A ConArg-based Library for Abstract Argumentation*

Stefano Bistarelli, Fabio Rossi, Francesco Santini

Department of Mathematics and Computer Science, Italy
[bista,rossi,francesco.santini]@dmi.unipg.it

Abstract. We present *ConArgLib*, a C++ library implemented to help programmers solve some of the most important problems related to extension-based Abstract Argumentation. The library is based on ConArg, which exploits *Constraint Programming* and, in particular, Gecode, a toolkit for developing constraint-based systems and applications. Given a semantics, such problems consist, for example, in enumerating all the extensions, and checking the credulous or sceptical acceptance of an argument passed as parameter. The goal is to let programmers use the library to quickly develop programs on top of it, as, for instance, implementing decision-making procedures based on the strongest arguments.

1 Introduction

We present *ConArgLib*, a C++ library implemented to help programmers solve some intractable problems related to extension-based Abstract Argumentation [12]. The library is based on a revised search engine developed for *ConArg* [3], a stand-alone solver submitted to *ICCMA15* [19] and (in its revised version) to *ICCMA17*[1], that is the *International Competition on Computational Models of Argumentation*. The search engine exploits *Gecode*[2], a toolkit for developing constraint-based systems and applications.

An *Abstract Argumentation Framework* (*AAF*) is simply a pair $\langle \mathcal{A}_{rgs}, R \rangle$ consisting of a set $\mathcal{A}_{rgs}$ whose elements are called arguments, and of a binary relation R on $\mathcal{A}_{rgs}$, called "attack" relation. For example, the framework $\langle \{a, b, c\}, \{R(a, b), R(b, c)\} \rangle$ has three arguments labelled as a, b, and c, and there is an attack between a and b, and b and c. A framework can be simply represented as a directed graph, where nodes are arguments, and edges model attacks. The sets of arguments (or *extensions* [12]) to be considered are then defined under different semantics, which represent different degrees of scepticism. Most of them are based on both the notion of conflict-freedom (extensions must not contain attacks), and the notion of defence: in the previous AAF, b attacks c, but a defends c since it attacks b.

* Research supported by project "Argumentation 360" (funded by Dept. of Mathematics and Computer Science, University of Perugia), and project "REMIX" (funded by Banca d'Italia and Fondazione Cassa di Risparmio di Perugia).

[1] http://www.dbai.tuwien.ac.at/iccma17/.

[2] http://www.gecode.org.

Recent years have seen researchers drawing their attention on computational aspects of Abstract Argumentation; in particular, on tools that are capable to solve classical problems, as, given a semantics, *i)* the *enumeration* of extensions satisfying it, *ii)* the *verification* whether a subset of arguments satisfies it, *iii)* and *iv)* the *sceptical* and *credulous* acceptance of an argument (if an argument belongs to resp. all or at least one of the extensions), *v)* the *existence* of at least one extension, and finally *vi)* the non-emptiness of at least one extension, i.e., if different from $\varnothing$.

In addition to AAFs [12], ConArgLib also solves a weighted extension of them we present in [2, 6]: *semiring-based Weighted Abstract Argumentation Frameworks* (simply *WAAFs* in the following), where attacks are associated with a value (taken an algebraic structure, i.e., a c-semiring [5]) representing *e.g.*, a strength score or an uncertainty degree. Then, in [2] we study two different relaxations of WAAFs: the first one is related to the new weighted defence we propose in [6], by checking the difference between the composition of inward and outward attack-weights. The second one is related to how much inconsistency we are willing to tolerate inside an extension; such an amount is computed by aggregating the weights of the attacks between any two arguments inside an extension. These relaxations are correlated: allowing a small conflict may lead to have more arguments into an extension, and consequently result in a stronger or weaker defence.

To code the library we take advantage of *Constraint Programming* (CP) [18] since the problems i-vi can be intractable [11], depending on the semantics. Despite the plethora of aforementioned solvers, as far as we know, ConArgLib represents the first attempt to provide a fast implementation of a library to support the solution of problems in Abstract Argumentation. The end programmer can use it to directly develop her own applications, instead of interfacing to an external solver: as an example, solving the existence of a non-empty extension, and the credulous/sceptical acceptance of arguments can be used to set-up a decision-making procedure by ranking the arguments, and then to select the decision supported by the strongest ones.

The paper is organised as follows: in Sec. 2 we provide the necessary background on Dung's AAFs [12], and c-semirings [5], i.e., the general algebraic structure we use to parametrise weights in WAAFs in [2, 6], whose presentation concludes this section. In Sec. 3 we provide a description of ConArgLib. Then, in Sec. 4 we report some implementation details to explain the engine of the library. Finally, in Sec. 5 we wrap up the paper with general conclusions and ideas about future work.

2 Background

This section is structured in three parts: first we collect the main background notions about AAFs [12] (Sec. 2.1), then we introduce c-semirings (Sec. 2.2), and finally semiring-based WAAFs (Sec. 2.3).

2.1 Abstract Argumentation Frameworks

In his pioneering work [12], Dung proposed *Abstract Frameworks* for Argumentation, where an argument is an abstract entity whose role is solely determined by its relations to other arguments:

Definition 1 (Argumentation Frameworks [12]). *An Abstract Argumentation Framework (AAF) is a pair $\langle \mathcal{A}_{rgs}, R \rangle$ of a set $\mathcal{A}_{rgs}$ of arguments and a binary relation R on $\mathcal{A}_{rgs}$, called attack relation. $\forall a_i, a_j \in \mathcal{A}_{rgs}$, $a_i R\, a_j$ (or $R(a_i, a_j)$) means that a_i attacks a_j (R is asymmetric).*

An *argumentation semantics* is the formal definition of a method (either declarative or procedural) ruling the argument evaluation process. In the *extension*-based approach, a semantics definition specifies how to derive from an AAF a set of extensions, where an extension $\mathcal{B}$ of an AAF $\langle \mathcal{A}_{rgs}, R \rangle$ is simply a subset of $\mathcal{A}_{rgs}$. In Def. 2 we define the first semantics, which is at the basis of all the others:

Definition 2 (Conflict-free [12]). *A set $\mathcal{B} \subseteq \mathcal{A}_{rgs}$ is conflict-free iff no two arguments a and b in $\mathcal{B}$ exist such that a attacks b.*

All the other semantics presented in this section rely (explicitly or implicitly) upon the concept of defence:

Definition 3 (Defence [12]). *An argument c is defended by a set $\mathcal{B} \subseteq \mathcal{A}_{rgs}$ (or $\mathcal{B}$ defends b) iff for any argument $a \in \mathcal{A}_{rgs}$, if $R(a, c)$ then $\exists b \in \mathcal{B}$ s.t., $R(b, a)$.*

An admissible set of arguments according to Dung must be a conflict-free set which defends all its elements. Formally:

Definition 4 (Admissible [12]). *A conflict-free set $\mathcal{B} \subseteq \mathcal{A}_{rgs}$ is an admissible set iff each argument in $\mathcal{B}$ is defended by $\mathcal{B}$.*

The following four definitions elaborate on conflict-freedom and admissibility:

Definition 5 (Complete [12]). *An admissible set $\mathcal{B} \subseteq \mathcal{A}_{rgs}$ is a complete extension iff each argument which is defended by $\mathcal{B}$ is in $\mathcal{B}$.*

Definition 6 (Preferred [12]). *A preferred extension is a maximal (w.r.t. set inclusion) admissible set of $\mathcal{A}_{rgs}$.*

The definition of stage [20] and semi-stable [10] semantics is based on the idea of prescribing the maximisation not only of the arguments included in an extension (as for the preferred extension), but also of those attacked by it:

Definition 7 (Stage [20] and semi-stable [10]). *Given a set $\mathcal{B} \subseteq \mathcal{A}_{rgs}$, the* range *of $\mathcal{B}$ is defined as $\mathcal{B} \cup \mathcal{B}^+$, where $\mathcal{B}^+ = \{a \in \mathcal{A}_{rgs} : \mathcal{B}$ attacks $a\}$. $\mathcal{B}$ is a stage extension iff is a conflict-free set with maximal (wrt. set inclusion) range. $\mathcal{B}$ is semi-stable extension iff $\mathcal{B}$ is a complete extension with maximal (w.r.t. set inclusion) range.*

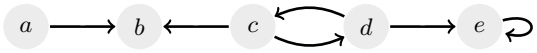

Fig. 1. An example of AAF.

Finally, the stable semantics corresponds to the most stringent among all:

Definition 8 (Stable [12]). *A conflict-free set $\mathcal{B} \subseteq \mathcal{A}_{rgs}$ is a stable extension iff for each argument which is not in $\mathcal{B}$, there exists an argument in $\mathcal{B}$ that attacks it.*

The last three semantics, i.e., the grounded [12], ideal [13], and eager [9], enforce a sceptical approach: these semantics that always yields exactly one extension.

Definition 9 (Grounded [12]). *The grounded extension is the minimal (w.r.t. set inclusion) complete extension [12].*

Definition 10 (Ideal [13]). *$\mathcal{B} \subseteq \mathcal{A}_{rgs}$ is ideal iff $\mathcal{B}$ is admissible and $\forall \mathcal{B}'$ s.t. $\mathcal{B}'$ is preferred, $\mathcal{B} \subseteq \mathcal{B}'$. The ideal extension is the maximal (w.r.t. set inclusion) ideal set.*

Definition 11 (Eager [9]). *$\mathcal{B} \subseteq \mathcal{A}_{rgs}$ is eager iff $\mathcal{B}$ is admissible and $\forall \mathcal{B}'$ s.t. $\mathcal{B}'$ is semi-stable, $\mathcal{B} \subseteq \mathcal{B}'$. The eager extension is the maximal (w.r.t. set inclusion) eager set.*

If $\sigma \in \{cf, adm, com, prf, sst, stg, stb\}$ respectively stand for conflict-free and admissible sets, complete, preferred, semi-stable, stage, and stable semantics, and $\xi_\sigma(F)$ is the set of all the extensions satisfying σ on a framework F, then we have $\xi_{stb}(F) \subseteq \xi_{sst}(F) \subseteq \xi_{prf}(F) \subseteq \xi_{com}(F) \subseteq \xi_{adm}(F) \subseteq \xi_{cf}(F)$, and $\xi_{stb}(F) \subseteq \xi_{stg}(F) \subseteq \xi_{cf}(F)$. Moreover, if *grd*, *ide*, and *eag* are respectively the grounded, ideal, and eager extension, then we have that $grd \subseteq ide \subseteq eag$ [9].

Consider the AAF $F = \langle A, R \rangle$ in Fig. 1, with $A = \{a, b, c, d, e\}$ and $R = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\}$. For example we have that $\xi_{adm}(F)$ corresponds to $\{\varnothing, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}\}$, $\xi_{com}(F) = \{\{a\}, \{a, c\}, \{a, d\}\}$, $\xi_{stb}(F) = \{\{a, c\}, \{a, d\}\}$, $\xi_{stb}(F) = \{\{a, d\}\}$, and $\xi_{grd} = \{\{a\}\}$.

2.2 C-semirings

C-semirings [5] are *commutative* ($\otimes$ is commutative) and *idempotent* semirings (i.e., $\oplus$ is idempotent), where $\oplus$ defines a complete lattice: every subset of elements have a *least upper bound*, or *lub*, and a *greatest lower bound*, or *glb*.

Definition 12 (C-semirings [5]). *A commutative semiring is a tuple $\mathbb{S} = \langle S, \oplus, \otimes, \bot, \top \rangle$ such that S is a set, $\top, \bot \in S$, and $\oplus, \otimes : S \times S \longrightarrow S$ are binary operators making the triples $\langle S, \oplus, \bot \rangle$ and $\langle S, \otimes, \top \rangle$ commutative monoids (semi-groups with identity), satisfying* i) $\forall s, t, u \in S. s \otimes (t \oplus u) = (s \otimes t) \oplus (s \otimes u)$

(distributivity), and ii) $\forall s \in S. s \otimes \bot = \bot$ *(annihilator). If* $\forall s, t \in S. s \oplus (s \otimes t) = s$, $\mathbb{S}$ *is said to be absorptive, and consequently* $\oplus$ *is idempotent. In short, c-semirings are defined as commutative and absorptive semirings.*

The idempotency of $\oplus$ leads to the definition of a partial ordering $\leqslant_{\mathbb{S}}$ over the set S (S is a poset). Such partial order is defined as $s \leqslant_{\mathbb{S}} t$ if and only if $s \oplus t = t$, and $\oplus$ returns the *lub* of s and t. This intuitively means that t is "better" than s. Some more properties can be derived on c-semirings [5]: *i)* both $\oplus$ and $\otimes$ are monotone over $\leqslant_{\mathbb{S}}$, *ii)* $\otimes$ is intensive (i.e., $s \otimes t \leqslant_{\mathbb{S}} s$), and *iii)* $\langle S, \leqslant_{\mathbb{S}} \rangle$ is a complete lattice. $\bot$ and $\top$ are respectively the bottom and top elements of such lattice. When also $\otimes$ is idempotent, *i)* $\oplus$ distributes over $\otimes$, *ii)* $\otimes$ returns the *glb* of two values in S, and *iii)* $\langle S, \leqslant_{\mathbb{S}} \rangle$ is a distributive lattice.

Some well-known instances of c-semirings in the literature are: $\mathbb{S}_{boolean} = \langle \{false, true\}, \vee, \wedge, false, true \rangle$[3], $\mathbb{S}_{fuzzy} = \langle [0,1], \max, \min, 0, 1 \rangle$, $\mathbb{S}_{bottleneck} = \langle \mathbb{R}^+ \cup \{+\infty\}, \max, \min, 0, \infty \rangle$, $\mathbb{S}_{probabilistic} = \langle [0,1], \max, \times, 0, 1 \rangle$ (or *Viterbi* semiring), $\mathbb{S}_{weighted} = \langle \mathbb{R}^+ \cup \{+\infty\}, min, +, +\infty, 0 \rangle$.

Although c-semirings have been historically used as monotonic structures, the need of removing values has raised in local consistency algorithms [2]. A solution comes from *residuation theory* [8], which allows for obtaining a division operator that represents a "weak" inverse of $\otimes$.

Definition 13 (Division). *A c-semiring* $\mathbb{S}$ *is residuated if the set* $\{x \in S \mid t \otimes x \leqslant s\}$ *admits a maximum for all elements* $s, t \in S$*, denoted* $s \oslash t$*.*

Since a complete[4] tropical-semiring is also residuated, we have that all the classical instances of c-semiring presented above are residuated, i.e., each element in S admits an "inverse", which can be also unique:

Definition 14 (Unique invertibility). *If* $\mathbb{S}$ *is absorptive and invertible, then it is uniquely invertible iff it is cancellative, i.e.,* $\forall s, t, u \in S. (s \otimes u = t \otimes u) \wedge (u \neq 0) \Rightarrow s = t$*.*

Since all the previously listed instances of c-semirings are cancellative, they are uniquely invertible as well. For instance,

$$\min\{x \mid b + x \geqslant a\} = \begin{cases} 0 & \text{if } b \geqslant a \\ a - b & \text{if } a > b \end{cases} \qquad \textit{weighted}$$

$$\max\{x \mid \min(b, x) \leqslant a\} = \begin{cases} 1 & \text{if } b \leqslant a \\ a & \text{if } a < b \end{cases} \qquad \textit{fuzzy}$$

[3] Boolean c-semirings can be used to model crisp problems and classical Argumentation [12].

[4] $\mathbb{S}$ is complete if it is closed with respect to infinite sums, and the distributivity law holds also for an infinite number of summands.

2.3 Semiring-based Weighted AAFs

In the beginning of this section we rephrase some of the classical definitions given in Sec. 2.1, with the purpose of parametrising them with the notion of weighted attack and c-semiring. The following definition reshapes the notion of WAAF into *semiring-based WAAF* [2], called $WAAF_{\mathbb{S}}$:

Definition 15 (Semiring-based WAAF). *A semiring-based Weighted AAF ($WAAF_{\mathbb{S}}$) is a quadruple $\langle \mathcal{A}_{rgs}, R, W, \mathbb{S} \rangle$, where $\mathbb{S}$ is a c-semiring $\langle S, \oplus, \otimes, \bot, \top \rangle$, $\mathcal{A}_{rgs}$ is a set of arguments, R the attack binary-relation on $\mathcal{A}_{rgs}$, and $W : \mathcal{A}_{rgs} \times \mathcal{A}_{rgs} \longrightarrow S$ is a binary function. Given $a, b \in \mathcal{A}_{rgs}$ and $R(a, b)$, then $W(a, b) = s$ means that a attacks b with a weight $s \in S$. Moreover, we require that $R(a, b)$ iff $W(a, b) <_{\mathbb{S}} \top$.*

In Fig. 2 we provide an example of a weighted interaction graph describing the $WAAF_{\mathbb{S}}$ defined by $\mathcal{A}_{rgs} = \{a, b, c, d, e\}$, $R = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\}$, with $W(a, b) = 7, W(c, b) = 8, W(c, d) = 9, W(d, c) = 8, W(d, e) = 5, W(e, e) = 6$, and $\mathbb{S} = \langle \mathbb{R}^+ \cup \{\infty\}, \min, +, \infty, 0 \rangle$ (i.e., the weighted semiring).

Therefore, each attack is associated with a semiring value that represents the "strength" of an attack between two arguments. We can consider the weights in Fig. 2 as supports to the associated attack, as similarly suggested in [14]. A semiring value equal to the top element of the c-semiring $\top$ represents a no-attack relation between two arguments.

In Def. 16 we define the attack strength for a set of arguments that attacks an argument, a different set of arguments, or an argument that attacks a set of arguments; the former and the latter are what we need to define w-defence. We use $\bigotimes$ to indicate the set-wise version of $\otimes$:

Definition 16 (Attacks to/from sets of arguments). *Given a $WAAF_{\mathbb{S}}$, $WF = \langle \mathcal{A}_{rgs}, R, W, \mathbb{S} \rangle$,*

- *a set of arguments $\mathcal{B}$ attacks an argument a with a weight of $k \in S$ if $W(\mathcal{B}, a) = \bigotimes_{b \in \mathcal{B}} W(b, a) = k$.*
- *an argument a attacks a set of arguments $\mathcal{B}$ with a weight of $k \in S$ if $W(a, \mathcal{B}) = \bigotimes_{b \in \mathcal{B}} W(a, b) = k$.*
- *a set of arguments $\mathcal{B}$ attacks a set of arguments $\mathcal{D}$ with a weight $k \in S$ if $W(\mathcal{B}, \mathcal{D}) = \bigotimes_{b \in \mathcal{B}, d \in \mathcal{D}} W(b, d) = k$.*

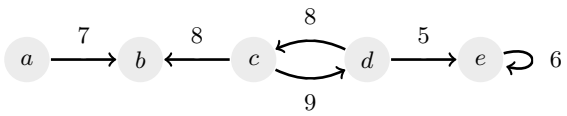

Fig. 2. An example of WAAF, adding weights to Fig. 1.

For example, looking at Fig. 2 we have that $W(\{a,c\},b) = 15$, $W(c,\{b,d\}) = 17$, and $W(\{a,c\},\{b,d\}) = 24$.

We are now ready to define our version of weighted defence, i.e., w-defence:

Definition 17 (w-defence ($\mathbb{D}_w$) [6]). *Given a $WAAF_{\mathbb{S}}$, $WF = \langle \mathcal{A}_{rgs}, R, W, S\rangle$, $\mathcal{B} \subseteq \mathcal{A}_{rgs}$ w-defends $b \in \mathcal{A}_{rgs}$ from $a \in \mathcal{A}_{rgs}$ s.t. $R(a,b)$, iff $W(a, \mathcal{B} \cup \{b\}) \geqslant_{\mathbb{S}} W(\mathcal{B}, a)$; $\mathcal{B}$ w-defends b iff it defends b from any a s.t. $R(a,b)$.*

Definition 17 can be relaxed to γ-defence, which is parametrised on a threshold-value γ: such γ is used to consider arguments that are not "fully" w-defended [6], i.e., for which $W(a, \mathcal{B} \cup \{b\}) \not\geqslant_{\mathbb{S}} W(\mathcal{B}, a)$:

Definition 18 (γ-defence ($\mathbb{D}_\gamma$) [2]). *Given $\langle \mathcal{A}_{rgs}, R, W, \mathbb{S} = \langle S, \oplus, \otimes, \bot, \top\rangle\rangle$ and $\gamma \in S$, $\mathcal{B} \subseteq \mathcal{A}_{rgs}$ γ-defends $b \in \mathcal{A}_{rgs}$ iff $\forall a \in \mathcal{A}_{rgs}$ such that $R(a,b)$ we have that $W(\mathcal{B}, a) \neq \top$ and*

$$\Big(W(a, \mathcal{B} \cup \{b\}) \oslash W(\mathcal{B}, a)\Big) \geqslant_{\mathbb{S}} \gamma$$

Considering the example in Fig. 2, for instance $\{d\}$ 1-defends d from c (i.e., $\gamma = 1$):

$$\Big(W(c,\{d\}) - W(\{d\},c)\Big) \leqslant 1,$$

since $9 - 8 = 1$ and $1 \leqslant 1$.

Having defined $\mathbb{D}_\gamma$ as a relaxation of $\mathbb{D}_w$, we can redefine all the classical semantics in Sec. 2.1 by exploiting both the notion of *i)* an inconsistency amount α inside an extension (to be tolerated), and *ii)* $\mathbb{D}_\gamma$.

In Def. 19 we redefine conflict-free sets: conflicts can be now part of the solution up to a cost-threshold α. They are now called α-conflict-free sets:

Definition 19. (α-conflict-free sets) *Given a $WAAF_{\mathbb{S}}$, $WF = \langle \mathcal{A}_{rgs}, R, W, \mathbb{S}\rangle$, a set of arguments $\mathcal{B} \subseteq \mathcal{A}_{rgs}$ is α-conflict-free iff $W(\mathcal{B}, \mathcal{B}) \geqslant_{\mathbb{S}} \alpha$.*

With respect to the $WAAF_{\mathbb{S}}$ in Fig. 2, while the set $\{a,b,c\}$ is not conflict-free in the crisp version of the problem (since it includes the attacks between a and b, and between c and b), $\{a,b,c\}$ is instead 15-conflict-free because $W(a,b) + W(c,b) = 15$. For instance, $\{a,b,c\}$ is also 16-conflict-free because it is a 15-conflict-free ($15 \geqslant_{\mathbb{S}} 16$ in the weighted semiring).[5] Therefore, in α-conflict-free extensions we tolerate an internal inconsistency-amount better than α.

$\mathbb{D}_\gamma$ leads to the definition of α^γ-admissible sets.

Definition 20. (α^γ-admissible sets) *Given $WF = \langle \mathcal{A}_{rgs}, R, W, \mathbb{S}\rangle$, an α-conflict-free set $\mathcal{B} \subseteq \mathcal{A}_{rgs}$ is α^γ-admissible iff the arguments in $\mathcal{B}$ are γ-defended by $\mathcal{B}$ from the arguments in $\mathcal{A}_{rgs} \backslash \mathcal{B}$.*

[5] In the weighted semiring, $\leqslant_{\mathbb{S}}$ is equivalent to $\geqslant$ over Real numbers, while in the probabilistic and fuzzy semirings, $\geqslant_{\mathbb{S}}$ corresponds to $\geqslant$ over Real numbers in the interval [0..1].

For instance, the set $\{d, e\}$ in Fig. 2 is 11^1-admissible, since it is 11-conflict-free, and d defends itself (and the whole $\{d, e\}$) from c by paying a penalty of $9 - 8 \leqslant 1$.

Four semantics that refine α^γ-admissibility are introduced from Def. 21 to Def. 23:

Definition 21 (α^γ-complete). *Given $\langle \mathcal{A}_{rgs}, R, W, \mathbb{S} \rangle$, an α^γ-admissible $\mathcal{B} \subseteq \mathcal{A}_{rgs}$ is α^γ-complete iff each argument $b \in \mathcal{A}_{rgs}$ that is γ-defended by $\mathcal{B}$ and s.t. $W(\mathcal{B} \cup \{b\}, \mathcal{B} \cup \{b\}) \geqslant_{\mathbb{S}} \alpha$ is in $\mathcal{B}$ (i.e., $b \in \mathcal{B}$).*

For instance, the set $\{a, d\}$ in Fig. 2 is complete according to [12] (i.e., not considering weights), but it is not 0^0-complete because d is not able to w-defend $\{a, d\}$ from c. However, $\{a, d\}$ is 0^1-complete by considering the weighted semiring and allowing $\gamma = 1$. Note that $\{a, d\}$ also 1-defends argument e, but $\{a, d, e\}$ is not 0^1-complete if we keep $\alpha = 0$: bringing e inside would lead to an internal conflict of $W(d, e) + W(e, e) = 11$.

Definition 22 (α^γ-preferred). *An α^γ-preferred extension is a maximal (with respect to set inclusion) α^γ-admissible subset of $\mathcal{A}_{rgs}$.*

Still considering Fig. 2, $\{a, c\}$ and $\{a, d\}$ are the two preferred extensions according to [12] (i.e., not considering weights). However, $\{a, c\}$ is the only 0^0-preferred extension, while $\{\{a, c\}, \{a, d\}\}$ is the set of 0^1-preferred extensions.

Finally, we define α^γ-stable semantics.

Definition 23. (α^γ-stable semantics) *Given $\langle \mathcal{A}_{rgs}, R, W, \mathbb{S} \rangle$, an α^γ-admissible set $\mathcal{B}$ is also an α^γ-stable extension iff $\forall a \notin \mathcal{B}, \exists b \in \mathcal{B}. W(b, a) \neq \top$, and $\mathcal{B} \cup \{a\}$ is not α^γ-admissible.*

For example, considering the problem in Fig. 2 as unweighted (i.e., as a classical AAF), the set $\{a, d\}$ corresponds to the only stable extension. However, considering weights, this set is not 0^0-stable, because $W(c, d) \not\geqslant_{\mathbb{S}} W(\{a, d\}, c)$, i.e., $9 \not\leqslant 8$. However, it is 0^1-stable, since $W(c, d) \oslash W(\{a, d\}, c) = 9 - 8 \leqslant 1$ satisfies $\gamma = 1$. Thus, in such example there is no 0^0-stable extension.

When $\alpha = \top$ and $\gamma = \top$, $\top^\top$-semantics are equivalent to their classic counterpart in Sec. 2.1 [2].

3 The Library

ConArgLib is available for Linux 64bit platform, and it has been compiled with gcc 4.9.2 and glibc 2.19 (using Debian 8). The pre-compiled package is freely available online.[6] ConArgLib sits on the layer offered by Gecode 5.1.0, which implements the search engine. Given a semantics σ, and the set of extensions $\xi_\sigma(F)$ that satisfy σ over a framework F, the decision problems solved by ConArgLib are:

[6] `http://www.dmi.unipg.it/conarg/`.

Enumeration. Given $F = \langle \mathcal{A}_{rgs}, R \rangle$ and a semantics σ, this problem consists in enumerating all $\mathcal{B} \in \xi_\sigma(F)$.

Credulous Acceptance. Given $F = \langle \mathcal{A}_{rgs}, R \rangle$, a semantics σ, and an argument $a \in \mathcal{A}_{rgs}$, a is credulously accepted if $\exists \mathcal{B} \in \xi_\sigma(F), a \in \mathcal{B}$.

Sceptical Acceptance. Given $F = \langle \mathcal{A}_{rgs}, R \rangle$, a semantics σ, and an argument $a \in \mathcal{A}_{rgs}$, a is sceptically accepted if $\forall \mathcal{B} \in \xi_\sigma(F), a \in \mathcal{B}$.

Verification. Given $F = \langle \mathcal{A}_{rgs}, R \rangle$, a semantics σ, and a subset of arguments $\mathcal{B}$, the verification of $\mathcal{B}$ consists in checking if $\mathcal{B} \in \xi_\sigma(F)$.

Existence. Given $F = \langle \mathcal{A}_{rgs}, R \rangle$ and a semantics σ, this problem corresponds to checking whether $\xi_\sigma(F) \neq \emptyset$.

Non-emptiness. Given $F = \langle \mathcal{A}_{rgs}, R \rangle$ and a semantics σ, this problem corresponds to checking whether there is any $\mathcal{B} \in \xi_\sigma(F)$ for which $\mathcal{B} \neq \emptyset$.

The complexity of extension enumeration is reported in [17]. The complexity of the other previouslt presented problems, e.g., credulous/sceptical acceptance, can be instead found in [11], [16], and [15].

We now introduce the functions we can use to solve each of the six problems previously introduced. Besides "Admissible", each of the four following functions can also be called by replacing it with "ConflictFree", "Complete", "Preferred", "SemiStable", "Stable", "Grounded", "Ideal", "Stage", and "Eager"; hence the library implements all the classical semantics in Sec. 2.1:

```
set<tSolution *> *GetAdmissibleEnum();
string GetAdmissibleCred(string &sArg);
string GetAdmissibleScept(string &sArg);
bool IsAdmissible(vector<string> &vecExtToCheck);
set<tSolution *> *GetAdmissibleExist();
set<tSolution *> *NonEmptinessAdmissible();
```

In addition, other functions have been coded to solve the relaxed framework presented in [2, 6]. It is then possible to pass as input parameters the chosen semiring (see in the following for implemented ones), the amount of internal relaxation (*dAlpha*), and the amount of relaxation on the defence (*dGamma*). All the following functions are presented for the admissible semantics, but by replacing "Admissible" sub-strings with "ConflictFree", "Complete", "Preferred", "Stable", or "Grounded", it is possible to find a solution for the corresponding semantics (all the semantics in Sec. 2.3):

```
set<tSolution *> *GetAdmissibleEnum(double dAlpha, double
    dGamma);
set<tSolution *> *GetAdmissibleExist(double dAlpha, double
    dGamma);
string GetAdmissibleCred(string &sArg, double dAlpha, double
    dGamma);
string GetAdmissibleScept(string &sArg, double dAlpha, double
    dGamma);
bool IsAdmissible(vector<string> &vecExtToCheck, double dAlpha
    , double dGamma);
set<tSolution *> *NonEmptinessAdmissible(double dAlpha, double
     dGamma);
```

The following semirings are at the moment defined and implemented in ConArgLib: $\langle\{false, true\}, \vee, \wedge, false\rangle$, $\langle[0,1], \max, \min, 0, 1\rangle$, and $\langle\mathbb{R}^+ \cup \{+\infty\}, min, +, +\infty, 0\rangle$.

```
#define CL_SRTYPE_BOOLEAN 0
#define CL_SRTYPE_FUZZY 1
#define CL_SRTYPE_WEIGHTED 2
```

The following calls are instead used to manage the library:

```
bool LoadFile(string &sInputFileName);
void PrintSol(set<tSolution *> *psetSol);
void FreeSol(set<tSolution *> *psetSol);
bool EngineIsInitialized();
int GetSolutionsCount(set<tSolution *> *psetSol);
double GetFuzzyBestValue();
double GetFuzzyWorstValue();
double GetWeightedBestValue();
double GetWeightedWorstValue();
```

LoadFile is used to load the description of an AAF by passing a file path. For classical AAFs we use ASPARTIX format (*apx*), which corresponds to a text file with a list of arguments, expressed as "*arg(argumentName).*", and a list of attacks between them, *e.g.*, "*att(argument1Name, argumentName2).*". To represent WAAFs as proposed in Sec. 2.3, we have modified the apx format into *wapx*: the list of arguments is expressed in the same way, while attacks are associated with a score, i.e., *att(argumentName1, argumentName2):-score.*

PrintSol is used to print all the solutions: hence, it is meaningful for the enumeration, existence, and non-emptiness problems.

FreeSol deallocates the passed set of solutions from dynamic memory.

EngineIsInitialized checks if the search engine has been correctly initialised; for instance, it checks whether a semiring type has been already selected.

GetSolutionsCount returns the number of solutions for the last solved problem (it is meaningful in case of enumeration).

GetFuzzyBestValue and **GetFuzzyWorstValue** can be used by the programmer to respectively retrieve the top and bottom elements in fuzzy semiring (see Sec. 2.2). The same holds for **GetWeightedBestValue** and **GetWeightedWorstValue**, w.r.t. the weighted semiring.

The following group of functions collects the functionalities to configure how solutions are found and printed. **SetShowSolutions** sets if the set of solutions is printed on the screen or silenced, besides being saved in memory.

SetRequestedSolutions stops the search as soon as a given number (input parameter) of solutions is found, while **SetRequestedAllSolutions** returns all the found solutions (these functions are meaningful only for the enumeration problem).

SetStatistics prints the search statistics obtained from Gecode: specifically, it returns the number of failed nodes in search tree, the number of nodes expanded, the maximum depth during search, the number of restarts, and, finally, the number of posted no-goods.

SetProboOutput sets the output of requested problems to be compliant with *Probo*[7], which is the benchmark framework used for computational argumentation competitions (i.e., ICCMA15 and ICCMA17).

SetPercentOutput also prints the rate of appearance (as percentage) of each argument in the current solutions. It can be used in combination with solving the enumeration problem over any semantics, in order to understand the degree of acceptability of each argument.

SetSilentMode trims part of the output, *e.g.*, removes messages concerning the kind of problem being solved.

All these function are also replicated with "Get" instead of "Set" in order to know how parameters are currently set. Finally, **ResetDefaults** restores them to the initial values (*e.g.*, output is not silenced by default).

```
void SetShowSolutions(bool bFlag);
void SetRequestedSolutions(long lRS);
void SetRequestedAllSolutions();
void SetStatistics(bool bStat);
void SetProboOutput(bool bProbo);
void SetPercentOutput(bool bPercent);
void SetSilentMode(bool bSilentMode);
void SetShowSolutionsCount(bool bShowSolutionsCount);
bool GetShowSolutions();
long GetRequestedSolutions();
bool GetStatistics();
bool GetProboOutput();
bool GetPercentOutput();
bool GetSilentMode();
void ResetDefaults();
```

An example of a program using ConArgLib is shown in Fig. 3: the code finds all the admissible sets and prints them on the screen. The example in Fig. 4 shows instead how to verify that the set $\{a, d\}$ is a 0^1-stable extension (see Def. 23) according to the weighted semiring.

4 Implementation Details

In this section we overview some technical details of the search engine, that is how we have implemented constraints and heuristics. Gecode supports the development of new constraints, branching strategies, and search engines. Constraints can be defined over Integers, Booleans, Sets, and Floats. In the following, *args[]* is the array of Boolean variables (i.e., instances of the class *BoolVar*) that represents the whole set of arguments $\mathcal{A}_{rgs}$; Boolean variables can only take 0 or 1 values (i.e., two integer values). An array of Boolean variables can be created with *BoolVarArray args[](space,* $|\mathcal{A}_{rgs}|$*, 0, 1)*, where *space* is the associated search-space.

[7] `https://sourceforge.net/projects/probo/`.

```
#include "AFEngine.h"
AFEngine *af = new AFEngine();
string sFileName1 = "file.apx";
af->LoadFile(sFileName1);
set<tSolution *> *tsetSol = af->GetAdmissibleEnum();
af->PrintSol(tsetSol);
cout << af->GetSolutionsCount(tsetSol) << "solution(s)
    found" << endl;
af->FreeSol(tsetSol);
```

Fig. 3. An example of a small program created by using ConArgLib. Function declarations are imported from *AFEngine.h*. Then a new engine is created, and a framework is imported from file. This piece of code finds and prints all the admissible sets in that framework. The output printed on the screen for the example in Fig. 1 is $[[c],[a,c],[a,d],[d],[],[a]]$ *6 solution(s) found.*

```
#include "AFEngine.h"
AFEngine *af = new AFEngine();
string sFileName2 = "file.wapx";
af->LoadFile(sFileName);
vector<string> vecArgs;
vecArgs.push_back("a");
vecArgs.push_back("d");
af->SetSemiringType(CL_SRTYPE_WEIGHTED);
if(af->IsStable(vecArgs, 0, 1))
cout << "{a,d} is a stable extension";
else
cout << "{a,d} is not a stable extension";
```

Fig. 4. A piece of code that verifies if $\{a,d\}$ is a 0^1-stable extension, using the weighted semiring; the answer is positive on the example in Fig. 1.

Constraints for modelling conflict-free-sets are based on the first order logic formula $\forall a,b \in \mathcal{A}_{arg}$ s.t. $R(a,b)$, then $a \Rightarrow \neg b$ (>> is the implication operator in Gecode):

```
rel(space, args[i] >> !args[j])
```

The procedure to enforce the constraints for modelling admissibility first sets conflict-free constraints, and then checks if at least one defender (c) of each attacker (b) of a is true, i.e., taken in an extension together with a: $\forall a,b,c \in \mathcal{A}_{arg}$ then $a \Rightarrow \bigwedge_{(b,a)\in R} (\bigvee_{(c,b)\in R} c)$. In this case, we deal with two different cases: when an argument *args[i]* is attacked but has no defender, then it has not to be taken (*IRT_EQ* imposes equality, with 0 in this case):

```
rel(space, args[i], IRT_EQ, 0, IPL_SPEED)
```

Otherwise, if an argument has at least one defender, then we use the *clause* constraint, where c is a *Boolvar* array that represents all the arguments c_i that defend a from each attacker b, and such a clause constraints models $\neg a \vee \bigvee_{c_i \in c} c_i$

```
clause(space, BOT_AND, a, c, 0)
```

IPL_SPEED is a Gecode flag set to optimise propagation for execution performance, at the expense of memory use.

Constraints for stable extensions are obtained by imposing admissible constraints and enforcing the formula $\forall a \in \mathcal{A}_{rgs}$, then $a \Leftrightarrow \bigwedge_{(b,a) \in R} \neg b$. This is modelled as

```
rel(space, m_bvaArgs[i], IRT_EQ, 1)
```

in case *args[i]* is never attacked (then it has to be taken). Otherwise, if *args[i]* has one or more attackers the constraint is

```
linear(space, a, x, IRT_GR, 0);
```

Given $a = [args[i], b_1, \ldots, b_n]$, where $b_i, \ldots b_n$ are the n attackers of *args[i]*, the previous constraint enforces $args[i] + x_1 \cdot b_1 + \cdots + x_n \cdot b_n > 0$; in this case, x is a vector of weights all set to 1 (with size $n + 1$). Hence, either an argument a or at least one of its attackers b_i have to be true.

We now introduce how we enumerate and verify preferred extensions. First of all, we set the constraints to find admissible sets, including constraints for conflict-freedom. Afterwards, we use the following branching strategy:

```
branch(space, args, INT_VAR_DEGREE_MAX(), INT_VALUES_MAX());
```

During search, at each step such a strategy selects the variable with the highest degree (i.e., the number of dependant propagators) and tries to assign it to true. This last condition automatically leads to finding a preferred extension:

Proposition 1 (Branching for preferred). *By imposing only constraints to find admissible sets, a branching strategy* INT_VALUES_MAX *always terminates with a preferred extension.*

This approach incrementally builds an extension by adding as more ("admissible") arguments as possible: eventually, it is not possible to find an admissible set that includes it.

In order to enumerate all preferred extensions, it is enough to *i)* launch the search using Prop. 1, *ii)* record the obtained solution, *iii)* remove it from the set of possible future solutions, and *iv)* restart search. The i-iv loop is terminated when no further solution is found. To verify if a given set $\mathcal{B}$ is a preferred extension, we set admissible constraints, and then we use Prop. 1 by first assigning to true all the arguments in $\mathcal{B}$. As soon as we find a solution different from $\mathcal{B}$, then we obtain that $\mathcal{B}$ is not a preferred extension.

4.1 Constraints for WAAFs

Constraints for WAAFs in Sec. 2.3 are similar to the ones proposed in this section for classical Abstract Argumentation [12]: in addition, we add some more constraints to limit the internal inconsistency and the difference between attack and defence weights to be less than resp. α and γ. In the following piece of code, the first constraint binds *extensionCost* to the aggregation (parametric w.r.t. the chosen semiring) of the weights of all attacks "active" in the considered set of arguments. Then, still depending on the semiring, such a value is compared against α:

```
rel(space, extensionCost == semiringTimesOperation(
    activeAttacks));
if(m_struct->GetSemiringType() == WEIGHTED)
rel(space, extensionCost, FRT_LQ, alpha);
if(m_struct->GetSemiringType() == FUZZY)
rel(space, extensionCost, FRT_GQ, alpha);
```

Concerning γ-defence (see Def. 18), the $\times$ of all the attackers (A) and defenders (B) of an argument is computed and compared against γ. *Expr* posts a Boolean expression and returns its value, which is then checked to be true:

```
expr(space, SemiringXAIsGammaWorseEqualThanXB(B, A));
if(m_struct->GetSemiringType() == WEIGHTED)
expr(space, (SemiringXOperation(B) - SemiringXOperation(A)) <=
    gamma);
if(m_struct->GetSemiringType() == FUZZY)
expr(space, (SemiringXOperation(B) - SemiringXOperation(A)) >=
    gamma);
```

5 Conclusion

We have presented ConArgLib, a C++ software library designed to offer to the end user functions to solve classical tasks in Abstract Argumentation: extension enumeration, existence, and verification, and credulous/sceptical acceptance of arguments. The engine at the core of the library offers good performance, and is the first attempt to provide such functionalities as a software library.

In the future, we plan to extend the library by providing higher-level functions, e.g., to check if two frameworks are equivalent on some given semantics. Moreover, we would like to implement a recent different definition of grounded semantics for semiring-based WAAFs [7], which, contrarily to all the other proposals in the literature for Weighted Argumentation Frameworks, always returns one extension as result, as in the classical formulation [12]. Finally, we will implement coalition-based extensions [4] and voting systems as in [1].

References

1. Benedetti, I., Bistarelli, S.: From argumentation frameworks to voting systems and back. Fundam. Inform. 150(1), 25–48 (2017)

2. Bistarelli, S., Rossi, F., Santini, F.: A relaxation of internal conflict and defence in weighted argumentation frameworks. In: European Conference on Logics in Artificial Intelligence, JELIA. LNCS, vol. 10021, pp. 127–143. Springer (2016)
3. Bistarelli, S., Santini, F.: Conarg: A constraint-based computational framework for argumentation systems. In: International Conference on Tools with Artificial Intelligence. pp. 605–612. ICTAI '11, IEEE Computer Society (2011)
4. Bistarelli, S., Santini, F.: Coalitions of arguments: An approach with constraint programming. Fundamenta Informaticae 124(4), 383–401 (2013)
5. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint satisfaction and optimization. Journal of the ACM 44(2), 201–236 (1997)
6. Bistarelli, S., Rossi, F., Santini, F.: A collective defence against grouped attacks for weighted abstract argumentation frameworks. In: Florida Artificial Intelligence Research Society Conference, FLAIRS. pp. 638–643. AAAI Press (2016)
7. Bistarelli, S., Santini, F.: A Hasse diagram for weighted sceptical semantics with a unique-status grounded semantics. In: Logic Programming and Nonmonotonic Reasoning, LPNMR. LNCS, vol. 10377, pp. 49–56. Springer (2017)
8. Blyth, T.S., Janowitz, M.F.: Residuation theory, vol. 102. Pergamon press Oxford (1972)
9. Caminada, M.: Comparing two unique extension semantics for formal argumentation: ideal and eager. In: Belgian-Dutch conference on artificial intelligence (BNAIC). pp. 81–87 (2007)
10. Caminada, M.: Semi-stable semantics. In: Computational Models of Argument: Proceedings of COMMA 2006. Frontiers in Artificial Intelligence and Applications, vol. 144, pp. 121–130. IOS Press (2006)
11. Charwat, G., Dvořák, W., Gaggl, S.A., Wallner, J.P., Woltran, S.: Methods for solving reasoning problems in abstract argumentation: A survey. Artificial Intelligence 220(0), 28 – 63 (2015)
12. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence 77(2), 321–357 (1995)
13. Dung, P.M., Mancarella, P., Toni, F.: A dialectic procedure for sceptical, assumption-based argumentation. In: Proceedings of the 2006 conference on Computational Models of Argument (COMMA). pp. 145–156. IOS Press (2006)
14. Dunne, P.E., Hunter, A., McBurney, P., Parsons, S., Wooldridge, M.: Weighted argument systems: Basic definitions, algorithms, and complexity results. Artif. Intell. 175(2), 457–486 (2011)
15. Dunne, P.E.: The computational complexity of ideal semantics. Artif. Intell. 173(18), 1559–1591 (2009)
16. Dvorák, W.: Computational Aspects of Abstract Argumentation. Ph.D. thesis, Technische Universität Wien (2012), `http://permalink.obvsg.at/AC07812708`
17. Kröll, M., Pichler, R., Woltran, S.: On the complexity of enumerating the extensions of abstract argumentation frameworks. In: International Joint Conference on Artificial Intelligence, IJCAI. pp. 1145–1152. ijcai.org (2017)
18. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming. Elsevier Science Inc., New York, NY, USA (2006)
19. Thimm, M., Villata, S., Cerutti, F., Oren, N., Strass, H., Vallati, M.: Summary report of the first international competition on computational models of argumentation. AI Magazine 37(1), 102 (2016)
20. Verheij, B.: Two approaches to dialectical argumentation: admissible sets and argumentation stages. Dutch Conference on Artificial Intelligence 96, 357–368 (1996)