Computing the Labellings of Higher-Order Abstract Argumentation Frameworks

Sylvie Doutre, Marie-Christine Lagasquie-Schiex

Computing the Labellings of Higher-Order Abstract Argumentation Frameworks

Sylvie Doutre[1], Marie-Christine Lagasquie-Schiex[2,**]

[1]IRIT, Toulouse 1 University, France
[2]IRIT, Toulouse 3 University, France

Abstract

The topic of this work is related to a computational issue concerning an enriched abstract argumentation framework called RAF ("Recursive Argumentation Framework"). A RAF is composed of a set of arguments and a binary relation modelling the attacks as in Dung's framework. The main difference between Dung's framework and a RAF is the fact that a RAF is able to take into account *higher-order* interactions (*i.e.* an attack can target an attack and not only an argument). Since this kind of framework is relatively recent, the efficient computation of the main semantics remains an open question. In this paper, we propose one of the first algorithms dedicated to this issue. We prove the soundness and completeness of this algorithm.

Keywords

Abstract Argumentation, Higher-Order Interactions, Algorithms

1. Motivation

Argumentation, by considering arguments and their interactions, is a way of reasoning that has proven successful in many contexts, multi-agent applications for instance (e.g. [1]). Considering a formal representation of this reasoning model, argumentation frameworks with higher-order attacks (e.g. [2, 3, 4, 5, 6]) are a rich extension of the classical Argumentation Framework (AF) by [7]: not only they consider arguments and attacks between arguments, but also attacks on attacks (see for instance [5, 6]). Among these frameworks, the Recursive Argumentation Framework (RAF) by [8] proposes a direct approach regarding acceptability, which outputs sets of arguments and/or attacks (defined under the notion of structure), keeping the full expressiveness of higher-order attacks. A correspondence between Dung's extension-based semantics for AFs and structure-based semantics of RAFs without any attack on attacks has been shown in [8], proving that RAFs are a conservative generalisation of AFs. This characteristic makes RAFs particularly interesting to consider.

The computation of semantics of RAFs has not been addressed so far but a simple way to do so can be to extend what is done for AFs: some of the most efficient algorithms for computing AF semantics are based on a cutting of the AF and then on a distributed and parallel

SAFA'22: Fourth International Workshop on Systems and Algorithms for Formal Argumentation 2022, September 13, 2022, Cardiff, Wales, United Kingdom

**Corresponding author.
doutre@irit.fr (S. Doutre); lagasq@irit.fr (M. Lagasquie-Schiex)

computation (see [9, 10, 11, 12]) using the notion of AF labellings (e.g. [13, 14])[1] and the fact that such semantics are decomposable (see [16]). Indeed, RAF labellings already exist and classical decision problems for AFs were also adapted to RAFs with an interesting result (see [17]): even if the expressive power of the frameworks with higher-order attacks is higher, the complexity of their decision problems remains the same as in an AF. Moreover, it has been proven in [18] that RAF semantics, as AF semantics, are decomposable. Thus, following the line of the *AFDivider* algorithm designed for AFs [12], all the mandatory elements are now present for the definition of some efficient algorithms for computing RAF labellings using a distributed and parallel method; this is the topic of the present paper.

The paper is organised as follows: Recursive Argumentation Frameworks (RAFs) and their semantics are recalled in Section 2. Section 3 gives some additional definitions mandatory before the presentation of the algorithm itself in Sections 4 and 5. An example of a clustering method is provided in Section 6. Section 7 draws conclusions and opens future perspectives. The proofs of the soundness and completeness of the approach can be found in [19].

2. Background on Recursive Argumentation Frameworks

Higher-order attacks (that is, possibly targeting attacks as well as arguments) have been introduced in [2] then developed in several papers among which one can cite the AFRA (Argumentation Framework with Recursive Attacks) approach described in [6] and the RAF (Recursive Argumentation Framework) approach introduced in [8]. RAF and AFRA differ on the way in which these attacks are handled despite the fact that there is no difference in the structure of the graph. This paper is concerned with the RAF approach. This section recalls its main definitions: basics, labellings and decomposability of the semantics.

Definition 1. *A* Recursive Argumentation Framework (RAF) $\mathcal{RAF} = \langle A, K, s, t \rangle$ *is a quadruple where A and K are (possibly infinite) disjoint sets respectively representing* arguments *and* attack *names, and where* $s : K \to A$ *and* $t : K \to A \cup K$ *are functions respectively mapping each attack name to its* source *and to its* target.

Figure 1 shows an example of a RAF. There are two different possibilities for defining the semantics of a RAF: either by selecting some specific structures (a pair composed of a set of arguments and a set of attacks) [8] or by using labellings [17].[2] Here we only present the latter approach.[3]

Definition 2. *Let* $\mathcal{RAF} = \langle A, K, s, t \rangle$ *be a recursive argumentation framework. A RAF labelling is a total function* $\mathcal{L} : A \cup K \to \{\mathtt{in}, \mathtt{out}, \mathtt{und}\}$. *We define* $\mathtt{in}(\mathcal{L})$ *(resp.* $\mathtt{out}(\mathcal{L})$, $\mathtt{und}(\mathcal{L})$*) as the set* $\{x \in A \cup K | \mathcal{L}(x) = \mathtt{in}$ *(resp.* out, und*)*$\}$.

$\mathcal{L}$ *is a* complete RAF labelling *iff it satisfies:* $\forall x \in (A \cup K)$,

[1] Whereas an extension assigns to its elements an accepted or a rejected status, a labelling considers a third status, undecided, which applies to arguments which are neither accepted, nor rejected. This enrichment has proven useful for the computation of acceptance statuses in AFs (see [15] for a survey).

[2] Relations between labelling-based semantics and structure-based semantics have been exhibited in [17].

[3] These labellings were called "structure labellings" in [17] and defined as a pair of sets (the labellings for arguments, the labellings for attacks). Here the definition and the name are simplified.

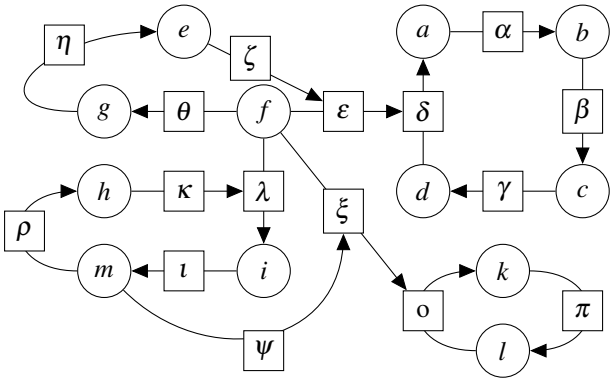


Figure 1: Running example: a RAF with argument names given in a circle and attack names in a square box

- $(\mathcal{L}(x) = \texttt{out}) \iff (\exists \alpha \in K \text{ s.t. } t(\alpha) = x,\ \mathcal{L}(\alpha) = \texttt{in} \text{ and } \mathcal{L}(s(\alpha)) = \texttt{in})$
- $(\mathcal{L}(x) = \texttt{in}) \iff (\forall \alpha \in K \text{ s.t. } t(\alpha) = x,\ \mathcal{L}(\alpha) = \texttt{out} \text{ or } \mathcal{L}(s(\alpha)) = \texttt{out})$

Let $x \in A \cup K$ be an element of $\mathcal{RAF}$. x is said to be legally labelled *in $\mathcal{L}$ iff $\mathcal{L}$ is a complete labelling and $x \in \texttt{und}(\mathcal{L})$ iff $((\nexists \alpha \in K$ s.t. $t(\alpha) = x$, $\mathcal{L}(\alpha) = \texttt{in}$ and $\mathcal{L}(s(\alpha)) = \texttt{in})$ and $(\exists \alpha \in K$ s.t. $t(\alpha) = x$, $\mathcal{L}(\alpha) \neq \texttt{out}$ and $\mathcal{L}(s(\alpha)) \neq \texttt{out}))$. $\mathcal{L}$ is said to be a* valid RAF labelling *if all its elements are legally labelled. A* preferred *(resp.* grounded*) labelling is a complete labelling that maximises (resp. minimises) the* in *elements. A* stable *labelling is a complete labelling for which there are no* und *elements.*

Regarding the RAF of Figure 1, Example 1 shows its grounded labelling.

Example 1. *The* grounded *labelling of the RAF illustrated in Figure 1 is:*

$$\mathcal{L}_{gr} = \left\{ \begin{array}{c} (a,\texttt{und}),(b,\texttt{und}),(c,\texttt{und}),(d,\texttt{und}),(e,\texttt{in}),(f,\texttt{in}),(g,\texttt{out}),(h,\texttt{und}),(i,\texttt{und}), \\ (k,\texttt{und}),(l,\texttt{und}),(m,\texttt{und})(\alpha,\texttt{in}),(\beta,\texttt{in}),(\gamma,\texttt{in}),(\delta,\texttt{in}),(\varepsilon,\texttt{out}),(\zeta,\texttt{in}),(\eta,\texttt{in}), \\ (\theta,\texttt{in}),(\iota,\texttt{in}),(\kappa,\texttt{in}),(\lambda,\texttt{und}),(\xi,\texttt{und}),(\mathrm{o},\texttt{und}),(\pi,\texttt{in}),(\rho,\texttt{in}),(\psi,\texttt{in}) \end{array} \right\}$$

In order to define an algorithm able to answer the enumeration problem[4] in the case of a RAF, similarly to the one given for an AF (*AFDivider* [12]), we must be able to split a RAF. Based on the notion introduced in [16], any AF can be split into several sub-frameworks by simply ignoring some attacks (that are always valid). Nevertheless, it is not the case for RAFs. Attacks, as arguments, can be labelled *in*, *out* or *und*. As a consequence, we cannot just ignore attacks to split a RAF. So, if we do not suppress attacks while splitting RAFs, we will have attacks without targets or without sources. Thus, the result of such a split does not produce a RAF but a *partial RAF*.

[4]The enumeration problem consists in enumerating all the solutions under a given semantics of a framework; see [20] for more details on this problem.

Definition 3. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF. A* partial RAF $\widetilde{\mathcal{RAF}} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t\rangle$ *of $\mathcal{RAF}$ is a tuple where $\tilde{A} \subseteq A$ (resp. $\tilde{K} \subseteq K$) is a set representing arguments (resp. attacks) and:*

- $\tilde{s} : \tilde{K} \to \{\texttt{true}, \texttt{false}\}$ *s.t.* $\forall \alpha \in \tilde{K}, \tilde{s}(\alpha) = \texttt{true}$ *if* $s(\alpha) \in \tilde{A}$ *otherwise* $\texttt{false}$
- $\tilde{t} : \tilde{K} \to \{\texttt{true}, \texttt{false}\}$ *s.t.* $\forall \alpha \in \tilde{K}, \tilde{t}(\alpha) = \texttt{true}$ *if* $t(\alpha) \in \tilde{A} \cup \tilde{K}$ *otherwise* $\texttt{false}$

Then, using the notion of partial RAF, a partition of a RAF can be defined:

Definition 4. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF. Let $\Omega = \{\omega_1, ..., \omega_n\}$ be a partition*[5] *of $(A \cup K)$. A* RAF partition *of $\mathcal{RAF}$ is a set of partial RAFs $\{\widetilde{\mathcal{RAF}}_1, ..., \widetilde{\mathcal{RAF}}_n\}$ s.t.: $\forall i, \widetilde{\mathcal{RAF}}_i = \langle \tilde{A}_i, \tilde{K}_i, \tilde{s}_i, \tilde{t}_i, s, t\rangle$ with $\tilde{A}_i = \omega_i \cap A$ and $\tilde{K}_i = \omega_i \cap K$.*

Considering a partial RAF implies to consider also its "inputs" (the elements that do not belong to the partial RAF but that can impact its labellings) and their labellings:

Definition 5. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\widetilde{\mathcal{RAF}} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t\rangle$ be a partial RAF of $\mathcal{RAF}$. The* input $\mathcal{I}$ *of $\widetilde{\mathcal{RAF}}$ is a tuple $\langle S^{inp}, Q^{inp}\rangle$ where: $S^{inp} = \{s(\alpha) \in (A \setminus \tilde{A}) | \alpha \in K$ and $t(\alpha) \in (\tilde{A} \cup \tilde{K})\}$ and $Q^{inp} = \{\alpha \in (K \setminus \tilde{K}) | t(\alpha) \in (\tilde{A} \cup \tilde{K})\}$. The tuple $\left\langle \widetilde{\mathcal{RAF}}, \mathcal{I}, \mathcal{L}^{inp}\right\rangle$ is called a* partial RAF with input*, where $\mathcal{L}^{inp}$ is a labelling of $\mathcal{I}$.*

Note that several partial RAFs with input can be built from a given partial RAF since several labellings can exist for its inputs.

The local function associates any partial RAF with input with a set of labellings:

Definition 6. *Let σ be a semantics. A local function $\mathcal{F}_\sigma^{raf}$ assigns to any partial RAF with input $\left\langle \widetilde{\mathcal{RAF}}, \mathcal{I}, \mathcal{L}^{inp}\right\rangle$ a (possibly empty) set of labellings of $\widetilde{\mathcal{RAF}}$ under σ, i.e. $\mathcal{F}_\sigma^{raf}(\widetilde{\mathcal{RAF}}, \mathcal{I}, \mathcal{L}^{inp}) \in 2^{\{\mathcal{L} | \mathcal{L} \text{ being any labelling over } \widetilde{\mathcal{RAF}}\}}$.*

The notion of semantics decomposability is thus as follows:

Definition 7. *A semantics σ is* full decomposable *iff there is a local function $\mathcal{F}_\sigma^{raf}$ s.t., for any RAF $\mathcal{RAF} = \langle A, K, s, t\rangle$ and any partition $\{\widetilde{\mathcal{RAF}}_1, ..., \widetilde{\mathcal{RAF}}_n\}$ of $\mathcal{RAF}$, the set of all possible labellings under the semantics σ of $\mathcal{RAF}$, denoted by $\mathcal{L}_\sigma(\mathcal{RAF})$, satisfies: $\mathcal{L}_\sigma(\mathcal{RAF}) = \{\mathcal{L}_1 \cup ... \cup \mathcal{L}_n | \forall i \in \{1, ..., n\}, \mathcal{L}_i \in \mathcal{F}_\sigma^{raf}(\widetilde{\mathcal{RAF}}_i, \mathcal{I}_i, \mathcal{L}_i^{inp})\}$ with $\mathcal{L}_i^{inp} = (\bigcup_{j \in \{1,...,n\} \text{ s.t. } j \neq i} \mathcal{L}_j) \downarrow_{\mathcal{I}_i}$.*[6]

A semantics σ is said to be top-down *(resp.* bottom-up*)* decomposable *iff:*

$$\mathcal{L}_\sigma(\mathcal{RAF}) \subseteq \text{ (resp. } \supseteq) \{\mathcal{L}_1 \cup ... \cup \mathcal{L}_n | \forall i \in \{1, ..., n\}, \mathcal{L}_i \in \mathcal{F}_\sigma^{raf}(\widetilde{\mathcal{RAF}}_i, \mathcal{I}_i, \mathcal{L}_i^{inp})\}$$

In [18], a specific RAF partition selector has been defined that produces a partition respecting the strongly connected components (SCC) of a RAF:[7]

[5] So the following property holds for Ω: $\forall (i, j) \in \{1, ..., n\}$ s.t. $i \neq j, \omega_i \cap \omega_j = \varnothing$ and $\bigcup_{i=1}^{n} \omega_i = A \cup K$.

[6] $\downarrow$ is the classical generic operator of restriction that allows the selection of a sub-part of a given "object" wrt to a given set of "elements". Here for instance, it produces the sub-part of the labellings concerning only the elements belonging to $\mathcal{I}_i$.

[7] See in [18] the details about the method for defining the SCC of a RAF.

Definition 8. *Let $\mathcal{RAF}$ be a RAF. Let $\mathcal{S}_{raf\text{-}USCC}$ be the RAF partition selector s.t.: $\mathcal{S}_{raf\text{-}USCC}(\mathcal{RAF}) = \{\Omega |\ \Omega$ is a partition of $\mathcal{RAF}$ and $\forall S \in SCCS_{raf}(\mathcal{RAF}), \exists \omega_i \in \Omega$ s.t. $S \subseteq \omega_i\}$. Let $S \subseteq A \cup K$ s.t. $S \in \mathcal{S}_{raf\text{-}USCC}(\mathcal{RAF})$, S is called an "$USCC_{raf}$".*

Then, in [18], the following proposition has been proven:

Proposition 1. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be any RAF. The semantics properties in Table 1 hold.*

	RAF semantics			
	complete	grounded	preferred	stable
Full decomposability	✓	×	×	✓
Top-down decomposability	✓	✓	✓	✓
Bottom-up decomposability	✓	×	×	✓
Full decomposability *w.r.t.* $\mathcal{S}_{raf\text{-}USCC}$ (so Top-down and Bottom-up decomposability)	✓	✓	✓	✓

"✓" (resp. "×") means that the semantics on the column has (resp. does not have) the property on the row.

Table 1
RAF Semantics decomposability properties

3. Some Preliminary Definitions

Before explaining our algorithm, some additional definitions are needed. First, Definition 9 gives an adaptation of the notion of *walk* and *path* to the case of partial RAF, then Definition 10 defines the notion of *connected elements* in a partial RAF.

Definition 9. *Let $\widetilde{\mathcal{RAF}} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t\rangle$ be a partial RAF and $e_1, e_n \in (\tilde{A} \cup \tilde{K})$ be elements of $\widetilde{\mathcal{RAF}}$. A* non-directed partial-RAF-walk *is a sequence $(e_1, ..., e_n)$ with $n \in \mathbb{N}^*$ and $\forall i, e_i \in (A \cup K)$ s.t.:*

- *If $n > 1$, $\forall i \in \{1, ..., n-1\}$, $e_i \in A \implies e_{i+1} \in K$ and $(\tilde{s}(e_{i+1}) = \texttt{true}$ and $e_i = s(e_{i+1})$ or $(\tilde{t}(e_{i+1}) = \texttt{true}$ and $e_i = t(e_{i+1})))$*
- *If $n > 1$, $\forall i \in \{1, ..., n-1\}$, $e_i \in K \implies (e_{i+1} \in A$ and $((\tilde{s}(e_i) = \texttt{true}$ and $s(e_i) = e_{i+1})$ or $(\tilde{t}(e_i) = \texttt{true}$ and $t(e_i) = e_{i+1})))$ or $(e_{i+1} \in K$ and $((\tilde{t}(e_{i+1}) = \texttt{true}$ and $e_i = t(e_{i+1}))$ or $(\tilde{t}(e_i) = \texttt{true}$ and $t(e_i) = e_{i+1})))$*

A non-directed partial-RAF-path *is a non-directed partial-RAF-walk in which all the elements are distinct.*

Considering the RAF given in Figure 1, $(f, \varepsilon, \delta, d)$ is an example of a non-directed partial-RAF-path.

Definition 10. *Let* $\widetilde{\mathcal{RAF}} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t \rangle$ *be a partial RAF.* $\widetilde{\mathcal{RAF}}$ *is a* connected *partial RAF if, for all distinct elements* $x_i \in \tilde{A} \cup \tilde{K}$ *and* $x_j \in \tilde{A} \cup \tilde{K}$*, there exists a non-directed partial-RAF-path* p *in* $\widetilde{\mathcal{RAF}}$ *s.t.* x_i *is the first element of* p *and* x_j *is the last element of* p*. Otherwise the partial RAF is* disconnected*.*

Considering the partial RAF obtained from the RAF given in Figure 1 by the removal of ξ, this partial RAF is disconnected since, for instance, none non-directed partial-RAF-path exists between f and k.

The very classical operator of restriction, denoted by $\downarrow$, can also be applied to partial RAF giving the partial RAF that is the restriction of a given partial RAF to a given set of elements (arguments and/or attacks). Then, using this operator, we can define the notion of "Partial RAF Connected Component":

Definition 11. *Let* $\widetilde{\mathcal{RAF}} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t \rangle$ *be a partial RAF. Let* $S \subseteq \tilde{A} \cup \tilde{K}$ *be a set of elements. The partial RAF* $\widetilde{\mathcal{RAF}}\downarrow_S$ *is a* connected component *of* $\widetilde{\mathcal{RAF}}$ *iff:* $\widetilde{\mathcal{RAF}}\downarrow_S$ *is connected and there exists no set* $S' \subseteq \tilde{A} \cup \tilde{K}$ *s.t.* $S \subset S'$ *and* $\widetilde{\mathcal{RAF}}\downarrow_{S'}$ *is connected.*

4. A Generic Algorithm for computing RAF semantics: Presentation by Example

The *RAFDivider* algorithm we propose in this paper is an adaptation of the *AFDivider* algorithm proposed in [12]. Similarly to *AFDivider*, it addresses the enumeration problem for the *complete*, *stable* and *preferred* semantics (finding all the possible solutions of a given semantics for a given RAF). It follows the same four major steps (see Figure 2):

1. A preprocessing on $\mathcal{RAF}$ removes "trivial" parts of it.
2. Clusters in $\mathcal{RAF}$ are identified.
3. The labellings under semantics σ in each cluster are computed in parallel.
4. The results of each cluster are reunified to get the labellings of $\mathcal{RAF}$.

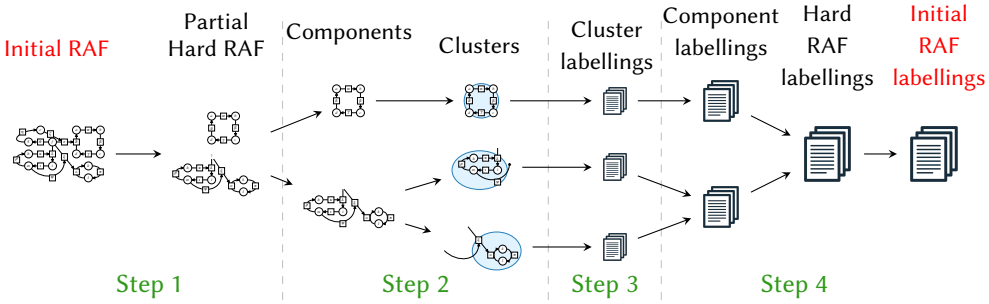


Figure 2: *RAFDivider* operating diagram

Before giving the formal definition of the algorithm, we describe its desired behaviour on a running example.

4.1. Preprocessing: Removing the Trivial Part

The first step is to identify the *trivial* part to remove. Similarly to an AF, it is built using the $\mathtt{in}$ and $\mathtt{out}$ elements produced by the *grounded* semantics. In the case of an AF, these elements are all removed. However, for a RAF, this complete removal is not possible. Consider for instance the grounded labelling given in Example 1. If we remove all elements that are not labelled $\mathtt{und}$ as it was done in the case of an AF, the resultant partial RAF would not capture the initial relations between the elements (for instance, the relation between the arguments a and b is expressed by the attack α labelled $\mathtt{in}$, so the removal of α is not possible). This leads to the following definition of the *RAF trivial part*:

Definition 12. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and let $\mathcal{L}_{gr}$ be the* grounded *labelling of $\mathcal{RAF}$. The* "trivial part" *of $\mathcal{RAF}$ is the structure of $\mathcal{RAF}$, denoted by $\mathcal{U}_{triv} = \langle S_{triv}, Q_{triv}\rangle$, with $S_{triv} = \{a \in A | a \notin \mathtt{und}(\mathcal{L}_{gr})\}$ and $Q_{triv} = \{\alpha \in K | \alpha \in \mathtt{out}(\mathcal{L}_{gr})$ or $(\alpha \in \mathtt{in}(\mathcal{L}_{gr})$ and $s(\alpha) \notin \mathtt{und}(\mathcal{L}_{gr}))\}$*

Example 2. *Following Example 1, the trivial part of $\mathcal{RAF}$ is: $\mathcal{U}_{triv} = \langle \{e, f, g\}, \{\varepsilon, \zeta, \eta, \theta\}\rangle$.*

Then the *partial hard RAF* and *partial hard RAF with input* are defined as follows:

Definition 13. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and let $\mathcal{U}_{triv} = \langle S_{triv}, Q_{triv}\rangle$ be the RAF trivial part of $\mathcal{RAF}$. The* partial hard RAF *of $\mathcal{RAF}$, denoted by $\widetilde{\mathcal{RAF}}_{hard}$, is defined as $\widetilde{\mathcal{RAF}}_{hard} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s_h, t_h\rangle$ with $\tilde{A} = A \setminus S_{triv}$, $\tilde{K} = K \setminus Q_{triv}$, $s_h : \tilde{K} \to A$ s.t. $\forall \alpha \in \tilde{K}, s_h(\alpha) = s(\alpha)$ and $t_h : \tilde{K} \to (A \cup K)$ s.t. $\forall \alpha \in \tilde{K}, t_h(\alpha) = t(\alpha)$ (\tilde{s} and \tilde{t} are defined as in Definition 3).*
The partial hard RAF with input *is $\left\langle \widetilde{\mathcal{RAF}}_{hard}, \mathcal{I} = \langle S^{inp}, Q^{inp}\rangle, \mathcal{L}^{inp}\right\rangle$, where $S^{inp} = \{s(\alpha) \in (A \setminus \tilde{A}) | \alpha \in \tilde{K}\}$, $Q^{inp} = \varnothing$ and $\mathcal{L}^{inp}$ is the grounded labelling of the elements in $\mathcal{I}$.*

Although a partial hard RAF is a partial RAF, we only consider the input labelling that coincides with the *grounded* labelling of the initial RAF. So the corresponding partial hard RAF with input is trivially *unique*, the labelling of its inputs being unique.

Example 3. *Figure 3 illustrates $\widetilde{\mathcal{RAF}}_{hard}$, the partial hard RAF corresponding to the RAF shown in Figure 1. In this partial hard RAF, two attacks have no source (λ and ξ). This partial hard RAF has one input; so $\mathcal{I} = \langle \{f\}, \varnothing\rangle$ and $\mathcal{L}^{inp} = \{(f, \mathtt{in})\}$.*

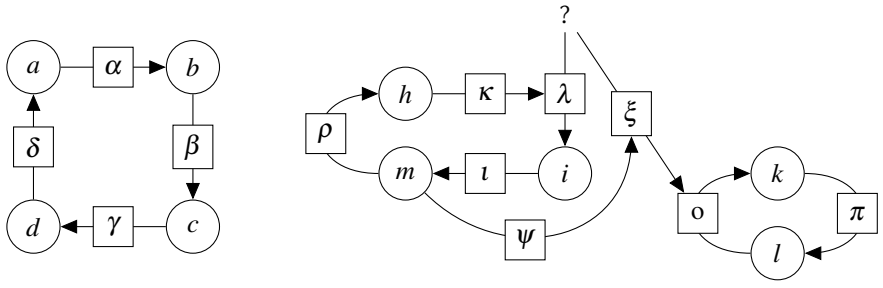
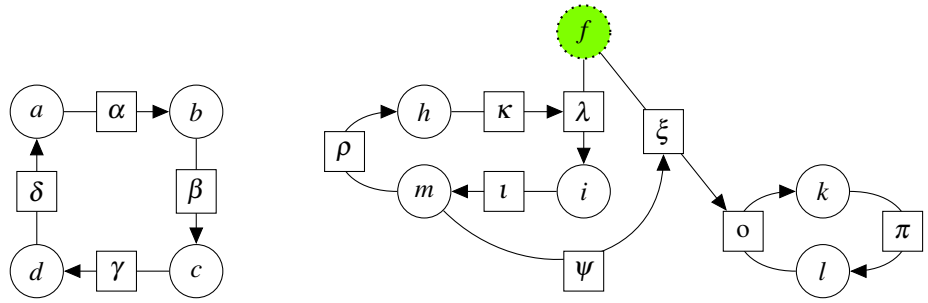


Figure 3: Partial Hard RAF (the attacks λ and ξ have no source)

For each input element, several cases have to be considered and this can be very time consuming. In order to avoid this cost for elements that are in the "trivial part", we simply cut that part from the

RAF and, only after that, look for clusters. So, given a RAF $\mathcal{RAF} = \langle A, K, s, t \rangle$, the *RAFDivider* algorithm starts by computing $\mathcal{L}_{gr}$, the *grounded* labelling of $\mathcal{RAF}$ and $\mathcal{U}_{triv}$ the trivial part corresponding to it. Once the trivial part has been computed, the algorithm removes it from $\mathcal{RAF}$ to produce $\widetilde{\mathcal{RAF}}_{hard}$ the partial hard RAF of $\mathcal{RAF}$, as well as $\widetilde{\mathcal{RAF}}_{hard}$ input elements $\mathcal{I}$ with their labelling issued from $\mathcal{L}_{gr}$. Then, if possible, $\widetilde{\mathcal{RAF}}_{hard}$ is split into several connected components (see Definition 10), producing the set *CCSet*. *CCSet* is a set of partial hard RAFs (with input).



(a) Component 1: $\widetilde{raf}_1$ (b) Component 2: $\widetilde{raf}_2$

Figure 4: The connected components of $\widetilde{\mathcal{RAF}}_{hard}$ with their inputs ($\widetilde{raf}_1$ has no input; $\widetilde{raf}_2$ has one input f)

Example 4. *Figure 4 illustrates the two connected components of $\widetilde{\mathcal{RAF}}_{hard}$, the partial hard RAF illustrated in Figure 3. The CCSet will then contain these two partial RAFs.*

4.2. Identifying Clusters

For each connected component, a clustering can be performed using any clustering method partitioning the partial RAF (even a random partition method). See in Section 6 such a method that returns the set of clusters identified, that is, a set of partial RAFs.

Example 5. *Following Example 4, let us consider that the chosen clustering method produces only one cluster for component 1. Let $\widetilde{raf}_2$ be the other component and let the following partition be the one produced by the chosen clustering method:*

$$\Omega = \{\omega_1 = \{h, i, m, \iota, \kappa, \lambda, \rho, \psi\}, \omega_2 = \{k, l, \xi, o, \pi\}\}$$

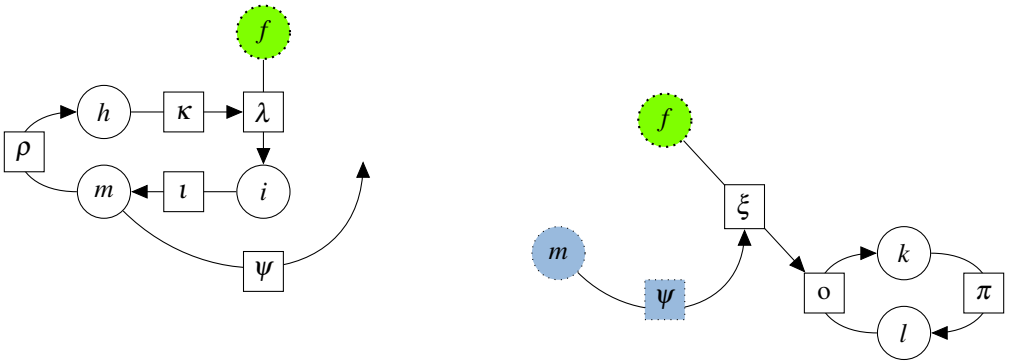
Figure 5 illustrates the partial RAFs corresponding to the partitioning of $\widetilde{raf}_2$, that is $\widetilde{raf}_2 \downarrow_{\omega_1}$[8] (also denoted by $\widetilde{raf}_{2.1}$) and $\widetilde{raf}_2 \downarrow_{\omega_2}$ (also denoted by $\widetilde{raf}_{2.2}$).

Note that, following this clustering, m and ψ also become inputs for $\widetilde{raf}_{2.2}$. Note also that this clustering must also take into account attacks and not only arguments (as it is the case for AFDivider*). See in Section 6 an example of such a clustering.*

4.3. Computing the Labellings

The next step is the computation of the component labellings in a distributed way relying on the clustering made. The σ-labellings of each cluster are computed simultaneously. Unlike the case

[8] $\widetilde{raf}_2 \downarrow_{\omega_1}$ produces a partial RAF built from $\widetilde{raf}_2$ keeping only the elements of ω_1.



(a) Cluster 1: $\widetilde{raf}_{2.1} = \widetilde{raf}_2 \downarrow_{\omega_1}$ (b) Cluster 2: $\widetilde{raf}_{2.2} = \widetilde{raf}_2 \downarrow_{\omega_2}$

Figure 5: Clusters of $\widetilde{raf}_2$ (the inputs m and ψ for $\widetilde{raf}_{2.2}$ are given in blue since their labellings are unknown at this time; whereas f is in green since its labelling is known: following $\mathcal{L}_{gr}$, f must be $\mathtt{in}$)

of connected components, the partial RAF corresponding to the computed clusters may admit several input labellings. In order to compute all the possible σ-labellings of a given cluster, every possible case concerning its input elements has to be considered. We call "context" a particular input labelling of a partial RAF:[9]

Definition 14. *Let $\widetilde{\mathcal{RAF}} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t \rangle$ be a partial RAF and $\mathcal{I} = \langle S^{inp}, Q^{inp} \rangle$ be the input of $\widetilde{\mathcal{RAF}}$. A context μ of a partial RAF is a labelling of $\mathcal{I}$.*

Example 6. *Following Figure 5, in the worst case, there exist 27 contexts of $\widetilde{raf}_{2.2}$ (3 inputs with 3 possible values, so $3^3 = 27$). Nevertheless, some of these 27 contexts are not compatible with the labelling $\mathcal{L}_{gr}$ (f must be labelled $\mathtt{in}$). So only 9 contexts are compatible.*

As one can notice from Example 6, it may be useless to consider some cluster contexts. So three optimizations can enhance the computation time:

- **First optimization**: Given a cluster, if one of its input elements is also an input element of the partial hard RAF then this element should only be labelled as in the *grounded* labelling $\mathcal{L}_{gr}$ (see Example 6).
- **Second optimization**: If an input attack is unattacked (a so-called valid attack) in the initial RAF, then this attack will always be labelled $\mathtt{in}$ following all semantics we are interested in.

  Example 7. *The attack ψ is an input of $\widetilde{raf}_{2.2}$ and a valid one in $\mathcal{RAF}$ and in $\widetilde{\mathcal{RAF}}_{hard}$. Hence, it is useless to consider contexts where ψ is not labelled $\mathtt{in}$.*

- **Third optimization**: Some contexts can lead to the same labellings. For instance, let us consider an inward attack y, the context putting y to $\mathtt{out}$, or the context putting $s(y)$ to $\mathtt{out}$give exactly the same labellings for $t(y)$. This can be used in order to decrease the number of contexts to consider (see [19] for more details).

[9] Obviously, each context of a partial RAF will induce a specific set of labellings of this partial RAF.

Example 8. *Applying the previous optimizations on the running example leads to 3 (resp. 3, 6) complete labellings for* $\widetilde{raf}_1$ *(resp.* $\widetilde{raf}_{2.1}$, $\widetilde{raf}_{2.2}$*) with 1 (resp. 1, 3) context. For instance, among the 3 (resp. 6) labellings for* $\widetilde{raf}_{2.1}$ *(resp.* $\widetilde{raf}_{2.2}$*), we have* $\mathcal{L}_{2.1.1} = \{(i, \mathtt{out}), (h, \mathtt{out}), (m, \mathtt{in}), (\iota, \mathtt{in}), (\kappa, \mathtt{in}), (\lambda, \mathtt{in}), (\rho, \mathtt{in}), (\psi, \mathtt{in})\}$ *(resp.* $\mathcal{L}_{2.2.1} = \{(k, \mathtt{in}), (l, \mathtt{out}), (\xi, \mathtt{out}), (\mathrm{o}, \mathtt{in}), (\pi, \mathtt{in})\}$*).*

4.4. Reunifying the Results

The labelling reunifying process is made in two steps: first, the reunification of the component labellings (*i.e.* the reunification of their cluster labellings together) and second, the reunification of the whole RAF labellings (*i.e.* the reunification of its component labellings together).

For the component labelling reunification, a CSP (Constraint Satisfaction Problem) is created. The aim is to identify the compatibility between the labellings of the elements that are in the "interface" of clusters in the same component (see our technical report [19] for details).

Example 9. *The reunification of the 3 labellings of* $\widetilde{raf}_{2.1}$ *with the 6 labellings of* $\widetilde{raf}_{2.2}$ *produces 6 labellings for the component* $\widetilde{raf}_2$*. And* $\mathcal{L}_{2.1} = \{(i, \mathtt{out}), (h, \mathtt{out}), (m, \mathtt{in}), (k, \mathtt{in}), (l, \mathtt{out}), (\iota, \mathtt{in}), (\kappa, \mathtt{in}), (\lambda, \mathtt{in}), (\rho, \mathtt{in}), (\psi, \mathtt{in}), (\xi, \mathtt{out}), (\mathrm{o}, \mathtt{in}), (\pi, \mathtt{in})\}$ *is one of them (built from the labellings given in Example 8:* $\mathcal{L}_{2.1.1}$ *for* $\widetilde{raf}_{2.1}$ *and* $\mathcal{L}_{2.2.1}$ *for* $\widetilde{raf}_{2.2}$*).*

A special step has to be carried out for the *preferred* semantics as this reunifying process does not ensure the maximality (*w.r.t.* $\sqsubseteq$) of the set of $\mathtt{in}$-labelled elements. Indeed, the preferred semantics is not bottom-up decomposable (see [16]). A maximality check must be done in order to keep only the wanted labellings. Moreover, when computing the *stable* semantics, the set of labellings $\mathscr{L}_\sigma$ returned by the algorithm may be empty. It happens when one of the component clusters has no *stable* labelling.

Now that all the component labellings are built, we can reunify the labellings of the whole RAF. Indeed, given that the trivial part is a fixed part of all σ-labellings of $\mathcal{RAF}$ and that each connected component has a unique context (these contexts being compatible with each other), the σ-labellings of the whole RAF are built by performing a simple cartesian product between the labellings of all the components and the trivial part labelling. If one of the components has no labelling then the whole RAF has no labelling (so $\mathscr{L}_\sigma = \varnothing$).

Example 10. *The* complete *semantics produces 18 labellings for the running example, with for instance the following labelling:*

$$\left\{\begin{array}{c}(e, \mathtt{in}), (f, \mathtt{in}), (g, \mathtt{out}), (\varepsilon, \mathtt{out}), (\zeta, \mathtt{in}), (\eta, \mathtt{in}), (\theta, \mathtt{in}),\\ (a, \mathtt{in}), (b, \mathtt{out}), (c, \mathtt{in}), (d, \mathtt{out}), (\alpha, \mathtt{in}), (\beta, \mathtt{in}), (\gamma, \mathtt{in}), (\delta, \mathtt{in}),\\ (i, \mathtt{out}), (h, \mathtt{out}), (m, \mathtt{in}), (k, \mathtt{in}), (l, \mathtt{out}),\\ (\iota, \mathtt{in}), (\kappa, \mathtt{in}), (\lambda, \mathtt{in}), (\rho, \mathtt{in}), (\psi, \mathtt{in}), (\xi, \mathtt{out}), (\mathrm{o}, \mathtt{in}), (\pi, \mathtt{in})\end{array}\right\}$$

(in this labelling, the first line corresponds to the grounded labelling for the trivial part, the second one is for $\widetilde{raf}_1$ *and the two last lines are the labelling* $\mathcal{L}_{2.1}$ *given in Example 9 for* $\widetilde{raf}_2$*)*

5. *RAFDivider*: Algorithms and Properties

Algorithms 1 and 2 give the formal definition of the *RAFDivider* algorithm. Similarly to *AFDivider*, they are said to be generic algorithms in the sense that any clustering method can be used to split the framework and any sound and complete procedure that computes the semantics σ, can be used to compute the labellings of the different clusters.

Algorithm 1: *RAFDivider* algorithm.

Input: Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and σ be a semantics
Result: $\mathscr{L}_\sigma \in 2^{\mathscr{L}(\mathcal{RAF})}$: the set of the σ-labellings of $\mathcal{RAF}$

$\mathcal{U}_{triv}, \mathcal{L}_{gr} \leftarrow$ `ComputeRAFTrivialPart`$(\mathcal{RAF})$
$\widetilde{\mathcal{RAF}}_{hard}, \mathcal{I} \leftarrow$ `RemoveRAFTrivialPart`$(\mathcal{RAF}, \mathcal{U}_{triv})$
$CCSet \leftarrow$ `SplitPartialRAFConnectedComponents`$(\widetilde{\mathcal{RAF}}_{hard})$
for all $\widetilde{raf}_i \in CCSet$ **do in parallel**
  $PartRAFSet \leftarrow$ `ComputePartRAFs`$(\widetilde{raf}_i)$ // clustering
  $\mathscr{L}_\sigma(\widetilde{raf}_i) \leftarrow$ `ComputeRAFCompLabs`$(\sigma, PartRAFSet, \mathcal{I}, \mathcal{L}_{gr})$
$\mathscr{L}_\sigma \leftarrow \{\mathcal{L}_{gr} \downarrow_{\mathcal{U}_{triv}}\} \times \prod_{\widetilde{raf}_i \in CCSet} \mathscr{L}_\sigma(\widetilde{raf}_i)$
return $\mathscr{L}_\sigma$

Algorithm 2: `ComputeRAFCompLabs` algorithm.

Input: Let σ be a semantics, $PartRAFSet$ be a set of clusters (partial RAFs) for a component $\widetilde{raf}_i$, $\mathcal{I}$ be the input of the partial hard RAF and $\mathcal{L}_{gr}$ be the grounded labelling of the initial RAF
Result: $\mathscr{L}_\sigma \in 2^{\mathscr{L}(\widetilde{raf}_i)}$: the set of the σ-labellings of $\widetilde{raf}_i$

for all $\widetilde{raf}_{i.j} \in PartRAFSet$ **do in parallel**
  $\mathscr{L}_\sigma^{\widetilde{raf}_{i.j}} \leftarrow$ `ComputePartRAFLabs`$(\sigma, \widetilde{raf}_{i.j}, \mathcal{I}, \mathcal{L}_{gr})$ // external solver call
$\mathscr{L}_\sigma \leftarrow$ `ReunifyComp`$(\{\mathscr{L}_\sigma^{\widetilde{raf}_{i.j}} | \widetilde{raf}_{i.j} \in PartRAFSet\})$ // using a CSP solver
if $\sigma = pr$ **then** $\mathscr{L}_\sigma \leftarrow \{\mathcal{L} | \mathcal{L} \in \mathscr{L}_\sigma$ s.t. $\nexists \mathcal{L}' \in \mathscr{L}_\sigma$ s.t. $in(\mathcal{L}) \sqsubset in(\mathcal{L}')\}$
return $\mathscr{L}_\sigma$

Before giving the properties of these algorithms, let us focus on two key elements of Algorithm 2:

- `ComputePartRAFLabs`: this function contains the call to an external solver able to compute the labellings of a given RAF. Such a trivial solver could be defined in three steps: (1) translation of any RAF into an AF using the flattening proposed in [18] then (2) computation of the labellings of this AF using any solver defined for AF (as the ones used for the ICCMA competition [21]) and finally (3) computation of the RAF labellings from these AF labellings using the properties that give the links between these AF labellings and the labellings of the initial RAF (see [18]). Another kind of solver could be built in two steps: (1) direct translation of a RAF into a logical base then (2) computation of the RAF semantics using the logical models of this base (see [22, 23]).

- `ReunifyComp`: this function is in charge of aggregating the labellings obtained for each cluster in order to obtain the labellings of the component containing these clusters (see Section 4.4). This aggregation, called reunification, must respect the compatibility of the resulting labellings; this is done in two steps: (1) creation of a CSP, then (2) resolution of this CSP using any CSP solver. The precise definition of this CSP is given in [19].[10]

The *RAFDivider* algorithm gives all the expected labellings (so it is complete) and only good labellings (it is sound) for the *complete*, *stable* and *preferred* semantics. The proof of the following proposition is given in [19]. It is very similar to the proofs given for *AFDivider* in [18].

Proposition 2. *Algorithms 2 and 1 are complete and sound for the* stable*,* complete *and* preferred *semantics.*

6. A Clustering Method

The main idea of the clustering presented in this section is to ensure that the Strongly Connected Components (SCC)[11] are not split into different clusters. The following method is inspired by those proposed in [12, 18] for testing the *AFDivider* algorithms. Given a RAF, the so-called "*USCC-clustering*" forms clusters as follows (each cluster being an $USCC_{raf}$, see Definition 8). First, the set of SCC is computed. Then neighbour SCC singletons are joined together in order to form a cluster using the following definition of neighbourhood:

Definition 15. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$. Let x and y be two elements of $\mathcal{RAF}$. x and y are considered as* neighbour *iff: either [$x \in K$ and ($y = s(x)$ or $y = t(x)$)] or [$x \in A$ and $y \in K$ and ($s(y) = x$ or $t(y) = x$)].*

In the third step, each SCC that is not a singleton is joined with its neighbour SCC singletons (those that are neighbours with at least an element in the non-singleton SCC) producing a cluster. This merging must respect the following constraint (the idea is to put the attacks and their source in the same cluster in order to have all the necessary elements for identifying the status of the target):

Definition 16. *Let $USCC_{raf}$ be a cluster. Let x be a singleton SCC that is a neighbour of $USCC_{raf}$. x will be joined with $USCC_{raf}$ if either $x \in K$ and $s(x) \in USCC_{raf}$ or $x \in A$ and $\exists y \in USCC_{raf}$ s.t. $s(y) = x$.*

The last step is to join clusters together so that there are not too many clusters of little size. This is done in an iterative way. The smallest group is merged to its smallest neighbour group, and that continues until there is no group of less than a certain number of arguments. Some experiments would be necessary in order to identify this threshold wrt the RAF that we take into account.

Using this method, we can retrieve the clustering proposed in Section 4.2 for the component $\widetilde{raf}_2$ (two clusters: $\{\lambda, i, \imath, m, \rho, h, \kappa, \psi\}$ and $\{k, \pi, l, \mathrm{o}, \xi\}$, see Figure 5).

[10] Note that this CSP contains two types of variables: one variable for each cluster and one variable for each input of these clusters. The domains of these variables correspond to their possible labellings. The constraints express the links between the labellings of the clusters and the labellings of their inputs.

[11] See in [18] the details about the definition of the SCCs for a RAF.

7. Conclusion and Future Works

This paper presents *RAFDivider*, one of the first algorithms for the enumeration of acceptable sets in a Recursive Argumentation Framework (RAF), an argumentation framework enriched with higher-order attacks. This algorithm, proven sound and complete, is based on a cutting of the framework which allows a distributed and parallel computation, technique successfully used for the enumeration of acceptable sets in an AF by *AFDivider*. An example of a clustering method, USCC-clustering, which can be used with this algorithm, is provided. An implementation of *RAFDivider* is to come with an experimental evaluation of such algorithms.

The extension of the algorithmic approach to other kinds of enriched argumentation frameworks may be investigated: argumentation frameworks which consider support interactions in addition to attacks, notably (see [24] for an overview of such enrichments).

To go further, such algorithms for argumentation frameworks with higher-order attacks may encourage the extension to RAFs of the reasoning tasks proposed for AFs at the International Competition on Computational Models of Argumentation (ICCMA) [21]. It would be the opportunity to define some benchmarks adapted to the higher-order frameworks that could be taken into account in the experimental evaluation previously mentioned.

Acknowledgments

The authors wish to thank Mickaël Lafages, the work of this article being mostly based on his PhD Thesis.

References

[1] Á. Carrera, C. A. Iglesias, A systematic review of argumentation techniques for multi-agent systems research, Artificial Intelligence Review 44 (2015) 509–535.

[2] H. Barringer, D. Gabbay, J. Woods, Temporal dynamics of support and attack networks: From argumentation to zoology, in: Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday. LNAI 2605, Springer Verlag, 2005, pp. 59–98.

[3] S. Modgil, An abstract theory of argumentation that accommodates defeasible reasoning about preferences, in: Proc. of ECSQARU, 2007, pp. 648–659.

[4] S. Modgil, Reasoning about preferences in argumentation frameworks, Artificial Intelligence 173 (2009) 901–934.

[5] P. Baroni, F. Cerutti, P. E. Dunne, M. Giacomin, Computing with infinite argumentation frameworks: The case of AFRAs, in: Proc. of TAFA, Revised Selected Papers, 2011, pp. 197–214.

[6] P. Baroni, F. Cerutti, M. Giacomin, G. Guida, AFRA: Argumentation framework with recursive attacks, Intl. Journal of Approximate Reasoning 52 (2011) 19–37.

[7] P. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and *n*-person games, Artificial Intelligence 77 (1995) 321–357.

[8] C. Cayrol, J. Fandinno, L. Fariñas del Cerro, M.-C. Lagasquie-Schiex, Valid attacks in argumentation frameworks with recursive attacks, AMAI Journal 89 (2020) 53–101.
[9] F. Cerutti, M. Giacomin, M. Vallati, M. Zanella, An SCC recursive meta-algorithm for computing preferred labellings in abstract argumentation, in: Proc. of KR, AAAI Press, 2014, pp. 42–51.
[10] W. Dvořák, R. Pichler, S. Woltran, Towards fixed-parameter tractable algorithms for abstract argumentation, Artificial Intelligence 186 (2012) 1–37.
[11] B. Liao, Toward incremental computation of argumentation semantics: A decomposition-based approach, Annals of Mathematics and Artificial Intelligence 67 (2013) 319–358.
[12] S. Doutre, M. Lafages, M.-C. Lagasquie-Schiex, A distributed and clustering-based algorithm for the enumeration problem in abstract argumentation, in: Proc. of PRIMA, Springer, 2019, pp. 87–105.
[13] M. Caminada, On the issue of reinstatement in argumentation, in: JELIA, 2006, pp. 111–123.
[14] P. Baroni, M. Caminada, M. Giacomin, An introduction to argumentation semantics, Knowledge Eng. Review 26 (2011) 365–410.
[15] G. Charwat, W. Dvořák, S. A. Gaggl, J. P. Wallner, S. Woltran, Methods for solving reasoning problems in abstract argumentation — A survey, Artificial Intelligence 220 (2015) 28–63.
[16] P. Baroni, G. Boella, F. Cerutti, M. Giacomin, L. Van Der Torre, S. Villata, On the input/output behavior of argumentation frameworks, Artificial Intelligence 217 (2014) 144–197.
[17] S. Doutre, M. Lafages, M.-C. Lagasquie-Schiex, Argumentation Frameworks with Higher-Order Attacks: Labellings and Complexity, in: M. Alamaniotis, S. Pan (Eds.), Proc. of ICTAI, IEEE, 2020, pp. 1210–1217.
[18] M. Lafages, Algorithms for Enriched Abstract Argumentation Frameworks for Large-scale Cases, Phd thesis, Toulouse University, Toulouse, France, 2021. URL: https://tel.archives-ouvertes.fr/tel-03664752.
[19] S. Doutre, M.-C. Lagasquie-Schiex, RAFDivider, Research Report IRIT/RR–2022–07–FR, IRIT - Institut de recherche en informatique de Toulouse, 2022.
[20] W. Dvorak, P. E. Dunne, Computational problems in formal argumentation and their complexity, in: Handbook of formal argumentation, College publication, 2018, pp. 631–688.
[21] ICCMA, International Competition on Computational Models of Argumentation, 2021. URL: http://argumentationcompetition.org.
[22] P. Besnard, C. Cayrol, M.-C. Lagasquie-Schiex, Logical theories and abstract argumentation: A survey of existing works, Argument and Computation 11 (2020) 41–102.
[23] C. Cayrol, M.-C. Lagasquie-Schiex, Logical Encoding of Argumentation Frameworks with Higher-order Attacks and Evidential Supports, International Journal on Artificial Intelligence Tools 29 (2020) 2060003. URL: https://hal.archives-ouvertes.fr/hal-02874146.
[24] C. Cayrol, A. Cohen, M. Lagasquie-Schiex, Higher-order interactions (bipolar or not) in abstract argumentation: A state of the art, in: Handbook of Formal Argumentation, Volume 2, College Publications, 2021, pp. 3–118.