoint Theory. In *Proc. KR 2014*, pages 101–110, Vienna, Austria, July 2014.

[10] B. Verheij. Two approaches to dialectical argumentation: admissible sets and argumentation stages. In J.-J. Ch. Meyer and L.C. van der Gaag, editors, *Proc. NAIC 1996*, pages 357–368, 1996.