Argument & Computation 11 (2020) 151–190
DOI 10.3233/AAC-190471
IOS Press

Structural constraints for dynamic operators in abstract argumentation

Johannes P. Wallner
Institute of Logic and Computation, TU Wien, Austria
E-mail: wallner@dbai.tuwien.ac.at

Abstract. Many recent studies of dynamics in formal argumentation within AI focus on the well-known formalism of Dung's argumentation frameworks (AFs). Despite the usefulness of AFs in many areas of argumentation, their abstract notion of arguments creates a barrier for operators that modify a given AF, e.g., in the case that dependencies between arguments have been abstracted away that are important for subsequent modifications. 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. To show applicability, we present two cases studies that adapt an existing extension enforcement operator that modifies AFs: in the first case study, we show how to utilize constraints in order to obtain an enforcement operator on ADFs that is allowed to only add support relations between arguments, and in the second case study we show how an enforcement operator on AFs can be defined that respects dependencies between arguments. We show feasibility of our approach by studying the complexity of the proposed structural constraints and the operators arising from the case studies, and by an experimental evaluation of an answer set programming (ASP) implementation of the enforcement operator based on supports.

Keywords: Abstract argumentation, dynamics of argumentation, argumentation frameworks, abstract dialectical frameworks, structured argumentation, computational complexity, constraints, answer set programming

1. Introduction

In the last decades there has been a steady stream of advances in formal approaches to argumentation within artificial intelligence (AI) [4,19]. Central to many formal models in argumentation are argumentation frameworks (AFs) due to the work by Dung in 1995 [53]. AFs can be seen as a reasoning 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 is witnessed by the fact that several approaches use AFs as their core reasoning engine [22,23,28,45,54,85,86]. Reasoning in this way is done via instantiating AFs in a principled way such that the arguments capture what can be argued for, and the attacks cover ways in which the arguments are conflicting. Semantics of AFs provide different means of conflict handling, in order to find acceptable arguments. Importantly, semantics of AFs only rely on the abstract view of AFs, i.e., abstract arguments and their conflict relations suffice to find acceptable arguments from an argumentative point of view for many applications. By inspecting the content of acceptable arguments one can trace back concrete claims that are acceptable.

AFs are also prominent as a formal basis for an emergent topic in formal argumentation, namely that of dynamics of argumentation [10,34,44,48,52], which can partially also be attributed to the easy-to-grasp conceptual nature of AFs. The investigation of dynamics of argumentation arose naturally in the

research community since argumentation is an inherently dynamic process of, e.g., posing arguments and counterarguments towards goals in a debate.

In their initial usage, AFs were applied on "static" forms of reasoning, i.e., reasoning in contexts that give rise to AFs that do not change. While the principles of dynamic scenarios are compatible with AFs, in that AFs can be dynamically adapted, care needs to be taken when utilizing AFs in a dynamic context. For instance, dynamic operations, if solely done on AFs, may miss certain dependencies between arguments that are apparent when considering the internal structure of the arguments, but which have been "abstracted away" during 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 the premise of a_1. 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 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.

While certain generalizations of AFs [31,39], particularly those that incorporate support relations, e.g., based on necessary supports [92], make it possible to explicate the apparently implicit support relation between a_1 and a_2, ultimately the same underlying problem remains: modifications on abstract frameworks may miss interdependencies between arguments.

In this paper we argue that despite these drawbacks of AFs when utilizing them in dynamic operations, abstract argumentation formalisms, as introduced by Dung [53] and to which this special issue is dedicated, 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 key structural constraints. Our goal in this paper is to provide means with which dynamic operations on abstract arguments can be extended so that (i) certain 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 [10]: given an AF and a set of arguments, the output is a modified AF whose semantics contains the given set of arguments as an extension. This operator finds interpretations as strategic moves in an argumentation [58]. Enforcement 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 Fig. 1(a). Analogously, one can view each layer of an operator as (a set of) constraints, each constraining the possible candidate solutions, see Fig. 1(b). In the case of enforcement, the first layer constrains the set of candidate AFs to be those that satisfy the semantical constraint, then the second layer constrains to choose, among those satisfying the first layer, those structures that are close to the input.

With this article, we contribute to the second layer, the structural layer, by giving properties that can be required by an output structurally, with the aim of giving developers of dynamic operators a further handle to extend operators to cover more application scenarios. As the formal foundation, we

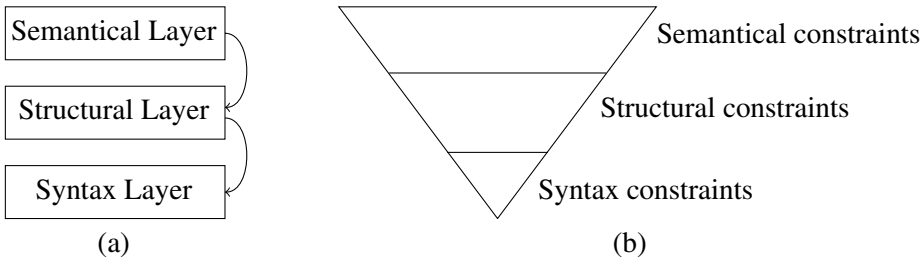

Fig. 1. (a) Three layers for dynamic operators and (b) outcomes that satisfy constraints of each layer.

use abstract dialectical frameworks (ADFs) [30,55] as the formal model that undergoes a change in this paper. Although ADFs are more complex than AFs, in the sense that the acceptance conditions of arguments 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 [95], and, thus, extend the reach of our contributions also to other formalisms than AFs.

To further our aim of supporting development of dynamic operators, we utilize the proposed constraints in two case studies. In the first case study we show how enforcement on AFs can be adapted to ADFs such that relations between arguments may only be modified in a way that corresponds to support between arguments, in contrast to attacks on AFs. In this way, we both aim to present a showcase of the constraints and show how 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.

In a second case study, we stay on the level of AFs, but adapt the existing enforcement operator to respect information of the instantiation. Concretely, for a general view on structured argumentation formalisms, we show how the constraints can be applied so that modifications to an AF do not miss interdependencies between arguments on their internal structure.

Since dynamic operators are, even without further constraints, often computationally hard [44,102], we study the complexity of the constraints we consider in this paper, and also the operators arising from the two case studies, in order to show which constraints can be feasibly "added" to operators. In order to have a clearer picture, we present an implementation in answer set programming (ASP) [29,71,88] and an empirical evaluation of the support enforcement on ADFs from the first case study. Our findings are that even when working in a more general abstract framework, and with complex constraints, state-of-the-art search engines (such as ASP) are capable of feasibly solving many instances up to certain sizes.

We summarize the main contributions of this article in the following.

- To situate our proposal in more concrete terms, we first adapt an existing extension enforcement operator for AFs [9,10,44] to ADFs.
- 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.
- We give two case studies for the constraints and adaptions of extension enforcement: one enforcement operation on ADFs that deals with supports and one enforcement operation on AFs that respects internal structure of arguments.

- To demonstrate feasibility, we show the theoretical complexity of checking whether an ADF satisfies the constraints, which is decidable in polynomial time in many cases, and we study the complexity of the operators arising from the case studies under admissible semantics.
- We implemented a prototype of the enforcement operation on ADFs using supports based on the Diamond system [62,63], and performed an experimental evaluation, showing good performance for a variety of ADF instances.

The article begins with recalling background in Section 2 (AFs and ADFs) and Section 3 (enforcement). We present and exemplify structural constraints in Section 4, and proceed to the case studies in Section 5 and Section 6. Our analysis of complexity of constraints and case studies is presented in Section 7, and our prototype implementation and experiments in Section 8. We close with a discussion on further existing operators for which our constraints can be applied (Section 9) and on related work (Section 10). This article significantly extends the conference version [101] by expanded discussion and illustration, more formal details on the first case study, inclusion of the second case study, giving full proofs, and updated experiments.

2. Abstract argumentation frameworks

In this section we recall the basics of two well-known abstract argumentation frameworks, namely Dung's argumentation frameworks (AFs) [53] and abstract dialectical frameworks (ADFs) [30]. There exists a whole range of abstract formalisms for argumentation, such as bipolar frameworks (BAFs) [3, 36], extended argumentation frameworks (EAFs) [83], argumentation frameworks with recursive attacks (AFRAs) [6] and their subsequent extensions [35,40,65], value-based frameworks (VAFs) [18], and preference-based frameworks (PAFs) [2], to name some of the prominent formalisms, which are also surveyed in recent articles [31,39]. Our choice of AFs and ADFs is justified by AFs being the common core to all these other approaches, and that ADFs generalize (translate to) many other approaches [95].

2.1. Argumentation frameworks

We start the formal preliminaries with Dung's argumentation frameworks (AFs) [53] and semantics for these frameworks [5]. An AF consists of a set of abstract arguments and directed attacks between these arguments.

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.

Example 2. AFs have a natural representation as directed graphs. Consider an AF $F = (\{a, b, c, d\}, R)$ with four arguments and the following attacks: $R = \{(a, b), (b, a), (a, c), (b, c), (c, d)\}$. This AF is illustrated in Fig. 2(a).

A central notion towards the semantics of AFs is that of defense of arguments.

Definition 2. Let $F = (A, R)$ be an AF. 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 *cf*, *adm*, *com*, *grd*, and

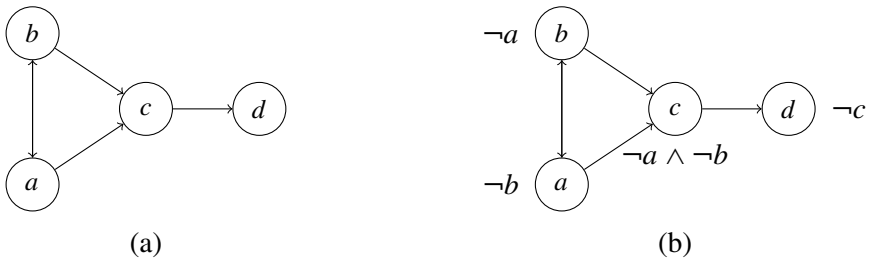

Fig. 2. Illustrations of the AF and the corresponding ADF from Example 3 and Example 4.

Table 1
Extensions of semantics σ from Example 2

σ	$\sigma(F)$
cf	$\{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, d\}, \{b, d\}\}$
adm	$\{\emptyset, \{a\}, \{b\}, \{a, d\}, \{b, d\}\}$
com	$\{\emptyset, \{a, d\}, \{b, d\}\}$
grd	$\{\emptyset\}$
prf	$\{\{a, d\}, \{b, d\}\}$

prf, which stand for conflict-free, 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 $\mathcal{F}_F(S) = \{x \in A \mid x \text{ is defended by } S\}$.

Definition 3. Let $F = (A, R)$ be an AF. An $S \subseteq A$ is conflict-free (in F) if there are no $a, b \in S$ such that $(a, b) \in R$. We denote the set of conflict-free sets by $cf(F)$. For an $S \in cf(F)$ it holds that

- $S \in adm(F)$ iff $S \subseteq \mathcal{F}_F(S)$;
- $S \in com(F)$ iff $S = \mathcal{F}_F(S)$;
- $S \in grd(F)$ iff S is the least fixed-point of $\mathcal{F}_F$; and
- $S \in prf(F)$ iff $S \in adm(F)$ and there is no $T \in adm(F)$ with $S \subset T$.

It is well-known that for any AF F it holds that $cf(F) \supseteq adm(F) \supseteq com(F) \supseteq prf(F)$. We use the term σ-extension to refer to an extension under a semantics $\sigma \in \{cf, adm, com, grd, prf\}$. We note that conflict-free sets and admissible sets are usually not referred to as a semantics, since, commonly, these two notions are seen as auxiliary concepts. Nevertheless, for the sake of uniformity, we will refer to the set of conflict-free sets and the set of admissible sets as conflict-free and admissible semantics. However, we do emphasize that by referring to conflict-free sets or admissible sets by conflict-free semantics or admissible semantics does not imply a prescriptive endeavor of placing these two concepts into the set of "full" argumentative semantics. Similarly, we sometimes call conflict-free sets or admissible sets conflict-free or admissible extensions, likewise with no intention of viewing them as being on the same level as extensions of other semantics.

Example 3. Consider the AF F in Example 2. For the considered semantics σ we show the corresponding extensions in Table 1.

2.2. Abstract dialectical frameworks

We recall ADFs from [30], which are based on earlier works [32]. We begin with basics from propositional logic and three-valued interpretations. Let A be a finite set of arguments (statements). 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 (**t**), false (**f**), or undefined (**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 $a \in A$. Further, let $I_{\mathbf{t}}$ ($I_{\mathbf{f}}$) be the interpretation assigning all arguments to **t** (**f**).

For Boolean formulas φ we consider the classical connectives of logical conjunction "$\wedge$", logical disjunction "$\vee$", logical negation "$\neg$", and material implication "$\rightarrow$". A two-valued interpretation I extends to the evaluation of a formula φ under I as usual, denoted by $I(\varphi)$.

For a formula φ and a three-valued interpretation I let $\varphi[I]$ be 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.

An interpretation I is equally or more informative than J, denoted by $J \leqslant_i I$, 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 $\leqslant_i$, i.e., $J <_i I$ if $J \leqslant_i I$ and $\exists a \in A$ s.t. $J(a) = \mathbf{u}$ and $I(a) \in \{\mathbf{t}, \mathbf{f}\}$.

Definition 4. 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 4. Fig. 2(b) 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.

Definition 5. Let $D = (A, L, C)$ be an ADF. The characteristic function Γ_D is defined by $\Gamma_D(I) = J$ with

$$J(a) = \begin{cases} \mathbf{t} & \text{if } \varphi_a[I] \text{ is a tautology,} \\ \mathbf{f} & \text{if } \varphi_a[I] \text{ is unsatisfiable, and} \\ \mathbf{u} & \text{otherwise.} \end{cases}$$

Semantics of ADFs are defined in a similar fashion as for AFs, based on the characteristic function.

Definition 6. Given an ADF D, an interpretation I

- is admissible in D iff $I \leqslant_i \Gamma_D(I)$;
- is complete in D iff $I = \Gamma_D(I)$;
- is grounded in D iff I is the least fixed-point of Γ_D; and
- is preferred in D iff I is $\leqslant_i$-maximal admissible in D.

We refer to the set of all admissible, complete, grounded, and preferred interpretations of an ADF D by $adm(D)$, $com(D)$, $grd(D)$, and $prf(D)$, respectively. In any ADF D, it holds that $prf(D) \subseteq com(D) \subseteq adm(D)$. Further, by definition, it holds that the grounded interpretation is a complete interpretation.

Table 2
Interpretations of semantics σ from Example 4

σ	$\sigma(D)$
com	$\{\{a \mapsto \mathbf{u}, b \mapsto \mathbf{u}, c \mapsto \mathbf{u}, d \mapsto \mathbf{u}\},$
	$\{a \mapsto \mathbf{t}, b \mapsto \mathbf{f}, c \mapsto \mathbf{f}, d \mapsto \mathbf{t}\},$
	$\{a \mapsto \mathbf{f}, b \mapsto \mathbf{t}, c \mapsto \mathbf{f}, d \mapsto \mathbf{t}\}\}$
grd	$\{\{a \mapsto \mathbf{u}, b \mapsto \mathbf{u}, c \mapsto \mathbf{u}, d \mapsto \mathbf{u}\}\}$
prf	$\{\{a \mapsto \mathbf{t}, b \mapsto \mathbf{f}, c \mapsto \mathbf{f}, d \mapsto \mathbf{t}\},$
	$\{a \mapsto \mathbf{f}, b \mapsto \mathbf{t}, c \mapsto \mathbf{f}, d \mapsto \mathbf{t}\}\}$

Example 5. For the ADF from Example 4, the σ-interpretations are shown in Table 2. Admissible interpretations are not shown due to their number: there are 11 admissible interpretations for this ADF.

We will also make use of a fragment of ADFs, called bipolar ADFs. Towards the definition, we recall the notion of attacking and supporting relations, as specified in ADFs.[1] Formally, we make use of an auxiliary notion: 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 $a \neq b$.

Definition 7. Let $D = (A, L, C)$ be an ADF. We say that a link $(b, a) \in L$ is

- *supporting* (in D) if for every two-valued interpretation I, $I(\varphi_a) = \mathbf{t}$ implies $I|^b_{\mathbf{t}}(\varphi_a) = \mathbf{t}$; and
- *attacking* (in D) if for every two-valued interpretation I, $I(\varphi_a) = \mathbf{f}$ implies $I|^b_{\mathbf{t}}(\varphi_a) = \mathbf{f}$.

Intuitively, if a link from a to b is attacking (supporting), then it cannot be the case that acceptance of a leads to a change of b's status to that of being accepted (not accepted).

An ADF D is called *bipolar* (is a BADF) if each link $(b, a) \in L$ is attacking or supporting. A link that is both attacking and supporting is called redundant.

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 Fig. 2). Note that the resulting ADF D_F has only attacking links which are not supporting. The semantics of an AF and its corresponding ADF coincide (by relating an extension E with interpretation I via $E = \{a \mid I(a) = \mathbf{t}\}$ [30]).

3. Enforcement on argumentation frameworks

Enforcement on AFs deals with the task of how to modify a given AF such that the modified AF satisfies certain semantical constraints. Usually also the number of modifications has to be minimum. Enforcement has been studied from several angles: possibility and impossibility results [9,10], representational aspects and use cases in a logical setting [25,51,58], computational aspects [44,90,102], and axiomatic studies [11,51,58,73].

Several variants of enforcement have been defined. For the purposes of this paper, we look at the enforcement variant called non-strict enforcement under strong expansions with a bounded number of expanded arguments [44]. This choice is mostly due to illustrative reasons; other variants fit the aims of the paper, as well. We base our results on this variant, since it is intuitively appealing: expansions relate

[1] We remark that the notion of "support" in abstract argumentation is not uniform. Common interpretations of support include support relation seen as deductive support [27], as necessary support [91,92], or as evidential support [93,94], see also [36].

to addition of arguments, with the strong variant indicating attacks originating only from new arguments. Compared to other operations studied, "strict" variants would place more restrictions on the semantic goal, and other types of expansions [8] have different restrictions which attacks may be added.

For an AF $F = (A, R)$, an expansion of F is any $F' = (A', R')$ with $A \subseteq A'$ and $R \subseteq R'$ (new arguments and new attacks may be added arbitrarily). Given an AF $F = (A, R)$, a strong expansion of F that adds new arguments A', with $A \cap A' = \emptyset$, is a modified AF $F' = (A \cup A', R')$ with $R \subseteq R'$ (all original attacks are in R') and if an attack $(a, b) \in R' \setminus R$ was added, then $a \in A'$ and $b \in A$ (new attacks originate from new arguments onto original arguments only).

Based on strong expansions, we define enforcement as follows.

Definition 8. Let $F = (A, R)$ be an AF, σ be a semantics, A' be a set with $A \cap A' = \emptyset$, and $S \subseteq A \cup A'$. An AF $F' = (A \cup A', R')$ non-strictly enforces S under σ if

- $R \subseteq R'$,
- for any $(a, b) \in R' \setminus R$ we have $a \in A'$ and $b \in A$, and
- $\exists S' \in \sigma(F')$ with $S \subseteq S'$.

That is, one needs to find a modified AF F' that strongly expands F by new arguments A' and S must be 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 S under σ and there is no $F' = (A \cup A', R')$ that non-strictly enforces S under σ and $|(R \Delta R')| < |(R \Delta R^*)|$, with Δ the usual symmetric difference: $X \Delta Y = (X \setminus Y) \cup (Y \setminus X)$.

Example 6. Consider the AF from Example 3. Say we want to enforce $\{a, d\}$ to be part of the grounded extension in a strongly expanded AF that can add argument e (i.e. $A' = \{e\}$). This can be achieved by adding the 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. The modified AF F^* is optimal: only one attack was added, and without modifications $\{a, d\}$ is not part of the grounded extension (the grounded extension is empty in the unmodified AF). See Fig. 3 for an illustration.

4. Structural constraints for dynamic operators

This section introduces several constraints to 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 and produce a modified ADF D^* as output. For the sake of readability, we will, unless stated otherwise, refer to a modified ADF by $D^* = (A^*, L^*, C^*)$. A dynamic operator then

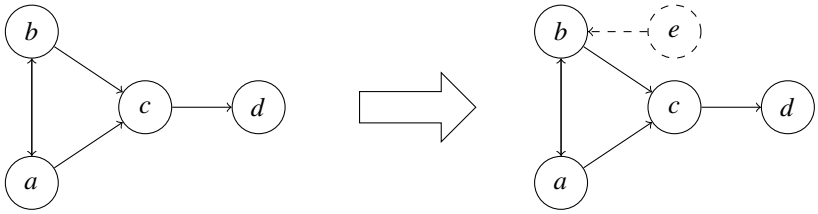

Fig. 3. Enforcing $\{a, d\}$ being part of the grounded extension, by adding argument e and attack (e, b) (Example 6).

defines certain constrains on the output, namely on (i) the semantics, (ii) the structure, and (iii) the concrete syntax. We first exemplify constraints on the enforcement operator from Section 3 when lifted from AFs to ADFs.

Definition 9. Let $D = (A, L, C)$ be an ADF, σ be a semantics, A' be a set of arguments to expand, and I be a three-valued interpretation over $A \cup A'$. An ADF $D' = (A \cup A', L', C')$ non-strictly enforces I under σ if

- $L \subseteq L'$,
- for any $(a, b) \in L' \setminus L$ we have $a \in A'$ and $b \in A$, and
- $\exists I' \in \sigma(D')$ s.t. $I \leqslant_i I'$.

That is, we want to enforce that I is "part" of a σ-interpretation by modifying D in the following way: there has to be a σ-interpretation I' such that if an argument a is assigned true (false) by I, then I' assigns true (false) to a, as well. Further, analogously as for AFs, one can look at optimality constraints: we say that an ADF $D^* = (A^*, L^*, C^*)$ is an optimal solution to non-strict enforcement if D^* non-strictly enforces I under σ, and there is no $D' = (A', L', C')$ with $|(L \Delta L')| < |(L \Delta L^*)|$ that non-strictly enforces I under σ. Intuitively, there has to be a σ-interpretation I' in the modified D^* such that the requirements of I are met (some arguments have to be true, some false, undecided arguments are not constrained). Compared to Definition 8 the only major change is to respect the three-valued semantics of ADFs.

Within the three-layered view, this adapted enforcement operator works as follows:

(1) the semantical constraint specifies the "goal", i.e., the arguments to be true/false in a σ-interpretation,
(2) the structural constraints state that only expansions of the original ADF may be considered (with possibly only considering optimal solutions), and
(3) a syntactical constraint stating that the output shall be an ADF (leaving the form of acceptance conditions, e.g., regarding normal forms, open).

Example 7. Consider again the ADF D from Example 4. Say we want to enforce that both a and d are true in the grounded interpretation. Further, say we may add only argument e. This means, we desire to enforce interpretation $I = \{a \mapsto \mathbf{t}, b \mapsto \mathbf{u}, c \mapsto \mathbf{u}, d \mapsto \mathbf{t}, e \mapsto \mathbf{u}\}$ (a and d must be true, all other arguments are unconstrained). Unmodified, both arguments are undecided in the grounded interpretation of D, since the grounded interpretation assigns all arguments to undecided. The enforcement can be achieved by adding an argument e (similarly as in Example 6 for enforcement on AFs) that "attacks" argument b, by adapting the acceptance condition of b to $\varphi'_b = \neg a \wedge \neg e$ (from originally $\varphi_b = \neg a$). This enforcement is shown in Fig. 4.

We now proceed to introducing several structural constraints that are intended to refine candidate ADFs for a dynamic operator on ADFs. The enforcement operator defined above is one example for such an operator, but in the following we do not assume a concrete operator (however, we connect several constraints to the enforcement operator). Formally, a constraint c is specified in such a way that one can decide whether an ADF satisfies the constraint. The intention is then to choose, among all ADFs satisfying the given semantical constraints, e.g., from the ADF enforcement operator defined above, one ADF satisfying a given structural constraint c. Many of the subsequent constraints are intended to be used in a mix of several constraints. That is why we begin with defining how to state combinations of constraints.

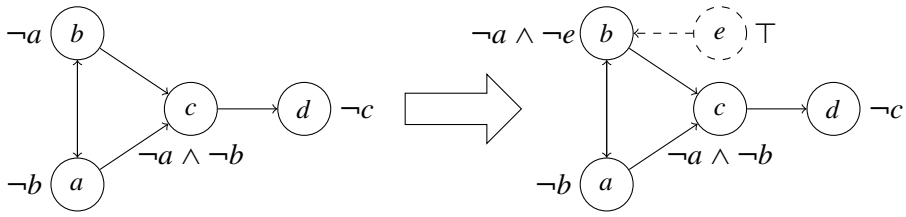

Fig. 4. Enforcing $\{a, d\}$ being true in the grounded interpretation: adding argument e, with $\varphi_e = \top$ and adapting φ_b to $\varphi_b' = \neg a \wedge \neg e$ (Example 7).

Boolean combinations of constraints. In the following, when $K = \{c_1, \ldots, c_n\}$ is a set of (structural) constraints, we consider also the 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. For concrete atomic 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), argument sets A and A', and sets of links 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 (in the constraints A and L give a lower bound and A' and L' give an upper bound).

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 that can expand the set of arguments, but the operator is not required to add all possible arguments (in contrast to the enforcement operator defined above). Say, that we can expand by arguments a_1 and a_2. Suppose further that by adding a_1 to the original framework we can enforce the given interpretation. However, consider now the case that a_1 is a super argument of a_2, which is specified by the contents of both arguments. 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 an implication: $((a_1 \in A^*) \rightarrow (a_2 \in A^*))$. That is, this Boolean constraint specifies that whenever a_1 is part of the output ADF D^* (e.g., by expansion), then also a_2 must be present in D^*. In Section 6 further uses of such implications are exemplified for an enforcement operator respecting the internal structure of arguments.

Constraints on links. Next, we consider constraints on links. For notation, we say that, in the ADF D^*, links 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^*) belongs to specified graph class

Example 9. Constraining the type of links can imply that the output ADF 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, one can constrain that a part of the ADF is a BADF.

If each link is constrained to be attacking and not supporting, then the output ADF belongs to the family of frameworks defined in [87], which we call SETAFs here, in case no acceptance condition is unsatisfiable. These SETAFs are similar to AFs, but sets of arguments attack an argument.

That ADFs with only attacking links and SETAFs have a connection has been established by a translation of SETAFs to ADFs [95]. That ADFs with only attacking links have a corresponding form as SETAFs, in case no acceptance condition is unsatisfiable, can be shown by considering that each acceptance condition in an attack-only ADF can be represented via a Boolean formula in conjunctive normal form with only negative literals. A similar result has been shown for BADFs in [61, Theorem 3.1.23]; we specialize this result to attack-only ADFs and state a direct proof for the sake of completeness.

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 $\varphi_a^* \in C^*$ with $par_{D^*}(a) \neq \emptyset$ can be expressed as a Boolean formula in conjunctive normal form (CNF) with only negative literals.*

Proof. Let $a \in A^*$. If $par_{D^*}(a) \neq \emptyset$, then φ_a^* is neither tautological nor unsatisfiable, since the incoming links are not redundant (are not supporting). Thus, φ_a^* is both satisfiable and refutable. Consider a two-valued interpretation I s.t. $I(\varphi_a^*)$ evaluates to false. By property of attacking-only links, it follows that switching the truth value of any set of parents to true in I does not change the outcome. Now, take each interpretation I s.t. (i) $I(\varphi_a^*)$ evaluates to false and (ii) there is no I' s.t. $I'(\varphi_a^*)$ evaluates to false and $\{x \mid I'(x) = \mathbf{t}\} \subsetneq \{x \mid I(x) = \mathbf{t}\}$ (I is subset-minimal w.r.t. arguments assigned to true). Let $\mathcal{I}$ be all such interpretations. Define $\varphi_a' = \bigwedge_{I \in \mathcal{I}}(\bigvee_{I(x)=\mathbf{t}} \neg x)$. We show that $\varphi_a^* \equiv \varphi_a'$ holds. Let J be a two-valued interpretation satisfying φ_a^*. Then for each $I \in \mathcal{I}$ we have $\{x \mid I(x) = \mathbf{t}\} \not\subseteq \{x \mid J(x) = \mathbf{t}\}$ (otherwise J would not satisfy φ_a^*). Thus, J satisfies $\bigvee_{I(x)=\mathbf{t}} \neg x$ (at least one argument assigned to true by I is false in J). This implies that J satisfies φ_a'. Assume that J satisfies φ_a'. If J does not satisfy φ_a^* then there is a $J' \in \mathcal{I}$ s.t. $\{x \mid J'(x) = \mathbf{t}\} \subseteq \{x \mid J(x) = \mathbf{t}\}$. This implies a clause $\bigvee_{J'(x)=\mathbf{t}} \neg x$ in φ_a'. This implies that J does not satisfy this clause, a contradiction. Thus, J satisfies φ_a^*. □

From this result it can be inferred, via [78, Proposition 1], that if each acceptance condition is either equal to $\top$ or in CNF with negative literals, the ADF in question can be written as a SETAF. However, unsatisfiable conditions ($\varphi_a = \bot$) have no direct analogue. The underlying intuition is that if a set of arguments X attacks, in a SETAF, an argument a, then this can be written as $\bigvee_{b \in X} \neg b$ (one of the attackers must be "out" or false for the set-attack to be countered). If more sets attacking a exist, then this can be represented as a Boolean formula in CNF: $\bigwedge_{(X,a)} \bigvee_{b \in X} \neg b$. An unsatisfiable formula $\varphi \equiv \bot$ cannot be directly represented in this way.

Example 10. Constraining that the output D^* belongs to a sub-family of ADFs also may lead to the case that there is no ADF (of that sub-family) that satisfies the given constraints. Consider constraining

the output to have two arguments $\{a, b\}$, and the output shall have as its complete semantics the exact correspondence $\Sigma = \{\{a \mid I(a) = \mathbf{t}\} \mid I \in com(D^*)\}$ with $\Sigma = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$. That is, the semantical constraint states that there has to be a complete interpretation assigning no argument to true, one that assigns only a to true, one that assigns only b to true, and one that assigns both arguments to true. Such a semantical constraint does not correspond to the requirement we defined for enforcement, however, dynamic operators requiring exact semantics in the output framework exist, we recall some in Section 9. Requiring that each link in the output is attacking and non-supporting leads to non-existence of solution ADFs. If there is no link in the ADF, then there would be only one complete (grounded) interpretation. If there is a link, such a link has to originate from either a or b and lead to either a or b. Since this link is not supporting, but attacking, it follows that one acceptance condition is not equivalent to $\top$ or $\bot$. This implies that assigning one argument to true (or both) leads to non-acceptance of an argument. This implies that $\{a \mapsto \mathbf{t}, b \mapsto \mathbf{t}\}$ is not complete. Another way of seeing this fact is by considering any (set-)attack: such an attack cannot exist, since $\{a \mapsto \mathbf{t}, b \mapsto \mathbf{t}\}$ is complete. To have Σ as the semantical result under complete semantics on AFs, one needs more arguments (see [12,57]). In general ADFs, such a correspondence is possible without more arguments when allowing the use of supports.

Constraining the underlying graph structure of an ADF, i.e., (A^*, L^*), can be useful, as well, e.g., with constraint (L4). Example graph classes are directed acyclic graphs or bipartite graphs. For instance, directed acyclic graphs have appealing properties: one can view "leaf" arguments (i.e., arguments x whose dependencies $par_D(x)$ are empty) as, e.g., undisputed or evidential facts, or assumptions. In some formal approaches to (abstract) argumentation, arguments without dependencies are necessary, for instance for evidential argumentation systems (EASes) [93]. Computationally, acyclicity exhibits milder complexity than general graph structures, for several reasoning tasks, both for AFs [53,59,60] and for ADFs [77]. Bipartite graphs also enjoy interesting properties [56], and can be seen as arguments from two parties corresponding to the bipartite partition.

Constraints on acceptance conditions. We proceed to constraints on acceptance conditions. Given an ADF $D^* = (A^*, L^*, C^*)$, a $\varphi_s \in C^*$, $v \in \{\mathbf{t}, \mathbf{f}\}$, a three-valued interpretation I, a two-valued interpretation I', and a formula ψ, we define the following constraints. Recall that by $\varphi[I]$ we denote the formula obtained by replacing arguments assigned to true by $\top$ and to false by $\bot$, and by $I'(\varphi)$ we denote the evaluation of the Boolean formula φ by a two-valued interpretation I'.

(C1) $I'(\varphi_s) = v$
(C2) $\varphi_s[I]$ satisfiable (refutable, tautological, or unsatisfiable)
(C3) $\varphi_s = \psi$
(C4) $\varphi_s[I] \equiv \psi$

That is, (C1) specifies that the acceptance condition of argument s shall evaluate to truth value v for a given two-valued interpretation I'. Constraints of type (C2) state that the partially evaluated acceptance condition of argument s must be satisfiable (or refutable, tautological, or unsatisfiable), indicating, e.g., that an argument can be accepted, in a certain context (or not). The third type of constraints says that an acceptance condition must be (exactly) equal to a given formula, and the last one that a partially evaluated acceptance condition (under three-valued interpretation I) must be equivalent to a given formula.

Example 11. Constraints of type (C2) can be used, e.g., to require the output ADF to belong to the subclass of AFs. Let $I_{\mathbf{u}}$ ($I_{\mathbf{f}}$) be an interpretation with all arguments assigned to undecided (false). One

can require that ($b \in A^* \to \varphi_b[I_{\mathbf{f}}]$ tautological) and ($(a, b) \in L^* \to \varphi_b[I_{\mathbf{u}}|^a_{\mathbf{t}}]$ unsatisfiable) for parents of b. In other words, argument b is acceptable if all parents are false ($\varphi_b[I_{\mathbf{f}}]$ is tautological), and not acceptable if parent a is true ($\varphi_b[I_{\mathbf{u}}|^a_{\mathbf{t}}]$ updates parent a to true and then b is not acceptable if the resulting formula is unsatisfiable). To make this concrete, consider that argument b shall be attacked by arguments a and c, in the sense of an AF. Say one specifies that a and c are the only parents of b. Define $I_1 = \{a \mapsto \mathbf{f}, b \mapsto \mathbf{f}, c \mapsto \mathbf{f}\}$ and $I_2 = \{a \mapsto \mathbf{u}, b \mapsto \mathbf{u}, c \mapsto \mathbf{u}\}$. Stating ($b \in A^* \to \varphi_a[I_1]$ tautological) says that, if b is in the output ADF, then $\varphi_a[I_1] \equiv \top$ (when all arguments are false, b is acceptable). Further, by stating ($(a, b) \in L^* \to \varphi_b[I_2|^a_{\mathbf{t}}$ unsatisfiable) we get an attack from a onto b: $I_2|^a_{\mathbf{t}} = \{a \mapsto \mathbf{t}, b \mapsto \mathbf{u}, c \mapsto \mathbf{u}\}$ and $\varphi_b[I_2|^a_{\mathbf{t}}] \equiv \bot$. Taken together, they imply that an argument can be accepted if all arguments are assigned false and assigning one argument with an incoming link to true implies non-acceptance.

Another use case for the constraints is when considering the internal structure of arguments. If two structures of arguments are logically inconsistent, and this is to be expressed directly within acceptance conditions, the preceding attack-like constraint can be used as well.

Further, one can constrain supports. For instance, one can specify that supporting links are a kind of necessary support: $((a, b) \in L^+) \to (\varphi_b[I_{\mathbf{u}}|^a_{\mathbf{f}}]$ unsatisfiable). Intuitively, this specifies that if a parent is false then the child argument cannot be acceptable.

Another type of constraint that can be expressed is to require no change between certain dependencies between arguments. For instance, say an argument a has several parents in an original ADF and a new one (b) in a modified ADF. Further, 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 $\varphi^*_s[I_{\mathbf{u}}|^b_{\mathbf{f}}] \equiv \varphi_s$ (C4), with φ_s being the original acceptance condition of s. A use case of such constraints is also discussed in the first case study in Section 5.

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' be arguments.

(Char) for each three-valued interpretation I it holds that $\Gamma_{D^*}(I)(s) = v$ implies $\Gamma_{D^*}(I)(s') = v'$

Note that we restrict the three-valued interpretations I to be over the universe of arguments of D^*.

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. A potential use of this constraint is that an argument shall be rejected whenever one of its sub arguments is rejected (e.g. via an attack from another argument). While this seemingly contradicts an intuition that dependencies between arguments are to be expressed as links, such indirect dependencies can occur: e.g., consider three arguments a, b, and c, with b a sub argument of c. When acceptance of a leads to non-acceptance of b, e.g., through an attack from a, then c should be, likewise, rejected (unless exceptional cases occur). While this can be expressed via links from b to c, a constraint of type (Char) can express such behavior inside acceptance conditions, modeling potentially more complex scenarios. For instance, in some approaches to instantiation of AFs, no direct relation between sub arguments is specified.

Weights and optimization. For optimization constraints, define the cost of an ADF D via $cost(D)$. For this paper, we let the cost function be abstract.

(O1) $cost(D^*) \leqslant k$
(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. That is, for an input ADF $D = (A, L, C)$ and output ADF $D^* = (A^*, L^*, C^*)$ the cost is $cost(D^*) = |L \Delta L^*|$, i.e., the cost is the cardinality of the symmetric difference between the input and output links.

5. Case study: An enforcement operator for ADFs based on supports

In this section we use some of the proposed constraints to adapt the enforcement operator on ADFs to allow for only supporting links to be added. For the sake of readability, we will continually develop this operator.

We adapt non-strict enforcement under strong expansions of ADFs, as defined in Definition 9. We briefly recall this operator. 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) (first two items of Definition 9). The second condition is that $\exists I' \in \sigma(D^*)$ s.t. $I \leqslant_i I'$. For illustrative purposes, we initially drop the condition of optimality on modified links.

Consider, first, a variant of this operator that respects the following constraint:

$$\big((a', b) \in L^*\big) \rightarrow \big(a', b\big) \in L^+ \quad \text{for each } b \in A \text{ and } a' \in A'.$$

That is, all new links shall be supporting. While, technically, this achieves a "support enforcement" operation, we remark that such an operator enjoys rather high freedom in achieving the enforcement.

Example 14. Consider ADF $D = (\{a, b, c\}, L, C)$ with $\varphi_a = b$, $\varphi_b = c$, and $\varphi_c = \bot$. Assume that we expand with an argument d and that we want to enforce a to be true in an admissible interpretation. This can be achieved simply by modifying φ_a to $\varphi'_a = b \vee \neg b$ (see Fig. 5(a)). Note that this modified ADF is an expansion, and new links are supporting (there are no new links).

Another potentially unintended example is when we desire to have an argument a false in an admissible interpretation of an expanded ADF. Say we have only one argument a, and may add argument b. The following modification can be applied: $\varphi^*_a = \varphi_a \wedge b$ and $\varphi^*_b \equiv \bot$. That is, one can specify that argument a is accepted only if b is true, and stating that b is never acceptable. Intuitively, one "supports" an argument by requiring a new argument, and declaring that the new argument is not acceptable.

Let us utilize the structural constraints to specify permissible changes in a more rigorous way. Consider that acceptance conditions shall be unchanged if new arguments are rejected (similarly as in Example 11 and constraint type (C4)):

$$I_{\mathbf{u}}|^{A'}_{\mathbf{f}}\big(\varphi^*_a\big) \equiv \varphi_a \quad \text{for each } a \in A.$$

That is, when evaluating an acceptance condition partially under the interpretation assigning all expanded arguments A' to false, the resulting formula shall be equivalent to the original (unmodified) acceptance condition.

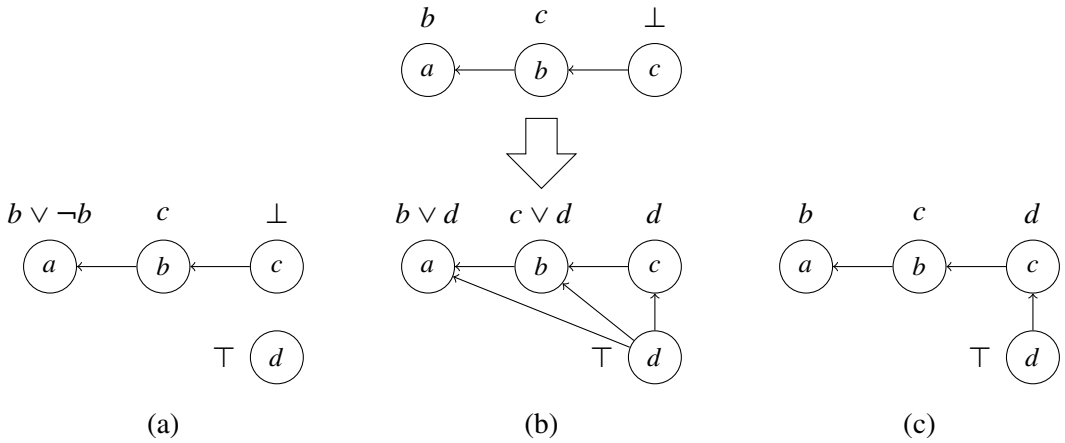

Fig. 5. Enforcing a being true in an admissible interpretation via support enforcement: without further constraints (a), without modifying acceptance conditions when expanded arguments are rejected (b), and additionally when optimizing added links (Example 14).

Example 15. Continuing Example 14, under the new constraint, the modification shown in Fig. 5(a) is not allowed anymore. The reason for this is that the constraint just introduced is violated: $\varphi'_a|^d_{\mathbf{f}} = b \vee \neg b[d \mapsto \bot] = b \vee \neg b \not\equiv \varphi_a$. That is, when evaluating the modified acceptance conditions partially for when d is false, the resulting formula is not equivalent to the original acceptance condition. One way to achieve the enforcement request of having a being true in an admissible interpretation is shown in Fig. 5(b), where several links were added from expanded argument d onto all others, requiring now d to be true.

In light of the preceding example, in addition we now specify that the cost of a solution D^* is $|L \Delta L^*|$ (similarly as for enforcement on AFs). A solution D^* is optimal if there is no other solution with strictly less cost.

Example 16. Under the optimization constraint, an optimal solution, for enforcement of Example 14, would be to having d support c, by modifying the acceptance condition to $\varphi'_c = d$ (see Fig. 5(c)). Another optimal solution is to directly support a via d, by modifying φ_a to $\varphi'_a = b \vee d$.

Finally, we specify that expanded arguments are always acceptable: for each $a' \in A'$ we have $\varphi^*_{a'} \equiv \top$. Summarizing, we use the following constraints (with constraint type on the right):

$$\big((a', b) \in L^*\big) \rightarrow \big(a', b\big) \in L^+ \quad \forall b \in A\ \forall a' \in A' \qquad \text{(L1), (L2)}$$
$$I_{\mathbf{u}}|^{A'}_{\mathbf{f}}\big(\varphi^*_a\big) \equiv \varphi_a \quad \forall a \in A \qquad \text{(C4)}$$
$$\varphi^*_{a'} \equiv \top \quad \forall a' \in A' \qquad \text{(C4)}$$

Additionally, this operator uses constraints of type (A1) and (L1) in order to specify expansions, and (O2) for the optimization. We call the corresponding operator that is defined via Definition 9 and respects these three constraints support enforcement.

Example 17. 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).

As the reader might have guessed, when looking at Example 16, support enforcement is possible if no argument is enforced to be false.

Proposition 2. *Let $D = (A, L, C)$ be an ADF, I be a three-valued interpretation over $A^* = (A \cup A')$ with $A \cap A' = \emptyset$ for a non-empty A', and σ be a semantics. If $\nexists a \in A^*$ s.t. $I(a) = \mathbf{f}$ then there exists an ADF $D^* = (A^*, L^*, C^*)$ that enforces I under σ for the support enforcement operator.*

Proof. For any argument a that is assigned true by I one can modify the acceptance condition by $\varphi_a^* = \varphi_a \vee b$ for any $b \in A'$ (then a is true in the grounded interpretation of the resulting modified ADF). □

The preceding proposition implies straightforward solutions to the support enforcement for arguments to be enforced to be true. The same simple modification is not applicable for arguments to be enforced to be false: a direct support of a new argument onto such an argument can only lead to more cases of acceptance. In such a case, an indirect support of, e.g., an attacker can lead to a successful enforcement. In any case, while we abstain in the above definition from introducing further constraints to support enforcement, several further constraints can be feasibly added, such as constraints which links are allowed to be added, and how acceptance conditions may change. In Section 7, we investigate which further constraints can be added to support enforcement (or other operations) without increased computational complexity.

Further, if enforcement is possible (independently of whether arguments are to be true/false), for a concrete instance, then it is possible with a singleton argument to expand, whenever the interpretation that represents the enforcement goal assigns undecided to the expanded arguments (if two ore more expanded arguments must be true, then one cannot restrict to one expanded argument). Formally, if one can enforce the desired status among the original arguments, then one can do so, for the original arguments, with a single new argument.

Proposition 3. *Let $D = (A, L, C)$ be an ADF, I be a three-valued interpretation over $A^* = (A \cup A')$ with $A \cap A' = \emptyset$ for a non-empty A', and σ be a semantics. If there is an ADF $D^* = (A^*, L^*, C^*)$ that enforces I under σ for the support enforcement operator, then there also exists an ADF $D_1^* = (A \cup \{a'\}, L_1^*, C_1^*)$, that enforces $I' = \{I(a) \mid a \in A\} \cup \{a' \mapsto \mathbf{t}\}$ under σ for the support enforcement operator for a singleton expanded argument set $\{a'\}$.*

Proof. Assume that D^* enforces I under σ, for any of the considered semantics, and that there is a $J \in \sigma(D^*)$ with $I \leqslant_i J$. Construct D_1^* from D^* by replacing each newly added variable (argument in A') by a distinguished argument a'. We claim that I' is enforced under σ with D_1^*. Consider first that J is complete (which is the case when $\sigma \in \{com, prf, grd\}$). We argue that $\Gamma_{D^*}(J)(x) = \Gamma_{D_1^*}(J)(x)$ for each $x \in A \cup \{a'\}$. Since J is complete it follows that J assigns $\mathbf{t}$ to all arguments in A' (they have tautological acceptance conditions). Replacing them with a single argument does not change (partial) evaluation of acceptance conditions. From this, it follows that J, when restricted to arguments $A \cup \{a'\}$ is complete in D_1^*. For complete-base semantics, the result follows in a straightforward fashion: if J is grounded (preferred) in D^*, then J, projected onto arguments in D_1^*, is grounded (preferred) in D_1^* (since partial

evaluations under three-valued interpretations that assign true to all expanded arguments are preserved from one framework to the other).

Consider now that $J \in adm(D^*)$ is admissible, but not necessarily complete. Let J' be the same interpretation, but restricted to $A \cup \{a'\}$ and assigning true to a'. Let $a \in A$ be an arbitrary argument in the original ADF. Let φ_a be the original acceptance condition of a, and φ_a^* be the modified one in D^*. Let $\varphi_a' = \varphi_a^*[A' \mapsto a']$, i.e., each expanded argument is replaced by a'. If $J(a) \in \{\mathbf{t}, \mathbf{f}\}$, then $\varphi_a^*[J]$ is tautological or unsatisfiable. In both cases, $\varphi_a'[J']$ has the same property, i.e., is tautological if $\varphi_a^*[J]$ is tautological and is unsatisfiable if $\varphi_a^*[J]$ is unsatisfiable (possibly, more arguments are assigned to be true in J' than in J w.r.t. a's acceptance condition, yet this preserves tautologies and unsatisfiability; due to expanded arguments' tautological acceptance conditions, no expanded argument is assigned false). Consider the last case, i.e., that a is assigned undecided by J. Then $\varphi_a^*[J]$ and $\varphi_a'[J']$ might not share the property of being a tautology or being unsatisfiable, yet, J' is, nevertheless, admissible (whenever $J'(a) = \mathbf{u}$ it holds trivially that $J'(a) \leqslant_i \Gamma_{D^*}(J')(a)$). In conclusion, J' is admissible in D_1^* and, by construction, $I'(x) \leqslant_i J'(x)$ for each $x \in A$. This achieves the enforcement request. □

6. Case study: An enforcement operator for structured argumentation

In this section we will use our structural constraints to adapt the non-strict enforcement operator on AFs (Definition 8) to respect knowledge bases from structured argumentation the AF was instantiated from. On a high-level, the enforcement operator we define receives as input an AF instantiated by a part of a given knowledge base, and aims to find an expansion of that AF enforcing certain arguments, with expansions respecting the knowledge base. Formal approaches to structured argumentation are quite heterogeneous [7], which is why we abstract from these approaches, in order to capture the intuitions of several formalisms.

Structured argumentation approaches [21,22,28,68,72,85] provide formal recipes for instantiating arguments and their relationships from a given knowledge base. In several cases the formalism that is instantiated is an AF [28,85]. We assume that a knowledge base is given as a set B of elements. This set may contain facts, rules, or further ingredients required to instantiate arguments. In this paper we do not specify these elements. For our purposes we assume only a few functions that a structured argumentation approach is composed of: (i) a function *args* that returns all (abstract) arguments that can be generated from a knowledge base, (ii) a given function *kb* that returns for a set of arguments their contents, i.e., returns the subset $B' \subseteq B$ that is needed to instantiate the arguments, and (iii) a function *att* that returns all attacks that can be instantiated. Formally, we also assume that the functions behave well, i.e., we have $kb(args(B)) \subseteq B$ for any knowledge base B (every knowledge base element, when inspecting instantiated arguments from knowledge base B, is in B) and we assume that it holds that $A \subseteq args(kb(A)$ (when looking at the base of a set of arguments, then all these arguments can be constructed from that base). For *att*, we assume that $(a, b) \in att(B)$ implies $\{a, b\} \subseteq args(B)$. Further, we assume that for $B' \subseteq B$ we get $f(B') \subseteq f(B)$ for $f \in \{args, att\}$, and for argument sets $A' \subseteq A$ we have $kb(A') \subseteq kb(A)$. This means that all functions *args*, *att*, and *kb* are $\subseteq$-monotone. While subset monotonicity for the functions *args* and *kb* appear rational, there are cases when *att* is, in fact, not subset-monotone: in presence of preferences, that are to be included in a knowledge base, addition of preferences might affect presence of attacks non-monotonically. We discuss impact of this at the end of this section, but leave the main part of this section to deal with the case of $\subseteq$-monotone *att* functions.

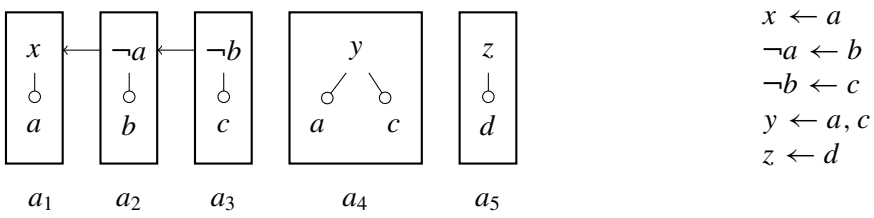

Fig. 6. A simple structured (assumption-based) argumentation framework (Example 18). For the ABA framework consisting of four assumptions a, b, c, and d, and rules as shown in (b), several arguments can be constructed, with five arguments shown in (a). Attacks arise when an argument concludes a negated assumption of another argument (e.g., argument a_2 attacks argument a_1 on assumption a).

Example 18. We exemplify the functions *args*, *kb*, and *att* by a simple knowledge base, and associated AF that is constructed in the case of assumption-based argumentation frameworks (ABA) [28]. Similar functions can be defined, e.g., for the ASPIC+ framework [85]. We will not explicate the definitions of ABA, but stay on an intuitive level, and also, for illustration purposes, simplify some notions.

Two main ingredients for ABA are assumptions and rules. Say we have four assumptions $\{a, b, c, d\}$, and rules as shown in Fig. 6: from a one can derive x, from b one can derive the negation (contrary) of a, from c one can derive the negation of b, from a and c together one can derive y, and finally from d one can derive z. These derivations directly correspond to the arguments a_1 to a_5. By the contraries the arguments derive, one can infer that a_2 attacks a_1 and that a_3 attacks a_2. That is, for $B = \{a, b, c, d\}$ we get $args(B) = \{a_1, a_2, a_3, a_4, a_5\}$, $att(B) = \{(a_2, a_1), (a_3, a_2)\}$, and e.g., $kb(\{a_1, a_4\}) = \{a, c\}$.[2] In Fig. 7 we show for each $B' \supseteq \{a, b\}$ the corresponding abstract argumentation framework. For instance, for $B' = \{a, b, c\}$ we have F_3.

In the following we assume that we are given both a knowledge base B, and an AF $F = (args(B), att(B))$ instantiated from this base. Further, we assume a subframework $F' = (A', R')$ as the current state of argumentation such that $B' \subseteq B$ and $A' = args(B')$ and $R' = att(B')$. One interpretation of this context is that the arguments in F' have been uttered, while the remaining part of the knowledge base $B \setminus B'$ has not been uttered (or, is "private" to an agent). For instance, in Fig. 7, F_4 corresponds to the full AF, for the knowledge base from Example 18, and F_1 could be a current AF.

The enforcement operator we now define aims to provide means to find a knowledge base B^*, with $args(B') \subseteq args(B^*) \subseteq args(B)$, such that a desired argument (or set of arguments) is acceptable, under a certain semantics σ. Towards the operator, we again adapt the existing enforcement operator in AFs (recall Definition 8), and modify this operator, as follows (the full definition is given in Definition 10). First, for the expansion we do not require strongness nor addition of all arguments that may added. Further, we apply the following constraints (the arguments that may be added are $args(B) \setminus args(B')$).

[2] In this example we included only assumptions as part of B, i.e., only assumptions as part of the knowledge base, and incorporated the rules (and contraries) into the functions *kb*, *args*, and *att*. We have chosen this way mainly for illustration. Including rules and contraries into the knowledge base B, and adapting the functions accordingly, poses no barrier to our approach. In fact, in the definition of ABA [28] one usually includes rules and contraries in a deductive system that can be seen as part of a knowledge base. With our formalization the enforcement operator we will define presumes that rules and contraries have to be included in a solution AF, while assumptions are optional. Finally, we note that in the original ABA definition further arguments are present, namely those that correspond to assumptions without application of rules, e.g., an argument corresponding only to assumption a without derivation of x. We omitted these arguments, again, for illustration purposes.

(1) $args(B') \subseteq A^* \subseteq args(B)$
(2) $att(B')|_{A^*} \subseteq R^* \subseteq att(B)|_{A^*}$
(3) $(a_1 \in A^* \wedge \cdots \wedge a_n \in A^*) \rightarrow b \in A^* \; \forall b \in args(kb(\{a_1, \ldots, a_n\}))$ and each $\{a_1, \ldots, a_n\} \subseteq args(B)$
(4) $(a \in A^* \wedge b \in A^*) \rightarrow (a, b) \in R^*$ for $(a, b) \in att(B)$

In words, we aim to find an AF $F^* = (A^*, R^*)$ such that (1) the set of arguments shall be between the current state, $\text{arg}(B')$, and the full knowledge base $\text{arg}(B)$; similarly (2) specifies that the attacks shall be between those two extremes, (3) states that whenever an argument b can be constructed from the components of existing arguments in A^* it has to be included, and (4) requires that no attack between arguments may be omitted. We dub the resulting operator structured enforcement. For clarity, the full formal definition is also given next.

Definition 10. Let B be a knowledge base and $B' \subseteq B$. Let $F = (A', R')$ be an AF such that $args(B') = A'$ and $att(B') = R'$, σ be a semantics, and $S \subseteq A'$. An AF $F^* = (A^*, R^*)$ non-strictly enforces S under σ if

- $\exists S' \in \sigma(F^*)$ such that $S \subseteq S'$,
- the constraints (1)–(4) are satisfied, and
- $|R' \Delta R^*|$ is minimum over all AFs satisfying the previous conditions.

Example 19. Consider again Example 18 and the corresponding knowledge base B. Say we desire to enforce (via structured enforcement) argument a_1 to be part of an admissible set, and the given AF (current state) is $F_1 = (A, R)$ with $A = \{a_1, a_2\}$ and $R = \{(a_2, a_1)\}$. In this AF, we have $adm(F_1) = \{\emptyset, \{a_2\}\}$. Since we have to include both a_1 and a_2, achieving a_1 being in an admissible set requires addition of a_3 that defeats a_2 and defends a_1. Note that simple addition of a_3 is not allowed, since $a_4 \in args(kb(\{a_1, a_2, a_3\}))$, and, thus, we have to satisfy the constraint $(a_1 \in A^* \wedge a_2 \in A^* \wedge a_3 \in A^*) \rightarrow a_4 \in A^*$. One solution is F_3, shown in Fig. 7, where we have added both a_3 and a_4, corresponding to addition of c to $B' = \{a, b\}$ (the base for F_1) in order to get $B^* = \{a, b, c\}$. See also Fig. 7, which shows all AFs that satisfy the structural constraints. For instance, F_4 also has an admissible set containing a, and is another solution to the instance for the structured enforcement operator (same number of attacks added).

Next we show that the structured enforcement operator behaves "well" regarding the given knowledge base. That is, if B is the full knowledge base and F' is the current AF, then (i) no elements outside the knowledge base are invented, (ii) a solution AF F^* contains all arguments that can be constructed when considering its base, and (iii) no attack was omitted.

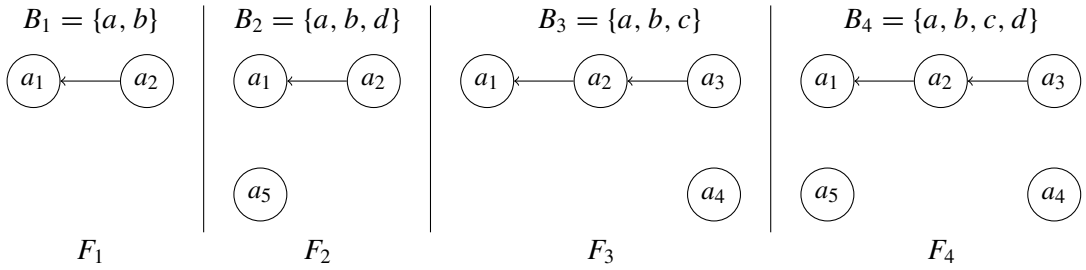

Fig. 7. AFs associated to parts of the knowledge base that contain a and b (Examples 18 and 19). For each AF F_x ($x \in \{1, \ldots, 4\}$) the corresponding knowledge base B_x is shown above. It holds that $kb(A_x) = B_x$ with A_x the argument set of F_x.

Proposition 4. *Let B be a knowledge base, $F' = (args(B'), att(B'))$ an AF for some $B' \subseteq B$, $S \subseteq args(B')$, and σ be a semantics. If $F^* = (A^*, R^*)$ non-strictly enforces S under σ for the structured enforcement operator for F', then it holds that*

- $kb(A^*) \subseteq B$,
- $args(kb(A^*)) = A^*$, *and*
- *if* $a, b \in A^*$ *holds we have* $(a, b) \in att(B)$ *iff* $(a, b) \in R^*$.

Proof. Recall that for any B we assume that $kb(\arg(B)) \subseteq B$ holds, and that all functions *args*, *att*, and *kb* are $\subseteq$-monotone. For the first item, since $args(B') \subseteq A^* \subseteq args(B)$ (by constraint (1)), it follows that $kb(A^*) \subseteq kb(args(B)) \subseteq B$. By constraint (3) we get that for any argument $b \in args(kb(A^*))$ the implication $\bigwedge_{a^* \in A^*} a^* \in A^* \rightarrow b \in A^*$ must be satisfied. The antecedent of that implication is trivially satisfied. This implies that $b \in A^*$ for any $b \in args(kb(A^*))$. Equality of item 2 then holds because we assume $A^* \subseteq args(kb(A^*)$. Assume that $a, b \in A^*$ holds. If $(a, b) \in att(B)$ holds, then by constraint (4) we get $(a, b) \in R^*$. If $(a, b) \in R^*$ holds, then $(a, b) \in att(B)$ follows due to constraints (1) and (2). □

We close this section with some remarks on the structured enforcement operator. First, we note that there are cases when an AF corresponding to the whole knowledge base does not achieve that a desired set of arguments is part of an extension, but a subframework might. That is, when interpreting the structured enforcement as "disclosing" information, in form of AF expansion for part of a knowledge base, an agent may withhold information in its own knowledge base, but still achieve the enforcement goal. We exemplify this behavior in a simple example.

Example 20. Assume a knowledge base B with the corresponding AF $F = (args(B), att(B))$ with $args(B) = \{a, b, c, d\}$ and $att(B) = \{(d, c), (c, b), (b, a)\}$, i.e., the AF is a simple chain of attacks. Say the current AF F_1 consists only of arguments a and b, and the attack (b, a), say via base $B_1 \subset B$. It holds that a is not part of an admissible set for either F or F_1, i.e., is not acceptable, under admissible semantics, when considering the current state and the full knowledge of base B. However, say, for B', with $B_1 \subset B' \subset B$ we have $F' = (args(B'), att(B'))$ with $args(B') = \{a, b, c\}$. In this AF F' there is an admissible set containing a, namely $\{a, c\}$, with c defending a against b. Since d is not part of F', the attack from d to c is neglected.

Regarding the optimization statement, we have chosen to use the same optimization statement as for the (standard) AF enforcement operator, i.e., the number of modified attacks has to be minimum. Alternatives to that can also be considered, e.g., minimizing the number of added arguments, or the size of the underlying knowledge base, also possibly with weights attached to elements in the base. The latter optimization can be interpreted as minimizing what parts of a knowledge base one has to "expose" (if the setting is a multi-agent setting, for instance), in order to be equipped with the right arguments to utter in a debate and achieve an enforcement goal.

Further, constraint (3) is arguably not well suited in all cases, particularly when considering computational properties (there can be exponentially many such constraints). We discuss computational aspects in Section 7, including a more efficient representation of this constraint.

As a final remark, when the *att* function is not $\subseteq$-monotone, a constraint of type (4) is not adequate anymore, since an attack (a, b) has to be present iff this attack is prescribed by $att(B^*)$ for the knowledge base underlying the solution AF F^*. Yet, as is the case with preferences in structured argumentation (see also the discussion at the beginning of this section), it does not hold that $B \subseteq B'$ implies $att(B) \subseteq$

$att(B')$ if a preference not present in B, but in B', prevents an attack. On a meta-level, one can define an associated constraint: if $(a, b) \in att(kb(A^*))$ then $(a, b) \in R^*$. This constraint is not directly defined on arguments and attacks, but on the underlying knowledge base.

7. Computational aspects

In this section we show complexity results for the considered constraints from Section 4, the support enforcement operator on ADFs from Section 5, and the structured enforcement operator from Section 6.

We make use of standard complexity classes for decision problems, which are problems that are answered either positively or negatively. Concretely, we use the classes P, NP, and $coNP$. The first class contains all decision problems that can be decided in deterministic polynomial time and NP contains all decision problems solvable in non-deterministic polynomial time. The third class, $coNP$, contains all complementary problems to problems in NP (i.e., the answers are switched).

7.1. Complexity of satisfaction of the structural constraints

In this section we investigate the feasibility of the constraints. More concretely, we 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 know 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. In Fig. 8, we see a high-level view of such an algorithm: after parsing the input, a framework, e.g., an AF or ADF, is constructed. After that the constructed framework is verified, i.e., checked whether all semantical, structural, and syntactical constraints are met. If the framework is deemed sufficient, then it is returned in the last step. If we assume that the framework construction is done non-deterministically, then if all constraints can be checked in polynomial time, we arrive at an "*NP*-algorithm".

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 [44,102]. That is, for these kind of operators augmentation with any of the polynomial-time decidable constraints comes at no additional cost complexity-wise, in case the algorithm follows the workflow shown in Fig. 8. In Section 8, we consider also experimentally effects on the runtime.

A further benefit of having polynomial-time decidable constraints is that they can, usually, be incorporated more easily in implementations that make direct use of constraint languages, such as Boolean

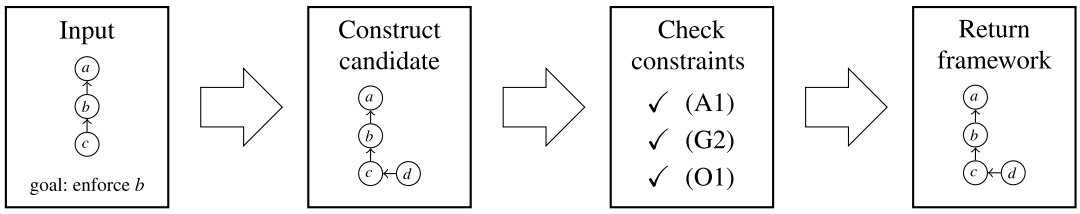

Fig. 8. High-level view on dynamic algorithms.

logic. In fact, many modern implementations of AF reasoning (dynamic or static) make use of solvers on Boolean logic, or variants thereof [37,38].

We begin analyzing the basic constraints, where polynomial-time decidability is immediate. For constraints of type (L4), the following results holds when the graph class is verifiable in polynomial time. We consider here directed acyclic graphs and bipartite graphs, as examples.

Proposition 5. *Verifying whether a given ADF satisfies constraints of type*

- *(G1), (G2),*
- *(A1),*
- *(L1), (L4),*
- *(C1), or (C3)*

is decidable in polynomial time.

Proof. Complexity of these constraints follows directly: checking constraints (G1), (G2), (A1), (L1) amount to check membership in a given set, checking constraint (L4) requires verifying whether the graph structure of the ADF is acyclic or bipartite (both can be decided in polynomial time), checking constraint (C1) amounts to evaluating a Boolean formula under a two-valued interpretation, and checking constraint (C3) requires checking equal syntax of two Boolean formulas. □

We proceed to more sophisticated constraints. By (C2) we refer to cases where we ask for satisfiability (refutability) and by (C2)′ to the cases asking for tautology (unsatisfiability).

Proposition 6. *Verifying whether a given ADF satisfies*

- *a constraint of type (C2) is NP-complete,*
- *and a constraint of type (L2), (L3), (C2)′, (C4), or (Char) is coNP-complete.*

Proof. We show each constraint separately.

- (C2): for membership note that checking whether a (partially evaluated) formula is satisfiable/refutable is in *NP*; Hardness follows from considering $\varphi[I_{\mathbf{u}}] = \varphi$ by a reduction from checking satisfiability (refutability) of a Boolean formula φ.
- (L2) and (L3): follows from [61,99] (checking whether a link is attacking/supporting is *coNP*-complete);
- (C2)′: analogously as for (C2), but for tautological or unsatisfiable formulas;
- (C4): immediate, since checking equivalence of (partially evaluated) formulas is an archetypical *coNP*-complete problem;
- (Char): let $D^* = (A^*, L^*, C^*)$, $\varphi_a, \varphi_b \in C^*$, and $v, v' \in \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ be an arbitrary instance of this problem. Consider the complementary problem, i.e., checking whether there is an I s.t. $\Gamma_D(I)(a) = v$ and $\Gamma_D(I)(b) \neq v'$. We look at the sub cases individually for the truth values, and argue that in each sub case a constant number of witnesses (interpretations) are sufficient to witness that the instance is a yes instance, and that they can be checked in polynomial time (implying that the complementary problem is in *coNP*).
  - $v \in \{\mathbf{t}, \mathbf{f}\}$ and $v' \in \{\mathbf{t}, \mathbf{f}\}$: consider the case that both are equal to $\mathbf{t}$ (analogous for other combinations of $\mathbf{t}$ and $\mathbf{f}$). Assume that there is an I s.t. $\Gamma_D(I)(a) = \mathbf{t}$ and $\Gamma_D(I)(b) \neq \mathbf{t}$. Then, for all

completions J of I we have $J \models \varphi_a$. Further, there is a completion J' of I s.t. $J' \not\models \varphi_b$ (since $\varphi_b[I]$ is not tautological). Then J' is a witness for $\Gamma_D(J')(a) = \mathbf{t}$ and $\Gamma_D(J')(b) \neq \mathbf{t}$, since $I \leqslant_i J'$. Concretely, J' is two-valued and $J' \models \varphi_a$ and $J' \not\models \varphi_b$;

* $v \in \{\mathbf{t}, \mathbf{f}\}$ and $v' = \mathbf{u}$: assume that there is an I s.t. $\Gamma_D(I)(a) = \mathbf{t}$ and $\Gamma_D(I)(b) \neq \mathbf{u}$. Then, $\varphi_b[I]$ is either tautological or unsatisfiable. Any completion J of I can be used as a witness for $\Gamma_D(I)(a) = \mathbf{t}$ and $\Gamma_D(I)(b) \neq \mathbf{u}$;
* $v = \mathbf{u}$ and $v' \in \{\mathbf{t}, \mathbf{f}\}$: assume that there is an I s.t. $\Gamma_D(I)(a) = \mathbf{u}$ and $\Gamma_D(I)(b) \neq \mathbf{t}$. Then this is witnessed by three completions of I: one that satisfies φ_a one that refutes φ_a and one that refutes φ_b.
* $v = \mathbf{u}$ and $v' = \mathbf{u}$: assume that there is an I s.t. $\Gamma_D(I)(a) = \mathbf{u}$ and $\Gamma_D(I)(b) \neq \mathbf{u}$. Then, $\varphi_b[I]$ is either tautological or unsatisfiable. There are two witnesses for $\Gamma_D(I)(a) = \mathbf{u}$: J and J', with the former satisfying φ_a and the latter refuting φ_a. There exist such witnesses that differ only in assignment to one variable (otherwise the formula $\varphi_a[I]$ would be tautological or unsatisfiable). Now take I' instead of I that assigns all variables as J and J', except where they differ. We claim that I' is also a witness, i.e., that $\Gamma_D(I')(a) = \mathbf{u}$ and $\Gamma_D(I')(b) \neq \mathbf{u}$ holds. It holds that there are still two witnesses for the first statement (via J and J'), and $\varphi_b[I']$ is also tautological or unsatisfiable (one just needs to check two completions of I' for φ_b).

Thus, one can always find a constant number of interpretations that witness the property. This means the problem is in *NP*, and the original problem for (Char) in *coNP*.
For hardness: consider the problem of checking whether a formula φ is tautological. Then $\Gamma_D(I)(a) = \mathbf{t}$ and $\Gamma_D(I)(b) \neq \mathbf{t}$ checks this property if $\varphi_a = \varphi$, $\varphi_b = \bot$, and $I = I_{\mathbf{u}}$. □

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 7. *Verifying whether a BADF with known link types satisfies*

- *a constraint of type (L2), (L3), or (C2) is decidable in P, and*
- *a constraint of type (C4) is coNP-complete.*

Proof. For (L2) and (L3) the claim is immediate (the link types are given). For (C2), the claim follows directly from [99]. For (C4), hardness follows from taking $\varphi_s = \bot \wedge \psi$ for some ψ containing all variables from a given φ; then checking whether $I_{\mathbf{u}}(\varphi_s) \equiv \varphi$ directly corresponds to checking whether φ is unsatisfiable. □

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 8. *Let k be a constant integer. Satisfaction of any constraint defined in Section 4, except (O1) and (O2), for a given ADF where each argument has at most k parents, is decidable in polynomial time.*

Proof. Membership follows, since under the restrictions it holds that one can construct the whole truth table in polynomial time (since we have a bounded number of variables in the conditions). Then one can check each property in polynomial time. □

7.2. Complexity of support enforcement under admissible semantics

We now show that the novel enforcement operator defined in Section 5 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 such that its cost is at most a given integer (which is a standard way to investigate complexity of optimization problems). As a further condition we require the number of parents and A' to be bounded by a constant.

Proposition 9. *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.*

Proof. Membership follows from constructing the truth table for each argument's acceptance condition, guessing new links from the expanded arguments, and guessing new entries in expanded truth tables. That is, for an argument a, construct the whole truth table for the original acceptance condition φ_a, which is exponential in the number of variables of a. If a new link is guessed as an incoming link to a, expand the truth table by the new incoming links as new variables. For each new entry in the expanded truth table, assign the guessed truth value. Since both the number of original parents of a and the expanded arguments are bounded by a constant, say k, the overall size of the expanded truth table is bounded from above by $2^{2 \cdot k}$.

Finally, non-deterministically construct a three-valued interpretation I and check whether I is admissible in the expanded ADF, and whether the given interpretation J is less informative, i.e., that $J \leqslant_i I$. Checking admissibility can be done in polynomial time if the truth tables are computed. To see this consider Definition 5. Checking whether $\varphi_a[I]$ is tautological or unsatisfiable amounts to consulting rows in the truth table.

For hardness, we show hardness holds already for AFs. Concretely, we adapt an existing proof for checking whether an argument is part of an admissible set of a given AF, which is an *NP*-complete problem. Let $\varphi = c_1 \wedge \cdots \wedge c_n$ be a Boolean formula in conjunctive normal form with at most three literals per clause over vocabulary X and clauses $C = \{c_1, \ldots, c_n\}$. Construct AF $F = (A, R)$ with

$$
\begin{aligned}
A &= X \cup \{\overline{x} \mid x \in X\} \cup C \cup \left\{c' \mid c \in C\right\} \\
R &= \left\{(x, \overline{x}), (\overline{x}, x) \mid x \in X\right\} \\
&\quad \cup \left\{(x, c) \mid c \in C, x \in c\right\} \\
&\quad \cup \left\{(\overline{x}, c) \mid c \in C, \neg x \in c\right\} \\
&\quad \cup \left\{(c, c' \mid c \in C\right\}
\end{aligned}
$$

Note that each argument has at most three parents: arguments x, $\overline{x}$, and c' have exactly one parent, and since each clause $c = l_1 \vee l_2 \vee l_3$ has at most three literals per assumption, also each c has at most three parents.

We claim that $E = \{c' \mid c \in C\}$ is part of an admissible set of F iff φ is satisfiable. It holds that

$$
\begin{aligned}
&\varphi \text{ is satisfiable} \\
&\quad \text{iff } \exists I, I \models \varphi
\end{aligned}
$$

iff $\exists I, I \models c\ \forall c \in C$

iff $\exists I$ and there is an $l \in c$ s.t. $I(l) = \mathbf{t}\ \forall c \in C$

iff $\exists S \subseteq X \cup \{\overline{x} \mid x \in X\}$ s.t. $\nexists(y, y') \in R$ for each $\{y, y'\} \subseteq S$ and $\exists z \in S$ with $(z, c) \in R\ \forall c \in C$

iff $\exists E' \in adm(F)$ s.t. $E \subseteq E'$.

We complete the hardness proof by φ is satisfiable iff one can enforce, for the support enforcement operator, E with 0 changes and arguments to expand $\{d\}$ (a dummy argument). □

We note that any constraint defined in Section 4, except for optimization statements, can be added to the support enforcement operator, with the mentioned restrictions, without increased complexity. On the other hand, our proofs for these results exhibit exponentiality in running times w.r.t. the number of parents and expanded arguments. The same observation holds for our implementation presented in Section 8.1. Coping with large values for these parameters requires further work.

7.3. Complexity of structured enforcement under admissible semantics

Before looking at the complexity of the structured enforcement operator, we first consider the sizes of the knowledge base B, the "full" AF $F = (args(B), att(B))$, and the number of constraints K described in Section 6.

Regarding the AF, we remark that many current approaches to AF instantiation in structured argumentation may produce large AFs, i.e., AFs whose number of arguments is not polynomially bounded by the size of the knowledge base. Studying approaches to more efficient AF generation is a current topic of research [76,103]. In this paper we make no concrete assumption on the size, but we do assume that both the knowledge base B and the full AF $F = (args(B), att(B))$ are given, and, further, based on these two given structures that one can in polynomial time check whether an argument (attack) is part of the function $args(B)$ or $att(B)$, and similarly for kb. We note that, if the knowledge base and the full instantiated AF are given, then poly-time decidable functions $args$, att, and kb are plausible: this can amount to checking already instantiated arguments and attacks. Nevertheless, a concrete algorithm's performance for structured enforcement hinges on the size metrics of the partial AF given as the current state and the full AF, since, in the worst case, all arguments and attacks have to be added to achieve an enforcement goal.

The structural constraints we utilized to define the structured enforcement operator are in number polynomial to the size of the AF F, except for (3). However, these constraints can be represented differently. Let $F^* = (A^*, R^*)$ be a candidate AF to be checked. Define the following constraint:

$$a \in args(kb(A^*)) \rightarrow a \in A^* \quad \text{for each } a \in args(B).$$

This constraint does not fall into the constraints we have defined in Section 4, since $a \in args(kb(A^*))$ does not refer directly to components of an AF/ADF, however representing the set of constraints of (3) in this way saves space. Concretely, there are at most $|A|$ many such constraints. Similarly as before, these constraints can be checked solely based on the given input: the base B and a candidate AF F^*.

We now investigate the complexity of the structured enforcement operator that is given as input a knowledge base B, represented as a set, an initial AF $F_0 = (args(B_0), att(B_0))$, corresponding to some

$B_0 \subseteq B$, the full AF $F = (args(B), att(B))$, and a set of arguments $S \subseteq args(B_0)$. For the semantics, we again consider the admissible semantics. We again consider the optimization variant and its decision problem, i.e., we ask whether enforcement is possible with at most k changes to the attacks.

Proposition 10. *The structured enforcement operator under admissible semantics is NP-complete, if functions args, att, and kb can be computed in polynomial time.*

Proof. For membership, consider a non-deterministic guess of an expanded AF. Checking each constraint can be done, under the stated assumptions, in polynomial time. Hardness, as before for support enforcement, follows from checking whether an AF without any changes has an admissible set containing a particular argument (this problem is *NP*-complete). For a given arbitrary AF, construct a knowledge base B such that its instantiated AF is exactly the given AF. □

8. Empirical evaluation of an answer set programming approach to support enforcement

In this section we give details to an answer set programming implementation of support enforcement (see Section 5), and an experimental evaluation of the resulting prototype.

8.1. Implementing support enforcement in ASP for admissible semantics

We have implemented support enforcement in answer set programming (ASP), based on goDiamond [62,63,98] v0.6.6, a solver for ADF reasoning tasks via ASP. In light of Proposition 3, we fixed, in our implementation, the set of expanded arguments to be a singleton set (since an enforcement using several expanded arguments can be straightforwardly transformed to using only one). We have applied all constraints we have defined Section 5, i.e., we have implemented the operation we call support enforcement.

High-level description. We begin with a high-level view on the operation we implemented, and a high-level view on how we implemented this operation. We have implemented support enforcement, as defined in Section 5, including all constraints defined for that operation, yet allowing only for one expanded argument. That is, our implementation takes as input an ADF and a three-valued interpretation (with the format specified below). The output is a modified ADF, including an expanded argument e, such that there is an admissible interpretation that is more informative than the given one (thereby enforcing the given interpretation). Further, only the following modifications are permitted: new links only from the expanded argument onto original arguments, all new links supporting, for each original argument a with original acceptance condition φ_a it holds that the modified acceptance condition satisfies $\varphi'_a[I_{\mathbf{u}}|^e_{\mathbf{f}}] \equiv \varphi_a$ (when the expanded argument is false the modified acceptance condition is equivalent to the original one). Finally, $\varphi_e \equiv \top$.

Our implementation approach, from a high-level viewpoint, works as follows. Let the input be the given ADF D, a three-valued interpretation I, and expanded argument e. Assume that the existing arguments are $A = \{a_1, \ldots, a_n\}$. We non-deterministically guess whether a new link is to be added or not, i.e., for each a_i whether to add (e, a_i) or not (other links are not allowed in strong expansions). In case no link was added onto a_i, that argument's acceptance condition φ_{a_i} remains unmodified. Otherwise, in case (e, a_i) was added, condition φ_{a_i} is modified. Due to the constraints, we can infer that the new condition can be represented by $(\varphi_{a_i} \wedge \neg e) \vee (\varphi'_{a_i} \wedge e)$, i.e., whenever e is assigned false no modification must have been taken place, otherwise the condition can be modified (φ'_{a_i} may not be logically

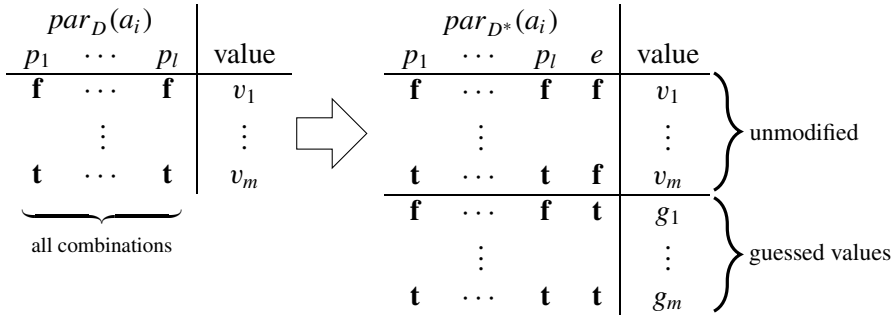

Fig. 9. Constructing a truth table with guessed entries.

equivalent to φ_{a_i}). In order to find φ'_{a_i}, we expand the formula to a full truth table. Say the parents of a_i are $par_D(a_i) = \{p_1, \ldots, p_l\}$. An illustration of the truth table is shown in Fig. 9. For each truth value combination of the parents we have an entry for a resulting truth value. For instance, under the interpretation assigning all arguments to false, φ_{a_i} evaluates to $v_1 \in \{\mathbf{t}, \mathbf{f}\}$. We construct a modified truth table (corresponding to φ'_{a_i}) by adding a new parent e, copying values v_1 to v_m from the original truth table to the cases where e is false, and assigning guessed entries $g_1, \ldots, g_m$ to the remaining rows. Since the new link must be supporting, we ensure that, for all i with $1 \leqslant i \leqslant m$, we have $v_i = \mathbf{t}$ implies $g_i = \mathbf{t}$ (otherwise the link would not be supporting). Finally, under the modified acceptance condition, we non-deterministically construct a three-valued interpretation and verify that there is an admissible interpretation J such that $I \leqslant_i J$ (i.e., the enforcement is successful).

ASP background. We recall briefly ASP background [29,71,88]. We fix a countable set U of *constants*. An atom is an expression $p(t_1, \ldots, t_n)$, where p is a predicate of arity $n \geqslant 0$ and each term t_i is either a variable or an element from U. An atom is *ground* if it is free of variables. BU denotes the set of all ground atoms over U. A rule r is of the form

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

with $m \geqslant k \geqslant 0$, where $a, b_1, \ldots, b_m$ are atoms, and "*not*" stands for default negation. The head of r is the set $head(r) = \{a\}$ and the body of r is $body(r) = \{b_1, \ldots, b_k, \mathit{not}\ b_{k+1}, \ldots, \mathit{not}\ b_m\}$. Furthermore, $body^+(r) = \{b_1, \ldots, b_k\}$ and $body^-(r) = \{b_{k+1}, \ldots, b_m\}$. A rule r is ground if r does not contain variables. A program is a finite set of rules. If each rule in a program is ground, we call the program ground.

For any program π, let UP be the set of all constants appearing in π. Define GP as the set of rules $r\sigma$ obtained by applying, to each rule $r \in \pi$, all possible substitutions σ from the variables in r to elements of UP. An interpretation $I \subseteq BU$ satisfies a ground rule r iff $head(r) \cap I \neq \emptyset$ whenever $body^+(r) \subseteq I$ and $body^-(r) \cap I = \emptyset$. I satisfies a ground program π, if each $r \in \pi$ is satisfied by I. A non-ground rule r (resp., a program π) is satisfied by an interpretation I iff I satisfies all groundings of r (resp., GP). An interpretation $I \subseteq BU$ is an answer set of π if it is a subset-minimal set satisfying the Gelfond–Lifschitz reduct $\pi^I = \{head(r) \leftarrow body^+(r) \mid I \cap body^-(r) = \emptyset, r \in GP\}$.

In this work we also consider optimization programs (following definitions from [33]), in particular programs with weak constraints of the form

$$\leftsquigarrow b_1, \ldots, b_n.[w, t_1, \ldots, t_m],$$

with each b_i an atom, each t_j a term, and w an integer. Towards optimal answer sets we define $weak(\pi, I) = \{(w, t_1, \ldots, t_m) \mid \leftsquigarrow b_1, \ldots, b_n.[w, t_1, \ldots, t_m] \in GP \text{ and } \{b_1, \ldots, b_n\} \subseteq I\}$, for an answer set I. In words, in $weak(\pi, I)$ we collect all weak constraints satisfied by I (terms t_i can be used to distinguish different sources with the same weight in the set $weak(\pi, I)$). The cost of I is then $\sum_{(w,t_1,\ldots,t_m)\in weak(\pi,I)} w$, i.e., the sum of weights of each satisfied weak constraint. An answer set I is optimal if there is no answer set J that has strictly lower cost than I.

Notation 1. By convention, constants in ASP are written with a beginning lower case letter and variables are written with a beginning upper case letter. That is, "a" is a constant and "A" is a variable. In the subsequent encoding, we denote by A variables intended for arguments, by V we denote variables intended for valuations (interpretations in the terminology of the ASP modules we make use of), and F for variables that refer to identifiers for (Boolean) formulas.

Implementing support enforcement in diamond. In (go)Diamond, an ADF is represented via ASP facts, as follows. Let $D = (A, L, C)$ be an ADF, its representation as ASP facts is defined as

$$\pi_{ADF}(D) = \big\{\mathbf{s}(a). \mid a \in A\big\} \cup \big\{\mathbf{ac}\big(a, t(\varphi_a)\big). \mid \varphi_a \in C\big\}\},$$

with $t(\varphi)$ a straightforward prefix notation of φ. For instance, $\varphi = a \wedge (b \vee \neg c)$ is translated to $t(\varphi) = and(a, or(b, neg(c)))$. A three-valued interpretation I (on A) to be enforced is represented by

$$\pi_{enf}(I) = \big\{\mathbf{enf}(a, v). \mid I(a) = v \in \{\mathbf{t}, \mathbf{f}\}\big\}.$$

That is, we represent all arguments enforced to be true or false as ASP facts (arguments assigned undecided are not constrained in an enforcement).

In (go)Diamond, several derived predicates are then provided, in particular for several ADF semantics. We utilize the encodings for admissible semantics. We only recall predicates relevant for our approach, for details we refer to the original references and the code. In (go)Diamond, for each argument a, the acceptance condition φ_a is expanded to a truth table. Each entry in the truth table is assigned a number, represented by $\mathbf{n\_v}(A, V)$, which indicates that, for argument A, the two-valued interpretation (truth table entry) with number V. For instance, if φ_a has 3 variables, there are $2^3 = 8$ such entries. With the auxiliary predicate $\mathbf{true}(A, V, F)$ it is derived whether entry V evaluates (sub)formula F to be true, or not, for argument A (i.e., F is a subformula of φ_a). Building upon **true**, the two predicates $\mathbf{model}(A, V)$ and $\mathbf{nomodel}(A, V)$ state that V is a model or not a model of φ_a. Finally, regarding predicates of importance for us, there are $\mathbf{t}(A)$, $\mathbf{f}(A)$, and $\mathbf{u}(A)$, stating that an argument A is assigned **t**, **f**, or **u**, respectively.

We adapted goDiamond's encoding, as follows, see also Listing 1. The first two lines represent a (standard) guess in ASP whether a link is to be added onto argument A (note that the origin is clear: the link comes from a distinguished expanded argument). Next, as illustrated in Fig. 9, we construct an expanded truth table. Since we are "copying" the entries when the expanded argument is false, and also can assume (see Proposition 3) that the new argument is assigned true, we only need to represent the guessed values $(g_1, \ldots, g_m)$ for a number of interpretations equal to the number of the original truth table. In the ASP encoding, we represent this via guessing either $\mathbf{model}(A, V)$ or $\mathbf{nomodel}(A, V)$ for each $\mathbf{n\_v}(A, V)$, whenever a link was actually added onto A. Since the new link has to be supporting,

```
% guess new links
newlink(A) ← not nonewlink(A), s(A).
nonewlink(A) ← not newlink(A), s(A).
% guess entries for truth table
model(A, V) ← n_v(A, V), newlink(A), not nomodel(A,V).
nomodel(A, V) ← n_v(A, V), newlink(A), not model(A,V).
% ensure that guessed entries correspond to support
← ac(A, F), true(A, V, F), newlink(A), nomodel(A,V).
% ensure that enforcement goals are met
← s(A), enf(A,f), t(A).
← s(A), enf(A,f), u(A).
← s(A), enf(A,t), f(A).
← s(A), enf(A,t), u(A).
% weak constraint for optimization of link addition
⇜ newlink(S). [1,A]
% derive difference to original acceptance condition
changedmodel(A,V) ← ac(A, F), true(A, V, F), nomodel(A,V).
changedmodel(A,V) ← ac(A, F), not true(A, V, F), model(A,V).
```

Listing 1. Partial ASP encoding for support enforcement

we apply the next constraint that ensures that when originally we evaluated to true (**true** holds) we are not allowed to guess **nomodel**. The next four constraints specify that the enforcement goals are to be met: e.g., if A is an argument, it was assigned true ($\mathbf{t}(A)$), then the enforcement goal cannot be false ($\mathbf{enf}(A, \mathbf{f})$). The weak constraint specifies unit weight (cost) to each added link. Finally, in order to be able to compute the modified acceptance condition, we derive each change of entries in a truth table. That is, $\mathbf{changedmodel}(a, i)$ indicates that $v_i \neq g_i$. In this way, the full truth table can be inferred for the modified $\varphi_a^* = (\varphi_a \wedge \neg e) \vee (\varphi_a' \wedge e)$, i.e., φ_a' is fully specified in this way.

The full ASP encodings are available at

https://www.dbai.tuwien.ac.at/research/project/embarg/supp-enf/

which, derived from (go)Diamond, are distributed under the GPL license.

Example 21. We show an example of support enforcement, which is taken from a part, and simplified form, of an instance ADF that has 216 arguments that was used in the experiments shown in the next section. For our illustration, nine arguments are relevant, which have the following unmodified acceptance conditions:

$$\varphi_{a_1} = \neg a_2 \vee a_3$$

$$\varphi_{a_2} = \neg a_1$$

$$\varphi_{a_3} = \neg a_4 \wedge \neg a_1$$

$$\varphi_{a_4} = a_5 \vee a_3$$

$$\varphi_{a_5} = \neg a_6$$

$$\varphi_{a_6} = \psi$$

$$\varphi_{a_7} = (a_8 \leftrightarrow a_2)$$
$$\varphi_{a_8} = \top$$

with ψ a formula with at least one variable (for our example the exact form of this condition is not relevant). Say we want to enforce a_7 to be true (other arguments are not constrained). When using the ASP encoding above (together with the Diamond encodings), we extracted a solution as follows. Consider the three-valued interpretation $I = \{a_1 \mapsto \mathbf{f}, a_2 \mapsto \mathbf{t}, a_3 \mapsto \mathbf{f}, a_4 \mapsto \mathbf{t}, a_5 \mapsto \mathbf{t}, a_6 \mapsto \mathbf{u}, a_7 \mapsto \mathbf{t}, a_8 \mapsto \mathbf{t}\}$. The interpretation I is not admissible in the unmodified ADF: $\varphi_{a_5}[I] = \varphi_{a_5} = \neg a_6$, but $I(a_5) = \mathbf{t}$. We now expand the ADF with argument e and add link (e, a_5) and adapt the acceptance condition to $\varphi^*_{a_5} = (\neg a_6 \wedge \neg e) \vee (\top \wedge e)$. This corresponds to having **changedmodel**$(a_5, 1)$ in an answer set: there are two entries in the truth table of φ_{a_5}, with the one identified with 0 being the one where a_6 is assigned false and the one with ID 1 the one where a_6 is true. Then, **changedmodel**$(a_5, 1)$ indicates that the new acceptance condition, with e true, has still a model where a_6 is false, but the previous non-model where a_6 is true shall become a model. Then, under $\varphi^*_{a_5}$, it holds that I is admissible, when expanded with $e \mapsto \mathbf{t}$, and a_7 is true in I, thus, a_7 is enforced to be true in the expansion. In other words, support enforcement identified that a link from e to a_5 that states that acceptance of e is sufficient to accept a_5 leads to acceptance of a_7, since this leads to a chain of changes of acceptance: due to acceptance of a_5, we can accept a_4, which, in turn, leads to rejection of a_3 if a_1 is likewise rejected. The argument a_1 can be rejected if a_2 is accepted and a_3 is rejected. By accepting a_2 we arrive at the situation that both a_2 and a_8 are accepted, implying that $a_8 \leftrightarrow a_2$ evaluates to true, and a_7 can be accepted.

8.2. Experiments

We report on an experimental evaluation of the support enforcement operator implemented in ASP.

Instances and experimental setup. We performed experiments using instances from www.dbai.tuwien.ac.at/proj/adf/yadf/, which are ADFs that were generated from the AFs of domains ABA2AF, Planning2AF, and Traffic from ICCMA 2017 [66,67]. The generation procedure is described in [49, Section 5]. Briefly put, the generator transforms an undirected graph (from the AF competition) into cyclic or acyclic ADFs, by inheriting the graph structure (for acyclic ADFs, by a topological ordering, inducing an acyclic directed graph). Acceptance conditions are constructed by considering the parents (as given by the graph generation), and grouping parents into five classes: attack, group-attack, support, group-support, or XOR, which represent different types of generated Boolean subformulas, which are then connected via logical conjunction and disjunction.

We analyzed characteristics of the ADF instances. We list statistics of each domain in Table 3. Overall, the benchmark set contains 600 ADFs, which vary in several statistics. For instance, the maximum number of parents of arguments (i.e., the maximum in-degree of the graph) varies from at most 2 to at most 131. For the calculation of (number of) parents we excluded self-loops. We computed polarity of each ADF, and found that, from the whole set, 48 are bipolar, 547 are not bipolar, and 5 instances could not be checked (note that it is coNP-complete to check whether a link is supporting or attacking [61]). All bipolar ADFs have at most 12 parents for each argument.

For each ADF we created three enforcement requests, i.e., interpretations to enforce, as follows. We selected, at random, a subset of the arguments in the ADF with probability $p \in \{\frac{1}{5}, \frac{1}{10}, \frac{1}{20}\}$ 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 $\frac{1}{5}$ that an argument is false. Thus having more

Table 3

Statistics of ADFs used in the experimentation. For each group of instances we list the total number of ADFs, the average number of arguments in the ADFs, the interval of maximum number of parents in the ADFs (excluding self-loops), the number of cyclic ADFs, and the number of bipolar ADFs

Planning	Number of ADFs	200
	Average number of arguments	202.64
	Interval of maximum number of parents	6–19
	Number of ADFs with cyclic dependency graph	100
	Number of bipolar ADFs	0
Traffic	Number of ADFs	200
	Average number of arguments	111.81
	Interval of maximum number of parents	2–19
	Number of ADFs with cyclic dependency graph	100
	Number of bipolar ADFs	35
ABA2AF	Number of ADFs	200
	Average number of arguments	43.83
	Interval of maximum number of parents	3–131
	Number of ADFs with cyclic dependency graph	100
	Number of bipolar ADFs	13
Overall	Number of ADFs	600
	Average number of arguments	119.427
	Interval of maximum number of parents	2–131
	Number of ADFs with cyclic dependency graph	300
	Number of bipolar ADFs	48

arguments to be enforced to be true, since arguments enforced to be false can lead to a no solution instance in contrast to arguments enforced to be true, which motivates this choice. This was also observed empirically, see below. This resulted in 600 enforcements per domain, and 1800 enforcements overall.

All experiments were run on a machine with two AMD Opteron Processors 6308, 12 × 16 GB RAM, and Debian 8. We used a timeout of 900 seconds on each individual instance (query), i.e., for each pair of ADF and enforcement. Furthermore, we restricted memory consumption (virtual memory) to at most 8 GB. In order to enforce these limits, and to have more logging information, we utilized the runsolver v3.4.0 tool[3] by Olivier Roussel, which was also used in several competitions. For the ASP solver, we used clingo v5.3.0 [70].

Results and interpretations. The results are summarized in Table 4. The rows show the instances grouped by domain, number of queries, number of successful computations (i.e., where clingo terminated with the result within the limits), number of instances where an optima was found, number of instances with optimum cost 0, number of instances where clingo reported unsatisfiability, number of timeouts, and the median runtimes in seconds. In the following, we interpret the results in more detail.

We first discuss successful and not successful runs. A run is deemed successful when clingo returned either with an optimal solution or reported unsatisfiability. In our experiments, we encountered four reasons for clingo not reporting successfully. The first is that the time limit was reached, which occurred rarely: only 8 runs timed out. In its current version, clingo does not support very high integers. Since in the ASP encoding used by us (and by Diamond), a truth table is constructed, and each entry is assigned

[3] http://www.cril.univ-artois.fr/~roussel/

Table 4
Summary of experimental results

Domain	#	#successful	Opt. found	0-cost	Unsat.	Timeout	Median time (sec) of succ. runs
Planning	600	586	211	4	375	8	2.442
Traffic	600	591	341	49	250	0	0.188
ABA2AF	600	267	220	57	47	0	0.612
All	1800	1444	772	110	672	8	0.862

an integer (an ID), we naturally have to deal with large integers. For instance, if the maximum number of parents is n, then an integer 2^n is required by the ASP encoding. We noted that 6 runs failed due to such high integers. The third reason is that the memory limit of 8 GB was reached, which happened for 63 runs. Finally, we encountered 279 runs where clingo could not solve the instance, but stayed with the limits and did not report an issue with large integers. We inspected these runs, and found that in all cases a large portion of memory was used: at least 6 GB, but in most cases almost 8 GB. While, strictly speaking, the memory limit was not reached, we interpret all these instances as failing due to the high memory consumption. Overall, out of 1800 runs, 1444 were successful and 356 were not. We looked at the maximum number of parents of the failed runs, and, except for the timed out runs, all failed runs had to cope with ADFs with a maximum number of parents of at least 15 and up to 131. In contrast, the successful runs had maximum number of parents between 2 and 16. We hypothesize that this is inherent to our method of transforming Boolean formulas by truth table expansion: the method was able to cope with instances up to 16 parents, but not more. Nevertheless, all instances with at most 14 parents (except for the 8 time outs) were solved successfully by clingo.

We move on to running time behavior of the successful runs. Table 4 summarizes that the median of these runs was (quite) low. Since, as observed above, the maximum number of parents seems to be a crucial parameter for difficulty, we detail running times for this parameter in Fig. 10. In the figure, the successfully solved instances are grouped by maximum number of parents (with the sample size in brackets) shown along the horizontal axis. As can be seen, there is a clear increase in running times when increasing the maximum number of parents, as expected. Nevertheless, even for a maximum number of parents of 16, running times were reasonable, e.g., below 200 seconds for most runs. We also looked at running time differences when considering cyclicity. As observed theoretically [77], acyclic ADFs have milder complexity. Within the successful runs, 726 were on acyclic ADFs and 718 were on cyclic ADFs. The cumulative running time (i.e., summing up all running times) were 5391.5 and 18112.7, respectively. While ADFs and enforcement requests are different for acyclic and cyclic instances, this result suggests that the difference in complexity was also observed empirically, in our experiments. Regarding bipolarity, the picture is not clear, since (recall description above) all bipolar ADFs have a low number of maximum number of parents (up to 12). Looking at the corresponding running times in Fig. 10 reveals that those runs were all solved quickly, independent of polarity. Such low running times do not give clear results of divergence of running times on bipolar or non-bipolar ADFs.

We now consider the three parameters for enforcement generation. When looking at those runs with the lowest number of arguments to be enforced to false ($p = \frac{1}{20}$), we have 335 runs where clingo reported an optimal solution and 145 runs which were deemed unsatisfiable. For $p = \frac{1}{10}$, we have 261 optimal solutions and 219 unsatisfiable queries. For $p = \frac{1}{5}$, we encountered 176 optimal solutions and 308 unsatisfiable queries. This is in line with Proposition 2, i.e., that arguments enforced to be true always have a solution, while arguments enforced to be false may not have a solution. Further, we noticed an increase of costs for the three groups: the average costs were 2.8, 4.1, and 5.1, respectively, which

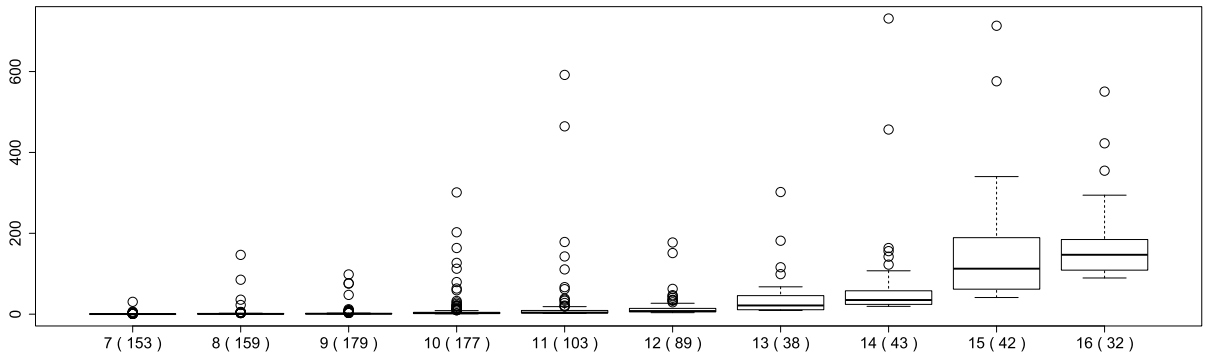

Fig. 10. Whisker plot for all successful runs of support enforcement. In the vertical axis running times (sec) are shown, and in the horizontal axis instances are grouped by maximum number of parents (with size of the group in parentheses). The box shows the lower quartile and upper quartile (the 25th and 75th percentiles), the median in bold, and the lines denote the "whiskers", i.e., these extend from the quartiles to the farthest points up to 1.5 times the interquartile range (range from lower and upper quartile) from upper/lower quartiles. Outliers are denoted as circles. Instances with a maximum number of parents less than seven are omitted (both their minimum and maximum running times are close to zero).

reflects the increasing number of arguments enforced. Regarding costs, in, overall, 110 runs an optimal solution had 0 costs. This indicates that the enforcement request was already met without any change (i.e., when enforcing I to be admissible, there already is an admissible J with $I \leqslant_i J$). Compared to 1444 successful runs, the number of trivial instances (w.r.t. costs) is low.

Summarizing our results, with the ASP approach, we could solve 1444 out of 1800 instances, most of these within (rather) low running times. Non-solvability was mostly due to high memory consumption, which is to be expected as a natural barrier for our approach. Nevertheless, ADFs with at most 16 maximum number of parents were solved regularly.

9. Constraints for further operations

In this section we give a brief overview on further operators that fit into our three-layered approach, and for which structural constraints can be applied. Concretely, we look at revision of AFs, merging (aggregation) of AFs, and synthesis (learning) of AFs. Each of these operations result in a (modified) AF that has to satisfy certain semantical constraints. In addition some of these operators allow for structural constraints. These operators can be augmented with structural constraints we looked at in Section 4. For instance, in any of these operations one can fix certain parts of the AF, or impose implications (e.g. presence of one attack implies further attacks), or add further optimizations.

Revision of argumentation frameworks. Revision of knowledge has a long tradition in AI [1,69,74], and revision of semantics of AFs [42,43,48] and ADFs [79] has been studied recently. From a high-level perspective, these operators revise a given framework and return a modified framework. Many of these revision operators specify the semantics of the modified framework, and some, but not all, give constraints on the structure. In this regard, in our three-layered point of view, these operators specify semantical constraints, in the form of exact semantics a modified AF shall have, and sometimes further constraints such as optimizations. Our structural constraints can support further development of such revision operators by stating how an AF may evolve structurally, in addition to the semantics or existing structural constraints.

Merging of argumentation frameworks. Similar to revision, merging or aggregation of (logical) structures has received interest in the AI community [75]. Aggregation, or merging, of AFs has recently found attention in the argumentation community [26,41]. As an example for a semantical merging operator, consider an operator studied in [47], which, given a tuple of AFs, merges the tuple into an AF, or a set of AFs. For simplicity we look at those operators returning a single AF.[4] Operators from this family specify the semantics of the output (i.e., specify a semantical constraint), and structural constraints that optimize the modifications of attacks. To such operators, similarly as for the revision operators, structural constraints can be added to further constrain the candidate solutions, e.g., to respect more interconnections between arguments and attacks.

Synthesis of argumentation frameworks. Broadly speaking, generation of structures from given information, also sometimes called learning, acquisition, or synthesis, has initiated several research directions within AI [20,24,46,100]. Focusing on the construction of argumentative structures [89,96,97], we recall the synthesis operation from [89], where our structural constraints can be applied. Simply put, the task in AF synthesis is to find an AF such that certain given examples of sets of arguments are (or are not) σ-extensions. An additional optimization criterion specifies that as many such examples as possible are satisfied. In its current form, AF synthesis incorporates structural constraints specifying that certain attacks have to part of the returned AF, or must be omitted. In addition, our structural constraints can specify, e.g., implications that specify that sets of attacks may be added as whole, or not at all.

10. Related work

A recent survey [52] gives an interesting overview over many approaches to dynamics of argumentation, constraints, and changes. In the following, we discuss relations of our work to several related works (including, but not restricting, to content of the survey).

Logical languages for dynamics and constraints. Several works considered constraints (in argumentation dynamics) specified in a logical language, which naturally relate to our work. In [50] control argumentation frameworks were proposed, where one can specify known, unknown, and controllable arguments by an agent, the last category being those arguments the agent can use in a debate. In [51,58] logical languages are proposed and used, in order to specify allowed changes to an argumentation framework. In contrast, we look ADFs (AFs in their works) and at particular families of constraints, consider their use cases, properties, computational aspects, and applied them in two case studies. Some of our constraints are already present in earlier works: limits of arguments and attacks, and mandatory (non-) presence of certain arguments and attacks is discussed, e.g., in [44,89,102]; similarly in enforcement and revision [42,43] optimization of modified attacks is part of the problem definition. We consider more general families of constraints. Constraints on presence of attacks is also, although in a quite different form, part of incorporation of preferences in structured argumentation, e.g., in [84], where a preference relation can make certain attacks not applicable. That is, a preference relation can be seen as a constraint on attacks.

[4]The assumption that the merging operators of [47] return a single AF does not hold in general for all input tuples of AFs; in fact they show that when considering operators satisfying certain properties there is none that always returns a single AF. Nevertheless, our structural constraints can be straightforwardly applied to a tuple (set) of AFs.

Realizability and expressivity of argumentation frameworks. While our work focuses on structural constraints, semantical constraints are likewise important. A recent research direction is to study the signature of AFs, also called realizability [12,57,78]. That is, these studies reveal which sets of arguments can be, e.g., a part of the preferred semantics of an AF. In this way, limits to semantical constraints are given. Such constraints are also useful for incorporation of structural constraints, as seen by Example 10: when restricting an ADF structurally to be part of a particular sub family of ADFs (e.g., bipolar ADFs) the signature of the semantics changes and can directly lead to unsatisfiability of semantical constraints. Further, similar to this work, in [57] complexity results were shown for checking whether a semantical structure is part of the signature. While the realizability problem requires that an AF has exactly a certain semantics, in the synthesis problem [89] this is generalized to allow also for partial semantics specifications. Similarly, insights from AF synthesis can help to understand which constraints (semantical and structural) lead to a set of constraints that is satisfiable.

Dialogues, strategic argumentation, and incomplete AFs. Enforcement under structural constraints, as studied in this work, relates to some approaches in the literature. Incomplete AFs [13–17] are AFs where some attacks or arguments are fixed, while others can be modified. A task is then to see whether a set of arguments is an extension in all completions of the incomplete AFs. Bounds such as these, can be represented by the constraints considered in this work. In strategic argumentation, e.g., in [80–82], one can desire to look for an AF among many choices of AFs that satisfy a certain strategic goal (such as acceptance of an argument when adding arguments/attacks). Enforcement naturally shares this goal. Also, our structural constraints can be utilized to specify which AFs one may choose from. For our structured enforcement operator, dialogues, as studied in [64], have a similar motivation. In their work, a goal is to "reveal" (through a dialogue) parts of a knowledge base, in order to have parts revealed that make up a framework satisfying certain (strategic) goals. Similarly, structured enforcement aims to find an expansion satisfying an enforcement goal that also reflects contents of a knowledge base. Furthermore, different to the three directions, we work with general ADFs, consider several families of constraints, look at complexity of such families, and allow for rather fine-grained constraints (e.g., not allowing all completions). Further, our structured enforcement operator differs from the dialogue approach by working on expansion of AFs that reflect (general) structured formalisms in contrast to engaging in a dialogue that utters parts of a knowledge base, and include optimizations on such AFs.

11. Conclusions

In this article we have proposed to extend current (and future) dynamic operators on frameworks in argumentation in AI with constraints that, e.g., suit conditions "abstracted away" during instantiation, or ensure other desirable properties. More concretely, we proposed a set of constraints to be used for such operators, exemplified several use cases, and properties, of such constraints, investigated in more detail two concrete case studies developing an enforcement operator for ADFs based on supports and an enforcement operator for structured argumentation. The two case studies witness that our constraints can be utilized to extend operators, such as enforcement, in two ways: adapt them to more general abstract frameworks (e.g., ADFs), and adapt them to respect structural information from instantiation. Furthermore, we looked at the computational properties of constraints and the two novel enforcement operators. We implemented support enforcement in ASP, and presented an empirical evaluation of the resulting prototype. Our findings, from the case studies, are that several constraints can be applied to

diverse scenarios, and our prototype implementation shows reasonable performance even when faced with complex optimization problems.

Directions for future work include, in particular, extending work on enforcement to complex contexts. For instance, our case study for a structural enforcement operator can be extended to one of the formalisms available in the field of structured argumentation (such as ABA and ASPIC+). However, such an endeavor requires care when instantiating: a large number of arguments (relations) should be avoided. We think that more efficient ways of instantiation, for which we already have seen early work [76,103], is a fruitful direction, since both dynamic operators we studied here, and static operators in argumentation can benefit from such efficiency gains.

Acknowledgements

We thank the anonymous reviewers of the preliminary conference version [101] and this version of the paper and Thomas Linsbichler for their useful comments. This work has been supported by the Austrian Science Fund (FWF): P30168 and I2854.

References

[1] C.E. Alchourrón, P. Gärdenfors and D. Makinson, On the logic of theory change: Partial meet contraction and revision functions, *Journal of Symbolic Logic* **50**(2) (1985), 510–530. doi:10.2307/2274239.
[2] L. Amgoud and C. Cayrol, A reasoning model based on the production of acceptable arguments, *Annals of Mathematics and Artificial Intelligence* **34**(1–3) (2002), 197–215. doi:10.1023/A:1014490210693.
[3] L. Amgoud, C. Cayrol, M. Lagasquie-Schiex and P. Livet, On bipolarity in argumentation frameworks, *International Journal of Intelligent Systems* **23**(10) (2008), 1062–1093. doi:10.1002/int.20307.
[4] K. Atkinson, P. Baroni, M. Giacomin, A. Hunter, H. Prakken, C. Reed, G.R. Simari, M. Thimm and S. Villata, Towards artificial argumentation, *AI Magazine* **38**(3) (2017), 25–36. doi:10.1609/aimag.v38i3.2704.
[5] P. Baroni, M. Caminada and M. Giacomin, An introduction to argumentation semantics, *Knowledge Engineering Review* **26**(4) (2011), 365–410. doi:10.1017/S0269888911000166.
[6] P. Baroni, F. Cerutti, M. Giacomin and G. Guida, AFRA: Argumentation framework with recursive attacks, *International Journal of Approximate Reasoning* **52**(1) (2011), 19–37. doi:10.1016/j.ijar.2010.05.004.
[7] P. Baroni, D. Gabbay, M. Giacomin and L. van der Torre (eds), *Handbook of Formal Argumentation*, College Publications, 2018.
[8] R. Baumann, Normal and strong expansion equivalence for argumentation frameworks, *Artificial Intelligence* **193** (2012), 18–44. doi:10.1016/j.artint.2012.08.004.
[9] R. Baumann, What does it take to enforce an argument? Minimal change in abstract argumentation, in: *Proc. ECAI*, L. De Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz and P.J.F. Lucas, eds, Frontiers in Artificial Intelligence and Applications, Vol. 242, IOS Press, 2012, pp. 127–132.
[10] R. Baumann and G. Brewka, Expanding argumentation frameworks: Enforcing and monotonicity results, in: *Proc. COMMA*, P. Baroni, F. Cerutti, M. Giacomin and G.R. Simari, eds, Frontiers in Artificial Intelligence and Applications, Vol. 216, IOS Press, 2010, pp. 75–86.
[11] R. Baumann and G. Brewka, Extension removal in abstract argumentation – an axiomatic approach, in: *Proc. AAAI*, P. Van Hentenryck and Z.-H. Zhou, eds, AAAI Press, 2019, pp. 2670–2677.
[12] R. Baumann, W. Dvořák, T. Linsbichler, H. Strass and S. Woltran, Compact argumentation frameworks, in: *Proc. ECAI*, T. Schaub, G. Friedrich and B. O'Sullivan, eds, Frontiers in Artificial Intelligence and Applications, Vol. 263, IOS Press, 2014, pp. 69–74.
[13] D. Baumeister, D. Neugebauer and J. Rothe, Verification in attack-incomplete argumentation frameworks, in: *Proc. ADT*, T. Walsh, ed., Lecture Notes in Computer Science, Vol. 9346, Springer, 2015, pp. 341–358.
[14] D. Baumeister, D. Neugebauer and J. Rothe, Credulous and skeptical acceptance in incomplete argumentation frameworks, in: *Proc. COMMA*, S. Modgil, K. Budzynska and J. Lawrence, eds, Frontiers in Artificial Intelligence and Applications, Vol. 305, IOS Press, 2018, pp. 181–192.
[15] D. Baumeister, D. Neugebauer, J. Rothe and H. Schadrack, Complexity of verification in incomplete argumentation frameworks, in: *Proc. AAAI*, S.A. McIlraith and K.Q. Weinberger, eds, AAAI Press, 2018, pp. 1753–1760.

[16] D. Baumeister, D. Neugebauer, J. Rothe and H. Schadrack, Verification in incomplete argumentation frameworks, *Artificial Intelligence* **264** (2018), 1–26. doi:10.1016/j.artint.2018.08.001.
[17] D. Baumeister, J. Rothe and H. Schadrack, Verification in argument-incomplete argumentation frameworks, in: *Proc. ADT*, T. Walsh, ed., Lecture Notes in Computer Science, Vol. 9346, Springer, 2015, pp. 359–376.
[18] T.J.M. Bench-Capon, Persuasion in practical argument using value-based argumentation frameworks, *Journal of Logic and Computation* **13**(3) (2003), 429–448. doi:10.1093/logcom/13.3.429.
[19] T.J.M. Bench-Capon and P.E. Dunne, Argumentation in artificial intelligence, *Artificial Intelligence* **171**(10–15) (2007), 619–641. doi:10.1016/j.artint.2007.05.001.
[20] F. Bergadano and D. Gunetti, *Inductive Logic Programming – from Machine Learning to Software Engineering*, MIT Press, 1996.
[21] P. Besnard, A.J. García, A. Hunter, S. Modgil, H. Prakken, G.R. Simari and F. Toni, Introduction to structured argumentation, *Argument & Computation* **5**(1) (2014), 1–4. doi:10.1080/19462166.2013.869764.
[22] P. Besnard and A. Hunter, *Elements of Argumentation*, MIT Press, 2008.
[23] P. Besnard and A. Hunter, A review of argumentation based on deductive arguments, in: *Handbook of Formal Argumentation*, P. Baroni, D. Gabbay, M. Giacomin and L. van der Torre, eds, College Publications, 2018, pp. 437–484, Chap. 9.
[24] C. Bessiere, F. Koriche, N. Lazaar and B. O'Sullivan, Constraint acquisition, *Artificial Intelligence* **244** (2017), 315–342. doi:10.1016/j.artint.2015.08.001.
[25] P. Bisquert, C. Cayrol, F. Dupin de Saint-Cyr and M. Lagasquie-Schiex, Enforcement in argumentation is a kind of update, in: *Proc. SUM*, W. Liu, V.S. Subrahmanian and J. Wijsen, eds, Lecture Notes in Computer Science, Vol. 8078, Springer, 2013, pp. 30–43.
[26] G.A. Bodanza, F. Tohmé and M. Auday, Collective argumentation: A survey of aggregation issues around argumentation frameworks, *Argument & Computation* **8**(1) (2017), 1–34. doi:10.3233/AAC-160014.
[27] G. Boella, D.M. Gabbay, L.W.N. van der Torre and S. Villata, Support in abstract argumentation, in: *Proc. COMMA*, P. Baroni, F. Cerutti, M. Giacomin and G.R. Simari, eds, Frontiers in Artificial Intelligence and Applications, Vol. 216, IOS Press, 2010, pp. 111–122.
[28] A. Bondarenko, P.M. Dung, R.A. Kowalski and F. Toni, An abstract, argumentation-theoretic approach to default reasoning, *Artificial Intelligence* **93** (1997), 63–101. doi:10.1016/S0004-3702(97)00015-5.
[29] G. Brewka, T. Eiter and M. Truszczynski, Answer set programming at a glance, *Communications of the ACM* **54**(12) (2011), 92–103. doi:10.1145/2043174.2043195.
[30] G. Brewka, S. Ellmauthaler, H. Strass, J.P. Wallner and S. Woltran, Abstract dialectical frameworks, in: *Handbook of Formal Argumentation*, P. Baroni, D. Gabbay, M. Giacomin and L. van der Torre, eds, College Publications, 2018, pp. 237–285, Chap. 5.
[31] G. Brewka, S. Polberg and S. Woltran, Generalizations of Dung frameworks and their role in formal argumentation, *IEEE Intelligent Systems* **29**(1) (2014), 30–38. doi:10.1109/MIS.2013.122.
[32] G. Brewka and S. Woltran, Abstract dialectical frameworks, in: *Proc. KR*, AAAI Press, 2010, pp. 102–111.
[33] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca and T. Schaub, *ASP-Core-2 Input Language Format*, 2012. https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.0.pdf.
[34] C. Cayrol, F. Dupin de Saint-Cyr and M. Lagasquie-Schiex, Change in abstract argumentation frameworks: Adding an argument, *Journal of Artificial Intelligence Research* **38** (2010), 49–84. doi:10.1613/jair.2965.
[35] C. Cayrol, J. Fandinno, L.F. del Cerro and M. Lagasquie-Schiex, Valid attacks in argumentation frameworks with recursive attacks, in: *Proc. COMMONSENSE*, A.S. Gordon, R. Miller and G. Turán, eds, CEUR Workshop Proceedings, Vol. 2052, CEUR-WS.org, 2017.
[36] C. Cayrol and M.-C. Lagasquie-Schiex, Bipolarity in argumentation graphs: Towards a better understanding, *International Journal of Approximate Reasoning* **54**(7) (2013), 876–899. doi:10.1016/j.ijar.2013.03.001.
[37] F. Cerutti, S.A. Gaggl, M. Thimm and J.P. Wallner, Foundations of implementations for formal argumentation, in: *Handbook of Formal Argumentation*, P. Baroni, D. Gabbay, M. Giacomin and L. van der Torre, eds, College Publications, 2018, pp. 688–767, Chap. 15.
[38] G. Charwat, W. Dvořák, S.A. Gaggl, J.P. Wallner and S. Woltran, Methods for solving reasoning problems in abstract argumentation – a survey, *Artificial Intelligence* **220** (2015), 28–63. doi:10.1016/j.artint.2014.11.008.
[39] A. Cohen, S. Gottifredi, A.J. García and G.R. Simari, A survey of different approaches to support in argumentation systems, *Knowledge Engineering Review* **29**(5) (2014), 513–550. doi:10.1017/S0269888913000325.
[40] A. Cohen, S. Gottifredi, A.J. García and G.R. Simari, An approach to abstract argumentation with recursive attack and support, *Journal of Applied Logic* **13**(4) (2015), 509–533. doi:10.1016/j.jal.2014.12.001.
[41] S. Coste-Marquis, C. Devred, S. Konieczny, M. Lagasquie-Schiex and P. Marquis, On the merging of Dung's argumentation systems, *Artificial Intelligence* **171**(10–15) (2007), 730–753. doi:10.1016/j.artint.2007.04.012.
[42] S. Coste-Marquis, S. Konieczny, J. Mailly and P. Marquis, On the revision of argumentation systems: Minimal change of arguments statuses, in: *Proc. KR*, C. Baral, G.D. Giacomo and T. Eiter, eds, AAAI Press, 2014, pp. 52–61.

[43] S. Coste-Marquis, S. Konieczny, J. Mailly and P. Marquis, A translation-based approach for revision of argumentation frameworks, in: *Proc. JELIA*, E. Fermé and J. Leite, eds, Lecture Notes in Computer Science, Vol. 8761, Springer, 2014, pp. 397–411.
[44] S. Coste-Marquis, S. Konieczny, J. Mailly and P. Marquis, Extension enforcement in abstract argumentation as an optimization problem, in: *Proc. IJCAI*, Q. Yang and M. Wooldridge, eds, AAAI Press, 2015, pp. 2876–2882.
[45] K. Cyras, X. Fan, C. Schulz and F. Toni, Assumption-based argumentation: Disputes, explanations, preferences, in: *Handbook of Formal Argumentation*, P. Baroni, D. Gabbay, M. Giacomin and L. van der Torre, eds, College Publications, 2018, pp. 365–408, Chap. 7.
[46] J. Davis and J. Ramon (eds), *Proc. ILP 2014, Revised Selected Papers*, Lecture Notes in Computer Science, Vol. 9046, Springer, 2015.
[47] J. Delobelle, A. Haret, S. Konieczny, J. Mailly, J. Rossit and S. Woltran, Merging of abstract argumentation frameworks, in: *Proc. KR*, C. Baral, J.P. Delgrande and F. Wolter, eds, AAAI Press, 2016, pp. 33–42.
[48] M. Diller, A. Haret, T. Linsbichler, S. Rümmele and S. Woltran, An extension-based approach to belief revision in abstract argumentation, *International Journal of Approximate Reasoning* **93** (2018), 395–423. doi:10.1016/j.ijar.2017.11.013.
[49] M. Diller, A. Keshavarzi Zafarghandi, T. Linsbichler and S. Woltran, Investigating subclasses of abstract dialectical frameworks, in: *Proc. COMMA*, S. Modgil, K. Budzynska and J. Lawrence, eds, Frontiers in Artificial Intelligence and Applications, Vol. 305, IOS Press, 2018, pp. 61–72.
[50] Y. Dimopoulos, J. Mailly and P. Moraitis, Control argumentation frameworks, in: *Proc. AAAI*, S.A. McIlraith and K.Q. Weinberger, eds, AAAI Press, 2018, pp. 4678–4685.
[51] S. Doutre, A. Herzig and L. Perrussel, A dynamic logic framework for abstract argumentation, in: *Proc. KR*, C. Baral, G.D. Giacomo and T. Eiter, eds, AAAI Press, 2014, pp. 62–71.
[52] S. Doutre and J. Mailly, Constraints and changes: A survey of abstract argumentation dynamics, *Argument & Computation* **9**(3) (2018), 223–248. doi:10.3233/AAC-180425.
[53] 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) (1995), 321–357. doi:10.1016/0004-3702(94)00041-X.
[54] P.M. Dung, An axiomatic analysis of structured argumentation with priorities, *Artificial Intelligence* **231** (2016), 107–150. doi:10.1016/j.artint.2015.10.005.
[55] P.M. Dung and P.M. Thang, Representing the semantics of abstract dialectical frameworks based on arguments and attacks, *Argument & Computation* **9**(3) (2018), 249–267. doi:10.3233/AAC-180427.
[56] P.E. Dunne, Computational properties of argument systems satisfying graph-theoretic constraints, *Artificial Intelligence* **171**(10–15) (2007), 701–729. doi:10.1016/j.artint.2007.03.006.
[57] P.E. Dunne, W. Dvořák, T. Linsbichler and S. Woltran, Characteristics of multiple viewpoints in abstract argumentation, *Artificial Intelligence* **228** (2015), 153–178. doi:10.1016/j.artint.2015.07.006.
[58] F. Dupin de Saint-Cyr, P. Bisquert, C. Cayrol and M. Lagasquie-Schiex, Argumentation update in YALLA (yet another logic language for argumentation), *International Journal of Approximate Reasoning* **75** (2016), 57–92. doi:10.1016/j.ijar.2016.04.003.
[59] W. Dvořák and P.E. Dunne, Computational problems in formal argumentation and their complexity, in: *Handbook of Formal Argumentation*, P. Baroni, D. Gabbay, M. Giacomin and L. van der Torre, eds, College Publications, 2018, pp. 631–688, Chap. 13.
[60] W. Dvořák, S. Ordyniak and S. Szeider, Augmenting tractable fragments of abstract argumentation, *Artificial Intelligence* **186** (2012), 157–173. doi:10.1016/j.artint.2012.03.002.
[61] S. Ellmauthaler, Abstract Dialectical Frameworks: Properties, Complexity, and Implementation, Master's thesis, Technische Universität Wien, Institut für Informationssysteme, 2012.
[62] S. Ellmauthaler and H. Strass, The DIAMOND system for computing with abstract dialectical frameworks, in: *Proc. COMMA*, S. Parsons, N. Oren, C. Reed and F. Cerutti, eds, Frontiers in Artificial Intelligence and Applications, Vol. 266, IOS Press, 2014, pp. 233–240.
[63] S. Ellmauthaler and H. Strass, DIAMOND 3.0 – a native C++ implementation of DIAMOND, in: *Proc. COMMA, Frontiers in Artificial Intelligence and Applications*, Vol. 287, IOS Press, 2016, pp. 471–472.
[64] X. Fan and F. Toni, A general framework for sound assumption-based argumentation dialogues, *Artificial Intelligence* **216** (2014), 20–54. doi:10.1016/j.artint.2014.06.001.
[65] D.M. Gabbay, Semantics for higher level attacks in extended argumentation frames part 1: Overview, *Studia Logica* **93**(2–3) (2009), 357–381. doi:10.1007/s11225-009-9211-4.
[66] S.A. Gaggl, T. Linsbichler, M. Maratea and S. Woltran, Introducing the second international competition on computational models of argumentation, in: *Proc. SAFA*, M. Thimm, F. Cerutti, H. Strass and M. Vallati, eds, CEUR Workshop Proceedings, Vol. 1672, CEUR-WS.org, 2016, pp. 4–9.
[67] S.A. Gaggl, T. Linsbichler, M. Maratea and S. Woltran, Summary report of the second international competition on computational models of argumentation, *AI Magazine* **39**(4) (2018), 77–79. doi:10.1609/aimag.v39i4.2781.

[68] A.J. García and G.R. Simari, Defeasible logic programming: An argumentative approach, *Theory and Practice of Logic Programming* **4**(1–2) (2004), 95–138. doi:10.1017/S1471068403001674.
[69] P. Gärdenfors, *Knowledge in Flux: Modelling the Dynamics of Epistemic States*, The MIT Press, Cambridge, MA, 1988.
[70] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub and P. Wanko, Theory solving made easy with clingo 5, in: *Technical Communications of ICLP*, 2016, pp. 2–1215.
[71] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming, in: *Proc. ICLP/SLP*, MIT Press, 1988, pp. 1070–1080.
[72] T.F. Gordon, H. Prakken and D. Walton, The Carneades model of argument and burden of proof, *Artificial Intelligence* **171**(10–15) (2007), 875–896. doi:10.1016/j.artint.2007.04.010.
[73] A. Haret, J.P. Wallner and S. Woltran, Two sides of the same coin: Belief revision and enforcing arguments, in: *Proc. IJCAI*, J. Lang, ed., ijcai.org, 2018, pp. 1854–1860.
[74] H. Katsuno and A.O. Mendelzon, On the difference between updating a knowledge base and revising it, in: *Proc. KR*, Morgan Kaufmann, 1991, pp. 387–394.
[75] S. Konieczny and R. Pino Pérez, Merging information under constraints: A logical framework, *Journal of Logic and Computation* **12**(5) (2002), 773–808. doi:10.1093/logcom/12.5.773.
[76] T. Lehtonen, J.P. Wallner and M. Järvisalo, From structured to abstract argumentation: Assumption-based acceptance via AF reasoning, in: *Proc. ECSQARU*, A. Antonucci, L. Cholvy and O. Papini, eds, Lecture Notes in Computer Science, Vol. 10369, Springer, 2017, pp. 57–68.
[77] T. Linsbichler, M. Maratea, A. Niskanen, J.P. Wallner and S. Woltran, Novel algorithms for abstract dialectical frameworks based on complexity analysis of subclasses and SAT solving, in: *Proc. IJCAI*, J. Lang, ed., ijcai.org, 2018, pp. 1905–1911.
[78] T. Linsbichler, J. Pührer and H. Strass, A uniform account of realizability in abstract argumentation, in: *Proc. ECAI*, G.A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum and F. van Harmelen, eds, Frontiers in Artificial Intelligence and Applications, Vol. 285, IOS Press, 2016, pp. 252–260.
[79] T. Linsbichler and S. Woltran, Revision of abstract dialectical frameworks: Preliminary report, in: *Proc. Arg-LPNMR*, S.A. Gaggl, J.C. Nieves and H. Strass, eds, 2016, proceedings available at: http://arxiv.org/abs/1611.02439.
[80] M.J. Maher, Corrupt strategic argumentation: The ideal and the naive, in: *Proc. 29th Australasian Joint Conference on Artificial Intelligence*, B.H. Kang and Q. Bai, eds, Lecture Notes in Computer Science, Vol. 9992, Springer, 2016, pp. 17–28.
[81] M.J. Maher, Resistance to corruption of general strategic argumentation, in: *Proc. PRIMA*, M. Baldoni, A.K. Chopra, T.C. Son, K. Hirayama and P. Torroni, eds, Lecture Notes in Computer Science, Vol. 9862, Springer, 2016, pp. 61–75.
[82] M.J. Maher, Resistance to corruption of strategic argumentation, in: *Proc. AAAI*, D. Schuurmans and M.P. Wellman, eds, AAAI Press, 2016, pp. 1030–1036.
[83] S. Modgil, Reasoning about preferences in argumentation frameworks, *Artificial Intelligence* **173**(9–10) (2009), 901–934. doi:10.1016/j.artint.2009.02.001.
[84] S. Modgil and H. Prakken, Resolutions in structured argumentation, in: *Proc. COMMA*, B. Verheij, S. Szeider and S. Woltran, eds, Frontiers in Artificial Intelligence and Applications, Vol. 245, IOS Press, 2012, pp. 310–321.
[85] S. Modgil and H. Prakken, A general account of argumentation with preferences, *Artificial Intelligence* **195** (2013), 361–397. doi:10.1016/j.artint.2012.10.008.
[86] S. Modgil and H. Prakken, Abstract rule-based argumentation, in: *Handbook of Formal Argumentation*, P. Baroni, D. Gabbay, M. Giacomin and L. van der Torre, eds, College Publications, 2018, pp. 287–364, Chap. 6.
[87] S. Nielsen and S. Parsons, A generalization of Dung's abstract framework for argumentation: Arguing with sets of attacking arguments, in: *Proc. ArgMAS*, N. Maudet, S. Parsons and I. Rahwan, eds, Lecture Notes in Computer Science, Vol. 4766, Springer, 2007, pp. 54–73.
[88] I. Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, *Annals of Mathematics and Artificial Intelligence* **25**(3–4) (1999), 241–273. doi:10.1023/A:1018930122475.
[89] A. Niskanen, J.P. Wallner and M. Järvisalo, Synthesizing argumentation frameworks from examples, in: *Proc. ECAI*, G.A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum and F. van Harmelen, eds, Frontiers in Artificial Intelligence and Applications, Vol. 285, IOS Press, 2016, pp. 551–559.
[90] A. Niskanen, J.P. Wallner and M. Järvisalo, Extension enforcement under grounded semantics in abstract argumentation, in: *Proc. KR*, M. Thielscher, F. Toni and F. Wolter, eds, AAAI Press, 2018, pp. 178–183.
[91] F. Nouioua and V. Risch, Bipolar argumentation frameworks with specialized supports, in: *Proc. ICTAI*, IEEE Computer Society, 2010, pp. 215–218.
[92] F. Nouioua and V. Risch, Argumentation frameworks with necessities, in: *Proc. SUM*, S. Benferhat and J. Grant, eds, Lecture Notes in Computer Science, Vol. 6929, Springer, 2011, pp. 163–176.
[93] N. Oren and T.J. Norman, Semantics for evidence-based argumentation, in: *Proc. COMMA*, P. Besnard, S. Doutre and A. Hunter, eds, Frontiers in Artificial Intelligence and Applications, Vol. 172, IOS Press, 2008, pp. 276–284.

[94] N. Oren, C. Reed and M. Luck, Moving between argumentation frameworks, in: *Proc. COMMA*, P. Baroni, F. Cerutti, M. Giacomin and G.R. Simari, eds, Frontiers in Artificial Intelligence and Applications, Vol. 216, IOS Press, 2010, pp. 379–390.
[95] S. Polberg, Understanding the abstract dialectical framework, in: *Proc. JELIA*, L. Michael and A.C. Kakas, eds, Lecture Notes in Computer Science, Vol. 10021, Springer, 2016, pp. 430–446.
[96] R. Riveret, On learning abstract argumentation graphs from bivalent statement labellings, in: *Proc. ICTAI*, IEEE Computer Society, 2016, pp. 190–195.
[97] R. Riveret and G. Governatori, On learning attacks in probabilistic abstract argumentation, in: *Proc. AAMAS*, C.M. Jonker, S. Marsella, J. Thangarajah and K. Tuyls, eds, ACM, 2016, pp. 653–661.
[98] H. Strass and S. Ellmauthaler, goDiamond 0.6.6 ICCMA 2017 System Description, 2017.
[99] H. Strass and J.P. Wallner, Analyzing the computational complexity of abstract dialectical frameworks via approximation fixpoint theory, *Artificial Intelligence* **226** (2015), 34–74. doi:10.1016/j.artint.2015.05.003.
[100] L.G. Valiant, A theory of the learnable, *Communications of the ACM* **27**(11) (1984), 1134–1142. doi:10.1145/1968.1972.
[101] J.P. Wallner, Structural constraints for dynamic operators in abstract argumentation, in: *Proc. COMMA*, S. Modgil, K. Budzynska and J. Lawrence, eds, Frontiers in Artificial Intelligence and Applications, Vol. 305, IOS Press, 2018, pp. 73–84.
[102] J.P. Wallner, A. Niskanen and M. Järvisalo, Complexity results and algorithms for extension enforcement in abstract argumentation, *Journal of Artificial Intelligence Research* **60** (2017), 1–40. doi:10.1613/jair.5415.
[103] B. Yun, S. Vesic and M. Croitoru, Toward a more efficient generation of structured argumentation graphs, in: *Proc. COMMA*, S. Modgil, K. Budzynska and J. Lawrence, eds, Frontiers in Artificial Intelligence and Applications, Vol. 305, IOS Press, 2018, pp. 205–212.