

A Dynamic Logic Framework for Abstract Argumentation

Sylvie Doutre
IRIT - Université de Toulouse

Andreas Herzig
IRIT - Université de Toulouse

Laurent Perrussel
IRIT - Université de Toulouse

Abstract

We provide a logical analysis of abstract argumentation frameworks and their dynamics. Following previous work, we express attack relation and argument status by means of propositional variables and define acceptability criteria by formulas of propositional logic. We here study the dynamics of argumentation frameworks in terms of basic operations on these propositional variables, viz. change of their truth values. We describe these operations in a uniform way within a well-known variant of Propositional Dynamic Logic PDL: the Dynamic Logic of Propositional Assignments, DL-PA. The atomic programs of DL-PA are assignments of propositional variables to truth values, and complex programs can be built by means of the connectives of sequential and nondeterministic composition and test. We start by showing that in DL-PA, the construction of extensions can be performed by a DL-PA program that is parametrized by the definition of acceptance. We then mainly focus on how the acceptance of one or more arguments can be enforced and show that this can be achieved by changing the truth values of the propositional variables describing the attack relation in a minimal way.

Introduction

Argumentation is a reasoning model based on the construction and on the evaluation of arguments. The seminal approach by Dung (Dung 1995) represents an argumentation framework \mathcal{AF} as a set of abstract arguments, the structure and origin of which are left unspecified, along with an attack relationship between arguments. This paper builds on this framework. Dung and his followers have defined semantics for the evaluation of the acceptability of arguments (see (Baroni and Giacomin 2009) for a comprehensive overview). We focus in this paper on extension-based semantics, that define collectively acceptable sets of arguments, called extensions.

Dung's \mathcal{AF} has already been represented in various logics, notably in propositional logic, starting with (Besnard and Doutre 2004). There, \mathcal{AF} is described by means of a boolean formula in a logical language whose propositional variables represent the attacks (the *attack variables*). Furthermore, extensions of the \mathcal{AF} under a given semantics σ can also be described by means of boolean formulas constraining valuations to correspond to the extensions under

the semantics. This is done in an extension of the language of attack variables by variables representing argument acceptance.

Based on such a logical representation, several authors have recently investigated the dynamics of the \mathcal{AF} , such as (Baumann 2012; Booth et al. 2013; Bisquert et al. 2013; Coste-Marquis et al. 2013). They start by distinguishing several kinds of modification of the \mathcal{AF} , such as the addition or the removal of attacks, or the enforcement of the acceptability of an argument a (e.g. such that a is part of at least one extension). All these papers build on previous work in belief change, either referring to AGM theory (Alchourrón, Gärdenfors, and Makinson 1985), such as (Booth et al. 2013; Coste-Marquis et al. 2013), or to KM theory (Katsuno and Mendelzon 1992), such as (Bisquert et al. 2013). They express the modification as a logical formula describing some *goal*, i.e., a property that \mathcal{AF} should satisfy: the task is to revise/update \mathcal{AF} so that this formula is true.

The above papers do not provide a single framework encompassing at the same time \mathcal{AF} , the logical definition of the enforcement constraint and the change operations: there is usually one language for representing \mathcal{AF} and another language for representing constraints, plus some definitions in the metalanguage connecting them. This has motivated us to provide a general, unified logical framework for the representation and the update of argumentation frameworks. We make use of a flexible yet simple logic: Dynamic Logic of Propositional Assignments, abbreviated DL-PA (Balbiani, Herzig, and Troquard 2013). DL-PA is a simple instantiation of Propositional Dynamic Logic PDL (Harel 1984; Harel, Kozen, and Tiuryn 2000) whose atomic programs are assignments of propositional variables to either true or false. Complex programs are built then from atomic programs by the standard PDL program operators of sequential composition, nondeterministic composition, and test. We here moreover add a less frequently considered PDL program operator, namely the converse operator. The language of DL-PA has formulas of the form $\langle \pi \rangle \varphi$ and $[\pi] \varphi$, where π is a program and φ is a formula. The former expresses that φ is true after *some* possible execution of π , and the latter expresses that φ is true after *every* possible execution of π . It is shown in (Balbiani, Herzig, and Troquard 2013) that every DL-PA formula can be reduced to an equivalent propositional formula. The reduction extends to the converse operator in a straight-

forward manner and provides a syntactical representation of the modified belief base.

We start by showing that the construction of extensions under a given semantics can be performed by a DL-PA program that is parametrized by the formula describing the semantics. Then we consider modifications of the attack relation and/or of the extensions. Modifications of the extensions are enforced by changing the attack relation only (addition or removal of attacks between the existing arguments). This can be achieved by changing the truth values of the attack variables. More precisely, to every input formula A describing the desired modification we associate a DL-PA program π_A implementing the update by A . We can then check whether a formula C is true in all (resp. in some) extensions of (the argumentation framework resulting from) the update of \mathcal{AF} by the goal A .

The paper is organized as follows. In the next section we recall the definitions of an argumentation framework and of various semantics as well as their encoding in propositional logic. We then introduce DL-PA and show how to use it to construct extensions, before applying it to the modification of the attack relation and of the extensions. After that, we discuss several ways to extend our framework in order to capture other kinds of modifications. The last section discusses related work and concludes.¹

Representing Argumentation Frameworks

In the present section we set the stage for our paper: we recall the definition of argumentation frameworks and provide a logical language to reason about them.

Argumentation Frameworks An abstract argumentation framework as defined by Dung in (Dung 1995) is a pair $\mathcal{AF} = (\mathcal{A}, R)$ where \mathcal{A} is a finite set of abstract arguments and $R \subseteq \mathcal{A} \times \mathcal{A}$ is a binary relation on \mathcal{A} , called the attack relation: $(a, b) \in R$ means that a attacks b . So an argumentation framework is a directed graph with arguments as vertices and attacks as edges.

An argumentation framework \mathcal{AF} can be represented in propositional logic as follows. First, a set of *attack variables* is associated to the set of arguments \mathcal{A} :

$$\text{ATT}_{\mathcal{A}} = \{\text{Att}_{a,b} : (a, b) \in \mathcal{A} \times \mathcal{A}\}$$

Let \mathcal{L}_{Att} be the set of all formulas that are built from attack variables. The theory of $\mathcal{AF} = (\mathcal{A}, R)$ is the boolean formula

$$\text{Th}_{\mathcal{AF}} = \left(\bigwedge_{(a,b) \in R} \text{Att}_{a,b} \right) \wedge \left(\bigwedge_{(a,b) \notin R} \neg \text{Att}_{a,b} \right)$$

Example 1. Consider the set of arguments $\mathcal{A}_1 = \{a, b\}$ and the two argumentation frameworks $\mathcal{AF}_1 = (\mathcal{A}_1, R_1)$ and $\mathcal{AF}_2 = (\mathcal{A}_1, R_2)$, with $R_1 = \{(a, b)\}$ and $R_2 = \{(a, b), (b, a)\}$. They are depicted in Figure 1.

\mathcal{AF}_1 and \mathcal{AF}_2 are described by the theories

$$\begin{aligned} \text{Th}_{\mathcal{AF}_1} &= \text{Att}_{a,b} \wedge \neg \text{Att}_{b,a} \wedge \neg \text{Att}_{a,a} \wedge \neg \text{Att}_{b,b} \\ \text{Th}_{\mathcal{AF}_2} &= \text{Att}_{a,b} \wedge \text{Att}_{b,a} \wedge \neg \text{Att}_{a,a} \wedge \neg \text{Att}_{b,b} \end{aligned}$$

¹A longer version of the paper including proofs is available at: <http://www.irit.fr/~Laurent.Perrussel/kr14long.pdf>

$$a \rightarrow b \quad a \leftrightarrow b$$

Figure 1: \mathcal{AF}_1 (on the left) and \mathcal{AF}_2 (on the right)

Argumentation Semantics Many semantics have been defined for acceptability. Some of them are extension-based, some others are labelling-based. In this paper we only consider the first: they prevail in the literature, and moreover equivalent extension-based formulations of labelling-based semantics are available (Baroni and Giacomin 2009).

For a given \mathcal{AF} , a semantics identifies sets of acceptable arguments, called extensions. There may be none, one or several such extensions. In order to characterize extensions we associate to \mathcal{A} a second set of propositional variables:

$$\text{IN}_{\mathcal{A}} = \{\text{In}_a : a \in \mathcal{A}\}$$

where In_a stands for “argument a is in the extension”.

As proposed in (Besnard and Doutre 2004) and as extensively discussed in (Gabbay 2011), several semantics can be captured in propositional logic. Among such semantics, we consider at first the well-known *stable*, *admissible* and *complete* semantics.²

We write the propositional formulas capturing these semantics in a language that is built from the set of propositional variables $\mathbb{P} = \text{ATT}_{\mathcal{A}} \cup \text{IN}_{\mathcal{A}}$ by means of the boolean connectives in the usual way. Let $\mathcal{L}_{\text{Att}, \text{In}}$ be our language of Boolean formulas.

A *stable* extension of $\mathcal{AF} = (\mathcal{A}, R)$ is a set $S \subseteq \mathcal{A}$ such that it does not exist two arguments a and b in S such that $(a, b) \in R$ (that is, S is *conflict-free*), and for each argument $b \notin S$, there exists $a \in S$ such that $(a, b) \in R$ (any argument outside the extension is attacked by the extension). This can be rephrased in propositional logic as follows (Besnard and Doutre 2004):

$$\text{Stable}_{\mathcal{A}} = \bigwedge_{a \in \mathcal{A}} \left(\text{In}_a \leftrightarrow \neg \bigvee_{b \in \mathcal{A}} (\text{In}_b \wedge \text{Att}_{b,a}) \right)$$

An *admissible* set of $\mathcal{AF} = (\mathcal{A}, R)$ is a conflict-free set $S \subseteq \mathcal{A}$ that *defends* all its elements: for all $a \in S$, if there exists b such that $(b, a) \in R$, then there is some argument $c \in S$ such that $(c, b) \in R$.

$$\text{Adm}_{\mathcal{A}} = \bigwedge_{a \in \mathcal{A}} \left(\text{In}_a \rightarrow \bigwedge_{b \in \mathcal{A}} \left(\text{Att}_{b,a} \rightarrow \left(\neg \text{In}_b \wedge \bigvee_{c \in \mathcal{A}} (\text{In}_c \wedge \text{Att}_{c,b}) \right) \right) \right)$$

A *complete* extension of $\mathcal{AF} = (\mathcal{A}, R)$ is an admissible set $S \subseteq \mathcal{A}$ that contains all the arguments it defends, that is: if an argument a is such that, for all b such that $(b, a) \in R$,

²Admissibility is usually considered to be a building brick of other, stronger semantics, but we consider it here as a semantics on its own, the construction of admissible sets following the same process as the construction of extensions.

there exists some $c \in S$ such that $(c, b) \in R$, then $a \in S$.

$$\text{Complete}_{\mathcal{A}} = \bigwedge_{a \in \mathcal{A}} \left(\left(\text{In}_a \rightarrow \bigvee_{b \in \mathcal{A}} (\text{In}_b \wedge \text{Att}_{b,a}) \right) \wedge \left(\text{In}_a \leftrightarrow \bigwedge_{b \in \mathcal{A}} \left(\text{Att}_{b,a} \rightarrow \bigvee_{c \in \mathcal{A}} (\text{In}_c \wedge \text{Att}_{c,b}) \right) \right) \right)$$

The *justification state* of an argument (Baroni and Giacomin 2009) depends on the extensions it belongs to; basically, an argument is *credulously* justified (resp. *skeptically* justified) under a given semantics, if it belongs to at least one of (resp. all) the extensions under the semantics.

Extensions and Valuations Argumentation frameworks and semantics being encoded in propositional logic, computing the extensions under the semantics consists in finding the models of the formulas representing the argumentation system and the semantics. A *valuation* is a subset of the set of propositional variables $\mathbb{P} = \text{ATT}_{\mathcal{A}} \cup \text{IN}_{\mathcal{A}}$. The set of all valuations is $2^{\mathbb{P}}$. We use v, v_1, v_2 , etc. to denote valuations. A given valuation determines the truth value of the boolean formulas of the language $\mathcal{L}_{\text{Att}, \text{In}}$ in the usual way. The set of valuations where A is true is called *the set of models of A* or *the set of A-valuations* and is noted $\|A\|$.

The following proposition, adapted from (Besnard and Doutre 2004), connects extensions and valuations.

Proposition 1. *Let $\mathcal{AF} = (\mathcal{A}, R)$ be an argumentation framework and let $E \subseteq \mathcal{A}$. Consider the valuation $v_E = \{\text{Att}_{a,b} : (a, b) \in R\} \cup \{\text{In}_a : a \in E\}$.*

1. *E is a stable extension of \mathcal{AF} if and only if v_E is a model of $\text{Th}_{\mathcal{AF}} \wedge \text{Stable}_{\mathcal{A}}$.*
2. *E is an admissible set of \mathcal{AF} if and only if v_E is a model of $\text{Th}_{\mathcal{AF}} \wedge \text{Adm}_{\mathcal{A}}$.*
3. *E is a complete extension of \mathcal{AF} if and only if v_E is a model of $\text{Th}_{\mathcal{AF}} \wedge \text{Complete}_{\mathcal{A}}$.*

Given a definition of a semantics σ (either stable, admissible, or complete), we denote the associated propositional formula describing the semantics by $\text{Semantics}_{\mathcal{A}}^{\sigma}$.

Corollary 1. *Let $\mathcal{AF} = (\mathcal{A}, R)$ be an argumentation framework. Let σ be a semantics. Let $C \in \mathcal{L}_{\text{Att}, \text{In}}$ be a formula describing some property of \mathcal{A} . All extensions of \mathcal{AF} have property C if and only if $\text{Th}_{\mathcal{AF}} \wedge \text{Semantics}_{\mathcal{A}}^{\sigma} \rightarrow C$ is valid.*

Example 2. *Let us consider again Example 1. \mathcal{AF}_1 has a single stable extension $\{a\}$, two admissible extensions $\{\}$ and $\{a\}$, and one complete extension $\{a\}$; that is:*

$$\begin{aligned} \text{Th}_{\mathcal{AF}_1} \wedge \text{Stable}_{\mathcal{A}_1} &\rightarrow \text{In}_a \wedge \neg \text{In}_b \\ \text{Th}_{\mathcal{AF}_1} \wedge \text{Adm}_{\mathcal{A}_1} &\rightarrow \neg \text{In}_b \\ \text{Th}_{\mathcal{AF}_1} \wedge \text{Complete}_{\mathcal{A}_1} &\rightarrow \text{In}_a \wedge \neg \text{In}_b \end{aligned}$$

DL-PA: Dynamic Logic of Propositional Assignments

We have just seen that looking for the extensions of an argumentation framework \mathcal{AF} with respect to some semantics σ amounts to looking for the models of the formulas

representing \mathcal{AF} and σ . We now consider a way to build these models in a logic which allows to perform operations on valuations: Dynamic Logic of Propositional Assignments DL-PA (Herzig et al. 2011; Balbiani, Herzig, and Troquard 2013). DL-PA is based on Propositional Dynamic Logic PDL (Harel 1984; Harel, Kozen, and Tiuryn 2000). Just as PDL, it has operators of sequential and nondeterministic composition of programs, test and iteration (the Kleene star), which enable reasoning about programs. The atomic programs of DL-PA are assignments of truth values to propositional variables, such as assigning true or false to p , respectively written $p \leftarrow \top$ and $p \leftarrow \perp$. The formulas of DL-PA can express how valuations are modified by programs so that after this modification stage, some property holds. In our context, they express how to build the valuations representing the extensions of \mathcal{AF} w.r.t. σ . Before showing this we define syntax and semantics of DL-PA and state complexity results.

We here base our work on the star-free version of DL-PA of (Herzig et al. 2011), that we extend by the operator of converse execution of programs. We keep on calling that logic DL-PA, because the converse operator can be linearly eliminated, as we shall see.

Language The language of star-free DL-PA is defined by the following grammar:

$$\begin{aligned} \pi &::= p \leftarrow \top \mid p \leftarrow \perp \mid \varphi? \mid \pi; \pi \mid \pi \cup \pi \mid \pi^- \\ \varphi &::= p \mid \top \mid \perp \mid \neg \varphi \mid \varphi \vee \varphi \mid \langle \pi \rangle \varphi \end{aligned}$$

where p ranges over \mathbb{P} .

Key formulas are: $[\pi]\varphi$, read “after every execution of the program π formula φ holds”, and $\langle \pi \rangle \varphi$, read “after some execution of the program π formula φ holds”. The formula $[\pi]\varphi$ abbreviates $\neg \langle \pi \rangle \neg \varphi$.

The *atomic programs* $p \leftarrow \top$ and $p \leftarrow \perp$ respectively make p true and make p false. The operators of sequential composition (“;”), nondeterministic composition (“ \cup ”) and test (“ $\langle \cdot \rangle$ ”) are familiar from PDL. The operator “ $\langle \cdot \rangle$ ” is the converse operator: formula $[\pi^-]\varphi$ stands for “before every execution of the program π formula φ was true” and $\langle \pi^- \rangle \varphi$ stands for “before some execution of the program π formula φ was true”.

The *length* of a formula φ , noted $|\varphi|$, is the number of symbols used to write down φ , without “ \langle ”, “ \rangle ”, and parentheses. For example, $|\neg(q \vee \neg r)| = 1 + (1 + 1 + 2) = 5$ and $|(q \leftarrow \top)(q \vee r)| = 3 + 3 = 6$. The length of a program π , noted $|\pi|$, is defined in the same way. For example, $|p \leftarrow \perp; p?| = 3 + 1 + 2 = 6$.

Conjunction (\wedge), implication (\rightarrow) and equivalence (\leftrightarrow) are considered with their usual meaning. We also make use the exclusive disjunction $\varphi \oplus \psi$ which abbreviates $(\varphi \wedge \neg \psi) \vee (\neg \varphi \wedge \psi)$. Furthermore, several program abbreviations are familiar from PDL. First, skip abbreviates $\top?$ (“nothing happens”). Second, for $n \geq 0$, we define inductively π^n and $\pi^{\leq n}$ as follows:

$$\begin{aligned} \pi^n &= \begin{cases} \text{skip} & \text{if } n = 0 \\ \pi; \pi^{n-1} & \text{if } n \geq 1 \end{cases} \\ \pi^{\leq n} &= \begin{cases} \text{skip} & \text{if } n = 0 \\ (\text{skip} \cup \pi); \pi^{\leq n-1} & \text{if } n \geq 1 \end{cases} \end{aligned}$$

$\ p \leftarrow \top\ = \{(v_1, v_2) : v_2 = v_1 \cup \{p\}\}$
$\ p \leftarrow \perp\ = \{(v_1, v_2) : v_2 = v_1 \setminus \{p\}\}$
$\ \varphi?\ = \{(v, v) : v \in \ \varphi\ \}$
$\ \pi; \pi'\ = \ \pi\ \circ \ \pi'\ $
$\ \pi \cup \pi'\ = \ \pi\ \cup \ \pi'\ $
$\ \pi^-\ = \ \pi\ ^{-1}$
$\ p\ = \{v : p \in v\}$
$\ \top\ = 2^{\mathbb{P}}$
$\ \perp\ = \{\}$
$\ \neg\varphi\ = 2^{\mathbb{P}} \setminus \ \varphi\ $
$\ \varphi \vee \psi\ = \ \varphi\ \cup \ \psi\ $
$\ \langle \pi \rangle \varphi\ = \{v : \text{there is } v' \text{ s.t. } (v, v') \in \ \pi\ \text{ and } v' \in \ \varphi\ \}$

Table 1: Interpretation of the DL-PA connectives

Finally, we introduce assignments of literals to variables by way of the following abbreviations:

$$p \leftarrow q = (q?; p \leftarrow \top) \cup (\neg q?; p \leftarrow \perp)$$

$$p \leftarrow \neg q = (q?; p \leftarrow \perp) \cup (\neg q?; p \leftarrow \top)$$

The former assigns to p the truth value of q , while the latter assigns to p the truth value of $\neg q$. So $p \leftarrow p$ does nothing and $p \leftarrow \neg p$ flips the truth value of p . Note that both abbreviations have constant length (viz. 14).

Semantics Models of DL-PA are nothing but models of classical propositional logic, i.e., subsets of the set of propositional variables \mathbb{P} , alias valuations. We sometimes write $v(p) = 1$ when $p \in v$ and $v(p) = 0$ when $p \notin v$. DL-PA programs are interpreted by means of a (unique) *relation between valuations*. For instance, suppose $\mathbb{P} = \{p\}$ and consider the atomic program $\pi = p \leftarrow \top$, and the two valuations $v_1 = \{\}$ and $v_2 = \{p\}$. The execution of π relates v_1 to v_2 , and v_2 to itself. In other words, π is interpreted as $\|\pi\| = \{(v_1, v_2), (v_2, v_2)\}$.

Atomic programs of the form $p \leftarrow \top$ and $p \leftarrow \perp$ are interpreted as update operations on valuations, and complex programs are interpreted just as in PDL by mutual recursion. Table 1 gives the interpretation of the DL-PA connectives.

Two formulas φ_1 and φ_2 are *formula equivalent* if $\|\varphi_1\| = \|\varphi_2\|$. Two programs π_1 and π_2 are *program equivalent* if $\|\pi_1\| = \|\pi_2\|$. In that case we write $\pi_1 \equiv \pi_2$. For example, the program equivalence $\pi; \text{skip} \equiv \pi$ holds.

An *expression* is a formula or a program. When we say that two expressions are equivalent we mean program equivalence if we are talking about programs, and formula equivalence otherwise. Equivalence is preserved under replacement of a sub-expression by an equivalent expression (Balbiani, Herzig, and Troquard 2013).

Eliminating the Modal Operators DL-PA formulas being interpreted through classical propositional logic valuations, modal operators can be eliminated. In order to do this, we first eliminate the converse operator. Table 2 lists

$(\varphi?)^- \equiv \varphi?$
$(\pi_1; \pi_2)^- \equiv \pi_2^-; \pi_1^-$
$(\pi_1 \cup \pi_2)^- \equiv \pi_1^- \cup \pi_2^-$
$p \leftarrow \top^- \equiv p?; (\text{skip} \cup p \leftarrow \perp)$
$\equiv p? \cup (p?; p \leftarrow \perp)$
$p \leftarrow \perp^- \equiv \neg p?; (\text{skip} \cup p \leftarrow \top)$
$\equiv \neg p? \cup (\neg p?; p \leftarrow \top)$

Table 2: Reduction axioms for the converse operator

the relevant program equivalences (aka reduction axioms). Iterating their application we can eliminate the converse operator with only linear increase in program size. For example, for assignments of variables to literals we have:

$$p \leftarrow q^- \equiv \begin{cases} p \leftarrow p & \text{if } p = q \\ p \leftrightarrow q?; (p \leftarrow \top \cup p \leftarrow \perp) & \text{otherwise} \end{cases}$$

$$p \leftarrow \neg q^- \equiv \begin{cases} p \leftarrow \neg p & \text{if } p = q \\ p \oplus q?; (p \leftarrow \top \cup p \leftarrow \perp) & \text{otherwise} \end{cases}$$

Proposition 2. *For every DL-PA formula φ there is an equivalent DL-PA formula φ' such that φ' is without $(.)^-$ and such that the length of φ' is linear in the length of φ .*

Next, we remind the following key result: for converse-free DL-PA, every formula is equivalent to a boolean formula (Herzig et al. 2011; Balbiani, Herzig, and Troquard 2013). With Proposition 2 we obtain the following result for our present language with converse.

Theorem 1. *For every DL-PA formula there is an equivalent boolean formula.*

For example, for different propositional variables r and p , the formula $\langle p \leftarrow \perp \rangle (p \vee r)$ is successively equivalent to $\langle p \leftarrow \perp \rangle p \vee \langle p \leftarrow \perp \rangle r$, to $\perp \vee r$, and to r .

Note that the boolean formula might be exponentially longer than the original formula.

Validity and Complexity Results A formula φ is DL-PA *valid* if it is equivalent to \top , i.e., if $\|\varphi\| = 2^{\mathbb{P}}$. It is DL-PA *satisfiable* if it is not a formula equivalent to \perp , i.e., if $\|\varphi\| \neq \{\}$. For example, the formulas $\langle p \leftarrow \top \rangle \top$ and $\langle p \leftarrow \top \rangle \varphi \leftrightarrow \neg \langle p \leftarrow \top \rangle \neg \varphi$ are DL-PA valid.

Due to Proposition 2, the complexity results of (Herzig et al. 2011) transfer to our extension with the converse operator: both model checking and satisfiability checking are PSPACE complete for our language.

Constructing Extensions with DL-PA

Let us first show how the extensions of an argumentation framework can be constructed by means of DL-PA programs. Remember that such programs modify valuations: our aim is to define a program which sets the truth values of the variables representing the acceptance of an argument so that after the execution of that program, the formula characterizing the semantics holds. We first present a few definitions that will help us to define that program.

Sorting Variables of $\mathcal{L}_{\text{Att}, \text{In}}$ Formulas For every $\mathcal{L}_{\text{Att}, \text{In}}$ formula A we consider two sets: the set of attack variables occurring in A , noted $\text{ATT}(A)$, and the set of accept variables occurring in A , noted $\text{IN}(A)$. For example, $\text{ATT}(\neg \text{Att}_{a,b} \vee \text{In}_a) = \{\text{Att}_{a,b}\}$ and $\text{IN}(\neg \text{Att}_{a,b} \vee \text{In}_a) = \{\text{In}_a\}$. It is the modifications of $\text{ATT}(A)$ that will be minimized in order to construct the extensions, while the modifications of the set $\text{IN}(A)$ will not be minimized: given the truth values of the attack variables, the semantics determines the truth values for the accept variables (or rather, the possible combinations of accept variables, because there may be several extensions).

Evaluating the Difference between Valuations The symmetric difference between two valuations $v_1, v_2 \subseteq \mathbb{P}$ is the number of variables whose truth values differ, formally defined as

$$v_1 \dot{-} v_2 = (v_1 \setminus v_2) \cup (v_2 \setminus v_1)$$

So $v_1 \dot{-} v_2$ is the set of all those p such that $v_1(p) \neq v_2(p)$. The *Hamming distance* between two valuations $v_1, v_2 \subseteq \mathbb{P}$ is the cardinality of their symmetric difference: $v_1 \dot{-} v_2$ is the number of all those p such that $v_1(p) \neq v_2(p)$.

Example 3. Suppose two valuations: $\{p, q\}$ and $\{q, r, s\}$. Their symmetric difference is $\{p, r, s\}$ and their Hamming distance is 3.

Changing Variables through DL-PA Programs Let us consider the two following DL-PA programs that can change the truth value of one or more propositional variables in the finite subset $P = \{p_1, \dots, p_n\}$ of the set of propositional variables $\mathbb{P} = \text{ATT}_{\mathcal{A}} \cup \text{IN}_{\mathcal{A}}$.

$$\text{vary}(P) = (p_1 \leftarrow \top \cup p_1 \leftarrow \perp); \dots; (p_n \leftarrow \top \cup p_n \leftarrow \perp)$$

$$\text{flip1}(P) = p_1 \leftarrow \neg p_1 \cup \dots \cup p_n \leftarrow \neg p_n$$

The intuitive meaning of program $\text{vary}(P)$ is “the program is composed of a sequence of n steps and at each step i , the value of variable p_i is nondeterministically set to either true or false”; one may also say that $\text{vary}(P)$ forgets the values of the variables in P . Second, program $\text{flip1}(P)$ should be read as “one variable from P is randomly chosen and is nondeterministically set to either true or false”. Observe that the programs $\text{vary}(P)$ and $(\text{flip1}(P))^{\leq n}$ are equivalent.

Note that the length of each program is linear in the cardinality of P .

The following lemma characterises the behaviour of the two programs in semantic terms.

Lemma 1. The following hold:

1. $(v_1, v_2) \in \|\text{vary}(P)\|$ iff $v_1 \dot{-} v_2 \subseteq P$
2. $(v_1, v_2) \in \|\text{flip1}(P)\|$ iff $v_1 \dot{-} v_2 = \{p_k\}$ for some $p_k \in P$

Given an $\mathcal{L}_{\text{Att}, \text{In}}$ formula A , \mathbb{P}_A is defined as the set of variables from \mathbb{P} occurring in A . The first item of Lemma 1 hence tells us that the program $\text{vary}(\mathbb{P}_A); A?$ accesses all relevant A -valuations (where ‘relevant’ means that only those A -valuations are accessed which keep constant the truth values of variables not occurring in A).

Example 4. Consider the formula $A = \text{In}_a \wedge \neg \text{In}_b$. The set of variables of A is $\mathbb{P}_A = \{\text{In}_a, \text{In}_b\}$. The models of A can

be built by, first, varying the truth values of the variables of the formula ($\text{vary}(\mathbb{P}_A)$), and second, by testing which of the resulting valuations satisfy the formula, and by keeping them only ($A?$). $\text{vary}(\mathbb{P}_A)$ is here:

$$(\text{In}_a \leftarrow \top \cup \text{In}_a \leftarrow \perp); (\text{In}_b \leftarrow \top \cup \text{In}_b \leftarrow \perp)$$

Its execution relates each valuation for \mathbb{P}_A , i.e., each of $\{\}, \{\text{In}_a\}, \{\text{In}_b\}$, and $\{\text{In}_a, \text{In}_b\}$, to every other, including itself. Among the couples of valuations making up that relation, there is only one relating the model of A (that is, $\{\text{In}_a\}$) to itself. So the execution of the test $A?$ only keeps this pair in the relation. So the pair $(\{\text{In}_a\}, \{\text{In}_a\})$ is a possible execution of the program $\text{vary}(\mathbb{P}_A); A?$.

Our example illustrates that the satisfiability of a boolean formula A can be expressed in DL-PA as the existence of valuation that is accessible by the program $\text{vary}(\mathbb{P}_A); A?$, i.e., by means of the formula

$$\langle \text{vary}(\mathbb{P}_A); A? \rangle \top$$

The formula says that there is a way of changing the truth value of some variables of A so that A is true afterwards.

Lemma 2. Let A be a propositional formula. A is (propositionally) satisfiable iff $\langle \text{vary}(\mathbb{P}_A); A? \rangle \top$ is DL-PA valid.

Defining Extensions in DL-PA Let us now show how a DL-PA program can express in a concise and natural way the construction of extensions of an argumentation framework \mathcal{AF} under a semantics σ . The program $\text{makeExt}_{\mathcal{A}}^{\sigma}$ below constructs the valuations representing extensions w.r.t. a semantics σ , of argumentation frameworks over the set of arguments \mathcal{A} . Remember that the set of attack variables is $\text{ATT}_{\mathcal{A}}$, that the set of accept variables is $\text{IN}_{\mathcal{A}}$, and that σ is either stable, complete, or admissible acceptability semantics. We define the DL-PA program $\text{makeExt}_{\mathcal{A}}^{\sigma}$ as follows:

$$\text{makeExt}_{\mathcal{A}}^{\sigma} = \text{vary}(\text{IN}_{\mathcal{A}}); \text{Semantics}_{\mathcal{A}}^{\sigma}$$

The following result says that if v_1 completely describes the attack relation then the set of accept variables of the valuations v_2 such that $(v_1, v_2) \in \|\text{makeExt}_{\mathcal{A}}^{\sigma}\|$ characterises the extensions of \mathcal{AF} w.r.t. semantics σ . It is a consequence of Corollary 1 and Lemma 2.

Lemma 3. Let \mathcal{AF} be an argumentation framework. Let σ be either the stable, complete or admissible semantics. Let v_1 be a model of $\text{Th}_{\mathcal{AF}}$. Then $(v_1, v_2) \in \|\text{makeExt}_{\mathcal{A}}^{\sigma}\|$ iff v_2 is a model of $\text{Th}_{\mathcal{AF}} \wedge \text{Semantics}_{\mathcal{A}}^{\sigma}$.

Lemma 3 entails that we can construct all the extensions of a given argumentation system w.r.t. some semantics by means of a DL-PA program.

Proposition 3. Let $\mathcal{AF} = (\mathcal{A}, R)$ be an argumentation framework and let C be a boolean formula. The boolean formula $\text{Th}_{\mathcal{AF}} \wedge \text{Semantics}_{\mathcal{A}}^{\sigma} \rightarrow C$ is propositionally valid if and only if the DL-PA formula $\text{Th}_{\mathcal{AF}} \rightarrow [\text{makeExt}_{\mathcal{A}}^{\sigma}]C$ is valid. Moreover, the equivalence

$$\text{Th}_{\mathcal{AF}} \wedge \text{Semantics}_{\mathcal{A}}^{\sigma} \leftrightarrow \langle (\text{makeExt}_{\mathcal{A}}^{\sigma})^{-} \rangle \text{Th}_{\mathcal{AF}}$$

is DL-PA valid.

According to Proposition 3, given a description of the attack relation, program $\text{makeExt}_{\mathcal{A}}^{\sigma}$ can construct valuations where the formula describing the semantics is true, and so in a way such that only acceptance variables are changed, while the attack variables do not change. Note that the equivalence takes care of situations where there is no extension. Let the $\mathcal{L}_{\text{Att}, \text{In}}$ formula A be a goal, i.e., a boolean formula that should be satisfied by the extensions of some \mathcal{AF} . By means of the program $\text{makeExt}_{\mathcal{A}}^{\sigma}$ one can check what has been called σ -consistency in (Coste-Marquis et al. 2013): whether there exists an argumentation framework \mathcal{AF} such that some or every extension w.r.t. σ satisfies A . Adopting the standard terms for characterizing a justification state, we distinguish the notions of credulous and skeptical consistency.

Definition 1 (σ -consistency). Let $\mathcal{AF} = (\mathcal{A}, R)$ be an argumentation framework, let σ be either the stable, admissible, or complete semantics. The $\mathcal{L}_{\text{Att}, \text{In}}$ formula A is σ -credulously-consistent for \mathcal{A} iff $\langle \text{makeExt}_{\mathcal{A}}^{\sigma} \rangle A$ is satisfiable; A is σ -skeptically-consistent for \mathcal{A} iff $[\text{makeExt}_{\mathcal{A}}^{\sigma}] A$ is satisfiable.

Example 5. Consider the set of arguments $\mathcal{A}_2 = \{a, b\}$ and the goal $A = \neg \text{In}_a \wedge \neg \text{In}_b \wedge \neg \text{Att}_{a,a}$. Then $\langle \text{makeExt}_{\mathcal{A}}^{\sigma} \rangle A$ is unsatisfiable; in particular, it is false in every valuation for $\text{Th}_{\mathcal{AF}_2}$ of Example 1. Therefore the formula A is stable-credulous inconsistent for \mathcal{A}_2 .

To sum it up, we have shown how extensions can be constructed through programs modifying propositional variables. In the next section we show how to modify an argumentation framework and its extensions.

Modifying an Argumentation Framework

We start by discussing several kinds of modifications of an argumentation framework \mathcal{AF} and then focus on the most involved family of operations: modifications enforcing a goal either in all extensions or in some extension of \mathcal{AF} .

Generally speaking, when we want to incorporate a new piece of information A into a belief base B then we want to build a new belief base that retains as much as possible the initial beliefs of B while entailing A . Neither AGM theory (Alchourrón, Gärdenfors, and Makinson 1985) nor KM (Katsuno and Mendelzon 1992) theory provide a single, concrete belief change operation: they rather constrain the set of ‘reasonable’ belief change operations by means of a set of postulates. Several concrete belief change operations satisfying the AGM or KM postulates have been defined in the literature. Among the most prominent are Winslett’s ‘possible models approach’ (PMA) (Winslett 1988; 1990), Forbus’s update operation (Forbus 1989), Winslett’s standard semantics (WSS) (Winslett 1995), and Dalal’s revision operation (Dalal 1988); see (Herzig and Rifi 1999; Lang 2007) for an overview. According to Katsuno and Mendelzon’s distinction between update and revision operations (Katsuno and Mendelzon 1992), the first three are update operations, usually written \diamond , while Dalal’s is a revision operation, usually written $*$.

Kinds of Change Several kinds of modifications of a given argumentation framework $\mathcal{AF} = (\mathcal{A}, R)$ have been distinguished (Cayrol, Bannay, and Lagasque-Schiex 2010):

1. One may modify the set of arguments \mathcal{A} by adding or removing an argument.
2. One may modify the attack relation R by adding or removing an edge between two arguments.
3. One may wish to modify the extensions of \mathcal{AF} .

The first kind of operation, viz. adding an argument to the set of arguments \mathcal{A} or removing it, requires more linguistic resources than our language $\mathcal{L}_{\text{Att}, \text{In}}$ provides; we will sketch a way of implementing it in DL-PA in an augmented language in the end of the paper.

Addition and removal of an attack edge between two arguments $a, b \in \mathcal{A}$ are rather simple operations: we just add or subtract (a, b) from R . In our logical representation, these operations correspond to making propositional variables true or false: the addition of (a, b) to R can be implemented by the update of $\text{Th}_{\mathcal{AF}}$ by $\text{Att}_{a,b}$, and the removal of (a, b) from R can be implemented by the update of $\text{Th}_{\mathcal{AF}}$ by $\neg \text{Att}_{a,b}$.³ In DL-PA, the update of $\text{Th}_{\mathcal{AF}}$ by $\text{Att}_{a,b}$ can be implemented by the program $\text{Att}_{a,b} \leftarrow \top$, and the update by $\neg \text{Att}_{a,b}$ by $\text{Att}_{a,b} \leftarrow \perp$. Indeed, we have:

$$\begin{aligned} \text{Th}_{\mathcal{AF}} \diamond \text{Att}_{a,b} &= \langle \text{Att}_{a,b} \leftarrow \top \rangle \text{Th}_{\mathcal{AF}} \\ \text{Th}_{\mathcal{AF}} \diamond \neg \text{Att}_{a,b} &= \langle \text{Att}_{a,b} \leftarrow \perp \rangle \text{Th}_{\mathcal{AF}} \end{aligned}$$

where \diamond is any of the above belief change operations. Once $\text{Th}_{\mathcal{AF}}$ has been updated by some $\text{Att}_{a,b}$ or $\neg \text{Att}_{a,b}$, the extensions of the resulting framework can be obtained by conjoining the result with $\text{Semantics}_{\mathcal{A}}^{\sigma}$.

Adding an argument to extensions or removing it is more involved because it has to be achieved indirectly, by changing the underlying attack relation of \mathcal{AF} (or by changing the set of arguments, but as we said, we disregard this option for the time being). Moreover, when an argumentation framework has several extensions, one may wish to change the justification status of a , so that it be skeptically justified (that is, added in all the extensions), or credulously justified (added in at least one extension), or credulously justified but not skeptically justified (added in at least one extension, but removed from at least one), or even not credulously justified (removed from all extensions).

While the distinction between update and revision was irrelevant for the modification of the attack relation, it gets important when it comes to the modification of extensions: which operation should we apply? We dedicate the next section to the problem of modifying an extension.

³We note that the question whether this is an update or a revision does not really play a role here because we are in a simple situation: as far as the language \mathcal{L}_{Att} is concerned, the formula $\text{Th}_{\mathcal{AF}}$ is complete (it has a unique model), and the update of complete theories by literals leads to the same result for all the belief change operations that we have mentioned, WSS, PMA, Forbus, and Dalal, be they update or revision operations.

The Modification of Extensions In order to discuss which change operation is the most appropriate, let us consider again the above argumentation framework $\mathcal{AF}_2 = (\mathcal{A}_2, R_2)$ and the stable semantics. \mathcal{AF}_2 has two stable extensions $E_a = \{a\}$ and $E_b = \{b\}$. We have seen that the associated boolean formula $\text{Th}_{\mathcal{AF}_2} \wedge \text{Stable}_{\mathcal{A}_2}$ has two $\mathcal{L}_{\text{Att}, \text{In}}$ models:

$$\begin{aligned} v_a &= \{\text{Att}_{a,b}, \text{Att}_{b,a}, \text{In}_a\} \\ v_b &= \{\text{Att}_{a,b}, \text{Att}_{b,a}, \text{In}_b\} \end{aligned}$$

Suppose we wish to modify \mathcal{AF}_2 such that a is in none of its stable extensions. What we have to do is to adapt the attack relation of \mathcal{AF}_2 in a way that is minimal and that guarantees that a does not occur in any of its extensions, while guaranteeing that a is rejected. We view minimality as minimality of the number of modifications of the attack relation; in contrast, the current extensions may be modified in a non-minimal way. This policy agrees with (Coste-Marquis et al. 2013), where the minimization of the changes on the attack relation is considered to be secondary.

What we do is to modify the models of $\text{Th}_{\mathcal{AF}_2} \wedge \text{Stable}_{\mathcal{A}_2}$ one by one so that In_a holds: we minimally modify the attack variables such that the result is stable. As we can see, *enforcing* a constraint is similar to update and we will hereafter commit to this reading.

To sum up, enforcing A consists in minimally updating the attack relation of an initial argumentation framework $\mathcal{AF} = (\mathcal{A}, R)$ with respect to some goal A in a way such that A holds in all of the extensions of the resulting $\mathcal{AF}' = (\mathcal{A}, R')$. Another perspective is however possible, too, where enforcing consists in minimally modifying the attack relation such that A holds in *some* of the extensions of the resulting $\mathcal{AF}' = (\mathcal{A}, R')$. This operation seems to be closer in spirit to a revision operation. So ‘enforcement’ can be understood in two different ways, leading to two different definitions. In both cases, there is a key difference with standard revision and update operations: in classical belief change, output is limited to one belief set, while here, several \mathcal{AF}' may be produced by the enforcement operations. Let $\mathcal{AF} \diamond^{\text{Cred}, \sigma} A$ denote the *credulous* enforcement of property A in \mathcal{AF} w.r.t. semantics σ , and let $\mathcal{AF} \diamond^{\text{Skep}, \sigma} A$ denote the *skeptical* enforcement of property A in \mathcal{AF} . We consider that both $\diamond^{\text{Cred}, \sigma}$ and $\diamond^{\text{Skep}, \sigma}$ are operations mapping an argumentation framework and an $\mathcal{L}_{\text{Att}, \text{In}}$ formula to a set of argumentation frameworks.

In line with the definitions of classical revision and update operations, we now define some postulates that ‘reasonable’ operations \diamond should satisfy. They are mainly inspired by (Coste-Marquis et al. 2013).

We have shown above that both attack relations and σ -consistency can be captured by DL-PA formulas. As moreover the goal A is an $\mathcal{L}_{\text{Att}, \text{In}}$ formula, too, our postulates can be formulated in terms of DL-PA formulas.

Definition 2 (enforcement postulates). Let \diamond be an operation mapping an argumentation framework and an $\mathcal{L}_{\text{Att}, \text{In}}$ formula to a set of argumentation frameworks. Let σ be either the stable, admissible, or complete semantics. The operation \diamond is a credulous enforcement operation iff it satisfies the following postulates:

- E1.C** For all $\mathcal{AF}' \in \mathcal{AF} \diamond^{\sigma} A$, $\models \text{Th}_{\mathcal{AF}'} \rightarrow \langle \text{makeExt}_{\mathcal{A}}^{\sigma} \rangle A$.
- E2.C** If $\models \text{Th}_{\mathcal{AF}} \wedge \text{Semantics}_{\mathcal{A}}^{\sigma} \rightarrow \langle \text{makeExt}_{\mathcal{A}}^{\sigma} \rangle A$ then $\mathcal{AF} \diamond^{\sigma} A = \{\mathcal{AF}\}$.
- E3** If $\models A_1 \leftrightarrow A_2$ then for every $\mathcal{AF}_1 \in \mathcal{AF} \diamond^{\sigma} A_1$ there exists $\mathcal{AF}_2 \in \mathcal{AF} \diamond^{\sigma} A_2$ such that $\models \text{Th}_{\mathcal{AF}_1} \leftrightarrow \text{Th}_{\mathcal{AF}_2}$.

The operation \diamond is said to be a skeptical enforcement operation iff it satisfies postulate **E3** plus the following:

- E1.S** For all $\mathcal{AF}' \in \mathcal{AF} \diamond^{\sigma} A$, $\models \text{Th}_{\mathcal{AF}'} \rightarrow [\text{makeExt}_{\mathcal{A}}^{\sigma}] A$.
- E2.S** If $\models \text{Th}_{\mathcal{AF}} \wedge \text{Semantics}_{\mathcal{A}}^{\sigma} \rightarrow [\text{makeExt}_{\mathcal{A}}^{\sigma}] A$ then $\mathcal{AF} \diamond^{\sigma} A = \{\mathcal{AF}\}$.

The postulates **E1.C** and **E1.S** say that success is required for credulous and skeptical enforcement. **E2** represents a minimal change principle: it states that if A already holds then \mathcal{AF} is unchanged. Postulate **E3** is the postulate of syntax independence: enforcement should be based on the content of a goal and not on its syntax.

Additional postulates may be formulated; see (Bisquert et al. 2013; Coste-Marquis et al. 2013) for more details. A key difference is that we do not consider postulates based on the expansion operation. The main reason is that this operation is actually useless: first, if an attack has to be changed then expansion cannot be used because, as we have seen above, $\text{Th}_{\mathcal{AF}}$ is complete for \mathcal{L}_{Att} . Now consider that an argument has to be enforced in a credulous way. We face two cases: (i) either the argument is already credulously acceptable and there is no reason to change the argumentation framework, or (ii) the argument is not credulously acceptable. Then as all possible extensions are considered, some attacks must be changed so that new extensions can be constructed (and A will then hold). The same reasoning can be made for skeptical acceptance.

Let us now show how DL-PA helps us to reason about enforcement.

Expressing Extension Modification in DL-PA

The aim of this section is to show how modification programs can be defined. These programs should satisfy the enforcement postulates that we have defined. Second, they should only minimally change the attack relations of an initial argumentation framework. As previously mentioned, enforcement is close to update, and we will root our programs in the update operations that guarantee minimal change. One of the most popular is Forbus’s update operation (Forbus 1989), where minimal change involves counting how many variables have been changed. Our solution is strongly connected to it.

The Hamming Distance Predicate Let us define the DL-PA formula $h(A, P, \geq m)$, where A is an $\mathcal{L}_{\text{Att}, \text{In}}$ formula, P is a set of propositional variables, and $m \geq 0$ is an integer:

$$h(A, P, \geq m) = \begin{cases} \top & \text{if } m = 0 \\ \neg \langle (\text{flip1}(P))^{\leq m-1} \rangle A & \text{if } m \geq 1 \end{cases}$$

We call $h(A, P, \geq m)$ the *Hamming distance predicate* w.r.t. the set of variables P : it is true at a valuation v exactly when the A -valuations v' that are closest to v in the sense of the Hamming distance differ in at least m variables from

v , where the computation of the distance only considers variables from the set P , while the other variables in $\mathbb{P} \setminus P$ keep their value.

Proposition 4. *Let v a valuation, A a boolean formula, P some set of propositional variables, and $m \geq 0$. Then*

1. $v \in \llbracket h(A, P, \geq m) \rrbracket$ iff the A -valuations that are closest to v w.r.t. P have Hamming distance at least m , i.e., iff $\text{card}(v \dot{-} v') \geq m$ for every $v' \in \llbracket A \rrbracket$ s.t. $v \dot{-} v' \subseteq P$.
2. $(v, v') \in \llbracket h(A, P, \geq m)?; (\text{flip1}(P))^m; A? \rrbracket$ iff v' is one of the A -valuations that is closest to v w.r.t. P , i.e., iff $v' \in \llbracket A \rrbracket$ and $\text{card}(v \dot{-} v') = m$ for every $v' \in \llbracket A \rrbracket$ s.t. $v \dot{-} v' \subseteq P$.

It follows from the first item of Proposition 4 that when P equals \mathbb{P}_A then $v \in \llbracket h(A, \mathbb{P}_A, \geq m) \rrbracket$ iff the A -valuations that are closest to v have Hamming distance at least m , i.e., iff $\text{card}(v \dot{-} v') \geq m$ for every $v' \in \llbracket A \rrbracket$. This is used in Forbus's update operation.

Forbus's Update Operation Forbus's update operation is based on minimization of the Hamming distance. First, the Forbus update of a valuation v by A is the set of those A -valuations whose Hamming distance to v is minimal. Second, the Forbus update of a belief base B by A collects the Forbus updates of all B -valuations by A .

The following DL-PA program performs Forbus's update operation:

$$\text{forbus}(A) = \bigcup_{m \leq \text{card}(\mathbb{P}_A)} \left(h(A, \mathbb{P}_A, \geq m)?; (\text{flip1}(\mathbb{P}_A))^m; A? \right)$$

The program nondeterministically chooses an integer m , checks if the Hamming distance to A -valuations is at least m and flips m of the variables of A . Finally, the test $A?$ only succeeds for A -valuations.

Proposition 5. *The formula C is true after the Forbus update of B by A if and only if $B \rightarrow [\text{forbus}(A)]C$ is DL-PA valid.*

Argumentation Framework Update The update of argumentation frameworks has some specificities: first, we are going to modify only the attack variables while leaving the accept variables unchanged; second, the target formula is not going to be a boolean formula, but a formula saying that A will be the case after building extensions. Let us define the program $\text{credEnf}(A)$ modifying \mathcal{AF} w.r.t. some semantics σ such that the boolean formula $A \in \mathcal{L}_{\text{Att}, \text{In}}$ becomes true in some σ -extensions:

$$\begin{aligned} \text{credEnf}(A) = & \bigcup_{m \leq \text{card}(\text{ATT}_{\mathcal{A}})} \left(h(\langle \text{makeExt}_{\mathcal{A}}^{\sigma} \rangle A, \text{ATT}_{\mathcal{A}}, \geq m)?; (\text{flip1}(\text{ATT}_{\mathcal{A}}))^m; \right. \\ & \left. \langle \text{makeExt}_{\mathcal{A}}^{\sigma} \rangle A? \right) \end{aligned}$$

The following program enforces a constraint in a skeptical way.

$$\begin{aligned} \text{skepEnf}(A) = & \bigcup_{m \leq \text{card}(\text{ATT}_{\mathcal{A}})} \left(h([\text{makeExt}_{\mathcal{A}}^{\sigma}]A, \text{ATT}_{\mathcal{A}}, \geq m)?; (\text{flip1}(\text{ATT}_{\mathcal{A}}))^m; \right. \\ & \left. [\text{makeExt}_{\mathcal{A}}^{\sigma}]A? \right) \end{aligned}$$

The length of these two programs is polynomial in the cardinality of \mathcal{A} . (The cardinality of the set $\text{ATT}_{\mathcal{A}}$ is quadratic in that of \mathcal{A} and the length of $(\text{flip1}(\text{ATT}_{\mathcal{A}}))^m$ is quadratic in that of \mathcal{A} .)

To check whether C is true in all extensions of \mathcal{AF} modified by A (for $A, C \in \mathcal{L}_{\text{Att}, \text{In}}$) can then be done by checking whether the DL-PA formula $\text{Th}_{\mathcal{AF}} \rightarrow [\text{skepEnf}(A)]C$ is valid. In particular we have the following proposition which states that if there is a chance to get A by modifying the attack variables then the enforcement will succeed:

Proposition 6.

$$\begin{aligned} & \models [\text{credEnf}(A)] \langle \text{makeExt}_{\mathcal{A}}^{\sigma} \rangle A \\ & \models [\text{skepEnf}(A)] [\text{makeExt}_{\mathcal{A}}^{\sigma}] A \end{aligned}$$

The second key property is that an argumentation framework is unchanged if the goal already holds.

Proposition 7. *For every goal A that is credulously justified, the credulous update program does not change anything:*

$$\models (\text{Th}_{\mathcal{AF}} \wedge \langle \text{makeExt}_{\mathcal{A}}^{\sigma} \rangle A \wedge C) \rightarrow [\text{credEnf}(A)]C$$

For every goal A that is skeptically justified, the skeptical update program does not change anything:

$$\models (\text{Th}_{\mathcal{AF}} \wedge [\text{makeExt}_{\mathcal{A}}^{\sigma}] A \wedge C) \rightarrow [\text{skepEnf}(A)]C$$

The modified argumentation frameworks can be extracted from the formulas $\langle \text{credEnf}(A) \rangle \text{Th}_{\mathcal{AF}}$ and $\langle \text{skepEnf}(A) \rangle \text{Th}_{\mathcal{AF}}$ representing it in DL-PA (or rather, their reduction) by forgetting the accept variables, as proposed in (Coste-Marquis et al. 2013). This operation can be implemented in our framework by the program $\text{vary}(\text{IN}_{\mathcal{A}})$.

Definition 3 ($\diamond^{\sigma, \text{enf}}$). *Let σ be either the stable, admissible, or complete semantics. Let enf be either the skepEnf or the credEnf program. Let $\diamond^{\sigma, \text{enf}}$ be an operation mapping an argumentation framework and an $\mathcal{L}_{\text{Att}, \text{In}}$ formula to a set of argumentation frameworks. The update of \mathcal{AF} by A under σ and enf is*

$$\mathcal{AF} \diamond^{\sigma, \text{enf}} A = \{(\mathcal{A}, R_v) : v \in \llbracket \langle \text{enf}(A) \rangle \text{Th}_{\mathcal{AF}} \rrbracket\}$$

where R_v is the attack relation extracted from v , defined as $R_v = \{(a, b) : \text{Att}_{a, b} \in v\}$.

The two preceding propositions guarantee that our enforcement operations satisfy the postulates.

Theorem 2. *Operation $\diamond^{\sigma, \text{credEnf}}$ satisfies E1.C and E2.C. Operation $\diamond^{\sigma, \text{skepEnf}}$ satisfies E1.S and E2.S. Both operations $\diamond^{\sigma, \text{credEnf}}$ and $\diamond^{\sigma, \text{skepEnf}}$ satisfy E3.*

This result is actually not a surprise, given that our tool for enforcement is a variant of Forbus's update operation.

Example 6. *Let us take up the argumentation framework \mathcal{AF}_2 of Example 1. Remember that $\mathcal{AF}_2 = (\mathcal{A}_1, R_2)$, with $\mathcal{A}_1 = \{a, b\}$ and $R_2 = \{(a, b), (b, a)\}$ and that $\text{Th}_{\mathcal{AF}_2} = \text{Att}_{a, b} \wedge \text{Att}_{b, a}$. Let us consider the stable semantics and suppose our goal is to enforce that a is always acceptable (skeptical enforcement). We disregard self-attacks for the sake of simplicity. The nondeterministic part $\bigcup_{m \leq \text{card}(\text{ATT}_{\mathcal{A}})} (\dots)$ of the program $\text{SkepEnf}(\text{In}_a)$ changes one variable from*

$\text{Th}_{\mathcal{AF}}$, either $\text{Att}_{a,b}$ or $\text{Att}_{b,a}$. This corresponds to two candidate extensions: one where a only attacks b and one where b only attacks a . Only the former case gives valuations where a is always acceptable. Hence:

$$\mathcal{AF} \diamond^{\text{Stable, skipEnf}} \text{In}_a = \{(\mathcal{A}_1, \{(a, b)\})\}$$

Going Further

We have illustrated how DL-PA offers a fruitful framework for representing argumentation framework and reasoning about them. We now sketch several ways of extending our account.

Other Argumentation Semantics Our exposition focused on the stable, admissible, and complete semantics. This can however be generalized to *every* semantics Σ whose extensions can be characterized by a propositional formula $\Sigma_{\mathcal{A}}$ built from the set of attack variables $\text{ATT}_{\mathcal{A}}$ and the set of accept variables $\text{IN}_{\mathcal{A}}$. In the previous sections we have given DL-PA programs modifying valuations (that represent argumentation frameworks) in a minimal way and producing (representations of) extensions. Therefore all minimization-based semantics can be handled in an elegant way, as well as all maximization-based semantics.

For instance, the preferred semantics (that captures the maximal, w.r.t. set-inclusion, admissible sets) can be handled as follows:

$$\text{Semantics}_{\mathcal{A}}^{\text{pref}} =$$

$$\text{Adm}_{\mathcal{A}} \wedge [\text{MakeTrueSome}(\text{IN}_{\mathcal{A}}); \text{h}(\text{Adm}_{\mathcal{A}}, \geq 1)?] \neg \text{Adm}_{\mathcal{A}}$$

where $\text{MakeTrueSome}(P) = (\neg p_1?; (p_1 \leftarrow \perp \cup p_1 \leftarrow \top)); \dots; (\neg p_n?; (p_n \leftarrow \perp \cup p_n \leftarrow \top))$ makes true some of the elements of P . $[\text{MakeTrueSome}(\text{IN}_{\mathcal{A}})]$ evaluates all possible combinations of changes in the arguments out of the admissible set. $[\text{MakeTrueSome}(\text{IN}_{\mathcal{A}}); \text{h}(\text{Adm}_{\mathcal{A}}, \geq 1)?] \neg \text{Adm}_{\mathcal{A}}$ tries to enforce all non-empty subsets of arguments currently out of the admissible set. Maximality of the admissible set is thus guaranteed. The resulting encoding is polynomial time.

Other Update Semantics The modification programs that we have defined in the present paper basically applies Forbus's update operation to the attack variables. It is possible to use other such operations, such as Winslett's Possible Models Approach (PMA) (Winslett 1988), or revision operations such as Dalal's (Dalal 1988).

Other Kinds of Change Up to now we did not cover addition and removal of some argument from an argumentation framework. Let us sketch how this could be done in a fairly straightforward way in the framework of DL-PA. First of all, we have to add a further ingredient to argumentation frameworks: let us consider triples $\mathcal{AF} = (\mathcal{A}_0, \mathcal{A}, R)$ where $\mathcal{A} \subseteq R \times R$ as before, and moreover \mathcal{A}_0 is a (possibly infinite) set such that $\mathcal{A} \subseteq \mathcal{A}_0$. We think of \mathcal{A}_0 as the background set of all possible arguments, while \mathcal{A} is the set of all arguments that are currently under consideration. We have to modify the logical language $\mathcal{L}_{\text{Att}, \text{In}}$ accordingly. First, we suppose that there are propositional variables $\text{Att}_{a,b}$ and In_a for every a, b in the background set \mathcal{A}_0 . Second, we add propositional variables Cons_a , one per argument $a \in \mathcal{A}_0$, where Cons_a

reads “ a is considered”. Let us denote the resulting language by $\mathcal{L}_{\text{Att}, \text{In}, \text{Cons}}$. Then the theory of $\mathcal{AF} = (\mathcal{A}_0, \mathcal{A}, R)$ is the boolean formula

$$\text{Th}_{\mathcal{AF}} = \left(\bigwedge_{(a,b) \in R} \text{Att}_{a,b} \right) \wedge \left(\bigwedge_{(a,b) \notin R} \neg \text{Att}_{a,b} \right) \wedge \left(\bigwedge_{a \in \mathcal{A}} \text{Cons}_a \right)$$

So $\text{Th}_{\mathcal{AF}}$ does not say anything about the arguments that are currently not under consideration: the models of $\text{Th}_{\mathcal{AF}}$ have arbitrary truth values for $\text{Att}_{a,b}$, $\text{Att}_{b,a}$, In_a , and Cons_a as soon as $a \notin \mathcal{A}$.

The semantic definitions have to be adapted, too, and should only quantify over arguments in \mathcal{A} , and not over those in \mathcal{A}_0 . For stable semantics we get:

$$\text{Stable}_{\mathcal{A}} = \bigwedge_{a \in \mathcal{A}} \left(\text{Cons}_a \rightarrow \left(\text{In}_a \leftrightarrow \neg \bigvee_{b \in \mathcal{A}} (\text{In}_b \wedge \text{Att}_{b,a}) \right) \right)$$

In this setting, the mere addition or deletion of an argument a can be achieved straightforwardly, viz. by changing the status of a from ‘disregarded’ to ‘considered’ by means of the assignments $\text{Cons}_a \leftarrow \top$ and $\text{Cons}_a \leftarrow \perp$. Subsequently, the attack relation can be modified as sketched above: one has to first delete all the $\text{Att}_{a,b}$ and $\text{Att}_{b,a}$ such that $b \in \mathcal{A}$, and then add the attack relation as desired.

Conclusion

The main result of this paper is the encoding of argumentation frameworks and their dynamics in DL-PA. More precisely, our contribution is threefold.

First, as long as argument acceptability can be expressed in propositional logic, finding acceptable arguments and enforcing acceptability can be done in DL-PA. Other logical frameworks allow capturing and computing argument acceptability (see (Charwat et al. 2013) for an overview), but few of them allow capturing and computing acceptability change as well.

Second, as DL-PA formulas can be rewritten as propositional logic formulas, the result of the modification of an argumentation framework is described by a propositional formula from the models of which one may retrieve the modified argumentation frameworks. Our proposal is hence more ‘operational’ than those of (Bisquert et al. 2013; Coste-Marquis et al. 2013) because we use a formal logic encompassing the representation of change operations. Moreover, we consider not only credulous acceptability changes, as most of the current approaches do (Baumann 2012), but skeptical acceptability change as well.

Third, our framework takes advantage of the complexity results for DL-PA: both model checking and satisfiability checking are in PSPACE. A closer look at the formulas expressing the modifications shows that the alternation of quantifications is bounded, which typically leads to complexity bounds at the second level of the polynomial hierarchy.

The proposed DL-PA encodings may be related to QBF encodings for argumentation (Arieli and Caminada 2013). As QBF has the same complexity as star-free DL-PA, all we do in DL-PA must be polynomially encodable into QBF.

However, the availability of assignment programs makes a difference: update programs such as `forbus(A)` and extension construction programs such as `makeExtAσ` capture things in a more general, flexible, and natural way than a QBF encoding.

To conclude, the richness of our framework makes it expandable to other kinds of changes, other update semantics, and other argumentation semantics beyond those that are detailed in the present paper. We plan to investigate this research avenue in future work.

Acknowledgements

This work benefited from the support of the AMANDE project (ANR-13-BS02-0004) of the French National Research Agency (ANR). We would like to thank Gabriele Kern-Isberner for triggering our work when DL-PA was presented at the Dagstuhl seminar “Belief Change and Argumentation in Multi-Agent Scenarios” in June 2013.

References

- Alchourrón, C.; Gärdenfors, P.; and Makinson, D. 1985. On the logic of theory change: Partial meet contraction and revision functions. *J. of Symbolic Logic* 50:510–530.
- Arieli, O., and Caminada, M. W. 2013. A QBF-based formalization of abstract argumentation semantics. *Journal of Applied Logic* 11(2):229–252.
- Balbani, P.; Herzig, A.; and Troquard, N. 2013. Dynamic logic of propositional assignments: a well-behaved variant of PDL. In *Logic in Computer Science (LICS)*. IEEE.
- Baroni, P., and Giacomin, M. 2009. Semantics of abstract argument systems. In Simari, G., and Rahwan, I., eds., *Argumentation in Artificial Intelligence*. Springer US. 25–44.
- Baumann, R. 2012. What does it take to enforce an argument? minimal change in abstract argumentation. In Raedt, L. D.; Bessi re, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P. J. F., eds., *ECAI 2012*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 127–132. IOS Press.
- Besnard, P., and Doutre, S. 2004. Checking the acceptability of a set of arguments. In *10th Int. Workshop on Non-Monotonic Reasoning (NMR’2004)*, 59–64.
- Bisquert, P.; Cayrol, C.; Bannay, F.; and Lagasquie-Schiex, M.-C. 2013. Enforcement in Argumentation is a kind of Update. In Liu, W.; Subrahmanian, V.; and Wijsen, J., eds., *International Conference on Scalable Uncertainty Management (SUM)*, Washington DC, USA, number 8078 in LNAI, 30–43. Springer Verlag.
- Booth, R.; Kaci, S.; Rienstra, T.; and van der Torre, L. 2013. A logical theory about dynamics in abstract argumentation. In Liu, W.; Subrahmanian, V. S.; and Wijsen, J., eds., *SUM*, volume 8078 of *Lecture Notes in Computer Science*, 148–161. Springer.
- Cayrol, C.; Bannay, F.; and Lagasquie-Schiex, M.-C. 2010. Change in Abstract Argumentation Frameworks: Adding an Argument. *Journal of Artificial Intelligence Research* 38:49–84.
- Charwat, G.; Dvor k, W.; Gaggl, S. A.; Wallner, J. P.; and Woltran, S. 2013. Implementing abstract argumentation – a survey. Technical report, DBAI-TR-2013-82.
- Coste-Marquis, S.; Konieczny, S.; Maily, J.-G.; and Marquis, P. 2013. On the revision of argumentation systems: Minimal change of arguments status. In *TAFI’13*.
- Dalal, M. 1988. Investigations into a theory of knowledge base revision: preliminary report. In *Proc. 7th Conf. on Artificial Intelligence (AAAI’88)*, 475–479.
- Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.* 77(2):321–357.
- Forbus, K. D. 1989. Introducing actions into qualitative simulation. In Sridharan, N. S., ed., *Proc. 11th Int. Joint Conf. on Artificial Intelligence (IJCAI’89)*, 1273–1278. Morgan Kaufmann Publishers.
- Gabbay, D. M. 2011. Dung’s argumentation is essentially equivalent to classical propositional logic with the peirce–quine dagger. *Logica Universalis* 5(2):255–318.
- Harel, D.; Kozen, D.; and Tiuryn, J. 2000. *Dynamic Logic*. MIT Press.
- Harel, D. 1984. Dynamic logic. In Gabbay, D. M., and G n thner, F., eds., *Handbook of Philosophical Logic*, volume II. D. Reidel, Dordrecht. 497–604.
- Herzig, A., and Rifi, O. 1999. Propositional belief base update and minimal change. *Artificial Intelligence Journal* 115(1):107–138.
- Herzig, A.; Lorini, E.; Moisan, F.; and Troquard, N. 2011. A dynamic logic of normative systems. In Walsh, T., ed., *International Joint Conference on Artificial Intelligence (IJCAI)*, 228–233. Barcelona: IJCAI/AAAI. Erratum at <http://www.irit.fr/~Andreas.Herzig/P/Ijcai11.html>.
- Katsuno, H., and Mendelzon, A. O. 1992. On the difference between updating a knowledge base and revising it. In G rdenfors, P., ed., *Belief revision*. Cambridge University Press. 183–203. (preliminary version in Allen, J.A., Fikes, R., and Sandewall, E., eds., *Principles of Knowledge Representation and Reasoning: Proc. 2nd Int. Conf.*, pages 387–394. Morgan Kaufmann Publishers, 1991).
- Lang, J. 2007. Belief update revisited. In *Proc. of the 10th International Joint Conference on Artificial Intelligence (IJCAI’07)*, 2517–2522.
- Winslett, M.-A. 1988. Reasoning about action using a possible models approach. In *Proc. 7th Conf. on Artificial Intelligence (AAAI’88)*, 89–93.
- Winslett, M.-A. 1990. *Updating Logical Databases*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.
- Winslett, M.-A. 1995. Updating logical databases. In Gabbay, D. M.; Hogger, C. J.; and Robinson, J. A., eds., *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4. Oxford University Press. 133–174.