Structural Constraints for Dynamic Operators in Abstract Argumentation

Johannes P. Wallner
TU Wien, Austria

Abstract. Many recent studies of dynamics of formal argumentation in AI focus on the well-known formalism of argumentation frameworks (AFs). Despite their usefulness in many areas of argumentation, their abstract notion of arguments creates a barrier for operators that modify a given AF, namely in the case that dependencies between arguments have been abstracted away that might be subsequently missed. In this paper we aim to support development of dynamic operators on formal models in abstract argumentation by providing constraints imposed on the modification of the structure that can be used to incorporate information that has been abstracted away. Towards a broad reach, we base our results on the general formalism of abstract dialectical frameworks (ADFs) in abstract argumentation, and study the complexity of the proposed structural constraints. To show applicability, we adapt an extension enforcement operator on AFs to ADFs that is allowed to only add support relations between arguments. We show feasibility of our approach by an experimental evaluation of an implementation of this operator.

1. Introduction

In the last decades there has been a steady stream of advances in formal approaches to argumentation in artificial intelligence (AI) [1]. Central to many formal models in argumentation are argumentation frameworks (AFs) [13], a formalism that is both foundational and simple: AFs consist of abstract arguments and directed attacks (conflicts) between these arguments. The simplicity of AFs is an appealing quality, which can be seen by the fact that several approaches use AFs as their core reasoning engine [5,19].

Since AFs are, in a sense, "easy to handle" from a conceptual point of view, they have also been prominent as a formal basis in studies of dynamics of argumentation [3, 7,8,10]. This topic naturally arose in this research community since argumentation is an inherently dynamic process. However, in contrast to forms of "static" reasoning, i.e., reasoning on AFs that do not change, dynamic operations, if solely done on AFs, may miss certain dependencies between arguments. For instance, there can be dependencies in the internal structures of arguments, which are hidden in the arguments in AFs whose structure was abstracted away during their instantiation.

Example 1. Consider two arguments, a_1 and a_2, and assume that a_1 is a sub argument of a_2, e.g., because a_1 concludes a premise of a_2. When both arguments are part of an AF then a modification of that AF may add an attack on a_1, say by a counter to a_1's premise. In many cases such a counter is deemed to, also, attack the super argument a_2, since its premise is attacked. However, if no further information than the AF is available, then this

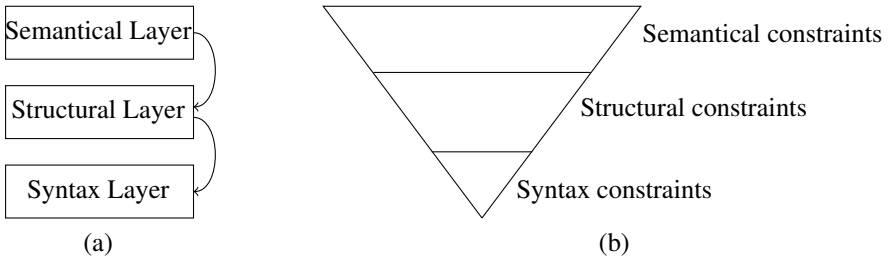

Figure 1. (a) Three layers for dynamic operators and (b) outcomes that satisfy constraints of each layer.

interdependency might be missed, which may lead to unintended results, like rejecting a_1 but finding a_2 to be acceptable, even though one is the premise of the other.

In this paper we argue that despite these drawbacks of AFs when utilizing them in dynamic operations, abstract formalisms, with AFs being the most famous ones, are still a very useful notion for studying dynamics of argumentation. We argue that existing dynamic operators, and new ones, can be augmented to incorporate vital structural constraints. Our goal in this paper is to provide means with which dynamic operations on abstract arguments can be extended so that (i) structural constraints are taken into account and (ii) operators can still work on a convenient and (partial) abstract level.

Towards our goal, we view dynamic operations on abstract arguments (and their relations) as operators working in three layers: a semantic layer, a structural layer, and a syntax layer. In the first layer, a semantical change is specified by an operator. As a concrete example, consider extension enforcement as defined in [3]: given an AF and a set of arguments, the output is an AF whose semantics contains the given set of arguments as an extension. That is, the operator defines certain conditions on the semantics of the output (or, equivalently, constraints on the semantics of an output AF). On the structural side, extension enforcement specifies that the output AF shall be close to the input AF (via a defined notion of distance between input and output AF structures). Finally, syntactically, the operator requires that the output is, again, an AF.

More broadly, we find that the semantical layer of a dynamic operator defines semantical constraints on the output, the structural layer specifies constraints on the structure of the output, and the syntax layer specifies the concrete syntax an output shall have. We illustrate this "workflow" in Figure 1(a). Analogously, one can view each layer of an operator as (a set of) constraints, each constraining the possible candidate solutions, see Figure 1(b). In the case of the extension enforcement operator, the first layer constrains the set of solution candidate AFs to be those that satisfy the semantical constraint, then constrains to choose, among those satisfying the first layer, those structures that are close to the input, and, finally, choose a particular AF.

With this paper, we contribute to the second layer, the structural layer, by giving several properties that can be required by an output structurally, with the aim of giving developers of dynamic operators an additional handle to extend operators. Since dynamic operators are themselves, even without further constraints, often computationally hard to solve [8,25], we study the complexity of the constraints we consider in this paper.

As the formal foundation, we use abstract dialectical frameworks (ADFs) [6] as the formal model that undergoes a change in this paper. Although ADFs are more complex than AFs, in the sense that arguments' acceptance conditions can be specified in a liberal manner, they are still an abstract formalism with abstract arguments, and, importantly,

have been shown to capture several abstract formalisms in argumentation in AI [21], and, thus, extend the reach of our contributions to also other formalisms than AFs. Last, but not least, using ADFs has also the important by-product that it allows to generalize existing dynamic operators on AFs to the more general setting of ADFs. Indeed, many dynamic approaches are currently restricted to AFs only. By using ADFs, and applying suitable structural constraints, one can lift current operators on AFs to ADFs, while keeping the intuitions of the operators. Our main contributions are as follows.

- To make our proposal concrete, we adapt two dynamic operators for AFs to ADFs: non-strict extension enforcement [3,2,8] and revision [10].
- We give a list of structural constraints on ADFs, i.e., constraints on the structure of ADFs, exemplify their use for dynamics, and study some of their properties.
- To demonstrate feasibility, we show the complexity of checking whether an ADF satisfies the constraints, which is decidable in polynomial time in many cases.
- We show a case study on how the given structural constraints can be utilized, by adapting enforcement to allow only to add support to arguments.
- We implemented a prototype of this new operation of enforcement with supports based on the goDiamond system [23], and performed an experimental evaluation.

Our prototype, instances, and proof details, are available at `dbai.tuwien.ac.at/proj/embarg/supp-enf`. We recall background in Section 2, present the constraints in Sect. 3, case study in Sect. 4, and our complexity results and our experiments in Sect. 5.

2. Background

Argumentation Frameworks We recall basics of argumentation frameworks (AFs) [13].

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

Semantics for argumentation frameworks are defined through a function σ which assigns to each AF $F = (A,R)$ a set $\sigma(F) \subseteq 2^A$ of extensions. We consider for σ the functions *adm*, *com*, *grd*, and *prf*, which stand for admissible, complete, grounded, and preferred, respectively. Towards the definition we make use of the characteristic function of AFs, defined for an AF $F = (A,R)$, by $\mathscr{F}_F(S) = \{x \in A \mid x \text{ is defended by } S\}$.

Definition 2. Let $F = (A,R)$ be an AF. An $S \subseteq A$ is conflict-free (in F), if there are no $a,b \in S$, s.t. $(a,b) \in R$. We denote conflict-free sets by $cf(F)$. For an $S \in cf(F)$, it holds that $S \in adm(F)$ iff $S \subseteq \mathscr{F}_F(S)$; $S \in com(F)$ iff $S = \mathscr{F}_F(S)$; $S \in grd(F)$ iff S is the least fixed-point of $\mathscr{F}_F$; and $S \in prf(F)$ iff $S \in adm(F)$ and there is no $T \in adm(F)$ with $S \subset T$.

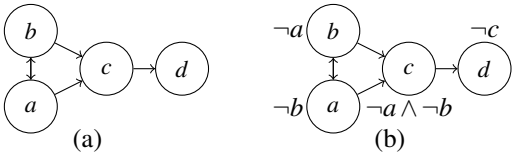

Figure 2. Illustrations of AF and corresponding ADF from Example 2 and Example 3.

Example 2. Let $F = (\{a,b,c,d\},R)$ be an AF with $R = \{(a,b), (b,a), (a,c), (b,c), (c,d)\}$ (Figure 2(a)). We have $adm(F) = \{\emptyset, \{a\}, \{b\}, \{a,d\}, \{b,d\}\}$, and $grd(F) = \{\emptyset\}$.

Abstract Dialectical Frameworks We recall the syntax and semantics of ADFs from [6]. Let A be a finite set of arguments A. An interpretation is a function I mapping arguments to one of the three truth values $I : A \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$. That is, an interpretation maps each argument to either true ($\mathbf{t}$), false ($\mathbf{f}$), or undefined ($\mathbf{u}$). An interpretation I is *two-valued* if $I(a) \in \{\mathbf{t}, \mathbf{f}\}$ for all $a \in A$, and *trivial*, denoted as $I_\mathbf{u}$, if $I(a) = \mathbf{u}$ for all $a \in A$. For a two-valued interpretation I, $I(\varphi)$ extends to the evaluation of a Boolean formula φ under I as usual. An interpretation I is equally or more informative than J, denoted by $J \leq_i I$, if $J(a) \in \{\mathbf{t}, \mathbf{f}\}$ implies $J(a) = I(a)$ for all $a \in A$. We denote by $<_i$ the strict version of $\leq_i$, i.e., $J <_i I$ if $J \leq_i I$ and $\exists a \in A$ s.t. $J(a) = \mathbf{u}$ and $I(a) \in \{\mathbf{t}, \mathbf{f}\}$.

An ADF is a tuple $D = (A, L, C)$ where A is a set of arguments, $L \subseteq A \times A$ is a set of links, and $C = \{\varphi_a\}_{a \in A}$ is a collection of acceptance conditions, each given by a formula over the parents of an argument: $par_D(a) = \{b \in A \mid (b,a) \in L\}$.

Example 3. Figure 2 shows an ADF $D = (\{a,b,c,d\},L,C)$ with $L = \{(a,b), (b,a), (a,c), (b,c), (c,d)\}$. The acceptance conditions are shown close to the arguments.

The semantics of ADFs are based on the characteristic function Γ_D mapping interpretations to updated interpretations and defined for an ADF D as $\Gamma_D(I) = J$ with $J(a) = \mathbf{t}$ if $\varphi_a[I]$ is a tautology, $J(a) = \mathbf{f}$ if $\varphi_a[I]$ is unsatisfiable, and $J(a) = \mathbf{u}$ otherwise, where $\varphi[I]$ is the formula obtained from φ with each argument that I assigns to either true or false being replaced by the corresponding truth constant, i.e., $\varphi[I] = \varphi[x \mapsto \top \mid I(x) = \mathbf{t}][x \mapsto \bot \mid I(x) = \mathbf{f}]$; arguments assigned to undefined are not modified.

Definition 3. Given an ADF D, an interpretation I is admissible in D iff $I \leq_i \Gamma_D(I)$; is complete in D iff $I = \Gamma_D(I)$; is grounded in D iff I is the $\mathrm{lfp}(\Gamma_D)$; and is preferred in D iff I is $\leq_i$-max. admissible in D.

We refer to the set of all admissible, grounded, and preferred interpretations of an ADF D as $adm(D)$, $grd(D)$, and $prf(D)$, respectively.

We denote the update of an interpretation I with truth value $\mathbf{x} \in \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ for argument b by $I|^b_\mathbf{x}$, i.e., $I|^b_\mathbf{x}(b) = \mathbf{x}$ and $I|^b_\mathbf{x}(a) = I(a)$ for $a \neq b$. In an ADF $D = (A, L, C)$, a link $(b,a) \in L$ is *supporting* (in D) if for every two-valued interpretation I, $I(\varphi_a) = \mathbf{t}$ implies $I|^b_\mathbf{t}(\varphi_a) = \mathbf{t}$; *attacking* (in D) if for every two-valued interpretation I, $I(\varphi_a) = \mathbf{f}$ implies $I|^b_\mathbf{t}(\varphi_a) = \mathbf{f}$. An ADF D is called *bipolar* if each link $(b,a) \in L$ is attacking or supporting.

Example 4. For the ADF from Example 3, we have $prf(D) = \{\{a \mapsto \mathbf{t}, b \mapsto \mathbf{f}, c \mapsto \mathbf{f}, d \mapsto \mathbf{f}\}, \{a \mapsto \mathbf{f}, b \mapsto \mathbf{t}, c \mapsto \mathbf{f}, d \mapsto \mathbf{f}\}\}$ and $grd(D) = \{\{a \mapsto \mathbf{u}, b \mapsto \mathbf{u}, c \mapsto \mathbf{u}, d \mapsto \mathbf{u}\}\}$.

An AF $F = (A,R)$ can be translated to an ADF by defining $D_F = (A, R, C)$ with $\varphi_a = \bigwedge_{(b,a) \in R} \neg b$ (see also Figure 2). The semantics of an AF and its corresponding ADF coincide (by relating an extension E with interpretation I via $E = \{a \mid I(a) = \mathbf{t}\}$ [6]).

Enforcement Intuitively, enforcement on AFs [3,2,9,8,25] deals with the question how to modify an AF s.t. the semantics of the modified AF satisfies certain constraints, and, typically, also the number of modifications has to be minimum.

We consider here the enforcement variant called non-strict enforcement under strong expansions with a bounded number of expanded arguments [8]. That is, formally, we

are given an AF $F = (A,R)$, a semantics σ, expanded arguments A', and a subset of the arguments $E \subseteq A$ as input. Now, an AF $F' = (A \cup A', R')$ non-strictly enforces E under σ if $R \subseteq R'$, for any $(a,b) \in R' \setminus R$ we have $a \in A'$ and $b \in A$, and $\exists E' \in \sigma(F')$ with $E \subseteq E'$. Thus, one needs to find a modified AF F' that has the new arguments A', newly added attacks coming from these arguments in A' onto arguments in A, and that E' is part of a σ-extension of F'. Among all candidates, typically, one further restricts solution AFs to be optimal w.r.t. the modifications to the attacks. This is specified by finding an optimal AF $F^* = (A \cup A', R^*)$ that non-strictly enforces E under σ and there is no $F' = (A \cup A', R')$ that non-strictly enforces E under σ and $|(R \Delta R')| < |(R \Delta R^*)|$.

Example 5. Consider the AF from Example 2. Say we want to enforce $\{a,d\}$ to part of the grounded extension in an expanded AF that can add argument e (i.e. $A' = \{e\}$). This can be achieved by adding attack (e,b), resulting in $F^* = (A^*, R^*)$ with $A^* = \{a,b,c,d,e\}$ and $R^* = \{(a,b), (b,a), (a,c), (b,c), (c,d), (e,b)\}$. We have $grd(F^*) = \{e,a,d\}$, implying that $\{a,d\}$ is non-strictly enforced under grounded semantics.

Revision We recall the revision operator $\circ_F^{\text{can}}$ [10, Definition 6] that revises a given AF $F = (A,R)$ by a given propositional formula ϕ under the preferred semantics. In [10] more sophisticated operators are defined, as well, but, for illustrative reasons, we give the (simple) canonical operator as an example. The idea behind the operator is to revise the given AF s.t. the extensions of the revised AF are a subset of the models of the formula, and satisfy certain postulates of revision. This operator is based on a ranking of extensions (models). The intuition of the ranking is to prefer smaller extensions (w.r.t. set cardinality). Ties (extension of the same size) are broken by a lexicographic ordering on extensions based on an ordering over the arguments.

Towards the exact definition, assume an ordering over the set of arguments, i.e., $(a_1, \ldots, a_n)$, $a_i \in A$. Given a set of extensions $\mathscr{S}$, define the ordering on (non-)extensions $E_1, E_2 \in \mathscr{S}$: $E_1 \leq^{\mathscr{S}} E_2$ iff $|E_1| < |E_2|$ or $|E_1| = |E_2| \wedge \vec{E}_1^{\#} \leq_{\text{lex}} \vec{E}_2^{\#}$ where $\vec{E}_1^{\#}$ is the ordered tuple $(x_1, \ldots, x_{|E_1|-1})$ obtained from E_1 by ordering the arguments in E_1 according the argument ordering, but excluding the argument ordered least. The comparison by $\leq_{\text{lex}}$ is then a lexicographic ordering of the tuples in the standard way (i.e. $(a,b) \leq_{\text{lex}} (a,c)$ if a is ordered lower than b and b is ordered lower than c). Based on this ordering, the canonical preorder $\leq_F^{\text{can}}$ is defined by $E_1 \leq_F^{\text{can}} E_2$ iff $E_1 \in \sigma(F)$ or $E_1, E_2 \notin \sigma(F) \wedge E_1 \leq^{2^A \setminus \sigma(F)} E_2$. Finally, given an AF $F = (A,R)$, and a propositional formula ϕ, define $[F \circ_F^{\text{can}} \phi] = min_{\leq_F^{\text{can}}}\{E \in \text{Mod}(\phi)\}$. That is, $[F \circ_F^{\text{can}} \phi]$ defines the extensions of the revised AF. In other words, an outcome would be an AF F' s.t. $\sigma(F') = [F \circ_F^{\text{can}} \phi]$.

Example 6. Let $F = (A,R)$ be an AF with $A = \{a,b,c\}$ and $R = \{(a,b),(b,a)\}$. This yields $prf(F) = \{\{a,c\},\{b,c\}\}$. Say, we revise by $\phi = a \wedge c$, i.e. $\text{Mod}(\phi) = \{\{a,c\},\{a,b,c\}\}$. The ordering $\leq^X$ orders four different levels as follows: $\emptyset \leq^X \{a\},\{b\},\{c\} \leq^X \{a,b\} \leq^X \{a,b,c\}$ for $X = 2^A \setminus prf(F)$. We get $\{a,b\},\{b,c\} \leq_F^{\text{can}} \emptyset \leq_F^{\text{can}} \{a\},\{b\},\{c\} \leq_F^{\text{can}} \{a,c\} \leq_F^{\text{can}} \{a,b,c\}$. Finally, the (in this case unique) model of ϕ ordered least, w.r.t. $\leq^{prf(F)}$ is $\{a,c\}$. Thus, $[F \circ_F^{\text{can}} \phi] = \{\{a,c\}\}$.

3. Structural Constraints for Dynamic Operators

This section introduces several constraints that can be used to extend current dynamic operators, and support development of new operators.

From a principled point of view, we consider dynamic operators that take an ADF D as input, with possibly further information, and produce a modified ADF D^* as output. These operators then define certain constrains on on the output, namely on (i) the semantics, (ii) the structure, and (iii) the concrete syntax. We exemplify the semantical constraints on the two operators from Section 2 when lifted from AFs to ADFs.

Example 7. Let $D = (A, L, C)$ be an ADF, σ a semantics, and $S \subseteq A$ a set of arguments. We now present a straightforward generalization of non-strict extension enforcement for AFs to the case of ADFs. That is, we want to enforce that S is part of a σ-interpretation by modifying D. An output ADF D^* then satisfies the constraint imposed by non-strict extension enforcement if $\exists I' \in \sigma(D^*)$ s.t. $I \leq_i I'$ with $I = \{s \mapsto \mathbf{t} \mid s \in S\} \cup \{x \mapsto \mathbf{u} \mid x \in A \setminus S\}$. Intuitively, there has to be a σ-interpretation I' in the modified D^* s.t. all arguments in S are assigned true (the status of other arguments is not constrained).

For the revision operator, the semantical constraint is such that the output AF shall have as its semantics a specified set of extension $\Sigma = \{E_1, \ldots, E_n\}$. Translated to ADFs, we have $\Sigma = \{I^{\mathbf{t}} \mid I \in \sigma(D^*)$ with $I^{\mathbf{t}} = \{x \mid I(x) = \mathbf{t}\}$.

The structural constraints we introduce refine the possible ADFs (those that satisfy the semantical constraints) further. Formally, a constraint c is specified in such a way that one can decide whether an ADF satisfies the constraint.

Boolean combinations of constraints In the following, when $K = \{c_1, \ldots, c_n\}$ is a set of (structural) constraints, we consider also the a constraint that is a Boolean combination of these constraints. That is, a Boolean formula Φ that has as its variables constraints in K. Satisfaction, for an ADF D, of Φ is then defined in a standard way: if D satisfies a $c_i \in K$, then $\Phi = c_i$ is satisfied by D; if $\Phi = \neg c_i$, then D satisfies Φ if it does not satisfy c_i; the connectives of conjunction and disjunction are defined in the standard way, as well.

General constraints We start with basic constraints, namely ones that specify limits of arguments and links. That is, for a given ADF $D^* = (A^*, L^*, C^*)$ (a potential output ADF) and A, A', L, and L', we define the following constraints, with $L|_{A^*} = L \cap (A^* \times A^*)$.

(G1) $A \subseteq A^* \subseteq A'$ (G2) $L|_{A^*} \subseteq L^* \subseteq L'|_{A^*}$

These constraints specify which arguments may be in the output (G1) and which links may be present (G2). For instance, via (G1), one can specify for enforcement under expansions how many arguments the expansion may add to the original framework.

Argument constraints The next basic constraint gives a concrete handle which arguments are present in the output. For an ADF $D^* = (A^*, L^*, C^*)$ and argument a we define:

(A1) $a \in A^*$

Example 8. Assume an enforcement operator under strong expansions and, say, that we can expand by arguments a_1, a_2, and a_3. Suppose further that by adding a_1 to the original framework we can enforce the given interpretation (e.g. a_1 is the only argument permitted to have a link onto an argument c that we want to be true in an admissible interpretation and adding a support from a_1 onto c gives a solution). However, consider now the case that a_1 is a super argument of a_2, which, in turn, is a super argument of a_3. Simple addition of a_1 might be reasonable, in certain cases, but it is likewise adequate to require that all sub arguments have to be present, as well. This can be specified by

stating that $((a_1 \in A^*) \rightarrow (a_2 \in A^*)) \wedge ((a_2 \in A^*) \rightarrow (a_3 \in A^*))$. These specify simple implications: if a_1 is present, so must be a_2 (and in turn a_3).

Constraints on links Next, we consider constraint on links. In the ADF D^* L^+ are supporting links and L^- are attacking links. For a link l define:

(L1) $l \in L^*$ (L2) $l \in L^+$
(L3) $l \in L^-$ (L4) (A^*, L^*) forms a directed acyclic graph

Example 9. Constraining the type of links can directly imply that the output ADF actually belongs to a proper sub family of ADFs: if we require each link to be either attacking or supporting (via $(l \in L) \rightarrow ((l \in L^+) \vee (l \in L^-))$), the output ADF is a BADF. Similarly, if each link is constrained to be attacking, then the output ADF, in fact, belongs to the family of frameworks defined in [20], which we call SETAFs here. These SETAFs are similar to AFs, but sets of arguments attack an argument.

Proposition 1. If an ADF $D^* = (A^*, L^*, C^*)$ satisfies constraint $K = \bigwedge_{l \in L^*}((l \in L^-) \wedge \neg(l \in L^+))$, then each acceptance condition $\varphi_a \in C^*$ can be expressed as a Boolean formula in conjunctive normal form (CNF) with only negative literals.

Proof. If $I(\varphi_a)$ is unsat. with only attacking parents, then switching the truth value of any parent to true does not change the outcome. Now, take each interpretation I s.t. (i) $I(\varphi_a)$ is unsat. and (ii) there is no I' with $I'(\varphi_a)$ unsat. and $\{x \mid I'(x) = \mathbf{t}\} \subseteq \{x \mid I(x) = \mathbf{t}\}$. Let $\mathscr{I}$ all such interpretations. Define $\varphi'_a = \bigwedge_{I \in \mathscr{I}}(\bigvee_{I(x)=\mathbf{t}} \neg x)$. It follows that $\varphi_a \equiv \varphi'_a$. □

From the previous result it can be inferred, via [18, Section 3.3], that if each acceptance condition is a CNF with negative literals, the ADF in question can be written as a SETAF.

Example 10. Constraining that the output D^* belongs to a proper sub family of ADFs also may lead to the case that there is no ADF (of that sub family) that satisfies the given structural constraints and semantical constraints. Consider constraining the output to have three arguments $\{a, b\}$, and the output shall have as its admissible semantics $com(D^*)$ the correspondence $\Sigma = \{I^{\mathbf{t}} \mid I \in com(D^*)$ with $\Sigma = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$ (similarly as for the revision operator recalled above). Requiring that each link in the output is attacking leads to non-existence of solution ADFs. To see that, consider any (set-)attack. Such an attack cannot exist, since $\{a \mapsto \mathbf{t}, b \mapsto \mathbf{t}\}$ is complete. Thus, $\varphi_a \equiv \varphi_b \equiv \top$, which is a contradiction that there is an interpretation corresponding to $\{a\}$. To have Σ as the semantical result under complete semantics on AFs, one needs more arguments (this follows from [4,14]). On general ADFs, such a correspondence is possible, however, only with use of supports.

Constraints on acceptance conditions We proceed to constraints on acceptance conditions. Given an ADF $D^* = (A^*, L^*, C^*)$, a $\varphi_a \in C^*$, $v \in \{\mathbf{t}, \mathbf{f}\}$, a three-valued interpretation I, a two-valued interpretation I', and a formula ψ, we define the following constraints.

(Ac1) $I'(\varphi_s) = v$ (Ac2) $I(\varphi_s)$ satisfiable (refutable, taut., unsat.)
(Ac3) $\varphi_s = \psi$ (Ac4) $\varphi_s \equiv \psi$

Example 11. Constraints of type (Ac2) can be used, e.g., to require the output ADF to belong to the subclass of AFs. Let $I_{\mathbf{f}}$ be an interpretation with all arguments assigned to

false. Now, one can require that $(b \in A^* \rightarrow I_{\mathbf{f}}(\varphi_b)$ tautological$) \wedge ((a,b) \in L \rightarrow I_{\mathbf{u}}|^a_{\mathbf{t}}(\varphi_b)$ unsatisfiable$)$, or, in other words, each argument is acceptable if all parents are false, and not acceptable if one parent is true. If two arguments' structures are logically inconsistent, and this is to be expressed directly in acceptance condition, the preceding attack-like constraint can be used as well.

Similarly, notions of support can also be further constraints. For instance, one can specify that certain supporting links are a kind of necessary support: $((a,b) \in L^+) \rightarrow (I|^a_{\mathbf{f}}\varphi_b$ unsatisfiable$)$. Intuitively, this specifies that if a parent is false (rejected) then the child argument cannot be acceptable.

Another type of constraint that can be expressed is to require no change between certain argument's dependencies. For instance, say an argument a has several parents and among them is b and that we want to state that in the output ADF all dependencies from parents, except b, are unchanged if b is false. This can be written as $I_{\mathbf{u}}|^b_{\mathbf{f}}(\varphi_s) \equiv \varphi_s$ (Ac4).

Constraints on characteristic functions Next we introduce a different type of constraint on the characteristic functions of ADFs Γ_{D^*}. Let $v, v' \in \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$, and s and s' arguments.

(Char) $\forall I.\Gamma_{D^*}(I)(s) = v$ implies $\Gamma_{D^*}(I)(s') = v'$

Example 12. A use case for this constraint is to state that two arguments cannot be both acceptable at the same time, even if there is no link between them. Consider $\forall I.\Gamma_{D^*}(I)(s) = \mathbf{t}$ implies $\Gamma_{D^*}(I)(s') = \mathbf{f}$, which implies that if s accepted, in a scenario encoded by an interpretation I, then s' cannot be accepted. Similarly, one may encode positive relations. Note that it is not required that s is a parent of s' or vice versa.

Weights and optimization Finally, we consider constraints that embody optimizations. Towards optimization constraints, define the cost of an ADF D via, $cost(D)$. A standard model of costs of objects is to define a set of weighted constraints that, if violated, contribute their weight to the total cost. For this paper, we let the cost function be abstract.

(O1) $cost(D^*) \leq k$ (O2) $cost(D^*)$ minimum over all ADFs

Example 13. A straightforward way to define weights, and costs, is exemplified by the extension enforcement operation, where a modified attack contributes unit weight to the overall cost. An adaption to ADFs would be then to consider for an input ADF $D = (A, L, C)$ and output ADF $D^* = (A^*, L^*, C^*)$ the cost $|L \Delta L^*|$, i.e., the cost being the cardinality of the symmetric difference between the input and output links.

4. Case study: an enforcement operator for ADFs based on supports

In this section we use the constraints to adapt the existing non-strict extension enforcement operator on ADFs to allow for only supporting links to be added. In this way, via this case study, we aim to show that our approach can be used to both add new features to existing operators and to lift them to more general settings.

We adapt non-strict enforcement under strong expansions of ADFs (as defined in the previous section): given an ADF $D = (A, L, C)$, a semantics σ, an interpretation I, and arguments to expand A', it holds that $D^* = (A^*, L^*, C^*)$ is a solution if two conditions hold. First (A^*, L^*) is to be a strong expansion of (A, L). The second condition is that

$\exists I' \in \sigma(D^*)$ s.t. $I \leq_i I'$. The cost of a solution D^* is $|L \Delta L^*|$. Further, a solution D^* is optimal if there is no other solution with less cost. Finally, new arguments a' have $\varphi_{a'} \equiv \top$

We utilize the structural constraints to specify certain permissible changes. (In the above definition we used basic constraints, constraints on links, and optimization.)

1. acceptance conditions unchanged if new arguments rejected: $I_{\mathbf{u}}|_{\mathbf{f}}^{A'} \equiv \phi_a \; \forall a \in A$
2. new links supporting: $((a',b) \in L^*) \rightarrow (a',b) \in L^+$ for each $b \in A$ and $a' \in A'$.

Example 14. Consider an ADF $D = (\{a,b,c\},L,C)$ with $\varphi_a = b$, $\varphi_b = c$, and $\varphi_c = \bot$. Further, assume that we may expand with an argument d and that we want to enforce $\{a\}$ to be true in an admissible interpretation. One way to enforce this goal is to add a supporting link (d,c) via changing φ_c to $\varphi_c' = d$ (simulating that d now supports acceptance of c). Then, $\{a \mapsto \mathbf{t}, b \mapsto \mathbf{t}, c \mapsto \mathbf{t}, d \mapsto \mathbf{t}\}$ is admissible, if we choose d's acceptance condition accordingly (e.g. $\varphi_d = \top$).

Support enforcement can lead to rejection of arguments. Assume two arguments, a and b with $\varphi_a = \neg b$ and $\varphi_b = \bot$. Enforcing a to be false in an admissible interpretation can be achieved via supporting b (e.g. via modification $\varphi_b = c$ for a supporter c).

Support enforcement, however, is not always possible: assume that we want to enforce that a is false in an admissible interpretation. If $\varphi_a \equiv \top$, then supporting enforcement fails (support cannot make a modification to a, or any other part of an ADF, that leads to a being false in an admissible interpretation).

5. Computational Aspects and Evaluation

Complexity In this section we investigate the feasibility of the constraints. More concretely, we first show the complexity of the task of verifying whether a given ADF satisfies certain constraints. This insight is important for developing dynamic operators that use such constraints. For instance, if we now that a certain constraint c can be checked in polynomial time, then a dynamic operator that constructs a candidate solution ADF can verify the constraint in polynomial time. If this algorithm is non-deterministic, then the complexity of the overall algorithm is likely to stay the same even if constraint c has to be considered (since a non-deterministic algorithm that runs in polynomial time can check a polynomial number of constraints that can be checked in polynomial time for a non-deterministically guessed object). There is evidence that dynamic operators in abstract argumentation are likely to be *NP* hard, e.g. extension enforcement on AFs is in many cases *NP* hard [25]. That is, for these kind of operators augmentation with any of the polynomial-time decidable constraints comes at not additional cost (complexity-wise). We begin with basic constraints where polynomial-time decidability is immediate.

Proposition 2. Verifying whether a given ADF satisfies constraints of type (G1), (G1), (A1), (L1), (L4), (Ac1), or (Ac3) is decidable in polynomial time.

We proceed to more sophisticated constraints. We distinguish by (Ac2) those cases where we ask satisfiability (refut.) and by (Ac2)' the cases when we ask for tautology (unsat.). (Proofs can be found at the URL from the introduction; several rely on [24].)

Proposition 3. Verifying whether a given ADF satisfies a constraint of type (Ac2) is *NP*-complete, and a constraint of type (L2), (L3), (Ac2)', (Ac4), or (Char) is *coNP*-complete.

While some of these useful constraints have relatively high complexity, if we restrict an output ADF D^* to be a bipolar ADF with known link types, we can infer the following.

Proposition 4. Verifying whether a BADF satisfies a constraint of type (L2), (L3), or (Ac2) is decidable in P, and a constraint of type (Ac4) is *coNP*-complete.

If one imposes the condition that each argument has at most k parents, for a given and fixed k (i.e. k is a constant), we can infer that all constraints can be checked in polynomial time, except for the optimization constraints.

Proposition 5. Let k be a constant integer. Satisfaction of any constraint defined in Section 3, except (O1) and (O2), for a given ADF, where each argument has at most k parents, is decidable in polynomial time.

We now show that the novel enforcement operator defined in Section 4 has the same complexity as the non-strict extension enforcement operator on AFs, under certain restrictions. We investigate the decision problem of the new enforcement operator, where we ask whether there exists a solution ADF s.t. its cost is at most a given integer. As a further condition we require the number of parents and A' to be bounded a constant.

Proposition 6. Support enforcement under admissible semantics on ADFs is *NP*-complete, if both the number of parents of each argument and the set of expanded arguments are bounded by a constant.

We note that, due to Proposition 5, any constraint defined in Section 3, except for further optimization statements, can be added to the support enforcement operator, with the mentioned restrictions, without increasing its complexity. For instance, adding constraints that whenever a support to an argument is added, all the super arguments of that argument have to be supported, as well, can be done without increased complexity.

Implementation and Experimentation We have implemented a prototype for the support enforcement operator on ADFs, and give some details on it and a first empirical assessment on the practical feasibility of this operator. All experiments were run on a machine with two AMD Opteron Processor 6308, 12 x 16GB RAM, and Debian 8. For our prototype implementation, we consider the case that the expanded arguments are a singleton set, i.e., $A' = \{x\}$ is a distinguished argument.

We adapted the answer-set programming (ASP) based goDiamond [23,15] 0.6.6 to implement support enforcement. Due to space constraints, we only highlight the main changes made. Briefly put, goDiamond can expand given acceptance conditions to a truth table. Then, changing the acceptance condition amounts to adding another parent to the truth table and modifying the outcome. We first make a non-deterministic guess whether to change an existing acceptance condition or not (i.e. whether to add a link). Since we require that the original acceptance conditions are equivalent to the modified ones when the new argument is false, we only need to consider the case when the new argument is true. We make a non-deterministic choice for each entry in the truth table whether it should be true or false. We implemented this via choice rules. Requiring that the new links are supporting can be done via ASP constraints. Finally, optimization is achieved via ASP optimization statements. We used clingo, v4.4.0 [17] as the ASP engine.

We performed experiments with this new prototype using instances from `www.dbai.tuwien.ac.at/proj/adf/yadf/`, which are ADFs that were generated from the

instance set	#	# w/o errors	opt. found	0-cost	unsat.	timeout	median # (# w/o errors)
ABA2AF	600	261	214	57	47	0	27.82 (0.51)
Planning	600	585	211	4	367	7	2.415 (2.23)
Traffic	600	591	341	49	250	0	0.18 (0.17)
all	1800	1437	766	110	664	7	1.67 (0.79)

Table 1. Summary of experimental results.

AFs of domains ABA2AF, Planning2AF, and Traffic from ICCMA 2017 [16]. From the whole set, we selected the most challenging instances, resulting in 200 ADFs per domain (600 total). For each ADF we created three enforcement requests, as follows. We selected, at random, a subset of the arguments in the ADF with probability $\frac{1}{5}$ to include an argument. For each such subset, we selected, again randomly, which argument shall be enforced to be true and which to be false, with a probability of $p \in \{\frac{1}{5}, \frac{1}{10}, \frac{1}{20}\}$ that an argument is false (thus having more arguments to be enforced to be true). This resulted in 600 enforcements per domain, and 1800 enforcements overall. We used a timeout of 600 seconds on each individual query (enforcement).

The results are summarized in Table 1. The columns show instance set, number of queries, number of errors, number of instances where an optima was found, number of instances with optimum cost equal to 0 (trivial solutions), number of unsatisfiable queries, number of timeouts, and the median overall queries (overall instances that did not report an error). The errors are queries that clingo could not solve and reported an error message. We did not yet pinpoint a cause for this error. We speculate that they are due to large truth tables. Nevertheless, clingo did not report an error on many queries (1437 out of 1800).

From the results shown in Table 1, we see that many instances were solved within a reasonable amount of time (e.g. median is a few seconds). Out of those runs that did not report an error and were solved within timeout, 62 runs had a running time of at least 100 seconds and 1368 had a running time of less than 100 seconds.

6. Conclusions and Related Work

We have proposed to extend current (and future) dynamic operators on abstract argumentation frameworks by constraints to suit conditions "abstracted away" during instantiation. Towards feasibility of doing so, we have both showed complexity results and an empirical evaluation of an operator arising from usage of said constraints.

Related to our approach is a recent framework by [11], called control argumentation frameworks. In this framework, one can specify known, unknown, and controllable arguments by an agent, the last category being those arguments the agent can use in a debate. Similarities to our work are that a dynamic process is formally modelled and constrained (via, e.g., an encoding for using controllable arguments), differences are that we aim at general constraints for ADFs (instead of AFs as in their work). In [9,12], they encode allowed (desired) changes, and dynamic goals, in a logical language. In that way, similarly as in our work, they give constraints (as formulas) to encode allowed changes. Differently to their work, we based our results on ADFs and showed complexity results of classes of constraints. Furthermore, there are papers dealing with dynamics of AFs in a formalized way (i.e. presenting a framework for AF dynamics) such as [22], from which we differ in our work by presenting general constraints to be used for dynamic operators, but not focusing on details of a dynamic operation itself.

References

[1] Katie Atkinson, Pietro Baroni, Massimiliano Giacomin, Anthony Hunter, Henry Prakken, Chris Reed, Guillermo Ricardo Simari, Matthias Thimm, and Serena Villata. Towards artificial argumentation. *AI Magazine*, 38(3):25–36, 2017.

[2] Ringo Baumann. What does it take to enforce an argument? Minimal change in abstract argumentation. In *Proc. ECAI*, volume 242 of *FAIA*, pages 127–132, 2012.

[3] Ringo Baumann and Gerhard Brewka. Expanding argumentation frameworks: Enforcing and monotonicity results. In *Proc. COMMA*, volume 216 of *FAIA*, pages 75–86, 2010.

[4] Ringo Baumann, Wolfgang Dvořák, Thomas Linsbichler, Hannes Strass, and Stefan Woltran. Compact argumentation frameworks. In *Proc. ECAI*, volume 263 of *FAIA*, pages 69–74, 2014.

[5] Andrei Bondarenko, Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artif. Intell.*, 93:63–101, 1997.

[6] 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.

[7] Claudette Cayrol, Florence Dupin de Saint-Cyr, and Marie-Christine Lagasquie-Schiex. Change in abstract argumentation frameworks: Adding an argument. *J. Artif. Intell. Res.*, 38:49–84, 2010.

[8] Sylvie Coste-Marquis, Sébastien Konieczny, Jean-Guy Mailly, and Pierre Marquis. Extension enforcement in abstract argumentation as an optimization problem. In *Proc. IJCAI*, pages 2876–2882, 2015.

[9] Florence Dupin de Saint-Cyr, Pierre Bisquert, Claudette Cayrol, and Marie-Christine Lagasquie-Schiex. Argumentation update in YALLA (Yet Another Logic Language for Argumentation). *Int. J. Approx. Reasoning*, 75:57–92, 2016.

[10] Martin Diller, Adrian Haret, Thomas Linsbichler, Stefan Rümmele, and Stefan Woltran. An extension-based approach to belief revision in abstract argumentation. *Int.J.Approx.Reasoning*, 93:395–423, 2018.

[11] Yannis Dimopoulos, Jean-Guy Mailly, and Pavlos Moraitis. Control argumentation frameworks. In *Proc. AAAI*, 2018. To appear. Accessed version: `http://www.math-info.univ-paris5.fr/~jmailly/downloads/control_afs_AAAI18.pdf`.

[12] Sylvie Doutre, Andreas Herzig, and Laurent Perrussel. A dynamic logic framework for abstract argumentation. In *Proc. KR*, pages 62–71, 2014.

[13] 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):321–357, 1995.

[14] Paul E. Dunne, Wolfgang Dvořák, Thomas Linsbichler, and Stefan Woltran. Characteristics of multiple viewpoints in abstract argumentation. *Artif. Intell.*, 228:153–178, 2015.

[15] Stefan Ellmauthaler and Hannes Strass. The DIAMOND system for computing with abstract dialectical frameworks. In *Proc. COMMA*, volume 266 of *FAIA*, pages 233–240, 2014.

[16] Sarah Alice Gaggl, Thomas Linsbichler, Marco Maratea, and Stefan Woltran. Introducing the second international competition on computational models of argumentation. In *Proc. SAFA*, volume 1672 of *CEUR Workshop Proceedings*, pages 4–9, 2016.

[17] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. Theory solving made easy with clingo 5. In *Techn. Commun. of ICLP*, pages 2:1–2:15, 2016.

[18] Thomas Linsbichler, Jörg Pührer, and Hannes Strass. A uniform account of realizability in abstract argumentation. In *Proc. ECAI*, volume 285 of *FAIA*, pages 252–260, 2016.

[19] Sanjay Modgil and Henry Prakken. A general account of argumentation with preferences. *Artif. Intell.*, 195:361–397, 2013.

[20] Søren Nielsen and Simon Parsons. A generalization of Dung's abstract framework for argumentation: Arguing with sets of attacking arguments. In *Proc. ArgMAS*, pages 54–73. 2007.

[21] Sylwia Polberg. Understanding the abstract dialectical framework. In *Proc. JELIA*, volume 10021 of *LNCS*, pages 430–446, 2016.

[22] Nicolás D. Rotstein, Martín O. Moguillansky, Alejandro Javier García, and Guillermo Ricardo Simari. A dynamic argumentation framework. In *Proc. COMMA*, volume 216 of *FAIA*, pages 427–438, 2010.

[23] Hannes Strass and Stefan Ellmauthaler. goDiamond 0.6.6. `http://argumentationcompetition.org/2017/goDIAMOND.pdf`, 2017.

[24] Hannes Strass and Johannes P. Wallner. Analyzing the computational complexity of abstract dialectical frameworks via approximation fixpoint theory. *Artif. Intell.*, 226:34–74, 2015.

[25] Johannes P. Wallner, Andreas Niskanen, and Matti Järvisalo. Complexity results and algorithms for extension enforcement in abstract argumentation. *J. Artif. Intell. Res.*, 60:1–40, 2017.