Université de Toulouse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *(15/12/2021)* **par :**

MICKAËL LAFAGES

Algorithms for Enriched Abstract Argumentation Frameworks for Large-scale Cases

Jury

MASSIMILIANO GIACOMIN	Full professor - Università degli Studi di Brescia	Rapporteur
DANIEL LE BERRE	Professeur des universités - Université d'Artois, CNRS, CRIL	Rapporteur
LEILA AMGOUD	Directrice de recherche - CNRS, IRIT	Présidente du Jury
JEAN-GUY MAILLY	Maître de conférences - Université de Paris, LIPADE	Membre du Jury
SYLVIE DOUTRE	Maître de conférences - Université Toulouse 1, IRIT	Co-directrice de Thèse
MARIE-CHRISTINE LAGASQUIE	Professeur des universités - Université Toulouse 3, IRIT	Directrice de Thèse

École doctorale et spécialité :
MITT : Domaine STIC : Intelligence Artificielle
Unité de Recherche :
Institut de Recherche en Informatique de Toulouse (UMR 5505)

Remerciements

Si on devait la comparer à une activité sportive, je décrirais la thèse comme étant à la fois une course de fond et une course d'orientation aux multiples arrivées ! C'est seulement après en avoir pris le départ qu'on en découvre le tracé, qu'on en réalise pleinement la dureté. Aujourd'hui, je tiens à remercier celles et ceux qui ont participé à faire de cette aventure une grande réussite !

Tout d'abord, je souhaiterais remercier mes encadrantes, Marie-Christine LAGASQUIE et Sylvie DOUTRE. En me faisant part de leur expertise, elles ont su me donner les conseils adéquats pour retrouver mon chemin quand je m'égarais, pour rebooster ma course quand je m'essoufflais à la tâche. Au delà de l'aspect technique de l'exercice, je tiens à vous remercier tout particulièrement pour votre approche humaine. J'ai traversé plusieurs moments éprouvants au cours de ces années et vous avez été très compréhensives ! J'en suis persuadé: avec d'autres encadrants, cela aurait été une toute autre histoire. Un grand merci à vous !

Je remercie mes deux rapporteurs, Daniel LEBERRE et Massimiliano GIACOMIN, dont les rapports ont été très profitables, pleins de remarques et pistes d'amélioration pertinentes !

Je remercie Leila AMGOUD, directrice du Jury, ainsi que l'ensemble des membres du Jury, Daniel LEBERRE, Massimiliano GIACOMIN, Jean-Guy MAILLY, Marie-Christine LAGASQUIE et Sylvie DOUTRE d'avoir récompensé le fruit de mon travail de recherche en m'accordant le titre de Docteur.

Je remercie ma mère qui, depuis mon plus jeune âge, m'a donné le goût de l'apprentissage et la confiance en mes capacités intellectuelles. Merci pour tout ...

Je remercie Bettina, Roddy et Kevin, mes frères et soeur pour tout leur soutien ! Vous avez été pour moi des piliers.

Je remercie mon oncle Martin qui n'a jamais été trop loin, qui m'a toujours soutenu et encouragé ! Merci encore pour la visioconférence de ma soutenance, pour ce setup digne d'un youtuber !

Je remercie toute ma famille ainsi que mes amis, pour leur engagement envers moi ! Je ne pourrai citer tout le monde alors je ferai simplement cette liste réduite: Mélanie, Stéphie, Wendy, Samantha, Nicolas, Élodie, Yehouda, Kalidou, Mickaël, Dina, Jean-Michel, Sébastien, Kamila, Anaïs, Priscille, Audren.

Je remercie mes collègues de bureau pour leurs conseils et les bons moments passés ensemble: Pierre-François, Julien, Victor, Audren.

Enfin, je veux dire un grand merci à celui qui a toute mon admiration, sans la sollicitude duquel rien aurait été possible ! Il se reconnaitra ...

Summary

Abstract argumentation theory proposes methods to represent and deal with contentious information, and to draw conclusions or take decision from it. Such an abstract approach focuses on how arguments affect each other. Arguments are seen as generic entities which interact positively (support relation) or negatively (attack relation) with each other.

This abstraction level allows to propose generic reasoning processes that can be applied to any concrete definition or formalism for arguments. Argumentation-based reasoning model has been of application in multi-agent systems for years now. The development of argumentation techniques and of their computation drives such applications. This is the very motivation of this thesis: enhancing the use of abstract argumentation by developing better tools for its application.

A lot of frameworks and semantics have been proposed to enhance expressivity in abstract argumentation. While a given framework specifies the way of representing and expressing an argumentation problem (types of relations between arguments, weight on attacks or arguments, higher-order relation, etc.), a semantics, defined for a specific argumentation framework, captures what is a solution of an argumentation problem, in the sense of what is acceptable.

In this thesis, I first focus on solving more efficiently argumentation problems which are expressed in the basic, seminal argumentation framework and semantics defined by Dung. Dung's semantics produce sets of jointly acceptable arguments, called extensions. A new distributed and clustering based algorithm to compute Dung's semantics is my first contribution. This algorithm has been designed for certain types of large-scale argumentation frameworks, that produce a large number of extensions. It has been implemented and tested. The results of these tests show its efficiency in the context of the large scale argumentation frameworks which are targeted.

Second, I focus on argumentation frameworks with higher order attacks, and especially Recursive Argumentation Frameworks (RAF). In this context, an attack may have as target an attack: an argument may thus be acceptable while one of its attack (receiving itself an attack) may be invalid, and so non pertinent against its target. Similarly to Dung's semantics which produce extensions, the RAF semantics produce "structures", pairs whose first element is a set of arguments and the second a set of attacks.

If algorithms already existed for Dung's framework, it was not the case for RAF. In order to address this issue, I start with studying the complexity of RAF semantics. I then extend the notion of labelling to RAF, another kind of characterization of acceptability which already existed for Dung's framework. The notion of *"strongly connected component"* is extended to RAF and decomposability properties of RAF semantics are studied. All these contributions pave the way for future algorithms to compute acceptability under RAF semantics.

Résumé

La théorie de l'argumentation abstraite propose des méthodes pour représenter et traiter les informations potentiellement incohérentes, et pour en tirer des conclusions ou prendre des décisions. Une telle approche est dite abstraite car elle se concentre uniquement sur la manière dont les arguments s'influencent mutuellement et pas sur la constitution des arguments. Les arguments sont donc considérés comme des entités génériques qui interagissent positivement (relation de support) ou négativement (relation d'attaque) les unes avec les autres.

Ce niveau d'abstraction permet de proposer des processus de raisonnement génériques qui peuvent être appliqués à toute définition ou formalisme concret des arguments. Le modèle de raisonnement basé sur l'argumentation est appliqué dans les systèmes multi-agents depuis des années. Le développement des techniques d'argumentation et de leur calcul est un point clé de ces applications. C'est la motivation même de mon travail : améliorer l'utilisation de l'argumentation abstraite en développant de meilleurs outils pour sa mise en oeuvre.

De nombreux cadres d'argumentation et sémantiques associées ont été proposés dans la littérature pour améliorer l'expressivité de l'argumentation abstraite. Alors qu'un cadre donné spécifie la manière de représenter et d'exprimer un problème d'argumentation (types de relations entre les arguments, poids des attaques ou des arguments, relation d'ordre supérieur, etc.), une sémantique, pour un cadre d'argumentation spécifique, capture ce qui est une solution d'un problème d'argumentation, dans le sens de ce qui est acceptable.

Dans mon travail, je me suis d'abord concentré sur la résolution efficace de certains problèmes d'argumentation qui sont exprimés dans le cadre d'argumentation classique et les sémantiques définis par Dung. Les sémantiques de Dung produisent des ensembles d'arguments conjointement acceptables, appelés extensions. Mon travail a conduit à la proposition d'un nouvel algorithme distribué et basé sur une technique de *clustering* pour calculer les extensions sous les sémantiques de Dung. Il a été conçu pour certains types de cadres d'argumentation de "grande échelle", produisant un grand nombre d'extensions. Il a été implémenté et testé. Les résultats des tests montrent toute son efficacité pour les cadres d'argumentation à grande échelle ciblés.

Je me suis ensuite intéressé aux cadres d'argumentation d'ordre supérieur, et en particulier au cadre d'argumentation récursif (RAF). Dans ce contexte, une attaque peut avoir comme cible une autre attaque : un argument peut ainsi être acceptable alors même qu'il est attaqué parce que cette attaque (recevant elle-même une attaque) peut être invalide, et donc non pertinente contre sa cible. Là où le cadre de Dung produit des extensions, les sémantiques des RAF produisent des "structures", des paires dont le premier élément est un ensemble d'arguments et le second un ensemble d'attaques.

Si des algorithmes existaient déjà pour le cadre de Dung, il n'en était pas de même pour les RAF. J'ai donc commencé par étudier la complexité des sémantiques des RAF. J'ai ensuite étendu la notion de *labelling* aux RAF, une autre caractérisation de l'acceptabilité déjà existante dans le cadre de Dung. La notion de *"composante fortement connexe"* a été élargie aux RAF, et les propriétés de décomposabilité des sémantiques des RAF ont été étudiées. Toutes ces contributions ouvrent la voie à de futurs algorithmes pour calculer l'acceptabilité sous plusieurs sémantiques des RAF.

Contents

Part I

Introduction

What is this thesis about?

Argumentation is a research field of *Artificial Intelligence* interested in managing contentious information. Two major sub-domains can be considered in Argumentation. The first one, called *"Argument Mining"*, is interested in extracting arguments and their relations with each others, from natural language speeches (oral or written), in order to create a formal model to reason with (See [62] for more information). The second one, that we will call *"Argumentation Reasoning"*, is the one that is interested in reasoning over some argumentation model. It is useful to conclude, decide, convince, persuade or explain some issue. This way of reasoning, by considering arguments and their interactions, has proven successful in many contexts, multi-agent applications for instance (e.g. [15]).

I focus my PhD studies on Argumentation Reasoning, and more precisely, on *"Abstract Argumentation"*, field so called because it does not focus neither on how to construct arguments nor on what the arguments are made of (their content), but rather on how arguments affect each other. Arguments are seen as generic entities that interact positively (*e.g.* support relation) or negatively (*e.g.* attack relation) with each other. This abstraction level allows to propose generic reasoning processes that could be applied to any precise definition or formalism for arguments.

There exist several approaches and formalisms to express argumentation problems. They differ on which *"Argumentation Frameworks"* and which *"semantics"* to use, to determine the argumentation solutions. These are two key notions in this research area:

- Considering the first key notion, here are some questions that have to be answered in order to "choose an Argumentation Framework" that fits with our need. Do we allow positive relations? If so, of which kind? Do we allow negative relations? If so, of which kind? Is there any notion of strength in arguments or in relations? The aim of making more complex Argumentation Frameworks is to be able to better capture human argumentation subtleties.

- Given an *Argumentation Framework*, the second key notion, *semantics*, corresponds to a formal way to say how the solution of the argumentation should be decided. It is really related with the notion of *"acceptability"*. How to define an *acceptable* argumentation problem solution?

The basic, seminal Argumentation Framework and semantics have been defined by Dung in [39], known as *Dung's Argumentation Framework* (**AF**). Since then, a lot of propositions have been made to enhance the expressivity in Abstract Argumentation (*e.g.* [16, 18, 6, 4, 29]).

Future innovations in the area of *Argument Mining* may revolutionize the field of Artificial Intelligence, leading to the establishment of large-scale Argumentation Frameworks, built from arguments and relations collected for instance over the entire World Wide Web. Such frameworks may be large in their number of arguments, in their number of relations, in their variety of relation types, in their structure. This perspective made me choose as subject for my PhD studies the enhancement of tools for *Abstract Argumentation*. Carried out in the *Institut de Recherche en Informatique de Toulouse* (IRIT) and supervised by Marie-Christine LAGASQUIE and Sylvie DOUTRE, my thesis has thus been entitled: "*Algorithms for enriched abstract argumentation frameworks for large-scale cases*".

A first milestone

In the first part of my PhD, I center my research on solving more efficiently argumentation problems that are expressed in Dung's AF and semantics. This is a necessary first step before considering studying an extension of this work to other enriched Argumentation Frameworks and semantics.

In Dung's setting, solutions of an argumentation problem are sets of arguments (defined under the notion of *extension*) which, when considered together, win the argumentation. Finding all the possible solutions of an argumentation problem, *i.e.* all its winning sets of arguments, can be very time consuming. Many argumentation problem instances, particularly large[1], are too hard to be solved in an acceptable time, as shown by the results of the ICCMA argumentation solver competition[2]. This hardness is not relative to the current state of the art but rather to the intrinsic theoretical complexity of the argumentation semantics that are tackled [41].

Considering the foreseen scaling-up challenge mentioned above in addition to the complexity, there is a need for heuristics, methods and algorithms efficient enough to tackle such issues and make possible the use of automated argumentation models, even in such settings. Enhancing the computational time of enumerating the solutions of an argumentation framework has been the object of study of many works, resulting in the elaboration of several recent algorithms such as [1, 25, 52, 3] (see [26] for an overview).

To address this issue, we propose the *AFDivider* algorithm, a *distributed* and *clustering*-based algorithm that has for main purpose to find all the possible solutions of an argumentation problem. Those solutions are defined in terms of semantics labellings [14, 10], a three status based function mapping that assigns to each argument of an AF an acceptance status: *accepted*, *rejected* or *undecided*. An empirical analysis of the *AFDivider* algorithm shows that the new approach of computing Dung-like semantics is relevant and very appropriate for some types of argumentation problems. This work led to several publications: [49, 31, 27, 32] (See Part III on page 27 for more information).

The next milestone

In the second part of my thesis, I focused on "*Argumentation Frameworks with Higher-Order Attacks*" (e.g. [12, 57, 58, 5, 6]). This type of Argumentation Framework is a rich extension of the classical Dung's AF: 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 [18] 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 of AF and structure-based semantics of RAF without any attack on attacks has been shown in [18], proving that RAF are a conservative generalisation of AF. This characteristic makes RAF particularly interesting to consider.

Given that the computation of RAF semantics has not been addressed so far, I dedicate the second part of my thesis to the developments of tools (new notions) for RAFs and the study of RAF properties, preparing thus the way for algorithm proposals. I first adapted the notion of AF *labelling* for RAF, so-called, "*structure labellings*". Secondly, I introduce a *flattening* process that transforms RAFs into AFs, ensuring interesting properties. Thirdly, relying on that *flattening*, I study the *complexities* of RAF semantics. Finally, I adapt the notion of *Strongly Connected Component* to RAFs and from this key notion, I study the *semantics decomposability* of RAF semantics (notion introduced for AFs in [8]). These works led to several publications: [34, 33, 36, 35] (See Part V on page 101 for more information).

[1]This notion of largeness of an argumentation framework is not so simple to define. It is related to the fact that the computation of the solutions is complex either because of the number of arguments, or of the number of interactions, or because of the structure of the argumentation framework.

[2]`http://argumentationcompetition.org`

How this thesis is organized? How to read it?

The main body of this thesis contains four parts (introduction and conclusion set apart):

- Part II on page 7: Dung Argumentation Framework: Background.
- Part III on page 27: Dung Argumentation Framework: Contribution.
- Part IV on page 83: Higher-Order Attack Argumentation Frameworks: Background.
- Part V on page 101: Higher-Order Attack Argumentation Frameworks: Contribution.

Parts II and III concern the first milestone of my studies, as mentioned above, while Parts IV and V concern the second one. In each background (Parts II and IV) the notions related to argumentation required to understand the contribution parts that follow (Parts III and V) are given. Notice that, each contribution part is quite independent from the other.[3] However, for some of them, mathematical notions are required to fully understand the contributions, especially Part III. Those notions are given in appendix as explained below.

The last part of the main body, Part VI on page 165, concludes this thesis and opens perspectives for future works.

This thesis has three appendices:

- Appendix 1: Mathematical Background (on page 170)

  In this part are given all the mathematical background required to understand the ins and outs of this thesis. By sake of clarity, it has been separated from the main body.

- Appendix 2: Tables (on page 194)

  In this part are given tables of symbols to help the reading of this thesis.

- Appendix 3: Proofs (on page 219)

  In this part are given all the proofs of the propositions and theorems proposed in the different contribution chapters.[4] In order to facilitate the reading of this thesis, a bi-direction linking has been made, allowing thus to go to and from some property and its proof.

Although the different parts, chapters and sections listed below are not fully independent, here is a guide for the reading of this thesis by topic. For the reader interested in:

- **Complexity**, read:
  - Section 16.4 on page 178 (in the appendix): Computational complexity theory
  - Chapter 3 on page 23: AF decision problems and complexities
  - Section 9.1 on page 91: Structure semantics
  - Chapter 12 on page 114: RAF flattening[5]

[3]Although there are some definitions of Parts II and III that are used in Part V, this does not require to read these parts first. Those definitions can be read when needed.

[4]Some additional lemmas with their proof are also given in this appendix.

[5]This part is necessary to understand how the complexity of RAF decision problems has been proven.

  - Chapter 13 on page 119: RAF decision problems and semantics complexities

- **algorithms**, read:
  - Section 16.2 on page 170 (in the appendix): Graph theory
  - Section 16.3 on page 175 (in the appendix): Matrices
  - Chapter 17 on page 186 (in the appendix): Mathematical problems
  - Chapter 1 on page 8: Semantics : extensions and labellings
  - Part III on page 27: Dung Argumentation Framework: Contribution

- **Semantics Decomposability**:
  - Section 16.1 on page 170 (in the appendix): Set theory
  - Chapter 2 on page 16: AF semantics decomposability
  - Section 9.1 on page 91: Structure semantics
  - Chapter 11 on page 107: Structure labellings and semantics
  - Chapter 12 on page 114: RAF flattening[6]
  - Chapter 14 on page 123: Hierarchical view of RAF and semantics decomposability

[6]This part is necessary to understand how the decomposability of RAF semantics has been proven.

Part II

Dung Argumentation Framework: Background

Part presentation:

In [39], Dung introduced the seminal abstract argumentation framework. It consists of a set of arguments and of a binary attack relation between them. An “Argumentation Framework” (denoted **AF**) can be represented as a directed graph in which nodes are arguments and directed edges are attack relations between arguments. Formally, it is defined as follows:

Definition 1 (Argumentation framework)**.** *An* argumentation framework *(**AF**) is a pair $\mathcal{AF} = \langle A, K \rangle$ where A is a finite set of abstract arguments and $K \subseteq A \times A$ is a binary relation on A, called the attack relation: $(a,b) \in K$ means that a attacks b. The set of all possible argumentation frameworks is denoted as Φ_{af}.*

Example 1. *Figure 1 shows an illustration of an AF. In all this document, arguments (in Latin letter) will be represented by a round box and attacks are represented by directed edges.*

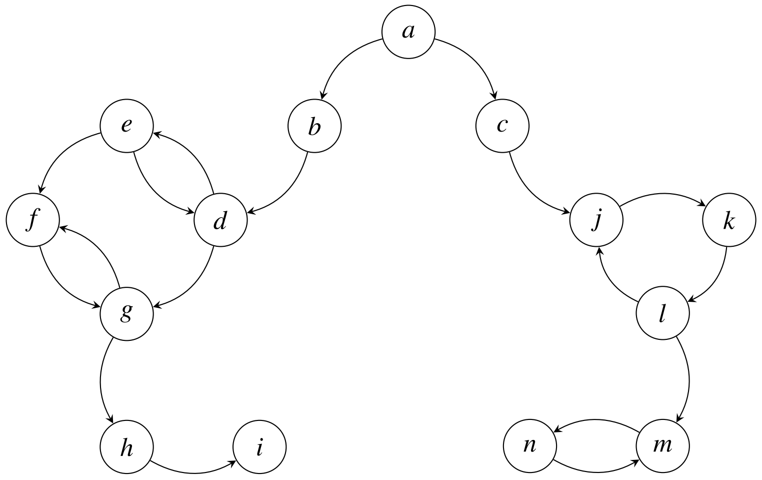


Figure 1: Example of an AF

This formalism provides a strong base to compute the “solutions” of the argumentation so represented. In Chapter 1 on the next page are presented different types of argumentation solutions, so called *semantics*. Chapter 2 on page 16 presents the notion of semantics semantics decomposability and gives some properties over the semantics we are interested in. Finally, in Chapter 3 on page 23 are defined AF decision problems and their complexities are given.

Chapter 1

Semantics : extensions and labellings

Basically, a semantics defines what is a "*solution*" of an argumentation. In this chapter we present two kinds of semantics types: *extension-based* ones (Section 1.1) and *labelling-based* ones (Section 1.2 on page 10). Then in Section 1.3 on page 12 are given the relations between these two kinds of semantics.

1.1 Extension-based semantics

In Dung-like semantics [39], so-called extension-based semantics, a solution of an argumentation is a group of arguments that, together, win the argumentation. The semantics thus define how to select those groups. Formally, a generic AF extension-based semantics is defined as follows:

Definition 2 (Extension-based Semantics)**.** *Let σ be a function over Φ_{af}. σ is said to be an AF extension-based semantics iff the following property holds:*

$$\forall \mathcal{AF} \in \Phi_{af},\ \sigma(\mathcal{AF}) \subseteq 2^{A},\ \ \textit{with } \mathcal{AF} = \langle A, K\rangle$$

A σ-extension is defined as follows:

Definition 3 (Extension)**.** *Let σ be an AF semantics and let $\mathcal{AF} = \langle A, K\rangle$ be an AF. Let $S \subseteq A$ be a set of arguments. We say that S is a σ-extension of $\mathcal{AF}$ iff $S \in \sigma(\mathcal{AF})$.*

Dung's semantics rely on the notion of *defeat* and *acceptability*.

Definition 4 (Defeat and acceptability in Dung's framework)**.** *Let $\mathcal{AF} = \langle A, K\rangle$ be an AF and $S \subseteq A$ be a set of arguments. An argument $a \in A$ is said to be:*

- *defeated w.r.t. S iff $\exists b \in S$ s.t. $(b, a) \in K$.*
- *accepted w.r.t. S iff $\forall (b, a) \in K, \exists c \in S$ s.t. $(c, b) \in K$.*

We define the sets of defeated and accepted arguments w.r.t. S as follows:

$$Def(S) = \{a \in A | \exists b \in S \text{ s.t. } (b, a) \in K\}$$
$$Acc(S) = \{a \in A | \forall (b, a) \in K, \exists c \in S \text{ s.t. } (c, b) \in K\}$$

Let now give the formal definition of some AF semantics :

Definition 5 (Some extension-based semantics of AF)**.** *Let $\mathcal{AF} = \langle A, K \rangle$ be an AF and $S \subseteq A$ be a set of arguments. S is said to be an extension:*

1. Conflict-free *iff* $S \cap Def(S) = \varnothing$.

2. Naive *iff it is a* $\subseteq$*-maximal* conflict-free *extension.*

3. Admissible *iff it is* conflict-free *and* $S \subseteq Acc(S)$.

4. Complete *iff it is* conflict-free *and* $S = Acc(S)$.

5. Preferred *iff it is a* $\subseteq$*-maximal* admissible *extension.*

6. Grounded *iff it is the* $\subseteq$*-minimal* complete *extension.*

7. Semi-stable *iff it is a* complete *extension such that* $S \cup Def(S)$ *is maximal w.r.t.* $\subseteq$.

8. Stable *iff it is* conflict-free *and* $S \cup Def(S) = A$.

In the following we will mainly focus on the *complete*, *preferred*, *grounded*, *semi-stable* and *stable* semantics as they are the most commonly used in the literature and so, given an AF $\mathcal{AF}$, $\sigma(\mathcal{AF})$ is the set of extensions of $\mathcal{AF}$ under semantics σ, with σ being as an example the *complete* (*co*), *grounded* (*gr*), *stable* (*st*), *semi-stable* (*sst*) or *preferred* (*pr*) semantics.

Example 2. *Table 1.1 on page 13 shows which extensions those semantics produce for the AF in Figure 1 on page 7. It can be noticed that on this particular AF, the* stable *semantics does not produce any extension.*

There exists a partial order over the semantics defined in Definition 5 (see [14] for proofs).

Proposition 1 (Extension-based semantics partial ordering)**.** *The following properties hold:*

- Stable *extensions are also* semi-stable *extensions*
- Stable *extensions are also* naive *extensions*
- Semi-stable *extensions are also* preferred *extensions*
- Preferred *extensions are also* complete *extensions*
- *The* Grounded *extension is also a* complete *extension*
- Complete *extensions are also* admissible *extensions*
- Admissible *extensions are also* conflict-free *extensions*
- Naive *extensions are also* conflict-free *extensions*

Proposition 2 (Extension-based semantics cardinality)**.** *The following properties hold:*

- *There is always at least one* conflict-free *extension.*
- *There is always at least one* naive *extension.*

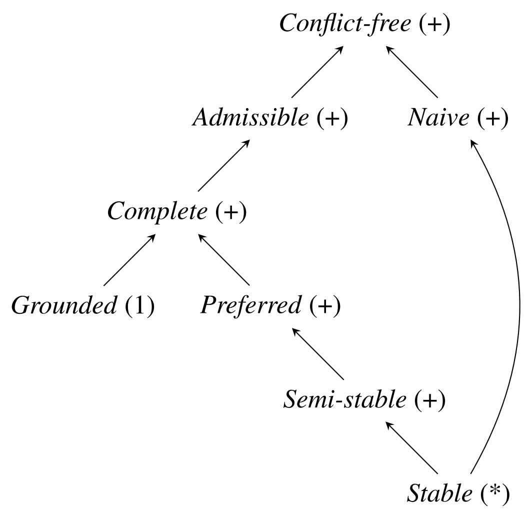


Figure 1.1: AF semantics partial ordering

The cardinality of each semantics, that is, the number of extensions which can be produced by each semantics, is represented between parentheses.
"$*$" means zero or more, "$+$" means one or more.

- *There is always at least one* admissible *extension.*
- *There is always at least one* complete *extension.*
- *There is always a unique* grounded *extension.*
- *There is always at least one* preferred *extension.*
- *There is always at least one* semi-stable *extension.*
- *It may be the case that there is no* stable *extension.*

Figure 1.1 illustrates Propositions 1 and 2.

1.2 Labelling-based semantics

Dung-like semantics can also be defined in terms of labellings as introduced in [14]. A labelling maps to each argument of an AF a value representing its acceptability status. This status may be accepted ($\mathtt{in}$), rejected ($\mathtt{out}$) or in an undecided state ($\mathtt{und}$). Formally:

Definition 6 (Labelling)**.** *Let $\mathcal{AF} = \langle A, K \rangle$ be an AF, and $S \subseteq A$. A* labelling *of S is a total function $\ell : S \to \{\mathtt{in}, \mathtt{out}, \mathtt{und}\}$. A* labelling *of $\mathcal{AF}$ is a labelling of A. The set of all labellings of $\mathcal{AF}$ is denoted as $\mathscr{L}(\mathcal{AF})$. The set of all labellings of a set of arguments S is denoted as $\mathscr{L}(S)$.*

We write $\mathtt{in}(\ell)$ for $\{a | \ell(a) = \mathtt{in}\}$, $\mathtt{out}(\ell)$ for $\{a | \ell(a) = \mathtt{out}\}$ and $\mathtt{und}(\ell)$ for $\{a | \ell(a) = \mathtt{und}\}$.

Definition 7 (legally labelled argument)**.** *Let $\mathcal{AF} = \langle A, K\rangle$ be an AF, and $\ell \in \mathscr{L}(\mathcal{AF})$ be a labelling.*

- *An* `in`*-labelled argument is said to be legally* `in` *iff all its attackers are labelled* `out`*.*
- *An* `out`*-labelled argument is said to be legally* `out` *iff at least one of its attackers is labelled* `in`*.*
- *An* `und`*-labelled argument is said to be legally* `und` *iff it does not have an attacker that is labelled* `in` *and one of its attackers is not labelled* `out`*.*

Definition 8 (*Conflict-free* labelling)**.** *Let $\mathcal{AF} = \langle A, K\rangle$ be an AF, and $\ell \in \mathscr{L}(\mathcal{AF})$ be a labelling. ℓ is an* conflict-free *labelling of $\mathcal{AF}$ iff for each $a \in$* `in`*(ℓ), a is legally* `in`*.*

Definition 9 (*Naive* labelling)**.** *Let $\mathcal{AF} = \langle A, K\rangle$ be an AF, and $\ell \in \mathscr{L}(\mathcal{AF})$ be a labelling. ℓ is a* naive *labelling of $\mathcal{AF}$ iff it is a* conflict-free *labelling of $\mathcal{AF}$ which maximizes (w.r.t $\subseteq$) the set of* `in`*-labelled arguments.*

Definition 10 (Admissible labelling)**.** *Let $\mathcal{AF} = \langle A, K\rangle$ be an AF, and $\ell \in \mathscr{L}(\mathcal{AF})$ be a labelling. ℓ is an* admissible *labelling of $\mathcal{AF}$ iff it satisfies the following conditions for any $a \in A$:*

- *For each $a \in$* `in`*(ℓ), a is legally* `in`*.*
- *For each $a \in$* `out`*(ℓ), a is legally* `out`*.*

Definition 11 (Complete labelling)**.** *Let $\mathcal{AF} = \langle A, K\rangle$ be an AF, and $\ell \in \mathscr{L}(\mathcal{AF})$ be a labelling. ℓ is a* complete *labelling of $\mathcal{AF}$ iff it satisfies the following conditions for any $a \in A$:*

- *For each $a \in$* `in`*(ℓ), a is legally* `in`*.*
- *For each $a \in$* `out`*(ℓ), a is legally* `out`*.*
- *For each $a \in$* `und`*(ℓ), a is legally* `und`*.*

Definition 12 (Grounded, preferred, semi-stable and stable labelling)**.**
Let $\mathcal{AF} = \langle A, K\rangle$ be an AF, and $\ell \in \mathscr{L}(\mathcal{AF})$ be a labelling.

- *ℓ is the* grounded *labelling of $\mathcal{AF}$ iff it is the* complete *labelling of $\mathcal{AF}$ which minimizes (w.r.t $\subseteq$) the set of* `in`*-labelled arguments.*
- *ℓ is a* preferred *labelling of $\mathcal{AF}$ iff it is a* complete *labelling of $\mathcal{AF}$ which maximizes (w.r.t $\subseteq$) the set of* `in`*-labelled arguments.*
- *ℓ is a* stable *labelling of $\mathcal{AF}$ iff it is a* complete *labelling of $\mathcal{AF}$ which has no* `und`*-labelled argument.*
- *ℓ is a* semi-stable *labelling of $\mathcal{AF}$ iff it is a* complete *labelling of $\mathcal{AF}$ which minimizes (w.r.t. $\subseteq$) the set of* `und`*-labelled arguments.*

It can be noticed that all complete labellings include the grounded labelling, and, as stable and preferred labellings are complete labellings, they also include the grounded labelling.

Definition 13 (Labelling-based Semantic)**.** *A labelling-based semantic σ is a total function $\sigma : \Phi_{af} \to 2^{\mathscr{L}(\mathcal{AF})}$ which associates to an AF $\mathcal{AF} = \langle A, K\rangle$ a subset of $\mathscr{L}(\mathcal{AF})$.*

Given an AF $\mathcal{AF} = \langle A, K\rangle$, the set of labellings under semantics σ is denoted $\mathscr{L}_\sigma(\mathcal{AF})$.

Example 3. *Let us consider the AF of Figure 1 on page 7. Table 1.2 on page 14 shows the labellings corresponding to the different semantics (the other possible labellings are not given). As you can see this AF has no stable labellings.*

The partial ordering and cardinalities of extension-based semantics stated in Propositions 1 and 2 on page 9 and on page 10 also hold for labelling-based semantics. See Figure 1.1 on page 10.

Proposition 3 (Labelling-based semantics partial ordering)**.** *The following properties hold:*

- Stable *labellings are also* semi-stable *labellings*
- Stable *labellings are also* naive *labellings*
- Semi-stable *labellings are also* preferred *labellings*
- Preferred *labellings are also* complete *labellings*
- *The* grounded *labelling is also a* complete *labelling*
- Complete *labellings are also* admissible *labellings*
- Admissible *labellings are also* conflict-free *labellings*
- Naive *labellings are also* conflict-free *labellings*

Proposition 4 (Labelling-based semantics cardinality)**.** *The following properties hold:*

- *There is always at least one* conflict-free *labelling.*
- *There is always at least one* naive *labelling.*
- *There is always at least one* admissible *labelling.*
- *There is always at least one* complete *labelling.*
- *There is always a unique* grounded *labelling.*
- *There is always at least one* preferred *labelling.*
- *There is always at least one* semi-stable *labelling.*
- *It may be the case that there is no* stable *labelling.*

1.3 Extension-based and labelling-based semantics relations

Now let consider the relation between extension-based semantics and labelling-based semantics. As proven in [14], Proposition 5 holds.

Proposition 5 (Semantics bijection)**.** *Let* $\sigma \in \{\text{complete, stable, grounded, preferred, semi-stable}\}$ *be a semantics. There exists a bijection between* σ*-extensions and* σ*-labellings. The correspondences stated in Table 1.3 on page 15 hold.*

Given that there is a bijection between extension-based and labelling-based semantics, in the rest of this document, we will not specify extension-based or labelling-based unless it is necessary.

	S_1	S_2	S_3	S_4	S_5	S_6
a	✓	✓	✓	✓	✓	✓
b						
c						
d			✓	✓		
e	✓	✓				
f			✓	✓		
g	✓	✓				
h			✓	✓		
i	✓	✓				
j						
k						
l						
m						
n		✓		✓		✓
grounded					●	
complete	●	●	●	●	●	●
preferred		●		●		
stable						

Table 1.1: Semantic extensions of the AF in Figure 1 on page 7

"✓" means that the argument on the row belongs to the extension on the column.
"●" means that the semantics on the row produces the extension on the column.

	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5	ℓ_6
a	*in*	*in*	*in*	*in*	*in*	*in*
b	*out*	*out*	*out*	*out*	*out*	*out*
c	*out*	*out*	*out*	*out*	*out*	*out*
d	*out*	*out*	*in*	*in*	*und*	*und*
e	*in*	*in*	*out*	*out*	*und*	*und*
f	*out*	*out*	*in*	*in*	*und*	*und*
g	*in*	*in*	*out*	*out*	*und*	*und*
h	*out*	*out*	*in*	*in*	*und*	*und*
i	*in*	*in*	*out*	*out*	*und*	*und*
j	*und*	*und*	*und*	*und*	*und*	*und*
k	*und*	*und*	*und*	*und*	*und*	*und*
l	*und*	*und*	*und*	*und*	*und*	*und*
m	*und*	*out*	*und*	*out*	*und*	*out*
n	*und*	*in*	*und*	*in*	*und*	*in*
grounded					●	
complete	●	●	●	●	●	●
preferred		●		●		
stable						

Table 1.2: Semantic labellings

"●" means that the semantics on the row produces the labelling on the column.

Restriction on AF reinstatement labelling	Semantics
no restrictions	*complete* semantics
empty und	*stable* semantics
maximal in	*preferred* semantics
maximal out	*preferred* semantics
maximal und	*grounded* semantics
minimal in	*grounded* semantics
minimal out	*grounded* semantics
minimal und	*semi-stable* semantics

Table 1.3: Reinstatement labelling and extension based semantics correspondence

Chapter 2

Semantics decomposability

In addition to semantics properties found for extension-based semantics, such as SCC-recursiveness (see [11] for more information), the labelling approach allowed the discovery of new ones. In this chapter we present the notion of semantics decomposability defined in [8] along with the different properties that are related to it.

Several notions are required to define the semantics decomposability properties of a semantics.

Definition 14 (Labelling restriction $\downarrow$)**.** *Let ℓ be a labelling. Let S be a set of arguments. The* restriction *of ℓ to S denoted as $\ell \downarrow_S$ is defined as $\ell \cap (S \times \{\mathtt{in}, \mathtt{out}, \mathtt{und}\})$.*

Definition 15 (Input arguments and conditioning relation)**.** *Let $\mathcal{AF} = \langle A, K\rangle$ be an AF and $S \subseteq A$ be a set. The input of S, denoted as S^{inp}, is the set $\{b \in A \setminus S | \exists a \in S, (b,a) \in K\}$. The conditioning relation of S, denoted as S^K, is defined as $K \cap (S^{inp} \times S)$.*

Example 4. *Let consider Figure 2.1 and let consider the set $S = \{f, g, h, i\}$. We have: $S^{inp} = \{d, e\}$ and $S^K = \{(e, f), (d, g)\}$.*

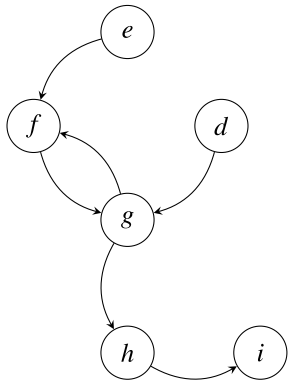


Figure 2.1: An AF

Definition 16 (AF with input). *An argumentation framework with input is a tuple $\left\langle \mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$, including an argumentation framework $\mathcal{AF} = \langle A, K \rangle$, a set of arguments $\mathcal{I}$ such that $\mathcal{I} \cap A = \emptyset$, a labelling $\ell^{\mathcal{I}}$ of the elements of $\mathcal{I}$ and a relation $K_{\mathcal{I}} \subseteq \mathcal{I} \times A$.*

Example 5. *Let consider the following AF with input, illustrated in Figure 2.2:*

$$\left\langle \mathcal{AF}, \mathcal{I} = \{e, d\}, \ell^{\mathcal{I}} = \{(e, \mathtt{out}), (d, \mathtt{und})\}, K_{\mathcal{I}} = \{(e, f), (d, g)\} \right\rangle$$

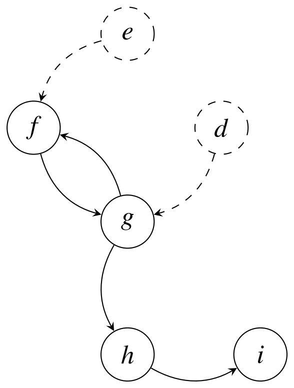


Figure 2.2: An illustration of the AF with input (Example 5)

Let $\left\langle \mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$ be an AF with input. Its standard AF is an AF that simulates the conditioning labelling of its input arguments $\mathcal{I}$. To do so, some fictive arguments and interactions are added. Formally:

Definition 17 (standard AF). *Given $\left\langle \mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$, an argumentation framework with input, the standard AF w.r.t. $\left\langle \mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$ is defined as:*

$$std\text{-}\mathcal{AF} = \left\langle A \cup \mathcal{I}', K \cup K'_{\mathcal{I}} \right\rangle$$

Where:

- $\mathcal{I}' = \mathcal{I} \cup \left\{ a' | a \in \mathcal{I} \cap \mathtt{out}(\ell^{\mathcal{I}}) \right\}$
- $K'_{\mathcal{I}} = K_{\mathcal{I}} \cup \left\{ (a', a) | a \in \mathcal{I} \cap \mathtt{out}(\ell^{\mathcal{I}}) \right\} \cup \left\{ (a, a) | a \in \mathcal{I} \cap \mathtt{und}(\ell^{\mathcal{I}}) \right\}$.[1]

`Note:` By definition, if the standard AF std-$\mathcal{AF}$ admits some labellings then, restricted to the input $\mathcal{I}$, those labellings are exactly the labelling $\ell^{\mathcal{I}}$. Notice that for the stable *semantics it may be the case that std-$\mathcal{AF}$ admits no labelling.*

[1]The fictive arguments are denoted by a' in the definition of $\mathcal{I}'$ and the fictive interactions are the pairs (a', a) or (a, a) appearing in the definition of $K'_{\mathcal{I}}$.

Example 6. *Let consider the AF with input* $\left\langle \mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$ *in Example 5 on the previous page. Given that the labelling of its input arguments is* $\ell^{\mathcal{I}} = \{(e, \mathtt{out}), (d, \mathtt{und})\}$*, we obtain as its corresponding standard AF the one illustrated in Figure 2.3.*

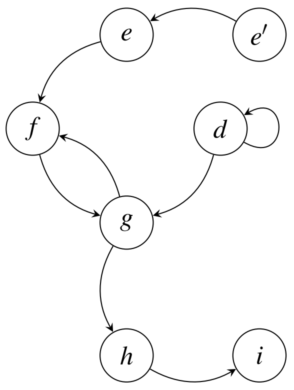


Figure 2.3: The standard AF corresponding to Example 5 on the previous page

Given an AF with input $\left\langle \mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$ (with $\mathcal{AF} = \langle A, K \rangle$) and its corresponding standard AF *std-*$\mathcal{AF}$, the canonical local function is simply a function that gives the set of labellings under a certain semantics of the sub-AF we are interested in: *std-*$\mathcal{AF} \downarrow_A = \mathcal{AF}$ (*i.e.* the input arguments and the other fictive arguments created are not in these labellings).

Definition 18 (Canonical local function)**.** *Let* σ *be a semantics,* $\left\langle \mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$ *be an AF with input (with* $\mathcal{AF} = \langle A, K \rangle$*) and std-*$\mathcal{AF}$ *be its standard AF. The canonical local function* $\mathscr{F}_{\sigma}^{af}$ *is the local function such that:*

$$\mathscr{F}_{\sigma}^{af}(\mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}}) = \left\{ \ell \downarrow_A | \ell \in \mathscr{L}_{\sigma}(std\text{-}\mathcal{AF}) \right\}$$

Example 7. *Considering Example 6, we have for* $\sigma \in \{$stable, preferred, semi-stable$\}$*:*

$$\mathscr{F}_{\sigma}^{af}(\mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}}) = \{\{(f, \mathtt{in}), (g, \mathtt{out}), (h, \mathtt{in}), (i, \mathtt{out})\}\}$$

For the grounded *semantics we have:*

$$\mathscr{F}_{gr}^{af}(\mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}}) = \{\{(f, \mathtt{und}), (g, \mathtt{und}), (h, \mathtt{und}), (i, \mathtt{und})\}\}$$

And for the complete *:*

$$\mathscr{F}_{co}^{af}(\mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}}) = \left\{ \begin{array}{c} \{(f, \mathtt{und}), (g, \mathtt{und}), (h, \mathtt{und}), (i, \mathtt{und})\}, \\ \{(f, \mathtt{in}), (g, \mathtt{out}), (h, \mathtt{in}), (i, \mathtt{out})\} \end{array} \right\}$$

Definition 19 (Semantics fully decomposability)**.** *A semantics* σ *is* fully decomposable *(or* decomposable*) if and only if there is a local function* $\mathscr{F}^{af}$ *such that for every AF* $\mathcal{AF} = \langle A, K \rangle$ *and every partition* $\Omega =$

$\{\omega_1,...,\omega_n\}$ *of A, we have:*

$$\mathscr{L}_\sigma(\mathcal{AF}) = \left\{\ell_1\cup...\cup\ell_n \middle| \forall i, \ell_i \in \mathscr{F}^{af}(\mathcal{AF}\downarrow_{\omega_i}, \omega_i^{inp}, (\bigcup_{j\in\{1,...,n\}\ s.t.\ j\neq i} \ell_j)\downarrow_{\omega_i^{inp}}, \omega_i^K)\right\}$$

Definition 20 (Initial argument). *Let $\mathcal{AF} = \langle A,K\rangle$ be an AF, and $b \in A$ be an argument. b is an initial argument of $\mathcal{AF}$ if there is no argument in $\mathcal{AF}$ attacking b.*

`Note:` *In graph theory, nodes corresponding to "initial arguments" are called source nodes.*

Definition 21 (Complete-compatibility). *A semantics σ is complete-compatible if and only if the following conditions hold:*

1. *For any AF $\mathcal{AF} = \langle A,K\rangle$, every labelling $\ell \in \mathscr{L}_\sigma(\mathcal{AF})$ satisfies the following conditions:*
   - *if $a \in A$ is initial, then $\ell(a) = \mathtt{in}$*
   - *if $b \in A$ and there is an initial argument in A which attacks b, then $\ell(b) = \mathtt{out}$*
   - *if $c \in A$ is self-attacking, and there is no attacker of c besides c itself, then $\ell(c) = \mathtt{und}$*
2. *For any set of arguments $\mathcal{I}$ and any labelling $\ell^{\mathcal{I}}$ of $\mathcal{I}$, the AF $\mathcal{AF}' = \langle \mathcal{I}', K'\rangle$, where $\mathcal{I}' = \mathcal{I}\cup\{a' | a \in \mathcal{I}\cap \mathtt{out}(\ell^{\mathcal{I}})\}$ and $K' = \left\{(a',a) | a \in \mathcal{I}\cap\mathtt{out}(\ell^{\mathcal{I}})\right\} \cup \left\{(a,a) | a\in\mathcal{I}\cap\mathtt{und}(\ell^{\mathcal{I}})\right\}$, admits a unique labelling, i.e. $|\mathscr{L}_\sigma(\mathcal{AF}')| = 1$.*

Proposition 6. *The* complete*,* stable*,* semi-stable*,* preferred *and the* grounded *semantics are complete-compatible.*

Proposition 7. *Given a complete-compatible semantics σ, if σ is fully decomposable then there is a unique local function satisfying the conditions of Definition 19, coinciding with the canonical local function $\mathscr{F}_\sigma^{af}$.*

Definition 22 (Top-down and bottom-up decomposability). *Let σ be a complete-compatible semantics and $\mathscr{F}_\sigma^{af}$ be the canonical local function corresponding to σ.*

σ is said to be top-down decomposable iff for any AF $\mathcal{AF} = \langle A,K\rangle$ and any partition $\Omega = \{\omega_1,...,\omega_n\}$ of A, it holds that:

$$\mathscr{L}_\sigma(\mathcal{AF}) \subseteq \left\{\ell_1\cup...\cup\ell_n \middle| \ell_i \in \mathscr{F}_\sigma^{af}\left(\mathcal{AF}\downarrow_{\omega_i}, \omega_i^{inp}, (\bigcup_{j\in\{1,...,n\}\ s.t.\ j\neq i} \ell_j)\downarrow_{\omega_i^{inp}}, \omega_i^K\right)\right\}$$

σ is said to be bottom-up decomposable iff for any AF $\mathcal{AF} = \langle A,K\rangle$ and any partition $\Omega = \{\omega_1,...,\omega_n\}$ of A, it holds that:

$$\mathscr{L}_\sigma(\mathcal{AF}) \supseteq \left\{\ell_1\cup...\cup\ell_n \middle| \ell_i \in \mathscr{F}_\sigma^{af}\left(\mathcal{AF}\downarrow_{\omega_i}, \omega_i^{inp}, (\bigcup_{j\in\{1,...,n\}\ s.t.\ j\neq i} \ell_j)\downarrow_{\omega_i^{inp}}, \omega_i^K\right)\right\}$$

Definition 23 (Partition selector)**.** *A partition selector $\mathscr{S}$ is a function receiving as input an AF $\mathcal{AF} = \langle A, K \rangle$ and returning a set of partitions of A.*

Definition 24 (Decomposability *w.r.t.* a partition selector $\mathscr{S}$)**.** *Let $\mathscr{S}$ be a partition selector. A complete-compatible semantics σ is top-down decomposable w.r.t. $\mathscr{S}$ iff for any AF $\mathcal{AF}$ and any partition $\Omega = \{\omega_1, ..., \omega_n\} \in \mathscr{S}(\mathcal{AF})$, it holds that:*

$$\mathscr{L}_\sigma(\mathcal{AF}) \subseteq \left\{ \ell_1 \cup ... \cup \ell_n \,\middle|\, \ell_i \in \mathscr{F}^{af}_\sigma \left(\mathcal{AF}\downarrow_{\omega_i}, \omega_i^{inp}, (\bigcup_{j \in \{1,...,n\} \text{ s.t. } j \neq i} \ell_j)\downarrow_{\omega_i^{inp}}, \omega_i^K \right) \right\}$$

A complete-compatible semantics σ is bottom-up decomposable w.r.t. $\mathscr{S}$ iff for any argumentation framework AF and any partition $\Omega = \{\omega_1, ..., \omega_n\} \in \mathscr{S}(\mathcal{AF})$, it holds that:

$$\mathscr{L}_\sigma(\mathcal{AF}) \supseteq \left\{ \ell_1 \cup ... \cup \ell_n \,\middle|\, \ell_i \in \mathscr{F}^{af}_\sigma \left(\mathcal{AF}\downarrow_{\omega_i}, \omega_i^{inp}, (\bigcup_{j \in \{1,...,n\} \text{ s.t. } j \neq i} \ell_j)\downarrow_{\omega_i^{inp}}, \omega_i^K \right) \right\}$$

A complete-compatible semantics is fully decomposable w.r.t. a partition selector $\mathscr{S}$ iff it is both top-down and bottom-up decomposable w.r.t. $\mathscr{S}$.

`Note:` *For a semantics σ, to be top-down (resp. bottom-up, fully) decomposable is equivalent to be top-down (resp. bottom-up, fully) decomposable w.r.t. the partition selector that produces all possible partitions of an AF.*

Let formally defined this partition selector:

Definition 25. *$\mathscr{S}_{D\text{-}af}$ is the AF partition selector defined as follows:*

$$\forall \mathcal{AF} \in \Phi_{af}, \mathscr{S}_{D\text{-}af}(\mathcal{AF}) \textit{ is the set of all possible partitions of } \mathcal{AF}$$

Let introduce some useful notations.[2] Let $\mathcal{AF}$ be an AF. We denote by:

- $Paths_{af}(\mathcal{AF})$ the set of all paths of $\mathcal{AF}$
- $Cycles_{af}(\mathcal{AF})$ the set of all cycles of $\mathcal{AF}$
- PE_{af} the path-equivalence relation over an AF (see Definition 128 on page 174), $PE_{af}(\mathcal{AF})$ being the path-equivalence relation over $\mathcal{AF}$
- SCC_{af} the abbreviation for an SCC of AF (that is, an equivalence class of arguments under PE_{af}. See Definition 129 on page 174)
- $SCCS_{af}(\mathcal{AF})$ the set of all SCCs of $\mathcal{AF}$.

Example 8. *Let consider the AF in Figure 2.3 on page 18. We have:*

- $(e', e, f) \in Paths_{af}(\mathcal{AF})$
- $(f, g, f) \in Cycles_{af}(\mathcal{AF})$

[2]See also Section 16.2 on page 170 in the appendix

- $SCCS_{af}(\mathcal{AF}) = \{\{e'\},\{e\},\{d\},\{f,g\},\{h\},\{i\}\}$

Definition 26 (SCC_{af} relation: $\preccurlyeq$)**.** *Let $\mathcal{AF} = \langle A,K\rangle$ be a AF, $S \in SCCS_{af}(\mathcal{AF})$ and $S' \in SCCS_{af}(\mathcal{AF})$. We define the relation $\preccurlyeq$ as a binary relation between elements of $SCCS_{af}(\mathcal{AF})$ as follows:*

$$\begin{gathered} S \preccurlyeq S' \textit{ if and only if} \\ (\exists(e_1,...,e_n) \in Paths_{af}(\mathcal{AF}) \textit{ s.t. } e_1 \in S \textit{ and } e_n \in S') \\ \textit{or} \\ S = S' \end{gathered}$$

Example 9. *Following Example 8, we have:*

- $\{h\} \preccurlyeq \{h\}$
- $\{f,g\} \preccurlyeq \{i\}$
- $\{i\} \not\preccurlyeq \{h\}$

Definition 27 (USCC partition selector)**.** *The USCC partition selector (denoted $\mathscr{S}_{USCC}$) is the partition selector such as for any AF $\mathcal{AF} = \langle A,K\rangle$:*

$$\begin{gathered} \mathscr{S}_{USCC}(\mathcal{AF}) \\ = \\ \left\{\Omega \middle| \ \Omega \textit{ is a partition of } A \textit{ and } \forall S \in SCCS_{af}(\mathcal{AF}), \exists \omega_i \in \Omega \textit{ s.t. } \omega_i \cap S \neq \varnothing \implies S \subseteq \omega_i \right\} \end{gathered}$$

Given an AF $\mathcal{AF} = \langle A,K\rangle$, we call "$USCC_{af}$" a subset $S \subseteq A$ such that S is a union of SCC_{af} of $\mathcal{AF}$.

`Note:` *This partition selector has an interesting property: it does not break the SCC_{af}.*

Example 10. *Following Example 8, as an illustration, we have:*

- $\{\{e',e,f,g\},\{d\},\{h,i\}\} \in \mathscr{S}_{USCC}(\mathcal{AF})$
- $\{\{e',e,f\},\{d,g\},\{h,i\}\} \notin \mathscr{S}_{USCC}(\mathcal{AF})$

Proposition 8. *The semantics properties in Table 2.1 on the next page hold.*

	Complete	*Grounded*	*Preferred*	*Semi-stable*	*Stable*
Full decomposability	✓	×	×	×	✓
Top-down decomposability	✓	✓	✓	×	✓
Bottom-up decomposability	✓	×	×	×	✓
Full decomposability *w.r.t.* $\mathscr{S}_{USCC}$	✓	✓	✓	×	✓
Top-down decomposability *w.r.t.* $\mathscr{S}_{USCC}$	✓	✓	✓	×	✓
Bottom-up decomposability *w.r.t.* $\mathscr{S}_{USCC}$	✓	✓	✓	×	✓

Table 2.1: AF Semantics decomposability properties

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

Chapter 3

AF decision problems and complexities

Dung's Argumentation Framework can be used in processes such as decision making, explanations and auctions. To do so, several decision problems can be useful. In this chapter, classical AF decision problems are defined (Section 3.1), then their complexities are given (Section 3.2 on the next page).

3.1 Definitions

Given the bijection between extension-based semantics and labelling-based semantics, and given that extensions can be transformed into labellings (and vice versa) in polynomial time, AF decision problems can be equivalently defined for the one or the other kind of semantics. We chose to define them using labelling-based semantics. Here are the classical ones:

Definition 28 (Decision Problems in Abstract Argumentation)**.**

- Credulous Acceptance Cred_σ*: Given an AF* $\mathcal{AF} = \langle A, K \rangle$ *and an argument* $a \in A$*. Is* a *labelled* $\mathtt{in}$ *in some* $\ell \in \sigma(\mathcal{AF})$*?*

- Skeptical Acceptance Skep_σ*: Given an AF* $\mathcal{AF} = \langle A, K \rangle$ *and an argument* $a \in A$*. Is* a *labelled* $\mathtt{in}$ *in each* $\ell \in \sigma(\mathcal{AF})$*?*

- Verification of a labelling Ver_σ*: Given an AF* $\mathcal{AF} = \langle A, K \rangle$ *and a labelling* ℓ*. Is* $\ell \in \sigma(\mathcal{AF})$*?*

- Existence of a labelling Exists_σ*: Given an AF* $\mathcal{AF} = \langle A, K \rangle$*. Is* $\sigma(\mathcal{AF}) \neq \varnothing$*?*

- Existence of a "non-empty" labelling $\mathit{Exists}_\sigma^{\neg\varnothing}$*: Given an AF* $\mathcal{AF} = \langle A, K \rangle$*. Does there exist a labelling* $\ell \in \sigma(\mathcal{AF})$ *s.t.* $\mathtt{in}(\ell) \neq \varnothing$*?*

- Uniqueness of a solution Unique_σ*: Given an AF* $\mathcal{AF} = \langle A, K \rangle$*. Is there a unique labelling* $\ell \in \sigma(\mathcal{AF})$*, i.e.* $\sigma(\mathcal{AF}) = \{\ell\}$*?*

Example 11. *Let* σ *be the* preferred *semantics. Let* $\mathcal{AF} = \langle A, K \rangle$ *be the AF represented in Figure 3.1 on the next page and* $\ell \in \mathscr{L}(\mathcal{AF})$ *be a labelling of it. Following Table 1.2 on page 14, we have two* preferred *labellings (ℓ_2 and ℓ_4):*

$$\sigma(\mathcal{AF}) = \left\{ \begin{array}{l} \ell_4 = \left\{ \begin{array}{c} (a, \mathtt{in}), (b, \mathtt{out}), (c, \mathtt{out}), (d, \mathtt{in}), (e, \mathtt{out}), \\ (f, \mathtt{in}), (g, \mathtt{out}), (h, \mathtt{in}), (i, \mathtt{out}), (j, \mathtt{und}), \\ (k, \mathtt{und}), (l, \mathtt{und}), (m, \mathtt{out}), (n, \mathtt{in}) \end{array} \right\}, \\ \ell_2 = \left\{ \begin{array}{c} (a, \mathtt{in}), (b, \mathtt{out}), (c, \mathtt{out}), (d, \mathtt{out}), (e, \mathtt{in}), \\ (f, \mathtt{out}), (g, \mathtt{in}), (h, \mathtt{out}), (i, \mathtt{in}), (j, \mathtt{und}), \\ (k, \mathtt{und}), (l, \mathtt{und}), (m, \mathtt{out}), (n, \mathtt{in}) \end{array} \right\} \end{array} \right\}$$

As a consequence, we have:

- $Cred_\sigma(\mathcal{AF}, l) = \mathtt{false}$
- $Cred_\sigma(\mathcal{AF}, d) = \mathtt{true}$
- $Skep_\sigma(\mathcal{AF}, d) = \mathtt{false}$
- $Skep_\sigma(\mathcal{AF}, n) = \mathtt{true}$
- $Ver_\sigma(\mathcal{AF}, \ell) = \mathtt{true}$ *iff* $\ell \in \{\ell_2, \ell_4\}$
- $Exists_\sigma(\mathcal{AF}) = \mathtt{true}$
- $Exists_\sigma^{\neg\emptyset}(\mathcal{AF}) = \mathtt{true}$
- $Unique_\sigma(\mathcal{AF}) = \mathtt{false}$

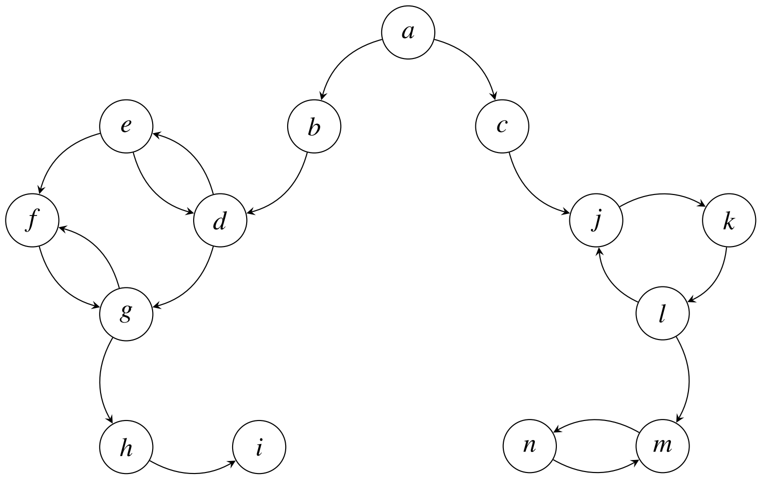


Figure 3.1: The AF of Example 1 on page 7

3.2 Complexities

Table 3.1 on the following page gives the complexity class of the mentioned decision problems for the *grounded*, *complete*, *preferred*, *stable* and *semi-stable* semantics. This table is the result of numerous works (see [41, 40] for a synthesis of these works).[1]

[1] See Section 16.4 on page 178, in the appendix, for the mathematical notions related to the complexity classes.

	$Cred_\sigma$	$Skep_\sigma$	Ver_σ	$Exists_\sigma$	$Exists_\sigma^{\neg\emptyset}$	$Unique_\sigma$
Grounded	$\mathtt{P}$-c	$\mathtt{P}$-c	$\mathtt{P}$-c	trivial	in $\mathtt{L}$	trivial
Complete	$\mathtt{NP}$-c	$\mathtt{P}$-c	in $\mathtt{L}$	trivial	$\mathtt{NP}$-c	$\mathtt{coNP}$-c
Preferred	$\mathtt{NP}$-c	$\Pi_2^{\mathtt{P}}$-c	$\mathtt{coNP}$-c	trivial	$\mathtt{NP}$-c	$\mathtt{coNP}$-c
Stable	$\mathtt{NP}$-c	$\mathtt{coNP}$-c	in $\mathtt{L}$	$\mathtt{NP}$-c	$\mathtt{NP}$-c	$\mathtt{DP}$-c
Semi-stable	$\Sigma_2^{\mathtt{P}}$-c	$\Pi_2^{\mathtt{P}}$-c	$\mathtt{coNP}$-c	trivial	$\mathtt{NP}$-c	in $\Theta_2^{\mathtt{P}}$

Table 3.1: Complexities of Dung's Abstract Framework

Part III

Dung Argumentation Framework: Contribution

Part presentation:

This part presents my contributions in the context of Dung's Argumentation Framework. The main contribution is a Distributed and Clustering-Based algorithm for the enumeration problem. This led to several publications:

- An IRIT report [49], which served as a support for subsequent works: it details the concepts used and provides the proofs that are not in some articles.
- An article in PRIMA 2019, the 22nd edition of the international conference of *Principles and Practice of Multi-Agent Systems* [31], and for which I received a student grant.
- An article in OHAAI 2019, the first edition of the *Online Handbook of Argumentation for Artificial Intelligence* [27]. OHAAI is a platform created in order that PhD students share with others their study subject and the aim of their thesis.
- An article in JIAF 2020, the 4th edition of a French national conference so-called *Journées d'Intelligence Artificielle Fondamentale* [32], which is a second publication of my PRIMA 2019 article for the French community.

The presentation of these contributions is organised as follows. Firstly, the *AFDivider* algorithm is presented and formally studied (Chapter 4 on the next page). Secondly, experiments with the *AFDivider* algorithm are presented (Chapter 5 on page 51). These results led to the idea of a so-called *Compact Enumeration Representation*, which is presented in a third step (Chapter 6 on page 64). To finish, related works are presented (Chapter 7 on page 70).

Chapter 4

AFDivider : presentation and formal analysis

4.1 Motivation

Finding all the possible solutions of a semantics (extension-based or equivalently labelling-based) for a given AF can be very time consuming. Many AF instances, particularly large,[1] are too hard to be solved in an acceptable amount of time, as shown by the results of the ICCMA argumentation solver competition.[2] Formally, the so-called *enumeration problem* is a *function problem* defined as follows:

Definition 29 (Enumeration Problem)**.** *Given an AF $\mathcal{AF} = \langle A, K\rangle$ and a semantics σ, compute the set $\sigma(\mathcal{AF})$ corresponding to the AF solutions.*

The hardness of this problem is not relative to the current state of the art but rather to the intrinsic theoretical complexity of the semantics that are tackled.[3] Moreover, there exists a recent research field in Artificial Intelligence called "Argument Mining" whose object of study is how to extract arguments from natural language speeches, oral or written (see [62] for more information). When major advances in this area will make available a lot of data for argumentation, this issue of solving time may become increasingly critical. There is a need for heuristics, methods and algorithms efficient enough to tackle such issues and make possible the use of automated argumentation models in the large-scale.

Enhancing the computational time of enumerating the solutions of an AF has been the object of study of many works, resulting in the elaboration of several recent algorithms such as [1, 25, 52, 3] (see [26] for an overview). During this thesis, I add my contribution to address this issue with the proposal of an algorithm, the so-called: *AFDivider*.

The idea that led to this algorithm is that argumentation frameworks constructed from real data should have a particular structure. Indeed, people have themes and goals while arguing. It is thus a reasonable

[1]This notion of largeness of an AF is not so simple to define. It is related to the fact that the computation of the solutions is complex either because of the number of arguments, or of the number of interactions, or because of the structure of the AF.

[2]`http://argumentationcompetition.org`

[3]Notice that in the literature it is the decision problem versions of function problems that are studied rather than the actual function problems. This is due to the convenience of *Decision Problem Theory*. The complexity of their decision versions is sufficient to give a good idea of their hardness. See [41] for an overview.

conjecture to say that the AFs obtained from real argumentation are not random and that they have a relatively low density of relations between arguments. As a matter of fact, it is very unlikely that, in some argumentation, each presented argument attacks a large part of the other ones.

The *AFDivider* algorithm takes advantage of this sparsity[4] of AF graphs. To do so, it uses methods that have not yet been considered for this purpose (namely clustering methods used in an original way), combined with techniques that have already been applied in other existing algorithms.

The next section presents formally the *AFDivider* algorithm and summarizes its main steps. Sections 4.2.1 to 4.2.4 on pages 30–42 detail each of these steps and illustrate them.

4.2 *AFDivider* : A Generic algorithm

First of all, let specify the kind of problems addressed. The *AFDivider* algorithm enumerates labelling-based semantics. It has been designed for Dung original semantics: the ***complete***, the ***stable*** and the ***preferred*** semantics. Given that the *grounded* semantics can be computed in linear time and that it gives only one labelling, computing it with the *AFDivider* is unappropriated.

Given an argumentation framework $\mathcal{AF} = \langle A, K \rangle$ and a semantics $\sigma \in \{complete,\ stable,\ preferred\}$, the *AFDivider* algorithm, rather than building labellings that cover the whole AF (which could be time consuming), computes the semantics labellings using a distributed and clustering-based method. Here are its four major steps graphically represented in Figure 4.1:

1. A pretreatment on $\mathcal{AF}$ removes "trivial" parts of it.

2. Clusters in $\mathcal{AF}$ are identified.

3. The labellings under semantics σ in each of these clusters are computed **in parallel**.

4. The results of each cluster are reunified to get the labellings of $\mathcal{AF}$.

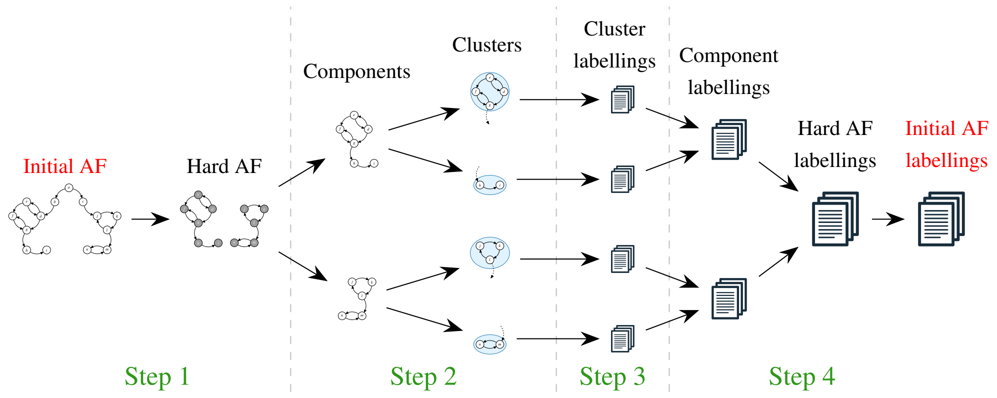


Figure 4.1: *AFDivider* operating diagram

Algorithms 1 and 2 on the next page and on page 31 give the formal definition of the *AFDivider* algorithm. They are said to be generic algorithms in the sense that:

[4] A graph is said to be sparse when its density is low.

- Any clustering method can be used to split the AF.
- Any sound and complete procedure that computes the semantics σ, can be used to compute the labellings of the different clusters.

Algorithm 1: *AFDivider* algorithm.

```
Input: Let $\mathcal{AF} = \langle A, K \rangle$ be an AF and $\sigma$ be a semantics
Result: $\mathscr{L}_\sigma \in 2^{\mathscr{L}(\mathcal{AF})}$: the set of the $\sigma$-labellings of $\mathcal{AF}$
Local variables:
    • $\ell'_{gr}$: the grounded labelling restricted to the arguments labelled in and out
    • CCSet: the set of connected components of $\mathcal{AF}_{hard}$
    • ClustSet: the set of cluster structures of $af_i$
    • $\mathscr{L}_\sigma(af_i)$: the set of all $\sigma$-labellings of $af_i$
1 $\ell'_{gr} \leftarrow$ ComputeGroundedLabelling($\mathcal{AF}$)
2 $CCSet \leftarrow$ SplitConnectedComponents($\mathcal{AF}, \ell'_{gr}$)
3 for all $af_i \in CCSet$ do in parallel
4     $ClustSet \leftarrow$ ComputeClusters($af_i$)
5     $\mathscr{L}_\sigma(af_i) \leftarrow$ ComputeCompLabs($\sigma, ClustSet$)
6 $\mathscr{L}_\sigma \leftarrow \varnothing$
7 if $\nexists af_i \in CCSet$ s.t. $\mathscr{L}_\sigma(af_i) = \varnothing$ then $\mathscr{L}_\sigma \leftarrow \{\ell'_{gr}\} \times \prod_{af_i \in CCSet} \mathscr{L}_\sigma(af_i)$
8 return $\mathscr{L}_\sigma$
```

Let now explain each step of the algorithm. As a running example, we will consider $\mathcal{AF} = \langle A, K \rangle$, the AF represented in Figure 4.2 on page 32, that has been used in Part II, and we will illustrate the algorithm with the *complete* semantics.

4.2.1 Pretreatement: removing AF trivial parts

What we call the “trivial part” (or “fixed part”) of an AF is simply a part of it that has a unique and fixed labelling that can be computed in linear time. As it will be seen in Section 4.2.3 on page 34, for each attack between clusters, several cases have to be considered and this can be very time consuming. In order to avoid this cost for attacks that are in the “trivial part”, we simply cut that part from the AF and, only after that, look for clusters.

Given that we are interested in the *complete*, *stable* and *preferred* semantics, a good way to remove that “trivial part” is to compute the *grounded* labelling of the AF. Indeed, all *complete*, *stable* and *preferred* labellings include the *grounded* one, that is, the arguments labelled *in* or *out* in the *grounded* labelling are labelled in the same way in all σ-labellings. Furthermore, the *grounded* labelling is computable in linear time. This idea of preprocessing has been exploited in [24].

So, given an argumentation framework $\mathcal{AF} = \langle A, K \rangle$, the *AFDivider* algorithm starts with computing the grounded labelling of $\mathcal{AF}$ (Algorithm 1, line 1).

Note: The function ComputeGroundedLabelling($\mathcal{AF}$) *returns a partial labelling of $\mathcal{AF}$ in which the arguments are labelled in or out. In the following we are going to use ℓ_{gr} for the* grounded *labelling and ℓ'_{gr} for the labelling returned by* ComputeGroundedLabelling($\mathcal{AF}$). *That is, und-labelled arguments according to the* grounded *semantics do not belong to ℓ'_{gr}. We have: $\ell'_{gr} = \ell_{gr} \downarrow_{(in(\ell_{gr}) \cup out(\ell_{gr}))}$.*

Algorithm 2: `ComputeCompLabs` algorithm.

```
Input: Let $ClustSet$ be a set of cluster structures for a component $af$, $\sigma$ be a semantics
Result: $\mathscr{L}_\sigma \in 2^{\mathscr{L}(af)}$: the set of the $\sigma$-labellings of $af$
Local variables:
    • $\kappa_j$: a cluster structure
    • $\mathscr{L}_\sigma^{\kappa_j}$: the set of all $\sigma$-labellings of $\kappa_j$
    • $\mathscr{P}^{\kappa_j}$: the set of configurations corresponding to the $\sigma$-labellings of $\kappa_j$
    • $\mathscr{P}$: the set of all reunified labelling profiles
1 for all $\kappa_j \in ClustSet$ do in parallel
2     $\mathscr{L}_\sigma^{\kappa_j} \leftarrow$ ComputeClustLabs$(\sigma, \kappa_j)$
3     $\mathscr{P}^{\kappa_j} \leftarrow$ IdentifyConfigs$(\mathscr{L}_\sigma^{\kappa_j}, \kappa_j)$
4 $\mathscr{L}_\sigma = \varnothing$
5 $\mathscr{P} =$ ReunifyCompConfigs$(\bigcup_{\kappa_j \in ClustSet} \mathscr{P}^{\kappa_j}, ClustSet)$
6 for all $p \in \mathscr{P}$ do
7     $\mathscr{L}_\sigma \leftarrow \mathscr{L}_\sigma \cup \left( \prod_{\xi \in p} \{\ell | \ell \in \texttt{ProfileLabellings}(\xi, \bigcup_{\kappa_j \in ClustSet} \mathscr{L}_\sigma^{\kappa_j})\} \right)$
8 if $\sigma = pr$ then $\mathscr{L}_\sigma \leftarrow \{\ell | \ell \in \mathscr{L}_\sigma$ s.t. $\nexists \ell' \in \mathscr{L}_\sigma$ s.t. $\mathtt{in}(\ell) \subset \mathtt{in}(\ell')\}$
9 return $\mathscr{L}_\sigma$
```

Once the grounded labelling is computed, we consider a restriction $\mathcal{AF}_{hard}$ of $\mathcal{AF}$ to those arguments that are labelled `und` in the grounded labelling:

$$\mathcal{AF}_{hard} = \mathcal{AF}{\downarrow}_{\mathtt{und}(\ell_{gr})}$$

$\mathcal{AF}_{hard}$ may possibly be a disconnected graph. We take advantage of that potential property in order to enhance the parallel computing as it will be explained in the following subsections. The function `SplitConnectedComponents`$(\mathcal{AF}, \ell'_{gr})$ thus split $\mathcal{AF}$ into disjoint sub-AFs obtained after removing the arguments labelled `in` or `out` in the grounded labelling (Algorithm 1, line 2). The *CCSet* variable is the set of the computed connected components.

Given that there are no relations between them, the identification of clusters inside them and the labelling computation of those connected components (Steps 2 and 3) can be made in a simultaneous way (Algorithm 1, line 3) according to the chosen semantics.

`Note:` *We refer to $\mathcal{AF}_{hard}$ in order to facilitate the algorithm explanation however $\mathcal{AF}_{hard}$ does not appear as a concrete data structure entity in the algorithm. When removing the trivial part, the rest of the AF is directly split following the connected components.*

Example 12. *Let consider $\mathcal{AF} = \langle A, K \rangle$, the AF represented in Figure 4.2 on the next page. Its grounded labelling is as follows: $\ell_{gr}(a) = \mathtt{in}$, $\ell_{gr}(b) = \ell_{gr}(c) = \mathtt{out}$ and $\forall x \in A \setminus \{a,b,c\}$, $\ell_{gr}(x) = \mathtt{und}$. Figure 4.3 on page 33 shows an illustration of it: the background color is white for the arguments labelled `in`, black for the labelled `out` and grey for the labelled `und`. We thus have: $\ell'_{gr} = \{(a, \mathtt{in}), (b, \mathtt{out}), (c, \mathtt{out})\}$.*

Given this trivial part, $\mathcal{AF}_{hard} = \mathcal{AF}{\downarrow}_{\{a | a \in A, \ell_{gr}(a) = \mathtt{und}\}}$ is represented in Figure 4.4 on page 34.

There are two connected components in $\mathcal{AF}_{hard}$. The algorithm then splits $\mathcal{AF}_{hard}$ into two distinct AFs that we will call af_1 and af_2, as represented in Figure 4.5 on page 35.

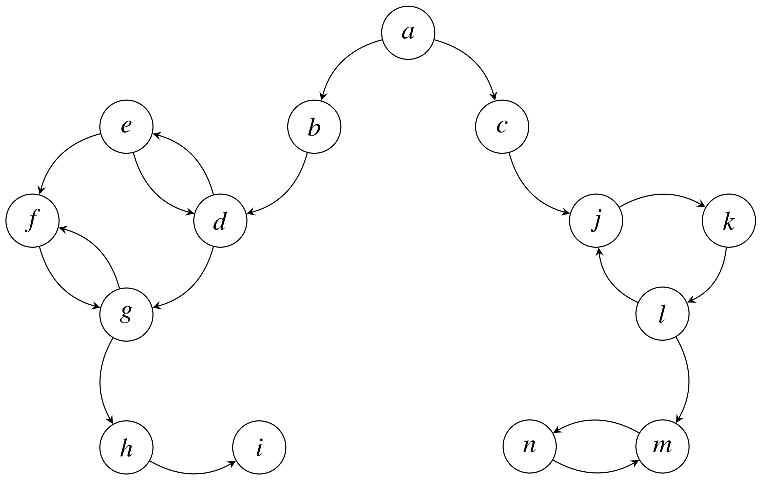


Figure 4.2: AF for the running example

4.2.2 Identifying Clusters

For each of these connected components, a clustering is made (Algorithm 1, line 4) using any clustering method partitioning the AF (even a random partition method). In our experiments, we analyse three clustering methods (see Section 5.2).

We define a data structure, so called "*cluster structure*", to represent each cluster corresponding to the computed partition. These *cluster structures* will be particularly useful for proving the soundness and completeness of our algorithm (See Section 4.3 on page 47).

Definition 30. *(Cluster structure). Let $\mathcal{AF} = \langle A, K \rangle$ be an AF, Ω be the partition of A, ω be an element of Ω (i.e. a set of arguments). $\kappa = \left\langle af, I, O, B \right\rangle$ is a cluster structure defined as follows:*

$$\begin{aligned} af &= \mathcal{AF}\downarrow_{\omega} \\ I &= \{(a,b) | (a,b) \in K, b \in \omega \text{ and } a \notin \omega\} \\ O &= \{(a,b) | (a,b) \in K, b \notin \omega \text{ and } a \in \omega\} \\ B &= \{a \in A | (a,b) \in O \text{ or } (b,a) \in I\} \end{aligned}$$

Note: "I" means "inward attacks", "O" means "outward attacks" and "B" means "border arguments".

Example 13. *Let suppose that the partition computed by the chosen clustering method produces the following partions:*

- *For af_1: $\{\{d,e,f,g\},\{h,i\}\}$.*
- *For af_2: $\{\{j,k,l\},\{m,n\}\}$.*

These partitions are illustrated in Figure 4.6 on page 36. Then a cluster structure is created in order to manipulate each cluster. Formally they are defined as follows:

- $\kappa_1 = \left\langle af_{1.1}, I_1, O_1, B_{1.1} \right\rangle$ *with:*

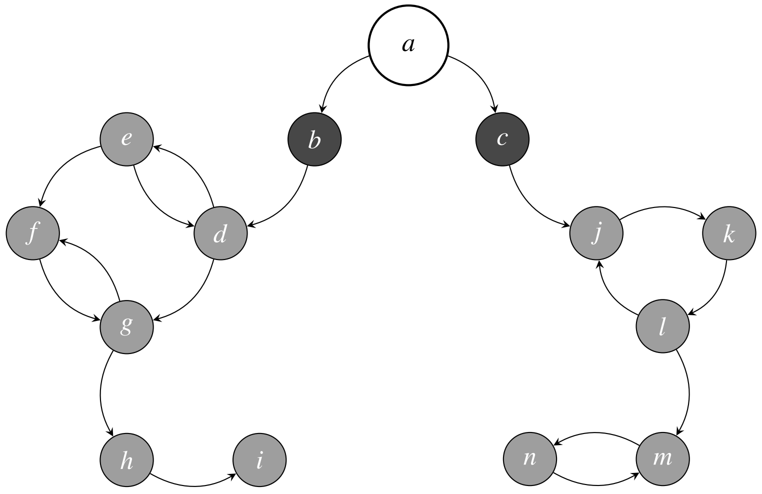


Figure 4.3: Grounded labelling

  - $af_{1.1} = \langle A_1, K_1 \rangle$, *with* $A_1 = \{d, e, f, g\}$ *and* $K_1 = \{(d,e),(e,d),(f,g),(g,f),(e,f),(d,g)\}$
  - $I_1 = \varnothing$
  - $O_1 = \{(g,h)\}$
  - $B_1 = \{g\}$

- $\kappa_2 = \left\langle af_{1.2}, I_2, O_2, B_2 \right\rangle$ *with:*
  - $af_{1.2} = \langle A_2, K_2 \rangle$, *with* $A_2 = \{h, i\}$ *and* $K_2 = \{(h,i)\}$
  - $I_2 = \{(g,h)\}$
  - $O_2 = \varnothing$
  - $B_2 = \{h\}$

- $\kappa_3 = \left\langle af_{1.3}, I_3, O_3, B_3 \right\rangle$ *with:*
  - $af_{1.3} = \langle A_3, K_3 \rangle$, *with* $A_3 = \{j, k, l\}$ *and* $K_3 = \{(j,k),(k,l),(l,j)\}$
  - $I_3 = \varnothing$
  - $O_3 = \{(l,m)\}$
  - $B_3 = \{l\}$

- $\kappa_3 = \left\langle af_{1.3}, I_3, O_3, B_3 \right\rangle$ *with:*
  - $af_{1.4} = \langle A_4, K_4 \rangle$, *with* $A_4 = \{m, n\}$ *and* $K_4 = \{(m,n),(n,m)\}$
  - $I_4 = \{(l,m)\}$
  - $O_4 = \varnothing$
  - $B_4 = \{m\}$

Figure 4.7 on page 37 illustrates them.

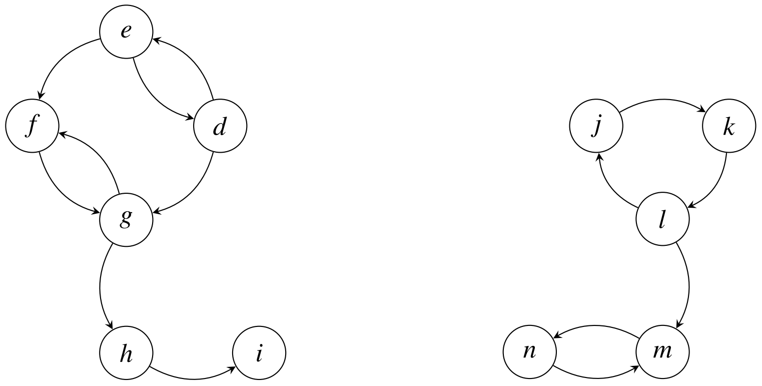


Figure 4.4: $\mathcal{AF}_{hard}$

4.2.3 Computing the labellings

Consider now `ComputeCompLabs` algorithm (Algorithm 2) that computes the component labellings in a distributed way (Algorithm 1, line 5), relying on the clustering made. The σ-labellings of each cluster are computed simultaneously (Algorithm 2, line 1). Unlike the case of connected components used in Algorithm 1, there exist attacks between clusters. In order to compute all the possible σ-labellings of a given cluster, every case concerning its inward attacks (attacks whose target is in the current cluster but the source is from another cluster) has to be considered. Given that the sources of an inward attack could be labelled `in`, `out` or `und` in their own cluster, the σ-labellings of the current cluster have to be computed for all the labelling combinations of inward attack sources.

`Note:` *The number of cases to consider is* 3^n*, with n being the number of inter cluster attack sources. When choosing a clustering, there is thus a threshold between the size of the clusters and the number of edges cut to consider as it effects the overall solving time.*

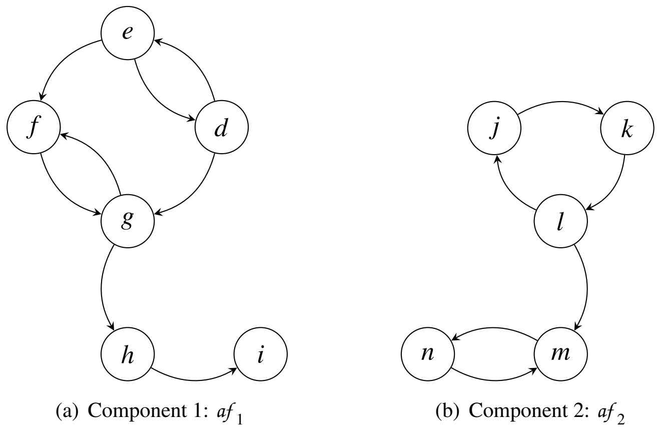
We call "context" a labelling of the cluster inward attack sources. It is formally defined as follows:

Definition 31. *(Context). Let* $\kappa = \langle af, I, O, B \rangle$ *be a cluster structure. A context* μ *of* κ *is a labelling of the inward attack sources of* κ*, i.e.* $\{a | (a,b) \in I\}$.

`Note:` *In the worst case there will be* $3^{|I|}$ *contexts. The exact number of contexts is* $3^{|\{a|(a,b)\in I\}|}$.

Each context induces an AF. Let $\kappa = \langle af, I, O, B \rangle$ be a cluster structure and μ be a context of κ. We can define af', the induced AF of κ for a particular context μ, using the following ideas:

1. af' receives a copy of af
2. $\forall s \in \{s | (s,t) \in I \text{ and } \mu(s) = \texttt{in}\}, \forall t \in \{t | (s,t) \in I\}$, t is removed from af' with all the attacks that have t as endpoint.
3. $\forall s \in \{s | (s,t) \in I \text{ and } \mu(s) = \texttt{und}\}, \forall t \in \{t | (s,t) \in I \text{ and } \nexists (s',t) \in I \text{ s.t. } \mu(s') = \texttt{in}\}$, the attack (t,t) is added to af'.



(a) Component 1: af_1 (b) Component 2: af_2

Figure 4.5: The connected components of $\mathcal{AF}_{hard}$

Note: If $\mu(s) = \mathtt{out}$ there is nothing to do as the attack would have no effect.

Formally induced AF are defined as follows:

Definition 32. *(Induced AF). Let $\kappa = \langle af, I, O, B\rangle$ be a cluster structure with $af = \langle A, K\rangle$ and let μ be a context of κ. The induced AF af' of κ under the context μ is defined as following:*

$$af' = \langle A', K'\rangle$$

With:

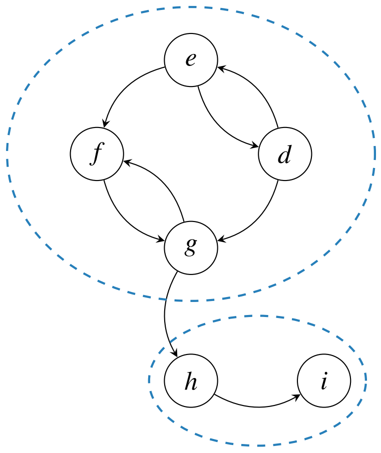
- $A' = A \setminus D$
- $D = \{a | a \in A \text{ and } (s,a) \in I \text{ and } s \in \mathtt{in}(\mu)\}$
- $K' = (K \setminus \{(s,t) | s \in D \text{ or } t \in D\}) \cup \{(a,a) | a \in A \text{ and } (s,a) \in I \text{ and } s \in \mathtt{und}(\mu)\}$

Example 14. *Following Example 13 on page 33, κ_2 has three possible contexts: $\mu_1 = \{(g, \mathtt{out})\}$, $\mu_2 = \{(g, \mathtt{in})\}$ and $\mu_3 = \{(g, \mathtt{und})\}$. Figure 4.8 on page 38 represents the three AFs induced from κ_2 under those contexts.*

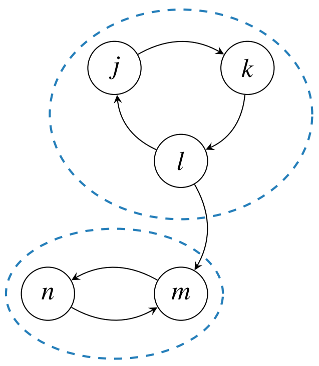
At this step, the computation of the labellings can be done in parallel for each induced AF, using any complete and sound procedure for the semantics σ. This is done by the function `ComputeClustLabs` (Algorithm 2, line 2). These labellings are so-called "*Induced labellings*".

Definition 33. *(Induced labellings). Let $\kappa = \langle af, I, O, B\rangle$ be a cluster structure, with $af = \langle A, K\rangle$, and let μ be a context of κ. Let af' be the induced AF of κ under the context μ, D be the set of arguments such that $D = \{a | a \in A \text{ and } (s,a) \in I \text{ and } s \in \mathtt{in}(\mu)\}$ and ℓ^D be the labelling defined as: $\{(a, \mathtt{out}) | a \in D\}$. The set of induced labellings $\mathscr{L}_\sigma^{\mu(\kappa)}$ of κ under the context μ is defined as following:*

$$\mathscr{L}_\sigma^{\mu(\kappa)} = \{\ell \cup \ell^D | \ell \in \mathscr{L}(af')\}$$



(a) Clusters of af_1



(b) Clusters of af_2

Figure 4.6: Cluster partitions

Once that, for all clusters, the `ComputeClustLabs` function has computed the σ-labellings for all the possible contexts, the σ-labellings are grouped according to their so-called "*configurations*" (Algorithm 2, line 3) to prepare the reunification[5] process. Each induced labelling ℓ is associated to a configuration ξ. This configuration expresses under which conditions an induced labelling, from a given cluster, can be reunified with another one from a neighbour cluster. This configuration is a 5-value labelling on the cluster border arguments (*i.e.* $\forall a \in B$). Configuration are formally defined as follows:

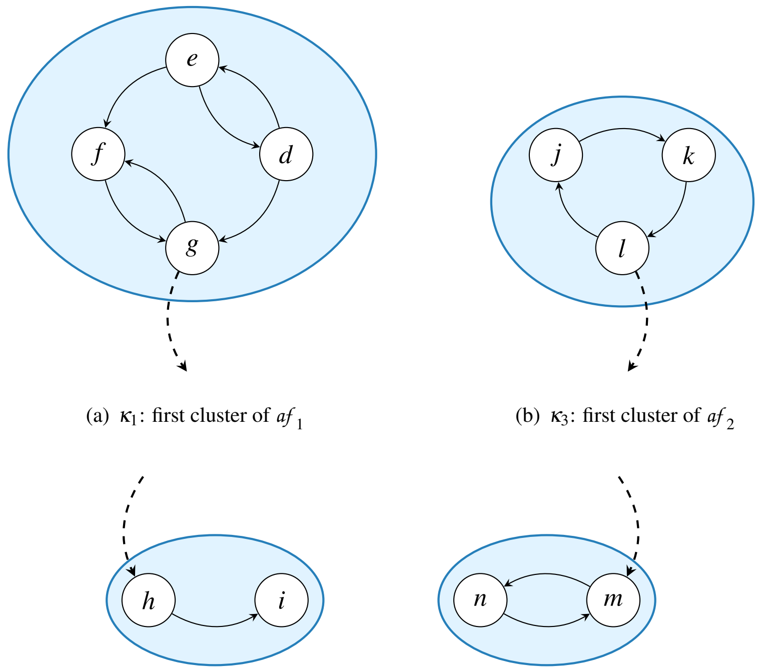
Definition 34. *(Configuration ξ). Let $\kappa = \langle af, I, O, B\rangle$ be a cluster structure, with $af = \langle A, K\rangle$, let μ be a context of κ, and $\ell \in \mathscr{L}_\sigma^{\mu(\kappa)}$ be a computed labelling of κ under μ. Given ℓ, a configuration is a total function $\xi : B \to \{$`in`, `out`, `iout`, `und`, `iund`$\}$ such that:*

$$\xi : a \in B \mapsto \begin{cases} \texttt{in} & \text{if } \ell(a) = \texttt{in} \\ \texttt{out} & \text{if } \ell(a) = \texttt{out} \text{ and } \exists (b,a) \in K \text{ s.t. } \ell(b) = \texttt{in} \\ \texttt{iout} & \text{if } \ell(a) = \texttt{out} \text{ and } \nexists (b,a) \in K \text{ s.t. } \ell(b) = \texttt{in} \\ \texttt{und} & \text{if } \ell(a) = \texttt{und} \text{ and } \nexists (b,a) \in I \text{ s.t. } \mu(b) = \texttt{und} \\ \texttt{iund} & \text{if } \ell(a) = \texttt{und} \text{ and } \exists (b,a) \in I \text{ s.t. } \mu(b) = \texttt{und} \end{cases}$$

In words, for an argument a:

- $\xi(a) =$ `in` means that a is successfully attacked neither from outside nor from inside the cluster. So it is legally `in`.
- $\xi(a) =$ `out` means a is legally `out` from cluster point of view.
- $\xi(a) =$ `iout` means that a is illegally `out` from the cluster point of view.

[5]This process is later described in details. For the moment, we will simply say that it produces valid labellings.



(a) κ_1: first cluster of af_1

(b) κ_3: first cluster of af_2

(c) κ_2: second cluster of af_1

(d) κ_4: second cluster of af_2

Figure 4.7: Cluster structures of the components

- $\xi(a) = \texttt{und}$ means that a is is legally `und` from cluster point of view.
- $\xi(a) = \texttt{iund}$ means that a is illegally `und` from cluster point of view.

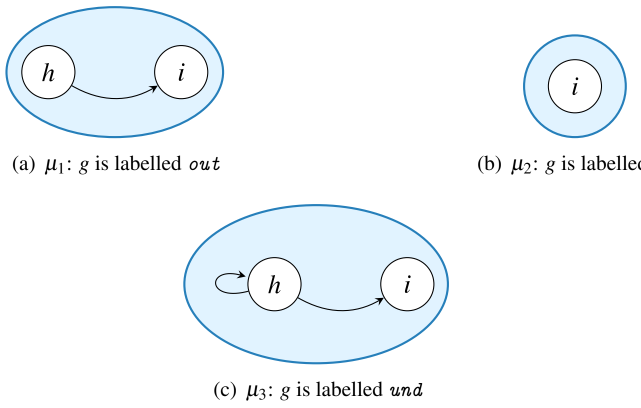
`Note:` *We do not need a value to represent the fact that an argument is illegaly* `in` *because if a border argument is* `in` *then all its attackers must be* `out`*. As defined in Section 4.2.4 on page 42, a simple constraint on endpoint attack labels is sufficient to ensure only such reunifications.*

This is not the case for the values `out` *and* `und`*. Let illustrate that fact. Consider the cluster structure shown in Figure 4.9 on the next page. Let say that because of a certain context μ, a_1 is labelled* `out`*. From the cluster point of view a_3 could be labelled* `in`*,* `out` *or* `und`*, and the same for a_4. Indeed, all these endpoint attack labels couples are valid. An extra constraint is needed to ensure that at least one between a_3 and a_4 is labelled* `in`*. That why we need to differentiate* `out` *and* `iout`*.*

The same reasoning shows that we also need two undecided states (`und` *and* `iund`*).*

`Note:` *When constructing a configuration ξ for a given labelling $\ell \in \mathscr{L}_\sigma^{\mu(\kappa)}$, the distinction between the labels* `out` *and* `iout` *relies on ℓ itself. That is, for an argument a such that $\ell(a) = \texttt{out}$, the value of $\xi(a)$ depends on the fact that there exists or not an attacker $b \in A$ of a such that $\ell(b) = \texttt{in}$.*

The distinction between the labels `und` *and* `iund` *rather relies on the labelling of external attackers in I. That is, for an argument a such that $\ell(a) = \texttt{und}$, the value of $\xi(a)$ depends on the fact that there exists or not an attack $(b,a) \in I$ such that $\ell(b) = \texttt{und}$.*



(a) μ_1: g is labelled $\texttt{out}$ (b) μ_2: g is labelled $\texttt{in}$

(c) μ_3: g is labelled $\texttt{und}$

Figure 4.8: AFs induced from κ_2

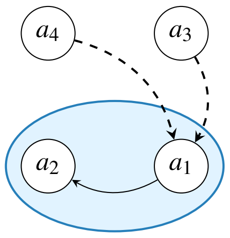


Figure 4.9: Example of the interest of the 5-value labelling.

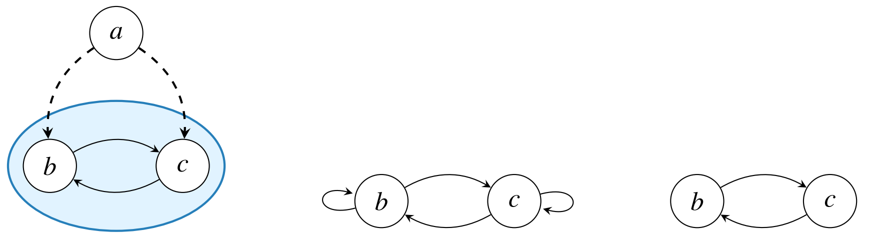
To explain why a different type of definition have been adopted to distinguish $\texttt{und}$ *from* $\texttt{iund}$*, let consider the cluster structure* $\kappa = \langle af, I, O, B \rangle$ *illustrated Figure 4.10 on the following page and its two induced AFs.*

We have: $\mathscr{L}_{pr}^{\mu_1(\kappa)} = \{\{(b,\texttt{und}),(c,\texttt{und})\}\}$ *and* $\mathscr{L}_{pr}^{\mu_2(\kappa)} = \{\{(b,\texttt{in}),(c,\texttt{out})\},\{(b,\texttt{out}),(c,\texttt{in})\}\}$*. Let consider the AF induced from* κ *under* $\mu_1 = \{(a,\texttt{und})\}$ *and the labelling* $\ell = \{(b,\texttt{und}),(c,\texttt{und})\}$*. While establishing the configuration* ξ *associated to* ℓ*, we can observe that although b has an undecided internal attacker, namely c, it is not sufficient to say that b is legally labelled* $\texttt{und}$ *from the cluster point of view, and so that* $\xi(b) = \texttt{und}$*. To prove this, let consider the AF induced from* κ *under* $\mu_2 = \{(a,\texttt{out})\}$*. This latter AF does not admit a* preferred *labelling such that b is labelled* $\texttt{und}$*. As a consequence, the fact that* $\ell(b) = \texttt{und}$ *is due to the fact that* $\mu_1(a) = \texttt{und}$*.*

Following this observation, the configuration of a labelling has been defined as stated in Definition 34 on page 36.

Example 15. *Here is the result according to the* complete *semantics for our running example (See Figure 4.7 on the previous page):*

- *For* κ_1 *we have only one context* $\mu_1^{\kappa_1} = \varnothing$ *(since* $I = \varnothing$*) that gives the labellings and their induced configurations shown in Table 4.1 on the following page.*



(a) Cluster κ (b) AF induced from κ under $\mu_1 = \{(a, \mathtt{und})\}$ (c) AF induced from κ under $\mu_2 = \{(a, \mathtt{out})\}$

Figure 4.10: Example for the distinction between $\mathtt{und}$ and $\mathtt{iund}$

	$\ell_1^{\kappa_1}$	$\xi_1^{\kappa_1}$	$\ell_2^{\kappa_1}$	$\xi_2^{\kappa_1}$	$\ell_3^{\kappa_1}$	$\xi_3^{\kappa_1}$
d	$\mathtt{out}$		$\mathtt{in}$		$\mathtt{und}$	
e	$\mathtt{in}$		$\mathtt{out}$		$\mathtt{und}$	
f	$\mathtt{out}$		$\mathtt{in}$		$\mathtt{und}$	
g	$\mathtt{in}$	$\mathtt{in}$	$\mathtt{out}$	$\mathtt{out}$	$\mathtt{und}$	$\mathtt{und}$

Table 4.1: κ_1 labellings and configurations under $\mu_1^{\kappa_1} = \varnothing$.

- *For κ_2 we have three contexts: $\mu_1^{\kappa_2} = \{(g, \mathtt{out})\}$, $\mu_2^{\kappa_2} = \{(g, \mathtt{in})\}$ and $\mu_3^{\kappa_2} = \{(g, \mathtt{und})\}$. Table 4.2 on the next page gives their corresponding labellings and induced configurations.*
- *For κ_3 we have only one context $\mu_1^{\kappa_3} = \varnothing$ (since $I = \varnothing$) that gives the labellings and their induced configurations shown in Table 4.3 on the next page.*
- *For κ_4 we have three contexts: $\mu_1^{\kappa_4} = \{(l, \mathtt{in})\}$, $\mu_2^{\kappa_4} = \{(l, \mathtt{out})\}$ and $\mu_3^{\kappa_4} = \{(l, \mathtt{und})\}$. Table 4.4 on page 41 gives their corresponding labellings and induced configurations.*

  `Note:` $\xi_1^{\kappa_4}(m) \neq \mathtt{iout}$ because $\ell_1^{\kappa_4}(n) = \mathtt{in}$ and the attack (n,m) exists in κ_4. $\xi_4^{\kappa_4}(m) \neq \mathtt{iund}$ because $\ell_4^{\kappa_4}(n) = \mathtt{und}$ and the attack (n,m) exists in κ_4.

After that we have computed the different labellings and their corresponding configuration, we keep only the "*distinct labellings*" with their "*merge configurations*".

Definition 35. *(Distinct labelling set). Let $\kappa = \langle af, I, O, B \rangle$ be a cluster, let $\mathscr{L}^\kappa = \{\ell_1^\kappa, ..., \ell_n^\kappa\}$ be the set of labellings computed from κ, and $\mathscr{L}_\mathscr{D}^\kappa$ be the distinct labelling set of κ.*

$\mathscr{L}_\mathscr{D}^\kappa$ is defined as following:

$$\mathscr{L}_\mathscr{D}^\kappa = \{\ell_i^\kappa | \ell_i^\kappa \in \mathscr{L}^\kappa \text{ and } \nexists \ell_j^\kappa \in \mathscr{L}^\kappa \text{ s.t. } \ell_j^\kappa = \ell_i^\kappa \text{ and } j < i\}$$

(a) Under $\mu_1^{\kappa_2} = \{(g, \mathtt{out})\}$

	$\ell_1^{\kappa_2}$	$\xi_1^{\kappa_2}$
h	*in*	*in*
i	*out*	

(b) Under $\mu_2^{\kappa_2} = \{(g, \mathtt{in})\}$

	$\ell_2^{\kappa_2}$	$\xi_2^{\kappa_2}$
h	*out*	*iout*
i	*in*	

(c) Under $\mu_3^{\kappa_2} = \{(g, \mathtt{und})\}$

	$\ell_3^{\kappa_2}$	$\xi_3^{\kappa_2}$
h	*und*	*iund*
i	*und*	

Table 4.2: Labellings and configurations of κ_2 (three contexts).

	$\ell_1^{\kappa_3}$	$\xi_1^{\kappa_3}$
j	*und*	
k	*und*	
l	*und*	*und*

Table 4.3: κ_3 labellings and configurations under $\mu_1^{\kappa_3} = \varnothing$.

Example 16. *For κ_4, we have: $\mathscr{L}_{\mathscr{D}}^{\kappa_4} = \{\ell_1^{\kappa_4}, \ell_2^{\kappa_4}, \ell_4^{\kappa_4}\}$.*

`Note:` *It is possible for a labelling to have several and different configurations. These configurations can only differ on* `und` *and* `iund` *labels. The following example illustrates that fact.*

Example 17. *Let consider Figure 4.11. There is no case in which the argument a can be labelled* `in`*. Let thus consider only the contexts of the right cluster that can possibly lead to valid reunified labellings, which are: $\{(a, \mathtt{out})\}$ and $\{(a, \mathtt{und})\}$. In both cases, we have a unique labelling $\{(b, \mathtt{und})\}$. Nevertheless, considering the configurations, we obtain two distinct ones. For the first context, we have: $\{(b, \mathtt{und})\}$ using the forth rule of the configuration definition (Definition 34 on page 36) and $\{(b, \mathtt{iund})\}$ using the last rule.*

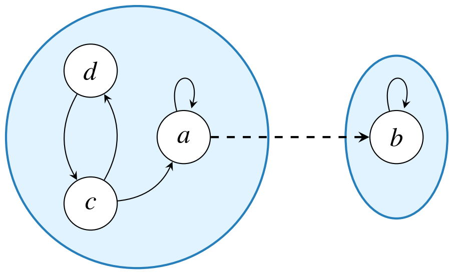


Figure 4.11: Illustration of merge configuration

Given that it is possible for a labelling to have several and different configurations, we introduce the notion of "*merge configuration*".

(a) Under $\mu_1^{\kappa_4} = \{(l, \mathtt{in})\}$

	$\ell_1^{\kappa_4}$	$\xi_1^{\kappa_4}$
m	$\mathtt{out}$	$\mathtt{out}$
n	$\mathtt{in}$	

(b) Under $\mu_2^{\kappa_4} = \{(l, \mathtt{out})\}$

	$\ell_2^{\kappa_4}$	$\xi_2^{\kappa_4}$	$\ell_3^{\kappa_4}$	$\xi_3^{\kappa_4}$	$\ell_4^{\kappa_4}$	$\xi_4^{\kappa_4}$
m	$\mathtt{in}$	$\mathtt{in}$	$\mathtt{out}$	$\mathtt{out}$	$\mathtt{und}$	$\mathtt{und}$
n	$\mathtt{out}$		$\mathtt{in}$		$\mathtt{und}$	

(c) Under $\mu_3^{\kappa_4} = \{(l, \mathtt{und})\}$

	$\ell_5^{\kappa_4}$	$\xi_5^{\kappa_4}$	$\ell_6^{\kappa_4}$	$\xi_6^{\kappa_4}$
m	$\mathtt{out}$	$\mathtt{out}$	$\mathtt{und}$	$\mathtt{iund}$
n	$\mathtt{in}$		$\mathtt{und}$	

Table 4.4: κ_4 labellings and configurations (three contexts).

Definition 36. *(Merge configuration). Let $\kappa = \langle af, I, O, B \rangle$ be a cluster, let $\mathscr{L}^\kappa = \{\ell_1^\kappa, ..., \ell_n^\kappa\}$ be the set of labellings and $\mathscr{C}^\kappa = \{\xi_1^\kappa, ..., \xi_n^\kappa\}$ the set of their corresponding configurations computed from κ, let $\ell_i^\kappa \in \mathscr{L}^\kappa$ be a labelling and $\mathscr{C}_{\ell_i^\kappa} = \{\xi_j^\kappa | \xi_j^\kappa \in \mathscr{C}^\kappa$ s.t. $\ell_j^\kappa = \ell_i^\kappa\}$ be the set of all its possible configurations. Let $\xi \in \mathscr{C}_{\ell_i^\kappa}$ be a possible configuration of ℓ_i^κ.*

The merge configuration $\xi_{\ell_i^\kappa}$ of ℓ_i^κ is defined as follows:

$$\forall a \in B, \xi_{\ell_i^\kappa}(a) = \begin{cases} \mathtt{in} & \textit{if } \ell_i^\kappa(a) = \mathtt{in} \\ \mathtt{out} & \textit{if } \ell_i^\kappa(a) = \mathtt{out} \textit{ and } \exists (b,a) \in af \textit{ s.t. } \ell(b) = \mathtt{in} \\ \mathtt{iout} & \textit{if } \ell_i^\kappa(a) = \mathtt{out} \textit{ and } \nexists (b,a) \in af \textit{ s.t. } \ell(b) = \mathtt{in} \\ \mathtt{und} & \textit{if } \ell_i^\kappa(a) = \mathtt{und} \textit{ and } \exists \xi \in \mathscr{C}_{\ell_i^\kappa} \textit{ s.t. } \xi(a) = \mathtt{und} \\ \mathtt{iund} & \textit{otherwise} \end{cases}$$

The merge configuration as defined is the most flexible configuration of a given labelling. It ensures all the requirements for a good reunification without adding unwanted restrictions.

Example 18. *This step will affect only the cluster κ_4 as $\ell_1^{\kappa_4} = \ell_3^{\kappa_4} = \ell_5^{\kappa_4}$ and $\ell_4^{\kappa_4} = \ell_6^{\kappa_4}$. The new set of labelling/configuration of κ_4 is shown on Table 4.5 on the next page.*

We can notice after this filtering and merging process that:

- One context can give several labellings.
- From a labelling is induced one and only one merge configuration.
- Several labellings can induce the same merge configuration (See Example 19 on the next page).

	${\ell_1}'^{\kappa_4}$	${\xi_1}'^{\kappa_4}$	${\ell_2}'^{\kappa_4}$	${\xi_2}'^{\kappa_4}$	${\ell_3}'^{\kappa_4}$	${\xi_3}'^{\kappa_4}$
m	$\mathtt{out}$	$\mathtt{out}$	$\mathtt{in}$	$\mathtt{in}$	$\mathtt{und}$	$\mathtt{und}$
n	$\mathtt{in}$		$\mathtt{out}$		$\mathtt{und}$	

Table 4.5: κ_4 labellings and configurations.

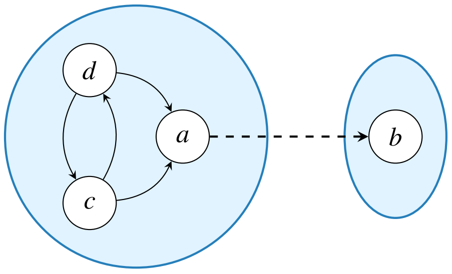


Figure 4.12: Same merge configuration for different labellings

At this step, everything is ready to start the reunification process.

Example 19. *Let κ be the cluster structure of left illustrated in Figure 4.12. κ has only one context: $\mu = \varnothing$ since it does not have inward attacks. κ has thus three* complete *labellings: $\ell_1 = \{(d, \mathtt{in}), (c, \mathtt{out}), (a, \mathtt{out})\}$, $\ell_2 = \{(d, \mathtt{out}), (c, \mathtt{in}), (a, \mathtt{out})\}$ and $\ell_3 = \{(d, \mathtt{und}), (c, \mathtt{und}), (a, \mathtt{und})\}$. You can notice that both ℓ_1 and ℓ_2 have the same merge configuration which is $\xi = \{(a, \mathtt{out})\}$.*

4.2.4 Reunifying the results

The labelling reunifying process is made in two steps: firstly, the reunification of the component labellings (*i.e.* the reunification of their cluster labellings together) and secondly, the reunification of the whole AF labellings (*i.e.* the reunification of its component labellings together). The first step is done in Algorithm 2 on page 31, lines 5 to 8. The second one is in Algorithm 1 on page 30, line 7. Let detail them.

4.2.4.1 Component labelling reunification

In [49, 31], the first version of our algorithm, the reunification was directed made on the cluster labellings. We, later, found out that this could be enhanced if we reunify the labelling configurations instead. Indeed, this deals with much less elements as several cluster labellings could have the same configuration. We introduced so the notion of "*reunified labelling profiles*".

Definition 37. *(Reunified labelling profiles). Let $af = \langle A, K\rangle$ be an AF and $\{\kappa_1, ..., \kappa_n\}$ be the set of cluster structures corresponding to the clustering of af. Let $\{\mathscr{L}_\sigma^{\kappa_1}, ..., \mathscr{L}_\sigma^{\kappa_n}\}$ be the set of distinct labelling sets of each cluster of af and $\{\xi^{\kappa_1}, ..., \xi^{\kappa_n}\}$ be the set of their corresponding merge configuration sets. Let $\{I_1, ..., I_n\}$ be the set of sets of inward attacks of the different clusters and $I = \bigcup_{i=1}^n I_i$ be the union of those sets. Let $\{B_1, ..., B_n\}$ be the set of the sets of their border arguments and $B = \bigcup_{i=1}^n B_i$ be the union of those sets. Let $\{\xi_1, ..., \xi_n\}$ be a set of configurations such that, for all $i \in \{1, ..., n\}$, $\xi_i \in \xi^{\kappa_i}$. Let $\Xi = (\bigcup_{i=1}^n \xi_i)$*

be the union of these configurations. $p = \{\xi_1, ..., \xi_n\}$ *is a* reunified labelling profile *(or equivalently, the configurations* $\xi_1, ..., \xi_n$ *are said to be compatible together) if and only if:*

$$\forall a \in B, \quad \begin{cases} \Xi(a) = \mathtt{iout} & \implies \exists (b,a) \in I \text{ s.t. } \Xi(b) = \mathtt{in} \\ \Xi(a) = \mathtt{iund} & \implies \begin{pmatrix} \exists (b,a) \in I \text{ s.t. } \Xi(b) = \mathtt{und} \text{ or } \Xi(b) = \mathtt{iund} \\ \text{and} \\ \nexists (b,a) \in I \text{ s.t. } \Xi(b) = \mathtt{in} \end{pmatrix} \\ \Xi(a) = \mathtt{in} & \implies \forall (b,a) \in I, \Xi(b) = \mathtt{out} \text{ or } \Xi(b) = \mathtt{iout} \\ \Xi(a) = \mathtt{und} & \implies \forall (b,a) \in I, \Xi(b) \neq \mathtt{in} \end{cases}$$

`Note:` *Distinct reunified labellings can have the same profile, but a labelling only has one profile.*

This notion defined, let continue the explanation of the *AFDivider* algorithm.

At this step, the `ReunifyCompConfigs` function is called (Algorithm 2, line 5) in order to reunify the compatible configurations of the cluster labellings together. To do that, the `ReunifyCompConfigs` transforms that reunifying problem into a *constraint satisfaction problem* (CSP).

`Note:` *We chose to use this method because in CSP modeling each variable can have an arbitrary value domain and constraints can have any arity and be of any nature. Those two properties make CSP modeling very straightforward and easy to automatise. Furthermore, there are a lot of CSP solvers available.*

This being said, here are the four steps of the transformation process:

1. For each cluster κ_i, a variable V_{κ_i} is created. For each of them, the domain is the set of their computed labelling merge configurations ξ^{κ_i}.
2. For each border argument a_j, a variable V_{a_j} is created with a domain corresponding to their possible labels, *i.e.* $\{\mathtt{in}, \mathtt{out}, \mathtt{und}\}$.
3. For each inter-cluster attack (a,b), a constraint is added with the following set of valid tuples:
   $\{(a = \mathtt{in}, b = \mathtt{out}), (a = \mathtt{out}, b = \mathtt{in}), (a = \mathtt{out}, b = \mathtt{out}),$
   $(a = \mathtt{out}, b = \mathtt{und}), (a = \mathtt{und}, b = \mathtt{out}), (a = \mathtt{und}, b = \mathtt{und})\}$

   Let $\kappa_i = \langle af, I, O, B \rangle$ be a cluster structure and ℓ be one of its induced labellings. Let $\xi_\ell^{\kappa_i}$ be a value of the domain of V_{κ_i} corresponding to the merge configuration of ℓ.

4. For each $\xi_\ell^{\kappa_i}$ in V_{κ_i} domain:

   (a) Constraints are added to map the merge configuration with the argument labels. The constraints are defined as following:

   $$\forall a_j \in B, \quad \begin{array}{l} (V_{\kappa_i} = \xi_\ell^{\kappa_i} \wedge \xi_\ell^{\kappa_i}(a_j) = \mathtt{in}) \implies V_{a_j} = \mathtt{in} \\ (V_{\kappa_i} = \xi_\ell^{\kappa_i} \wedge (\xi_\ell^{\kappa_i}(a_j) = \mathtt{out} \vee \xi_\ell^{\kappa_i}(a_j) = \mathtt{iout})) \implies V_{a_j} = \mathtt{out} \\ (V_{\kappa_i} = \xi_\ell^{\kappa_i} \wedge (\xi_\ell^{\kappa_i}(a_j) = \mathtt{und} \vee \xi_\ell^{\kappa_i}(a_j) = \mathtt{iund})) \implies V_{a_j} = \mathtt{und} \end{array}$$

   (b) Constraints are added for all arguments labelled $\mathtt{iout}$ in $\xi_\ell^{\kappa_i}$:

   $$\forall a_j \in \{a | \xi_\ell^{\kappa_i}(a) = \mathtt{iout}\}, \qquad V_{\kappa_i} = \xi_\ell^{\kappa_i} \implies \exists (a_k, a_j) \in I \text{ s.t. } V_{a_k} = \mathtt{in}$$

(c) Constraint are added for all arguments labelled $\mathtt{iund}$ in $\xi_\ell^{\kappa_i}$:

$$\forall a_j \in \{a|\xi_\ell^{\kappa_i}(a) = \mathtt{iund}\}, \qquad V_{\kappa_i} = \xi_\ell^{\kappa_i} \implies \exists(a_k, a_j) \in I \text{ s.t. } V_{a_k} = \mathtt{und}$$

`Note`*: The constraints have to be seen as declarative rules. For example the rule:* $V_{\kappa_i} = \xi_\ell^{\kappa_i} \implies \exists(a_k, a_j) \in I$ *s.t.* $V_{a_k} = \mathtt{und}$ *as to be understand as "If the variable* V_{κ_i} *has the value* $\xi_\ell^{\kappa_i}$*, there must be a variable corresponding to one of the attackers of* a_j *that takes the value* $\mathtt{und}$*".*

The solutions of that CSP modelling are the reunified labelling profiles (corresponding to values of the V_{κ_i} variables).

Example 20. *Let illustrate this with the CSP modelisation for the reunification of* af_1*. Let* $\Psi_{af_1} = \langle X, D, C\rangle$ *be that modelling.* Ψ_{af_1} *is defined as following:*

- $X = \{V_{\kappa_1}, V_{\kappa_2}, V_g, V_h\}$
- $D = \left\{ \begin{array}{l} D(V_{\kappa_1}) = \{\xi_1^{\kappa_1}, \xi_2^{\kappa_1}, \xi_3^{\kappa_1}\}, \\ D(V_{\kappa_2}) = \{\xi_1^{\kappa_2}, \xi_2^{\kappa_2}, \xi_3^{\kappa_2}\}, \\ D(V_g) = \{\mathtt{in}, \mathtt{out}, \mathtt{und}\}, \\ D(V_h) = \{\mathtt{in}, \mathtt{out}, \mathtt{und}\} \end{array} \right\}$
- $C = \{c_1, c_2, c_3, c_4, c_5\}$ *is a set of constraints, with*
  - c_1 *being the constraint that expresses the attack relation from g to h, corresponding to Step 3,*
  - c_2 *being the constraint expressing the fact that the merge configurations of* κ_1 *impose a label on each of its border arguments (i.e. on g), corresponding to Step 4a,*
  - c_3 *being the constraint expressing the fact that the merge configurations of* κ_2 *impose a label on each of its border arguments (i.e. on h), corresponding to Step 4a,*
  - c_4 *being the constraint expressing the fact that* $\xi_2^{\kappa_2}$ *can only be reunified with a merge configuration of* κ_1 *in which g is labelled* $\mathtt{in}$*, corresponding to Step 4b (see Table 4.1 on page 39 and Table 4.2 on page 40),*
  - c_5 *being the constraint expressing the fact that* $\xi_3^{\kappa_2}$ *can only be reunified with a merge configuration of* κ_1 *in which g is labelled* $\mathtt{und}$*, corresponding to Step 4c (see Table 4.1 on page 39 and Table 4.2 on page 40).*

`Note:` c_4 and c_5 are constraints only for precise merge configurations of κ_2. c_4 and c_5 must allow g being labelled with any label if the reunification is about another merge configuration of κ_2.

c_1 *accepts only the following tuples:*

- $(V_g = \mathtt{in}, V_h = \mathtt{out})$
- $(V_g = \mathtt{out}, V_h = \mathtt{in})$
- $(V_g = \mathtt{out}, V_h = \mathtt{out})$
- $(V_g = \mathtt{out}, V_h = \mathtt{und})$

- $(V_g = \mathtt{und}, V_h = \mathtt{out})$
- $(V_g = \mathtt{und}, V_h = \mathtt{und})$

c_2 accepts only the following tuples (see Table 4.1 on page 39):

- $(V_{\kappa_1} = \xi_1^{\kappa_1}, V_g = \mathtt{in})$
- $(V_{\kappa_1} = \xi_2^{\kappa_1}, V_g = \mathtt{out})$
- $(V_{\kappa_1} = \xi_3^{\kappa_1}, V_g = \mathtt{und})$

c_3 accepts only the following tuples (see Table 4.2 on page 40):[6]

- $(V_{\kappa_2} = \xi_1^{\kappa_2}, V_h = \mathtt{in})$
- $(V_{\kappa_2} = \xi_2^{\kappa_2}, V_h = \mathtt{out})$
- $(V_{\kappa_2} = \xi_3^{\kappa_2}, V_h = \mathtt{und})$

c_4 accepts only the following tuples:

- $(V_{\kappa_2} = \xi_2^{\kappa_2}, V_g = \mathtt{in})$
- $(V_{\kappa_2} = \xi_3^{\kappa_2}, V_g = \mathtt{in})$
- $(V_{\kappa_2} = \xi_3^{\kappa_2}, V_g = \mathtt{out})$
- $(V_{\kappa_2} = \xi_3^{\kappa_2}, V_g = \mathtt{und})$
- $(V_{\kappa_2} = \xi_1^{\kappa_2}, V_g = \mathtt{in})$
- $(V_{\kappa_2} = \xi_1^{\kappa_2}, V_g = \mathtt{out})$
- $(V_{\kappa_2} = \xi_1^{\kappa_2}, V_g = \mathtt{und})$

c_5 accepts only the following tuples:

- $(V_{\kappa_2} = \xi_3^{\kappa_2}, V_g = \mathtt{und})$
- $(V_{\kappa_2} = \xi_2^{\kappa_2}, V_g = \mathtt{in})$
- $(V_{\kappa_2} = \xi_2^{\kappa_2}, V_g = \mathtt{out})$
- $(V_{\kappa_2} = \xi_2^{\kappa_2}, V_g = \mathtt{und})$
- $(V_{\kappa_2} = \xi_1^{\kappa_2}, V_g = \mathtt{in})$
- $(V_{\kappa_2} = \xi_1^{\kappa_2}, V_g = \mathtt{out})$
- $(V_{\kappa_2} = \xi_1^{\kappa_2}, V_g = \mathtt{und})$

`Note:` Both c_4 and c_5 have to be respected. As a consequence, the valid tuples concerning V_{κ_2} and V_g are the ones which are both in c_4's valid tuples and c_5's valid tuples. So the second and third tuples accepted by c_4 and the third and fourth tuples accepted by c_5 will be useless.

The solutions for af_1 are:

[6]Notice that in the CSP modeling the border arguments can only be labelled $\mathtt{in}$, $\mathtt{out}$ or $\mathtt{und}$. The potential illegal aspect of labellings (that is, labels $\mathtt{iout}$ and $\mathtt{iund}$, as it can be seen in Table 4.2 on page 40) is captured by the constraints added in Steps 4b and 4c. In this example, it corresponds to constraints c_4 and c_5.

- $\{\xi_1^{\kappa_1} = \{(g, \mathtt{in})\}, \xi_2^{\kappa_2} = \{(h, \mathtt{iout})\}\}$
- $\{\xi_2^{\kappa_1} = \{(g, \mathtt{out})\}, \xi_1^{\kappa_2} = \{(h, \mathtt{in})\}\}$
- $\{\xi_3^{\kappa_1} = \{(g, \mathtt{und})\}, \xi_3^{\kappa_2} = \{(h, \mathtt{iund})\}\}$

Using the same process we obtain the following results for af_2:

- $\{\xi_1^{\kappa_3} = \{(l, \mathtt{und})\}, \xi_1'^{\kappa_4} = \{(m, \mathtt{out})\}\}$
- $\{\xi_1^{\kappa_3} = \{(l, \mathtt{und})\}, \xi_3'^{\kappa_4} = \{(m, \mathtt{und})\}\}$

For each of the reunified labelling profile computed (Algorithm 2, line 6 and 7), labelling parts corresponding to the configurations forming the reunified profile are reunified together.

Example 21. *Following Example 20, Table 4.6 shows the* complete *labellings obtained for af_1 and Table 4.7 shows the ones for af_2.*

	$\ell_{1.1}$	$\ell_{1.2}$	$\ell_{1.3}$
d	out	in	und
e	in	out	und
f	out	in	und
g	in	out	und
h	out	in	und
i	in	out	und

Table 4.6: Complete labellings

	$\ell_{2.1}$	$\ell_{2.2}$
j	und	und
k	und	und
l	und	und
m	und	out
n	und	in

Table 4.7: Complete labellings

A special step has to be done for the *preferred* semantics as this reunifying process does not ensure the maximality (*w.r.t.* $\subseteq$) of the set of $\mathtt{in}$-labelled arguments (so not all of the labellings produced in Algorithm 2, line 7 are *preferred* ones). Indeed, the preferred semantics is not bottom-up decomposable (see Proposition 8 on page 21). A maximality check is done (Algorithm 2, line 8) in order to keep only the wanted labellings.

`Note:` *This maximality check has a complexity of $\Theta(n^2)$, n being the number of component labelling produced by the algorithm. As a consequence, if n is large, it could be very time consuming. To enhance this check, an optimization have been made: check only the reunified labellings whose reunified profile labelling contains an $\mathtt{und}$-labelled argument. Indeed, the* preferred *semantics is top-down decomposable. As all the cluster labellings are maximal w.r.t. their corresponding context, the only way for a reunified labelling to not be maximal is to have an $\mathtt{und}$-labelled argument at the cluster border. Experimental analysis show the interest of this optimization. See Section 5.3.4 on page 58.*

`Note:` *When computing the* preferred *semantics with a partition selector that does not cut SCC_{af}, the maximality check is not necessary. Indeed, any reunified component labelling will be maximal. See the proof of Proposition 11 for more details.*

`Note:` *When computing the* stable *semantics, the set of labellings $\mathscr{L}_\sigma$ returned by* `ComputeCompLabs` *may be empty. It happens when one of the component clusters has no* stable *labelling.*

4.2.4.2 Whole AF labelling reunification

Now that all the component labellings are built, we can reunify the labellings of the whole AF. Indeed, given that ℓ_{gr} is a fixed part of all σ-labellings of $\mathcal{AF}$ and that all the connected components are completely independent, the building of the σ-labellings of the whole AF is made with a simple Cartesian product (Algorithm 1, line 7) between the labellings of all the components and the grounded one. If one of the components has no labelling then the whole AF has no labelling (so $\mathscr{L}_\sigma = \varnothing$).

Example 22. *To finish our illustration, following Example 21 on the previous page and following Example 12 on page 31, the* complete *labellings of $\mathcal{AF}$ are shown in Table 4.8 on the next page.*

4.3 *AFDivider* properties: soundness and completeness

In this section we give properties ensuring that the *AFDivider* algorithm works well. More precisely, we ensure that it gives all the expected labellings for the *complete*, *stable* and *preferred* semantics; this is the notion of completeness, and that the algorithm gives only good labellings for the semantics *complete*, *stable* and *preferred* ; this is the notion of soundness.

In [9], Baroni et al. introduce several notions and proved semantics properties that are useful to prove that our proposed algorithms are sound and complete. They are presented in Chapter 2 on page 16, the most important being fully decomposable and top-down decomposable semantics properties.

A semantics will be a fully decomposable or top-down decomposable semantics if for any AF and any partition of this AF, it is possible to reconstruct all the labellings of the whole AF by combining the labellings (under the same semantic) of the partition parts.

To be more precise, the difference between a top-down decomposable semantics and a fully decomposable one is that for a top-down decomposable one, when doing this process of labelling part reunification,

	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5	ℓ_6
a	*in*	*in*	*in*	*in*	*in*	*in*
b	*out*	*out*	*out*	*out*	*out*	*out*
c	*out*	*out*	*out*	*out*	*out*	*out*
d	*out*	*out*	*in*	*in*	*und*	*und*
e	*in*	*in*	*out*	*out*	*und*	*und*
f	*out*	*out*	*in*	*in*	*und*	*und*
g	*in*	*in*	*out*	*out*	*und*	*und*
h	*out*	*out*	*in*	*in*	*und*	*und*
i	*in*	*in*	*out*	*out*	*und*	*und*
j	*und*	*und*	*und*	*und*	*und*	*und*
k	*und*	*und*	*und*	*und*	*und*	*und*
l	*und*	*und*	*und*	*und*	*und*	*und*
m	*und*	*out*	*und*	*out*	*und*	*out*
n	*und*	*in*	*und*	*in*	*und*	*in*

Table 4.8: *Complete* labellings of $\mathcal{AF}$

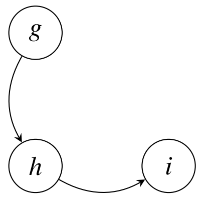
all the semantics labellings will be found but it is also possible to obtain non correct labellings, whereas, for a fully decomposable all and only the correct semantics labellings will be obtained.

As one may notice, in the *AFDivider* algorithm we do not use the notion of AF with input (introduced in [9], see Chapter 2). Instead, we use the notion of cluster structure. There is definitely a link between these two notions, link that we will present before proving the soundness and the completeness of our algorithm.

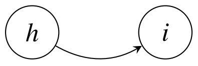
4.3.1 Relation between AFs with input and cluster structures

The aim of this comparison is to use semantics decomposability properties for cluster structures. The following example illustrates the differences between the two approaches.

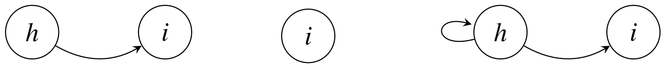
Example 23. *Consider the following AF, $\mathcal{AF} = \langle A, K \rangle$:*



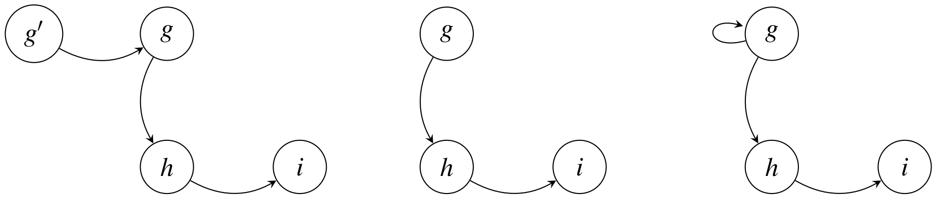
Given $\omega = \{h, i\}$, $af = \mathcal{AF} \downarrow_{\omega}$ is represented as follows:



Considering our approach, the cluster structure for ω is $\kappa = \langle af, I = \{(g,h)\}, O = \varnothing, B = \{h\}\rangle$.
Then three contexts exist: $\mu_1 = \{(g, \mathtt{out})\}$, $\mu_2 = \{(g, \mathtt{in})\}$, $\mu_3 = \{(g, \mathtt{und})\}$. And so three induced AFs can be defined (for respectively μ_1, μ_2, and μ_3):



Considering the approach proposed by Baroni and co., the AF with input corresponding to ω is defined by $\langle af, \{g\}, \mu, \{(g,h)\}\rangle$ with μ being either μ_1, or μ_2, or μ_3. So three standard AFs can be defined (for respectively μ_1, μ_2, and μ_3):



Relying on the notion of complete-based semantics (Definition 38), Proposition 9 gives the correspondence between our induced AFs and the standard AFs.

Definition 38 (Complete-based semantics)**.** *A semantics σ is complete-based if and only if the following condition holds:*

$$\forall \mathcal{AF} \in \Phi_{af}, \mathscr{L}_{\sigma}(\mathcal{AF}) \subseteq \mathscr{L}_{co}(\mathcal{AF})$$

`Note:` *By definition complete-based semantics are also complete-compatible (See Definition 21 on page 19).*

Proposition 9. *Let σ be a complete-based semantics. Let $\mathcal{AF} = \langle A, K\rangle$ be an AF and $\omega \subseteq A$ be a set of arguments. Let $af = \langle \omega, K \cap (\omega \times \omega)\rangle$ be the restricted AF corresponding to $\mathcal{AF} \downarrow_{\omega}$. Let $\kappa = \langle af, I = \omega^K$, $O = K \cap (\omega \times (A \setminus \omega))$, $B = \{a | (a,b) \in O \text{ or } (b,a) \in I\}\rangle$ be the cluster structure corresponding to ω. Let μ be a context of κ. The following equation holds:*

$$\mathscr{L}_{\sigma}^{\mu(\kappa)} = \mathscr{F}_{\sigma}(af, \omega^{inp}, \mu, \omega^K)$$

`Note:` As a recall, see Definition 15 on page 16 for ω^K and ω^{inp}.

☐ Proof of Proposition 9: link (See page 219).

4.3.2 Soundness and completeness

First, we have for Algorithm 2 the two following properties:

Proposition 10 (Completeness of Algorithm 2)**.** *Algorithm 2 is complete for the* stable*,* complete *and* preferred *semantics.*

☐ Proof of Proposition 10: link (See page 221).

Proposition 11 (Soundness of Algorithm 2)**.** *The following properties hold:*

1. *Algorithm 2 is sound for the* stable *and* complete *semantics*
2. *Algorithm 2 is sound for the* preferred *semantics*

☐ Proof of Proposition 11: link (See page 222).

`Note:` *Proposition 11 and afterward Proposition 13 are separated into two assertions because the proofs for the* preferred *semantics are different.*

From Propositions 10 and 11, we can prove that the entire algorithm is sound and complete for the *stable*, *preferred* and *complete* semantics.

Regarding Algorithm 1, two similar properties can be established:

Proposition 12 (Completeness of Algorithm 1)**.** *Algorithm 1 is complete for the* stable*,* complete *and* preferred *semantics.*

☐ Proof of Proposition 12: link (See page 223).

Proposition 13 (Soundness of Algorithm 1)**.** *The following properties hold:*

1. *Algorithm 1 is sound for the* stable *and* complete *semantics.*
2. *Algorithm 1 is sound for the* preferred *semantics.*

☐ Proof of Proposition 13: link (See page 224).

Chapter 5

AFDivider : Experimental analysis

5.1 Introduction

In this chapter we present experimental results conducted with three *AFDivider* variants presented in Section 5.2 on the next page.

`Note:` *A detailed technical documentation of the whole AFDivider project (user manual of the solver, software sources, project installation, experimental environment) can be found in [37].*

The experiments have been made on some hard instances of the ICCMA competition for the *preferred*, *stable* and *complete* semantics and for the *enumeration problem.*[1] The AF instances studied are of Barabási–Albert (BA), Erdős–Rényi (ER), Watts-Strogatz (WS), Traffic (TR), Block world (BW) and Ferry (F2) types. The three first types have been generated by *AFBenchGen2* (see [21]), the fourth type are AF generated from real traffic data (see [30]) and the last two types are block world and ferry planning problems transformed into AF problems (see [20]).

To compute the labellings of a cluster given a particular labelling of its inward attack sources, we have used an already existing solver called "μ-*toksia*", the winner of all enumeration problem tracks of the ICCMA 2019 session, which transforms the AF labelling problem into a SAT problem [59]. In these experiments, we compare our algorithms (using μ-*toksia*) with μ-*toksia* itself, and with several other solvers presented in ICCMA 2017 and 2019. In the following, solver names are suffixed by the year of the ICCMA session for which they participated. For each experiment, we used 6 cores of an Intel Xeon Gold 6136 processor, each core having a frequency of 3 GHz. The RAM size was 45GB. The timeout was set to 1 hour for the real time. Analysis have been conducted on a number of solvers. The ones which are presented here are those which succeed to solve at least one of these selected hard instances.[2]

After the presentation of the three *AFDivider* variants (Section 5.2 on the next page), we analyse our result experiments (Section 5.3 on page 55), by focusing, in a first step, on the success rate of the solvers, in a second step, on their solving time, and in a third step, by comparing the clustering time with the resolution time of our *AFDivider* variants.

[1]In ICCMA Competition the enumeration problems for the *preferred*, *stable* and *complete* semantics are named respectively "*EE-PR*", "*EE-ST*" and "*EE-CO*"

[2]In addition to the solvers presented in Table 5.1 on page 56, *Eqargsolver2019*, *Taas-dredd2019* and *Yonas2019* have also been analysed.

5.2 Clustering methods

Among the various clustering methods, three of them, particularly well suited for the kind of graphs that we want to address (see Chapter 5 on the previous page), are presented here. For each clustering method associated with the *AFDivider* algorithm, an implementation has been made.

5.2.1 Spectral clustering

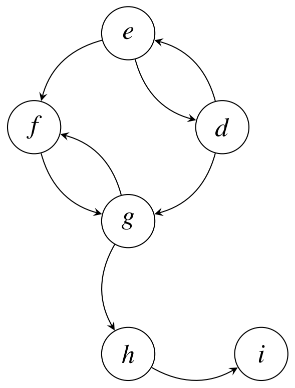
The first clustering method that we used for the *AFDivider* algorithm is the so-called "***spectral clustering***". This method is presented and illustrated in details in the *Mathematical Background*, Part VI, Section 17.2 on page 188.

This clustering method, originally used for data mining, relies on a similarity matrix that represents how much a data in a dataset is similar to another one. To adapt it for AF clustering, we defined the similarity between two arguments *a* and *b* as the number of attacks between them. Thus, this number may be 0, 1 or 2. Formally we defined a transformation that produces a non-directed graph from an AF.

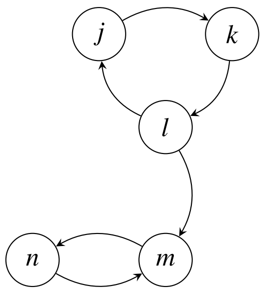
Definition 39. *(Undirection transformation) Let $\mathcal{AF} = \langle A, K \rangle$ be an AF. The non-directed graph $G = (V, E, W)$ obtained by the undirection transformation of $\mathcal{AF}$ (noted $\mathcal{U}(\mathcal{AF})$) is defined as following:*

- $V = A$
- $E = \{(a,b) | (a,b) \in K \text{ or } (b,a) \in K\}$
- $W : E \to \{0,1,2\}$ *is defined as following:*

$$W : (a,b) \mapsto \begin{cases} 0 \text{ if } (a,b) \notin K \text{ and } (b,a) \notin K, \\ 1 \text{ if } ((a,b) \in K \text{ and } (b,a) \notin K) \text{ or } ((a,b) \notin K \text{ and } (b,a) \in K), \\ 2 \text{ otherwise} \end{cases}$$



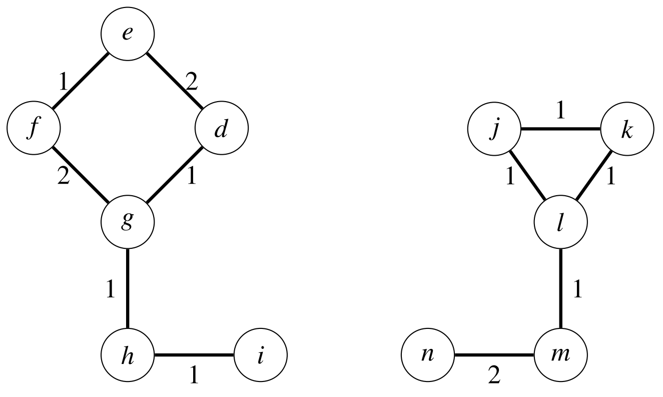
(a) Component 1: af_1



(b) Component 2: af_2

Figure 5.1: The connected components of $\mathcal{AF}_{hard}$

Example 24. *Let consider the running example used in the previous section. Figure 5.1 on the previous page recalls the two components obtained after the removing of the trivial part of the original AF. Figure 5.2 shows the non-directed graphs obtained by the undirection transformation of af_1 and af_2. These non-*



(a) Component 1: af_1 (b) Component 2: af_2

Figure 5.2: $\mathscr{U}(af_1)$ and $\mathscr{U}(af_2)$

.

directed weighted graphs can be represented by their adjacency matrices defined as:

$$M_a^{af_1} = \begin{array}{c|cccccc|} & d & e & f & g & h & i \\ d & \mathbf{0} & 2 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} \\ e & 2 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ f & \mathbf{0} & 1 & \mathbf{0} & 2 & \mathbf{0} & \mathbf{0} \\ g & 1 & \mathbf{0} & 2 & \mathbf{0} & 1 & \mathbf{0} \\ h & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} & 1 \\ i & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} \end{array} \qquad M_a^{af_2} = \begin{array}{c|ccccc|} & j & k & l & m & n \\ j & \mathbf{0} & 1 & 1 & \mathbf{0} & \mathbf{0} \\ k & 1 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} \\ l & 1 & 1 & \mathbf{0} & 1 & \mathbf{0} \\ m & \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} & 2 \\ n & \mathbf{0} & \mathbf{0} & \mathbf{0} & 2 & \mathbf{0} \end{array}$$

Given an AF, the AF spectral clustering relies thus on a kind of adjacency matrix where the directionality of edges is omitted and where the matrix values are the number of edges between two arguments. Basically, the more an argument will be related to another, the more similar the two arguments will be considered. This similarity criterion is particularly relevant for non-dense graphs with a clustered structure. Indeed, it produces sparse matrices and as a consequence the eigenvector equation system to solve is simplified as there are many zero values. This is what motivated our choice for the spectral clustering method.

Example 25. *Following Example 24, the spectral clustering produces the partitions represented in Figure 5.3 on page 62.*

`Note:` *For a step by step illustration of how we end up with this result, see Section 17.2 on page 188.*

Notice that when using the spectral clustering method for the *preferred* semantics computation, the optimization described in the first note of Page 47 is used for the maximality check (made in Algorithm 2, line 8).

5.2.2 USCC based clusterings

While the idea of the previous clustering was to find groups of arguments highly related in terms of attack relations, gathering them together regardless of their membership to common SCC structures, the two others clustering studied have been proposed to ensure that SCCs are not split into different clusters.

Given an AF, the so-called "***USCC Chain***" clustering forms clusters as following (each cluster being a $USCC_{af}$):

1. First, the set of SCC is computed.
2. Then neighbour SCC singletons are joined together as chains. The first element of such a chain is thus a singleton that is not attacked by a singleton.
3. If a singleton belongs to several chains, it is kept only by the chain that has the least inward attacks (attacks coming from arguments that are not in the chain). Note that the intersection of the so built chains is empty. Ties are broken arbitrarily.
4. The last step is to join SCC and chains 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 until there is no group of less than a certain number of arguments. We experimentally choose to fix this threshold to 10.

The third clustering studied is the so-called "***USCC Tree***" clustering. It has several common steps with the *USCC Chain* method, but it differs on one point: instead of forming chains that do not intersect, chains that have common arguments are merged together. This process thus produces trees.

Example 26. *Both USCC based clustering methods give the same result on the previous AF example. In order to highlight the differences between the two, let consider $\mathcal{AF} = \langle A, K \rangle$, the AF shown in Figure 5.4 on page 62. For the sake of brevity, the AF chosen is small. Therefore we will not illustrate the fourth step (otherwise the AF would have to be to big).*
The two first steps are the same for both methods:

- ***Step 1:*** *the SCC_{af} of $\mathcal{AF}$ are computed. They are graphically represented in Figure 5.5 on page 63.*
- ***Step 2:*** *In the second step the singletons are joined together as shown in Figure 5.6 on page 63.*

*At **Step 3**, they will differ:*

- *The* USCC Chain *method will produce the partition:* $\{\{a,b\},\{c,d\},\{e,f\},\{g,h,i\},\{j,k\}\}$
- *The* USCC Tree *method will produce the partition:* $\{\{a,b\},\{c,d,e,f\},\{g,h,i\},\{j,k\}\}$

Notice that when using these two clustering methods for the *preferred* semantics computation, we skip the maximality check made in Algorithm 2, line 8, as explained in the second note of Page 47.

5.3 Result presentation

5.3.1 Success Count Comparison

Table 5.1 on the next page shows the number of successes for each solver, by AF type selection. For each selection (columns) there are three values in each cell corresponding respectively to the *preferred*, the *stable* and the *complete* semantics. The following list provides references for the different solvers analysed:

- For *AFDiv-spectral*, see Section 5.2.1
- For *AFDiv-USCC-Chain*, see Section 5.2.2
- For *AFDiv-USCC-Tree*, see Section 5.2.2
- For *ArgSemSAT2017*, see [22]
- For *Argmat-dvisat2017*, see [60]
- For *Argmat-sat2017*, see [61]
- For *Aspartix2019*, see [44]
- For *Cegartix2017*, see [42]
- For *Coquiaas2019*, see [50]
- For *μ-toksia2019*, see [59]
- For *Pyglaf2017*, see [2]
- For *Pyglaf2019*, see [3]

`Note:` The P-SCC-REC *algorithm presented in Section 7.2.3 on page 79 has not been included in this study because no solver were available, as best as we know.*

Among the first things we can observe is that the AF type, has a great impact on the aptitude of a given solver to enumerate the labellings under a given semantics. As an example, for the *preferred* semantics, *Argmat-dvisat2017* solves nine BA instances (which is the best score among other solvers than ours) but only three WS instances. Likewise, we can also observe that for a given solver and for a given AF type, the aptitude to succeed depends on the semantics. As an example, for the ER type, *Pyglaf2017* succeeds in solving six instances for the *preferred* semantics but four for the *stable* one. Such a table of experimental results is then a good tool to identify solvers specificities and capabilities.

Another observation is that the *complete* semantics is much harder than the two other ones. Although several decision problems under the *preferred* and the *stable* semantics are harder than under the *complete* one, in practice the first stumbling block to enumerate the labellings of a given semantics is (for most of the studied AF instances) the number of labellings it produces.

Let focus on our solvers results. One interesting fact is that the clustering criterion used by the *AFDivider* algorithm has an impact on the AF types for which it will be well suited. Indeed there are two different behaviour classes: one for the USCC clustering variants and another one for the spectral clustering one. Both behaviour classes are good on BA type and bad for BW and F2 types (whatever the semantics) but we can see that the USCC variants are good on ER and WS types which is not the case for the spectral one while the latter is better than them on TR type. These results were expected. Indeed, ER and WS type instances do not have a structure with groups of arguments such that the intra density (within group) is greater than the one inter groups (outside group). That is precisely what spectral clustering is seeking for. In contrast, some TR instances have a structure which is much less adequate to USCC variants. Given that for each cut attack three cases have to be consider when computing a cluster labellings, clusterings as the three proposed ones are not suited for BW and F2 instance types. Indeed these instances are translation of planning problems in which the notion of sequential constraints (over time or resources) is very present. This leads to AFs with particular shape for which a sequential reasoning (even though multi-threaded) is better. Nevertheless, it is worth noting that spectral clustering gives better result than USCC ones on BW type. Unlike planning

	AF type selections																										
	BA			ER			TR			WS			BW			F2			All types			BA to WS			BA-ER-WS		
Nb of instances	15	15	15	15	15	15	15	15	15	15	15	15	9	9	9	6	6	6	75	75	75	60	60	60	45	45	45
AFDiv-spectral	**10**	10	**2**	0	2	0	4	5	0	0	0	0	3	5	0	0	0	0	17	22	2	14	17	2	10	12	2
AFDiv-USCC-Chain	**10**	10	**2**	**6**	6	5	1	1	0	4	**5**	4	0	0	0	0	0	0	21	22	11	21	22	11	**20**	21	11
AFDiv-USCC-Tree	**10**	10	**2**	5	6	**6**	1	1	0	4	**5**	4	0	0	0	0	0	0	20	22	12	20	22	**12**	19	21	**12**
ArgSemSAT2017	0	2	0	**6**	3	1	0	4	0	5	2	0	0	4	0	0	5	0	11	20	1	11	11	1	11	7	1
Argmat-dvisat2017	9	**11**	**2**	2	3	3	4	5	1	3	**5**	4	**9**	**9**	**1**	**5**	5	**5**	32	38	**16**	18	24	10	14	19	9
Argmat-sat2017	2	3	0	2	3	3	2	5	0	3	**5**	4	5	8	0	**5**	5	0	19	29	7	9	16	7	7	11	7
Aspartix2019	6	**11**	**2**	**6**	**8**	**6**	3	8	0	**5**	**5**	4	8	**9**	0	**5**	**6**	3	33	**47**	15	20	**32**	**12**	17	**24**	**12**
Cegartix2017	6	3	0	2	2	2	3	5	0	3	4	**5**	8	8	0	**5**	5	0	27	27	7	14	14	7	11	9	7
Coquiaas2019	4	2	0	1	4	2	3	4	0	0	**5**	4	8	7	0	**5**	5	0	21	27	6	8	15	6	5	11	6
μ-toksia2019	6	3	0	5	6	**6**	3	5	0	4	**5**	4	8	8	0	**5**	5	0	31	32	10	18	19	10	15	14	10
Pyglaf2017	6	**11**	**2**	**6**	4	2	**9**	**9**	**2**	3	**5**	4	8	**9**	**1**	**5**	**6**	**5**	**37**	44	**16**	**24**	29	10	15	20	8
Pyglaf2019	6	8	**2**	**6**	6	**6**	4	6	0	3	**5**	4	8	**9**	0	**5**	5	0	32	39	12	19	25	**12**	15	19	**12**

Table 5.1: Success count for *preferred* - *stable* - *complete* semantics (best values in bold and large font).

AF, the BA type is completely adequate to such a cutting process. Although not presented in the following tables, other clustering methods (fully random partition, among them) have been studied for the *AFDivider* algorithm. All of them give pretty good results on the BA type, even better than most of the studied solvers. This shows that a clustering approach (even with a random clustering) can give interesting results.

We can see that our USCC solvers are among the best considering BA, ER and WS types (last column). The success rate over all types and for all semantics is of 43.11% for *Pyglaf2017* (best rate), 42.22% for *Aspartix2019* (second rate) and of 24% for both of the USCC variants. When considering only the BA, ER and WS types, we have a rate of 39.23% for *Aspartix2019* (best rate), 38.52% for both of the USCC variants (second ones), 34.07% for *Pyglaf2019* (third one) and 31.85% for *Pyglaf2017* (forth rate).

It is also interesting to consider the reasons why some experiments failed. We observe that, for solvers other than the *AFDivider* variants, about 2% of the failures are due to memory overflow while 98% are due to timeout. For the *AFDivider* variants, about 56% are due to memory overflow and 44% are due to timeout. Knowing that in most cases, it is not because of time limitation that the *AFDivider* fail, a better memory management could increase its already good performances as highlighted in Section 5.3.5 on page 59.

5.3.2 Resolution Time Comparison

Let now consider the resolution time of those solvers. Given that the different solvers do not succeed for the same instances, making an overall average time could be misleading. Instead, in Table 5.2 on the following page, we compare the solvers two by two on instances solved by both of them (that is, same couple instance/semantics succeeded). Because of that, the average time is computed regardless of the AF type and of the semantics. To illustrate how to read this table, let consider the last cell of the first column. Over all their common instances successes, in average *Pyglaf2019* takes 552 seconds more than *AFDiv-spectral*.

While the *AFDiv-spectral* does not stand out on the previous analysis, this table shows that it is faster than all other solvers (except ours). On the instances which have been successfully solved by the USCC variants and *AFDiv-spectral* (which are mainly of BA type), the USCC ones are faster. We can observe that no solvers are better than all others, only 3 of 12 are better than 8 other ones: *AFDiv-spectral*, *AFDiv-USCC-Chain* and *Aspartix2019*. The three *AFDivider* variants give good results compared to other solvers.

	AFDiv-spectral	*AFDiv-USCC-Chain*	*AFDiv-USCC-Tree*	*ArgSemSAT2017*	*Argmat-dvisat2017*	*Argmat-sat2017*	*Aspartix2019*	*Cegartix2017*	*Coquiaas2019*	*μ-toksia2019*	*Pyglaf2017*	*Pyglaf2019*
AFDiv-spectral	0	45	36	**-192**	**-30**	**-252**	**-145**	**-368**	**-547**	**-212**	**-191**	**-552**
AFDiv-USCC-Chain	**-45**	0	**-87**	132	**-68**	2	279	**-736**	**-495**	**-159**	**-101**	**-311**
AFDiv-USCC-Tree	**-36**	87	0	144	**-64**	29	278	**-708**	**-451**	**-15**	**-64**	**-232**
ArgSemSAT2017	192	**-132**	**-144**	0	306	275	402	**-185**	169	118	153	187
Argmat-dvisat2017	30	68	64	**-306**	0	**-266**	8	**-405**	**-477**	**-133**	**-54**	**-329**
Argmat-sat2017	252	**-2**	**-29**	**-275**	266	0	448	**-91**	**-208**	188	151	176
Aspartix2019	145	**-279**	**-278**	**-402**	**-8**	**-448**	0	**-430**	**-562**	**-328**	**-52**	**-423**
Cegartix2017	368	736	708	185	405	91	430	0	**-177**	311	242	270
Coquiaas2019	547	495	451	**-169**	477	208	562	177	0	402	291	350
μ-toksia2019	212	159	15	**-118**	133	**-188**	328	**-311**	**-402**	0	**-38**	**-23**
Pyglaf2017	191	101	64	**-153**	54	**-151**	52	**-242**	**-291**	38	0	**-104**
Pyglaf2019	552	311	232	**-187**	329	**-176**	423	**-270**	**-350**	23	104	0

Table 5.2: Average real time difference between solvers on same set of solved instances (in seconds)
The values in bold and large font correspond to the case where the solver given in the line is faster than the solver given in the column.

Given that several studied solvers are multi-threaded, a similar analysis has been made considering the CPU time. Even though the values may slightly differ from this table, the comparison between the solvers stays the same. As there are few changes, this table is not relevant.

5.3.3 Clustering Impact Comparison

Let now consider the impact of the clustering against the resolution time of our *AFDivider* variants (Table 5.3 on the next page).

First of all, we can observe that the clustering is very time-efficient compare to other steps of the algorithm.

As expected, the *USCC Tree* clustering is faster than the *USCC Chain* one. Indeed merging chains is simpler than wisely separating chains sharing common arguments. We can also observe that the spectral clustering is more time consuming than the USCC ones.

	AFDiv-spectral	*AFDiv-USCC-Chain*	*AFDiv-USCC-Tree*
Clustering and cutting time	0.37	0.16	0.15
Cluster and component labelling time	57.09	31.10	30.52
Cartesian product time	40.63	40.66	40.65
Printing time	24.88	24.46	25.89
Total resolution time	122.97	96.39	97.20

Table 5.3: Average real time comparison (in seconds) of AFDivider variants over 26 instances.

AFDiv-spectral takes more time to compute the labellings than the USCC variants, which means that on those common solved instances the USCC partitions are better chosen.

As Table 5.2 on the previous page shows, on common instances *AFDiv-spectral* is slower than the two other ones. Nevertheless, it is worth noting, as it has been said previously, that they form two distinct behaviour classes of solvers, that do not share so many successes. The common ones are mainly of BA type.

Table 5.3 shows another important fact. Most of the resolution time, on those instances at least, comes from the cartesian product and the result printing. Indeed, some instances may admit millions and even more of labellings which take much time to print. Even without taking the printing time into account, we can see that the cartesian product alone takes between 40% (in the spectral case) and 55% (in the other cases) of the effective calculation time. The "*Compact Enumeration Representation*" introduced in Chapter 6 on page 64 is a proposal to address this issue.

5.3.4 Maximality Check Impact Comparison

When computing the *preferred* semantics with the *AFDivider* algorithm, a maximality check of the component labellings is needed. In Page 47 two optimizations for this check have been described:

1. Checking only profiles with `und`-labelled arguments
2. Skipping the maximality check when using USCC clustring methods

Let now consider the impact of those optimizations.

5.3.4.1 Checking only profiles with `und`-labelled arguments

To show its interest, we compare the labelling of components with and without the optimization, the components being clustered following the very same partition ensuring so a fair comparison.

Table 5.4 on the following page sums up the labelling step details of each experiment. We choose the first component from a TR AF type and the second one from a BW AF type. All durations are given in second. The percentages in brackets represent the duration rate of each step compared with the total component labelling time.

Without the optimization, each computed component labelling has to be checked. This step thus has a complexity of $\Theta(n^2)$, n being the number of computed component labellings. As shown by Table 5.4, it can be very consuming. The first component produces 3408 labellings while the second one 76366.

We can observe that with the optimization the maximality check is 100 times faster for the first component (2.251s against 0.021s), while the total component labelling is 1.5 times faster (6.631s against 4.370s).

	Component 1		Component 2	
Number of clusters	4		2	
Number of component labellings	3408		76366	
With Optimization	no	yes	no	yes
Cutting time	0.637 (9.60%)	0.934 (21.37%)	0.7 (0.03%)	0.685 (0.26%)
Clustering time	0.002 (0.03%)	0.007 (0.16%)	0.008 (0.00%)	0.010 (0.00%)
Labelling time	2.869 (43.27%)	2.725 (62.36%)	87.785 (3.94%)	88.784 (33.31%)
CSP solving time	0.872 (13.15%)	0.683 (15.63%)	178.819 (8.02%)	177.046 (66.43%)
Maximality check time	2.251 (33.95%)	0.021 (0.48%)	1961.143 (88.01%)	8×10^{-6} (0.00%)
Total Component labelling time	6.631 (100%)	4.370 (100%)	2228.455 (100%)	266.525 (100%)
Component labelling reunification time (CSP solving + Maximality check)	3.123 (47.10%)	0.704 (16.11%)	2139.962 (96.03%)	177.046 (66.43%)

Table 5.4: Component labelling time details (in second)

For the second component, with the optimization the maximality check is 2.5×10^8 times faster (1961.143s against 8×10^{-6}s) while the total component labelling is 8.4 times faster (2228.455s against 266.525s).

Notice that this optimization may produce an empty set of labellings to check. The value 2.5×10^8 for the maximality check of second component indicated that the set to check is empty.

5.3.4.2 Skipping the maximality check when using USCC clustering methods

In this experiment, we compared the use of the optimization analysed above against no maximality checking while using an USCC clustering.

None of the experiments made produced profiles with *und*-labelled arguments. As a consequence, in both cases the component labelling process was almost identical. The impact of skipping the maximality check was thus negligible, on those instances.

However, if we want to compare the use of this optimization against the use of no optimization at all (that is, the use of the naive maximality check in $\Theta(n^2)$) we can refer to the previous section to have an idea of the benefit we can get.

5.3.5 Memory Overflow Analysis

Table 5.5 on page 61 shows the details of the memory overflows that occurred during our experiments. Following AF types, it shows which percentage of them happened at each step of the algorithm. As an example on how to read the table, let analyse the memory overflows made by *AFDiv-spectral* on the TR instances: 12 of the 45 TR instances ended up with an overflow, 41.67% of them (that is, 41.67% of the 12) occurred during the clustering and cutting step, 8.33% during the cluster and component labelling step, 33.33% during the cartesian product and 16.67% during the printing step (which gives a total of 100%).

`Note:` *The values "NaN" are used in the table when no instance of the corresponding AF type has produced a memory overflow.*

We can observe that, BA instances put aside, most of the memory overflow occurs during the clustering and cutting phase and secondarily during the cartesian product phase. For BA instances, it is during the cartesian product phase and secondarily the labelling phase.

Further investigations need to be made to understand the memory overflows that occur during the clustering and cutting phase, especially for the USCC clusterings. Indeed this could be due to implementation flaws.

If it is only an implementation flaw supposition in the clustering and cutting phase, it is a certainty for the printing phase. If stored properly, it must be possible to print all the computed labellings without using much space (as an example with the help of buffers, iterators, etc).

The memory overflows that occurred during the cartesian product and the printing phases can be avoided by using the "*Compact Enumeration Representation*" introduced in Chapter 6 on page 64.

5.4 Synthesis

As a synthesis, in our experimental analysis, we instantiated the generic distributed and clustering based algorithm *AFDivider*, that enumerates the *complete*, *stable* and *preferred* semantics labellings, with three different clustering methods. We compare their performances, to other solvers, according to AF types, over success rate and resolution time. It comes out from this study that the *AFDiv-spectral* variant is faster than most solvers (except ours) in average on common successful instances while the *AFDiv-USCC-Tree* and *AFDiv-USCC-Chain* variants succeed to solve most instances when considering Barabási–Albert (BA), Erdős–Rényi (ER) and Watts-Strogatz (WS) AF types, for the *complete*, *stable* and *preferred* semantics.

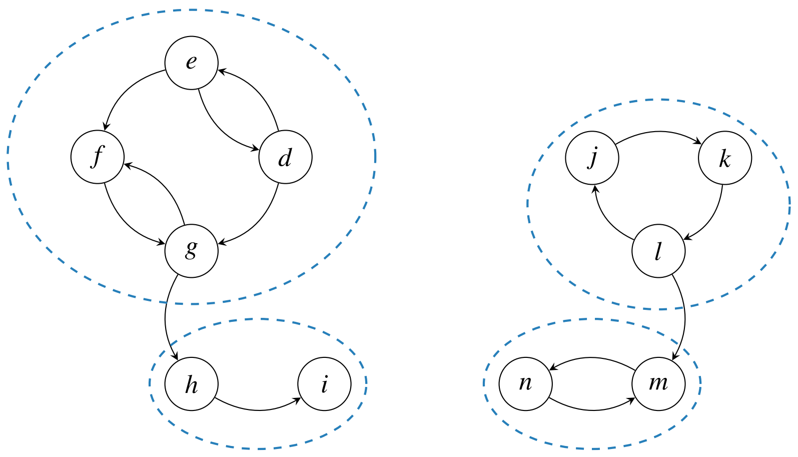
Based on the AF types that they solve efficiently, we identify two behaviour classes among our three solvers: one for the spectral clustering and one for the USCC based clusterings. This shows that the clustering method which is used has an important effect on the performances of the *AFDivider* algorithm on a particular AF type.

A major improvement of the *AFDivider* algorithm would then be to "know" in advance which clustering method (including ones other than those presented in Section 5.2 on page 52) should be used for a particular AF instance. Experiments could be conducted to learn, for example with a neural network, which one to use. To go further, even the cutting process could be supervised by a neural network trained to cluster AF instances following their structure. As a consequence, for any known AF type, the most appropriate clustering method would be used to solve each AF instance efficiently.

The experiments also show that a better memory management and a more compact labelling representation could enhance the efficiency of the algorithm. This is what we tried to solve introducing the notion of "*compact enumeration representation*" presented in Chapter 6 on page 64.

		AFDiv-spectral	*AFDiv-USCC-Chain*	*AFDiv-USCC-Tree*
BA	Total number of overflows	21 over 45	20 over 45	20 over 45
	Clustering and cutting	0%	0%	0%
	Cluster and component labelling	+19.05%	+10%	+10%
	Cartesian product	+80.95%	+85%	+85%
	Printing	+0%	+5%	+5%
ER	Total number of overflows	31 over 45	0 over 45	0 over 45
	Clustering and cutting	100%	NaN	NaN
	Cluster and component labelling	+0%	NaN	NaN
	Cartesian product	+0%	NaN	NaN
	Printing	+0%	NaN	NaN
TR	Total number of overflows	12 over 45	26 over 45	26 over 45
	Clustering and cutting	41.67%	80.77%	80.77%
	Cluster and component labelling	+8.33%	+7.69%	+7.69%
	Cartesian product	+33.33%	+3.85%	+3.85%
	Printing	+16.67%	+7.69%	+7.69%
WS	Total number of overflows	39 over 45	0 over 45	0 over 45
	Clustering and cutting	100%	NaN	NaN
	Cluster and component labelling	+0%	NaN	NaN
	Cartesian product	+0%	NaN	NaN
	Printing	+0%	NaN	NaN
BW	Total number of overflows	12 over 27	27 over 27	27 over 27
	Clustering and cutting	75%	100%	100%
	Cluster and component labelling	+16.67%	+0%	+0%
	Cartesian product	+8.33%	+0%	+0%
	Printing	+0%	+0%	+0%
F2	Total number of overflows	2 over 18	17 over 18	17 over 18
	Clustering and cutting	100%	100%	100%
	Cluster and component labelling	+0%	+0%	+0%
	Cartesian product	+0%	+0%	+0%
	Printing	+0%	+0%	+0%
All	Total number of overflows	117 over 225	90 over 225	90 over 225
	Clustering and cutting	73.50%	72.22%	72.22%
	Cluster and component labelling	+5.99%	+4.45%	+4.45%
	Cartesian product	+18.8%	+20.0%	+20.0%
	Printing	+1.71%	+3.33%	+3.33%

Table 5.5: Memory overflow analysis: rate of instances passing algorithm steps



(a) Clusters of af_1 (b) Clusters of af_2

Figure 5.3: Cluster partition

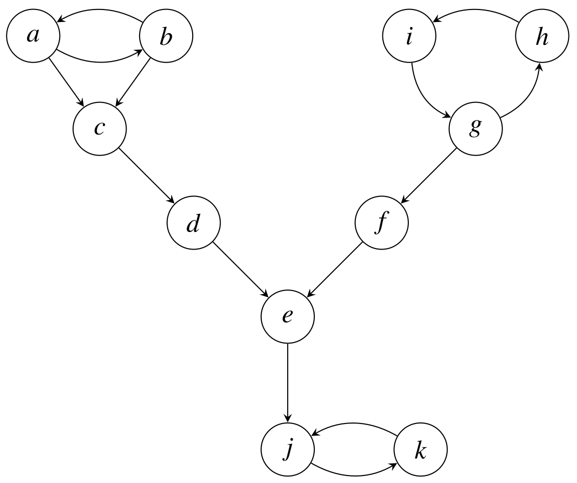


Figure 5.4: AF example

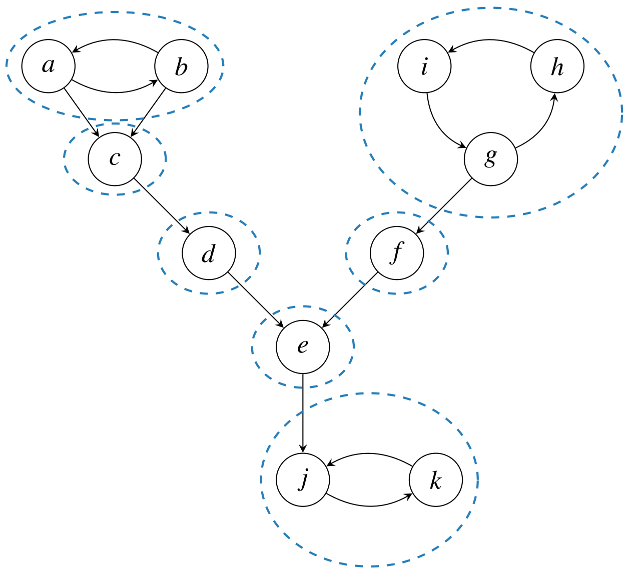


Figure 5.5: AF example

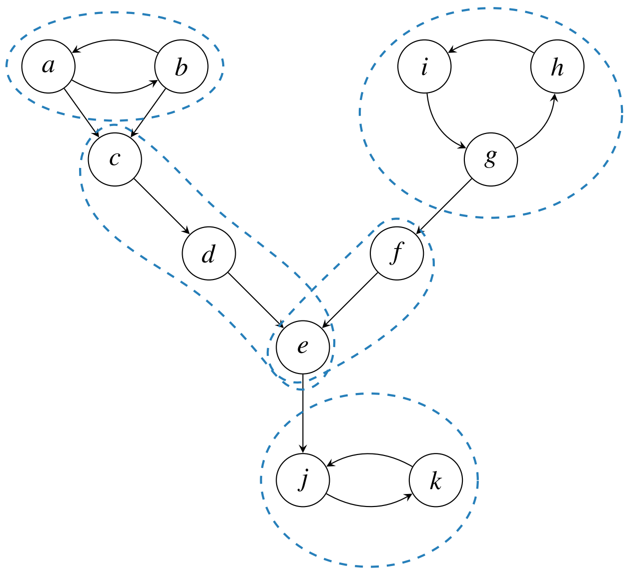


Figure 5.6: AF example

Chapter 6

AFDivider : Compact representation

6.1 Motivation and Definition

The last step of the *AFDivider* algorithm (Algorithm 1 on page 30, line 7) is a huge cartesian product between the fixed labelling part and all the component labellings. Experimental analysis shows that in some cases this enumeration construction is very time consuming. Furthermore, we realise that in order to answer common AF decision problems, this enumeration could not be necessary. These observations make us think about a "compact enumeration representation". Notice that the aim of this new algorithm is no more to enumerate the labellings/extensions under a given semantics but rather to provide a data structure from which all AF classical decision problems, and even more, can be answered. In few words, the compact enumeration representation is the set of the component labelling sets (and the fixed part).

Definition 40 (Compact Enumeration Representation)**.** *Let σ be a semantics. Let $\mathcal{AF} = \langle A, K\rangle$ be an AF and $\Omega = \{\omega_0, ..., \omega_n\}$ be a partition of A such that ω_0 is the trivial part of $\mathcal{AF}$ (i.e. $\omega_0 = A \cap (\mathtt{in}(\ell_{gr}) \cup \mathtt{out}(\ell_{gr})))$ and $\omega_1, ..., \omega_n$ correspond to the sets of arguments of the connected components of $\mathcal{AF}_{hard}$, following the computation made by Algorithm 1. Let $\{af_0, ..., af_n\}$ be the set of sub-AF corresponding to Ω. The compact enumeration representation $\mathtt{Comp}_\sigma(\mathcal{AF})$ is the set: $\{\mathscr{L}_\sigma(af_0), ..., \mathscr{L}_\sigma(af_n)\}$.*

Example 27. *Let consider one more time our running example. Let $\mathcal{AF} = \langle A, K\rangle$ be the AF shown in Figure 6.1 on the following page. The compact enumeration representation corresponding to the* complete *semantics is the set $\{A, B, C\}$ with:*

$$A = \mathscr{L}_\sigma(af_0) = \{\{(a, \mathtt{in}), (b, \mathtt{out}), (c, \mathtt{out})\}\}$$

$$B = \mathscr{L}_\sigma(af_1) = \left\{ \begin{array}{c} \{(e, \mathtt{in}), (f, \mathtt{out}), (g, \mathtt{in}), (h, \mathtt{out})\} \\ \{(e, \mathtt{out}), (f, \mathtt{in}), (g, \mathtt{out}), (h, \mathtt{in})\} \\ \{(e, \mathtt{und}), (f, \mathtt{und}), (g, \mathtt{und}), (h, \mathtt{und})\} \end{array} \right\}$$

$$C = \mathscr{L}_\sigma(af_2) = \left\{ \begin{array}{c} \{(j, \mathtt{und}), (k, \mathtt{und}), (l, \mathtt{und}), (m, \mathtt{out}), (n, \mathtt{in})\} \\ \{(j, \mathtt{und}), (k, \mathtt{und}), (l, \mathtt{und}), (m, \mathtt{und}), (n, \mathtt{und})\} \end{array} \right\}$$

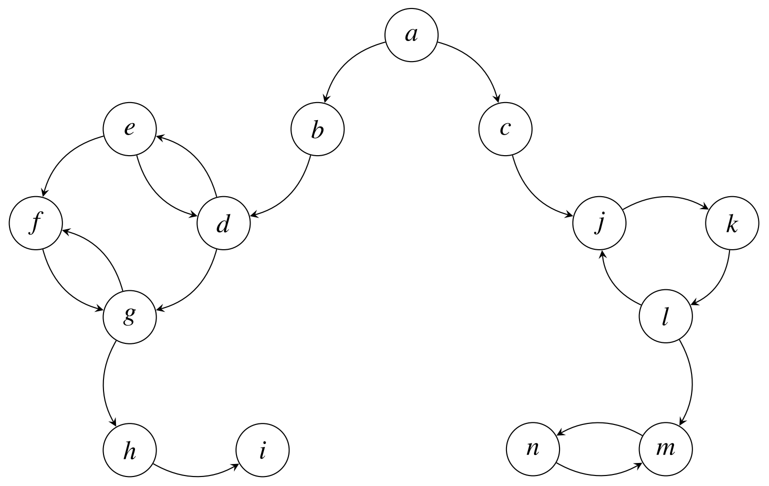


Figure 6.1: The AF of the running example

6.2 Decision Problems with Compact Enumeration

Given that any combination of component labelling parts produces a valid labelling of the computed semantics, all classical AF decision problems can be answered.

For the credulous (resp. skeptical) acceptance problem of an argument a, we just have to look if a is at least in one (resp. in any) labelling part in the component in which a is. For the verification problem of a labelling ℓ, we just have to look if there exists a combination of component labelling parts that produces ℓ.

For the existence problem, we just have to look if at least a combination is possible. In the case that there is no labelling the compact enumeration representation is the empty set. No product is thus possible.

For the non-empty existence problem, we just have to look if a combination produces a labelling having at least an $\mathtt{in}$-labelled argument.

Finally, for the uniqueness problem, we just have to verify that one and only one combination is possible, that is, each set of the enumeration compact representation has exactly one element.

Let formally define the AF decision problems using this representation and then show that they are equivalent to the classical ones.

Definition 41 (AF decision Problems with compact enumeration representation)**.**
Let $\mathcal{AF} = \langle A, K\rangle$ be an AF decomposed into n components and $\mathtt{Comp}_\sigma(\mathcal{AF}) = \{\mathscr{L}_\sigma(af_0), ..., \mathscr{L}_\sigma(af_n)\}$ be the compact enumeration representation corresponding to $\sigma(\mathcal{AF})$.

- Credulous Acceptance *Comp-Cred$_\sigma$: Given an AF $\mathcal{AF} = \langle A, K\rangle$ and an argument $a \in A$. Is it the case that: $\forall \mathscr{L}_\sigma(af) \in \mathtt{Comp}_\sigma(\mathcal{AF})$, $\mathscr{L}_\sigma(af) \neq \varnothing$ (i.e. Comp-Exists$_\sigma(\mathcal{AF})$ is true) and that there exists a set $\mathscr{L}_\sigma(af) \in \mathtt{Comp}_\sigma(\mathcal{AF})$ such that $\exists \ell \in \mathscr{L}_\sigma(af)$ and $\ell(a) = \mathtt{in}$?*

- Skeptical Acceptance *Comp-Skep$_\sigma$: Given an AF $\mathcal{AF} = \langle A, K\rangle$ and an argument $a \in A$. Is it the case that: $\forall \mathscr{L}_\sigma(af) \in \mathtt{Comp}_\sigma(\mathcal{AF})$, $\mathscr{L}_\sigma(af) \neq \varnothing$ (i.e. Comp-Exists$_\sigma(\mathcal{AF})$ is true) and that there exists a set $\mathscr{L}_\sigma(af) \in \mathtt{Comp}_\sigma(\mathcal{AF})$ such that $\forall \ell \in \mathscr{L}_\sigma(af)$, $\ell(a) = \mathtt{in}$?*

- Verification of an extension *Comp-Ver*$_\sigma$*: Given an AF* $\mathcal{AF} = \langle A, K\rangle$ *and a labelling* ℓ*. Is there a combination of component labellings* $\ell_1, ..., \ell_n$ *with* $\ell_i \in \mathscr{L}_\sigma(af_i)$ *and* $\mathscr{L}_\sigma(af_i) \in \texttt{Comp}_\sigma(\mathcal{AF})$ *such that:* $\ell = \bigcup_{i=1}^{n} \ell_i$ *?*

- Existence of an extension *Comp-Exists*$_\sigma$*: Given an AF* $\mathcal{AF} = \langle A, K\rangle$*. Is it the case that:* $\forall \mathscr{L}_\sigma(af) \in \texttt{Comp}_\sigma(\mathcal{AF})$*,* $\mathscr{L}_\sigma(af) \neq \varnothing$*?*

- Existence of a non-empty extension *Comp-Exists*$_\sigma^{\neg\varnothing}$*: Given an AF* $\mathcal{AF} = \langle A, K\rangle$*. Does there exist a combination of component labellings* $\ell_1, ..., \ell_n$*, with* $\ell_i \in \mathscr{L}_\sigma(af_i)$ *and* $\mathscr{L}_\sigma(af_i) \in \texttt{Comp}_\sigma(\mathcal{AF})$*, such that:* $\exists i \in \{1, ..., n\}$*,* $in(\ell_i) \neq \varnothing$*?*

- Uniqueness of a solution *Comp-Unique*$_\sigma$*: Given an AF* $\mathcal{AF} = \langle A, K\rangle$*. Is it the case that:* $\forall \mathscr{L}_\sigma(af_i) \in \texttt{Comp}_\sigma(\mathcal{AF})$*,* $|\mathscr{L}_\sigma(af_i)| = 1$*?*

Proposition 14. *Let* $\sigma \in \{\text{complete}, \text{stable}, \text{preferred}\}$ *be a semantics. Let* $\mathcal{AF} = \langle A, K\rangle$ *be any AF,* $a \in A$ *be an argument and* ℓ *be any labelling of* $\mathcal{AF}$*. We have the following equivalence (that is, in any case both decision problems give the same answer):*

1. *Cred*$_\sigma(\mathcal{AF}, a) \equiv$ *Comp-Cred*$_\sigma(\mathcal{AF}, a)$
2. *Skep*$_\sigma(\mathcal{AF}, a) \equiv$ *Comp-Skep*$_\sigma(\mathcal{AF}, a)$
3. *Ver*$_\sigma(\mathcal{AF}, \ell) \equiv$ *Comp-Ver*$_\sigma(\mathcal{AF}, \ell)$
4. *Exists*$_\sigma(\mathcal{AF}) \equiv$ *Comp-Exists*$_\sigma(\mathcal{AF})$
5. *Exists*$_\sigma^{\neg\varnothing}(\mathcal{AF}) \equiv$ *Comp-Exists*$_\sigma^{\neg\varnothing}(\mathcal{AF})$
6. *Unique*$_\sigma(\mathcal{AF}) \equiv$ *Comp-Unique*$_\sigma(\mathcal{AF})$

☐ Proof of Proposition 14: link (See page 226).

As stated by Proposition 14, the compact enumeration representation is sufficient to answer all classical decision problems. But more than that, it could be used for other type of problems such as gradual acceptability : "Given $a \in \mathcal{AF}$, what is the portion of extensions a belongs to?". This portion is the same portion as the one in its own component.

6.3 Implementation ideas

In order to enhance the time needed to answer the first three decision problems *Comp-Cred*$_\sigma$, *Comp-Skep*$_\sigma$ and *Comp-Ver*$_\sigma$, we can produce as output two dictionaries: one linking the component name to its labellings and the other one linking each argument to the name of the component they belong to.

Example 28. *For the running example, we would have :*

- *Dictionary 1 :*

$$\left\{\begin{array}{l} A : \{\{(a,\texttt{in}),(b,\texttt{out}),(c,\texttt{out})\}\} \\ B : \left\{\begin{array}{l} \{(e,\texttt{in}),(f,\texttt{out}),(g,\texttt{in}),(h,\texttt{out})\} \\ \{(e,\texttt{out}),(f,\texttt{in}),(g,\texttt{out}),(h,\texttt{in})\} \\ \{(e,\texttt{und}),(f,\texttt{und}),(g,\texttt{und}),(h,\texttt{und})\} \end{array}\right\} \\ C : \left\{\begin{array}{l} \{(j,\texttt{und}),(k,\texttt{und}),(l,\texttt{und}),(m,\texttt{out}),(n,\texttt{in})\} \\ \{(j,\texttt{und}),(k,\texttt{und}),(l,\texttt{und}),(m,\texttt{und}),(n,\texttt{und})\} \end{array}\right\} \end{array}\right\}$$

- *Dictionary 2 :* $\left\{\begin{array}{ccccccc} a:A, & b:A, & c:A, & d:B, & e:B, & f:B, & g:B, \\ h:B, & i:B, & j:C, & k:C, & l:C, & m:C, & n:C \end{array}\right\}$

Given that a labelling is also a dictionary, once this representation is computed, verifying the credulous or skeptical acceptance of a given argument can be made in linear time according to the number of labellings of the argument component.[1] For the verification problem it can also be made in linear time according to the total number of component labellings and arguments.

Example 29. *Let consider the answer process for these already given examples*

- *Is* $\ell = \left\{\begin{array}{c} (a,\texttt{in}),(b,\texttt{out}),(c,\texttt{out}),(e,\texttt{out}),(f,\texttt{in}),(g,\texttt{out}), \\ (h,\texttt{in}),(j,\texttt{und}),(k,\texttt{und}),(l,\texttt{und}),(m,\texttt{und}),(n,\texttt{und}) \end{array}\right\} \in \mathscr{L}_{co}(\mathcal{AF})$*?*

  Given that finding an element in a dictionary can be done in constant time[2], splitting ℓ following the components using Dictionnary 2 can be done in linear time according to the number of arguments. This split produces $\{(a,\texttt{in}),(b,\texttt{out}),(c,\texttt{out})\}$ for A, $\{(e,\texttt{out}),(f,\texttt{in}),(g,\texttt{out}),(h,\texttt{in})\}$ for B and $\{(j,\texttt{und}),(k,\texttt{und}),(l,\texttt{und}),(m,\texttt{und}),(n,\texttt{und})\}$ for C. Given that labellings are also dictionaries, checking if those labellings parts belong to their corresponding component can be done in linear time according to the number of the component labellings. For A, at most one check has to be done, three for B and two for C. As a consequence, six checks at most have to be performed, which is the total number of component labellings. Let v be the number of component labellings and w be the number of arguments of the AF, the complexity of checking if a labelling is produced by a given semantics is thus in $\Theta(w+v)$,[3] so in linear time according to the number of arguments and of component labellings.

- *For any labelling $\ell \in \mathscr{L}_{co}(\mathcal{AF})$, is $\ell(n) = \texttt{in}$?*

  Finding the component of the argument n using Dictionary 2 is done in constant time. Then checking if n is `in`*-labelled in any labelling of C is done in linear time according to the number of labellings of C, which is two.*

[1] This is due to the fact that finding an element in a dictionnary is done in constant time (*i.e.* $\Theta(1)$) in the average case as explained by the following footnote.

[2] Good implementations of dictionaries use some hash function to create the keys of the stored elements. If the hash function is well chosen *w.r.t.* the size of the dictionary itself and the definition domain of the stored elements, then the verification of the membership of an element to the dictionary can be done in average in constant time (*i.e.* $\Theta(1)$).

[3] See Section 16.5 on page 184 for an explanation on $\Theta(w+v)$.

6.4 Compact Enumeration Representation: Experimental Analysis

The experiments with the compact enumeration representation have been made using the very same setting as the ones presented in Chapter 5 on page 51. The only difference is that, instead of using *μ-toksia2019* [59], to compute the labellings of a cluster given a particular labelling of its inward attack sources, we used *Aspartix2019* [44], a solver which transforms the AF labelling problem into an ASP problem. Indeed, although *μ-toksia2019* is the winner of all enumeration problem tracks of the ICCMA 2019 session, our experiments show that *Aspartix2019* is better on the AF instances we selected (See Section 5.3 on page 55).

Now let consider Table 6.1 on the following page. Using this representation improves the algorithm performances for the types BA, TR and BW. We do not see effect on the other ones because (at least for the studied instances) they do not lead to multiple connected components. As a consequence, doing the enumeration is strictly equivalent to not doing it.

As one can observe, avoiding the enumeration allows to resolve in some cases more instances but above all it allows to compute much faster the semantics labellings: 97.03 times much faster for BA type, 2.46 for TR and 1.15 for BW.[4]

It is also worth noting that much less memory is used. A lot of failures were due to memory overflow (see Section 5.3.1 on page 55). Using the compact enumeration representation allows *AFDiv-spectral* to solve 26 instances more (24 of them were supposed to produce a memory overflow, the two other ones a timeout). For *AFDiv-USCC-Chain*, 22 new instances have been solved (21 of them were supposed to produce a memory overflow, the left one a timeout). For *AFDiv-USCC-Tree*, 22 new instances have been solved (20 of them were supposed to produce a memory overflow, the two other ones a timeout).

6.5 Synthesis

As a synthesis for this chapter, we show that the complete enumeration of all labellings is not necessary for solving classical AF problems. This can be done with a more compact data structure. This improvement has been shown interesting for the most difficult instances (especially for BA and TR, but also for BW) of the ICCMA benchmarks (see Table 6.1 on the following page). On these instances, the new version of the algorithm (without the complete enumeration) goes faster and solves more instances that the original one. The impact of the "no-enumeration" is particularly significant for the *complete* semantics due to the huge number of labellings that causes memory overflow during the enumeration.

[4]These comparisons do not take into account average solving time of the compact enumeration representation mode when standard enumeration mode gives no result.

		Enum.	*AFDiv-spectral*			*AFDiv-USCC-Chain*			*AFDiv-USCC-Tree*		
BA	Nb Suc.	yes	10	10	2	10	10	2	10	10	2
		no	14	14	13	14	14	13	14	14	13
	Time avg.	yes	59.34	59.62	167.80	61.84	61.66	169.74	60.25	59.57	167.63
		no	4.82	0.80	0.56	0.49	0.52	0.40	0.50	0.50	0.35
ER	Nb Suc.	yes	0	2	0	6	6	6	5	6	4
		no	0	2	0	6	6	6	5	6	4
	Time avg.	yes	NaN	2133.79	NaN	1737.72	1311.29	1917.24	1695.76	1471.12	1919.99
		no	NaN	2133.79	NaN	1737.72	1311.29	1917.24	1695.76	1471.12	1919.99
TR	Nb Suc.	yes	4	5	0	1	1	0	1	1	0
		no	5	6	4	2	2	1	2	2	1
	Time avg.	yes	154.89	76.64	NaN	288.41	209.99	NaN	244.01	209.89	NaN
		no	101.47	45.82	21.69	106.40	71.57	0.64	83.66	70.91	0.68
WS	Nb Suc.	yes	0	0	0	4	5	4	4	5	4
		no	0	0	0	4	5	4	4	5	4
	Time avg.	yes	NaN	NaN	NaN	1304.46	849.96	1385.25	1524.22	875.35	1473.26
		no	NaN	NaN	NaN	1304.47	849.96	1385.25	1524.22	875.35	1473.26
BW	Nb Suc.	yes	4	5	0	0	0	0	0	0	0
		no	4	5	1	0	0	0	0	0	0
	Time avg.	yes	72.90	90.41	NaN	NaN	NaN	NaN	NaN	NaN	NaN
		no	69.21	72.77	2867.53	NaN	NaN	NaN	NaN	NaN	NaN

Table 6.1: *AFDivider* solvers success with and without enumeration for *preferred* - *stable* - *complete* semantics.

Chapter 7

Related Work: Algorithms for AF

In this chapter are presented the related works associated to our algorithm proposal. We emphasize the comparison with direct approach based algorithms, as *AFDivider* is itself such an algorithm.

7.1 Indirect approach based algorithms

In this section, we briefly present some indirect approaches to solve AF problems, with an emphasis on SAT based algorithms, as it is the most common transformation used to solve argumentation problems.

7.1.1 SAT based algorithms

Logic and Abstract Argumentation are interrelated since the seminal work of Dung (see [13] for an overview about these links). Moreover, given the experience and hindsight we have today, SAT solvers are really efficient (See [51] as an example). Many problem considered as intractable are today, at least for some instances, within reach and that through the use of SAT solvers. There has been so much progress in the field that, when considering a computational problem, it is important to ask whether or not a transformation of our problem into a SAT one is advantageous or not. Indeed, this option could save us development costs and possibly, for better result in the end.

Following this observation, many argumentation solvers actually transform argumentation problems into SAT problems (See [38, 68, 23, 3] as examples). Given that they are as many approaches as there are SAT based argumentation solvers and AF semantics, we will simply recall some encoding examples for the labelling-based *complete* semantics, that have been presented in [23].

Given that SAT solvers take as input a CNF formula,[1] the encoding must be of that form. To do so, let introduce some notions. Let $\mathcal{AF} = \langle A, K \rangle$ be an AF such that $|A| = k$ and $\phi : \{1, ..., k\} \rightarrow A$ be an indexing of A.

- $\phi(i)$ is thus the argument i of $\mathcal{AF}$. We denote by $\phi(i)^-$ the set of attackers of the argument i.
- For each argument i, three boolean variables are defined: I_i, O_i, U_i. For I_i to be `true` (resp. O_i, U_i), it means that the argument i is labelled *in* (resp. *out*, *und*).

[1]A CNF is a propositional formula which is a conjunction of clauses. A clause is a propositional formula which is a disjunction of literals. A literal is a propositional variable or the complement of a propositional variable (*e.g.* v or $\neg v$).

- The set of boolean variables corresponding to $\mathcal{AF}$, denoted as $\mathcal{V}(\mathcal{AF})$, corresponds to:

$$\mathcal{V}(\mathcal{AF}) = \bigcup_{i \in \{1,\dots,n\}} \{I_i, O_i, U_i\}$$

Following what has been said above, let define a first encoding for the *complete* semantics.

Definition 42. *Let $\mathcal{AF} = \langle A, K \rangle$ be an AF, with $|A| = k$ and $\phi : \{1,...,k\} \to A$ be an indexing of A. The SAT encoding C_1 defined on $\mathcal{V}(\mathcal{AF})$, is given by the conjunction of the formulae listed below:*

$$\bigwedge_{i \in \{1,\dots,n\}} ((I_i \vee O_i \vee U_i) \wedge (\neg I_i \vee \neg O_i) \wedge (\neg I_i \vee \neg U_i) \wedge (\neg O_i \vee \neg U_i)) \tag{7.1}$$

$$\bigwedge_{\{i \mid \phi(i)^- = \varnothing\}} (I_i \vee \neg O_i \vee \neg U_i) \tag{7.2}$$

$$\bigwedge_{\{i \mid \phi(i)^- \neq \varnothing\}} \left(I_i \vee \left(\bigvee_{\{j \mid (\phi(j),\phi(i)) \in K\}} \neg O_j \right) \right) \tag{7.3}$$

$$\bigwedge_{\{i \mid \phi(i)^- \neq \varnothing\}} \left(\bigwedge_{\{j \mid (\phi(j),\phi(i)) \in K\}} \neg I_i \vee O_j \right) \tag{7.4}$$

$$\bigwedge_{\{i \mid \phi(i)^- \neq \varnothing\}} \left(\bigwedge_{\{j \mid (\phi(j),\phi(i)) \in K\}} \neg I_j \vee O_i \right) \tag{7.5}$$

$$\bigwedge_{\{i \mid \phi(i)^- \neq \varnothing\}} \left(\neg O_i \vee \left(\bigvee_{\{j \mid (\phi(j),\phi(i)) \in K\}} I_j \right) \right) \tag{7.6}$$

$$\bigwedge_{\{i \mid \phi(i)^- \neq \varnothing\}} \left(\bigwedge_{\{k \mid (\phi(k),\phi(i)) \in K\}} \left(U_i \vee \neg U_k \vee \left(\bigvee_{\{j \mid (\phi(j),\phi(i)) \in K\}} I_j \right) \right) \right) \tag{7.7}$$

$$\bigwedge_{\{i \mid \phi(i)^- \neq \varnothing\}} \left(\left(\bigwedge_{\{j \mid (\phi(j),\phi(i)) \in K\}} (\neg U_i \vee \neg I_j) \right) \wedge \left(\neg U_i \vee \left(\bigvee_{\{j \mid (\phi(j),\phi(i)) \in K\}} U_j \right) \right) \right) \tag{7.8}$$

$$\bigvee_{i \in \{1,\dots,n\}} I_i \tag{7.9}$$

`Note:` The last clause has been added for technical reasons, due to the algorithm introduced in [23]. It restricts the result to "non-empty" complete *labellings (that is, at least one argument is labelled `in`). Verifying if the "empty" labelling is a* complete *labellings can be done afterward trivially by adding conjunctions forcing all U_i to be `true`.*

Let describe each part of C_1:

- Equation (7.1) states that for each argument i one and only one label has to be assigned.
- Equation (7.2) states that each unattacked argument must be labelled *`in`*.

- Equation (7.3) states that argument i is labelled $\mathtt{in}$ if all its attackers are labelled $\mathtt{out}$.
- Equation (7.4) settles the reverse (*i.e.* the "only if") condition of the precedent point.
- Equation (7.5) corresponds to the constraint that argument i is labelled $\mathtt{out}$ if at least one of its attackers is labelled $\mathtt{in}$.
- Equation (7.6) corresponds to the "only if" condition of the precedent point.
- Equation (7.7) states that argument i is labelled $\mathtt{und}$ if none of its attackers is labelled $\mathtt{in}$ and at least one of its attackers is labelled $\mathtt{und}$.
- Equation (7.8) corresponds to the "only if" condition of the precedent point.
- Equation (7.9) ensures non-emptiness, *i.e.* that at least one argument is labelled $\mathtt{in}$.

In [23], it has been given the six equivalent encodings as stated by the following proposition:

Proposition 15 ([23]). *Referring to the formulae listed in Definition 42 on the previous page, the following encodings are equivalent:*

- $C_1 : (7.1) \wedge (7.2) \wedge (7.3) \wedge (7.4) \wedge (7.5) \wedge (7.6) \wedge (7.7) \wedge (7.8) \wedge (7.9)$
- $C_2 : (7.1) \wedge (7.2) \wedge (7.3) \wedge (7.4) \wedge (7.5) \wedge (7.6) \wedge (7.7) \wedge (7.9)$
- $C_3 : (7.1) \wedge (7.2) \wedge (7.3) \wedge (7.5) \wedge (7.6) \wedge (7.7) \wedge (7.8) \wedge (7.9)$
- $C_4 : (7.1) \wedge (7.2) \wedge (7.3) \wedge (7.4) \wedge (7.6) \wedge (7.7) \wedge (7.8) \wedge (7.9)$
- $C_5 : (7.1) \wedge (7.2) \wedge (7.4) \wedge (7.6) \wedge (7.8) \wedge (7.9)$
- $C_6 : (7.1) \wedge (7.2) \wedge (7.3) \wedge (7.5) \wedge (7.7) \wedge (7.9)$

Most semantics can be encoded with such rules. Now for semantics that need some maximisation such as the *preferred* semantics, the SAT solver can be called iteratively until a maximal solution is found. There exists a huge amount of search strategies, each one leading to a different SAT-based algorithm. By sake of brevity, they will not be presented in this paper but the reader interested can refer to [38, 68, 23, 3] as examples.

7.1.2 Other indirect approach based algorithms

Other types of transformation can be used to solve argumentation problems. Here is a non exhaustive list of approaches. References are given to go further:

- ASP-based algorithms (*e.g.* [44]). Notice that in ICCMA 2019 session, *Aspartix*, the solver presented in [44], gave very good results.
- Neural network based algorithms (*e.g.* [55]). Although the ICCMA 2017 session shew that this solver is not efficient, it opens perspectives for a radical paradigm change in argumentation problem solving.

7.2 Direct or semi-direct approach based algorithms

In this section we compare the behaviour of our algorithm to other existing ones, using direct or "semi-direct" approach. By "semi-direct" we mean that the algorithm mainly and directly deals with AF and takes advantage of its structure, but in some cases, for some sub-problems as the computation of labellings/extensions of a part of the AF, uses an indirect solving method such as a transformation to SAT.

Notice that several algorithms, other than those presented in this section, use solving methods similar to the ones of the algorithms presented. By sake of brevity, our algorithm is compared only to one algorithm of each method type.

In order to illustrate how these other algorithms work, we will consider the AF shown in Figure 7.1 as running example and show how the preferred labellings are computed following the different algorithms. This particular AF has been chosen because its structure let appear clusters in it, it has four SCCs and there is an interesting hierarchy between them. These two last points are very relevant for the algorithms presented in Sections 7.2.1 and 7.2.2 on the current page and on page 78.

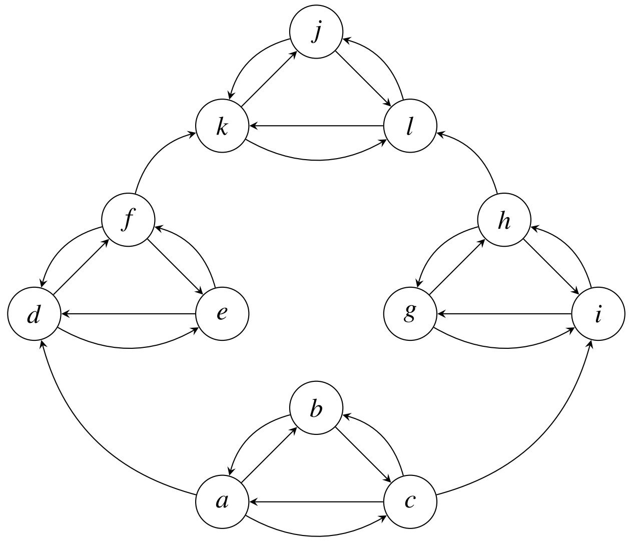


Figure 7.1: AF example $\mathcal{AF}$

7.2.1 Dynamic programming algorithm

In [43], Dvořák et al. proposed an algorithm based on a dynamic analysis of an argumentation framework. In the interest of brevity, we will just highlight the main idea of this algorithm (see [43] for a more detailed explanation).

Basically, this algorithm relies on the nice tree decomposition of a graph.

Definition 43. *(Tree decomposition). Let $G = (V,E)$ be a non directed graph. A tree decomposition of G is a pair $\langle \mathcal{T}, \mathcal{X} \rangle$ where $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$ is a tree and $\mathcal{X} = (X_t)_{t \in V_{\mathcal{T}}}$ is a set of so-called bags, which has to satisfy the following conditions:*

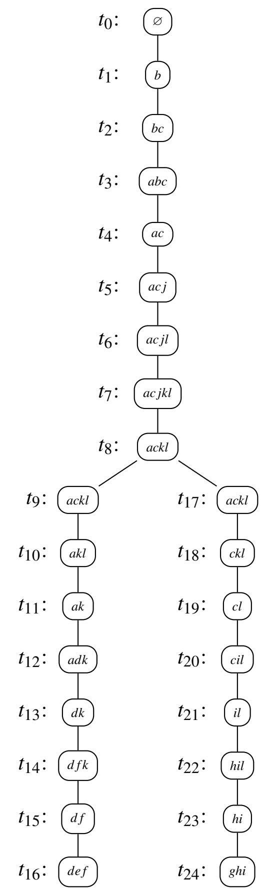


Figure 7.2: Nice tree decomposition of $\mathcal{AF}$

- $\bigcup_{t \in V_{\mathcal{T}}} X_t = V$, *i.e.* $\mathcal{X}$ *is a set covering of* V.
- *for each* $v \in V$, $\mathcal{T} \downarrow_{\{t | v \in X_t\}}$ *is a connected tree.*
- *for each* $\{v_i, v_j\} \in E$, $\{v_i, v_j\} \subseteq X_t$ *for some* $t \in V_{\mathcal{T}}$.

Definition 44. *(Width of a tree decomposition). Let* $\langle \mathcal{T}, \mathcal{X} \rangle$ *be a tree decomposition where* $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$ *is a tree and* $\mathcal{X} = (X_t)_{t \in V_{\mathcal{T}}}$ *is a set of so-called bags. The width of such a tree decomposition is given by:*

$$max\{card(X_t) | t \in V_{\mathcal{T}}\} - 1$$

Definition 45. *(Tree-width of a graph). Let* $G = (V, E)$ *be a non directed graph. The tree-width of G is defined by the minimum width over all its tree decompositions.*

Definition 46. *(Nice tree decomposition). A tree decomposition* $\langle \mathcal{T}, \mathcal{X} \rangle$ *of a graph G is called nice if* $\mathcal{T}$ *is a rooted tree and if each node* $t \in \mathcal{T}$ *is one of the following types:*

- *LEAF: t is a leaf of* $\mathcal{T}$

- *FORGET: t has only one child t' and $X_t = X_{t'} \setminus \{v\}$ for some $v \in X_{t'}$*
- *INSERT: t has only one child t' and $X_t = X_{t'} \cup \{v\}$ for some $v \notin X_{t'}$*
- *JOIN: t has two children t', t'' and $X_t = X_{t'} = X_{t''}$*

$v \backslash X_t$	X_{t_0}	X_{t_1}	X_{t_2}	X_{t_3}	X_{t_4}	X_{t_5}	X_{t_6}	X_{t_7}	X_{t_8}	X_{t_9}	$X_{t_{10}}$	$X_{t_{11}}$	$X_{t_{12}}$	$X_{t_{13}}$	$X_{t_{14}}$	$X_{t_{15}}$	$X_{t_{16}}$	$X_{t_{17}}$	$X_{t_{18}}$	$X_{t_{19}}$	$X_{t_{20}}$	$X_{t_{21}}$	$X_{t_{22}}$	$X_{t_{23}}$	$X_{t_{24}}$
a				✓	✓	✓	✓	✓	✓	✓	✓	✓	✓					✓							
b		✓	✓	✓																					
c			✓	✓	✓	✓	✓	✓	✓	✓								✓	✓	✓	✓				
d													✓	✓	✓	✓	✓								
e																	✓								
f															✓	✓	✓								
g																									✓
h																							✓	✓	✓
i																					✓	✓	✓	✓	✓
j						✓	✓	✓																	
k								✓	✓	✓	✓	✓	✓	✓	✓			✓	✓						
l							✓	✓	✓	✓	✓							✓	✓	✓	✓	✓	✓		

Table 7.1: Bags of $\mathscr{X}$

"✓" means that the vertex v corresponding to the current line belongs to the bag X_t corresponding to the current column

Example 30. *Figure 7.2 on the previous page shows one nice tree decomposition of the AF $\mathcal{AF}$, $\langle \mathscr{T}, \mathscr{X} \rangle$ where:*

- *$\mathscr{T} = (V_{\mathscr{T}}, E_{\mathscr{T}})$ with:*
  - *$V_{\mathscr{T}} = \{t_i | i \in [\![0, 24]\!]\}$*
  - *$E_{\mathscr{T}} = \{(t_i, t_{i+1}) | i \in [\![0, 15]\!] \cup [\![17, 23]\!]\} \cup \{(t_8, t_{17})\}$*
- *$\mathscr{X} = \{X_{t_i} | i \in [\![0, 24]\!]\}$ with each bag being as summarised in Table 7.1.*

As node type examples, according to Definition 46:

- *t_{24} is a LEAF type node (no child)*
- *t_2 is a FORGET type node (t_3 being a child of t_2)*
- *t_{10} is a INSERT type node (t_{11} being a child of t_{10})*
- *t_8 is a JOIN type node (t_9 and t_{17} being the children of t_8)*

The nice tree decomposition shown in Figure 7.2 on the previous page is one among all tree decompositions of $\mathcal{AF}$ with the minimal width, which is 4. In other words, the tree-width of $\mathcal{AF}$ is 4.

There exist other possible nice tree decompositions of $\mathcal{AF}$ with non minimal width. As an example, the one shown in Figure 7.3 on the next page has a width of 11.

To each nice tree node is associated a sub AF defined as following:

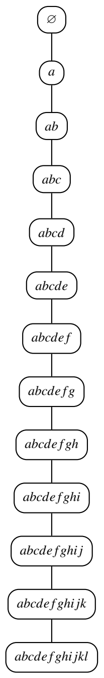


Figure 7.3: Nice tree decomposition of $\mathcal{AF}$

Definition 47. *(Tree node associated AF). Let $\mathcal{AF}$ be an AF and $\langle \mathcal{T}, \mathcal{X} \rangle$ be its tree decomposition where $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$ is a tree and $\mathcal{X} = (X_t)_{t \in V_{\mathcal{T}}}$ is a set of so-called bags. We denote by $X_{\geqslant t}$ the union of all bags $X_s \in \mathcal{X}$ such that s occurs in the subtree of $\mathcal{T}$ rooted at t.*

Let $t \in V_{\mathcal{T}}$ be a tree node. The AF af associated with t is defined as following:

$$af = \mathcal{AF} \downarrow_{X_{\geqslant t}}$$

Example 31. *Let take as example the node t_{12} in Figure 7.2 on page 74. According to Definition 47, we have:*

$$X_{\geqslant t_{12}} = X_{t_{12}} \cup X_{t_{13}} \cup X_{t_{14}} \cup X_{t_{15}} \cup X_{t_{16}} = \{a, d, e, f, k\}$$

We have so:

$$af = \mathcal{AF} \downarrow_{X_{\geqslant t_{12}}} = \mathcal{AF} \downarrow_{\{a,d,e,f,k\}}$$

Figure 7.4 on the following page shows the AF af associated with the node t_{12}.

Once the AF nice tree determined and the sub AFs associated to each tree node identified, the tree is explored from the bottom up. On each tree node, the labellings of its associated AF are computed. The node type (LEAF, INSERT, FORGET or JOIN) indicates which operations to do in order to update the computed set of labellings.

Notice that the sub AF associated with the tree root is the whole AF. So, at the tree root, all the labellings of the AF are found.

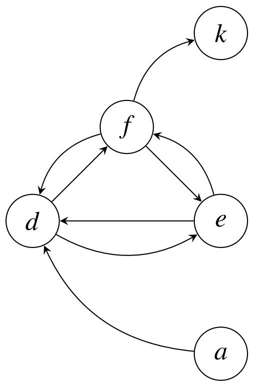


Figure 7.4: af, the AF associated with the node t_{12}

This is basically the general idea of this algorithm.

This algorithm is dynamic in the sense that we are interested in the labellings of sub AF that evolve dynamically following the nice tree decomposition. To each leaf is associated an initial AF that will be transformed forgetting and inserting argument nodes in it. This approach has the advantage of breaking the SCC and eventually the hardness of the AF problem. Nevertheless it has also some disadvantages.

Indeed, each step adds or removes at most one argument. The consequence is that a lot of updates are useless and a lot of space is used for potential correct labellings.

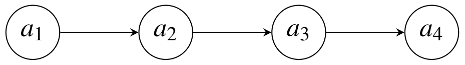


Figure 7.5: AF example

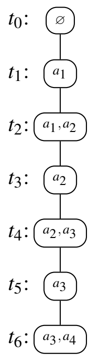


Figure 7.6: Nice tree decomposition

Example 32. *Let take as example the AF in Figure 7.5 and its nice tree decomposition in Figure 7.6.*

Although this AF admits only one preferred labelling which is $\{(a_1, \mathtt{in}), (a_2, \mathtt{out}), (a_3, \mathtt{in}), (a_4, \mathtt{out})\}$, *as we go from the leaf to the top the set of partial labellings will be updated 6 times and, at each tree node,* *we will have to consider all the potential partial labellings.*

Actually, this algorithm does not work directly with labellings but with “colorings”; a coloring is a 4-state argument mapping from which are determined the semantic extensions we are interested in. Without going too deep into the details of how this coloring works, we will just highlight the fact that for each argument attacked by an argument outside the current associated AF, four colorings have to be considered, according to the four possible status of that argument. As a consequence, a lot of space is used in order to ensure that all possibilities have been explored.

This algorithm and the *AFDivider* algorithm have both the ability to break the SCC and hopefully the hardness the AF. However, they differ on other points and the main one is how the combinatorial effect of potential labelling number is tackled. Although the *AFDivider* algorithm computes all cases for a given cluster, this combinatorial effect is limited to that particular cluster and is not propagated on the whole AF. As a consequence, space and computational time are spared.

7.2.2 SCC decomposition based algorithms

In [52], Beishui Liao proposed an algorithm that computes the labellings of an AF following its SCC decomposition.

Notice that if each SCC of a graph is considered as a super node, the resulting super graph will be acyclic.

We can thus have a hierarchical representation of this super graph: in the first layer are SCCs with no parents, in the second layer are contained all the SCCs whose parents are in the previous layers, and so on.

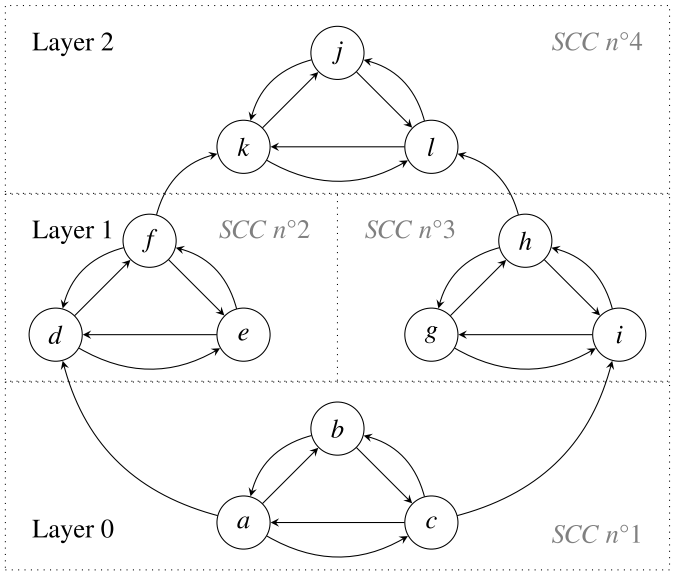


Figure 7.7: The SCC decomposition for $\mathcal{AF}$ (the first layer being Layer 0)

Example 33. *Figure 7.7 is the SCC hierarchical view of $\mathcal{AF}$.*

Given that the labellings of each SCC are influenced only by the ones of its parents, it is possible to guide the research of labellings following the hierarchical representation of the SCCs of the AF. This is the main idea of the algorithm.

Example 34. *In a first step, the labellings of the SCC n°1 are computed. The result is the following set of labellings:*

$$\{\ell_1^{scc1}, \ell_2^{scc1}, \ell_3^{scc1}\} \text{ with } \begin{cases} \ell_1^{scc1} &= \{(a, in)\}, \{(b, out)\}, \{(c, out)\}, \\ \ell_2^{scc1} &= \{(a, out)\}, \{(b, in)\}, \{(c, out)\}, \\ \ell_3^{scc1} &= \{(a, out)\}, \{(b, out)\}, \{(c, in)\} \end{cases}$$

Then, possible labellings of the SCCs n°2 and n°3 are computed considered the labellings of the parents SCCs, in this case SCC n°1. For instance, considering $\ell_1^{scc1} = \{(a, in)\}, \{(b, out)\}, \{(c, out)\}$:

- *For SCC n°2 we have:*
$$\{\ell_1^{scc2}, \ell_2^{scc2}\} \text{ with } \begin{cases} \ell_1^{scc2} &= \{(d, out)\}, \{(e, out)\}, \{(f, in)\}, \\ \ell_2^{scc2} &= \{(d, out)\}, \{(e, in)\}, \{(f, out)\} \end{cases}$$

- *For SCC n°3 we have:*
$$\{\ell_1^{scc3}, \ell_2^{scc3}, \ell_3^{scc3}\} \text{ with } \begin{cases} \ell_1^{scc3} &= \{(g, out)\}, \{(h, out)\}, \{(i, in)\}, \\ \ell_2^{scc3} &= \{(g, out)\}, \{(h, in)\}, \{(i, out)\}, \\ \ell_3^{scc3} &= \{(g, in)\}, \{(h, out)\}, \{(i, out)\} \end{cases}$$

The same thing must be done considering ℓ_2^{scc1} and ℓ_3^{scc1}.

Afterwards, the labellings of SCC n°4 are computed according to the compatible SCC parents labellings. In the interest of brevity we will not give the entire result as this AF has 47 distinct preferred labellings.

The great advantage of this approach is that no useless computation is made. When going from one layer to another, only possible labellings are considered. This reduces considerably the computational time.

Although not proposed in this paper, it is possible to parallelize the computation when there are independent branches in the acyclic super graph. But even though a distributed version of this algorithm had been proposed, it would still be very different from the *AFDivider* algorithm.

Indeed, this algorithm is profitable only if there are several SCCs and if the hardness of solving the AF problem is not inside the SCCs. The major difference is that the *AFDivider* algorithm is able to look inside SCCs and hopefully break the hardness by finding clusters in it. Another difference is that the way *AFDivider* parallelizes the labelling computation is not subject to any hierarchy of SCCs. As a consequence, there are no sequential constraints on the distributed computation made to construct the labellings. Finally, the used clustering method tries to balance the cluster sizes (in terms of number of arguments) so that hopefully the workload may be also balanced.

7.2.3 Parallel algorithms

The algorithm proposed by Cerruti et al. in [25], named *P-SCC-REC*, has several common points with the *AFDivider* algorithm. Indeed, both algorithms are distributed and they are able to look inside SCCs. Nevertheless, the way of distributing and of "cutting" of the AF are completely different.

The *P-SCC-REC* algorithm is rather complex. We are going to highlight its main concepts (see [25] for additional information).

It is a recursive algorithm. In one recursion level, the following steps are performed:

- As in the *AFDivider* algorithm, the grounded labelling is computed and only the hard part of the AF is considered for the next steps.
- As in the Beishui Liao's algorithm, an SCC hierarchical view of the AF is determined.
- For each SCC, a greedy labelling computing is performed, considering that all arguments attacking the given SCC is labelled $\mathtt{out}$. This computation is made in a distributed way, parallelized following the SCCs.
- For each layer:
  - The labelling of the SCCs are computed according to the labelling of their SCC ancestors. This computation is made in a distributed way, parallelized both following the SCCs and the SCC ancestors labellings.
    - ∗ In some cases when the SCC ancestors labelling does not allow to determine quickly the labellings of the current SCC, *P-SCC-REC* is called recursively. The sub-AF on which it is called is that particular SCC, sligthly modified to fit with the labelling of attackers from its SCC ancestors: arguments that are attacked by $\mathtt{in}$-labelled arguments from its SCC ancestors are removed.[2]
  - Following the previous step, the set of SCC ancestors labellings of the next layer is determined.
- Some computations of SCC labelling are made using a transformation to SAT.

Example 35. *Applied to $\mathcal{AF}$,* P-SCC-REC *will behave a bit like Beishui Liao's algorithm as there is no argument labelled $\mathtt{in}$ or $\mathtt{out}$ in the grounded labelling of $\mathcal{AF}$.*

Notice that, given the labelling $\ell_1^{scc1} = \{(a, \mathtt{in})\}, \{(b, \mathtt{out})\}, \{(c, \mathtt{out})\}$, when computing the labellings of the SCC n°2, P-SCC-REC *will be recursively called on the AF shown in Figure 7.8.*

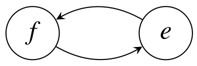


Figure 7.8: *SCC n*°2 under ℓ_1^{scc1}

The *P-SCC-REC* algorithm will look inside an SCC if its SCC ancestor labelling allows it, not according to the size of this SCC and its possible hardness, whereas the *AFDivider* algorithm will try to found clusters similar in size whether it is necessary to break SCCs or not.

There is another aspect of *P-SCC-REC* algorithm that may narrow its performance. If we put aside the greedy phase of the algorithm, the algorithm follows the SCC hierarchical view of the AF and parallelizes following the SCCs in one layer, and following the ancestor labellings. This later parallelization causes two problems:

1. Most of the time, it makes the number of threads explodes and so overloads the CPUs.
2. It leads to redundant computation as the computation cases are not based on the states of input arguments of the current SCC.

[2] *P-SCC-REC* is called recursively with a parameter which corresponds to the set of arguments attacked by $\mathtt{und}$-labelled arguments from the SCC ancestors.

Example 36. *Let consider the step to compute the SCC n°4 labellings.*

- *As an illustration of point 1:*
  - *We have 21 distinct SCC ancestor labellings and so 21 threads will be created. Although $\mathcal{AF}$ is a small AF, the amount of threads is rather important. On a bigger one, the number of threads could quickly overload the CPUs.*
- *As an example of point 2:*
  - *Even if several distinct SCC ancestor labellings are equal when restricted to the arguments f and h, the labelling computation will be made for each of them, which is highly redundant.*

It is true that some of the cases computed by the *AFDivider* algorithm may be unused in the reunifying phase (bear in mind that it is not possible to know them in advance) but there is no waiting time due to a hierarchical view of the AF, and there is no redundant computation. Furthermore, if the AF is not too dense, the number of threads will not explode, even though the number of labellings is huge.

Part IV

Higher-Order Attack Argumentation Frameworks: Background

Part presentation:

Higher-Order Attack Argumentation Frameworks are Argumentation Frameworks in which an attack can have as target an attack. In this background are presented two of such frameworks:

- The so-called "*Recursive Argumentation Framework*" (denoted **RAF**), introduced in [18], consists of a set of arguments, a set of attacks, and mapping functions that associate to each attack a source and a target.

- The so-called "*Argumentation Framework with Recursive Attack*" (denoted **AFRA**), introduced in [5, 6], consists of a set of arguments and a set of named attacks.

This implies that they cannot be represented as a directed graph like AFs. However their graphical representation is as much as intuitive, as shown in Figure 7.9.

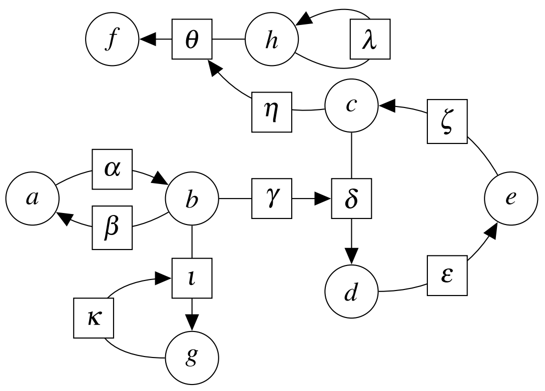


Figure 7.9: Example of a RAF/AFRA

Formally, RAFs and AFRAs are defined as follows:

Definition 48 (Recursive argumentation framework - RAF)**.** *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 $s : K \rightarrow A$ and $t : K \rightarrow A \cup K$ are functions respectively mapping each attack to its source and to its target. The set of all possible RAFs is denoted as Φ_{raf}.*

Definition 49 (AFRA)**.** *An Argumentation Framework with Recursive Attacks (AFRA) is a pair $\mathcal{AFRA} = \langle A, K\rangle$ where A is a set of arguments and K is a set of named attacks, namely pairs (a, x) such that $a \in A$ and $x \in (A \cup K)$. Given an attack $\alpha = (a, x)$, we say that a is the source of α (denoted as "$src(\alpha)$") and x is the target of α (denoted as "$trg(\alpha)$"). The set of all possible AFRAs is denoted as Φ_{afra}.*

`Note:` *Although the definitions of AFRA and RAF are equivalent and that both frameworks can represent the same relations between arguments, substantial differences appear afterward in the way that semantics are computed.*

`Note:` *In the following, to simplify the notation and highlight the link between AFRA and RAF, while refering to AFRAs, we will use "$s(\alpha)$" (resp. "$t(\alpha)$") instead of "$src(\alpha)$" (resp. "$trg(\alpha)$") to represent the source (resp. the target) of an attack α.*

Example 37. *Figure 7.9 on the previous page shows a RAF/AFRA example. In all this document, arguments (in Latin letter) will be represented by a round box, while attacks (in Greek letters) will be represented by directed edges from a "circular node" to another "node" (squared or circular) through a "square node" containing the name of the attack. As one can notice, the attacks γ, κ and η have as target an attack.*

In Chapter 8 are presented the background for AFRAs and in Chapter 9 the background for RAFs.

Chapter 8

Argumentation Framework with Recursive Attacks (AFRA)

In this chapter, we first present AFRA *semantics* (Section 8.1). Then, we present the relations between AFRA and AF (Section 8.2 on page 90).

8.1 Extension-based Semantics

What differs from AF to AFRA is that in an AFRA an attack can have an attack for target. As a consequence, an attack is not always "*acceptable*". In order to express this fact, an AFRA "*extension*-based semantic", that is, a function that defines the solutions of an AFRA, produces AFRA-*extension* that not only contains arguments but also attacks. As for extensions in AF, the idea behind the notion of AFRA-*extension* is that when presented together, the elements of the AFRA-*extension* (*i.e.* arguments plus attacks) win the argumentation.

8.1.1 Definitions

Formally, an AFRA *extension*-based semantic is defined as follows:

Definition 50 (Extension-based Semantics)**.** *Let σ be a function over Φ_{afra}. σ is said to be an AFRA extension-based semantics iff the following property holds:*

$$\forall \mathcal{AFRA} \in \Phi_{afra},\ \sigma(\mathcal{AFRA}) \subseteq 2^{A \cup K},\ \text{with } \mathcal{AFRA} = \langle A, K \rangle$$

An AFRA-*extension* is thus defined as follows:

Definition 51 (AFRA-extension)**.** *A set S is said to be an AFRA-*extension *of some $\mathcal{AFRA} = \langle A, K \rangle$ if it satisfies: $S \subseteq A \cup K$.*

Intuitively, any attack that does not belong to S is understood as non-acceptable and, in this sense, it cannot defeat its target. Two types of *defeat relation* are defined for AFRAs:

Definition 52 (Direct defeat [6]). *Let $\mathcal{AFRA} = \langle A, K\rangle$ be an AFRA, $\alpha \in K$ be an attack and $x \in (A \cup K)$ be an argument or an attack. We say that α directly defeats x iff $t(\alpha) = x$.*

Definition 53 (Indirect defeat [6]). *Let $\mathcal{AFRA} = \langle A, K\rangle$ be an AFRA, $\alpha \in K$ be an attack and $\beta \in K$ be an attack. We say that α indirectly defeats β iff $t(\alpha) = s(\beta)$.*

The following notion captures both *defeat relations*:

Definition 54 (Defeat [6]). *Let $\mathcal{AFRA} = \langle A, K\rangle$ be an AFRA, $\alpha \in K$ be an attacks and $x \in (A \cup K)$ be an argument or an attack. We say that α defeats β, denoted as $\alpha \to_K x$, iff α directly or indirectly defeats β.*

From the defeat relation, we formally define the notion of *acceptability* as follows:

Definition 55 (Acceptability [6]). *Let $\mathcal{AFRA} = \langle A, K\rangle$ be an AFRA, $S \subseteq (A \cup K)$ be a subset of the elements of $\mathcal{AFRA}$ and $x \in (A \cup K)$ be an argument or an attack. We say that x is acceptable w.r.t. S (or defended by S) iff $\forall \alpha \in K$ s.t. $\alpha \to_K x$, $\exists \beta \in S$ s.t. $\beta \to_K \alpha$.*

From these notion of defeat and acceptability, we can define the sets of defeated and acceptable element *w.r.t.* some set of elements *S*.

Definition 56 (Defeat and acceptable set). *Let $\mathcal{AFRA} = \langle A, K\rangle$ be an AFRA.*

- *Let S be a subset of the elements of $\mathcal{AFRA}$. We denote by AFRA-$Def(S) = \{x | x \in (A \cup K), \exists \alpha \in S$ s.t. $\alpha \to_K x\}$ the set of all the elements defeated by S.*
- *Let S be a subset of the elements of $\mathcal{AFRA}$. We denote by AFRA-$Acc(S) = \{x | x \in (A \cup K), \forall \alpha \in K$ s.t. $\alpha \to_K x$, $\exists \beta \in S$ s.t. $\beta \to_K \alpha\}$ the set of all the elements defended by S.*

`Note`*: These definitions of defeat and acceptable sets slightly differ from the ones given in [6], but they are equivalent. We chose these ones to be closer the corresponding definitions for RAF (once more, in order to facilitate the comparison between the two approaches).*

As for Dung's Argumentation Framework, based on the notion of acceptability, semantics have been defined for AFRAs. We will focus only on semantics we are interested in although much semantics have been defined.

Definition 57 (AFRA Semantics [6]). *Let $\mathcal{AFRA} = \langle A, K\rangle$ be an AFRA and $S \subseteq (A \cup K)$ be a subset of its elements. S is said to be an extension:*

1. *AFRA*-conflict-free *iff $S \cap$ AFRA-$Def(S) = \varnothing$.*
2. *AFRA*-admissible *iff it is AFRA*-conflict-free *and $S \subseteq$ AFRA-$Acc(S)$.*
3. *AFRA*-complete *iff it is AFRA*-conflict-free *and $S =$ AFRA-$Acc(S)$.*
4. *AFRA*-preferred *iff it is a $\subseteq$-maximal AFRA*-admissible *extension.*
5. *AFRA*-grounded *iff it is a $\subseteq$-minimal AFRA*-complete *extension.*
6. *AFRA*-stable *iff it is AFRA*-conflict-free *and $S \cup$ AFRA-$Def(S) = (A \cup K)$.*
7. *AFRA*-semi-stable *extension iff it is an AFRA*-complete *extension such that $S \cup$ AFRA-$Def(S)$ is maximal w.r.t. $\subseteq$.*

Given an AFRA $\mathfrak{AFRA}$, we denote by $co(\mathfrak{AFRA})$ (resp. $gr(\mathfrak{AFRA})$, $st(\mathfrak{AFRA})$, $sst(\mathfrak{AFRA})$, $pr(\mathfrak{AFRA})$) the set of AFRA-extensions of $\mathfrak{AFRA}$ under the semantics AFRA-*complete* (resp. AFRA-*grounded*, AFRA-*stable*, AFRA-*semi-stable* and AFRA-*preferred*).

Example 38. *Let consider the AFRA shown in Figure 7.9 on page 83. The AFRA-extensions corresponding to the semantics mentioned are given in Table 8.1 on the next page. By sake of space the AFRA-*admissible *and the AFRA-*conflict-free *semantics are not given.*

8.1.2 Properties

In [6], the following propositions and theorem have been proven.

Proposition 16. *The set of all AFRA-*admissible *extensions forms a complete partial order with respect to* $\subseteq$.

Theorem 1. *The following assertions hold:*

- *every AFRA-*admissible *extension is also AFRA-*conflict-free
- *every AFRA-*complete *extension is also AFRA-*admissible
- *the AFRA-*grounded *extension is also AFRA-*complete
- *every AFRA-*preferred *extension is also AFRA-*complete
- *every AFRA-*semi-stable *extension is also AFRA-*preferred
- *every AFRA-*stable *extension is also AFRA-*semi-stable

Proposition 17 (extension semantics cardinality)**.** *The following properties hold:*

- *There is always at least one AFRA-*conflict-free *extension.*
- *There is always at least one AFRA-*admissible *extension.*
- *There is always at least one AFRA-*complete *extension.*
- *There is always a unique AFRA-*grounded *extension.*
- *There is always at least one AFRA-*preferred *extension.*
- *There is always at least one AFRA-*semi-stable *extension.*
- *It may be the case that there is no AFRA-*stable *extension.*

Figure 8.1 on page 89 illustrates Theorem 1 and Proposition 17. Notice that we have a partial ordering similar to the one for AF (see Figure 1.1 on page 10).

		AFRA-extensions			
		S_1	S_2	S_3	S_4
Arguments or attacks	a		✓		
	b			✓	✓
	c			✓	✓
	d			✓	✓
	e				
	f			✓	✓
	g		✓	✓	
	h				
	α		✓		
	β			✓	✓
	γ			✓	✓
	δ		✓		
	ε			✓	✓
	ζ				
	η		✓	✓	✓
	θ				
	ι				✓
	κ		✓	✓	
	λ				
AFRA Semantics	AFRA-*complete*	✓	✓	✓	✓
	AFRA-*grounded*	✓			
	AFRA-*preferred*		✓	✓	✓
	AFRA-*stable*				

In the first part of the table, $i \checkmark j$ means that element i belongs to extension j.
In the second part of the table, $i \checkmark j$ means that j is an extension of the semantics i.

Table 8.1: AFRA semantics

$$\begin{tikzcd}
 & & & \text{AFRA-}\textit{conflict-free}\ (+) & \\
 & & \text{AFRA-}\textit{admissible}\ (+) \arrow[ur] & & \\
 & \text{AFRA-}\textit{complete}\ (+) \arrow[ur] & & & \\
\text{AFRA-}\textit{grounded}\ (1) \arrow[ur] & & \text{AFRA-}\textit{preferred}\ (+) \arrow[ul] & & \\
 & & & \text{AFRA-}\textit{semi-stable}\ (+) \arrow[ul] & \\
 & & & & \text{AFRA-}\textit{stable}\ (*) \arrow[ul]
\end{tikzcd}$$

Figure 8.1: AFRA semantics partial ordering

The cardinality of each semantics is represented between parenthesis.
"$*$" means zero or more, "$+$" means one or more.

8.2 Relation between AFRA and AF

There exists a way to express AFRAs as AFs.

Definition 58 (AFRA expressed as AF [6])**.** *Let $\mathcal{AFRA} = \langle A, K \rangle$ be an AFRA. The corresponding AF of $\mathcal{AFRA}$, $\widetilde{\mathcal{AFRA}} = \left\langle \widetilde{A}, \widetilde{K} \right\rangle$ is defined as following:*

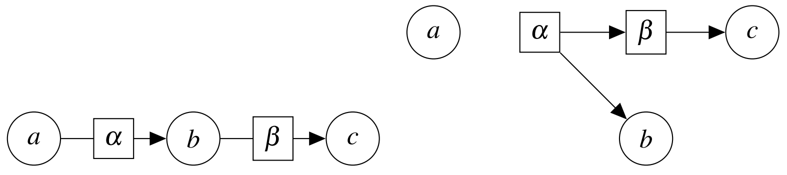
- $\widetilde{A} = A \cup K$
- $\widetilde{K} = \{(a,b) | (a,b) \in (A \cup K)^2$ *and* $a \rightarrow_K b\}$

In [6] has been shown a very important result concerning AFRAs and their corresponding AFs: there exists a one-to-one correspondence between extensions in AFRAs and their corresponding AFs for some semantics. Here we will focus on Dung's semantics and the *semi-stable* one but the result shown in [6] concerned much semantics.

Proposition 18 (Semantics correspondence: AFRA expressed as AF [6])**.** *Let $\mathcal{AFRA} = \langle A, K \rangle$ be an AFRA and $\widetilde{\mathcal{AFRA}} = \left\langle \widetilde{A}, \widetilde{K} \right\rangle$ its corresponding AF. Let $S \subseteq A \cup K$.*

- *S is an AFRA-*complete *extension for $\mathcal{AFRA}$ iff S is a* complete *extension for $\widetilde{\mathcal{AFRA}}$.*
- *S is an AFRA-*preferred *extension for $\mathcal{AFRA}$ iff S is a* preferred *extension for $\widetilde{\mathcal{AFRA}}$.*
- *S is an AFRA-*grounded *extension for $\mathcal{AFRA}$ iff S is a* grounded *extension for $\widetilde{\mathcal{AFRA}}$.*
- *S is an AFRA-*stable *extension for $\mathcal{AFRA}$ iff S is a* stable *extension for $\widetilde{\mathcal{AFRA}}$.*
- *S is an AFRA-*semi-stable *extension for $\mathcal{AFRA}$ iff S is a* semi-stable *extension for $\widetilde{\mathcal{AFRA}}$.*

`Note:` *This correspondence does not correspond to a conservative generalization of AF. Indeed, if we consider for instance $\mathcal{AFRA} = \langle A, K \rangle$, the non recursive AFRA illustrated in Figure 8.2(a), then the set $\{\alpha, c\}$ is an AFRA-*admissible *set. Notice that $\{\alpha, c\}$ is an admissible extension of $\widetilde{\mathcal{AFRA}}$ as represented in Figure 8.2(b). Nevertheless, if we read $\mathcal{AFRA}$ as an AF (i.e. without naming its attacks) c cannot be accepted without a. This is due to the fact that the link between an attack and its source is broken in the AFRA semantics (as it can be seen in Figure 8.2(b)).*



(a) A non recursive AFRA (b) Corresponding AFRA expressed as AF

Figure 8.2: An example of AFRA expressed as AF

`Note:` *See Section 9.3 on page 99 for the relation between RAFs and AFRAs.*

Chapter 9

Recursive Argumentation Framework (RAF)

In this chapter, we first present RAF *structure-based semantics*, the counterpart of AF extension-based semantics (Section 9.1). Then, we present the relations between RAF and AF that already exist in the literature (Section 9.2 on page 94). Notice that several RAF notions/ideas are similar to those defined for AFRAs. A comparison between the two approaches is given in Section 9.3 on page 99.

9.1 Structure Semantics

As in AFRA, what differs from AF to RAF is that in a RAF an attack can have an attack for target. As a consequence, an attack is not always "*valid*". In order to express this fact, a RAF "*structure*-based semantic", that is, a function that defines the solutions of a RAF produces *structures*: a couple whose first element is a set of arguments and the second, a set of attacks. As for extensions in AF (or in AFRA), the idea behind the notion of *structure* is that when presented together, the elements of the structure (*i.e.* arguments plus attacks) win the argumentation.

9.1.1 Definitions

Formally, a RAF *structure*-based semantic is defined as follows:

Definition 59 (Structure-based Semantics). *Let σ be a function over Φ_{raf}. σ is said to be a RAF structure-based semantics iff the following property holds:*

$$\forall \mathcal{RAF} \in \Phi_{raf},\ \sigma(\mathcal{RAF}) \subseteq 2^A \times 2^K,\ \textit{with}\ \mathcal{RAF} = \langle A, K, s, t \rangle$$

A *structure* is thus defined as follows:

Definition 60 (Structure). *A pair $\mathfrak{U} = \langle S, Q \rangle$ is said to be a* structure *of some $\mathcal{RAF} = \langle A, K, s, t \rangle$ if it satisfies: $S \subseteq A$ and $Q \subseteq K$. Notice that by $x \in \mathfrak{U}$ we mean: $x \in S \cup Q$.*

Intuitively, the set S represents the set of "acceptable arguments" *w.r.t.* the structure $\mathcal{U}$, while Q represents the set of "valid attacks" *w.r.t.* $\mathcal{U}$. Any attack that does not belong to Q is understood as non-valid and, in this sense, it cannot defeat its target.

Definition 61 (Defeat and Inhibition in RAF). *Let $\mathcal{U} = \langle S, Q \rangle$ be a structure. The set of all arguments* defeated *by $\mathcal{U}$, denoted $RAF\text{-}Def(\mathcal{U})$, is defined as follows:*

$$RAF\text{-}Def(\mathcal{U}) = \{a \in A | \exists \alpha \in Q \text{ s.t. } s(\alpha) \in S \text{ and } t(\alpha) = a\}$$

The set of all attacks inhibited *by $\mathcal{U}$, denoted $RAF\text{-}Inh(\mathcal{U})$, is defined as follows:*

$$RAF\text{-}Inh(\mathcal{U}) = \{\alpha \in K | \exists \beta \in Q \text{ s.t. } s(\beta) \in S \text{ and } t(\beta) = \alpha\}$$

The counterpart of defeat/inhibition is the notion of *acceptance*:

Definition 62 (RAF Acceptability). *An element $x \in (A \cup K)$ is said to be acceptable w.r.t. some structure $\mathcal{U}$ iff every attack $\alpha \in K$ with $t(\alpha) = x$ satisfies one of the two following conditions:*

- $s(\alpha) \in RAF\text{-}Def(\mathcal{U})$
- $\alpha \in RAF\text{-}Inh(\mathcal{U})$

By $RAF\text{-}Acc(\mathcal{U})$ we denote the set containing all acceptable arguments and attacks with respect to $\mathcal{U}$.

For any pair of structures $\mathcal{U} = \langle S, Q \rangle$ and $\mathcal{U}' = \langle S', Q' \rangle$, we write $\mathcal{U}' \sqsubseteq \mathcal{U}'$ *iff* $(S \cup Q) \subseteq (S' \cup Q')$ and we write $\mathcal{U} \sqsubseteq_{ar} \mathcal{U}'$ *iff* $S \subseteq S'$. As usual, we say that a structure $\mathcal{U}$ is $\sqsubseteq$-maximal (resp. $\sqsubseteq_{ar}$-maximal) *iff* every $\mathcal{U}'$ that satisfies $\mathcal{U} \sqsubseteq \mathcal{U}'$ (resp. $\mathcal{U} \sqsubseteq_{ar} \mathcal{U}'$) also satisfies $\mathcal{U}' \sqsubseteq \mathcal{U}$ (resp. $\mathcal{U}' \sqsubseteq_{ar} \mathcal{U}$).

Inspired by Dung's AF semantics, the first RAF structure-based semantics, that have been defined in [18], are the following ones:

Definition 63 (RAF structure semantics). *Let $\mathcal{U} = \langle S, Q \rangle$ be a structure over some RAF $\mathcal{RAF} = \langle A, K, s, t \rangle$. $\mathcal{U}$ is said to be:*

1. *RAF*-conflict-free *iff $S \cap RAF\text{-}Def(\mathcal{U}) = \varnothing$ and $Q \cap RAF\text{-}Inh(\mathcal{U}) = \varnothing$.*
2. *RAF*-naive *iff it is a $\sqsubseteq$-maximal RAF*-conflict-free *structure.*
3. *RAF*-admissible *iff it is RAF*-conflict-free *and $(S \cup Q) \subseteq RAF\text{-}Acc(\mathcal{U})$.*
4. *RAF*-complete *iff it is RAF*-conflict-free *and $(S \cup Q) = RAF\text{-}Acc(\mathcal{U})$.*
5. *RAF*-grounded *iff it is a $\sqsubseteq$-minimal RAF*-complete *structure.*
6. *RAF*-preferred *iff it is a $\sqsubseteq$-maximal RAF*-admissible *structure.*
7. *RAF*-arg-preferred *iff it is a $\sqsubseteq_{ar}$-maximal RAF*-preferred *structure.*
8. *RAF*-stable *iff $S = A \setminus RAF\text{-}Def(\mathcal{U})$ and $Q = K \setminus RAF\text{-}Inh(\mathcal{U})$.*

Notice that the RAF-*semi-stable* semantics has not been defined in [18]. This is a contribution of this thesis. See Section 10.1 on page 102.

Example 39. *Let consider the RAF shown in Figure 7.9 on page 83. The structures corresponding to the semantics mentioned in Definition 63 are given in Table 9.1 on the following page. By sake of space the RAF*-admissible *and the RAF*-conflict-free *semantics are not given.*

		Structures			
		$\mathcal{U}_1$	$\mathcal{U}_2$	$\mathcal{U}_3$	$\mathcal{U}_4$
Arguments or attacks	a		✓		
	b			✓	✓
	c			✓	✓
	d			✓	✓
	e				
	f			✓	✓
	g		✓	✓	
	h				
	α	✓	✓	✓	✓
	β	✓	✓	✓	✓
	γ	✓	✓	✓	✓
	δ		✓		
	ε	✓	✓	✓	✓
	ζ	✓	✓	✓	✓
	η	✓	✓	✓	✓
	θ				
	ι				✓
	κ	✓	✓	✓	✓
	λ	✓	✓	✓	✓
Semantics with structures	RAF-*complete*	✓	✓	✓	✓
	RAF-*grounded*	✓			
	RAF-*preferred*		✓	✓	✓
	RAF-*arg-preferred*		✓	✓	
	RAF-*stable*				

In the first part of the table, $i \checkmark j$ means that element i belongs to structure j.
In the second part of the table, $i \checkmark j$ means that j is a structure of the semantics i.

Table 9.1: RAF semantics with structures

9.1.2 Properties

In [18], Propositions 19 and 20 and Theorem 2 have been proven.

Proposition 19. *The set of all RAF-*admissible *structures forms a complete partial order with respect to* $\sqsubseteq$*. Furthermore, for every RAF-*admissible *structure* $\mathfrak{U}$*, there exists a RAF-*preferred *(and a RAF-*arg-preferred*)* $\mathfrak{U}'$ *such that* $\mathfrak{U} \sqsubseteq \mathfrak{U}'$*.*

Theorem 2. *The following assertions hold:*

- *every RAF-*naive *structure is also RAF-*conflict-free
- *every RAF-*admissible *structure is also RAF-*conflict-free
- *every RAF-*complete *structure is also RAF-*admissible
- *the RAF-*grounded *structure is also RAF-*complete
- *every RAF-*preferred *structure is also RAF-*complete
- *every RAF-*arg-preferred *structure is also RAF-*preferred
- *every RAF-*stable *structure is also RAF-*arg-preferred
- *every RAF-*stable *structure is also a RAF-*naive

Proposition 20 (Structure semantics cardinality)**.** *The following properties hold:*

- *There is always at least one RAF-*conflict-free *extension.*
- *There is always at least one RAF-*naive *extension.*
- *There is always at least one RAF-*admissible *extension.*
- *There is always at least one RAF-*complete *extension.*
- *There is always a unique RAF-*grounded *extension.*
- *There is always at least one RAF-*preferred *extension.*
- *There is always at least one RAF-*arg-preferred *extension.*
- *It may be the case that there is no RAF-*stable *extension.*

Figure 9.1 on the following page illustrates Theorem 2 and Proposition 20. Notice that we have a partial ordering similar to the one for AF (see Figure 1.1 on page 10).

9.2 Relation between RAF and AF

In this section the relation between RAFs and AFs is briefly presented. We first present semantics relation (Section 9.2.1 on page 96) then a flattening process that transforms RAF into AF (Section 9.2.2 on page 98).

$$\begin{tikzcd}[column sep=tiny]
 & & & \text{RAF-\textit{conflict-free} (+)} & \\
 & & \text{RAF-\textit{admissible} (+)} \arrow[ur] & & \text{RAF-\textit{naive} (+)} \arrow[ul] \\
 & \text{RAF-\textit{complete} (+)} \arrow[ur] & & & \\
\text{RAF-\textit{grounded} (1)} \arrow[ur] & & \text{RAF-\textit{preferred} (+)} \arrow[ul] & & \\
 & & & \text{RAF-\textit{arg-preferred} (+)} \arrow[ul] & \\
 & & & & \text{RAF-\textit{stable} (*)} \arrow[ul] \arrow[uuuu, bend right]
\end{tikzcd}$$

Figure 9.1: RAF semantics partial ordering

The cardinality of each semantics is represented between parenthesis.
"$*$" means zero or more, "$+$" means one or more.

9.2.1 Semantics correspondence

In order to establish a point of comparison, we define the notion of *D-structure*:

Definition 64 (D-structure)**.** *A* d-structure $\mathcal{U} = \langle S, Q\rangle$ *is a structure that satisfies:*

$$(RAF\text{-}Acc(\mathcal{U}) \cap K) \subseteq Q$$

`Note:` *Following the previous definition, all valid attacks w.r.t. a d-structure $\mathcal{U}$ belong to $\mathcal{U}$.*

Example 40. *Let consider $\mathcal{RAF} = \langle A, K, s, t\rangle$, the RAF illustrated in Figure 7.9 on page 83 and let consider $\mathcal{U}_1$, $\mathcal{U}_2$, $\mathcal{U}_3$ and $\mathcal{U}_4$, the structures shown in Table 9.1 on page 93. We have: $\mathcal{U}_1$, $\mathcal{U}_2$, $\mathcal{U}_3$ and $\mathcal{U}_4$ being d-structures.*

RAF semantics for d-structures can be defined as follows:

Definition 65. *A* conflict-free *(respectively* naive, admissible, complete, preferred, grounded, stable*) d-structure is a RAF-*conflict-free *(respectively RAF-*naive*, RAF-*admissible*, RAF-*complete*, RAF-*preferred*, RAF-*grounded*, RAF-*stable*) structure which is also a d-structure.*

Given the definition of the RAF-*complete* semantics, the following property holds:

Proposition 21. *Every RAF-*complete *structure is a d-structure.*

It is interesting to note that an AF can be viewed as a RAF without recursive attack, that is without attack whose target is an attack. Such a RAF is called a "*non recursive framework*" and is formally defined as follows:

Definition 66 (Non recursive framework)**.** *A framework $\mathcal{RAF} = \langle A, K, s, t\rangle$ is said to be non-recursive iff :*

$$\forall \alpha \in K,\ t(\alpha) \in A$$

Example 41. *Let consider $\mathcal{RAF} = \langle A, K, s, t\rangle$, the RAF illustrated in Figure 9.2. $\mathcal{RAF}$ is a non recursive framework.*

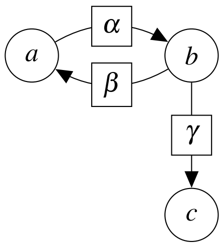


Figure 9.2: A non recursive RAF

AF extension-based semantics produce sets of arguments. Unlikely, RAF structure-based semantics produce pairs of arguments and attacks. In order to make a bridge between those two frameworks and compare them, the notion of extensions *w.r.t.* a given $\mathcal{RAF}$ has to be defined:

Definition 67 (Argument extensions)**.** *Given $\mathcal{RAF} = \langle A, K, s, t\rangle$. Let $S \subseteq A$ be a set of arguments. S is said to be a* conflict-free *extension (respectively* naive, admissible, complete, preferred, grounded, stable*) w.r.t. $\mathcal{RAF}$ iff there is some $Q \subseteq K$ such that $\mathfrak{U} = \langle S, Q\rangle$ is a* conflict-free *(respectively* naive, admissible, complete, preferred, grounded, stable*) d-structure of $\mathcal{RAF}$.*

Example 42. *Let consider the non recursive framework in Figure 9.2 on the previous page. We have: $\{a, c\}$ being a* complete *extension w.r.t. $\mathcal{RAF}$. Indeed: $\mathfrak{U} = \langle\{a, c\}, \{\alpha, \beta, \gamma\}\rangle$ is a RAF-*complete *d-structure of $\mathcal{RAF}$.*

Finally, with the notions defined above, the following theorem and its corollary can be established:

Theorem 3. *For each semantics $\sigma \in \{$*conflict-free, naive, admissible, complete, preferred, grounded, stable*$\}$: A set of arguments $S \subseteq A$ is a σ-extension w.r.t. some non-recursive $\mathcal{RAF} = \langle A, K, s, t\rangle$ iff it is a σ-extension w.r.t. $\mathcal{AF} = \langle A, \{(s(\alpha), t(\alpha)) | \alpha \in K\}\rangle$.*

Corollary 1. *For each semantics $\sigma \in \{$*complete, preferred, grounded, stable*$\}$: $\mathfrak{U} = \langle S, K\rangle$ is a σ-structure w.r.t. a non-recursive $\mathcal{RAF} = \langle A, K, s, t\rangle$ iff S is σ-extension w.r.t. $\mathcal{AF} = \langle A, \{(s(\alpha), t(\alpha)) | \alpha \in K\}\rangle$.*

Example 43. *Let $\mathcal{AF} = \langle A, K\rangle$ be the AF show in Figure 9.3 and $\mathcal{RAF}$ be the RAF of Example 41 on the previous page. For $\sigma \in \{$*complete, preferred, grounded, stable*$\}$, we can verify that: $\mathfrak{U} = \langle S, Q\rangle$ is a RAF-σ-structure of $\mathcal{RAF}$ iff S is σ-extension of $\mathcal{AF}$.*

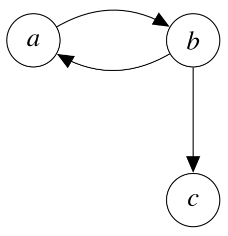
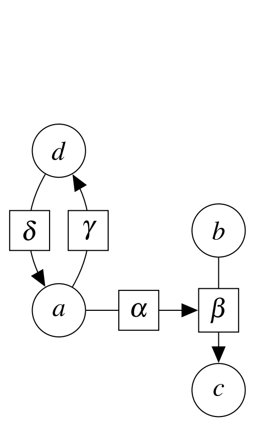


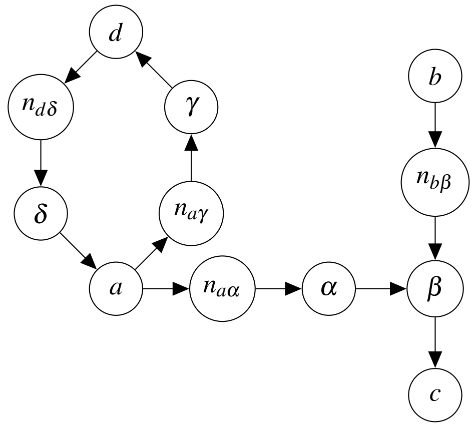
Figure 9.3: The AF corresponding to the non recursive framework of Figure 9.2 on the previous page

Theorem 3 and Corollary 1 prove that RAFs are a conservative generalization of AFs. Indeed there is a one-to-one correspondence between the structures of a RAF without recursive attacks and their corresponding Dung's extensions for the semantics *complete*, *preferred*, *grounded* and *stable*. Furthermore, RAFs conserve the notion conflict-freeness.

`Note:` *In Section 10.1 on page 102, we prove that there is also a one-to-one correspondence between the structures of a RAF without recursive attacks and their corresponding Dung's extensions for* semi-stable *semantics.*



(a) A RAF



(b) Corresponding MAF/AF

Figure 9.4: Example of RAF flattening with $\texttt{Raf2Af}_{\texttt{maf}}$

9.2.2 RAF flattening

In [18], a transformation has been already proposed in order to establish a link between RAF and MAF (Meta-Argumentation Framework). This flattening, inspired by the works presented in [46] and [16], transforms a RAF into MAF. Given that MAFs are basically AFs, a brief presentation of this flattening process is then relevant.

Definition 68 (RAF flattening to MAF/AF)**.** *Let* $\texttt{Raf2Af}_{\texttt{maf}} : \Phi_{raf} \to \Phi_{af}$ *be the function transforming a RAF into an AF.* $\texttt{Raf2Af}_{\texttt{maf}}$ *is defined as follows:*

$$\forall\, \mathcal{RAF} = \langle A, K, s, t\rangle \in \Phi_{raf},\ \texttt{Raf2Af}_{\texttt{maf}} : \mathcal{RAF} \mapsto \mathcal{MAF} = \langle A', K'\rangle$$

$$\begin{aligned} \textit{With: } A' &= A \cup K \cup N \\ N &= \{n_{s(\alpha)\alpha} | \alpha \in K\} \\ K' &= \{(s(\alpha), n_{s(\alpha)\alpha}) | \alpha \in K\} \cup \{(n_{s(\alpha)\alpha}, \alpha) | \alpha \in K\} \cup \{(\alpha, t(\alpha)) | \alpha \in K\} \end{aligned}$$

Example 44. *Let consider the RAF* $\mathcal{RAF}$ *illustrated in Figure 9.4(a). Figure 9.4(b) illustrates* $\mathcal{AF} = \texttt{Raf2Af}_{\texttt{maf}}(\mathcal{RAF})$*, the MAF/AF corresponding to* $\mathcal{RAF}$*.*

Definition 69 (RAF Structure to MAF/AF extension)**.** *Let* $\mathcal{RAF} = \langle A, K, s, t\rangle$ *be a RAF and* $\mathcal{U} = \langle S, Q\rangle$ *be a structure. We denote by* $\texttt{str2MafExt}(\mathcal{U})$ *the MAF/AF extension corresponding to* $\mathcal{U}$*. It is defined as:*

$$\texttt{str2MafExt}(\mathcal{U}) = S \cup \{\alpha \in Q | s(\alpha) \in S\} \cup \{n_{s(\alpha)\alpha} \in N \,|\, s(\alpha) \notin S \textit{ and } s(\alpha) \in \textit{RAF-Def}(\mathcal{U})\}$$

Theorem 4. *For each semantics* $\sigma \in \{\text{complete}, \text{stable}, \text{preferred}, \text{grounded}\}$ *and for any* $\mathcal{RAF} \in \Phi_{raf}$*, the* *function* `str2MafExt`$(\cdot)$ *is a one-to-one correspondence between the sets of all* σ*-structures of* $\mathcal{RAF}$ *and the set of all* σ*-extensions of* $\texttt{Raf2Af}_{\texttt{maf}}(\mathcal{RAF})$*.*

It is important to note that even thought there is a one-to-one correspondence between the sets of all σ-structures and the set of all σ-extensions, this correspondence does not guarantee that each acceptable element *w.r.t.* a structure will be acceptable *w.r.t.* its corresponding extension in the resulting MAF/AF.

Example 45. *As an illustration of that fact, we have following Example 44 on the previous page:*

- $\mathcal{U} = \langle\{b\}, \varnothing\rangle$ *being a RAF-*admissible *structure of* $\mathcal{RAF}$*. We have: RAF-Acc*$(\mathcal{U}) = \{\alpha, \delta, \gamma\}$*.* $\langle\{b\}, \{\alpha, \delta, \gamma\}\rangle$ *is thus a RAF-*complete *structure. By the way, it is also the RAF-*grounded *structure.*
- $\texttt{Raf2Af}_{\texttt{maf}}(\mathcal{U}) = \{b\}$ *is an* admissible *extension of* $\mathcal{MAF}$ *and it is also its* grounded *extension. We have: Acc*$(\{b\}) = \varnothing$*. As a consequence,* α *is not acceptable w.r.t.* $\{b\}$*.*

This remark will serve as motivation for a new flattening process. See Chapter 12 on page 114.

9.3 Relation between RAF and AFRA

In this section the relation between RAFs and AFs is briefly presented. In order to establish a point of comparison, we define a transformation from RAF structures to AFRA extensions:

Definition 70 (RAF Structure to AFRA extension [18])**.** *Let* $\mathcal{RAF} = \langle A, K, s, t\rangle$ *be a RAF and* $\mathcal{U} = \langle S, Q\rangle$ *be a structure. We denote by* `str2afraExt`$(\mathcal{U})$ *the AFRA extension corresponding to* $\mathcal{U}$*. It is defined as:*

$$\texttt{str2afraExt}(\mathcal{U}) = S \cup \{\alpha \in Q | s(\alpha) \in S\}$$

In [18], Proposition 22 and Theorem 5 have been proven.

Proposition 22. *Let* $\mathcal{RAF} = \langle A, K, s, t\rangle$ *be a RAF and* $\mathcal{U} = \langle S, Q\rangle$ *be some RAF-*conflict-free *(respectively RAF-*admissible *) structure. Then* `str2afraExt`$(\mathcal{U})$ *is AFRA-*conflict-free *(respectively AFRA-*admissible*).*

Theorem 5. *For each semantics* $\sigma \in \{\text{complete}, \text{stable}, \text{preferred}, \text{grounded}\}$*, the function* `str2afraExt`$(\cdot)$ *is a one-to-one correspondence between the sets of all* σ*-structures and the set of all AFRA-*σ*-extensions.*

Example 46. *Let consider the RAF/AFRA shown in Figure 8.2 on page 90. Following Examples 38 and 39 on page 87 and on page 92 and as shown by the result in Table 8.1 on page 88 and Table 9.1 on page 93, there is a one-to-one correspondence between the sets of all structures and the set of all AFRA-extensions for the all the* complete*-based semantics mentioned. We have:*

- S_1 *corresponding to* $\mathcal{U}_1$
- S_2 *corresponding to* $\mathcal{U}_2$
- S_3 *corresponding to* $\mathcal{U}_3$
- S_4 *corresponding to* $\mathcal{U}_4$

Part V

Higher-Order Attack Argumentation Frameworks: Contribution

Part presentation:

In this part is presented all the works about Recursive Argumentation Frameworks done during my thesis. The two main contributions of this part are the adaptation of the notion of AF labelling for RAF, so-called, *structure labellings*, and the study of *semantics decomposability* property of RAF semantics. These works lead to several publications:

- An IRIT report about RAF *structure labellings*, [34], which serves as support for subsequent works, providing details on concepts and the properties proofs that are not in articles.
- An IRIT report about RAF *Complexities*, [33], which serves to the same purpose.
- A poster in KR 2020, the 7th *International Conference on Principles of Knowledge Representation and Reasoning*, [36], about *structure labellings* and RAF semantics *complexities*.
- An article in ICTAI 2020, the 32th *International Conference on Tools with Artificial Intelligence* [35], about the *structure labellings* and RAF semantics *complexities*.
- A journal article that has been submitted and currently being reviewed in IJAIT, the *International Journal of Artificial Intelligence Tools*, gathering most of the works that have been done on RAFs.

Firstly, the notion of Dung *labellings* and the *semi-stable* semantics are extended to RAF. Secondly, a *Flattening* process that transforms RAFs into AFs, ensuring interesting properties, is introduced. Thirdly, relying on that *Flattening*, the *complexities* of RAF semantics is studied. Fourthly, the notion of *Strongly Connected Component* is adapted to RAFs and from this key notion, the *semantics decomposability* and the *directionality* of RAF semantics are studied. Finally, related works are presented.

Chapter 10

New semantics for RAF

In this chapter is introduced the *semi-stable* semantics for RAF, and a focus on RAFs with no recursive attacks is done. The notion of AF reinstatement labelling introduced in [14] is also generalized for RAF (so called "reinstatement RAF labelling").

`Note:` *The RAF illustrated in Figure 10.1 will serve as running example.*

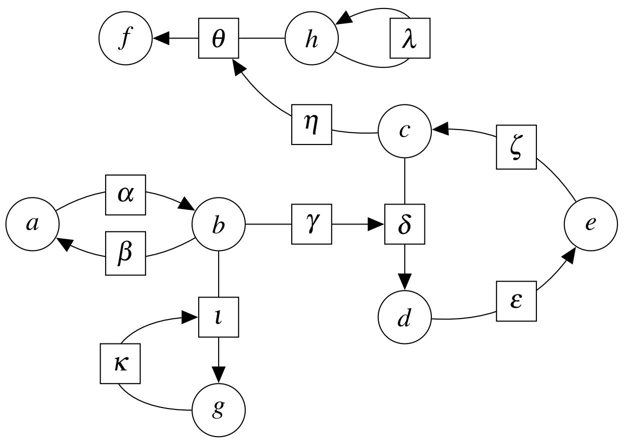


Figure 10.1: Running example

10.1 The Semi-stable semantics

While introducing labellings for AF and studying labellings over some constraints, Caminada et al. ([14]) highlighted a non yet discovered semantics: the *semi-stable* semantics. From the *semi-stable* labelling semantics has been defined the *semi-stable* extension semantics.

10.1.1 Definition and some properties

As for AF, we propose that *semi-stable* structures be the ones that decide the most on the acceptance or the rejection of arguments and attacks. They are formally defined as follows:

Definition 71. *(Semi-stable structure). Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{U} = \langle S, Q\rangle$ be some structure over it. $\mathcal{U}$ is said to be a* semi-stable *structure iff $\mathcal{U}$ is a* complete *structure such that:*

$$S \cup Q \cup RAF\text{-}Def(\mathcal{U}) \cup RAF\text{-}Inh(\mathcal{U}) \text{ is maximal w.r.t. to inclusion.}$$

Theorem 6. *The following assertions hold:*

1. *Every* stable *structure is a* semi-stable *structure*

2. *Every* semi-stable *structure is a* preferred *structure*

□ Proof of Theorem 6: link (See page 230).

Theorem 7. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF. If there exists a* stable *structure, then the* semi-stable *structures coincide with the* stable *structures.*

□ Proof of Theorem 7: link (See page 231).

Example 47. *The* complete, grounded, preferred, semi-stable, arg-preferred *and* stable *semantics corresponding to Figure 10.1 on the previous page are given in Table 10.1 on page 105. We can observe that the* stable *semantics produces no structure for that RAF. This example shows that a* semi-stable *structure is not always a* stable *one (see $\mathcal{U}_3$ and $\mathcal{U}_4$) and that a* preferred *structure is not always a* semi-stable *one (see $\mathcal{U}_2$).*

10.1.2 The case of RAF with no recursive attacks

As stated in Section 9.2 on page 94, it has been proven in [18] that in RAFs without recursive attacks there is a one-to-one correspondence between structures and Dung's extensions for the *complete*, *grounded*, *preferred* and *stable* semantics. Let now consider the case of the *semi-stable* semantics and show that the set of *semi-stable* extensions coincides with the set of *semi-stable* structures on RAF with no recursive attacks (*i.e.* RAF that happened to be simple AF). Notice that all the structures of such a RAF contain all the attacks.

Proposition 23 (*Semi-stable* extensions and structures)**.** *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF such that $\forall \alpha \in K$, $t(\alpha) \in A$. $\mathcal{RAF}$ can thus be considered as a simple AF. Let $\mathcal{AF} = \langle A, K\rangle$ be the AF version of $\mathcal{RAF}$.*

$$\mathcal{U} = \langle S, K\rangle \text{ is a semi-stable structure of } \mathcal{RAF} \text{ iff } S \text{ is a semi-stable extension of } \mathcal{AF}$$

□ Proof of Proposition 23: link (See page 231).

10.2 Reinstatement RAF labellings

Now that relations between structure semantics and between structure and extensions semantics have been stated, we introduce the notion of labelling on RAF.

The reason why we are interested in the labelling approach to compute semantics is that labellings are more precise than structures (as there are three statuses to describe the acceptance of elements) and especially because it seems to be more practical for finding algorithms.

Definition 72. *(RAF labelling). Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a recursive argumentation framework. A RAF labelling is a tuple $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ such that ℓ_A is a total function $\ell_A : A \to \{\mathtt{in}, \mathtt{out}, \mathtt{und}\}$ and ℓ_K, a total function $\ell_K : K \to \{\mathtt{in}, \mathtt{out}, \mathtt{und}\}$.*

We define:

- *$\mathtt{in}(\mathcal{L})$ as the tuple $\langle \{a \in A | \ell_A(a) = \mathtt{in}\}, \{\alpha \in K | \ell_K(\alpha) = \mathtt{in}\} \rangle$,*
- *$\mathtt{und}(\mathcal{L})$ as the tuple $\langle \{a \in A | \ell_A(a) = \mathtt{und}\}, \{\alpha \in K | \ell_K(\alpha) = \mathtt{und}\} \rangle$ and*
- *$\mathtt{out}(\mathcal{L})$ as the tuple $\langle \{a \in A | \ell_A(a) = \mathtt{out}\}, \{\alpha \in K | \ell_K(\alpha) = \mathtt{out}\} \rangle$.*

Let $x \in (A \cup K)$. Given a certain $\mathcal{L}$, we use the notation $\mathcal{L}(x)$ to indicate the labelling of x in $\mathcal{L}$. It could mean $\ell_A(x)$ or $\ell_K(x)$, following the nature of x.

Definition 73. *(Reinstatement RAF labelling). Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a recursive argumentation framework and $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ be a RAF labelling. $\mathcal{L}$ is a reinstatement RAF labelling iff it satisfies the following conditions: $\forall x \in (A \cup K)$,*

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

An equivalent definition of reinstatement RAF labelling can be made, as for AF, using the notion of "*legally labelled argument*". An $\mathtt{in}$-labelled element is said to be *legally $\mathtt{in}$ iff* all its attackers or their involved attacks are labelled $\mathtt{out}$. An $\mathtt{out}$-labelled element is said to be *legally $\mathtt{out}$ iff* at least one of its attackers and the involved attack are labelled $\mathtt{in}$. An $\mathtt{und}$-labelled element is said to be *legally $\mathtt{und}$ iff* it does not have any attacker and its involved attack that are labelled $\mathtt{in}$ and one of its attackers and the involved attack are not labelled $\mathtt{out}$. Formally, "*valid labellings*" (notion equivalent to reinstatement RAF labellings) are defined as follows:

Definition 74 (Legally labelled elements, valid RAF labelling)**.**
Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a recursive argumentation framework and $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ be a RAF labelling over $\mathcal{RAF}$. Let x be an argument or an attack of $\mathcal{RAF}$. x is said to be legally labelled *in $\mathcal{L}$ if and only if the 3 following conditions hold:*

- *$x \in \mathtt{in}(\mathcal{L})$ iff $(\forall \alpha \in K \text{ s.t. } t(\alpha) = x, \ell_K(\alpha) = \mathtt{out} \text{ or } \ell_A(s(\alpha)) = \mathtt{out})$*
- *$x \in \mathtt{out}(\mathcal{L})$ iff $(\exists \alpha \in K \text{ s.t. } t(\alpha) = x, \ell_K(\alpha) = \mathtt{in} \text{ and } \ell_A(s(\alpha)) = \mathtt{in})$*
- *$x \in \mathtt{und}(\mathcal{L})$ iff $((\nexists \alpha \in K \text{ s.t. } t(\alpha) = x, \ell_K(\alpha) = \mathtt{in} \text{ and } \ell_A(s(\alpha)) = \mathtt{in})$
  and $(\exists \alpha \in K \text{ s.t. } t(\alpha) = x, \ell_K(\alpha) \neq \mathtt{out} \text{ and } \ell_A(s(\alpha)) \neq \mathtt{out}))$*

$\mathcal{L}$ is said to be a valid RAF labelling *if all its elements are* legally labelled.

Example 48. *As an illustration, the labelling version of Table 10.1 on the following page about Figure 10.1 on page 102 is shown in Table 10.2 on page 106.*

		$\mathcal{U}_1$	$\mathcal{U}_2$	$\mathcal{U}_3$	$\mathcal{U}_4$
Arguments or attacks	a		✓		
	b			✓	✓
	c			✓	✓
	d			✓	✓
	e				
	f			✓	✓
	g		✓	✓	
	h				
	α	✓	✓	✓	✓
	β	✓	✓	✓	✓
	γ	✓	✓	✓	✓
	δ		✓		
	ε	✓	✓	✓	✓
	ζ	✓	✓	✓	✓
	η	✓	✓	✓	✓
	θ				
	ι				✓
	κ	✓	✓	✓	✓
	λ	✓	✓	✓	✓
Semantics with structures	*grounded*	✓			
	complete	✓	✓	✓	✓
	preferred		✓	✓	✓
	arg-preferred		✓	✓	
	semi-stable			✓	✓
	stable				

In the first part of the table, $i \checkmark j$ means that the element i belongs to the structure j.
In the second part of the table, $i \checkmark j$ means that j is a structure of the semantics i.

Table 10.1: Semantics structures

Arguments or attacks		$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{L}_4$
	a	*und*	*in*	*out*	*out*
	b	*und*	*out*	*in*	*in*
	c	*und*	*und*	*in*	*in*
	d	*und*	*und*	*in*	*in*
	e	*und*	*und*	*out*	*out*
	f	*und*	*und*	*in*	*in*
	g	*und*	*in*	*in*	*out*
	h	*und*	*und*	*und*	*und*
	α	*in*	*in*	*in*	*in*
	β	*in*	*in*	*in*	*in*
	γ	*in*	*in*	*in*	*in*
	δ	*und*	*in*	*out*	*out*
	ε	*in*	*in*	*in*	*in*
	ζ	*in*	*in*	*in*	*in*
	η	*in*	*in*	*in*	*in*
	θ	*und*	*und*	*out*	*out*
	ι	*und*	*out*	*out*	*in*
	κ	*in*	*in*	*in*	*in*
	λ	*in*	*in*	*in*	*in*

Table 10.2: RAF labellings

Chapter 11

Structure labellings and semantics

In this chapter we show that there exists a one-to-one mapping between RAF labellings and structures. Specific semantics structures happen to be coinciding with RAF labellings under some constraints. In order to prove it, we introduce the two following functions to go from structures to labellings and vice versa:

Definition 75. *(*`Struct2Lab` *and* `Lab2Struct`*). Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF, $\mathcal{U} = \langle S, Q \rangle$ be a structure and $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ be a RAF labelling. The functions* $\texttt{Struct2Lab}_{\mathcal{RAF}}$ *and* $\texttt{Lab2Struct}_{\mathcal{RAF}}$ *are defined as following:*

- $\texttt{Struct2Lab}_{\mathcal{RAF}}(\mathcal{U}) = \langle \ell_A, \ell_K \rangle$*, a RAF labelling with:*
  - $\ell_A = \{(a, \mathtt{in}) | a \in S\} \cup \{(a, \mathtt{out}) | a \in (A \setminus S)$ *and* $a \in \textit{RAF-Def}(\mathcal{U})\} \cup \{(a, \mathtt{und}) | a \in (A \setminus S)$ *and* $a \notin \textit{RAF-Def}(\mathcal{U})\}$
  - $\ell_K = \{(\alpha, \mathtt{in}) | \alpha \in Q\} \cup \{(\alpha, \mathtt{out}) | \alpha \in (K \setminus Q)$ *and* $\alpha \in \textit{RAF-Inh}(\mathcal{U})\} \cup \{(\alpha, \mathtt{und}) | \alpha \in (K \setminus Q)$ *and* $\alpha \notin \textit{RAF-Inh}(\mathcal{U})\}$
- $\texttt{Lab2Struct}_{\mathcal{RAF}}(\mathcal{L}) = \langle S, Q \rangle$*, a structure with:*
  - $S = \{a | \ell_A(a) = \mathtt{in}\}$
  - $Q = \{\alpha | \ell_K(\alpha) = \mathtt{in}\}$

We write `Struct2Lab` *and* `Lab2Struct` *instead of* $\texttt{Struct2Lab}_{\mathcal{RAF}}$ *and* $\texttt{Lab2Struct}_{\mathcal{RAF}}$ *when there is no ambiguity about the RAF $\mathcal{RAF}$ we refer to.*

11.1 Complete semantics

Reinstatement RAF labellings coincide with *complete* structures as stated by Theorems 8 and 9 on the current page and on the next page.

Theorem 8. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and let $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ be a reinstatement RAF labelling. Then* `Lab2Struct`$(\mathcal{L})$ *is a* complete *structure.*

☐ Proof of Theorem 8: link (See page 232).

Theorem 9. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and let $\mathcal{U} = \langle S, Q \rangle$ be a complete structure. Then* $\texttt{Struct2Lab}(\mathcal{U})$ *is a reinstatement RAF labelling.*

□ Proof of Theorem 9: link (See page 233).

11.2 Preferred semantics

In this section we show that several constraints on reinstatement RAF labellings lead to the *preferred* semantics.

11.2.1 Reinstatement RAF labellings with maximal *in*

Reinstatement RAF labellings such that $in(\mathcal{L})$ is maximal coincide with *preferred* structures as stated by Theorems 10 and 11.

Theorem 10. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and let $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ be a reinstatement RAF labelling such that $in(\mathcal{L})$ is maximal. Then* $\texttt{Lab2Struct}(\mathcal{L})$ *is a* preferred *structure.*

□ Proof of Theorem 10: link (See page 234).

Theorem 11. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and let $\mathcal{U} = \langle S, Q \rangle$ be a* preferred *structure. Then $\mathcal{L} =$* $\texttt{Struct2Lab}(\mathcal{U})$ *is a reinstatement RAF labelling such that $in(\mathcal{L})$ is maximal.*

□ Proof of Theorem 11: link (See page 234).

11.2.2 Reinstatement RAF labellings with maximal *out*

Reinstatement RAF labellings such that $out(\mathcal{L})$ is maximal also coincide with *preferred* structures.

Theorem 12. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and let $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ be a reinstatement RAF labelling such that $out(\mathcal{L})$ is maximal. Then* $\texttt{Lab2Struct}(\mathcal{L})$ *is a* preferred *structure.*

□ Proof of Theorem 12: link (See page 236).

Theorem 13. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and let $\mathcal{U} = \langle S, Q \rangle$ be a* preferred *structure. Then $\mathcal{L} =$* $\texttt{Struct2Lab}(\mathcal{U})$ *is a reinstatement RAF labelling such that $out(\mathcal{L})$ is maximal.*

□ Proof of Theorem 13: link (See page 236).

11.3 Stable semantics: reinstatement RAF labellings with empty *und*

Reinstatement RAF labellings such that $und(\mathcal{L})$ is empty coincide with *stable* structures as stated by Theorems 14 and 15 on the following page.

Theorem 14. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and let $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ be a reinstatement RAF labelling such that $\mathtt{und}(\mathcal{L}) = \varnothing$. Then* `Lab2Struct`$(\mathcal{L})$ *is a* stable *structure.*

□ Proof of Theorem 14: link (See page 236).

Theorem 15. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and let $\mathcal{U} = \langle S, Q \rangle$ be a* stable *structure. Then $\mathcal{L} =$* `Struct2Lab`$(\mathcal{U})$ *is a reinstatement RAF labelling such that $\mathtt{und}(\mathcal{L})$ is empty.*

□ Proof of Theorem 15: link (See page 236).

11.4 Grounded semantics

In this section we show that several constraints on reinstatement RAF labellings lead to the *grounded* semantics.

11.4.1 Reinstatement RAF labellings with maximal *und*

Reinstatement RAF labellings such that $\mathtt{und}(\mathcal{L})$ is maximal coincide with the *grounded* structure as stated by Theorems 16 and 17.

Theorem 16. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and let $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ be a reinstatement RAF labelling such that $\mathtt{und}(\mathcal{L})$ is maximal. Then* `Lab2Struct`$(\mathcal{L})$ *is the* grounded *structure.*

□ Proof of Theorem 16: link (See page 237).

Theorem 17. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and let $\mathcal{U} = \langle S, Q \rangle$ be the* grounded *structure. Then $\mathcal{L} =$* `Struct2Lab`$(\mathcal{U})$ *is a reinstatement RAF labelling such that $\mathtt{und}(\mathcal{L})$ is maximal.*

□ Proof of Theorem 17: link (See page 237).

11.4.2 Reinstatement RAF labellings with minimal *in*

Reinstatement RAF labellings such that $\mathtt{in}(\mathcal{L})$ is minimal coincide with the *grounded* structure as stated by Theorems 18 and 19.

Theorem 18. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and let $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ be a reinstatement RAF labelling such that $\mathtt{in}(\mathcal{L})$ is minimal. Then* `Lab2Struct`$(\mathcal{L})$ *is the* grounded *structure.*

□ Proof of Theorem 18: link (See page 237).

Theorem 19. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and let $\mathcal{U} = \langle S, Q \rangle$ be the* grounded *structure. Then $\mathcal{L} =$* `Struct2Lab`$(\mathcal{U})$ *is a reinstatement RAF labelling such that $\mathtt{in}(\mathcal{L})$ is minimal.*

□ Proof of Theorem 19: link (See page 237).

11.4.3 Reinstatement RAF labellings with minimal `out`

Note that reinstatement RAF labellings such that $\mathtt{out}(\mathcal{L})$ is minimal also coincide with the *grounded* structure as stated by Theorems 20 and 21.

Theorem 20. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and let $\mathcal{L} = \langle \ell_A, \ell_K\rangle$ be a reinstatement RAF labelling such that $\mathtt{out}(\mathcal{L})$ is minimal. Then* $\mathtt{Lab2Struct}(\mathcal{L})$ *is the* grounded *structure.*

□ Proof of Theorem 20: link (See page 237).

Theorem 21. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and let $\mathcal{U} = \langle S, Q\rangle$ be the* grounded *structure. Then $\mathcal{L} =$ $\mathtt{Struct2Lab}(\mathcal{U})$ is a reinstatement RAF labelling such that $\mathtt{out}(\mathcal{L})$ is minimal.*

□ Proof of Theorem 21: link (See page 237).

11.5 Semi-stable semantics

Reinstatement RAF labellings such that $\mathtt{und}(\mathcal{L})$ is minimal coincide with *semi-stable* structures as stated by Theorems 22 and 23.

Theorem 22. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and let $\mathcal{L} = \langle \ell_A, \ell_K\rangle$ be a reinstatement RAF labelling such that $\mathtt{und}(\mathcal{L})$ is minimal. Then* $\mathtt{Lab2Struct}(\mathcal{L})$ *is a* semi-stable *structure.*

□ Proof of Theorem 22: link (See page 238).

Theorem 23. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and let $\mathcal{U} = \langle S, Q\rangle$ be a* semi-stable *structure. Then $\mathcal{L} =$ $\mathtt{Struct2Lab}(\mathcal{U})$ is a reinstatement RAF labelling such that $\mathtt{und}(\mathcal{L})$ is minimal.*

□ Proof of Theorem 23: link (See page 238).

11.6 A one-to-one mapping

In this section are summarized the relations between labellings and structures in RAF and are also presented the links between labellings in AF and RAF with no recursive attacks.

11.6.1 Structures and labellings in RAF

Table 11.1 on the following page sums up the whole previous sections of Chapter 11. It shows the correspondence between structure semantics and reinstatement RAF labellings.

Example 49. *Following Example 48 on page 104 and the different properties proven in this chapter, the semantics labellings of the RAF illustrated in Figure 11.1 on page 112 are given in Table 11.2 on page 113.*

Restriction on Reinstatement RAF labelling	Semantics	Theorems
no restrictions	*complete* semantics	Theorems 8 and 9
empty $\mathtt{und}$	*stable* semantics	Theorems 14 and 15
maximal $\mathtt{in}$	*preferred* semantics	Theorems 10 and 11
maximal $\mathtt{out}$	*preferred* semantics	Theorems 12 and 13
maximal $\mathtt{und}$	*grounded* semantics	Theorems 16 and 17
minimal $\mathtt{in}$	*grounded* semantics	Theorems 18 and 19
minimal $\mathtt{out}$	*grounded* semantics	Theorems 20 and 21
minimal $\mathtt{und}$	*semi-stable* semantics	Theorems 22 and 23

Table 11.1: Reinstatement RAF labellings and structures semantics

11.6.2 AF labellings and RAF labellings when no recursive attack exists

As stated in Section 10.1.2 on page 103, there exists a one-to-one mapping between structures and extensions in RAF without recursive attacks for the *complete*, *grounded*, *preferred*, *semi-stable* and *stable* semantics.

[14] established a one-to-one mapping between AF extensions and AF reinstatement labellings for the mentioned semantics. In this thesis, as summarized in Section 11.6.1 on the previous page, is established a one-to-one mapping between RAF structures and reinstatement RAF labellings for the same semantics.

As a consequence, for RAF with no recursive attacks, there exists obviously a one-to-one mapping between reinstatement labellings (AF notion) and structures (RAF notion) and also between reinstatement labellings (AF notion) and reinstatement RAF labellings (RAF notion).

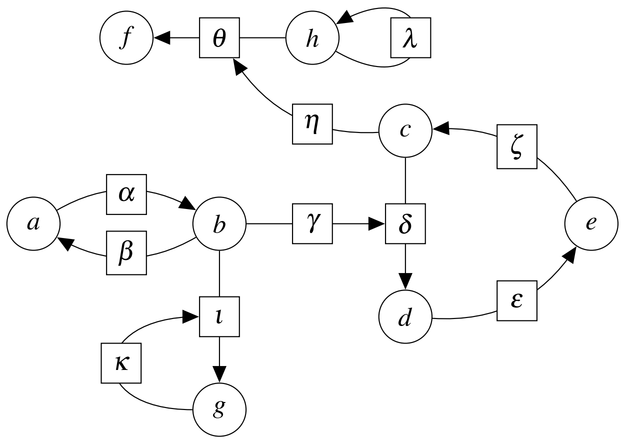


Figure 11.1: Running example

		$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{L}_4$
Arguments or attacks	a	*und*	*in*	*out*	*out*
	b	*und*	*out*	*in*	*in*
	c	*und*	*und*	*in*	*in*
	d	*und*	*und*	*in*	*in*
	e	*und*	*und*	*out*	*out*
	f	*und*	*und*	*in*	*in*
	g	*und*	*in*	*in*	*out*
	h	*und*	*und*	*und*	*und*
	α	*in*	*in*	*in*	*in*
	β	*in*	*in*	*in*	*in*
	γ	*in*	*in*	*in*	*in*
	δ	*und*	*in*	*out*	*out*
	ε	*in*	*in*	*in*	*in*
	ζ	*in*	*in*	*in*	*in*
	η	*in*	*in*	*in*	*in*
	θ	*und*	*und*	*out*	*out*
	ι	*und*	*out*	*out*	*in*
	κ	*in*	*in*	*in*	*in*
	λ	*in*	*in*	*in*	*in*
Semantics with RAF labellings	*grounded*	✓			
	complete	✓	✓	✓	✓
	preferred		✓	✓	✓
	arg-preferred		✓	✓	
	semi-stable			✓	✓
	stable				

$i \checkmark j$ means that j is a labelling of semantics i.

Table 11.2: RAF Semantics labellings

Chapter 12

RAF flattening

In this chapter is presented a new RAF flattening process to convert RAF into AF. We first discuss the interest of such process (Section 12.1). We, then, define it and illustrate it (Section 12.2 on the following page). Finally, some semantics properties are studied (Section 12.3 on page 116).

12.1 Motivation

What motivated the search for a flattening process that transforms RAFs into AFs is that such a tool could probably opened up perspectives for the study of RAF properties. Indeed, Dung's Argumentation Theory has been explored for decades now. A lot of properties on AFs and semantics have been defined and studied. Having a way to transform RAFs into AFs could then probably help for the extension of notions and properties defined for AF to RAF. We can think, as examples, of AF shape properties (SCC, autonomous fragments [7]), semantics properties (complexity, SCC-recursiveness [11], semantics decomposability [8]). Furthermore, such a transformation could be used for computational purposes given that there is a wide range of AF solvers nowadays.

This being said, it important to know that such a transformation has been already proposed in [18], as seen in Section 9.2.2 on page 98. Even though this transformation proves a one-to-one semantics correspondence between RAF and MAF (and also between RAF and AF, since MAF are AF), it is not enough for proving properties as semantics complexity. Indeed, as explained in Example 45 on page 99, this correspondences do not guarantee that each acceptable element *w.r.t.* a structure will be acceptable *w.r.t.* its corresponding extension in the resulting MAF/AF.

`Note:` *Such a transformation has also been proposed in [6], as seen in Section 8.2 on page 90, to transform AFRA into AF. As the previous mentioned one, this transformation ensures a one-to-one semantics correspondence between AFRA and AF, but not between acceptable elements.*

Moreover, in our sense, this transformation presents a counter-intuitive meaning of the link between an attack and its source. Indeed each attack is related to its source by a sequence of attacks meaning that the attack is "defended" by its source. So, every attack is always attacked in the resulting MAF even if it is not the target of an attack in the initial RAF. This has also a potential impact on properties related to the shape of the AF.

Let now introduce a new flattening process for RAF that fixes those two problems.

12.2 A new flattening process

Introduced in [35], the new flattening process is formally defined as follows:

Definition 76. *Let* `Raf2Af` $: \Phi_{raf} \rightarrow \Phi_{af}$ *be the function transforming a RAF into an AF.* `Raf2Af` *is defined as follows:*

$$\forall\, \mathcal{RAF} = \langle A, K, s, t\rangle \in \Phi_{raf},\ \texttt{Raf2Af} : \mathcal{RAF} \mapsto \mathcal{AF} = \langle A', K'\rangle$$

$$\begin{aligned}
\textit{With: } A' &= A \cup K \cup Not_A \cup Not_K \cup And_{A,K}\\
K' &= K'_1 \cup K'_2 \cup K'_3 \cup K'_4 \cup K'_5\\
Not_A &= \{\neg a | a \in A\}\\
Not_K &= \{\neg\beta | \beta \in K\}\\
And_{A,K} &= \{a.\beta | \beta \in K, a = s(\beta)\}\\
K'_1 &= \{(a, \neg a) | a \in A\}\\
K'_2 &= \{(\beta, \neg\beta) | \beta \in K\}\\
K'_3 &= \{(\neg a, a.\beta) | a \in A, s(\beta) = a\}\\
K'_4 &= \{(\neg\beta, a.\beta) | \beta \in K, s(\beta) = a\}\\
K'_5 &= \{(a.\beta, t(\beta)) | \beta \in K, s(\beta) = a\}
\end{aligned}$$

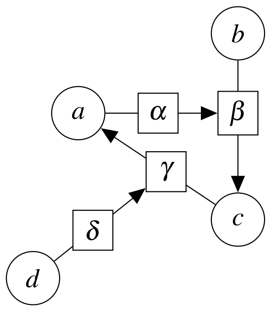
`Note:` *"$\neg a$", "$\neg\beta$" and "$a.\beta$" are just simple argument names that represent respectively, the "negation" of argument a, the "negation" of attack β and the "conjunction" of attack β with its source a.*[1]

This transformation represents, with AFs, the semantics of RAF defeat relation, by mean of additional arguments. Let a be an argument attacking an element β through the attack α in the RAF $\mathcal{RAF}$. Given that to β be defeated by a, α must be valid (non-inhibited) and a accepted (not defeated), we represent this by creating an additional argument named "$a.\alpha$" accepted in $\mathcal{AF}$ only when both a and α are. To do that we create two others arguments named "$\neg a$" and "$\neg\alpha$". We create an attack going from a to $\neg a$, another going from α to $\neg\alpha$, two others going from $\neg a$ to $a.\alpha$ and $\neg\alpha$ to $a.\alpha$, and finally a last one going from $a.\alpha$ to β. An argument (corresponding to an element of the original RAF) is thus defeated in the resulting AF if and only if there exists a valid attack in the original RAF that targets this argument and whose source is accepted.

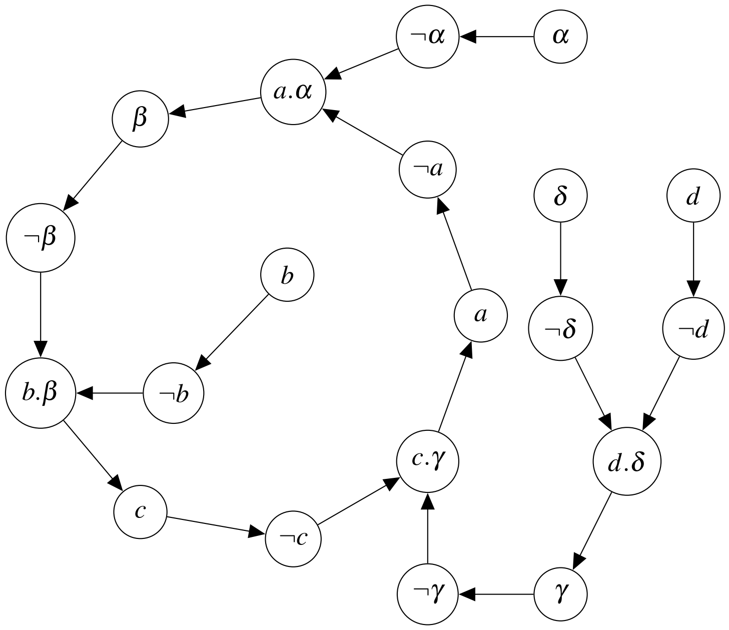
Example 50. *Let consider Figure 12.1 on the next page. Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be the RAF represented in Figure 12.1(a). We have $\mathcal{AF} =$* `Raf2Af`*$(\mathcal{RAF})$ being the AF represented in Figure 12.1(b).*

`Note:` *Interesting properties are formally proven in Section 12.3 and Chapters 13 and 14 on the next page, on page 119 and on page 123. Until then we can already notice that the shape structure is preserved: all and only elements that are attacked in the original RAF are attacked in the flattened version. Moreover, the correspondence between structure and extension seems intuitive. Let $S = \{a, b, c, d, \alpha, \delta, \neg\beta, \neg\gamma, d.\delta, a.\alpha\}$. S is a* complete *extension of $\mathcal{AF}$. We have the intuition that removing the created arguments during the flattening and putting arguments and attacks apart, lead to a RAF-*complete *structure of $\mathcal{RAF}$: $\mathfrak{U} = \langle\{a, b, c, d\}, \{\alpha, \delta\}\rangle$ (that is indeed RAF-*complete*).*

[1]The words "negation" and "conjunction" are used only by abuse of language and not in a logic meaning (even if there is a link, see [19]).



(a) A RAF



(b) Flattened version of the RAF in Figure 14.4(a) on page 128

Figure 12.1: RAF flattening illustration

12.3 Properties

In order to study the semantics properties, we defined what is an extension (of the RAF flattened version) corresponding to a structure (of the initial RAF).

Definition 77 (Extension corresponding to structure)**.** *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be an AF (with $\mathcal{AF} = \langle A', K'\rangle$). Let $\mathcal{U} = \langle S, Q\rangle$ be a structure in $\mathcal{RAF}$. We denote by "$\varepsilon_{\mathcal{U}}$" the extension in $\mathcal{AF}$ corresponding to a structure $\mathcal{U}$, defined by:*

$$\begin{aligned}\varepsilon_{\mathcal{U}} = S \cup Q \cup &\{\neg a \in Not_A | a \in \textit{RAF-Def}(\mathcal{U})\}\\ \cup &\{\neg\beta \in Not_K | \beta \in \textit{RAF-Inh}(\mathcal{U})\}\\ \cup &\{s(\beta).\beta \in And_{A,K} | \beta \in Q, s(\beta) \in S\}\end{aligned}$$

The next proposition establishes the link between the RAF-Def and RAF-Inh relations of the original RAF with Def relation in the AF, and the link between the RAF-Acc relation in the RAF with the Acc relation in the AF.

Proposition 24. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be an AF (with $\mathcal{AF} = \langle A', K'\rangle$). Let $\mathcal{U} = \langle S, Q\rangle$ be a structure in $\mathcal{RAF}$. The following properties holds:*

1. $\textit{RAF-Def}(\mathcal{U}) \cup \textit{RAF-Inh}(\mathcal{U}) =$

$$Def(\varepsilon_{\mathcal{U}}) \setminus \left(\begin{array}{l} \{\neg a \in Not_A | a \in \varepsilon_{\mathcal{U}}\} \\ \cup\{\neg\beta \in Not_K | \beta \in \varepsilon_{\mathcal{U}}\} \\ \cup\{s(\beta).\beta \in And_{A,K} | \beta \in Def(\varepsilon_{\mathcal{U}}) \text{ or } s(\beta) \in Def(\varepsilon_{\mathcal{U}})\} \end{array} \right)$$

2. $RAF\text{-}Acc(\mathcal{U}) = Acc(\varepsilon_{\mathcal{U}}) \setminus \left(\begin{array}{l} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{\neg\beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \end{array} \right)$

☐ Proof of Proposition 24: link (See page 238).

From the previous proposition, we can state the following semantics correspondence:

Proposition 25. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} =$ `Raf2Af`$(\mathcal{RAF})$ be an AF (with $\mathcal{AF} = \langle A', K'\rangle$). The following properties holds:*

1. *$\mathcal{U} = \langle S, Q\rangle$ is a RAF-*complete *structure in $\mathcal{RAF}$ iff $\varepsilon_{\mathcal{U}}$ is a* complete *extension in $\mathcal{AF}$.*
2. *$\mathcal{U} = \langle S, Q\rangle$ is a RAF-*grounded *structure in $\mathcal{RAF}$ iff $\varepsilon_{\mathcal{U}}$ is a* grounded *extension in $\mathcal{AF}$.*
3. *$\mathcal{U} = \langle S, Q\rangle$ is a RAF-*preferred *structure in $\mathcal{RAF}$ iff $\varepsilon_{\mathcal{U}}$ is a* preferred *extension in $\mathcal{AF}$.*
4. *$\mathcal{U} = \langle S, Q\rangle$ is a RAF-*stable *structure in $\mathcal{RAF}$ iff $\varepsilon_{\mathcal{U}}$ is a* stable *extension in $\mathcal{AF}$.*
5. *$\mathcal{U} = \langle S, Q\rangle$ is a RAF-*semi-stable *structure in $\mathcal{RAF}$ iff $\varepsilon_{\mathcal{U}}$ is a* semi-stable *extension in $\mathcal{AF}$.*

☐ Proof of Proposition 25: link (See page 242).

The study of the semantics properties can also be done the other way around. Let define what is a structure (of the initial RAF) corresponding to an extension (of the RAF flattened version).

Definition 78 (RAF labelling and AF labelling)**.** *Let σ be a semantics. Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} =$ `Raf2Af`$(\mathcal{RAF})$ be an AF (with $\mathcal{AF} = \langle A', K'\rangle$).*

The function `rafLab2AfLab` $: \mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF}) \to \mathscr{L}_{\sigma}(\mathcal{AF})$*, which maps to each structure labelling of $\mathcal{RAF}$ an AF labelling of $\mathcal{AF}$, is defined as following. Let $\mathcal{L}$ be a structure labelling of $\mathcal{RAF}$ and let $\ell =$* `rafLab2AfLab`$(\mathcal{L})$*. We have:*

- $\forall x \in (A \cup K)$*:*
  - $\mathcal{L}(x) = \ell(x)$
  - $\ell(\neg x) = \mathtt{und} \iff \mathcal{L}(x) = \mathtt{und}$
  - $\ell(\neg x) = \mathtt{in} \iff \mathcal{L}(x) = \mathtt{out}$
  - $\ell(\neg x) = \mathtt{out} \iff \mathcal{L}(x) = \mathtt{in}$
- $\forall \alpha \in K$*:*
  - $\ell(s(\alpha).\alpha) = \mathtt{in} \iff (\mathcal{L}(s(\alpha)) = \mathtt{in}\ \text{and}\ \mathcal{L}(\alpha) = \mathtt{in})$

- $\ell(s(\alpha).\alpha) = \mathtt{out} \iff (\mathcal{L}(s(\alpha)) = \mathtt{out}$ *or* $\mathcal{L}(\alpha) = \mathtt{out})$
- $\ell(s(\alpha).\alpha) = \mathtt{und} \iff (\mathcal{L}(s(\alpha)) \neq \mathtt{out}$ *and* $\mathcal{L}(\alpha) \neq \mathtt{out}$ *and* $(\mathcal{L}(s(\alpha)) = \mathtt{und}$ *or* $\mathcal{L}(\alpha) = \mathtt{und})$

The function `afLab2RafLab` $: \mathscr{L}_\sigma(\mathcal{AF}) \to \mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF})$*, which maps to each AF labelling of* $\mathcal{AF}$ *a structure labelling of* $\mathcal{RAF}$*, is defined as following. Let* ℓ *be a labelling of* $\mathcal{AF}$ *and let* $\mathcal{L} = \mathtt{afLab2RafLab}(\ell)$*. We have:*

$$\mathcal{L} = \left\langle \ell\downarrow_A, \ell\downarrow_K \right\rangle$$

The following propositions are trivially induced by Proposition 25 on the previous page.

Proposition 26. *Let* $\mathcal{RAF} = \langle A, K, s, t\rangle$ *be a RAF and* $\mathcal{AF} = \mathtt{Raf2Af}(\mathcal{RAF})$ *be an AF (with* $\mathcal{AF} = \langle A', K'\rangle$*). The following property holds:* $\mathcal{L} = \langle \ell_A, \ell_K\rangle$ *is a RAF-*complete *(resp. RAF-*grounded*, RAF-*preferred*, RAF-*stable *and RAF-*semi-stable*) structure labelling of* $\mathcal{RAF}$ *iff* $\ell = \mathtt{rafLab2AfLab}(\mathcal{L})$ *is a* complete *(resp.* grounded, preferred, stable *and* semi-stable*) labelling of* $\mathcal{AF}$*.*

Proposition 27. *Let* $\mathcal{RAF} = \langle A, K, s, t\rangle$ *be a RAF and* $\mathcal{AF} = \mathtt{Raf2Af}(\mathcal{RAF})$ *be an AF. The following property holds:* ℓ *is a* complete *(resp.* grounded, preferred, stable *and* semi-stable*) labelling of* $\mathcal{AF}$ *iff* $\mathcal{L} = \mathtt{afLab2RafLab}(\ell)$ *is a RAF-*complete *(resp. RAF-*grounded*, RAF-*preferred*, RAF-*stable *and RAF-*semi-stable*) structure labelling of* $\mathcal{RAF}$*.*

Chapter 13

RAF Decision Problems and semantics complexities

13.1 RAF Decision problems

Although in [18], results about RAF semantics complexities are given for the *RAF-Cred*$_\sigma$ problem (for the RAF-*complete*, RAF-*preferred* and RAF-*stable* semantics) and for the *RAF-Skep*$_\sigma$ problem (for the RAF-*preferred* and RAF-*stable* semantics), the decisions problems have not been explicitly defined nor the detailed proofs for the result given. In Definition 79, they are formally defined and in Section 13.2 on the next page their complexities are studied.

Definition 79 (Decision Problems in RAF)**.**

- Credulous Acceptance *Cred*$_\sigma$*: Given an RAF* $\mathcal{RAF} = \langle A, K, s, t\rangle$ *and an element* $x \in A \cup K$*. Is x contained in some* $\mathcal{U} \in \sigma(\mathcal{RAF})$*?*

- Skeptical Acceptance *Skep*$_\sigma$*: Given an AF* $\mathcal{RAF} = \langle A, K, s, t\rangle$ *and an element* $x \in A \cup K$*. Is x contained in each* $\mathcal{U} \in \sigma(\mathcal{RAF})$*?*

- Verification of a structure *Ver*$_\sigma$*: Given an RAF* $\mathcal{RAF} = \langle A, K, s, t\rangle$ *and a structure* $\mathcal{U}$*. Is* $\mathcal{U} \in \sigma(\mathcal{RAF})$*?*

- Existence of a structure *Exists*$_\sigma$*: Given an RAF* $\mathcal{RAF} = \langle A, K, s, t\rangle$*. Is* $\sigma(\mathcal{RAF}) \neq \varnothing$*?*

- Existence of a non-empty structure *Exists*$_\sigma^{\neg\varnothing}$*: Given an RAF* $\mathcal{RAF} = \langle A, K, s, t\rangle$*. Does there exist a structure* $\mathcal{U} \neq \varnothing$ *such that* $\mathcal{U} \in \sigma(\mathcal{RAF})$*?*

- Uniqueness of a solution *Unique*$_\sigma$*: Given an RAF* $\mathcal{RAF} = \langle A, K, s, t\rangle$*. Is there a unique structure* $\mathcal{U} \in \sigma(\mathcal{RAF})$*, i.e.* $\sigma(\mathcal{RAF}) = \{\mathcal{U}\}$*?*

Example 51. *Let* σ *be the* preferred *semantics. Let* $\mathcal{RAF} = \langle A, K, s, t\rangle$ *be the RAF represented in Figure 13.1 on the next page and* $\mathcal{U}$ *be any structure of it. Following Table 9.1 on page 93, we have three* preferred *structures:*

$$\sigma(\mathcal{RAF}) = \left\{ \begin{array}{c} \mathcal{U}_2 = \{a, g, \alpha, \beta, \gamma, \delta, \varepsilon, \zeta, \eta, \kappa, \lambda\} \\ \mathcal{U}_3 = \{b, c, d, f, g, \alpha, \beta, \gamma, \varepsilon, \zeta, \eta, \kappa, \lambda\} \\ \mathcal{U}_4 = \{b, c, d, f, \alpha, \beta, \gamma, \varepsilon, \zeta, \eta, \iota, \kappa, \lambda\} \end{array} \right\}$$

As a consequence, we have so:

- $\textit{RAF-Cred}_\sigma(\mathcal{RAF}, e) = \mathtt{false}$
- $\textit{RAF-Cred}_\sigma(\mathcal{RAF}, a) = \mathtt{true}$
- $\textit{RAF-Skep}_\sigma(\mathcal{RAF}, b) = \mathtt{false}$
- $\textit{RAF-Skep}_\sigma(\mathcal{RAF}, \alpha) = \mathtt{true}$
- $\textit{RAF-Ver}_\sigma(\mathcal{RAF}, \mathcal{U}) = \mathtt{true}$ *iff* $\mathcal{U} \in \{\mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4\}$
- $\textit{RAF-Exists}_\sigma(\mathcal{RAF}) = \mathtt{true}$
- $\textit{RAF-Exists}_\sigma^{\neg\varnothing}(\mathcal{RAF}) = \mathtt{true}$
- $\textit{RAF-Unique}_\sigma(\mathcal{RAF}) = \mathtt{false}$

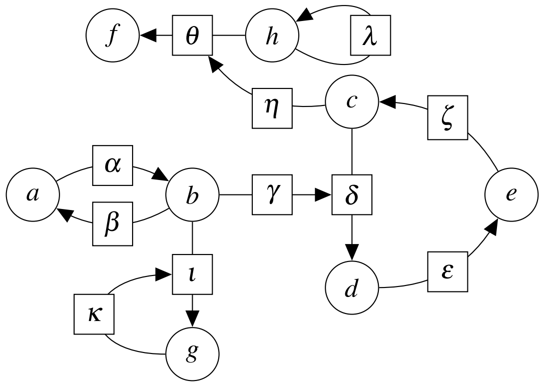


Figure 13.1: Same RAF as Figure 10.1 on page 102

13.2 RAF Semantics Complexities

In Section 9.2 on page 94 we saw that in terms of semantics, RAF is a generalization of Dung's AF. Knowing this, we searched for a polynomial reduction to transform RAF decision problems into AF ones and vice versa. Thereby, we could compare RAF decision problems complexities to the known AF complexities and potentially identify precisely the complexity classes of RAF decision problems.

As seen in Section 12.2 on page 115, we defined a function, `Raf2Af`, that transforms RAF in into AF. Let define a function, `Af2Raf`, that does the opposite.

Definition 80. *Let* `Af2Raf` $: \Phi_{af} \rightarrow \Phi_{raf}$ *be the function transforming an AF into an RAF.* `Af2Raf` *is defined as following:*

$$\forall\, \mathcal{AF} = \langle A, K \rangle \in \Phi_{af},\ \mathtt{Af2Raf} : \mathcal{AF} \mapsto \mathcal{RAF} = \left\langle A', K', s, t \right\rangle$$

With:

- $A' = A$
- $K' = \{\alpha | \alpha = (a,b) \in K\}$.[1]
- $\forall \alpha = (a,b) \in K$, $t(\alpha) = b$ *and* $s(\alpha) = a$

The definition of `Af2Raf` that transforms an AF into an RAF is trivial since an RAF without higher-order attacks is an AF. So it is enough to name the attacks of the AF in order to obtain a RAF. Note that, following the previous definition, no attack can be inhibited (as none of them is a target) in the RAF obtained by `Af2Raf`.

Both `Raf2Af` and `Af2Raf` are polynomial time and log-space functions. Let thus take advantage of these properties and prove that AF decision problems can be reduced into RAF ones and vice versa.

Proposition 28. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be an RAF and $\mathcal{AF} =$ `Raf2Af`$(\mathcal{RAF})$ be an AF (with $\mathcal{AF} = \langle A', K' \rangle$). Let $a \in (A \cup K)$ be an element in $\mathcal{RAF}$ and an argument in $\mathcal{AF}$, following the definition of* `Raf2Af`. *Let $\mathcal{U} = \langle S, Q \rangle$ be a structure of $\mathcal{RAF}$.*

*For each semantics $\sigma \in \{$*complete, semi-stable, stable, preferred, grounded*$\}$, we have:*

1. *RAF-Cred$_\sigma$ accepts $(\mathcal{RAF}, a)$ iff Cred$_\sigma$ accepts $(\mathcal{AF}, a)$.*
2. *RAF-Skep$_\sigma$ accepts $(\mathcal{RAF}, a)$ iff Skep$_\sigma$ accepts $(\mathcal{AF}, a)$.*
3. *RAF-Ver$_\sigma$ accepts $(\mathcal{RAF}, \mathcal{U})$ iff Ver$_\sigma$ accepts $(\mathcal{AF}, \varepsilon_{\mathcal{U}})$.*
4. *RAF-Exists$_\sigma$ accepts $\mathcal{RAF}$ iff Exists$_\sigma$ accepts $\mathcal{AF}$.*
5. *RAF-Exists$_\sigma^{\neg\varnothing}$ accepts $\mathcal{RAF}$ iff Exists$_\sigma^{\neg\varnothing}$ accepts $\mathcal{AF}$.*
6. *RAF-Unique$_\sigma$ accepts $\mathcal{RAF}$ iff Unique$_\sigma$ accepts $\mathcal{AF}$.*

□ Proof of Proposition 28: link (See page 251).

Given the previous property and given that `Raf2Af` is a polynomial time and log-space function, we can assert that:

Proposition 29. *The complexities of AF decision problems are at least as hard as RAF ones, for the semantics* complete, semi-stable, stable, preferred, grounded.

□ Proof of Proposition 29: link (See page 252).

Let now do the other way around process.

Proposition 30. *Let $\mathcal{AF} = \langle A, K \rangle$ be an AF and $\mathcal{RAF} =$ `Af2Raf`$(\mathcal{AF})$ be an RAF. Let $a \in A$ be an argument in $\mathcal{AF}$ and in $\mathcal{RAF}$, following the definition of* `Af2Raf`. *For each semantics $\sigma \in \{$*complete, semi-stable, stable, preferred, grounded*$\}$, we have:*

1. *Cred$_\sigma$ accepts $(\mathcal{AF}, a)$ iff RAF-Cred$_\sigma$ accepts $(\mathcal{RAF}, a)$.*
2. *Skep$_\sigma$ accepts $(\mathcal{AF}, a)$ iff RAF-Skep$_\sigma$ accepts $(\mathcal{RAF}, a)$.*

[1]This means that K' received the names of the attacks that are in K.

3. *Ver_σ accepts $(\mathcal{AF}, S)$ iff RAF-Ver_σ accepts $(\mathcal{RAF}, \mathfrak{U} = \langle S, K\rangle)$.*
4. *$Exists_\sigma$ accepts $\mathcal{AF}$ iff RAF-$Exists_\sigma$ accepts $\mathcal{RAF}$.*
5. *$Exists_\sigma^{\neg\varnothing}$ accepts $\mathcal{AF}$ iff RAF-$Exists_\sigma^{\neg\varnothing}$ accepts $\mathcal{RAF}$.*
6. *$Unique_\sigma$ accepts $\mathcal{AF}$ iff RAF-$Unique_\sigma$ accepts $\mathcal{RAF}$.*

□ Proof of Proposition 30: link (See page 252).

Given the previous property and given that `Af2Raf` is a polynomial time and log-space function, we can assert that:

Proposition 31. *The complexities of RAF decision problems are at least as hard as AF ones, for the semantics* complete, semi-stable, stable, preferred, grounded.

□ Proof of Proposition 31: link (See page 252).

Finally, we obtain the following proposition:

Proposition 32. *The complexities of RAF decision problems are the same as AF ones, for the semantics* complete, semi-stable, stable, preferred, grounded, *as stated in Table 13.1.*

□ Proof of Proposition 32: link (See page 253).

σ	*RAF-*					
	$Cred_\sigma$	$Skep_\sigma$	Ver_σ	$Exists_\sigma$	$Exists_\sigma^{\neg\varnothing}$	$Unique_\sigma$
Grounded	P-*c*	P-*c*	P-*c*	trivial	in L	trivial
Complete	NP-*c*	P-*c*	in L	trivial	NP-*c*	coNP-*c*
Preferred	NP-*c*	Π_2^P-*c*	coNP-*c*	trivial	NP-*c*	coNP-*c*
Stable	NP-*c*	coNP-*c*	in L	NP-*c*	NP-*c*	DP-*c*
Semi-stable	Σ_2^P-*c*	Π_2^P-*c*	coNP-*c*	trivial	NP-*c*	in Θ_2^P

Table 13.1: Complexities of RAF decision problems

Proposition 32 is a very interesting one. Indeed, the complexities for the decision problems in the context of RAF, are the same as the ones in Dung's framework, despite all the additional expressivity that is brought by the higher-order attacks.

Chapter 14

Hierarchical view of RAF and semantics decomposability

In this chapter, the notion of SCC_{af} is extended to RAF (Section 14.1). Based on it, the semantics decomposability of RAF semantics is studied (Section 14.3 on page 133). Finally, a hierarchical view of RAF is proposed (Section 14.2 on page 131). These new properties open perspectives for future algorithms.

14.1 RAF Strongly Connected Component

In this section we build up from basic ones the notion of SCC for RAF. We also established the link between the SCCs of a given RAF and the SCCs of its AF flattened version.

Definition 81 (RAF-walk)**.** *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and $e_1, ..., e_n \in (A \cup K)$ be elements of $\mathcal{RAF}$. A RAF-walk is a sequence $(e_1, ..., e_n)$ with $n \in \mathbb{N}^*$ such that:*

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

Definition 82 (RAF-path)**.** *A RAF-path is a RAF-walk in which all the elements are distinct. Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF. We denote by "$Paths_{raf}(\mathcal{RAF})$" the set of RAF-paths of $\mathcal{RAF}$.*

Example 52. *Let consider Figure 14.1 on the next page. As an example, we have: $(b, \gamma, \delta, d, \varepsilon, e) \in Paths_{raf}(\mathcal{RAF})$ (in green) and $(h, f) \notin Paths_{raf}(\mathcal{RAF})$ (in red). Note that $(h) \in Paths_{raf}(\mathcal{RAF})$ and $(f) \in Paths_{raf}(\mathcal{RAF})$.*

Definition 83 (RAF-cycle)**.** *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF. A RAF-cycle is a sequence $(e_1, ..., e_n)$ with $n \geq 2$ such that:*

- $(e_2, ..., e_n)$ *is a RAF-path*

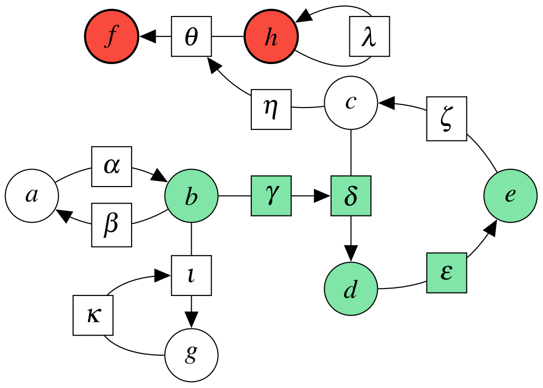


Figure 14.1: Example of RAF-paths: valid in green, invalid in red

- $(e_1,...,e_{n-1})$ *is a RAF-path*
- $e_1 = e_n$

We denote by "$Cycles_{raf}(\mathcal{RAF})$" the set of RAF-cycles of $\mathcal{RAF}$.

Example 53. *Let consider Figure 14.1. As an example, we have: $(\delta,d,\varepsilon,e,\zeta,c,\delta) \in Cycles_{raf}(\mathcal{RAF})$ and $(d,e,c,d) \notin Cycles_{raf}(\mathcal{RAF})$.*

Definition 84 (RAF-closed-walk)**.** *Let $\mathcal{RAF} = \langle A,K,s,t\rangle$ be a RAF. A sequence $(e_1,...,e_n)$ is said to be a RAF-closed-walk iff $(e_1,...,e_n)$ is a RAF-walk and that $e_1 = e_n$.*

We denote by "$ClosedWalk_{raf}(\mathcal{RAF})$" the set of RAF-closed-walk of $\mathcal{RAF}$.

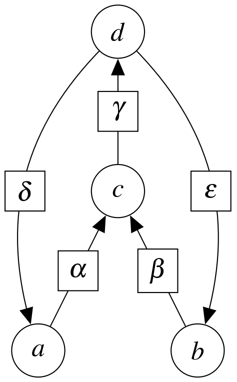


Figure 14.2: Example for RAF-closed-walk

Example 54. *Let consider the RAF in Figure 14.2. As an example, we have: $(a,\alpha,c,\gamma,d,\varepsilon,b,\beta,c,\gamma,d,\delta,a) \in ClosedWalk_{raf}(\mathcal{RAF})$.*

`Note:` *A RAF-cycle is by definition a RAF-closed-walk. A RAF-closed-walk is formed by the agglomeration of one or several cycles.*

Definition 85. *Let $\mathcal{RAF} = \langle A,K,s,t\rangle$ be a RAF. A RAF-walk $(e_1,\ldots,e_n)$ attacks an element $x \in (A \cup K)$ iff there exists e_i in the walk such that $x = t(e_i)$.*

Example 55. *Let consider Figure 14.1 on the previous page. As an example, the RAF-walk (d,ε,e,ζ) attacks c that is not in the RAF-walk. An other example: the RAF-cycle $(\delta,d,\varepsilon,e,\zeta,c,\delta)$ attacks c and does not attack δ.*

`Note:` *Following Definitions 84 and 85 on the previous page and on the current page, a RAF-cycle or a RAF-closed-walk can only attack elements that belongs to it.*

An equivalence relation can be defined on the notion of RAF-paths, establishing the fact that two elements x and y belong to the same equivalence class if and only if there exists a path from x to y *attacking* y and vice-versa (a path from y to x *attacking* x) in the RAF. This is an important difference with the definition of SCC in graph theory. Formally, this equivalence relation is defined, as follows:

Definition 86 (RAF Path Equivalence - PE_{raf})**.** *Given a RAF $\mathcal{RAF} = \langle A,K,s,t\rangle$, a RAF Path Equivalence noted $PE_{raf}(\mathcal{RAF})$ is a binary relation between elements of $\mathcal{RAF}$ such that:*

- $PE_{raf}(\mathcal{RAF}) \subseteq (A \cup K)^2$
- $\forall x \in (A \cup K), (x,x) \in PE_{raf}(\mathcal{RAF})$
- *Given two distinct elements $x,y \in (A \cup K)$, $(x,y) \in PE_{raf}(\mathcal{RAF})$ if and only if there exist:*
  - $p \in Paths_{raf}(\mathcal{RAF})$ *such that* $p = (x,...,e_{n-1},y)$ *and* $y = t(e_{n-1})$.
  - $p' \in Paths_{raf}(\mathcal{RAF})$ *such that* $p' = (y,...,e_{m-1},x)$ *and* $x = t(e_{m-1})$.

We introduce the notation $x \underset{\mathcal{RAF}}{\equiv} y$ to state that $(x,y) \in PE_{raf}(\mathcal{RAF})$.

The idea behind the previous definition is the following:

- y is attacked by p (in the sense of Definition 85) so x has an effect on y.
- x is attacked by p' (in the sense of Definition 85) so y has an effect on x.

As both affect each other they are considered as equivalent under the PE_{raf} relation.

Definition 87 (RAF Strongly Connected Component - SCC_{raf})**.**
Given a RAF $\mathcal{RAF} = \langle A,K,s,t\rangle$, a RAF Strongly Connected Component (SCC_{raf}) is an equivalence class of elements under the relation PE_{raf}. The set of SCC_{raf} of $\mathcal{RAF}$ is denoted by $SCCS_{raf}(\mathcal{RAF})$.

An interesting property of PE_{raf} relation is that if two elements are equivalent then there exists a RAF-closed-walk that contains and attacks them:

Proposition 33. *Let $\mathcal{RAF} = \langle A,K,s,t\rangle$ be a RAF and let x and y be two distinct elements of $\mathcal{RAF}$. The following property holds:*

$x \underset{\mathcal{RAF}}{\equiv} y$ *iff there exists a RAF-closed-walk* $c = (e_1,...,e_n) \in ClosedWalk_{raf}(\mathcal{RAF})$ *such that:*

- *$x \in c$. So there exists $i \in \{2,...,n\}$ such that: $x = e_i$*
- *$y \in c$. So there exists $j \in \{2,...,n\}$ such that: $y = e_j$*
- *$e_{i-1} \in K$*
- *$e_{j-1} \in K$*

□ *Proof of Proposition 33: link (See page 253).*

`Note:` *The two last conditions of the previous proposition are related to the fact that x (respectively y) must be attacked by the RAF-closed-walk.*

Example 56. *Let consider Figure 14.3 on the following page.*

- *$\{\iota, g\} \in SCCS_{raf}(\mathcal{RAF})$:*

  Indeed, there exists a RAF-path from g to ι attacking ι which is (g,κ,ι) and another one from ι to g attacking g which is (ι, g). As a consequence, we have: $\iota \underset{\mathcal{RAF}}{\equiv} g$. As there is no other element in $x \in \mathcal{RAF}$ such that $x \underset{\mathcal{RAF}}{\equiv} \iota$ or $x \underset{\mathcal{RAF}}{\equiv} g$ then $\{\iota, g\}$ is a SCC_{raf}. Note that, although $\kappa \in (g,\kappa,\iota)$, κ does not belong to this SCC_{raf} since it is not attacked by (g,κ,ι).

- *$\{\delta, d, \varepsilon, e, \zeta, c\} \notin SCCS_{raf}(\mathcal{RAF})$:*

  Although there exists a RAF-path from δ to d attacking d (which is (δ, d)), there is no RAF-path from d to δ attacking δ. The only RAF-path from d to δ is $(d,\varepsilon,e,\zeta,c,\delta)$ and it doesn't attack δ. As a consequence $d \underset{\mathcal{RAF}}{\not\equiv} \delta$ and so $\{\delta, d, \varepsilon, e, \zeta, c\}$ is not an SCC_{raf}.

- *$\{d, e, c\} \in SCCS_{raf}(\mathcal{RAF})$:*

  In the RAF-cycle $(\delta, d, \varepsilon, e, \zeta, c, \delta)$, d, e and c are all attacked. As a consequence we have: $d \underset{\mathcal{RAF}}{\equiv} e \underset{\mathcal{RAF}}{\equiv} c$. As there is no other element in $\mathcal{RAF}$ equivalent to them w.r.t. PE_{raf}, then $\{d,e,c\} \in SCCS_{raf}(\mathcal{RAF})$.

In this example we have:

$$SCCS_{raf}(\mathcal{RAF}) = \left\{ \begin{array}{c} \{\iota, g\}, \{d,e,c\}, \{a,b\}, \{\alpha\}, \{\beta\}, \{\kappa\}, \{\gamma\}, \\ \{\delta\}, \{\varepsilon\}, \{\zeta\}, \{\eta\}, \{\lambda\}, \{\theta\}, \{h\}, \{f\} \end{array} \right\}$$

Following the previous proposition, all elements attacked by a given RAF-closed-walk belong to the same SCC_{raf}:

Proposition 34. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF. Let $c = (e_1, ..., e_n) \in ClosedWalk_{raf}(\mathcal{RAF})$. Let $U = \{e_i | i \in \{2,...,n\}$ s.t. $e_{i-1} \in K\}$ be the set of attacked elements by the RAF-closed-walk c. The following property holds:*

U is included in some $S \in SCCS_{raf}(\mathcal{RAF})$

□ *Proof of Proposition 34: link (See page 253).*

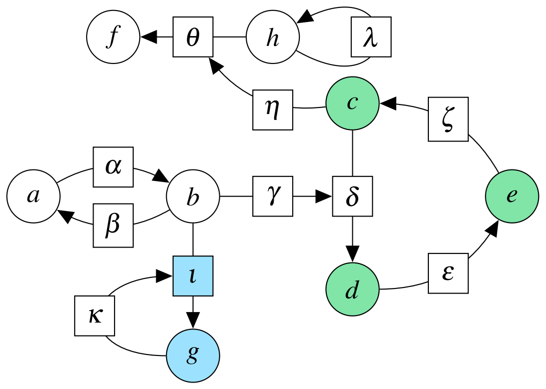


Figure 14.3: Example of two SCC_{raf}: one in green and one in blue

Examples 57 to 61 on pages 127–128 illustrate several cases and some interesting properties that lead to Propositions 35 and 36 on page 129.

Example 57. *Let consider Figure 14.4 on the next page and $\mathcal{RAF} = \langle A, K, s, t\rangle$ the RAF illustrated in Figure 14.4(a) on the next page. As you can see, considering the attack $\alpha \in \mathcal{RAF}$, we have in* `Raf2Af`$(\mathcal{RAF})$*: $(\alpha, \neg\alpha, a.\alpha, \beta) \in Paths_{af}($*`Raf2Af`*$(\mathcal{RAF}))$ and $(a, \neg a, a.\alpha, \beta) \in Paths_{af}($*`Raf2Af`*$(\mathcal{RAF}))$.*

Let consider now the RAF $\mathcal{RAF} = \langle A, K, s, t\rangle$ and the AF $\mathcal{AF} =$ `Raf2Af`*$(\mathcal{RAF})$ illustrated in Figure 14.5 on page 129:*

- *Considering the attack $\beta \in \mathcal{RAF}$, we have in* `Raf2Af`*$(\mathcal{RAF})$: $(a, \neg a, a.\beta, a) \in Cycles_{af}(\mathcal{AF})$ and there exists a unique path in* `Raf2Af`*$(\mathcal{RAF})$ whose first element is β and whose last is a, that is $(\beta, \neg\beta, a.\beta, a)$.*

- *Considering the attack $\alpha \in \mathcal{RAF}$, we have in $\mathcal{AF}$: $(\alpha, \neg\alpha, a.\alpha, \alpha) \in Cycles_{af}(\mathcal{AF})$ being the unique cycle in $\mathcal{AF}$ whose first and last elements are α and $(a, \neg a, a.\alpha, \alpha)$ is a path in $\mathcal{AF}$.*

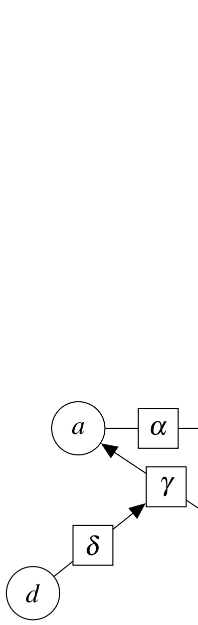
Example 58. *Let consider Figure 14.4 on the next page. Let $\mathcal{RAF}$ be the RAF in Figure 14.4(a) on the next page and let consider the RAF-path (a, α, β).*

- *In* `Raf2Af`*$(\mathcal{RAF})$, there is a path from a to β which is $(a, \neg a, a.\alpha, \beta)$. But in a slightly different RAF it could be possible that there exists another RAF-path from a to β and so another path from a to β in* `Raf2Af`*$(\mathcal{RAF})$. See Example 59 on the next page.*

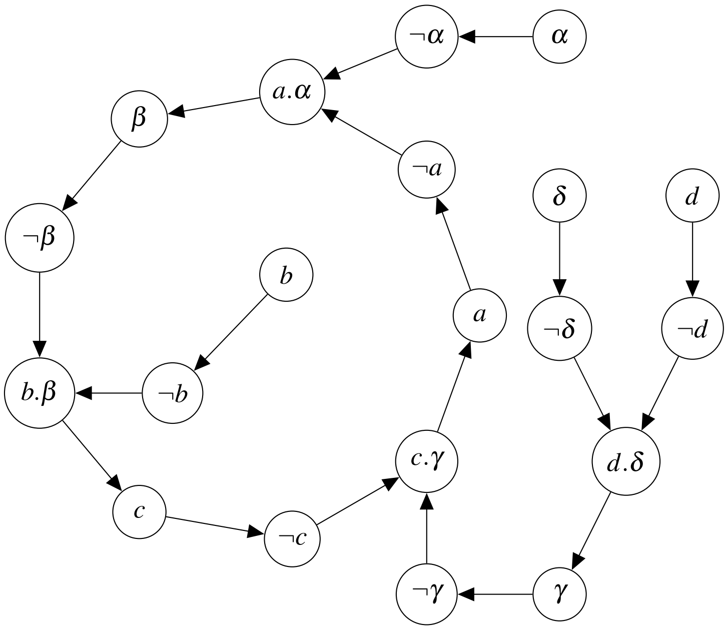
- *In* `Raf2Af`*$(\mathcal{RAF})$, there is a path from α to β which is $(\alpha, \neg\alpha, a.\alpha, \beta)$. This path is unique as an attack has a unique target.*

Example 59. *Let consider Figure 14.6 on page 130. Let $\mathcal{RAF}$ be the RAF in Figure 14.6(a) on page 130 and $\mathcal{AF}$ its flattened version in Figure 14.6(b) on page 130. As one can notice there are two RAF-paths in $\mathcal{RAF}$ from a to β which are:*

(a, α, β) and $(a, \gamma, d, \delta, \beta)$



(a) A RAF



(b) Flattened version of the RAF in Figure 14.4(a)

Figure 14.4: RAF flattening illustration (Same as Figure 12.1)

Similarly there are two paths in $\mathcal{AF}$ from a to β which are:

$$(a, \neg a, a.\alpha, \beta) \text{ and } (a, \neg a, a.\gamma, d, \neg d, d.\delta, \beta)$$

Example 60. *Let consider Figure 14.4. Let $\mathcal{RAF}$ be the RAF in Figure 14.4(a) and let consider the RAF-path $(d, \delta, \gamma, a, \alpha, \beta)$. There is a path in* Raf2Af($\mathcal{RAF}$) *from d to β which is $(d, \neg d, d.\delta, \gamma, \neg\gamma, c.\gamma, a, \neg a, a.\alpha, \beta)$.*

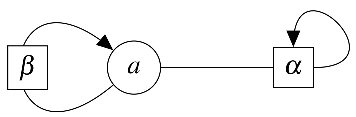
Example 61. *Let consider Figure 14.4. Let $\mathcal{RAF}$ be the RAF in Figure 14.4(a).*

- *As $(d, \delta, \gamma, a, \alpha, \beta)$ is a RAF-path, there is a path in* Raf2Af($\mathcal{RAF}$) *from d to β which is $(d, \neg d, d.\delta, \gamma, \neg\gamma, c.\gamma, a, \neg a, a.\alpha, \beta)$.*
- *Furthermore, as $(d, \neg d, d.\delta, \gamma, \neg\gamma, c.\gamma, a, \neg a, a.\alpha, \beta)$ is a path and that $d \in \mathcal{RAF}$ and $\beta \in \mathcal{RAF}$ then there is also a RAF-path in $\mathcal{RAF}$ from d to β such that the before last element is an attack. The path $(d, \delta, \gamma, a, \alpha, \beta)$ satisfies those conditions.*

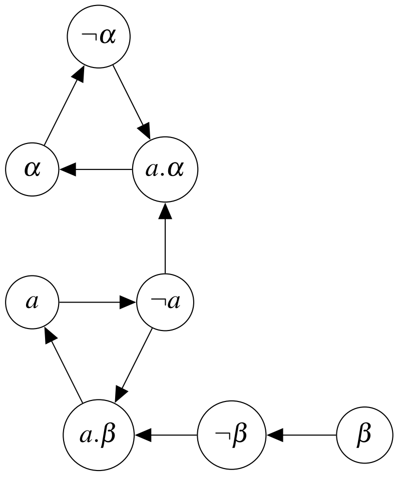
The following proposition is very important to establish the correspondence between SCC_{raf} and SCC_{af} in a RAF flattened version:

Proposition 35. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $U \subseteq A \cup K$ be a subset of elements of $\mathcal{RAF}$.*

*U is included in some $S \in SCCS_{raf}(\mathcal{RAF})$ iff U is included in some $S' \in SCCS_{af}($*Raf2Af($\mathcal{RAF}$)*$)$*



(a) A RAF



(b) Flattened version of the RAF in Figure 14.5(a)

Figure 14.5: RAF flattening illustration

□ *Proof of Proposition 35: link (See page 261).*

Example 62. *As an illustration of Proposition 35, let consider Figure 14.4 on the previous page. Let $\mathcal{RAF}$ be the RAF in Figure 14.4(a) on the previous page.*

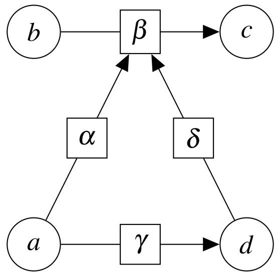
$(a,\beta) \in PE_{af}(\texttt{Raf2Af}(\mathcal{RAF}))$. As a consequence they are in the same SCC_{raf} in $\mathcal{RAF}$ which is: $S = \{a,\beta,c\}$. $(a,\beta) \in PE_{raf}(\mathcal{RAF})$. As a consequence they are in the same SCC_{af} in $\texttt{Raf2Af}(\mathcal{RAF})$ which is: $S' = \{a,\neg a,a.\alpha,\beta,\neg\beta,b.\beta,c,\neg c,c.\gamma\}$. We have: $\{a,\beta\} \subseteq S$ and $\{a,\beta\} \subseteq S'$.

Finally, the correspondence SCC_{raf} and SCC_{af} is fully established by the following proposition:

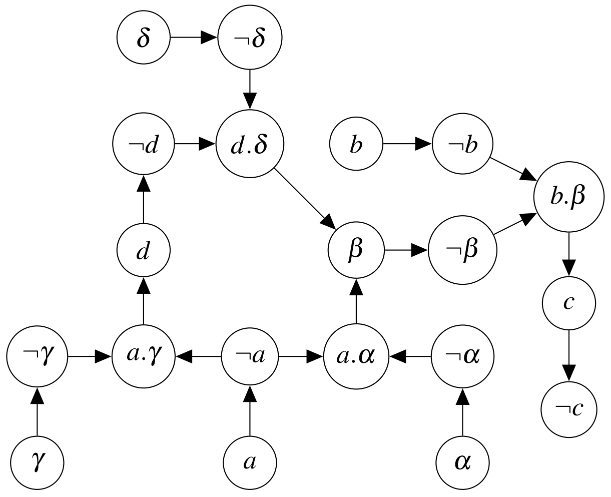
Proposition 36. *Let $\mathcal{RAF} = \langle A,K,s,t\rangle$ be a RAF and $S \subseteq A \cup K$ be a subset of elements of $\mathcal{RAF}$.*

$$S \in SCCS_{raf}(\mathcal{RAF})$$
$$\textit{iff}$$
$$\left(\begin{array}{l} S\ \cup \\ \{\neg a \in Not_A | a \in S \textit{ and } (|S| > 1 \textit{ or } (\exists \alpha \in K \textit{ s.t. } s(\alpha) = a \textit{ and } t(\alpha) = a))\}\ \cup \\ \{\neg\alpha \in Not_K | \alpha \in S \textit{ and } (|S| > 1 \textit{ or } t(\alpha) = \alpha)\}\ \cup \\ \{s(\alpha).\alpha \in And_{A,K} | \alpha \in S \textit{ and } (|S| > 1 \textit{ or } t(\alpha) = \alpha)\}\ \cup \\ \{s(\alpha).\alpha \in And_{A,K} | s(\alpha) \in S \textit{ and } t(\alpha) \in S\} \end{array}\right) \in SCCS_{af}(\texttt{Raf2Af}(\mathcal{RAF}))$$

□ *Proof of Proposition 36: link (See page 262).*



(a) A RAF



(b) Flattened version of the RAF in Figure 14.6(a)

Figure 14.6: RAF flattening illustration n°2

Example 63. *Let consider the RAF* Γ *and its corresponding AF in Figure 14.7 on the following page.*

- *See* $S = \{a, \beta, c\} \in SCCS_{raf}(\mathcal{RAF})$ *and* $S' = \{a, \neg a, a.\alpha, \beta, \neg\beta, b.\beta, c, \neg c, c.\gamma\} \in SCCS_{af}(\texttt{Raf2Af}(\mathcal{RAF}))$ *in blue. In details, following the successive unions in Proposition 36 on the previous page, we have:*

$$S' = S \cup \{\neg a, \neg c\} \cup \{\neg\beta\} \cup \{b.\beta\} \cup \{a.\alpha, c.\gamma\}$$

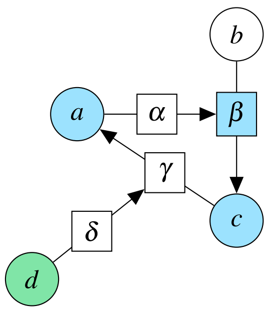
- *See* $T = \{d\} \in SCCS_{raf}(\mathcal{RAF})$ *and* $T' = \{d\} \in SCCS_{af}(\texttt{Raf2Af}(\mathcal{RAF}))$ *in green. Indeed following Proposition 36 on the previous page, we have:*
  - $\{\neg a \in Not_A | a \in T \text{ and } (|T| > 1 \text{ or } (a,a) \in K)\} = \varnothing$
  - $\{\neg\alpha \in Not_K | \alpha \in T \text{ and } |T| > 1\} = \varnothing$
  - $\{s(\alpha).\alpha \in And_{A,K} | \alpha \in T \text{ and } |T| > 1\} = \varnothing$
  - $\{s(\alpha).\alpha \in And_{A,K} | s(\alpha) \in T \text{ and } t(\alpha) \in T\} = \varnothing$

Example 64. *Let consider the RAF* Γ *and its corresponding AF in Figure 14.8 on page 132.*

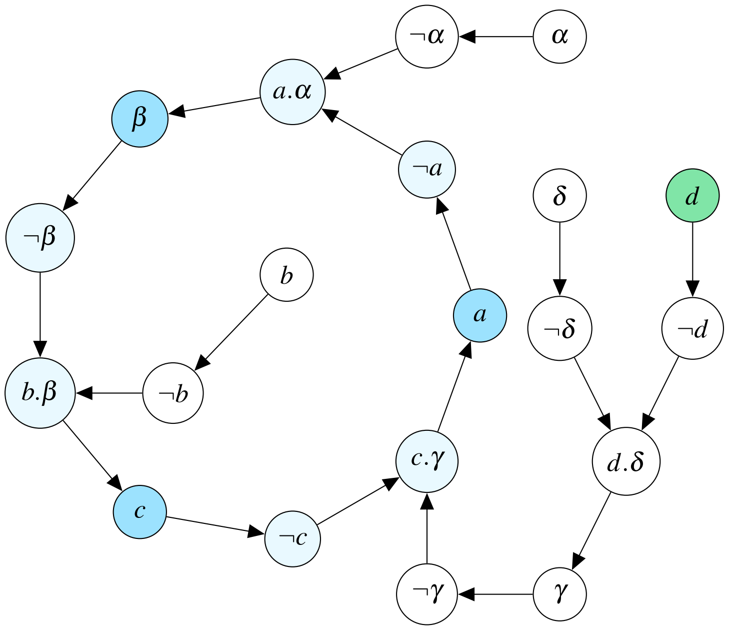
- *See* $S = \{a\} \in SCCS_{raf}(\mathcal{RAF})$ *and* $S' = \{a, \neg a, a.\alpha\} \in SCCS_{af}(\texttt{Raf2Af}(\mathcal{RAF}))$ *in blue. In details, following the successive unions in Proposition 36 on the previous page, we have:*

$$S' = S \cup \{\neg a\} \cup \{\} \cup \{\} \cup \{a.\alpha\}$$

- *See* $T = \{\alpha\} \in SCCS_{raf}(\mathcal{RAF})$ *and* $T' = \{\alpha\} \in SCCS_{af}(\texttt{Raf2Af}(\mathcal{RAF}))$ *in green. Indeed following Proposition 36 on the previous page, we have:*
  - $\{\neg a \in Not_A | a \in T \text{ and } (|T| > 1 \text{ or } (a,a) \in K)\} = \varnothing$
  - $\{\neg\alpha \in Not_K | \alpha \in T \text{ and } |T| > 1\} = \varnothing$



(a) RAF example



(b) Flattened version of the RAF in Figure 14.7(a)

Figure 14.7: SCC_{raf} to SCC_{af} example
One SCC_{raf} and its corresponding SCC_{af} in green
One SCC_{raf} and its corresponding SCC_{af} in blue
(light blue used for elements not in the initial RAF)

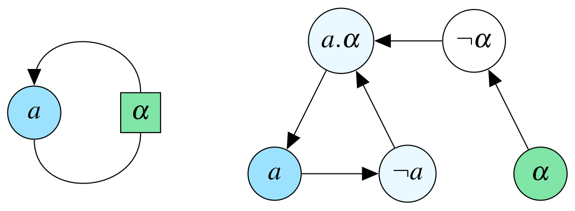
– $\{s(\alpha).\alpha \in And_{A,K} | \alpha \in T \text{ and } |T| > 1\} = \varnothing$
– $\{s(\alpha).\alpha \in And_{A,K} | s(\alpha) \in T \text{ and } t(\alpha) \in T\} = \varnothing$

Definition 88. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} =$ `Raf2Af`$(\mathcal{RAF})$ be the corresponding AF of $\mathcal{RAF}$. Let $S \in SCCS_{raf}(\mathcal{RAF})$ be a SCC_{raf} and let $S' \in SCCS_{af}(\mathcal{AF})$ be an SCC_{af}. We say that S' is the SCC_{af} corresponding to S (and vice-versa) iff :*

$$S' = \left(\begin{array}{l} S \cup \\ \{\neg a \in Not_A | a \in S \text{ and } (|S| > 1 \text{ or } (\exists \alpha \in K \text{ s.t. } s(\alpha) = a \text{ and } t(\alpha) = a))\} \cup \\ \{\neg \alpha \in Not_K | \alpha \in S \text{ and } (|S| > 1 \text{ or } t(\alpha) = \alpha)\} \cup \\ \{s(\alpha).\alpha \in And_{A,K} | \alpha \in S \text{ and } (|S| > 1 \text{ or } t(\alpha) = \alpha)\} \cup \\ \{s(\alpha).\alpha \in And_{A,K} | s(\alpha) \in S \text{ and } t(\alpha) \in S\} \end{array} \right)$$

14.2 SCC partial order and hierarchy

In this section, we highlight the fact that there exists a partial order over the SCC_{raf} of RAF according to the binary relation defined in Definition 89 on the next page.



(a) A RAF (b) Flattened version of the RAF in Figure 14.8(a)

Figure 14.8: SCC_{raf} to SCC_{af} 2nd example
One SCC_{raf} and its corresponding SCC_{af} in green
One SCC_{raf} and its corresponding SCC_{af} in blue
(light blue used for elements not in the initial RAF)

Definition 89 (SCC_{raf} relation: $\preccurlyeq$)**.** *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF, $S \in SCCS_{raf}(\mathcal{RAF})$ and $S' \in SCCS_{raf}(\mathcal{RAF})$. We define the relation $\preccurlyeq$ as a binary relation between elements of $SCCS_{raf}(\mathcal{RAF})$ as following:*

$$S \preccurlyeq S' \text{ if and only if}$$
$$(\exists (e_1, ..., e_{n-1}, e_n) \in Paths_{raf}(\mathcal{RAF}) \text{ s.t. } e_1 \in S \text{ and } e_n \in S' \text{ and } e_{n-1} \in K)$$
$$\text{or}$$
$$S = S'$$

Proposition 37. *The relation $\preccurlyeq$ is a partial order.*

□ Proof of Proposition 37: link (See page 266).

Given this partial ordering, we can define the notions of predecessors and successors of an SCC_{raf}:

Definition 90 (Predecessor and successor of an SCC_{raf})**.** *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and let $S_1 \in SCCS_{raf}(\mathcal{RAF})$ and $S_2 \in SCCS_{raf}(\mathcal{RAF})$ be two distinct SCC_{raf} of $\mathcal{RAF}$. We say that S_1 is a predecessor of S_2 (resp. S_2 is a successor of S_1) if and only if:*

$$S_1 \preccurlyeq S_2 \text{ and } \nexists S_3 \in SCCS_{raf}(\mathcal{RAF}) \text{ s.t. } S_3 \neq S_1 \text{ and } S_3 \neq S_2 \text{ and } S_1 \preccurlyeq S_3 \text{ and } S_3 \preccurlyeq S_2$$

Given two SCC_{raf} such that one is the predecessor of the other, the following property holds:

Proposition 38. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and let $S \in SCCS_{raf}(\mathcal{RAF})$ and $S' \in SCCS_{raf}(\mathcal{RAF})$ be two distinct SCC_{raf}. If S is the predecessor of S' (resp. S' is the successor of S) then for any path $p = (e_1, ..., e_{n-1}, e_n) \in Paths_{raf}(\mathcal{RAF})$ such that $e_1 \in S$ and $e_n \in S'$ and $e_{n-1} \in K$ then for all $i \in \{1, ..., n\}$, we have the following property:*

$$e_i \in p \cap K \implies t(e_i) \in (S \cup S')$$

□ Proof of Proposition 38: link (See page 267).

Example 65. *Let consider the RAF in Figure 14.7(a) on the previous page. We have:*

$$SCCS_{raf}(\mathcal{RAF}) = \{\{b\}, \{d\}, \{a, \beta, c\}, \{\gamma\}, \{\delta\}, \{\alpha\}\}$$

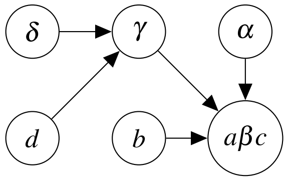


Figure 14.9: $Dag_{scc}(\mathcal{RAF})$ corresponding to the RAF in Figure 14.7(a) on page 131

Here is the partial order among these SCC_{raf}*:*

- $\{b\} \preccurlyeq \{a,\beta,c\}$
- $\{\alpha\} \preccurlyeq \{a,\beta,c\}$
- $\{\gamma\} \preccurlyeq \{a,\beta,c\}$
- $\{d\} \preccurlyeq \{\gamma\}$ *and* $\{d\} \preccurlyeq \{a,\beta,c\}$
- $\{\delta\} \preccurlyeq \{\gamma\}$ *and* $\{\delta\} \preccurlyeq \{a,\beta,c\}$

Given a RAF, if we reduce all its SCC_{raf} to super nodes then we obtain, as for AFs (or more generally: as for directed graphs), a "*DAG*" (Directed Acyclic Graph) as stated by Proposition 39. Definition 91 describes the transformation that produces the *DAG* corresponding to a given RAF.

Definition 91 (Dag_{scc} transformation)**.** *Let* $\mathcal{RAF} = \langle A,K,s,t\rangle$ *be a RAF and let associate to each* $S_i \in SCCS_{raf}(\mathcal{RAF})$ *a super node* n_i*.* $Dag_{scc}(\mathcal{RAF})$ *is the directed graph defined as following:*

$$Dag_{scc}(\mathcal{RAF}) = \big\langle \{n_i | S_i \in SCCS_{raf}(\mathcal{RAF})\}, \{(n_i,n_j) | S_i \textit{ is a predecessor of } S_j\}\big\rangle$$

Proposition 39. *Let* $\mathcal{RAF} = \langle A,K,s,t\rangle$ *be a RAF.* $Dag_{scc}(\mathcal{RAF})$ *is acyclic.*

□ Proof of Proposition 39: link (See page 267).

Example 66. *Let consider the RAF in Figure 14.7(a) on page 131 and let illustrate the notion of* Dag_{scc}*. Let* $a\beta c$ *be the super node corresponding to the* SCC_{raf} $\{a,\beta,c\}$*. Given that the other* SCC_{raf} *of* $\mathcal{RAF}$ *are all singletons we associate to each of them a super node having as name the element it possesses. See the resulting DAG in Figure 14.9.*

This property of SCC_{raf} partial ordering opens perspectives for algorithms computing RAF semantics following a certain hierarchical view of a given RAF.

14.3 Decomposability of semantics

In this chapter the *decomposability* property of semantics defined for AF (introduced in [9]) is extended to RAF. We first give some formal definitions to introduce the *decomposability* of RAF semantics (Section 14.3.1 on the next page). Then, we give an illustration of this notion for the RAF-*complete* semantics (Section 14.3.2 on page 142). Finally, the *decomposability* properties of the RAF-*complete*, RAF-*grounded*, RAF-*preferred*, RAF-*semi-stable* and RAF-*stable* semantics are established and proven (Section 14.3.3 on page 153).

14.3.1 Definitions

In Dung's Argumentation Framework attacks are always valid. Based on the notion introduced in [9] and [8], any AF can be splitted into several sub-frameworks by simply ignoring some attacks. Indeed the influence of an input attack on a sub-framework only depends to the acceptance state of its source in its own sub-framework. It is not the case for RAF. Attacks, as arguments, can be labelled *in*, *out* or *und*. As a consequence, we cannot just ignore attacks to split a RAF. Furthermore, as attacks can be attacked in RAF, the removal of an attack could require some other removals in cascade (see Example 67).

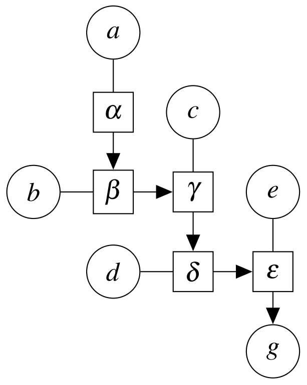


Figure 14.10: Example of attacks in cascade

Example 67. *Let consider the RAF in Figure 14.10. If we want to split this RAF so that the argument g may be alone in its sub-framework, we cannot just ignore the existence of ε, because it will require the same treatment for δ, and then for γ, then β and α.*

Now, if we do not suppress attacks while splitting RAFs, we will have attacks without targets or without sources. Given that, the result of such a split do not produce RAFs. A new notion is thus necessary to capture this idea of RAF splitting: "*Partial RAF*".

Definition 92 (Partial RAF)**.** *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF. A partial RAF $\widetilde{\mathcal{RAF}} = \left\langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t\right\rangle$ of $\mathcal{RAF}$ is a tuple where:*

- *$\tilde{A} \subseteq A$ is a set representing arguments*
- *$\tilde{K} \subseteq K$ is a set representing attacks*
- *$\tilde{s} : \tilde{K} \to \{\mathtt{true}, \mathtt{false}\}$ is a boolean function that indicates whether or not an attack in $\tilde{K}$ has its source in $\tilde{A}$ defined as following:*

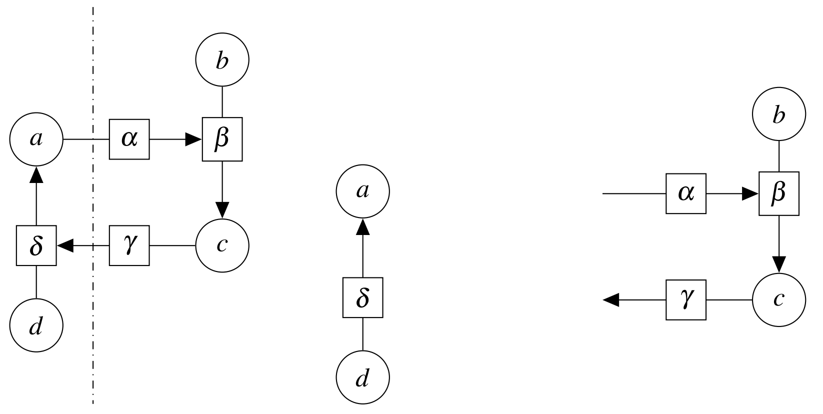
$$\forall \alpha \in \tilde{K}, \tilde{s}(\alpha) = \mathtt{true} \text{ if } s(\alpha) \in \tilde{A} \text{ otherwise } \mathtt{false}$$

- *$\tilde{t} : \tilde{K} \to \{\mathtt{true}, \mathtt{false}\}$ is a boolean function that indicates whether or not an attack in $\tilde{K}$ has its target in $\tilde{A} \cup \tilde{K}$*

$$\forall \alpha \in \tilde{K}, \tilde{t}(\alpha) = \mathtt{true} \text{ if } t(\alpha) \in \tilde{A} \cup \tilde{K} \text{ otherwise } \mathtt{false}$$

Example 68. *Figure 14.11 gives an illustration of partial RAFs. We have:*

- $\mathcal{RAF} = \langle A, K, s, t \rangle$ *with:*
  - $A = \{a, b, c, d\}$ *and* $K = \{\alpha, \beta, \gamma, \delta\}$
  - $s(\alpha) = a$, $s(\beta) = b$, $s(\gamma) = c$, $s(\delta) = d$
  - $t(\alpha) = \beta$, $t(\beta) = c$, $t(\gamma) = \delta$, $t(\delta) = a$
- $\widetilde{\mathcal{RAF}}_1 = \langle \tilde{A}_1, \tilde{K}_1, \tilde{s}_1, \tilde{t}_1, s, t \rangle$ *with:*
  - $\tilde{A}_1 = \{a, d\}$ *and* $\tilde{K}_1 = \{\delta\}$
  - $\tilde{s}_1(\delta) = \mathtt{true}$
  - $\tilde{t}_1(\delta) = \mathtt{true}$
- $\widetilde{\mathcal{RAF}}_2 = \langle \tilde{A}_2, \tilde{K}_2, \tilde{s}_2, \tilde{t}_2, s, t \rangle$ *with:*
  - $\tilde{A}_2 = \{b, c\}$ *and* $\tilde{K}_2 = \{\alpha, \beta, \gamma\}$
  - $\tilde{s}_2(\alpha) = \mathtt{false}$, $\tilde{s}_2(\beta) = \mathtt{true}$, $\tilde{s}_2(\gamma) = \mathtt{true}$
  - $\tilde{t}_2(\alpha) = \mathtt{true}$, $\tilde{t}_2(\beta) = \mathtt{true}$, $\tilde{t}_2(\gamma) = \mathtt{false}$



(a) RAF example: $\mathcal{RAF}$ (b) Partial RAF n°1: $\widetilde{\mathcal{RAF}}_1$ (c) Partial RAF n°2: $\widetilde{\mathcal{RAF}}_2$

Figure 14.11: Illustration of partial RAFs

So using the notion of partial RAF we are able to define a partition of a RAF:

Definition 93 (RAF partition)**.** *Let* $\mathcal{RAF} = \langle A, K, s, t \rangle$ *be a RAF. Let* $\Omega = \{\omega_1, ..., \omega_n\}$ *be a partition*[1] *of* $(A \cup K)$*. A RAF partition of* $\mathcal{RAF}$ *is a set of partial RAFs* $\{\widetilde{\mathcal{RAF}}_1, ..., \widetilde{\mathcal{RAF}}_n\}$ *such that:*

$$\forall \omega_i \in \Omega, \widetilde{\mathcal{RAF}}_i = \langle \tilde{A}_i, \tilde{K}_i, \tilde{s}_i, \tilde{t}_i, s, t \rangle \text{ with:}$$

[1]So the following property holds for Ω:

- $\forall (i, j) \in \{1, ..., n\}$ s.t. $i \neq j, \omega_i \cap \omega_j = \varnothing$
- $\bigcup_{i=1}^{n} \omega_i = A \cup K$

- $\tilde{A}_i = \omega_i \cap A$
- $\tilde{K}_i = \omega_i \cap K$
- $\tilde{s}_i : \tilde{K}_i \to \{\texttt{true}, \texttt{false}\}$ *is a boolean function that indicates whether or not an attack in* $\tilde{K}_i$ *has its source in* $\tilde{A}_i$ *defined as following:*

$$\forall \alpha \in \tilde{K}_i, \tilde{s}_i(\alpha) = \texttt{true} \textit{ if } s(\alpha) \in \tilde{A}_i \textit{ otherwise } \texttt{false}$$

- $\tilde{t}_i : \tilde{K}_i \to \{\texttt{true}, \texttt{false}\}$ *is a boolean function that indicates whether or not an attack in* $\tilde{K}_i$ *has its target in* $\tilde{A}_i \cup \tilde{K}_i$

$$\forall \alpha \in \tilde{K}_i, \tilde{t}_i(\alpha) = \texttt{true} \textit{ if } t(\alpha) \in \tilde{A}_i \cup \tilde{K}_i \textit{ otherwise } \texttt{false}$$

Example 69. *Following Example 68 on the previous page,* $\widetilde{\mathcal{RAF}}_1$ *and* $\widetilde{\mathcal{RAF}}_2$ *form a RAF partition of* $\mathcal{RAF}$ *as* $\tilde{A}_1 \cap \tilde{A}_2 = \varnothing$*,* $\tilde{K}_1 \cap \tilde{K}_2 = \varnothing$ *and* $(A \cup K) = (\tilde{A}_1 \cup \tilde{K}_1) \cup (\tilde{A}_2 \cup \tilde{K}_2)$.

Among all possible partial RAFs, we highlight to particular types: the *"Well-founded partial RAF"* and the *"Independant partial RAF"*. The first type corresponds to partial RAFs in which all attacks have their sources. The second one corresponds to well founded partial RAFs that are not attacked from outside. Formally, we have the following definitions:

Definition 94 (Well founded Partial RAF)**.** *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* $\mathcal{RAF}$. $\widetilde{\mathcal{RAF}}$ *is said to be "well founded" if and only if the following property holds:* $\forall \alpha \in \tilde{K}$, $\tilde{s}(\alpha)$ *is* $\texttt{true}$.

Definition 95 (Independent Partial RAF)**.** *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* $\mathcal{RAF}$. $\widetilde{\mathcal{RAF}}$ *is said to be "independent" if and only if the following property holds:*

$$\widetilde{\mathcal{RAF}} \textit{ is well founded and } \forall \alpha \in K \textit{ s.t. } t(\alpha) \in (\tilde{A} \cup \tilde{K}),\ \alpha \in \tilde{K}$$

Example 70. *Following Example 68 on the previous page:* $\widetilde{\mathcal{RAF}}_1$ *is well founded but not independent,* $\widetilde{\mathcal{RAF}}_2$ *is neither well founded nor independent.*

Considering a partial RAF implies to consider also its "inputs":

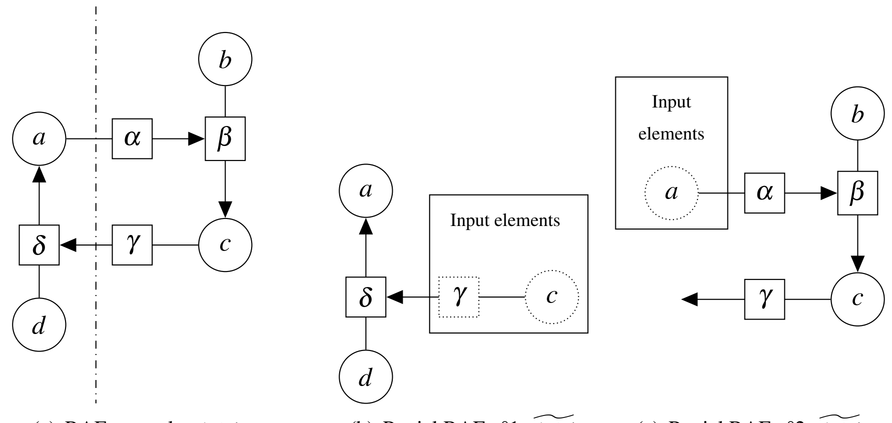
Definition 96 (Partial RAF with input)**.** *Let* $\mathcal{RAF} = \langle A, K, s, t\rangle$ *be a RAF and* $\widetilde{\mathcal{RAF}} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t\rangle$ *be a partial RAF of* $\mathcal{RAF}$. *The input* $\mathfrak{I}$ *of* $\widetilde{\mathcal{RAF}}$ *is a tuple* $\langle S^{inp}, Q^{inp}\rangle$ *where:*

- S^{inp} *is the set of arguments defined by* $S^{inp} = \{s(\alpha) \in (A \setminus \tilde{A}) | \alpha \in K \textit{ and } t(\alpha) \in (\tilde{A} \cup \tilde{K})\}$
- Q^{inp} *is the set of attacks defined by* $Q^{inp} = \{\alpha \in (K \setminus \tilde{K}) | t(\alpha) \in (\tilde{A} \cup \tilde{K})\}$

The tuple $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ *is called a "partial RAF with input", where* $\mathcal{L}^{inp}$ *is a structure labelling of the elements in* S^{inp} *and* Q^{inp}.

Example 71. *Let consider the partial RAFs* $\widetilde{\mathcal{RAF}}_1$ *and* $\widetilde{\mathcal{RAF}}_2$ *of Example 68 on the previous page. Figure 14.12 on the following page illustrates the notion of inputs for these partial RAFs.*

Let $\mathfrak{I}_1 = \left\langle S_1^{inp}, Q_1^{inp} \right\rangle$ *be the input of* $\widetilde{\mathcal{RAF}}_1$ *and* $\mathfrak{I}_2 = \left\langle S_2^{inp}, Q_2^{inp} \right\rangle$ *be the input of* $\widetilde{\mathcal{RAF}}_2$. *Let* $\mathcal{L}_1^{inp}$ *(respectively* $\mathcal{L}_2^{inp}$*) be a structure labelling associated to* $\mathfrak{I}_1$ *(respectively* $\mathfrak{I}_2$*). We have:*



(a) RAF example: $\mathcal{RAF}$ (b) Partial RAF n°1: $\widetilde{\mathcal{RAF}}_1$ (c) Partial RAF n°2: $\widetilde{\mathcal{RAF}}_2$

Figure 14.12: Illustration of partial RAFs with inputs
Note that input elements do not belong to the partial RAFs.

- $S_1^{inp} = \{c\}$, $Q_1^{inp} = \{\gamma\}$. *As an example, we may have:* $\mathcal{L}_1^{inp} = \langle\{(c, \mathtt{out})\}, \{(\gamma, \mathtt{und})\}\rangle$.
- $S_2^{inp} = \{a\}$, $Q_2^{inp} = \varnothing$. *As an example, we may have:* $\mathcal{L}_2^{inp} = \langle\{(a, \mathtt{in})\}, \varnothing\rangle$.

Then a standard RAF is the RAF that can be built from a partial RAF with inputs:

Definition 97 (Standard RAF). *Let* $\mathcal{RAF} = \langle A, K, s, t\rangle$ *be a RAF. Let* $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp}\right\rangle$ *be a partial RAF with input such that* $\widetilde{\mathcal{RAF}} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t\rangle$ *is a partial RAF of* $\mathcal{RAF}$. *The standard RAF w.r.t.* $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp}\right\rangle$ *is a RAF defined as* $\widetilde{\mathcal{RAF}}_s = \langle A_s, K_s, s_s, t_s\rangle$ *where:*

- $A_s = \tilde{A} \cup S^{inp} \cup \{\upsilon, \rho, \zeta\}$
- $K_s = \tilde{K} \cup Q^{inp} \cup N \cup \{\theta\}$, *with:*
  - $N = \{\alpha_x | x \in (Und \cup Out)\}$
  - $Out = \mathtt{out}(\mathcal{L}^{inp})$
  - $Und = \mathtt{und}(\mathcal{L}^{inp})$

And where $s_s : K_s \to A_s$ *and* $t_s : K_s \to (A_s \cup K_s)$ *are functions respectively mapping each attack to its source and to its target and such that:*

- $\forall \alpha \in (\tilde{K} \cup Q^{inp})$, $s_s(\alpha) = s(\alpha)$
- $\forall \alpha \in Q^{inp} \cup (\tilde{K} \setminus \{\alpha | \alpha \in \tilde{K} \text{ s.t. } \tilde{t}(\alpha) \text{ is } \mathtt{false}\})$, $t_s(\alpha) = t(\alpha)$
- $\forall \alpha \in \{\alpha | \alpha \in \tilde{K} \text{ s.t. } \tilde{t}(\alpha) \text{ is } \mathtt{false}\}$, $t_s(\alpha) = \zeta$
- $\forall \alpha_x \in \{\alpha_x \in N | x \in Out\}$, $s_s(\alpha_x) = \rho$
- $\forall \alpha_x \in \{\alpha_x \in N | x \in Und\}$, $s_s(\alpha_x) = \upsilon$
- $\forall \alpha_x \in N$, $t_s(\alpha_x) = x$

- $s_s(\theta) = \upsilon$
- $t_s(\theta) = \upsilon$

The following list gives the intuition of the new elements added in the standard RAF:

- υ is the argument that will serve to label `und` an element of the RAF input.
- θ is the attack whose source and target is υ, making υ a self attacking argument and thus an argument that will be labelled `und`.
- ρ is the argument that will serve to label `out` an element of the RAF input.
- N is the set of attacks that will link υ and ρ to all elements of the RAF input that should be labelled `out` or `und`.
- ζ is an argument that will serve as the target of all attacks of the partial RAF whose target does not belong to the partial RAF.

*`Note:` By definition, all RAF-*complete *labellings of the standard RAF*[2] $\widetilde{\mathcal{RAF}}_s$ *restricted to the elements of the input* $\mathfrak{I}$ *coincide with the labelling* $\mathcal{L}^{inp}$.

Example 72. *Figure 14.13 on the following page gives illustrations of standard RAFs.* $\widetilde{\mathcal{RAF}}_{s_1}$ *is the standard RAF corresponding to the partial RAF with input:*

$$\left\langle \widetilde{\mathcal{RAF}}_1, \mathfrak{I}_1 = \left\langle S_1^{inp} = \{c\}, Q_1^{inp} = \{\gamma\} \right\rangle, \mathcal{L}_1^{inp} = \langle \{(c, \mathtt{out})\}, \{(\gamma, \mathtt{und})\} \rangle \right\rangle$$

$\widetilde{\mathcal{RAF}}_{s_2}$ *is the standard RAF corresponding to the partial RAF with input:*

$$\left\langle \widetilde{\mathcal{RAF}}_2, \mathfrak{I}_2 = \left\langle S_2^{inp} = \{a\}, Q_2^{inp} = \varnothing \right\rangle, \mathcal{L}_2^{inp} = \langle \{(a, \mathtt{in})\}, \varnothing \rangle \right\rangle$$

Notice that ζ, υ *and* ρ *may be disconnected from the rest of the standard RAF following the partial RAF structure and the input labelling.*

Let define an operator in order to select a sub-part of a structure labelling:

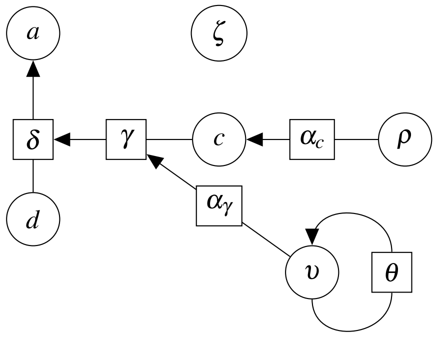
Definition 98. *(Structure labelling restriction $\downarrow$). Let* $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ *be a structure labelling. Let* $\mathcal{U} = \langle S, Q \rangle$ *be a structure. The* restriction *of* $\mathcal{L}$ *to* $\mathcal{U}$ *denoted as* $\mathcal{L} \downarrow_{\mathcal{U}}$ *is defined as:*

$$\left\langle \ell_A \cap (S \times \{\mathtt{in}, \mathtt{out}, \mathtt{und}\}), \ell_K \cap (Q \times \{\mathtt{in}, \mathtt{out}, \mathtt{und}\}) \right\rangle$$

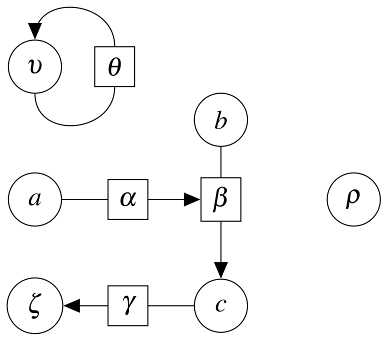
Given a RAF with input, the canonical local function is simply a function that gives the set of labellings under a certain semantics of the sub-RAF we are interested in (*i.e.* the input elements and the other fictive elements created are not in these labellings).

Definition 99 (RAF canonical local function)**.** *Let* $\mathcal{RAF} = \langle A, K, s, t \rangle$ *be a RAF,* $\widetilde{\mathcal{RAF}} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t \rangle$ *be a partial RAF of* $\mathcal{RAF}$, σ *be a semantics,* $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ *be a RAF with input, and* $\widetilde{\mathcal{RAF}}_s$ *be its standard RAF.*

[2] A standard RAF being a RAF, standard RAF labellings are simply RAF labellings.



(a) $\widetilde{\mathcal{RAF}}_{s_1}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_1, \mathfrak{I}_1, \mathcal{L}_1^{inp} \right\rangle$ with $N = \{\alpha_c, \alpha_\gamma\}$, $Out = \{c\}$, $Und = \{\gamma\}$



(b) $\widetilde{\mathcal{RAF}}_{s_2}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_2, \mathfrak{I}_2, \mathcal{L}_2^{inp} \right\rangle$ with $N = Out = Und = \varnothing$

Figure 14.13: Standard RAFs example

A local function $\mathscr{F}_\sigma^{raf}$ assigns to any partial RAF with input $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ a (possibly empty) set of labellings of $\widetilde{\mathcal{RAF}}$, i.e. $\mathscr{F}_\sigma^{raf}(\widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp}) \in 2^{\{\mathcal{L} | \mathcal{L} \text{ being any structure labelling over } \langle \tilde{A}, \tilde{K} \rangle\}}$.

The canonical local function $\mathscr{F}_\sigma^{raf}$ is the local function such that $\mathscr{F}_\sigma^{raf}(\widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp}) = \{\mathcal{L}\downarrow_{\langle \tilde{A} \cup \tilde{K} \rangle} | \mathcal{L} \in \mathscr{L}_{\sigma\text{-}raf}(\widetilde{\mathcal{RAF}}_s)\}$.

Example 73. *Following Example 72 on the previous page, we have for the RAF-*complete *semantics in the case of $\widetilde{\mathcal{RAF}}_1$:*

$$\mathscr{F}_{co}^{raf}(\widetilde{\mathcal{RAF}}_1, \mathfrak{I}_1, \mathcal{L}_1^{inp}) = \{\langle \{(a, \mathtt{out}), (d, \mathtt{in})\}, \{(\delta, \mathtt{in})\} \rangle\}$$

The next step consists in establishing a link between a RAF partition and a partition of its flattened version. Let first define a RAF partition selector:

Definition 100. *(RAF Partition selector). A RAF partition selector $\mathscr{S}$ is a function receiving as input a RAF $\mathcal{RAF} = \langle A, K, s, t \rangle$ and returning a set of partitions of $A \cup K$.*

While considering partitions of flattened RAF, some specific partitions must be considered:

Definition 101 (RAF-compliant partition selector)**.** *An AF partition selector $\mathscr{S}$ is said to be "RAF-compliant" iff for any RAF $\mathcal{RAF} = \langle A, K, s, t \rangle$ and its corresponding AF $\mathcal{AF} =$ `Raf2Af`$(\mathcal{RAF})$ (with $\mathcal{AF} = \langle A', K' \rangle$), we have the following property:*

$$\forall \Omega' \in \mathscr{S}(\mathcal{AF}), \forall \omega' \in \Omega', \begin{cases} x \in (A \cup K) \cap \omega' & \Longrightarrow \quad \neg x \in \omega' \\ \alpha \in K \cap \omega' & \Longrightarrow \quad s(\alpha).\alpha \in \omega' \end{cases}$$

To precise that a partition selector is RAF-compliant we use the notation: $\mathscr{S}_{raf\text{-}c}$. Let $\Omega' \in \mathscr{S}_{raf\text{-}c}(\mathcal{AF})$ be a partition of $\mathcal{AF}$ selected by some RAF-compliant selector. We say that Ω' is a RAF-compliant partition of $\mathcal{AF}$.

`Note`*: Definition 101 on the previous page describe a property, the property for AF partition selector to be RAF-compliant. But it does not define any selector. The following definition defines the default RAF-compliant partition selector, which is the AF partition selector that produces, given a flattened RAF, all RAF-compliant partitions.*

Definition 102 (Default RAF-compliant partition selector)**.** *The* default *RAF-compliant selector, denoted by $\mathscr{S}_{D\text{-}raf\text{-}c}$, is the AF RAF-compliant partition selector defined as follows:*

$$\forall \mathcal{RAF} = \langle A, K, s, t\rangle \in \Phi_{raf},$$

$$\mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{AF}) = \left\{ \Omega' \;\middle|\; \begin{array}{l} \text{Let } \Omega' \text{ be any partition of } \mathcal{AF} \\ \text{and } (\forall \omega'_i \in \Omega', \forall x \in (A \cup K), x \in \omega'_i \implies \neg x \in \omega'_i) \\ \text{and } (\forall \omega'_i \in \Omega', \forall \alpha \in K, \alpha \in \omega'_i \implies s(\alpha).\alpha \in \omega'_i) \end{array} \right\}, \text{ with } \mathcal{AF} = \mathtt{Raf2Af}(\mathcal{RAF})$$

The following definition establishes the relation between RAF partition selectors and AF RAF-compliant partition selectors:

Definition 103. *(AF and RAF partition selector correspondance) Let $\mathscr{S}$ be a RAF partition selector and let $\mathscr{S}_{raf\text{-}c}$ be an AF RAF-compliant partition selector. We say that $\mathscr{S}$ is the RAF counterpart of $\mathscr{S}_{raf\text{-}c}$ (and vice-versa) iff :*

$$\forall \mathcal{RAF} \in \Phi_{raf}, \mathscr{S}(\mathcal{RAF}) = \{\{\omega' \cap (A \cup K) | \omega' \in \Omega'\} | \Omega' \in \mathscr{S}_{raf\text{-}c}(\mathtt{Raf2Af}(\mathcal{RAF}))\}$$

The idea behind the previous definition is that, when we consider the partition of the flattened version of a RAF, we want that any element in $And_{A,K}$ belongs to the same part as the attack it is related to, any element in Not_A (resp. Not_K) belongs to the same part as the argument (resp. the attack) it is related to.

Although it could be defined otherwise, we choose this definition because:

1. The acceptance of an element $x \in (A \cup K)$ is intrinsically related to the acceptance of the argument $\neg x$:
   - x is labelled `in` *iff* $\neg x$ is labelled `out`
   - x is labelled `out` *iff* $\neg x$ is labelled `in`
   - x is labelled `und` *iff* $\neg x$ is labelled `und`
2. A RAF argument may be the source of several attacks, whereas an attack has only one target. It seems thus reasonable to put arguments that belong to $And_{A,K}$ in the same part as the argument corresponding to their attack.

The following example illustrates Definition 103.

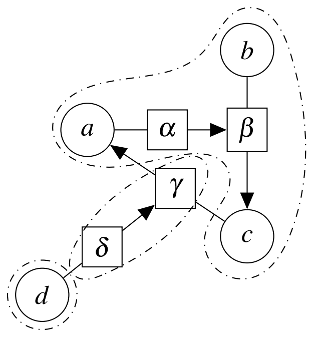
Example 74. *Let consider the frameworks in Figure 14.14 on the following page. Let $\mathscr{S}_{raf\text{-}c}$ be an AF "RAF-compliant" partition selector and $\mathscr{S}$ be its RAF counterpart.*

Let consider $\Omega \in \mathscr{S}$ such that:

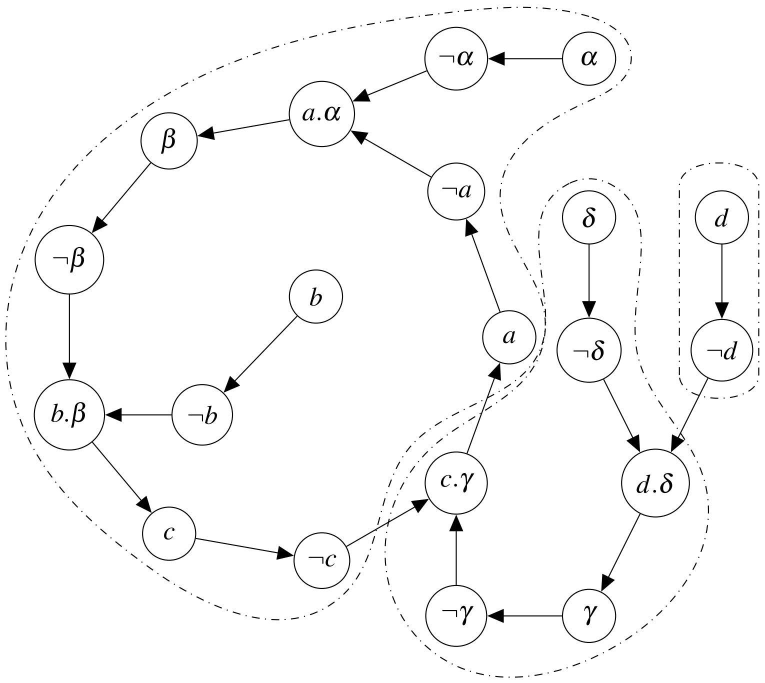
$$\Omega = \{\{d\}, \{\delta, \gamma\}, \{\alpha, \beta, b, c, a\}\}$$

We have thus $\Omega' \in \mathscr{S}_{raf\text{-}c}(\mathcal{AF})$ such that:

$$\Omega' = \{\{d, \neg d\}, \{\delta, \neg\delta, d.\delta, \gamma, \neg\gamma, c.\gamma\}, \{\alpha, \neg\alpha, a.\alpha, \beta, \neg\beta, b.\beta, b, \neg b, c, \neg c, a, \neg a\}\}$$



(a) RAF



(b) Flattened version of the RAF in Figure 14.14(a)

Figure 14.14: RAF-compliant partition example

And finally the last but not the least notion corresponds to the notion of semantics decomposability:

Definition 104. *(Semantics decomposability). A semantics σ is* fully decomposable *(or simply* decomposable*) if and only if there is a local function $\mathscr{F}_\sigma^{raf}$ such that for every RAF $\mathcal{RAF} = \langle A,K,s,t\rangle$ and every partition $\Omega = \{\omega_1,...,\omega_n\}$ of $(A\cup K)$ and $\{\widetilde{\mathcal{RAF}}_1,...,\widetilde{\mathcal{RAF}}_n\}$ the partition of $\mathcal{RAF}$ corresponding to Ω, the following property holds:*

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

With $\widetilde{\mathcal{RAF}}_i = \langle\tilde{A}_i,\tilde{K}_i,\tilde{s}_i,\tilde{t}_i,s,t\rangle$ and $\mathfrak{I}_i = \left\langle S_i^{inp},Q_i^{inp}\right\rangle$ and $\mathcal{L}_i^{inp}$ defined as following:

- $S_i^{inp} = \{s(\alpha)\notin\tilde{A}_i|\exists\alpha\in K \text{ s.t. } t(\alpha)\in(\tilde{A}_i\cup\tilde{K}_i)\}$
- $Q_i^{inp} = \{\alpha\notin\tilde{K}_i|\exists\alpha\in K \text{ s.t. } t(\alpha)\in(\tilde{A}_i\cup\tilde{K}_i)\}$
- $\mathcal{L}_i^{inp} = (\bigcup_{j\in\{1,...,n\}\text{ s.t. } j\neq i}\mathcal{L}_j)\downarrow_{\left\langle S_i^{inp},Q_i^{inp}\right\rangle}$

A semantics σ is said to be top-down decomposable *if and only if the following property holds:*

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

A semantics σ is said to be bottom-up decomposable *if and only if the following property holds:*

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

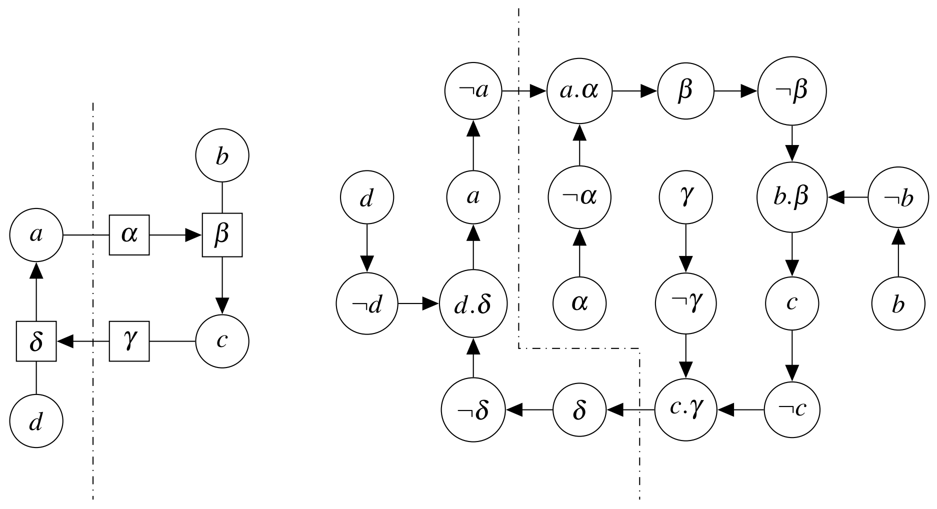
14.3.2 Illustration

In this section we illustrate the decomposability property of the complete semantics, and show the link between this property in AF and in RAF. For this purpose, let consider Figure 14.15 on the following page.

Let $\mathcal{RAF} = \langle A,K,s,t\rangle$ be the RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ (with $\mathcal{AF} = \langle A',K'\rangle$) be the flattened version of $\mathcal{RAF}$, as represented in Figure 14.15 on the following page. Let $\Omega' = \{\{d,\neg d,\delta,\neg\delta,d.\delta,a,\neg a\},\{\alpha,\neg\alpha, a.\alpha,\beta,\ \neg\beta,b,\neg b,b.\beta,\ c,\neg c,\gamma,\ \neg\gamma,c.\gamma\}\}$ be a partition of A'. We have: $\Omega'\in\mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{AF})$ (See Definition 102 on page 140). Let $\Omega = \Omega'\cap(A\cup K)$ be the partition of $A\cup K$ corresponding to Ω'. Let us split $\mathcal{RAF}$ along the partition Ω and $\mathcal{AF}$ along the partition Ω'. The partial RAFs $\widetilde{\mathcal{RAF}}_1$ and $\widetilde{\mathcal{RAF}}_2$ produced by the split of $\mathcal{RAF}$ are represented in Figure 14.16 on page 144 and the clusters κ_1 and κ_2 produced by the split of $\mathcal{AF}$ are represented in Figure 14.17 on page 144.

We have:

- $\widetilde{\mathcal{RAF}}_1 = \langle\tilde{A}_1,\tilde{K}_1,\tilde{s}_1,\tilde{t}_1,s,t\rangle$ and $\mathfrak{I}_1 = \langle\{c\},\{\gamma\}\rangle$ with:
  - $\tilde{A}_1 = \{a,d\}$ and $\tilde{K}_1 = \{\delta\}$
  - $\tilde{s}_1(\delta) = \texttt{true}$
  - $\tilde{t}_1(\delta) = \texttt{true}$



(a) RAF example: $\mathcal{RAF}$ (b) $\mathcal{AF}$: the flattened version of $\mathcal{RAF}$

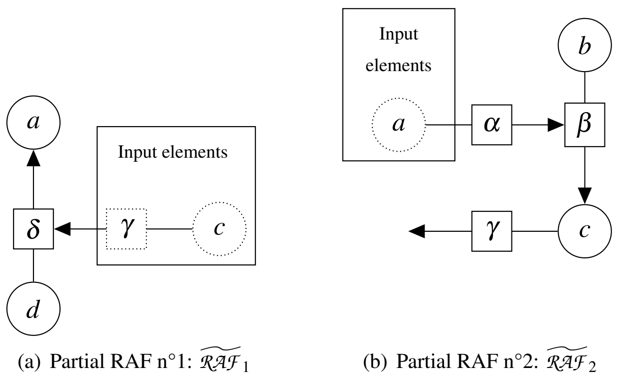
Figure 14.15: Running example for semantics decomposability illustration

- $\widetilde{\mathcal{RAF}}_2 = \langle \tilde{A}_2, \tilde{K}_2, \tilde{s}_2, \tilde{t}_2, s, t \rangle$ and $\mathfrak{I}_2 = \langle \{a\}, \varnothing \rangle$ with:
  - $\tilde{A}_2 = \{b, c\}$ and $\tilde{K}_2 = \{\alpha, \beta, \gamma\}$
  - $\tilde{s}_2(\alpha) = \texttt{false}$, $\tilde{s}_2(\beta) = \texttt{true}$, $\tilde{s}_2(\gamma) = \texttt{true}$
  - $\tilde{t}_2(\alpha) = \texttt{true}$, $\tilde{t}_2(\beta) = \texttt{true}$, $\tilde{t}_2(\gamma) = \texttt{false}$

And:

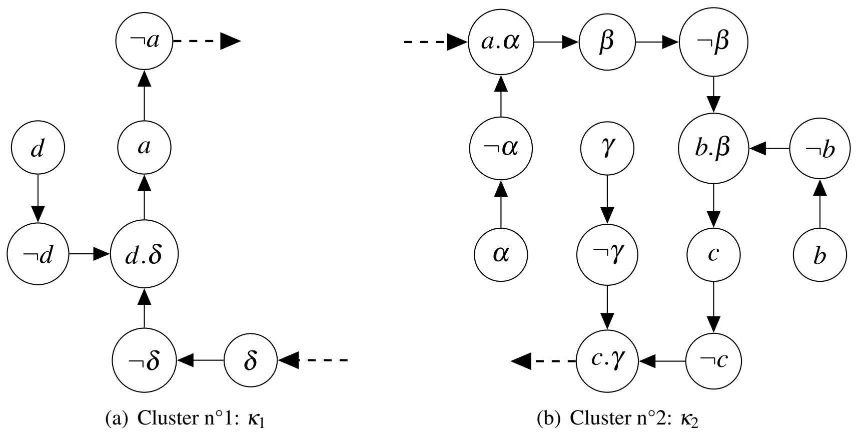
- $\kappa_1 = \langle af_1, I_1, O_1, B_1 \rangle$ with:
  - $af_1 = \left\langle A_{af_1}, K_{af_1} \right\rangle$ being an AF with:
    - $A_{af_1} = \{d, \neg d, \delta, \neg\delta, d.\delta, a, \neg a\}$
    - $K_{af_1} = \{(\delta, \neg\delta), (d, \neg d), (\neg d, d.\gamma), (\neg\delta, d.\gamma), (d.\delta, a), (a, \neg a)\}$
  - $I_1 = \{(c.\gamma, \delta)\}$ (the inward attacks)
  - $O_1 = \{(\neg a, a.\alpha)\}$ (the outward attacks)
  - $B_1 = \{\neg a, \delta\}$ (the border arguments)
- $\kappa_2 = \langle af_2, I_2, O_2, B_2 \rangle$ with:
  - $af_2 = \left\langle A_{af_2}, K_{af_2} \right\rangle$ being an AF with:
    - $A_{af_2} = \{\alpha, \neg\alpha, a.\alpha, \beta, \neg\beta, b, \neg b, b.\beta, c, \neg c, \gamma, \neg\gamma, c.\gamma\}$
    - $K_{af_2} = \{(\alpha, \neg\alpha), (\neg\alpha, a.\alpha), (a.\alpha, \beta), (\beta, \neg\beta), (b, \neg b), (\neg\beta, b.\beta), (\neg b, b.\beta), (b.\beta, c),$ $(c, \neg c), (\gamma, \neg\gamma), (\neg\gamma, c.\gamma), (\neg c, c.\gamma)\}$
  - $I_2 = \{(\neg a, a.\alpha)\}$

- $O_2 = \{(c.\gamma, \delta)\}$
- $B_2 = \{a.\alpha, c.\gamma\}$



(a) Partial RAF n°1: $\widetilde{\mathcal{RAF}}_1$ (b) Partial RAF n°2: $\widetilde{\mathcal{RAF}}_2$

Figure 14.16: Partial RAFs of $\mathcal{RAF}$
Notice that input elements do not belong to the partial RAFs.

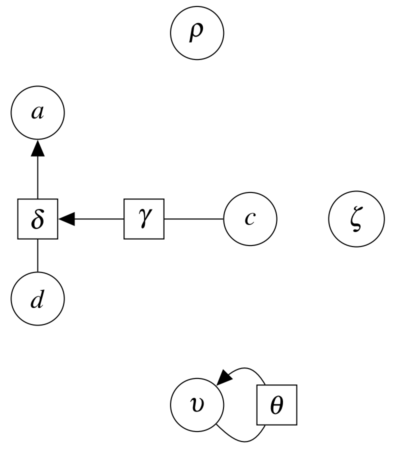


(a) Cluster n°1: κ_1 (b) Cluster n°2: κ_2

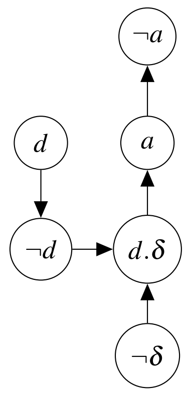
Figure 14.17: Clusters of $\mathcal{AF}$

Given the input elements of the partial RAFs of $\mathcal{RAF}$, let define the possible labellings associated with them. For $\widetilde{\mathcal{RAF}}_1$, we have:

- $\mathcal{L}^{inp}_{1.1} = \langle\{(c, \mathtt{in})\}, \{(\gamma, \mathtt{in})\}\rangle$
- $\mathcal{L}^{inp}_{1.2} = \langle\{(c, \mathtt{in})\}, \{(\gamma, \mathtt{out})\}\rangle$
- $\mathcal{L}^{inp}_{1.3} = \langle\{(c, \mathtt{in})\}, \{(\gamma, \mathtt{und})\}\rangle$
- $\mathcal{L}^{inp}_{1.4} = \langle\{(c, \mathtt{out})\}, \{(\gamma, \mathtt{in})\}\rangle$
- $\mathcal{L}^{inp}_{1.5} = \langle\{(c, \mathtt{out})\}, \{(\gamma, \mathtt{out})\}\rangle$
- $\mathcal{L}^{inp}_{1.6} = \langle\{(c, \mathtt{out})\}, \{(\gamma, \mathtt{und})\}\rangle$



(a) $\widetilde{\mathcal{RAF}}_{s_{1.1}}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_1, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.1} \right\rangle$



(b) $af'_{1.1}$ the induced AF of κ_1 under $\mu_{1.1}$ (Notice that δ has disappeared since its attacker is labelled *in*).

Figure 14.18: When c and γ are accepted

With:

- $\mu_{1.1} = \{(c.\gamma, \mathit{in})\}$
- $\mathcal{L}^{inp}_{1.1} = \langle\{(c, \mathit{in})\}, \{(\gamma, \mathit{in})\}\rangle$

- $\mathcal{L}^{inp}_{1.7} = \langle\{(c, \mathit{und})\}, \{(\gamma, \mathit{in})\}\rangle$
- $\mathcal{L}^{inp}_{1.8} = \langle\{(c, \mathit{und})\}, \{(\gamma, \mathit{out})\}\rangle$
- $\mathcal{L}^{inp}_{1.9} = \langle\{(c, \mathit{und})\}, \{(\gamma, \mathit{und})\}\rangle$

For $\widetilde{\mathcal{RAF}}_2$, we have:

- $\mathcal{L}^{inp}_{2.1} = \langle\{(a, \mathit{in})\}, \varnothing\rangle$
- $\mathcal{L}^{inp}_{2.2} = \langle\{(a, \mathit{out})\}, \varnothing\rangle$
- $\mathcal{L}^{inp}_{2.3} = \langle\{(a, \mathit{und})\}, \varnothing\rangle$

Likewise given the border arguments of the clusters of $\mathcal{AF}$, let define the possible contexts associated with them. For κ_1, we have:

- $\mu_{1.1} = \{(c.\gamma, \mathit{in})\}$
- $\mu_{1.2} = \{(c.\gamma, \mathit{out})\}$
- $\mu_{1.3} = \{(c.\gamma, \mathit{und})\}$

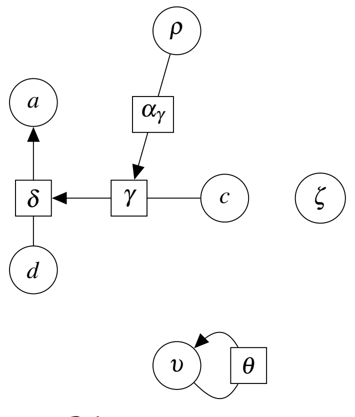
For κ_2, we have:

- $\mu_{2.1} = \{(\neg a, \mathit{out})\}$
- $\mu_{2.2} = \{(\neg a, \mathit{in})\}$
- $\mu_{2.3} = \{(\neg a, \mathit{und})\}$

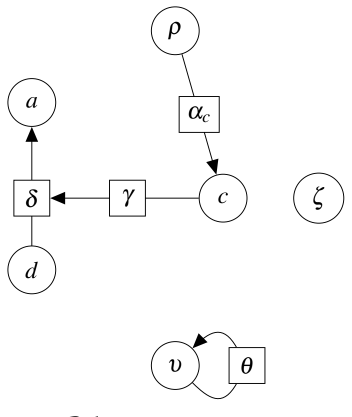
Figures 14.18 to 14.23 on pages 145–150 show the standard RAFs produced from $\widetilde{\mathcal{RAF}}_1$ and $\widetilde{\mathcal{RAF}}_2$ associated with the induced AF produced from κ_1 and κ_2. It is worth to notice that several standard RAFs can be associated to a same induced AF (see for instance Figures 14.19 and 14.20 on the next page and on page 147).

Let consider Figure 14.18. Notice that, only for this figure, the elements belonging to the partial RAF are highlighted in green and those being the RAF inputs in blue, we have:[3]

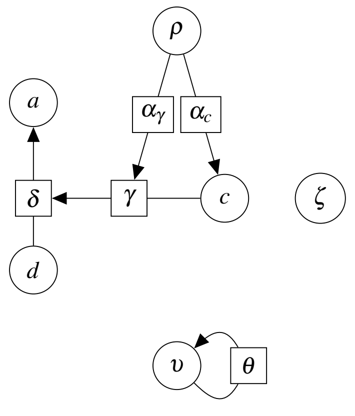
[3]The use of these colors illustrates more easily the common points between the RAF side and the AF side and also the fact that the inputs can still appear on the RAF side but not on the AF side.



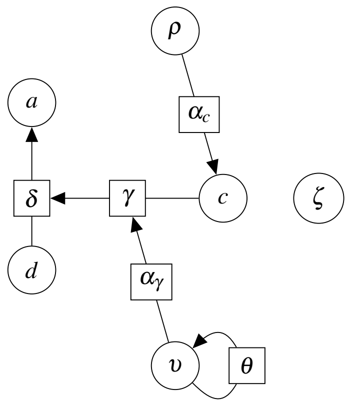
(a) $\widetilde{\mathcal{RAF}}_{s_{1.2}}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_1, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.2} \right\rangle$



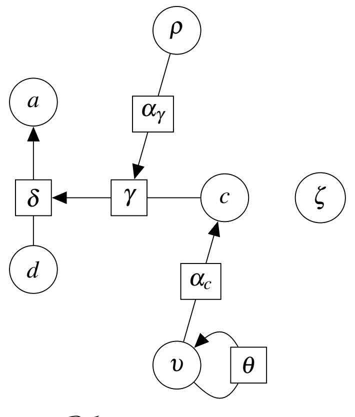
(b) $\widetilde{\mathcal{RAF}}_{s_{1.4}}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_1, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.4} \right\rangle$



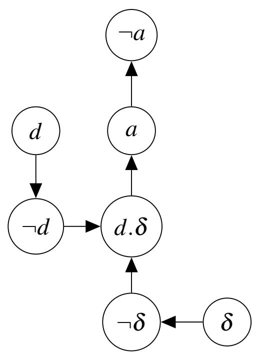
(c) $\widetilde{\mathcal{RAF}}_{s_{1.5}}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_1, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.5} \right\rangle$



(d) $\widetilde{\mathcal{RAF}}_{s_{1.6}}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_1, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.6} \right\rangle$



(e) $\widetilde{\mathcal{RAF}}_{s_{1.8}}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_1, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.8} \right\rangle$

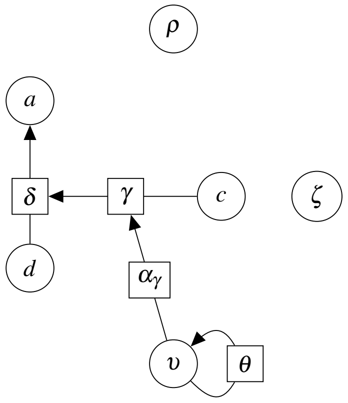


(f) $af'_{1.2}$ the induced AF of κ_1 under $\mu_{1.2}$

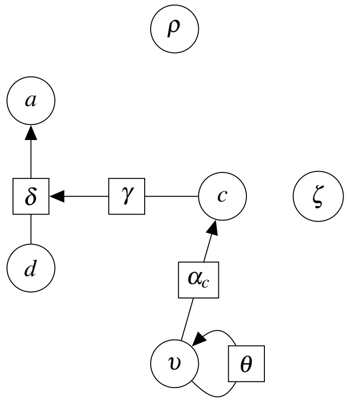
Figure 14.19: When c or γ is rejected

With:

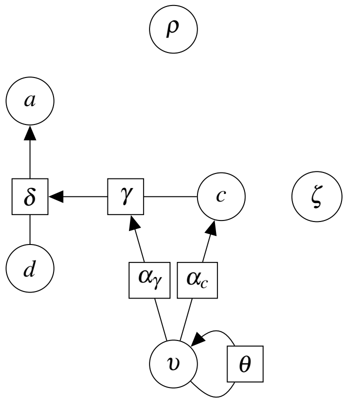
- $\mu_{1.2} = \{(c.\gamma, out)\}$
- $\mathcal{L}^{inp}_{1.2} = \langle\{(c, in)\}, \{(\gamma, out)\}\rangle$
- $\mathcal{L}^{inp}_{1.4} = \langle\{(c, out)\}, \{(\gamma, in)\}\rangle$
- $\mathcal{L}^{inp}_{1.5} = \langle\{(c, out)\}, \{(\gamma, out)\}\rangle$
- $\mathcal{L}^{inp}_{1.6} = \langle\{(c, out)\}, \{(\gamma, und)\}\rangle$
- $\mathcal{L}^{inp}_{1.8} = \langle\{(c, und)\}, \{(\gamma, out)\}\rangle$



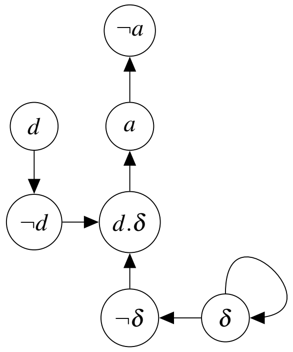
(a) $\widetilde{\mathcal{RAF}}_{s_{1.3}}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_1, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.3} \right\rangle$



(b) $\widetilde{\mathcal{RAF}}_{s_{1.7}}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_1, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.7} \right\rangle$



(c) $\widetilde{\mathcal{RAF}}_{s_{1.9}}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_1, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.9} \right\rangle$

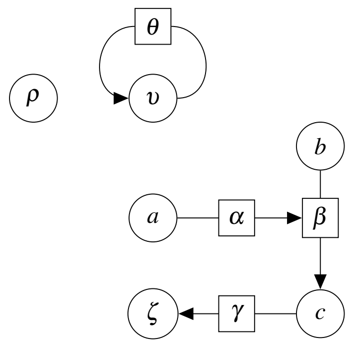


(d) $af'_{1.3}$ the induced AF of κ_1 under $\mu_{1.3}$

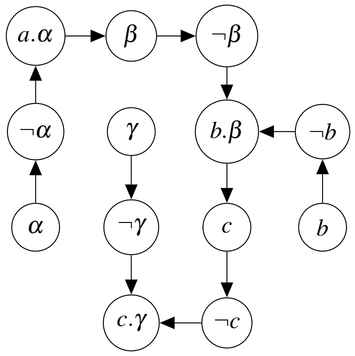
Figure 14.20: When neither c nor γ is $\mathtt{out}$ and one of them is $\mathtt{und}$

With:

- $\mu_{1.3} = \{(c.\gamma, \mathtt{und})\}$
- $\mathcal{L}^{inp}_{1.3} = \langle\{(c, \mathtt{in})\}, \{(\gamma, \mathtt{und})\}\rangle$
- $\mathcal{L}^{inp}_{1.7} = \langle\{(c, \mathtt{und})\}, \{(\gamma, \mathtt{in})\}\rangle$
- $\mathcal{L}^{inp}_{1.9} = \langle\{(c, \mathtt{und})\}, \{(\gamma, \mathtt{und})\}\rangle$



(a) $\widetilde{\mathcal{RAF}}_{s_{2.1}}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_2, \mathfrak{I}_2, \mathcal{L}^{inp}_{2.1} \right\rangle$



(b) $af'_{2.1}$ the induced AF of κ_2 under $\mu_{2.1}$

Figure 14.21: When a is accepted

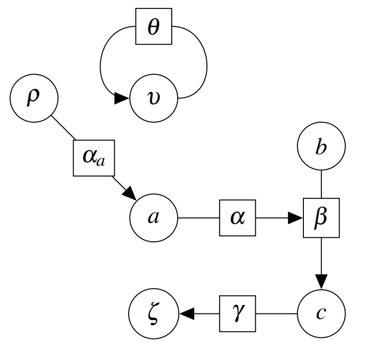
With:

- $\mathcal{L}^{inp}_{2.1} = \langle \{(a, in)\}, \varnothing \rangle$
- $\mu_{2.1} = \{(\neg a, out)\}$

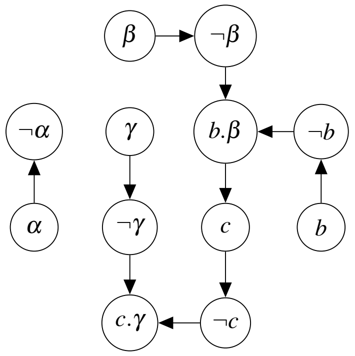
- $\mathcal{L}_{co\text{-}raf}(\widetilde{\mathcal{RAF}}_{s_{1.1}}) = \left\{ \left\langle \begin{array}{c} \{(a, in), (d, in), (c, in), (\rho, in), (\upsilon, und), (\zeta, in)\}, \\ \{(\delta, out), (\gamma, in), (\theta, in)\} \end{array} \right\rangle \right\}$
- $\mathcal{L}_{co}^{\mu_{1.1}(\kappa_1)} = \{\{(d, in), (\neg d, out), (\delta, out), (\neg\delta, in), (d.\delta, out), (a, in), (\neg a, out)\}\}$

Let consider Figure 14.19 on page 146. We have:

- $\mathcal{L}_{co\text{-}raf}(\widetilde{\mathcal{RAF}}_{s_{1.2}}) = \left\{ \left\langle \begin{array}{c} \{(a, out), (d, in), (c, in), (\rho, in), (\upsilon, und), (\zeta, in)\}, \\ \{(\delta, in), (\gamma, out), (\theta, in), (\alpha_\gamma, in)\} \end{array} \right\rangle \right\}$
- $\mathcal{L}_{co\text{-}raf}(\widetilde{\mathcal{RAF}}_{s_{1.4}}) = \left\{ \left\langle \begin{array}{c} \{(a, out), (d, in), (c, out), (\rho, in), (\upsilon, und), (\zeta, in)\}, \\ \{(\delta, in), (\gamma, in), (\theta, in), (\alpha_c, in)\} \end{array} \right\rangle \right\}$
- $\mathcal{L}_{co\text{-}raf}(\widetilde{\mathcal{RAF}}_{s_{1.5}}) = \left\{ \left\langle \begin{array}{c} \{(a, out), (d, in), (c, out), (\rho, in), (\upsilon, und), (\zeta, in)\}, \\ \{(\delta, in), (\gamma, out), (\theta, in), (\alpha_\gamma, in), (\alpha_c, in)\} \end{array} \right\rangle \right\}$
- $\mathcal{L}_{co\text{-}raf}(\widetilde{\mathcal{RAF}}_{s_{1.6}}) = \left\{ \left\langle \begin{array}{c} \{(a, out), (d, in), (c, out), (\rho, in), (\upsilon, und), (\zeta, in)\}, \\ \{(\delta, in), (\gamma, und), (\theta, in), (\alpha_\gamma, in), (\alpha_c, in)\} \end{array} \right\rangle \right\}$
- $\mathcal{L}_{co\text{-}raf}(\widetilde{\mathcal{RAF}}_{s_{1.8}}) = \left\{ \left\langle \begin{array}{c} \{(a, out), (d, in), (c, und), (\rho, in), (\upsilon, und), (\zeta, in)\}, \\ \{(\delta, in), (\gamma, out), (\theta, in), (\alpha_\gamma, in), (\alpha_c, in)\} \end{array} \right\rangle \right\}$



(a) $\widetilde{\mathcal{RAF}}_{s_{2.2}}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_2, \mathfrak{I}_2, \mathcal{L}_{2.2}^{inp} \right\rangle$



(b) $af'_{2.2}$ the induced AF of κ_2 under $\mu_{2.2}$

Figure 14.22: When a is rejected

With:

- $\mathcal{L}_{2.2}^{inp} = \langle \{(a, out)\}, \varnothing \rangle$
- $\mu_{2.2} = \{(\neg a, in)\}$

- $\mathscr{L}_{co}^{\mu_{1.2}(\kappa_1)} = \{\{(d, in), (\neg d, out), (\delta, in), (\neg\delta, out), (d.\delta, in), (a, out), (\neg a, in)\}\}$

Let consider Figure 14.20 on page 147. We have:

- $\mathscr{L}_{co\text{-}raf}(\widetilde{\mathcal{RAF}}_{s_{1.3}}) = \left\{ \left\langle \begin{array}{c} \{(a, und), (d, in), (c, in), (\rho, in), (\upsilon, und), (\zeta, in)\}, \\ \{(\delta, und), (\gamma, und), (\theta, in), (\alpha_\gamma, in)\} \end{array} \right\rangle \right\}$

- $\mathscr{L}_{co\text{-}raf}(\widetilde{\mathcal{RAF}}_{s_{1.7}}) = \left\{ \left\langle \begin{array}{c} \{(a, und), (d, in), (c, und), (\rho, in), (\upsilon, und), (\zeta, in)\}, \\ \{(\delta, und), (\gamma, in), (\theta, in), (\alpha_c, in)\} \end{array} \right\rangle \right\}$

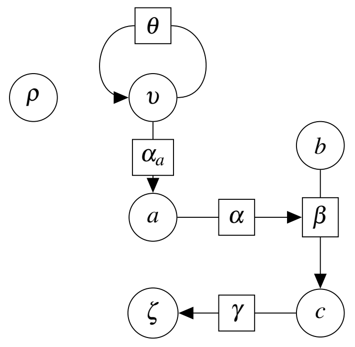
- $\mathscr{L}_{co\text{-}raf}(\widetilde{\mathcal{RAF}}_{s_{1.9}}) = \left\{ \left\langle \begin{array}{c} \{(a, und), (d, in), (c, und), (\rho, in), (\upsilon, und), (\zeta, in)\}, \\ \{(\delta, und), (\gamma, und), (\theta, in), (\alpha_\gamma, in), (\alpha_c, in)\} \end{array} \right\rangle \right\}$

- $\mathscr{L}_{co}^{\mu_{1.3}(\kappa_1)} = \{\{(d, in), (\neg d, out), (\delta, und), (\neg\delta, und), (d.\delta, und), (a, und), (\neg a, und)\}\}$

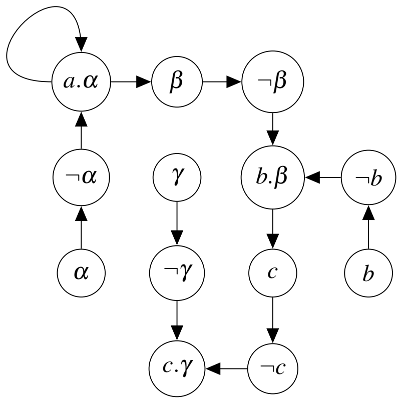
Let consider Figure 14.21 on the previous page. We have:

- $\mathscr{L}_{co\text{-}raf}(\widetilde{\mathcal{RAF}}_{s_{2.1}}) = \left\{ \left\langle \begin{array}{c} \{(a, in), (b, in), (c, in), (\rho, in), (\upsilon, und), (\zeta, out)\}, \\ \{(\alpha, in), (\beta, out), (\gamma, in), (\theta, in)\} \end{array} \right\rangle \right\}$

- $\mathscr{L}_{co}^{\mu_{2.1}(\kappa_2)} = \left\{ \left\{ \begin{array}{c} (\alpha, in), (\neg\alpha, out), (a.\alpha, in), (\beta, out), (\neg\beta, in), (b.\beta, out), \\ (b, in), (\neg b, out), (c, in), (\neg c, out), (\gamma, in), (\neg\gamma, out), (c.\gamma, in) \end{array} \right\} \right\}$



(a) $\widetilde{\mathcal{RAF}}_{s_{2.3}}$ corresponding to $\left\langle \widetilde{\mathcal{RAF}}_2, \mathfrak{I}_2, \mathcal{L}_{2.3}^{inp} \right\rangle$



(b) $af'_{2.3}$ the induced AF of κ_2 under $\mu_{2.3}$

Figure 14.23: When a is undecided

With:

- $\mathcal{L}_{2.3}^{inp} = \langle \{(a, \mathit{und})\}, \varnothing \rangle$
- $\mu_{2.3} = \{(\neg a, \mathit{und})\}$

Let consider Figure 14.22 on the previous page. We have:

- $\mathscr{L}_{co\text{-}raf}(\widetilde{\mathcal{RAF}}_{s_{2.2}}) = \left\{ \left\langle \begin{array}{c} \{(a, \mathit{out}), (b, \mathit{in}), (c, \mathit{out}), (\rho, \mathit{in}), (\upsilon, \mathit{und}), (\zeta, \mathit{in})\}, \\ \{(\alpha, \mathit{in}), (\beta, \mathit{in}), (\gamma, \mathit{in}), (\theta, \mathit{in}), (\alpha_a, \mathit{in})\} \end{array} \right\rangle \right\}$
- $\mathscr{L}_{co}^{\mu_{2.2}(\kappa_2)} = \left\{ \left\{ \begin{array}{c} (\alpha, \mathit{in}), (\neg\alpha, \mathit{out}), (a.\alpha, \mathit{out}), (\beta, \mathit{in}), (\neg\beta, \mathit{in}), (b.\beta, \mathit{in}), \\ (b, \mathit{in}), (\neg b, \mathit{out}), (c, \mathit{out}), (\neg c, \mathit{in}), (\gamma, \mathit{in}), (\neg\gamma, \mathit{out}), (c.\gamma, \mathit{out}) \end{array} \right\} \right\}$

Let consider Figure 14.23. We have:

- $\mathscr{L}_{co\text{-}raf}(\widetilde{\mathcal{RAF}}_{s_{2.3}}) = \left\{ \left\langle \begin{array}{c} \{(a, \mathit{und}), (b, \mathit{in}), (c, \mathit{und}), (\rho, \mathit{in}), (\upsilon, \mathit{und}), (\zeta, \mathit{und})\}, \\ \{(\alpha, \mathit{in}), (\beta, \mathit{und}), (\gamma, \mathit{in}), (\theta, \mathit{in}), (\alpha_a, \mathit{in})\} \end{array} \right\rangle \right\}$
- $\mathscr{L}_{co}^{\mu_{2.3}(\kappa_2)} = \left\{ \left\{ \begin{array}{c} (\alpha, \mathit{in}), (\neg\alpha, \mathit{out}), (a.\alpha, \mathit{und}), (\beta, \mathit{und}), (\neg\beta, \mathit{und}), (b.\beta, \mathit{und}), \\ (b, \mathit{in}), (\neg b, \mathit{out}), (c, \mathit{und}), (\neg c, \mathit{und}), (\gamma, \mathit{in}), (\neg\gamma, \mathit{out}), (c.\gamma, \mathit{und}) \end{array} \right\} \right\}$

Notice that we have the following equalities:

- $$\begin{aligned} \mathscr{F}_{\sigma}^{raf}(\widetilde{\mathcal{RAF}}_{s_{1.1}}, \mathfrak{I}_1, \mathcal{L}_{1.1}^{inp}) &= \left\{ \left\langle \ell\downarrow_{\tilde{A}_1}, \ell\downarrow_{\tilde{K}_1} \right\rangle | \ell \in \mathscr{L}_{co}^{\mu_{1.1}(\kappa_1)} \right\} \\ &= \{\langle \{(a, \mathit{in}), (d, \mathit{in})\}, \{(\delta, \mathit{out})\} \rangle\} \end{aligned}$$

- $$\begin{aligned}\mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_{s_{1.2}}, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.2}) &= \mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_{s_{1.4}}, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.4}) \\ &= \mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_{s_{1.5}}, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.5}) \\ &= \mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_{s_{1.6}}, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.6}) \\ &= \mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_{s_{1.8}}, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.8}) \\ &= \left\{\left\langle \ell\downarrow_{\tilde{A}_1}, \ell\downarrow_{\tilde{K}_1}\right\rangle | \ell \in \mathscr{L}^{\mu_{1.2}(\kappa_1)}_{co}\right\} \\ &= \{\langle\{(a, \mathtt{out}), (d, \mathtt{in})\}, \{(\delta, \mathtt{in})\}\rangle\}\end{aligned}$$

- $$\begin{aligned}\mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_{s_{1.3}}, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.3}) &= \mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_{s_{1.7}}, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.7}) \\ &= \mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_{s_{1.9}}, \mathfrak{I}_1, \mathcal{L}^{inp}_{1.9}) \\ &= \left\{\left\langle \ell\downarrow_{\tilde{A}_1}, \ell\downarrow_{\tilde{K}_1}\right\rangle | \ell \in \mathscr{L}^{\mu_{1.3}(\kappa_1)}_{co}\right\} \\ &= \{\langle\{(a, \mathtt{und}), (d, \mathtt{in})\}, \{(\delta, \mathtt{und})\}\rangle\}\end{aligned}$$

- $$\begin{aligned}\mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_{s_{2.1}}, \mathfrak{I}_2, \mathcal{L}^{inp}_{2.1}) &= \left\{\left\langle \ell\downarrow_{\tilde{A}_2}, \ell\downarrow_{\tilde{K}_2}\right\rangle | \ell \in \mathscr{L}^{\mu_{2.1}(\kappa_2)}_{co}\right\} \\ &= \{\langle\{(b, \mathtt{in}), (c, \mathtt{in})\}, \{(\alpha, \mathtt{in}), (\beta, \mathtt{out}), (\gamma, \mathtt{in})\}\rangle\}\end{aligned}$$

- $$\begin{aligned}\mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_{s_{2.2}}, \mathfrak{I}_2, \mathcal{L}^{inp}_{2.2}) &= \left\{\left\langle \ell\downarrow_{\tilde{A}_2}, \ell\downarrow_{\tilde{K}_2}\right\rangle | \ell \in \mathscr{L}^{\mu_{2.2}(\kappa_2)}_{co}\right\} \\ &= \{\langle\{(b, \mathtt{in}), (c, \mathtt{out})\}, \{(\alpha, \mathtt{in}), (\beta, \mathtt{in}), (\gamma, \mathtt{in})\}\rangle\}\end{aligned}$$

- $$\begin{aligned}\mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_{s_{2.3}}, \mathfrak{I}_2, \mathcal{L}^{inp}_{2.3}) &= \left\{\left\langle \ell\downarrow_{\tilde{A}_2}, \ell\downarrow_{\tilde{K}_2}\right\rangle | \ell \in \mathscr{L}^{\mu_{2.3}(\kappa_2)}_{co}\right\} \\ &= \{\langle\{(b, \mathtt{in}), (c, \mathtt{und})\}, \{(\alpha, \mathtt{in}), (\beta, \mathtt{und}), (\gamma, \mathtt{in})\}\rangle\}\end{aligned}$$

Finally, we can reunify compatible structure labellings of the partial RAFs of $\mathcal{RAF}$, ensuring that any produced labelling is valid:

$$\mathscr{L}_{co\text{-}raf}(\mathcal{RAF}) = \left\{\begin{array}{c}\langle\{(a, \mathtt{in}), (d, \mathtt{in}), (b, \mathtt{in}), (c, \mathtt{in})\}, \{(\delta, \mathtt{out}), (\alpha, \mathtt{in}), (\beta, \mathtt{out}), (\gamma, \mathtt{in})\}\rangle, \\ \langle\{(a, \mathtt{out}), (d, \mathtt{in}), (b, \mathtt{in}), (c, \mathtt{out})\}, \{(\delta, \mathtt{in}), (\alpha, \mathtt{in}), (\beta, \mathtt{in}), (\gamma, \mathtt{in})\}\rangle, \\ \langle\{(a, \mathtt{und}), (d, \mathtt{in}), (b, \mathtt{in}), (c, \mathtt{und})\}, \{(\delta, \mathtt{und}), (\alpha, \mathtt{in}), (\beta, \mathtt{und}), (\gamma, \mathtt{in})\}\rangle\end{array}\right\}$$

These structure labellings coincide with the valid labellings produced by the reunification of the cluster labellings of $\mathcal{AF}$:

$$\mathscr{L}_{co}(\mathcal{AF}) = \left\{ \begin{array}{c} \left\{ \begin{array}{c} (a, \mathit{in}), (\neg a, \mathit{out}), (d, \mathit{in}), (\neg d, \mathit{out}), (b, \mathit{in}), (\neg b, \mathit{out}), (c, \mathit{in}), (\neg c, \mathit{out}), \\ (\delta, \mathit{out}), (\neg\delta, \mathit{in}), (\alpha, \mathit{in}), (\neg\alpha, \mathit{out}), (\beta, \mathit{out}), (\neg\beta, \mathit{in}), (\gamma, \mathit{in}), (\neg\gamma, \mathit{out}), \\ (d.\delta, \mathit{out}), (a.\alpha, \mathit{in}), (b.\beta, \mathit{out}), (c.\gamma, \mathit{in}) \end{array} \right\}, \\ \left\{ \begin{array}{c} (a, \mathit{out}), (\neg a, \mathit{in}), (d, \mathit{in}), (\neg d, \mathit{out}), (b, \mathit{in}), (\neg b, \mathit{out}), (c, \mathit{out}), (\neg c, \mathit{in}), \\ (\delta, \mathit{in}), (\neg\delta, \mathit{out}), (\alpha, \mathit{in}), (\neg\alpha, \mathit{out}), (\beta, \mathit{in}), (\neg\beta, \mathit{out}), (\gamma, \mathit{in}), (\neg\gamma, \mathit{out}), \\ (d.\delta, \mathit{in}), (a.\alpha, \mathit{out}), (b.\beta, \mathit{in}), (c.\gamma, \mathit{out}) \end{array} \right\}, \\ \left\{ \begin{array}{c} (a, \mathit{und}), (\neg a, \mathit{und}), (d, \mathit{in}), (\neg d, \mathit{out}), (b, \mathit{in}), (\neg b, \mathit{out}), (c, \mathit{und}), (\neg c, \mathit{und}), \\ (\delta, \mathit{und}), (\neg\delta, \mathit{und}), (\alpha, \mathit{in}), (\neg\alpha, \mathit{out}), (\beta, \mathit{und}), (\neg\beta, \mathit{und}), (\gamma, \mathit{in}), (\neg\gamma, \mathit{out}), \\ (d.\delta, \mathit{und}), (a.\alpha, \mathit{und}), (b.\beta, \mathit{und}), (c.\gamma, \mathit{und}) \end{array} \right\} \end{array} \right\}$$

14.3.3 Properties

In this section, we prove the decomposability properties of RAF semantics from those of AF semantics. The first steps of the demonstration consist in highlighting labellings correspondence between RAF and flattened RAF and *w.r.t.* some partitioning. Figure 14.24 gives an overview of those steps, leading to Proposition 41 on page 159. From this property, the second steps of the demonstration consist in establishing equivalences between RAF and AF semantics decomposability properties (See Propositions 42, 43 and 45 on pages 160–161). Figure 14.25 on the next page gives an overview of those steps, leading ultimately to Proposition 47 on page 162.

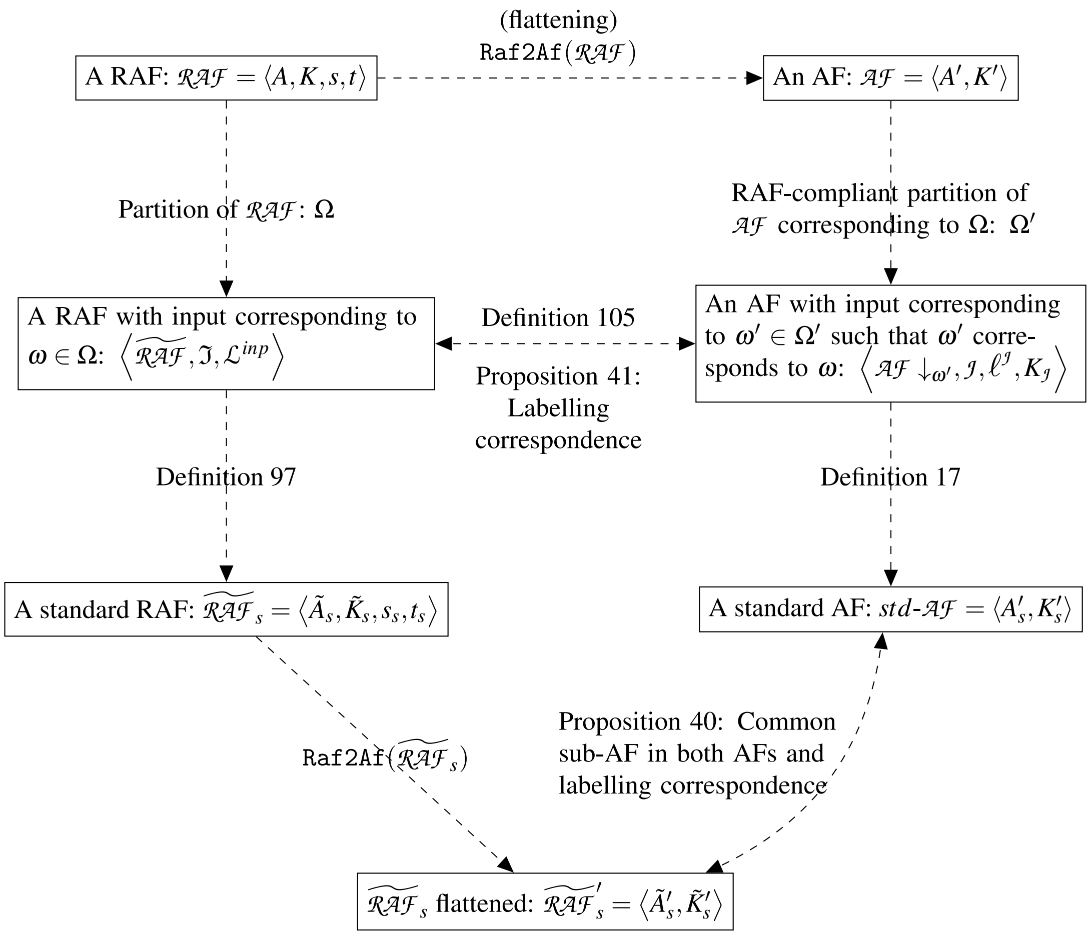

(flattening)
$\mathtt{Raf2Af}(\mathcal{RAF})$
A RAF: $\mathcal{RAF} = \langle A, K, s, t\rangle$
An AF: $\mathcal{AF} = \langle A', K'\rangle$
Partition of $\mathcal{RAF}$: Ω
RAF-compliant partition of $\mathcal{AF}$ corresponding to Ω: Ω'
A RAF with input corresponding to $\omega \in \Omega$: $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$
Definition 105
Proposition 41: Labelling correspondence
An AF with input corresponding to $\omega' \in \Omega'$ such that ω' corresponds to ω: $\left\langle \mathcal{AF}\downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$
Definition 97
Definition 17
A standard RAF: $\widetilde{\mathcal{RAF}}_s = \langle \tilde{A}_s, \tilde{K}_s, s_s, t_s\rangle$
A standard AF: $std\text{-}\mathcal{AF} = \langle A'_s, K'_s\rangle$
$\mathtt{Raf2Af}(\widetilde{\mathcal{RAF}}_s)$
Proposition 40: Common sub-AF in both AFs and labelling correspondence
$\widetilde{\mathcal{RAF}}_s$ flattened: $\widetilde{\mathcal{RAF}}'_s = \langle \tilde{A}'_s, \tilde{K}'_s\rangle$

Figure 14.24: Demonstration overview: schema n°1

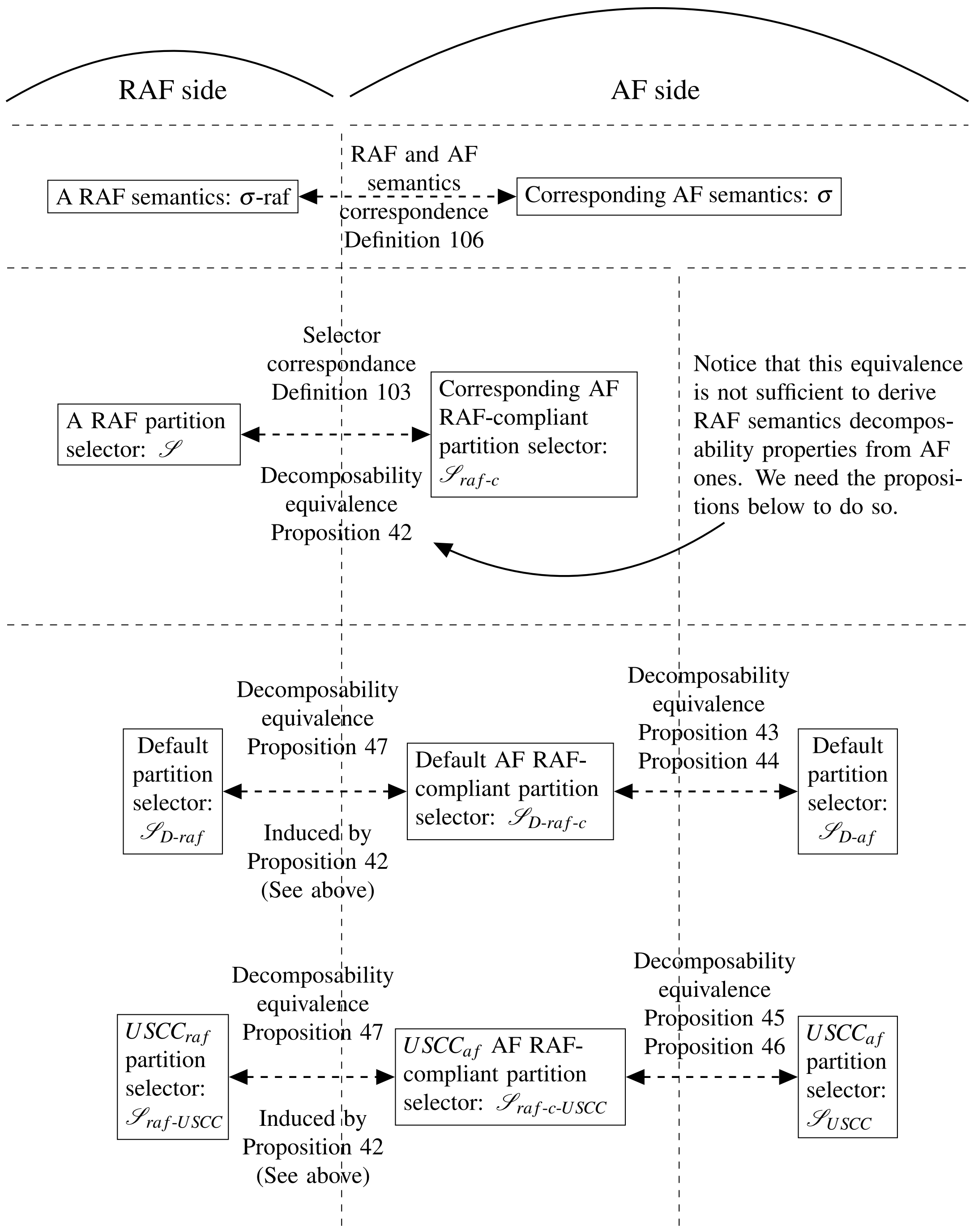

RAF side
AF side
RAF and AF semantics correspondence Definition 106
A RAF semantics: σ-raf
Corresponding AF semantics: σ
Selector correspondance Definition 103
A RAF partition selector: $\mathscr{S}$
Corresponding AF RAF-compliant partition selector: $\mathscr{S}_{raf\text{-}c}$
Decomposability equivalence Proposition 42
Notice that this equivalence is not sufficient to derive RAF semantics decomposability properties from AF ones. We need the propositions below to do so.
Decomposability equivalence Proposition 47
Default partition selector: $\mathscr{S}_{D\text{-}raf}$
Default AF RAF-compliant partition selector: $\mathscr{S}_{D\text{-}raf\text{-}c}$
Decomposability equivalence Proposition 43 Proposition 44
Default partition selector: $\mathscr{S}_{D\text{-}af}$
Induced by Proposition 42 (See above)
Decomposability equivalence Proposition 47
$USCC_{raf}$ partition selector: $\mathscr{S}_{raf\text{-}USCC}$
$USCC_{af}$ AF RAF-compliant partition selector: $\mathscr{S}_{raf\text{-}c\text{-}USCC}$
Decomposability equivalence Proposition 45 Proposition 46
$USCC_{af}$ partition selector: $\mathscr{S}_{USCC}$
Induced by Proposition 42 (See above)

Figure 14.25: Demonstration overview: schema n°2

In order to establish a point of comparison between RAF semantics decomposability and AF semantics decomposability we introduce with the following definition the correspondence between an AF with input and a RAF with input:

Definition 105 (AF with input corresponding to RAF with input). *Let $\mathcal{RAF} = \langle A,K,s,t\rangle$ be a RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be the corresponding AF of $\mathcal{RAF}$ (with $\mathcal{AF} = \langle A',K'\rangle$). Let Ω be a partition of $(A\cup K)$ and $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{AF})$ be the RAF-compliant partition of A' corresponding to Ω, i.e. $\Omega' = \{\omega' = \omega \cup \{\neg x | x \in \omega\} \cup \{s(\alpha).\alpha \in And_{A,K} | \alpha \in \omega\} | \omega \in \Omega\}$. Let $\omega \in \Omega$ and $\omega' \in \Omega'$ be its counterpart in $\mathcal{AF}$. Let $\widetilde{\mathcal{RAF}} = \langle \tilde{A},\tilde{K},\tilde{s},\tilde{t},s,t\rangle$ be the partial RAF corresponding to ω. Let $\mathfrak{I} = \left\langle S^{inp},Q^{inp}\right\rangle$ be the input elements of $\widetilde{\mathcal{RAF}}$ and $\mathcal{L}^{inp}$ be a structure labelling of them. Let $\left\langle \widetilde{\mathcal{RAF}},\mathfrak{I},\mathcal{L}^{inp}\right\rangle$ be a RAF with input. We define $\left\langle \mathcal{AF}\downarrow_{\omega'},\mathcal{J},\ell^{\mathcal{J}},K_{\mathcal{J}}\right\rangle$ as the AF with input corresponding to $\left\langle \widetilde{\mathcal{RAF}},\mathfrak{I},\mathcal{L}^{inp}\right\rangle$ with:*

- $\mathcal{J} = \{s(\alpha).\alpha | \alpha \in Q^{inp}\} \cup \{\neg a | a \in S^{inp}$ *s.t.* $\exists \alpha \in \tilde{K}$ *and* $\tilde{s}(\alpha) = \texttt{false}$ *and* $s(\alpha) = a\}$
- $\forall (s(\alpha).\alpha) \in \mathcal{J}$ *s.t.* $\alpha \in Q^{inp}$, $\ell^{\mathcal{J}}(s(\alpha).\alpha)$ *is defined as following:*

$$\ell^{\mathcal{J}}(s(\alpha).\alpha) = \begin{cases} \texttt{in} \iff \mathcal{L}^{inp}(\alpha) = \texttt{in} \text{ and } \mathcal{L}^{inp}(s(\alpha)) = \texttt{in} \\ \texttt{out} \iff \mathcal{L}^{inp}(\alpha) = \texttt{out} \text{ or } \mathcal{L}^{inp}(s(\alpha)) = \texttt{out} \\ \texttt{und} \iff \left(\begin{array}{l} \mathcal{L}^{inp}(\alpha) \neq \texttt{out} \text{ and } \mathcal{L}^{inp}(s(\alpha)) \neq \texttt{out} \text{ and} \\ (\mathcal{L}^{inp}(\alpha) = \texttt{und} \text{ or } \mathcal{L}^{inp}(s(\alpha)) = \texttt{und}) \end{array} \right) \end{cases}$$

- $\forall \neg a \in \mathcal{J}$ *s.t.* $a \in S^{inp}$, $\ell^{\mathcal{J}}(\neg a)$ *is defined as following:*

$$\ell^{\mathcal{J}}(\neg a) = \begin{cases} \texttt{in} \iff \mathcal{L}^{inp}(a) = \texttt{out} \\ \texttt{out} \iff \mathcal{L}^{inp}(a) = \texttt{in} \\ \texttt{und} \iff \mathcal{L}^{inp}(a) = \texttt{und} \end{cases}$$

- $K_{\mathcal{J}} = \{(s(\alpha).\alpha, t(\alpha)) | \alpha \in Q^{inp}\} \cup \{(\neg s(\alpha), s(\alpha).\alpha) | \alpha \in \tilde{K}$ *and* $\tilde{s}(\alpha) = \texttt{false}\}$

Example 75. *Let $\mathcal{RAF} = \langle A,K,s,t\rangle$ and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be respectively the RAF and the AF represented in Figure 14.26 on the next page. Let $\Omega' = \{\{d,\neg d,c,\neg c,\gamma,\neg\gamma,c.\gamma\},\{\delta,\neg\delta,d.\delta,a,\neg a,\alpha,\neg\alpha, a.\alpha,b,\neg b\}\}$ be a partition of A'. We have: $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{AF})$. Let $\Omega = \Omega' \cap (A\cup K)$ be the partition of $A\cup K$ corresponding to Ω'. Let us split $\mathcal{RAF}$ along the partition Ω and $\mathcal{AF}$ along the partition Ω'.*

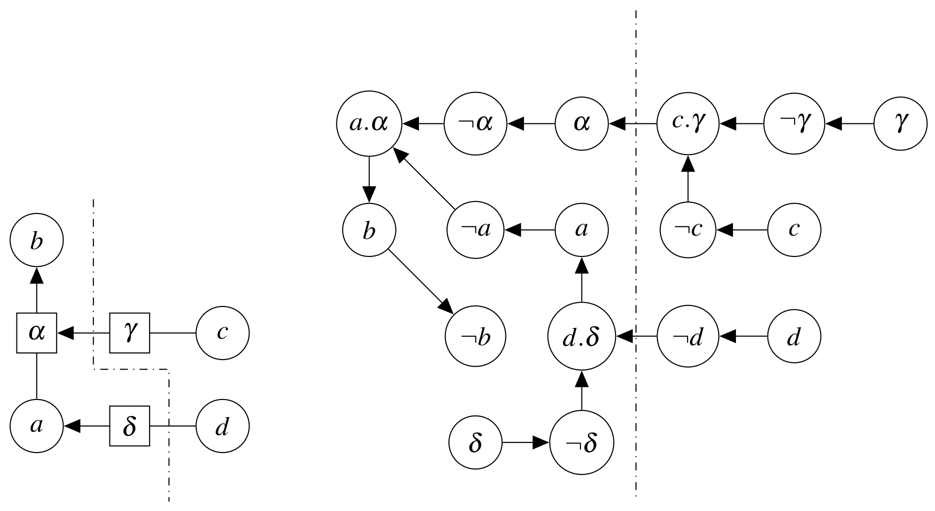
Let choose the left part of the RAF/AF for the illustration. We select thus: $\omega' = \{\delta,\neg\delta,d.\delta,a,\neg a,\alpha,\neg\alpha, a.\alpha,b,\neg b\}$. We have so: $\omega = \{\delta,a,\alpha,b\}$

Let $\widetilde{\mathcal{RAF}}_1 = \langle \tilde{A}_1,\tilde{K}_1,\tilde{s}_1,\tilde{t}_1,s,t\rangle$ be a partial RAF of $\mathcal{RAF}$ corresponding to ω, with:

- $A_1 = \{a,b\}$ *and* $K_1 = \{\alpha,\delta\}$
- $\tilde{s}_1(\delta) = \texttt{false}$ *and* $\tilde{s}_1(\alpha) = \texttt{true}$
- $\tilde{t}_1(\delta) = \texttt{true}$ *and* $\tilde{t}_1(\alpha) = \texttt{true}$

Let $\mathfrak{I}_1 = \langle\{c,d\},\{\gamma\}\rangle$ be the input corresponding to $\widetilde{\mathcal{RAF}}_1$, as represented in Figure 14.27 on the next page. Let as an example $\mathcal{L}_1^{inp} = \langle\{(c,\texttt{in}),(d,\texttt{und})\},\{(\gamma,\texttt{out})\}\rangle$ be a labelling of $\mathfrak{I}_1$.

Following Definition 105, $\left\langle \mathcal{AF}\downarrow_{\omega'},\mathcal{J},\ell^{\mathcal{J}},K_{\mathcal{J}}\right\rangle$ is the AF with input corresponding to $\left\langle \widetilde{\mathcal{RAF}}_1,\mathfrak{I}_1,\mathcal{L}_1^{inp}\right\rangle$ with:



(a) RAF example: $\mathcal{RAF}$ (b) $\mathcal{AF}$: the flattened version of $\mathcal{RAF}$

Figure 14.26: Decomposability illustration

- $\mathcal{I} = \{c.\gamma, \neg d\}$
- $\ell^{\mathcal{I}} = \{(c.\gamma, \mathtt{out}), (\neg d, \mathtt{und})\}$
- $K_{\mathcal{I}} = \{(c.\gamma, \alpha), (\neg d, d.\delta)\}$

Let std-$\mathcal{AF}$ be the standard AF corresponding to $\left\langle \mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$. Following the definition of standard AF (See Definition 17 on page 17) we have: std-$\mathcal{AF} = \langle A' \cup \mathcal{I}', K' \cup K'_{\mathcal{I}} \rangle$, where:

- $\mathcal{I}' = \mathcal{I} \cup \left\{ a' | a \in \mathcal{I} \cap \mathtt{out}(\ell^{\mathcal{I}}) \right\}$
- $K'_{\mathcal{I}} = K_{\mathcal{I}} \cup \left\{ (a', a) | a \in \mathcal{I} \cap \mathtt{out}(\ell^{\mathcal{I}}) \right\} \cup \left\{ (a, a) | a \in \mathcal{I} \cap \mathtt{und}(\ell^{\mathcal{I}}) \right\}$

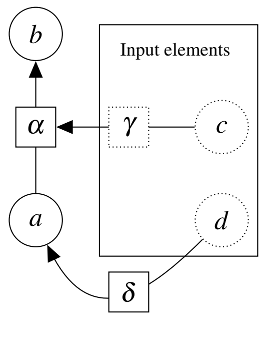


Figure 14.27: $\widetilde{\mathcal{RAF}}_1$

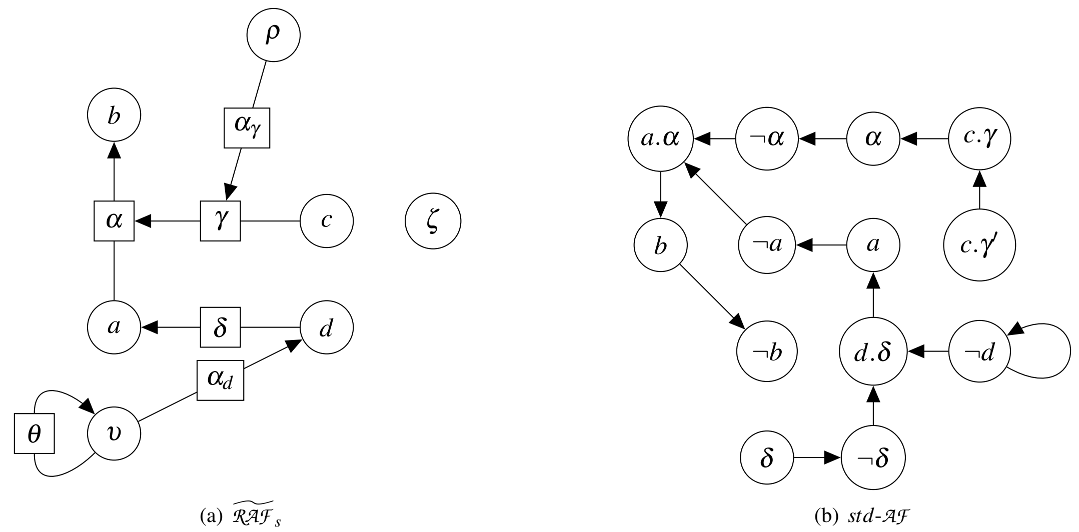


Figure 14.28: When c is accepted, γ rejected and d undefined

We have so: $\mathcal{I}' = \{c.\gamma, c.\gamma'\}$ *and* $K'_{\mathcal{I}} = \{(c.\gamma,\alpha),(c.\gamma',c.\gamma),(\neg d, d.\delta),(\neg d,\neg d)\}$.

Let $\widetilde{\mathcal{RAF}}_s$ *be the standard RAF corresponding to* $\left\langle \widetilde{\mathcal{RAF}}_1, \mathfrak{I}_1, \mathcal{L}_1^{inp}\right\rangle$ *(See Definition 97 on page 138).* $\widetilde{\mathcal{RAF}}_s$ *and* $std\text{-}\mathcal{AF}$ *are such represented in Figure 14.28.*

An important property is the fact that, given a RAF with input $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp}\right\rangle$ and its corresponding AF with input $\left\langle \mathcal{AF}\downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}}\right\rangle$, the flattening of the standard RAF associated with $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp}\right\rangle$ and the standard AF $std\text{-}\mathcal{AF}$ corresponding to $\left\langle \mathcal{AF}\downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}}\right\rangle$ will have a common sub-AF, attacked by same attackers (See Figure 14.29 on the next page).

Example 76. *Following Example 75, let consider* $\widetilde{\mathcal{RAF}}'_s$*, the RAF illustrated in Figure 14.29 on the next page. There is a common sub-RAF in* $\widetilde{\mathcal{RAF}}'_s$ *and in* $std\text{-}\mathcal{AF}$ *(See Figure 14.28(b)). Indeed, we have:* $\omega' = \{\delta,\neg\delta, d.\delta, a,\neg a,\alpha,\neg\alpha, a.\alpha, b,\neg b\}$, $\mathcal{I} = \{\neg d, c.\gamma\}$ *and we can verify that:*

- $\widetilde{\mathcal{RAF}}'_s\downarrow_{\omega'} = std\text{-}\mathcal{AF}\downarrow_{\omega'}$ *(all additional arguments are not concerned)*
- $\mathcal{I} \subseteq \tilde{K}'_s$ *and* $\mathcal{I} \subseteq K'_s$ *(the input arguments are present in both AFs. Here:* $c.\gamma$ *and* $\neg d$*)*
- $(\mathcal{I}\times\omega')\cap\tilde{K}'_s = (\mathcal{I}\times\omega')\cap K'_s$ *(the attacks from the input arguments are the same in both AFs)*

Moreover, arguments attacking this sub-AF are subject to same labellings, given a "*complete-based*" semantics.

The following example illustrates this fact:

Example 77. *Considering Examples 75 and 76, we have:*

$$\forall \ell \in \mathscr{L}_\sigma(\widetilde{\mathcal{RAF}}'_s), \forall \ell' \in \mathscr{L}_\sigma(std\text{-}\mathcal{AF}), \ell'(c.\gamma) = \ell(c.\gamma) = \mathtt{out} \text{ and } \ell'(\neg d) = \ell(\neg d) = \mathtt{und}$$

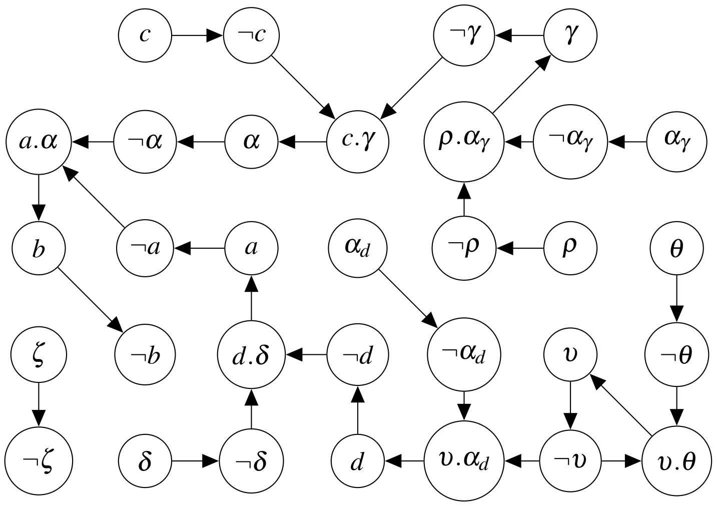


Figure 14.29: $\widetilde{\mathcal{RAF}}'_s$: the flattened version of $\widetilde{\mathcal{RAF}}_s$

As a consequence, restricted to the area of interest, both the standard AF of $\left\langle \mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$ and the flattening of the standard RAF associated with $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ will produce the same labellings as stated by Proposition 40.

Proposition 40. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \mathtt{Raf2Af}(\mathcal{RAF})$ be the corresponding AF of $\mathcal{RAF}$ (with $\mathcal{AF} = \langle A', K'\rangle$). Let $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ be a RAF with input of $\mathcal{RAF}$ and $\left\langle \mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$ be its corresponding AF with input, as defined in Definition 105 on page 155. Let $\widetilde{\mathcal{RAF}}_s = \langle \tilde{A}_s, \tilde{K}_s, s_s, t_s\rangle$ be the standard RAF of $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ and let std-$\mathcal{AF} = \langle A'_s, K'_s\rangle$ be the standard AF corresponding to $\left\langle \mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$. Let $\widetilde{\mathcal{RAF}}'_s = \mathtt{Raf2Af}(\widetilde{\mathcal{RAF}}_s)$ be the AF corresponding to the flattening of $\widetilde{\mathcal{RAF}}_s$ (with $\widetilde{\mathcal{RAF}}'_s = \langle \tilde{A}'_s, \tilde{K}'_s\rangle$). Let σ be an AF complete-based semantics. The following assertion holds:*

$$\{\ell \downarrow_{\omega' \cup \mathcal{I}} | \ell \in \mathscr{L}_\sigma(\widetilde{\mathcal{RAF}}'_s)\} = \{\ell \downarrow_{\omega' \cup \mathcal{I}} | \ell \in \mathscr{L}_\sigma(std\text{-}\mathcal{AF})\}$$

□ Proof of Proposition 40: link (See page 279).

Following Proposition 40, we can now establish the relation between the labellings of a partial RAF with input and the labellings of its corresponding AF with input for a given semantics, as made in Definition 106 and Proposition 41 on the following page.

Definition 106 (Semantics correspondence)**.** *Let σ be an AF semantics. Following the definitions of* `afLab2RafLab` *and* `rafLab2AfLab` *(See Definition 78 on page 118), we say that σ-raf is the RAF semantics corresponding to σ if and only if:*

$$\forall \mathcal{RAF} \in \Phi_{raf},\ \mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF}) = \{\mathtt{afLab2RafLab}(\ell) | \ell \in \mathscr{L}_\sigma(\mathtt{Raf2Af}(\mathcal{RAF}))\}$$

Or equivalently, if and only if:

$$\forall \mathcal{RAF} \in \Phi_{raf},\ \mathscr{L}_\sigma(\texttt{Raf2Af}(\mathcal{RAF})) = \{\texttt{rafLab2AfLab}(\mathcal{L}) | \mathcal{L} \in \mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF})\}$$

Proposition 41. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be the corresponding AF of $\mathcal{RAF}$ (with $\mathcal{AF} = \langle A', K'\rangle$). Let $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp}\right\rangle$ be a RAF with input of $\mathcal{RAF}$ and $\left\langle \mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}}\right\rangle$ be its corresponding AF with input, as defined in Definition 105 on page 155. Let σ be an AF complete-based semantics and let σ-raf be its corresponding counterpart for RAF. The following property holds:*

$$\{\mathcal{L} \downarrow_{\langle \tilde{A}, \tilde{K}\rangle} | \mathcal{L} \in \mathscr{L}_{\sigma\text{-}raf}(\widetilde{\mathcal{RAF}}_s)\} = \left\{\left\langle \ell\downarrow_A, \ell\downarrow_K\right\rangle \middle| \ell \in \mathscr{F}^{af}_\sigma(\mathcal{AF}\downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}})\right\}$$

Or equivalently that:

$$\mathscr{F}^{raf}_\sigma(\widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp}) = \left\{\left\langle \ell\downarrow_A, \ell\downarrow_K\right\rangle \middle| \ell \in \mathscr{F}^{af}_\sigma(\mathcal{AF}\downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}})\right\}$$

□ Proof of Proposition 41: link (See page 280).

In Definition 104 on page 142 has been defined the notion of decomposibity for RAF. Definition 107 refines this notion by making it relative to a given partition selector :

Definition 107 (Top-down, bottom-up and fully decomposability *w.r.t.* a RAF partition selector $\mathscr{S}$)**.** *Let $\mathscr{S}$ be a RAF partition selector. A RAF semantics σ is top-down decomposable w.r.t. $\mathscr{S}$ iff for any RAF $\mathcal{RAF}$ and any partition $\Omega = \{\omega_1, ..., \omega_n\} \in \mathscr{S}(\mathcal{RAF})$, it holds that:*

$$\mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF}) \subseteq \left\{\mathcal{L}_1 \cup ... \cup \mathcal{L}_n \middle| \mathcal{L}_i \in \mathscr{F}^{raf}_\sigma(\widetilde{\mathcal{RAF}}_i, \mathfrak{I}_i, \mathcal{L}^{inp}_i)\right\}$$
$$\text{With: } \widetilde{\mathcal{RAF}}_i \text{ built from } \omega_i \text{ and } \mathcal{L}^{inp}_i = (\textstyle\bigcup_{j\in\{1,...,n\} \text{ s.t. } j\neq i} \mathcal{L}_j)\downarrow_{\mathfrak{I}_i}$$

A RAF semantics σ is bottom-up decomposable w.r.t. $\mathscr{S}$ iff for any RAF and any partition $\Omega = \{\omega_1, ..., \omega_n\} \in \mathscr{S}(\mathcal{RAF})$, it holds that:

$$\mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF}) \supseteq \left\{\mathcal{L}_1 \cup ... \cup \mathcal{L}_n \middle| \mathcal{L}_i \in \mathscr{F}^{raf}_\sigma(\widetilde{\mathcal{RAF}}_i, \mathfrak{I}_i, \mathcal{L}^{inp}_i)\right\}$$
$$\text{With: } \widetilde{\mathcal{RAF}}_i \text{ built from } \omega_i \text{ and } \mathcal{L}^{inp}_i = (\textstyle\bigcup_{j\in\{1,...,n\} \text{ s.t. } j\neq i} \mathcal{L}_j)\downarrow_{\mathfrak{I}_i}$$

A RAF semantics is fully decomposable (or simply decomposable) w.r.t. a partition selector $\mathscr{S}$ iff it is both top-down and bottom-up decomposable w.r.t. $\mathscr{S}$.

`Note:` For a RAF semantics σ-raf, to be top-down (resp. bottom-up, fully) decomposable is equivalent to be top-down (resp. bottom-up, fully) decomposable w.r.t. the partition selector that produces all possible partition of a RAF.

Let formally defined this partition selector:

Definition 108. *$\mathscr{S}_{D\text{-}raf}$ is the RAF partition selector defined as follows:*

$$\forall \mathcal{RAF} \in \Phi_{raf}, \mathscr{S}_{D\text{-}raf}(\mathcal{RAF}) \text{ is the set of all possible partition of } \mathcal{RAF}$$

Now, in order to take advantage of the decomposability properties already proven for AF semantics, we have to show that the decomposability of RAF semantics *w.r.t.* a given partition selector $\mathscr{S}$ is equivalent to the decomposability property of the corresponding AF semantics *w.r.t.* a "RAF-compliant" version of $\mathscr{S}$. Following the relation stated in Proposition 41 on the previous page, we can establish the following proposition:

Proposition 42. *Let σ be an AF complete-based semantics and let σ-raf be the RAF semantics corresponding to σ. Let $\mathscr{S}$ be a RAF partition selector and let $\mathscr{S}_{raf\text{-}c}$ be the AF RAF-compliant partition selector corresponding to $\mathscr{S}$. The following properties holds:*

1. *σ-raf is top-down decomposable w.r.t. $\mathscr{S}$ iff σ is top-down decomposable w.r.t. $\mathscr{S}_{raf\text{-}c}$.*
2. *σ-raf is bottom-up decomposable w.r.t. $\mathscr{S}$ iff σ is bottom-up decomposable w.r.t. $\mathscr{S}_{raf\text{-}c}$.*
3. *σ-raf is fully decomposable w.r.t. $\mathscr{S}$ iff σ is fully decomposable w.r.t. $\mathscr{S}_{raf\text{-}c}$.*

□ Proof of Proposition 42: link (See page 281).

Proposition 42 is not sufficient to derive RAF semantics decomposability properties from AF ones. Indeed the equivalences are established between a RAF selector (whatever it is) and its AF RAF-compliant version. To access all the wanted properties, we have to show that, for AF semantics, having a certain decomposability property *w.r.t.* the default AF RAF-compliant partition selector $\mathscr{S}_{D\text{-}raf\text{-}c}$ is equivalent to having the same property *w.r.t.* the default AF partition selector $\mathscr{S}_{D\text{-}af}$ (See Definition 25 on page 20),[4] and having a certain decomposability property *w.r.t.* the $USCC_{af}$ RAF-compliant partition selector $\mathscr{S}_{raf\text{-}c\text{-}USCC}$ (See Definition 109 on the following page) is equivalent to having the same property *w.r.t.* the $USCC_{af}$ partition selector $\mathscr{S}_{USCC}$.

Proposition 43 establishes, for a given AF semanctics σ (so this property concerns AF), the equivalence between the decomposability properties of σ for any partition and the decomposability properties of σ *w.r.t.* $\mathscr{S}_{D\text{-}raf\text{-}c}$:

Proposition 43. *Let σ be an AF complete-based semantics. The following properties hold:*

1. *σ is top-down decomposable (equivalently w.r.t. $\mathscr{S}_{D\text{-}af}$) iff σ is top-down decomposable w.r.t. $\mathscr{S}_{D\text{-}raf\text{-}c}$.*
2. *σ is bottom-up decomposable (equivalently w.r.t. $\mathscr{S}_{D\text{-}af}$) iff σ is bottom-up decomposable w.r.t. $\mathscr{S}_{D\text{-}raf\text{-}c}$.*
3. *σ is fully decomposable (equivalently w.r.t. $\mathscr{S}_{D\text{-}af}$) iff σ is fully decomposable w.r.t. $\mathscr{S}_{D\text{-}raf\text{-}c}$.*

□ Proof of Proposition 43: link (See page 288).

From Proposition 43, new results concerning AF semantics are induced. Indeed, we can now have the decomposability properties *w.r.t.* $\mathscr{S}_{D\text{-}raf\text{-}c}$ of AF semantics, and state the following proposition:

Proposition 44. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be any RAF and $\mathcal{AF} = \mathtt{Raf2Af}(\mathcal{RAF})$ be the corresponding AF of $\mathcal{RAF}$. The semantics properties in Table 14.1 on the following page hold for the flattened RAF.*

□ Proof of Proposition 44: link (See page 289).

	co	*gr*	*pr*	*sst*	*st*
Full decomposability *w.r.t.* $\mathscr{S}_{D\text{-}raf\text{-}c}$	✓	×	×	×	✓
Top-down decomposability *w.r.t.* $\mathscr{S}_{D\text{-}raf\text{-}c}$	✓	✓	✓	×	✓
Bottom-up decomposability *w.r.t.* $\mathscr{S}_{D\text{-}raf\text{-}c}$	✓	×	×	×	✓

Table 14.1: AF Semantics decomposability properties *w.r.t.* $\mathscr{S}_{D\text{-}raf\text{-}c}$

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

Some AF semantics, as the *preferred* semantics, are not fully decomposable except *w.r.t.* the specific selector $\mathscr{S}_{USCC}$. To study the decomposability of such semantics for RAF, we have to define a selector equivalent to $\mathscr{S}_{USCC}$ for RAF and then its "RAF-compliant" version.

Definition 109. *(USCC RAF partition selector and AF correspondence) Let $\mathcal{RAF}$ be a RAF. Let $\mathscr{S}_{raf\text{-}USCC}$ be the RAF partition selector such that :*

$$\mathscr{S}_{raf\text{-}USCC}(\mathcal{RAF})$$
$$=$$
$$\{\Omega \mid \Omega \text{ is a partition of } \mathcal{RAF} \text{ and } \forall S \in SCCS_{raf}(\mathcal{RAF}), \exists \omega_i \in \Omega \text{ s.t. } \omega_i \cap S \neq \varnothing \implies S \subseteq \omega_i\}$$

Given a RAF $\mathcal{RAF} = \langle A, K, s, t\rangle$, we call "$USCC_{raf}$" a subset $S \subseteq A \cup K$ such that $S \in \mathscr{S}_{raf\text{-}USCC}(\mathcal{RAF})$. We define $\mathscr{S}_{raf\text{-}c\text{-}USCC}$ as the RAF-compliant AF partition selector corresponding to $\mathscr{S}_{raf\text{-}USCC}$.

Those selectors defined, we can establish a property similar to Proposition 43 on the previous page, but *w.r.t.* $\mathscr{S}_{USCC}$ and $\mathscr{S}_{raf\text{-}c\text{-}USCC}$.

Proposition 45. *Let σ be an AF complete-based semantics. The following properties hold:*

1. *σ is top-down decomposable w.r.t. $\mathscr{S}_{USCC}$ iff σ is top-down decomposable w.r.t. $\mathscr{S}_{raf\text{-}c\text{-}USCC}$.*
2. *σ is bottom-up decomposable w.r.t. $\mathscr{S}_{USCC}$ iff σ is bottom-up decomposable w.r.t. $\mathscr{S}_{raf\text{-}c\text{-}USCC}$.*
3. *σ is fully decomposable w.r.t. $\mathscr{S}_{USCC}$ iff σ is fully decomposable w.r.t. $\mathscr{S}_{raf\text{-}c\text{-}USCC}$.*

□ Proof of Proposition 45: link (See page 289).

From Proposition 45, new results concerning AF semantics are induced. Indeed, we can now have the decomposability properties *w.r.t.* $\mathscr{S}_{raf\text{-}c\text{-}USCC}$ of AF semantics, and state the following proposition:

Proposition 46. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \mathtt{Raf2Af}(\mathcal{RAF})$ be the corresponding AF of $\mathcal{RAF}$. The semantics properties in Table 14.2 on the next page hold for the flattened RAF.*

□ Proof of Proposition 46: link (See page 290).

[4]As a reminder, for an AF semantics σ, to be top-down (resp. bottom-up, fully) decomposable is equivalent to be top-down (resp. bottom-up, fully) decomposable *w.r.t.* $\mathscr{S}_{D\text{-}af}$.

	co	*gr*	*pr*	*sst*	*st*
Full decomposability *w.r.t.* $\mathscr{S}_{raf\text{-}c\text{-}USCC}$	✓	✓	✓	×	✓
Top-down decomposability *w.r.t.* $\mathscr{S}_{raf\text{-}c\text{-}USCC}$	✓	✓	✓	×	✓
Bottom-up decomposability *w.r.t.* $\mathscr{S}_{raf\text{-}c\text{-}USCC}$	✓	✓	✓	×	✓

Table 14.2: AF Semantics decomposability properties *w.r.t.* $\mathscr{S}_{raf\text{-}c\text{-}USCC}$

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

Finally, the way is now clear to access all AF decomposability properties and know the decomposability properties of RAF semantics, *w.r.t.* $\mathscr{S}_{raf\text{-}USCC}$ or not, as stated by Proposition 47 (result concerning RAF).

Proposition 47. *Let* $\mathcal{RAF} = \langle A, K, s, t\rangle$ *be any RAF. The semantics properties in Table 14.3 hold.*

□ Proof of Proposition 47: link (See page 290).

	RAF-*co*	RAF-*gr*	RAF-*pr*	RAF-*sst*	RAF-*st*
Full decomposability	✓	×	×	×	✓
Top-down decomposability	✓	✓	✓	×	✓
Bottom-up decomposability	✓	×	×	×	✓
Full decomposability *w.r.t.* $\mathscr{S}_{raf\text{-}USCC}$	✓	✓	✓	×	✓
Top-down decomposability *w.r.t.* $\mathscr{S}_{raf\text{-}USCC}$	✓	✓	✓	×	✓
Bottom-up decomposability *w.r.t.* $\mathscr{S}_{raf\text{-}USCC}$	✓	✓	✓	×	✓

Table 14.3: RAF Semantics decomposability properties

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

Chapter 15

Related Work

There are very few work related to the contributions made in this part:

- No *labelling* has previously been proposed for RAF.
- Before the one proposed in Chapter 12 on page 114, a flattening method to transform RAF into AF had already been introduced in [18]. It is discussed in Sections 9.2.2 and 12.1 on page 98 and on page 114.
- In [17], results about complexities in RAF are given on the $RAF\text{-}Cred_\sigma$ problem (for the *complete*, *preferred* and *stable* semantics) and on the $RAF\text{-}Skep_\sigma$ problem (for the *preferred* and *stable* semantics), but not for all RAF decision problems and semantics. Furthermore, proofs are not formally given.
- No work attempts to extend the notion of *Strongly Connected Component* to RAF and the *decomposability* of RAF semantics has not been studied before our contribution.

Part VI

Conclusion and Perspectives

This thesis has been for me a very rewarding experience, of course, in terms of research methodology but also in terms of human experience. From time to time, I had to cross long deserts, littered with failures and without any potential discoveries in sight. I learned that perseverance is the key for the success. I overcame all these hard times and, today, I am glad to conclude this thesis and present below all the contributions I have made during those three years of PhD. Afterward, I present the perspectives opened by my works.

Conclusion of the first milestone

In the first contribution part of my thesis (Part III on page 27), I address the issue of the *enumeration problem* solving time issue of Dung's AF semantics. I proposed a generic algorithm, so-called *AFDivider*, that computes *labellings* of the *complete*, *stable* and *preferred* semantics in a distributed and clustering-based fashion. This algorithm is generic in the sense that it can be used with any clustering method to split the AF into *clusters structures* and can be associated to any sound and complete procedure that computes the labellings of the different *clusters structure* for the wanted semantics. It has been proven to be complete and sound for all three mentioned semantics. That is, the algorithm produces all the expected labellings of the wanted semantics and each produced labelling is indeed a correct one.

Three clustering methods have been proposed, which lead to three *AFDivider* solver versions. The first one, so-called *AFDiv-spectral*, uses a *spectral clustering* method to split the AF. To the best of my knowledge, it is the first algorithm that uses this methods to compute semantic labellings. The second and the third ones, so-called *AFDiv-USCC-Tree* and *AFDiv-USCC-Chain*, split the AF following a *USCC* partition of it (that is, a partition that does not cut the AF SCC). They differ in the way the partition is selected.

An experimental analysis of those solvers has been conducted to identify their performances, compared to other solvers, according to AF types, over success rate and resolution time. It comes out from this study that the *AFDiv-spectral* variant is faster than most solvers in average on common successful instances while the *AFDiv-USCC-Tree* and *AFDiv-USCC-Chain* variants succeed to solve most instances when considering Barabási–Albert (BA), Erdős–Rényi (ER) and Watts-Strogatz (WS) AF types, for the *complete*, *stable* and *preferred* semantics.

Based on the AF types that they solve efficiently, I identify two behaviour classes among our three solvers: one for the spectral clustering and one for the USCC based clusterings. This shows that the clustering method which is used has an important effect on the performances of the *AFDivider* algorithm on a particular AF type. This is an important observation that must be taken into consideration for the development of new AF solvers.

The main advantage of the *AFDivider* algorithm I want to highlight is the fact that cutting the AF into clusters has the great advantage of limiting the solving hardness to the clusters. Indeed, in those other approaches, the combinatorial effect due to the number of labellings is propagated to the whole AF whereas, in the *AFDivider* algorithm, it is limited to the clusters. This property makes it well suited for non dense AF with a clustered structure.

The *Compact Enumeration Representation* I proposed takes full advantage of that fact. While it has been proven sufficient to answer all classical argumentation problems, experiments show that the *Compact Enumeration Representation* allows to solve more hard instances of some AF types such as BA, TR (Traffic) and BW (Block World). Furthermore, it produces an output much more faster than when enumerating all the labellings, especially for BA type AF (97.03 times much faster). The impact of the *Compact Enumeration Representation* is particularly significant for the *complete* semantics due to the huge number of labellings that causes memory overflow during the enumeration.

The *AFDivider* algorithm, with or without *Compact Enumeration Representation*, opens thus a new way to approach argumentation problems and many perspectives.

Perspectives of the first milestone

Here are some ideas to go further with the *AFDivider* algorithm:

- A recursive clustering version of this algorithm could be made. Indeed, after the cutting process, an induced AF could still be hard to solve. It may be possible that applying recursively the same clustering process on AF parts (until a certain criterion is satisfied) could enhance the global solving time.
- Other clustering methods than those tested could be more appropriate for some AF types. This could be studied in future works.
- The *Compact Enumeration Representation* could be exploited to answer non classical problems such as: “What is the labelling rate in which an argument a is labelled $\mathtt{in}$?”, “Is there an argument labelled $\mathtt{in}$ in more that 70% of the labellings ?”, “Is the set of arguments S is accepted (together) in more than 60% of the labellings” and other questions of that type, and that, without explicitly enumerating all the labellings, avoiding the costly cartesian product of component labellings.
- It would be interesting to extend this work to further generalizations of AFs, using several types of relation (not only attacks but also supports), with relation and argument strength, recursive relations, and so for more complex semantics.
- Recently, a new category of AF problems appeared in ICCMA competition: *Dynamic* AF semantics computation. Given an AF, it consists of computing some semantics of the initial AF in a first place, then of its altered version (arguments/attacks are added/removed from the initial AF). *Dynamic* algorithms use the previous computed result to compute the current altered AF semantics. It would be interesting to explore the possibilities for a *dynamic* version of *AFDivider*.
- Explainability is becoming essential for reliable IA information systems. In the domain of Abstract Argumentation, having an explanation for the acceptance or the reject of some argument, more concise and informative than a whole extension/labelling, has an important interest. A work could be made to study how the AF decomposition made by our algorithm (given some clustering method) could help to explain such things.

I would like to emphasize two major improvements of the *AFDivider* algorithm:

- The first would be to “know” in advance which clustering method (including ones other than those presented in this paper) should be used for a particular AF instance. Experiments could be conducted to learn, for example with a neural network, which one to use.
- The second, which is the most interesting, is to go further and have even the cutting process being supervised by a neural network trained to cluster AF instances following their structure. As a consequence, for any known AF type, the most appropriate clustering method would be used to solve each AF instance efficiently. The clustering might be in a certain sense dynamic, as the rules applied to cut may be different from one AF area to another.

Conclusion of the second milestone

In the second contribution part of my thesis (Part V on page 101), I created a certain number of tools for the study of RAF structure and RAF semantics complexity and properties, with the objective of proposing a first algorithm to compute RAF semantics.

In a first step, I define new semantics for RAF. The *semi-stable* semantics have been defined for RAF and it has been shown that RAF are still a conservative generalization of Dung's AF, even for the *semi-stable* semantics. The notion of labellings has been extended to RAF. Given a RAF, a *structure labelling* or *RAF labelling* is a tuple in which the first element is a labelling over its arguments and the second one is a labelling over its attacks. As for AF labellings, these *RAF labellings* are three-value based, indicating the degree of acceptance of a RAF element (*i.e.* accepted: `in`, rejected: `out` or undecided: `und`).

In a second step, RAF semantics have been redefined in terms of *RAF labelling* semantics. A one-to-one mapping between structures and RAF labellings have been identified using two linking functions: `Struct2Lab`, that transforms a structure into a *RAF labelling* and `Lab2Struct`, that transforms a *RAF labelling* into a structure. *Reinstatement RAF labellings* have been formally defined as coherent RAF labellings. It is shown (see Chapter 11 on page 107) that the *complete*, *grounded*, *preferred*, *semi-stable* and *stable* structure semantics correspond to precise types of *reinstatement RAF labellings*. Table 11.1 on page 111 gives the correspondence between structure semantics and reinstatement RAF labellings. Moreover, this work confirms that RAF are a conservative generalization of Dung's AF. Indeed in RAF with no recursive attacks, there is a one-to-one mapping between reinstatement labellings (AF notion) and two RAF notions (structures and reinstatement RAF labellings) for the *complete*, *grounded*, *preferred*, *semi-stable* and *stable* semantics. This additional precision on the acceptability status of a RAF element, from the information a simple structure can give (that is, a binary acceptability status: given a structure, whether an element belongs to it or not), opens a whole new field of research for RAF solving algorithms.

In a third step, I defined a new *flattening* process that transforms a RAF into an AF and this, while preserving the meaning of the RAF defeat relation, ensuring thus interesting properties such as shape or acceptability related properties that will be used in the following steps. This transformation is not only polynomial in time but also logarithmic in space. Although it has not been used in this way in this thesis, it allows the use of AF solvers to solve RAF semantics problems.

In a fourth step, *decision problems* for RAF have been defined and their *complexity* studied. Using the *flattening* process, an important result has been found: the complexities of RAF decision problems are the same as the ones in Dung's framework, despite all the additional expressiveness that is brought by the higher-order attacks.

In a fifth step, the notion of *Strong Connected Component* (SCC) has been extended to RAF. A bijection has been shown between the SCC_{raf} of a given RAF and the SCC_{af} of its flattened version. Then the *decomposability* properties of RAF semantics have been studied. It has been shown that the decomposability properties of RAF semantics are equivalent to the ones of their corresponding AF semantics.

Perspectives of the second milestone

As perspectives for this milestone, here is a list of interesting ideas to explore:

- More AF semantics could be extended to RAF, whether structure-based or labelling-based semantics.
- The SCC-recursiveness [11] of RAF semantics could be studied, based on the *flattening* process proposed.

- Other notions of graph theory, such as tree-width, could be extended to RAF, allowing thus the study of other types of properties.
- RAF generator following different structure types as well as benchmarks for RAF semantics computation could be given, as no work has been done to address those questions so far.
- Beside decision problems, other problems are of interest for argumentation frameworks, whether they have higher-order attacks or not: function problems.[1] The functional counterpart of $Cred_\sigma$ and $Exists_\sigma^{\neg\varnothing}$ may turn to be particularly useful in the context of a dialogue between agents, the output being here the concerned acceptable set. Defining such problems, and investigating their complexity, would be interesting.
- All those contributions pave the way for an algorithmic investigation of the computation of RAF semantics and RAF decision problem solving. A sound and complete *AFDivider*-like algorithm for RAF could be proposed.

[1]In computational complexity theory, a function problem is a computational problem where a single output is expected for every input, but the output is more complex than that of a decision problem: it is not simply "yes" or "no".

Appendix 1: Mathematical Background

Chapter 16

Mathematical Theories

In this chapter are presented firstly, basic notions of *Set Theory* (Section 16.1). Secondly, some notions of *Graph Theory* (Section 16.2). Thirdly, definitions are given about matrices (Section 16.3 on page 175). Fourthly, the *computational complexity theory* is presented in Section 16.4 on page 178. Finally, basic notions of *algorithm analysis* are presented in Section 16.5 on page 184. Notice that the scope of this background is limited to the necessary. For an overview of set theory see [45], of graph theory see [48], of matrix computation see [47], of complexity theory see [28] and of algorithm analysis see [66].

16.1 Set theory

The notions of *set* and *set partition* are used throughout this document.

Definition 110 (Set)**.** *A set is a collection of distinct elements.*

Definition 111 (Partition of a set)**.** *A partition* $\Omega = \{\omega_1, ..., \omega_n\}$ *of a set* O *is a set of subsets of* O *such that:*

- $\forall i, j \in \{1, ..., n\}$ *s.t.* $i \neq j$, $\omega_i \cap \omega_j = \varnothing$
- $\bigcup_{i \in \{1,...,n\}} \omega_i = \mathrm{O}$

16.2 Graph theory

In this section different types of graph are presented, and also notions related to nodes, relations, paths, subgraphs and topology.

16.2.1 Graph types

Definition 112. *(Non-directed and directed graph). A non-directed (respectively directed) graph is an ordered pair* $G = (V, E)$ *where:*

- V *is a set whose elements are called nodes or vertices;*

- *E is a set of unordered (respectively ordered) pairs of vertices called non-directed edges (respectively directed edges).*

Example 78. *Figure 16.1 shows an example of a directed and a non-directed graph.*

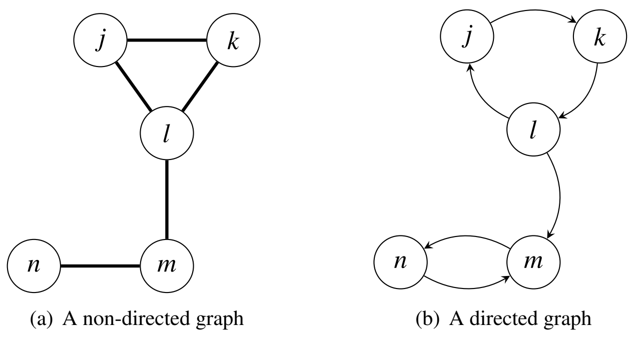


Figure 16.1: Example of graphs

`Note:` *E is a set. As a consequence, in this document non-directed (respectively directed) graphs have only distinct unordered (respectively ordered) pairs in E. Non-directed (respectively directed) multigraphs, non-directed (respectively directed) graphs in which is permitted to have multiple non-directed (respectively directed) edges that have the same endpoints,*[1] *are not considered.*

Definition 113. *(Weighted graph). A weighted directed (respectively non-directed) graph is an ordered pair $G = (V,E,W)$ where:*

- *(V,E) is a directed (respectively non-directed) graph.*
- *$W : E \to \mathbb{R}$ is a total function that associates a weight to each directed (respectively non-directed) edge in E.*

Example 79. *Figure 16.2 on the next page shows examples of weighted graphs.*

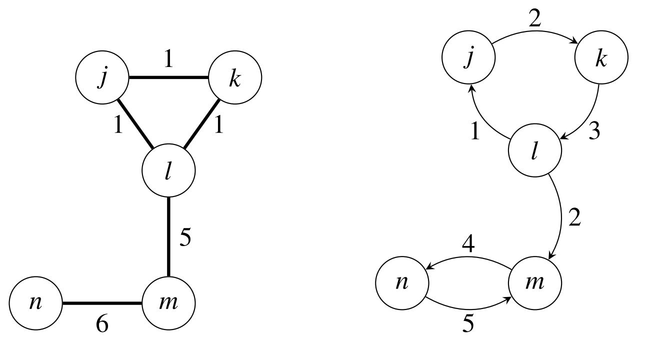
In the following we will implicitly consider that a non-weighted directed (respectively non-directed) graph $G = (V,E)$ is a weighted directed (respectively non-directed) graph whose edges are weighed 1.

16.2.2 Node and edge relations

Definition 114 (Incidence)**.** *Let $G = (V,E)$ be a graph (directed or not), and* $\mathrm{e} = (v_i, v_j) \in E$ *be an edge. We say that* e *is incident to v_i and v_j, or joins v_i and v_j. Similarly, v_i and v_j are incident to* e.

Definition 115 (Adjacency)**.** *Let $G = (V,E)$ be a graph (directed or not), and $v_i \in V$, $v_j \in V$ be two nodes of G. We say that v_i and v_j are adjacent if $(v_i, v_j) \in E$ or $(v_j, v_i) \in E$.*

[1]The endpoints of an edge are the vertices incident to it. See Definition 114.



(a) A non-directed weighted graph

(b) A directed weighted graph

Figure 16.2: Example of weighted graphs

Definition 116 (Degree in a non-directed graph)**.** *Let $G = (V,E)$ be a non-directed graph and $v \in V$ be a vertex. The degree of v in G, noted deg(G,v), is its number of incident edges.*

Definition 117 (Degree in a directed graph)**.** *In directed graphs three degrees are associated to each vertex. Let $G = (V,E)$ be a directed graph and $v \in V$ be a vertex. We have:*

- *The* inward degree *of v, denoted in-*$\deg(G,v)$*, is its number of incident edges such that v is the second element of the edge.*
- *The* outward degree *of v, denoted out-*$\deg(G,v)$*, is its number of incident edges such that v is the first element of the edge.*
- *The* degree *of v, denoted deg(G,v), is its number of incident edges. That is: deg$(G,v) =$ out-*$\deg(G,v) +$ *in-*$\deg(G,v)$.

Example 80. *Let consider Figure 16.1 on the previous page. In both graphs:*

- *m and n are incident to* $\mathrm{e} = (m,n)$ *and vice versa*
- *m and n are adjacent*

In the non-directed graph, we have: deg$(G,m) = 2$.
In the directed graph, we have: deg$(G,m) = 3$, in-$\deg(G,m) = 2$ *and out-*$\deg(G,m) = 1$.

Definition 118. *(Weighted degree). Let $G = (V,E,W)$ be a weighted graph (directed or not). Let $v \in V$ be a vertex and I the set of its incident edges. We define the weighted degree of v, noted* $\deg_w(G,v)$*, as the weight sum of its incident edges:*

$$\deg_w(G,v) = \sum_{e \in I} W(e)$$

`Note:` *In order to simplify the notation, deg(G,v) (resp.* $\deg_w(G,v)$*) will be noted deg(v) (resp.* $\deg_w(v)$*) when there is no ambiguity about the graph in which the degree is measured.*

Example 81. *Let consider Figure 16.2. In both graphs we have: deg$(m) = 11$.*

16.2.3 Connectivity

Definition 119 (Walk). *Let $G = (V,E)$ be a graph (directed or not). A walk is a sequence $(v_1,...,v_n)$ such that:*

- $\forall i \in \{1,...,n\}$, $v_i \in V$
- $\forall i \in \{1,...,n-1\}$, $(v_i,v_{i+1}) \in E$

Definition 120 (Non-directed walk). *Let $G = (V,E)$ be a directed graph. A non-directed walk is a sequence $(v_1,...,v_n)$ such that:*

- $\forall i \in \{1,...,n\}$, $v_i \in V$
- $\forall i \in \{1,...,n-1\}$, $(v_i,v_{i+1}) \in E$ *or* $(v_{i+1},v_i) \in E$

In this document, we will restrict the notion of graph paths to strict simple paths (*i.e.* they contains only distinct elements).

Definition 121 (Path). *Let $G = (V,E)$ be a graph (directed or not). A path is a walk which contains distinct vertices.*

Definition 122 (Non-directed path). *Let $G = (V,E)$ be a directed graph. A non-directed path is a non-directed walk which contains distinct vertices.*

Definition 123 (Cycle). *Let $G = (V,E)$ be a graph (directed or not). A cycle is a sequence $(v_1,...,v_n)$ with $n \geq 2$ such that:*

- $(v_2,...,v_n)$ *is a path*
- $(v_1,...,v_{n-1})$ *is a path*
- $v_1 = v_n$

Example 82. *Let consider Figure 16.1(b) on page 171. (l, j, k, l, m) is a walk, (j, k, l) is a path and (j, k, l, j) is a cycle.*

Definition 124 (Connected and disconnected graph). *Let $G = (V,E)$ be a directed (respectively non-directed) graph. G is a* connected *graph if, for all distinct vertices $v_i \in V$ and $v_j \in V$, there exists a non-directed path* p *(respectively a path* p*) in G s.t. v_i is the first vertex of* p *and v_j is the last vertex of* p*. Otherwise the graph is called a* disconnected *graph.*

Definition 125 (Subgraph). *Let $G = (V,E)$ be a directed graph (respectively non-directed graph). A subgraph* $\mathrm{S} = (V',E')$ *of G is a directed graph (respectively non-directed graph) such that:*

- $V' \subseteq V$.
- $E' \subseteq E$.
- $\forall (v_i,v_j) \in E', v_i \in V'$ *and* $v_j \in V'$.

Definition 126 (Graph restriction $\downarrow$). *Let $G = (V,E)$ be a graph and $S \subseteq V$ be a set of vertices. The* restriction *of G to S is the subgraph of G defined as $G \downarrow_S \equiv (S, E \cap (S \times S))$.*

`Note:` *Notice that when restricting a graph G to a set S of nodes, any edge of G whose endpoints are both in S must be kept. It is not the case for the more general subgraph definition. Indeed, a subgraph of G whose set of nodes coincides with S may not keep all these edges.*

Example 83. *Let consider the graph $G = (V,E)$ in Figure 16.1(b) on page 171. We have: $G\downarrow_{\{m,n\}} = (\{m,n\},\{(m,n),(n,m)\})$.*

Definition 127 (Connected component)**.** *Let G be a graph (directed or not). Let H be a subgraph of G such that:*

- *H is connected.*
- *H is not contained in any connected subgraph of G which has more vertices or edges than H has.*

Then H is a connected component of G.

In the following by "component" we mean "connected component".

Definition 128 (Path-equivalence relation)**.** *Let $G = (V,E)$ be a directed graph. The binary relation of path-equivalence between nodes, denoted as $PE_G \subseteq (V \times V)$, is defined as follows:*

- $\forall v_i \in V, (v_i, v_i) \in PE_G$.
- *given two distinct nodes $v_i, v_j \in V, (v_i, v_j) \in PE_G$ if and only if there is a path from v_i to v_j and a path from v_j to v_i.*

Definition 129 (SCC)**.** *The strongly connected components of a directed graph G are the equivalence classes of nodes under the relation of path-equivalence. Basically, an SCC is a (directed) subgraph in which there is a path between each pair of its vertices.*

Example 84. *Let consider the graph in Figure 16.1(b) on page 171. $\{m,n\}$ and $\{j,k,l\}$ are SCCs.*

16.2.4 Topology

Several contributions made in this thesis have been inspired by ideas from graph topologies (that is their "*shape*") and especially about *"clusters"* of nodes in graphs. Formally, *clusters* can be defined as follows:

Definition 130 (Cluster)**.** *Let G be a graph. A cluster of G is a connected subgraph of G.*

To express the gathering, the connectivity between the nodes we use the notion of *relation density*:

Definition 131 (Relation density)**.** *Let $G = (V,E)$ be a graph, S be a subset of V. The relation density $\mathscr{R}_d(G)$ of the graph G is defined by:*

$$\mathscr{R}_d(G) = \frac{|E|}{|V|}$$

The relation density $\mathscr{R}_d(G\downarrow_S)$ of the subgraph $G\downarrow_S$ is defined by:

$$\mathscr{R}_d(G\downarrow_S) = \frac{|E \cap (S \times S)|}{|S|}$$

In practice, given an initial graph, we will be interested by some of its connected subgraphs which have similar sizes (number of nodes) and such that their inside relation density is greater than their neighbouring relation density.

Example 85. *Following Example 83, we can consider $G\downarrow_{\{m,n\}}$ as a cluster. We have: $\mathscr{R}_d(G\downarrow_{\{m,n\}}) = 1$*

16.3 Matrices

In this section are presented notions on matrices, the key notions being the eigenvectors and values, and laplacian matrices.

Definition 132 (Matrix). *A matrix* M *with m lines and n columns, or a* $m \times n$ *matrix, with values in some field of scalars* K *is an application of* $\{1, 2, ..., m-1, m\} \times \{1, 2, ..., n-1, n\}$ *in* K . $\mathrm{M}_{i,j} \in \mathrm{K}$ *is the image of the couple* (i, j). *i is called the line index and j the column index.*

Definition 133 (Eigenvector and eigenvalue). *Let* O *be a vector space over some field* K *of scalars, let* u *be a linear transformation mapping* O *into* O *(i.e.* $\mathrm{u} : \mathrm{O} \to \mathrm{O}$*), and let* $v \in \mathrm{O}$ *be a non-zero vector.*

v is an eigenvector of u *if and only if there exists a scalar* $\lambda \in \mathrm{K}$ *such that:*

$$\mathrm{u}(v) = \lambda \cdot v$$

In this case λ *is called eigenvalue (associated with the eigenvector v).*

For more details on eigenvectors and eigenvalues see [56], Chapter 6.

Definition 134 (Adjacency matrix). *Let* $G = (V, E, W)$ *be a weighted non-directed graph. The adjacency matrix* M_a *of G is an* $n \times n$ *matrix (with* $n = |V|$*) defined as:*

$$(\mathrm{M}_a)_{i,j} = \begin{cases} W((v_i, v_j)) \text{ if } (v_i, v_j) \in E \\ 0 \text{ otherwise} \end{cases}$$

If the weights of a graph G represent similarity measures then adjacency matrix is called the similarity matrix of G.

Definition 135 (Degree matrix). *Given a weighted non-directed graph* $G = (V, E, W)$, *the degree matrix* M_d *for G is an* $n \times n$ *matrix (with* $n = |V|$*) defined as:*

$$(\mathrm{M}_d)_{i,j} = \begin{cases} deg_w(v_i) \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}$$

`Note:` M_d *is a diagonal matrix.*

Definition 136 (Laplacian matrix). *Given a weighted non-directed graph* $G = (V, E, W)$, *the laplacian matrix* M_l *for G is an* $n \times n$ *matrix (with* $n = |V|$*) defined as:*

$$\mathrm{M}_l = \mathrm{M}_d - \mathrm{M}_a$$

Example 86. *Let illustrate these notions while considering the non-directed graph* $G = (V, E, W)$ *in Figure 16.3 on the next page.*

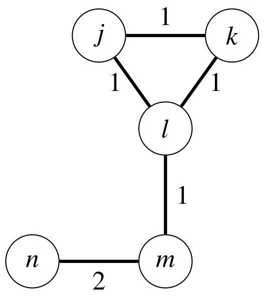


Figure 16.3: A weighted non-directed graph

The adjacency matrix M_a *of G is:*

$$\mathrm{M}_a = \begin{array}{c} \\ j \\ k \\ l \\ m \\ n \end{array} \begin{array}{c} \begin{array}{ccccc} j & k & l & m & n \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 2 & 0 \end{bmatrix} \end{array}$$

The degree matrix M_d *of G is:*

$$\mathrm{M}_d = \begin{array}{c} \\ j \\ k \\ l \\ m \\ n \end{array} \begin{array}{c} \begin{array}{ccccc} j & k & l & m & n \end{array} \\ \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} \end{array}$$

And then, its laplacian matrix M_l *is:*

$$\mathrm{M}_d - \mathrm{M}_a = \mathrm{M}_l = \begin{array}{c} \\ j \\ k \\ l \\ m \\ n \end{array} \begin{array}{c} \begin{array}{ccccc} j & k & l & m & n \end{array} \\ \begin{bmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 \\ 0 & 0 & -1 & 3 & -2 \\ 0 & 0 & 0 & -2 & 2 \end{bmatrix} \end{array}$$

Let $v \in \mathbb{R}^5$ be a vector, with:

$$v = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

Let u *be the linear transformation mapping $\mathbb{R}^5$ into $\mathbb{R}^5$ (i.e.* $\mathrm{u} : \mathbb{R}^5 \to \mathbb{R}^5$*), whose coefficients correspond to the matrix* M_l*, that is, the application whose definition corresponds to matrix multiplication between* M_l *and $v \in \mathbb{R}^5$.*

`Note:` In practice we do not search for the exact expression of u, but rather directly solve the system of equations corresponding to: $M_l \times v = \lambda \cdot v$, with v being a non-zero vector.

Given that:

$$\mathrm{M}_l \times v = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 \\ 0 & 0 & -1 & 3 & -2 \\ 0 & 0 & 0 & -2 & 2 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \times x_1 & - & 1 \times x_2 & - & 1 \times x_3 & + & 0 \times x_4 & + & 0 \times x_5 \\ -1 \times x_1 & + & 2 \times x_2 & - & 1 \times x_3 & + & 0 \times x_4 & + & 0 \times x_5 \\ -1 \times x_1 & - & 1 \times x_2 & + & 3 \times x_3 & - & 1 \times x_4 & + & 0 \times x_5 \\ 0 \times x_1 & + & 0 \times x_2 & - & 1 \times x_3 & + & 3 \times x_4 & - & 2 \times x_5 \\ 0 \times x_1 & + & 0 \times x_2 & + & 0 \times x_3 & - & 2 \times x_4 & + & 2 \times x_5 \end{bmatrix}$$

We have so:

$$\mathrm{u}(v) = \begin{pmatrix} 2 \times x_1 - x_2 - x_3, \\ -x_1 + 2 \times x_2 - x_3, \\ -x_1 - x_2 + 3 \times x_3 - x_4, \\ -x_3 + 3 \times x_4 - 2 \times x_5, \\ -2 \times x_4 + 2 \times x_5 \end{pmatrix}$$

Given the expression of $\mathrm{u}(v)$ *and given that v is an eigenvector of* M_l *if and only if v is a non-zero vector*[2] *and that there exists a value $\lambda \in \mathbb{R}$ such that:* $\mathrm{u}(v) = \lambda \cdot v$*, we have thus to solve the following system of*

[2]That is, whether $x_1 \neq 0$, $x_2 \neq 0$, $x_3 \neq 0$, $x_4 \neq 0$ or $x_5 \neq 0$

equations:

$$\begin{cases} 2\times x_1 - x_2 - x_3 & = & \lambda \times x_1 \\ -x_1 + 2\times x_2 - x_3 & = & \lambda \times x_2 \\ -x_1 - x_2 + 3\times x_3 - x_4 & = & \lambda \times x_3 \\ -x_3 + 3\times x_4 - 2\times x_5 & = & \lambda \times x_4 \\ -2\times x_4 + 2\times x_5 & = & \lambda \times x_5 \\ |x_1| + |x_2| + |x_3| + |x_4| + |x_5| & \neq & 0 \end{cases}$$

There are five solutions (five eigenvectors v and their corresponding eigenvalues λ):

- *The eigenvectors of* M_l *are:*

$$\begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \\ \left[\begin{array}{c} -0.4472136 \\ -0.4472136 \\ -0.4472136 \\ -0.4472136 \\ -0.4472136 \end{array}\right. & \begin{array}{c} 0.4397326 \\ 0.4397326 \\ 0.1821432 \\ -0.4397326 \\ -0.6218758 \end{array} & \begin{array}{c} 7.071068\times 1\times 10^{-1} \\ -7.071068\times 1\times 10^{-1} \\ -5.551115\times 1\times 10^{-17} \\ -2.775558\times 1\times 10^{-16} \\ -1.665335\times 1\times 10^{-16} \end{array} & \begin{array}{c} 0.3038906 \\ 0.3038906 \\ -0.7336569 \\ -0.3038906 \\ 0.4297663 \end{array} & \left.\begin{array}{c} 0.1195229 \\ 0.1195229 \\ -0.4780914 \\ 0.7171372 \\ -0.4780914 \end{array}\right] \end{array}$$

- *And their associated eigenvalues are:*

$$\begin{array}{ccccc} \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 \\ \left[\, 2.476651\times 1\times 10^{-16} \right. & 5.857864\times 1\times 10^{-1} & 3.000000 & 3.414214 & \left. 5.000000 \,\right] \end{array}$$

16.4 Computational complexity theory

In this section is given a succinct overview of computational complexity theory. For a more complete view on computational complexity theory see [28].

16.4.1 Principles

Computational complexity theory is a field of computer science whose purpose is to cluster computational problems into "complexity classes". Problems are gathered according to some criterion on the resources required to solve them. Generally the measure used to differentiate them is the time (*i.e.* the number of steps taken by an algorithm) needed or the space (*i.e.* the amount of memory) needed to solve them, but a clustering could be based on any other resource criterion. In this report, we will consider only time complexity classes.

We say that a problem $\mathscr{P}$ belongs to the complexity class $\mathscr{C}$ (or $\mathscr{P}$ has complexity $\mathscr{C}$) if there exists an algorithm that solves $\mathscr{P}$ satisfying the resource requirements of $\mathscr{C}$. Basically, the more a problem requires resources the more it will be considered has difficult.

In order to rank them in a fair way and so, form coherent complexity classes, problems are considered in their generic form. This means that the comparison is not made on specific "problem instances" (*i.e.* the problem applied to concrete data inputs). The resource requirements are expressed according to the problem "input size".

Nevertheless, considering problem's generic form is not sufficient for a proper comparison. Indeed, one can say that solving a given problem on such or such machines (that differ for example on their software or hardware architecture) would induce different resource requirements. In order to fix this issue, in computational complexity theory, we consider that algorithms are executed on some standard "model of computation", such as the so-called *"(Deterministic) Turing Machine"* introduced by Alan Turing in [65].

For the sake of brevity, we will not explain in details how it works but simply give the intuition of it. The *Turing Machine* is an abstract model of computational machine. It is composed of a *tape* on which *symbols* (0 or 1) can be read and written by an *head* that can move the tape left and right one *cell* at a time. An algorithm written for a *Turing Machine* is simply a set of *transitions* going through some so-called "states". A state indicates what to do given the symbol read on the current tape cell. What is initially written on the tape corresponds to the input (*i.e.* an encoding of it using the *Turing Machine* symbols). Given an input, the number of steps made by a *Turing Machine* to execute an algorithm is used as a time measurement and the number of cells used on the tape is used as a space measurement.

For a particular problem the input size could be expressed with a more understandable measure than the encoding size. For example for *Argumentation Framework* problems, we can use the number of arguments of a given an *Argumentation Framework* or the number of attacks.

Now, saying that "a problem $\mathscr{P}_1$ has a lower time complexity than another problem $\mathscr{P}_2$" means "for inputs of size n there exists an algorithm that solves $\mathscr{P}_1$ with fewer steps than any other algorithm that solves $\mathscr{P}_2$". Notice that, is taken into account the number of steps for the worst possible case of input of size n (*i.e.* the input of size n that induces the most transitions to solve the problem). Finally, it is the asymptotic behaviour of $\mathscr{P}_1$ and $\mathscr{P}_2$ as n grows that is considered.

The comparison method being fair, problems can now be grouped in suitable complexity classes. Those correspond basically to different orders of magnitude of steps.

Now let consider the complexity classes of so called "decision problems".

16.4.2 Decision problem theory

A decision problem is a type of computational problem that has for output a boolean. That is, given an input the solution of the problem is whether "yes" (equivalently true or 1) or "no" (equivalently false or 0). We say that the problem "accepts" or "rejects" the input.

One of the most famous decision problem, for its importance in computational complexity theory is the satisfaction problem, so-called SAT problem. It is defined as follows:

"Given a propositional formula ϕ, is ϕ satisfiable?[3]*"*

Decision problems are probably the most studied type of computational problems. Over the decades a lot of complexity classes and hierarchies between them have been established. Let consider some interesting complexity classes for our work.

[3]A propositional formula is said to be satisfiable if there exists a model of it, that is a value (true or false) assignation of its propositional variables for which ϕ is true.

16.4.3 Decision time complexity classes

16.4.3.1 Polynomial time: `P` and `L`

The polynomial time class `P` regroups computational problems for which there exists an algorithm that solves them in a number of steps that is polynomially related to the size of the input. Problems belonging to this class are considered as "easy" or "tractable".

`P` has a subclass called "logarithmic space" denoted by `L` that regroups the problems of `P` that require an amount of space (excluded the input and the output) that is logarithmically related to the size of the input.

16.4.3.2 Non-deterministic polynomial time: `NP`

The non-deterministic polynomial time class `NP` relies on the notion of *witness*.

Given an input x, a witness of x can be seen as a potential *proof by example* that the answer of the decision problem is positive for x. Let illustrate this considering the SAT problem. Given a propositional formula ϕ, a witness of SAT for ϕ is an interpretation of ϕ, *i.e.* a value assignation of the propositional variables of ϕ. Given an input x, a *valid witness* is a valid *proof by example* that the decision problem accepts x.

A problem $\mathscr{P}$ is in `NP` if and only if:

1. for any instance input x, all potential witnesses of x are of polynomial size *w.r.t.* $|x|$ (the size of x),
2. any witness of a given input can be verified in a polynomial number of steps *w.r.t.* $|x|$,
3. given an input x, $\mathscr{P}$ accepts x if and only if x has a valid witness.

As an example, the SAT problem is in `NP`.

`NP` can also be defined as the set of problems which can be solved in polynomial time on a *non-deterministic Turing Machine*.

The difference between a *Deterministic Turing Machine* and a *non-deterministic* one is that for each step several transitions are possible simultaneously. To illustrate this, one can imagine that at each step a new *Deterministic Turing Machine* could be added (a copy of the machine in its current state) for the problem solving. While a *Deterministic Turing Machine* follows a single computation path, a *non-deterministic Turing Machine* follows a computation tree. A given decision problem accepts x if there exists a non deterministic algorithm such that at least one computation branch followed by the *non-deterministic Turing Machine* accepts the input.

`NP` problems are thus computational problems for which there exists a non-deterministic algorithm that can solve them on *non-deterministic Turing Machine* and doing so, following a computation tree having a polynomial depth and a number of leaves relative to the input size.

`NP` is thus the class of computational problems for which a solution (a proposed *proof by example*) can be verified easily. Although there is no formal proof (at the time of writing) that $\texttt{P} \neq \texttt{NP}$, we will consider that this inequation holds in the following as it is the standard assumption.

16.4.3.3 The `coNP` class

The `coNP` class is the class regrouping the complement problems of those of `NP`. As for `NP`, `coNP` relies on the same notion of witness and the same witness properties, *i.e.* for any input x, all potential witnesses of x are of polynomial size *w.r.t.* $|x|$, and any witness of x is verifiable in an amount of steps polynomial *w.r.t.* $|x|$.

The difference between NP and coNP is that coNP regroups the decision problems for which we want all the witnesses for a certain property to be invalid.

As illustration, the coNP problem relative to the SAT problem is the following one:

"Given a propositional formula ϕ, is ϕ unsatisfiable?[4]"

Here the property of interest is the satifiability of ϕ. UNSAT will accept ϕ if and only if no witness of ϕ (*i.e.* no value assignation of its propositional variables) makes ϕ satisfied.

16.4.3.4 The polynomial-time hierarchy

The notion of "*oracle*" is very important to understand what is the polynomial-time hierarchy. An *oracle* is a black-box abstract machine that can solve a problem of a certain complexity class in one single step. Complexity classes can be expressed via this notion.

Given a problem $\mathscr{P}$, we say that $\mathscr{P}$ is in the complexity class $\mathscr{C}^{\mathscr{D}}$, if there exists an algorithm solving $\mathscr{P}$ with a complexity $\mathscr{C}$ and calling an oracle that solves in one operation a sub-problem of complexity class $\mathscr{D}$.

As an example, let consider the $\exists_2$QBF problem. Let ϕ be a propositional formula over the set of propositional variables Ω. Let $v_1 \subset \Omega$ and $v_2 \subset \Omega$ be two subsets of propositional formula such that $\{v_1, v_2\}$ is a partition[5] of Ω. The $\exists_2$QBF is the following decision problem:

"$\exists v_1$ such that $\forall v_2$, ϕ is true?"

Which means:

"Does there exist a valuation of the variables of v_1 such that for all valuations of the variables of v_2, ϕ is true?"

Let propose a non-deterministic algorithm to solve the $\exists_2$QBF problem. Let $\mathscr{O}$ be an oracle witnessing that a given propositional formula is valid.[6] $\mathscr{O}$ is in coNP. Indeed, to decide if ϕ is valid is equivalent to decide if $\neg\phi$ is unsatisfiable. The algorithm $\mathscr{A}$ that non-deterministically guesses a valuation of v_1 and then verifies if for all valuations of v_2 the combined valuations (of v_1 and v_2) are models of ϕ, can be viewed an NP algorithm using $\mathscr{O}$ as oracle. As $\mathscr{A}$ solves $\exists_2$QBF, we have $\exists_2$QBF belonging to the class $\mathtt{NP}^{\mathtt{coNP}}$.

The polynomial hierarchy, denoted by PH, is the complexity class hierarchy defined as follows:

- $\Sigma_0^{\mathtt{P}} = \Pi_0^{\mathtt{P}} = \Theta_0^{\mathtt{P}} = \mathtt{P}$
- $\Sigma_{k+1}^{\mathtt{P}} = \mathtt{NP}^{\Sigma_k^{\mathtt{P}}}$
- $\Theta_{k+1}^{\mathtt{P}} = \mathtt{P}^{\Sigma_k^{\mathtt{P}}}$
- $\Pi_{k+1}^{\mathtt{P}} = \mathtt{coNP}^{\Sigma_k^{\mathtt{P}}}$

The polynomial hierarchy is the union of all these complexity classes:

$$\mathrm{PH} = \bigcup_{k=0}^{\infty} \Sigma_k^{\mathtt{P}} = \bigcup_{k=0}^{\infty} \Pi_k^{\mathtt{P}}$$

Figure 16.4 on the next page illustrates this hierarchy.

[4]A propositional formula is said to be unsatisfiable *iff* there exists no model of it.

[5]$\Omega = v_1 \cup v_2$ and $v_1 \cap v_2 = \varnothing$.

[6]A formula ϕ is said to be valid if all valuations of its propositional variables are models of ϕ, *i.e.* ϕ is always true.

$$\begin{tikzcd}
 & \mathtt{NP} = \Sigma_1^{\mathtt{P}} \arrow[dr] & & \Sigma_2^{\mathtt{P}} \arrow[dr] & & \Sigma_3^{\mathtt{P}} \arrow[dr] & \\
\mathtt{P} \arrow[ur] \arrow[dr] & & \Theta_2^{\mathtt{P}} \arrow[ur] \arrow[dr] & & \Theta_3^{\mathtt{P}} \arrow[ur] \arrow[dr] & & \cdots \\
 & \mathtt{coNP} = \Pi_1^{\mathtt{P}} \arrow[ur] & & \Pi_2^{\mathtt{P}} \arrow[ur] & & \Pi_3^{\mathtt{P}} \arrow[ur] &
\end{tikzcd}$$

Figure 16.4: Polynomial hierarchy

Notice that calling a polynomial oracle from a non-deterministic algorithm doesn't add any complexity. As a consequence, we especially have $\mathtt{NP}^{\mathtt{P}} = \mathtt{NP}$.

Notice also that using as oracle, in an algorithm, a NP based oracle or coNP based oracle of same class *level* (*i.e.* $\Sigma_k^{\mathtt{P}}$ or $\Pi_k^{\mathtt{P}}$ for a given level k) doesn't matter. Indeed the answer of one of these oracles can be switched to correspond to the one solving the complementary problem.

As a consequence, we especially have $\mathtt{NP}^{\mathtt{coNP}} = \mathtt{NP}^{\mathtt{NP}} = \Sigma_2^{\mathtt{P}}$. And so, we have $\exists_2$QBF belonging to $\Sigma_2^{\mathtt{P}}$.

16.4.3.5 The difference class: DP

The so-called "difference class" denoted by DP is a kind of conjunction of the NP and coNP classes. A problem $\mathscr{P}$ belongs to DP if and only if it is composed of two sub-problems, $\mathscr{P}_1$ belonging to NP and $\mathscr{P}_2$ belonging to coNP, and for all input x, x is accepted by $\mathscr{P}$ if and only if x is accepted by $\mathscr{P}_1$ and $\mathscr{P}_2$.

As an illustration, the SAT-UNSAT problem belongs to DP. It is defined as follows:

"Given a couple of propositional formulas $\langle \phi, \Psi \rangle$, is ϕ satisfiable and Ψ unsatisfiable?"

Following the polynomial hierarchy introduced in the previous section, the DP-hierarchy is defined as follows:

$$\mathtt{DP}_k = \Sigma_k^{\mathtt{P}} \wedge \Pi_k^{\mathtt{P}}, \quad \text{with } k \in [\![1, +\infty[\![$$

Notice that "$\wedge$" means "conjunction of problems" as explained above. It is not the intersection of sets of problems.

16.4.4 Problem reduction, completeness and hardness

16.4.4.1 Problem reduction

Let $\mathscr{P}_1$ and $\mathscr{P}_2$ be two decision problems. We denote by $I_{\mathscr{P}_1}$ and $I_{\mathscr{P}_2}$ the sets of all the instances of $\mathscr{P}_1$ and $\mathscr{P}_2$. Let $f : I_{\mathscr{P}_1} \to I_{\mathscr{P}_2}$ be an *efficient*[7] procedure that transforms any instance of $\mathscr{P}_1$ into one instance of $\mathscr{P}_2$ such that for all $x \in I_{\mathscr{P}_1}$, $\mathscr{P}_1$ accepts x *iff* $\mathscr{P}_2$ accepts $f(x)$.

If such a procedure exists, it means that any algorithm solving $\mathscr{P}_2$ could be used to solve $\mathscr{P}_1$ by firstly converting $\mathscr{P}_1$ instances into $\mathscr{P}_2$ ones.

[7]The complexity of f should be "*easy*" compared to the complexity of solving $\mathscr{P}_1$ or $\mathscr{P}_2$.

Now if it holds that $\mathscr{P}_2$ is in some complexity class $\mathscr{C}$, it means that $\mathscr{P}_1$ is also in $\mathscr{C}$ considering that f is an efficient problem transformer. Likewise if $\mathscr{P}_2$ is not in $\mathscr{C}$, then $\mathscr{P}_1$ is not in $\mathscr{C}$.

In computational complexity theory, polynomial reduction are considered as efficient. Polynomial reductions are thus applicable to problems in $\mathtt{P}$ or complexity classes above. We denote by $\mathscr{P}_1 \leq_{\mathtt{P}} \mathscr{P}_2$ the relation expressing that $\mathscr{P}_1$ is polynomially reducible to $\mathscr{P}_2$, and by $\mathscr{P}_1 \leq^f_{\mathtt{P}} \mathscr{P}_2$ that the relation holds by using f. Usually, we use polynomial reductions in $\mathtt{P}$ while studying problems in $\mathtt{NP}$ and harder complexity classes, and *log-space reductions*, that is procedures belonging to $\mathtt{L}$ (denoted by $\leq_{\mathtt{L}}$), while studying complexity classes within $\mathtt{P}$.

16.4.4.2 Completeness and hardness

We consider that a problem is hard for a certain class $\mathscr{C}$ if an efficient algorithm solving it could be used to efficiently solve, by mean of reductions, all the problems in $\mathscr{C}$. It is formally defined as follows: let $\mathscr{P}_1$ be a problem of complexity class $\mathscr{C}$. $\mathscr{P}_1$ is said to be hard *w.r.t.* $\mathscr{C}$, denoted by $\mathscr{C}$*-hard*, if:

$$\forall \mathscr{P}_2 \in \mathscr{C},\ \mathscr{P}_2 \leq_{\mathtt{P}} \mathscr{P}_1$$

A problem $\mathscr{P}$ is said to be complete for $\mathscr{C}$, denoted by $\mathscr{C}$*-c*, if $\mathscr{P} \in \mathscr{C}$ and $\mathscr{P}$ is $\mathscr{C}$*-hard*.

16.4.5 Function problems

As such type of problems has not been studied in this thesis, we do not detail the *function problem Theory* as it has been done for *Decision Problem Theory*. We rather give a definition of what a *function problem* is and give one example.

Formally:

Definition 137 (Function problem)**.** *A function problem $\mathscr{P}$ is defined as a relation $\mathcal{R}$ over strings of an arbitrary alphabet Σ:*

$$\mathcal{R} \subseteq \Sigma^* \times \Sigma^*, \textit{with:}$$
$$\Sigma^* \textit{being the set of all strings over symbols in } \Sigma\textit{, including the empty string}$$

An algorithm solves $\mathscr{P}$ if for every input x such that there exists a value y satisfying $(x,y) \in \mathcal{R}$, the algorithm produces one such y.

In simple words, a function problem is a computational problem that produces an output more complex than a boolean, as it is the case for decision problems. Here is a concrete example of function problem:

Example 87. *The functional version of the SAT problem presented in Section 16.4.2 on page 179, so-called FSAT, is the following:*

Given a boolean formula ϕ with variables $x_1, \ldots, x_n$,
find an assignment $x_i \rightarrow \{\mathtt{true}, \mathtt{false}\}$ such that ϕ evaluates to $\mathtt{true}$
or decide that no such assignment exists.

16.5 Analysis of Algorithms

16.5.1 Presentation

As seen in the previous section the *computational complexity theory* classifies computational problems following their hardness into complexity classes. In this theory, the subjects of interest are thus theoretical problems. It is not about concrete implementations of theoretical problems. Instead, the study of the complexity of explicitly given algorithms is called analysis of algorithms.

Both are highly related. Indeed, given a problem $\mathscr{P}$, if a concrete algorithm $\mathcal{A}$ solves $\mathscr{P}$ then the complexity class of $\mathscr{P}$ is a lower bound for the complexity of $\mathcal{A}$ and the complexity of $\mathcal{A}$ is an upper bound for the complexity of $\mathscr{P}$.

It is however important to note that, as far as we know about the current state of computational technologies, there are no such things as oracles in the real world. Even quantum machines are not *non-deterministic turing machines*. As a consequence, on a deterministic real world computer, algorithms solving theoretical problems whose complexity class $\mathscr{C}$ belongs to the polynomial hierarchy such that $\mathscr{C} \notin \mathtt{P}$, are thus algorithms of complexity exponential.

This being said, in this section we present basic notions to analyse the complexity of algorithms.

16.5.2 Asymptotic analysis

Given an algorithm $\mathcal{A}$, we are interested in estimating its running time as a function of a given machine-independent parameter $n \in \mathbb{N}$. Moreover, to have a machine-independent measurement, we identify the calculation time with the number of executed instructions. The parameter n could be for example the length of an array or the number of arguments/attacks in an AF.

To compare algorithms, we consider only their behavior for a large n (that is what we call the *"asymptotic complexity"*) and consider their "*order of magnitude*" rather than the precise number of instructions executed, ensuring that the complexity measure is independent of the programming language and the machine on which the algorithm runs.

This *Order of magnitude* can be declined into several relation types. Here are the three most common ones:

Definition 138 (Order of magnitude relations)**.** *Let T and f be positive non-zero functions. O, Ω and Θ are order of magnitude relations defined as follows:*

- *"Big O": $T(n) \in O(f)$ if $\exists c \in \mathbb{R}^*$ and $n_0 \in \mathbb{N}$ s.t. $\forall n \geq n_0$, $T(n) \leq c \times f(n)$. T is said to be* asymptotically *dominated by f. f is an* asymptotic *upper bound for T.*
- *"Big Ω": $T(n) \in \Omega(f)$ if $\exists c \in \mathbb{R}^*$ and $n_0 \in \mathbb{N}$ s.t. $\forall n \geq n_0$, $T(n) \geq c \times f(n)$. f is an* asymptotic *lower bound for T.*
- *"Big Θ": $T(n) \in \Theta(f)$ if $T(n) \in \Omega(f)$ and $T(n) \in O(f)$.*

The *"asymptotic complexity"* of an algorithm is formally defined, as follows:

Definition 139 (Asymptotic complexity)**.** *Let $\mathcal{A}$ be an algorithm and n be its measurement parameter. The* asymptotic complexity *of $\mathcal{A}$, denoted by $T(n)$, is the "*order of magnitude*" of its execution time (in terms of number of executed instructions) when $n \to \infty$.*

`Note:` An asymptotic complexity can be studied for the worst (denoted T_{max}), the best (denoted T_{min}) or the average (denoted T_{ave}) cases for an input of size n.[8] *Generally, we are interested in the complexities for the worst and average cases.*

Example 88. *Let $\mathcal{A}$ be an algorithm computing the product of two square matrices of size n, using the basic method non optimized. We have:*

- $T_{min}(n) = T_{max}(n) = T_{ave}(n) \in \Theta(n^2)$
- $T_{min}(n) = T_{max}(n) = T_{ave}(n) \in O(e^n)$
- $T_{min}(n) = T_{max}(n) = T_{ave}(n) \in \Omega(n)$

Notice that $O(e^n)$ is far to be the best upper bound for $T_{min}(n)$ (resp. $T_{max}(n)$, $T_{ave}(n)$), as $T_{min}(n) \in O(n^2)$ (resp. $T_{max}(n) \in O(n^2)$, $T_{ave}(n) \in O(n^2)$). Likewise, $\Omega(n)$ is not the best lower bound for $T_{min}(n)$ (resp. $T_{max}(n)$, $T_{ave}(n)$) as $T_{min}(n) \in \Omega(n^2)$ (resp. $T_{max}(n) \in \Omega(n^2)$, $T_{ave}(n) \in \Omega(n^2)$)

[8] For *n* fixed, there are several possible inputs. Among them are worst and best cases.

Chapter 17

Mathematical Problems

This chapter presents two mathematical problems used in this thesis. Section 17.1 presents the *Constraint Satisfaction Problem* and Section 17.2 on page 188 a graph/data clustering method called *Spectral Clustering*. Notice that for the sake of space we will not go into details. The scope of this background is limited to the necessary.

17.1 Constraint Satisfaction problem

In this section is presented the formal definition of a constraint satisfaction problem (CSP).[1]

Given a set of changeable state objects, a Constraint Satisfaction Problem (CSP) is a mathematical problem in which we look for a configuration of object states (*i.e.* a mapping where each object has a particular state) that satisfies a certain number of constraints.

Definition 140 (Constraint Satisfaction Problem)**.** *A CSP is defined by a triplet $\Psi = \langle X, D, C \rangle$ where:*

- $X = \{X_1, ..., X_n\}$ *is a set of variables.*
- $D = \{D(X_1), ..., D(X_n)\}$ *is a set of domains, where $D(X_i) \subset \mathbb{Z}$ is the finite set of values that variable X_i can take (i.e. $D(X_i)$ is the domain of X_i).*
- $C = \{c_1, ..., c_e\}$ *is a set of constraints.*

`Note:` *In real problems, the value of the variables are not always integers but they can be converted to integers. Any value of a given variable domain can be associated with an integer. Therefore, this definition of CSP is sufficient to capture all real constraint type problems.*

Definition 141 (Constraint)**.** *A constraint c_i is a boolean function involving a sequence of variables $X(c_i) = (X_{i_1}, ..., X_{i_q})$ called its scheme. The function is defined on $\mathbb{Z}^q$. A combination of values (or tuple) $\tau \in \mathbb{Z}^q$ satisfies c_i if $c_i(\tau) = 1$ (also noted $\tau \in c_i$). If $c_i(\tau) = 0$ (or $\tau \notin c_i$), τ violates c_i.*

[1] A CSP modelling is used for in *AFDivider* algorithm. See Section 4.2.4 on page 42.

Definition 142 (Instantiation)**.** *An instantiation of the X variables is a mapping where each X_i takes a value in its domain $D(X_i)$.*

Definition 143 (CSP Solution)**.** *A solution of a CSP is an instantiation of the X variables that violates no constraint.*

`Note:` *For more details on CSP see [64].*

Example 89. *Let consider the CSP $\Psi = \langle X, D, C \rangle$ illustrated in Figure 17.1 where:*

- $X = \{X_1, X_2, X_3, X_4\}$ *is the set of variables of* Ψ
- $D = \left\{ \begin{array}{l} D(X_1) = \{1,4\}, \\ D(X_2) = \{1,2,3,4\}, \\ D(X_3) = \{1,3\}, \\ D(X_4) = \{1,3\} \end{array} \right\}$ *is a set of their domains*
- $C = \{c_1, c_2, c_3, c_4\}$ *is a set of constraints, with c_i for $i \in \{1,...,4\}$, being an inequality constraint. Their schemes, that is the set of variables on which they are applied, are:*
  - $X(c_1) = (X_1, X_2)$
  - $X(c_2) = (X_1, X_3)$
  - $X(c_3) = (X_3, X_4)$
  - $X(c_4) = (X_1, X_4)$

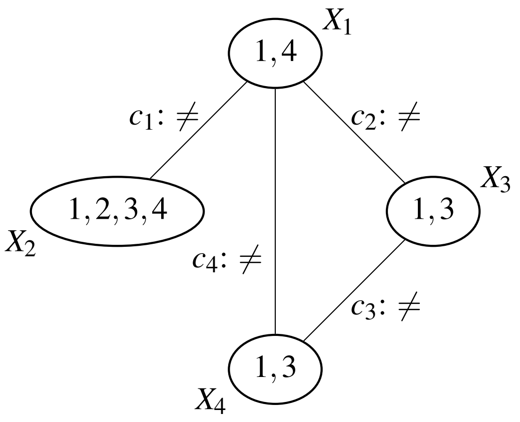


Figure 17.1: CSP illustration

Now that the CSP problem has been properly defined, let consider some facts. We have:

- *If* $\tau = (1,1)$ *then* $c_1(\tau) =$ `false`

- *If* $\tau = (1,2)$ *then* $c_1(\tau) = \texttt{true}$
- *If* $\tau = (3,2)$ *then* $c_1(\tau) = \texttt{false}$ *as* $3 \notin D(X_1)$
- $\{X_1 = 4,\ X_2 = 1,\ X_3 = 3,\ X_4 = 3\}$ *is an instanciation that does not solve* Ψ *as it violates* c_3
- $\{X_1 = 4,\ X_2 = 2,\ X_3 = 3,\ X_4 = 1\}$ *is a solution of* Ψ

17.2 Spectral Clustering

Finding clusters in graph is a subject that has been widely studied. In this section we will present an approach that we used for argumentation framework clustering: the *spectral clustering*. For an overview of non-directed graph clustering algorithms see [63] and for directed ones see [54].

The spectral clustering is a clustering method which is based on the spectral analysis of a similarity laplacian matrix.

A similarity matrix is a square matrix in which the lines and the columns describe the same set of elements. The matrix coefficients (*i.e.* the cell values) represent how much a element is similar to another, according to given similarity measure.

In short, here is how the spectral clustering works:

- Given a similarity matrix, the laplacian of this matrix is computed.
  - The lines of the laplacian matrix correspond to the coordinates of the elements in a certain similarity space.
- The eigenvectors of the laplacian matrix with their associated eigenvalues are computed.
- The eigenvalues computed are sorted increasing order. A number n of them is kept with their associated eigenvectors.
  - This solving and sorting process is done in order to project the datapoints in a new space which maximizes the closeness of similar elements. This space basis is formed by the computed eigenvectors. The eigenvalue of an eigenvector represents how much the datapoints are scattered on the eigenvector corresponding axis. Given that we are interested in the dimensions that maximize the best similarity (axes on which the datapoints are closed to each other), we keep the n smallest eigenvalues and their eigenvectors.
  - If there are clusters in a data set, it is reasonable to think that the number of small eigenvalues are the number of groups identified (the datapoints being rather homogeneous following that axis). An heuristic to find the appropriate number of dimensions to keep is to detect the jump in the eigenvalues sequence (sorted in increasing order).
- A matrix whose columns are the remaining eigenvectors is constructed. The lines of it represent the new elements coordinates.

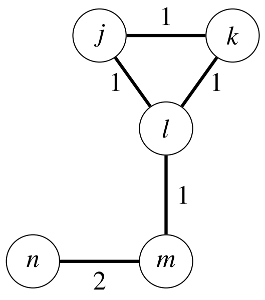


Figure 17.2: The weighted non-directed graph of Example 86

- Once this data treatment is done, a simple algorithm of clustering such as *KMeans* is applied to that new data set, seeking for a partition into *n* parts, based on the coordinates of the elements (see [53] for more information about *KMeans* algorithm).

Let illustrate this algorithm on a concrete example.

Example 90. *Let consider the graph $G = (V,E,W)$ shown in Figure 17.2 and let consider the weight over the edges as the similarity between two nodes. Following Example 86 on page 178, the similarity matrix corresponding to G is the following adjacency matrix* M_a*:*

$$\mathrm{M}_a = \begin{array}{c} \\ j \\ k \\ l \\ m \\ n \end{array} \begin{array}{c} \begin{array}{ccccc} j & k & l & m & n \end{array} \\ \left[\begin{array}{ccccc} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 2 & 0 \end{array}\right] \end{array}$$

The degree matrix M_d *of G is:*

$$\mathrm{M}_d = \begin{array}{c} \\ j \\ k \\ l \\ m \\ n \end{array} \begin{array}{c} \begin{array}{ccccc} j & k & l & m & n \end{array} \\ \left[\begin{array}{ccccc} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{array}\right] \end{array}$$

And then, the laplacian matrix M_l *of G is:*

$$\mathrm{M}_d - \mathrm{M}_a = \mathrm{M}_l = \begin{array}{c} \\ j \\ k \\ l \\ m \\ n \end{array} \begin{array}{c} \begin{array}{ccccc} j & k & l & m & n \end{array} \\ \left[\begin{array}{ccccc} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 \\ 0 & 0 & -1 & 3 & -2 \\ 0 & 0 & 0 & -2 & 2 \end{array}\right] \end{array}$$

From the graph similarity matrix and by means of the laplacian matrix, the datapoints are projected in a new space in which similarity is maximised. The idea is that, if a certain structure exists in the data set, we will see in that space appear some agglomerates corresponding to the node clusters.

To do that, we compute the n smallest eigenvalues[2] *of the laplacian matrix obtained from the similarity matrix and the vectors associated with them (this n is an arbitrary parameter).*

Indeed, the eigenvectors found will correspond to the basis of that similarity space and the eigenvalues to the variance on the corresponding axes. Given that we are looking for homogeneous groups, we will consider only the axis on which the variance is low, and so the eigenvectors that have small eigenvalues. The space whose basis is the n selected eigenvectors (corresponding to the n smallest eigenvalues) is then a compression of similarity space (i.e. we keep only the dimension useful for a clustering).

For the sake of the illustration let fix the parameter n to its maximal value: $n = 5$. *Following Example 86 on page 178, we have:*

- *The eigenvectors of* M_l *are:*

$$\begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \left[\begin{array}{ccccc} -0.4472136 & 0.4397326 & 7.071068 \times 1 \times 10^{-1} & 0.3038906 & 0.1195229 \\ -0.4472136 & 0.4397326 & -7.071068 \times 1 \times 10^{-1} & 0.3038906 & 0.1195229 \\ -0.4472136 & 0.1821432 & -5.551115 \times 1 \times 10^{-17} & -0.7336569 & -0.4780914 \\ -0.4472136 & -0.4397326 & -2.775558 \times 1 \times 10^{-16} & -0.3038906 & 0.7171372 \\ -0.4472136 & -0.6218758 & -1.665335 \times 1 \times 10^{-16} & 0.4297663 & -0.4780914 \end{array}\right] \end{array}$$

- *And their associated eigenvalues are:*

$$\begin{array}{c} \begin{array}{ccccc} \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 \end{array} \\ \left[\begin{array}{ccccc} 2.476651 \times 1 \times 10^{-16} & 5.857864 \times 1 \times 10^{-1} & 3.000000 & 3.414214 & 5.000000 \end{array}\right] \end{array}$$

Now that the similarity space is found, another important step is to find how many groups we have in that space. Intuitively, the number of eigenvectors with small eigenvalues, and so, the number of axes with small

[2]There exist algorithms, such as *Krylov-Schur* method, able to compute eigenvectors from smallest to greatest eigenvalue and stop at any wanted step (*e.g.* number of vectors found). With such an algorithm it is not necessary to find all the solutions as we are interested only in the small eigenvalues.

variance is the number of clusters. However, within the n smallest eigenvalues determined, it is difficult to formally say what is a small eigenvalue, and so, what is the number of clusters to chose.

Sorted in ascending order, the eigenvalue sequence represents how the similarity within clusters increases as the number of clusters grows. Obviously, the more clusters, the more homogeneous they will get. The idea is to find a compromise between number of clusters and homogeneity.

As eigenvalues say, in the end, how much the corresponding clusters will be homogeneous, the heuristic we have chosen to consider is to look for the "best elbow" in that ascending order sequence. We look for the number of dimensions to keep just before the quick growth of the variance. If the topology of the graph let appear some clusters then we will indeed have elbows. We can see in Figure 17.3 that this "best elbow" in the eigenvalues sequence (blue line with squares) is in second position. In that case the number of clusters determined by that heuristic is so 2.

To compute that "best elbow" we consider the second derivative (green line with triangles) of the ascending order sequence. As the second derivative represents the concavity of the eigenvalue sequence, we can take the first value of the second derivative above a certain threshold (red line without symbol) determined experimentally (i.e. the first position where the eigenvalue sequence is enough convex).

As you can see the first point of the second derivative, corresponding to the concavity formed by the first three eigenvalues, is the first value above the threshold and then we determine that the "best elbow" is in position 2.

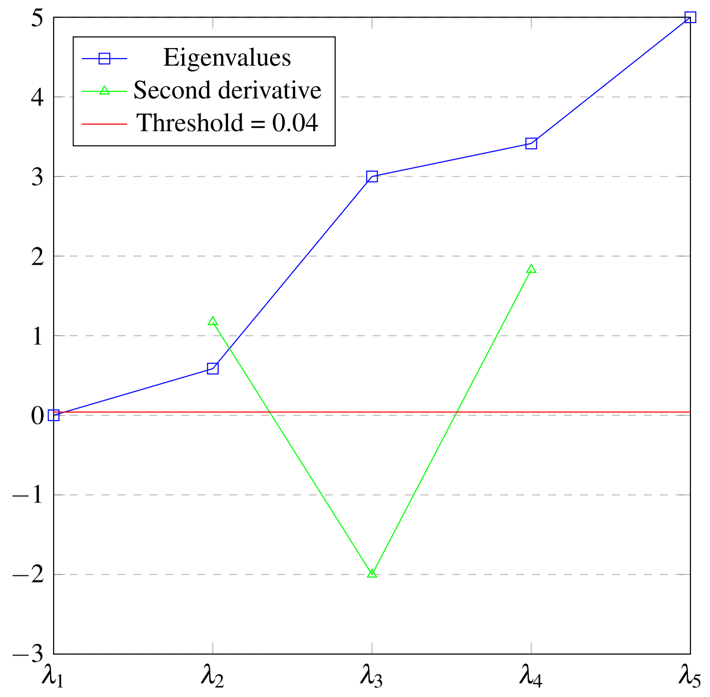


Figure 17.3: eigenvalues sorted by ascending order

Once the number of clusters is chosen, we remove from the similarity matrix the columns that are after this number (i.e. we remove the dimensions we are not interested in for the clustering). The lines of the *resulting matrix, which columns are the kept eigenvectors, correspond to the coordinates of the nodes in that new compressed similarity space.*

Finally, we just have to apply a KMeans *type algorithm [53] to find the groups of datapoint in that space and so have the partition of nodes we wanted.*

Given that the chosen number of clusters is 2*, we keep only the vectors* v_1 *and* v_2 *and when binded by column the lines they form correspond to the coordinates of the nodes in a new space that maximizes the similarity.*

$$\begin{array}{c} \\ j \\ k \\ l \\ m \\ n \end{array}
\begin{array}{c} \begin{array}{cc} \quad v_1 \quad & \quad v_2 \quad \end{array} \\
\begin{bmatrix} -0.4472136 & 0.4397326 \\ -0.4472136 & 0.4397326 \\ -0.4472136 & 0.1821432 \\ -0.4472136 & -0.4397326 \\ -0.4472136 & -0.6218758 \end{bmatrix} \end{array}$$

As you can see the v_1 *dimension is useless. In practice it is removed.*

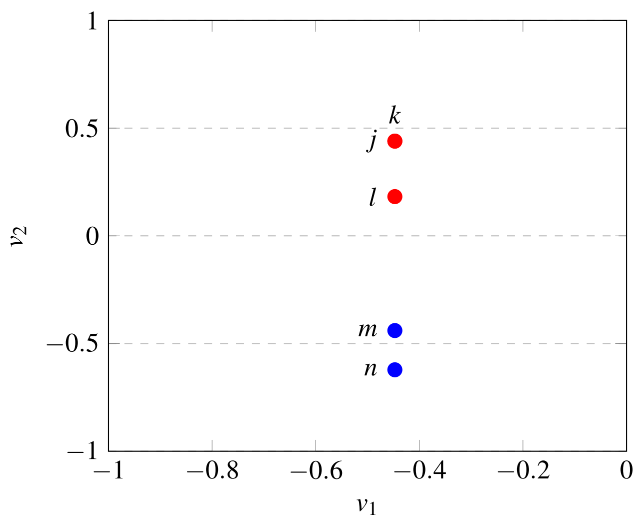


Figure 17.4: Node datapoints projected in similarity space

Figure 17.4 shows two clusters as expected: $G\downarrow_{\{m,n\}}$ *and* $G\downarrow_{\{j,k,l\}}$.

`Note:` *For more information on spectral clustering see [67].*

Appendix 2: Tables

Tables of symbols

Table 18.1: Shortcut symbols

Symbol	Meaning
i.e.	The abbreviation for the Latin phrase “**id est**”, meaning “**that is**”
e.g.	The abbreviation for the Latin phrase “**exempli gratia**”, meaning “**for example**”
w.r.t.	The abbreviation for “**with regard to**”
s.t.	The abbreviation for “**such that**”
iff	The abbreviation for “**if and only if**”
AF	The abbreviation for “**Argumentation Framework (Dung)**”
SCC_{af}	The abbreviation for “**AF SCC**”
RAF	The abbreviation for “**Recursive Argumentation Framework**”
SCC_{raf}	The abbreviation for “**RAF SCC**”
$USCC_{af}$	The abbreviation for “**AF USCC**”. See Definition 27 on page 21
$USCC_{raf}$	The abbreviation for “**RAF USCC**”. See Definition 109 on page 161
BA, ER, WS, TR, F2,BW	The abbreviations for respectively Barabási–Albert, Erdős–Rényi, Watts-Strogatz, Traffic, Ferry and Block world graph types.

Table 18.2: Graph and matrix symbols

Symbol	Meaning
$G = (V, E)$	A graph G with V being a set of nodes and E being a set of edges
$G = (V, E, W)$	A graph G with V being a set of nodes, E being a set of edges and W being a function that associates a weight to any $e \in E$
$deg(G, v)$	Let $G = (V, E)$ be a non-weighted graph and $v \in V$. $deg(G, v)$ is the number of incident edges to v in G. See Definitions 116 and 117 on page 172
$\deg_w(G, v)$	Let $G = (V, E, W)$ be a weighted graph and $v \in V$. $deg(G, v)$ is the weight sum of the incident edges to v in G. See Definition 118 on page 172
$\mathscr{R}_d(G)$	Let G be a graph. $\mathscr{R}_d(G)$ is the relation density in G with: $$\mathscr{R}_d(G) = \frac{\lvert E \rvert}{\lvert V \rvert}$$ See Definition 131 on page 174
M	A matrix. See Definition 132 on page 175
$(M)_{i,j}$	The cell of row i and column j of the matrix M
M_a	The adjacency matrix of a certain graph. See Definition 134 on page 175
M_d	The degree matrix of a certain graph. See Definition 135 on page 175
M_l	The laplacian matrix of a certain graph. See Definition 136 on page 175

Table 18.3: AF and RAF related symbols

Symbol	Meaning
σ	A semantics
S	A set of arguments/elements

Continued on next page ...

Table 18.3: AF and RAF related symbols (continued)

Symbol	Meaning
$\mathtt{in}$	A labelling acceptability value indicating that the argument/element is **accepted**
$\mathtt{out}$	A labelling acceptability value indicating that the argument/element is **rejected**
$\mathtt{und}$	A labelling acceptability value indicating that the argument/element is **undecided**
$\mathtt{iout}$	A labelling acceptability value indicating that the argument/element of a cluster structure/partial RAF is **illegally rejected**. See Definition 34 on page 36
$\mathtt{iund}$	A labelling acceptability value indicating that the argument/element of a cluster structure/partial RAF is **illegally undecided**. See Definition 34 on page 36
$\mathtt{in}(.)$, $\mathtt{out}(.)$, $\mathtt{und}(.)$	The sets of arguments/elements in "." that are labelled respectively $\mathtt{in}$, $\mathtt{out}$ and $\mathtt{und}$
$\Omega \quad \omega$	The symbols Ω and ω are used to represent respectively partitions and parts in a given partition. In the case of AFs, we have, given an AF $\mathcal{AF} = \langle A, K \rangle$: $\Omega = \{\omega_1, ..., \omega_n\}$ being a partition of A. In the case of RAFs, we have, given a RAF $\mathcal{RAF} = \langle A, K, s, t \rangle$: $\Omega = \{\omega_1, ..., \omega_n\}$ being a partition of $A \cup K$

Table 18.4: AF related Symbols

Symbol	Meaning
Φ_{af}	The set of all possible AF
$\mathcal{AF} = \langle A, K \rangle$	An AF where: • A is a set of arguments • K is a set of attacks (*i.e.* $K \subseteq A \times A$)
$af = \langle A, K \rangle$	Same definition as the previous one. This notation is used occasionally to represent a sub-AF and especially connected component sub-AFs of an AF named $\mathcal{AF}$. See for example Definition 40 on page 64.

Continued on next page ...

Table 18.4: AF related Symbols (continued)

Symbol	Meaning
$Def(S)$	The set of arguments defeated *w.r.t.* the set of arguments S
$Acc(S)$	The set of arguments acceptable *w.r.t.* the set of arguments S
$\mathscr{L}(\mathcal{AF})$	Let $\mathcal{AF}$ be an AF. $\mathscr{L}(\mathcal{AF})$ is the set of all possible labellings of $\mathcal{AF}$
$\mathscr{L}(S)$	Let S be a set of arguments. $\mathscr{L}(S)$ is the set of all possible labellings of S
$\mathscr{L}_\sigma(\mathcal{AF})$	Let $\mathcal{AF}$ be an AF. $\mathscr{L}_\sigma(\mathcal{AF})$ is the set of $\mathcal{AF}$ labellings under the semantics σ
ℓ	A labelling
S^{inp}	Let $\mathcal{AF} = \langle A, K\rangle$ be an AF and let $S \subseteq A$. S^{inp} is the set of input arguments of S, that is: $S^{inp} = \{b \in A \setminus S \mid \exists a \in S, (b,a) \in K\}$
S^K	Let $\mathcal{AF} = \langle A, K\rangle$ be an AF and let $S \subseteq A$. S^K is the conditioning relation of S, that is: $K \cap (S^{inp} \times S)$
$\left\langle \mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$	An AF with input, with: • $\mathcal{AF}$ being an AF • $\mathcal{I}$ being a set of input arguments • $\ell^{\mathcal{I}}$ being a labelling of $\mathcal{I}$ • $K_{\mathcal{I}}$ being a set of conditioning relations such that $K_{\mathcal{I}} \subseteq \mathcal{I} \times A$
$std\text{-}\mathcal{AF}$	Standard AF corresponding to an AF with input $\left\langle \mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$ is defined as $std\text{-}\mathcal{AF} = \langle A \cup \mathcal{I}', K \cup K'_{\mathcal{I}}\rangle$, where $\mathcal{I}' = \mathcal{I} \cup \left\{a' \mid a \in \mathcal{I} \cap \mathtt{out}(\ell^{\mathcal{I}})\right\}$ and $K'_{\mathcal{I}} = K_{\mathcal{I}} \cup \left\{(a',a) \mid a \in \mathcal{I} \cap \mathtt{out}(\ell^{\mathcal{I}})\right\} \cup \left\{(a,a) \mid a \in \mathcal{I} \cap \mathtt{und}(\ell^{\mathcal{I}})\right\}$. See Definition 17 on page 17
$\mathcal{I}'$	See $std\text{-}\mathcal{AF}$
$\mathscr{F}^{af}_{\sigma}(\mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}})$	The canonical local function computing the σ-labellings associated with the AF with input $\left\langle \mathcal{AF}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$ (see Definition 18 on page 18)
$Paths_{af}(\mathcal{AF})$	The set of paths of $\mathcal{AF}$
$Walks_{af}(\mathcal{AF})$	The set of walks of $\mathcal{AF}$

Continued on next page ...

Table 18.4: AF related Symbols (continued)

Symbol	Meaning
$Cycles_{af}(\mathcal{AF})$	The set of cycles of $\mathcal{AF}$
$PE_{af}(\mathcal{AF})$	The path-equivalence relation over $\mathcal{AF}$
$SCCS_{af}(\mathcal{AF})$	The set of SCC_{af} of $\mathcal{AF}$
$\mathscr{S}(\mathcal{AF})$	A partition selector $\mathscr{S}$ is a function receiving as input an AF $\mathcal{AF} = \langle A, K\rangle$ and returning a set of partitions of A
$\mathscr{S}_{D\text{-}af}$	The "*default AF partition selector*" is the partition selector that produces all possible partitions of an AF. See Definition 25 on page 20
$\mathscr{S}_{USCC}$	A partition selector that does not split SCC_{af}. See Definition 27 on page 21
$\mathcal{AF}_{hard}$	Let $\mathcal{AF} = \langle A, K\rangle$ be an AF. The "*hard part*" of $\mathcal{AF}$ is the sub AF defined as: $\mathcal{AF}_{hard} = \mathcal{AF}\downarrow_{\{a\mid a\in A, \ell_{gr}(a)=\mathtt{und}\}}$. See Section 4.2.1 on page 30
$\kappa = \left\langle af, I, O, B\right\rangle$	Let $\mathcal{AF} = \langle A, K\rangle$ be an AF κ is a cluster structure where: • af is a sub AF of $\mathcal{AF}$ such that for some $\omega \subseteq A$, $af = \mathcal{AF}\downarrow_\omega$ • $I = \{(a,b) \mid (a,b) \in K, b \in \omega$ and $a \notin \omega\}$ • $O = \{(a,b) \mid (a,b) \in K, b \notin \omega$ and $a \in \omega\}$ • $B = \{a \mid (a,b) \in O$ or $(b,a) \in I\}$ See Definition 30 on page 32
μ	Let $\kappa = \left\langle af, I, O, B\right\rangle$ be a cluster structure. A context μ of κ is a labelling of the inward attack sources of κ, *i.e.* $\{a \mid (a,b) \in I\}$. See Definition 31 on page 34
$\mathscr{L}_\sigma^{\mu(\kappa)}$	The set of induced labelling produced by the cluster structure κ under the context μ. See Definition 33 on page 35
ξ	Let $\kappa = \left\langle af, I, O, B\right\rangle$ be a cluster structure, $\mathscr{L}_\sigma^{\mu(\kappa)}$ a set of induced labellings and $\ell \in \mathscr{L}_\sigma^{\mu(\kappa)}$ be a labelling. The configuration ξ corresponding to ℓ is a five value-based labelling of the border arguments of κ. See Definition 34 on page 36
af'	Let $\kappa = \left\langle af, I, O, B\right\rangle$ be a cluster structure and μ be a context of κ. We denote by af' the induced AF from κ under μ. See Definition 32 on page 35
D	The set of deleted arguments from the an induced AF. See Definition 32 on page 35

Continued on next page ...

Table 18.4: AF related Symbols (continued)

Symbol	Meaning
$\mathscr{L}^{\kappa}_{\mathscr{D}}$	The set of distinct labellings of a cluster structure κ. See Definition 35 on page 39
$\xi_{\ell_i^{\kappa}}$	The merge configuration corresponding to the labelling ℓ_i of the cluster structure κ. See Definition 36 on page 41
$p = \{\xi_1, ..., \xi_n\}$	A reunified labelling profile. See Definition 37 on page 43
Ξ	Let $p = \{\xi_1, ..., \xi_n\}$ be a reunified labelling profile. Ξ is the union of those configurations: $\Xi = (\bigcup_{i=1}^{n} \xi_i)$
$\mathscr{P}^{\kappa}$	The set of configurations corresponding to the cluster structure κ. See Algorithm 2 on page 31
$\mathscr{P}$	A set of reunified labelling profiles
$\mathscr{U}(\mathcal{AF})$	The undirection transformation that transforms an AF into a weighted non-directed graph. See Definition 39 on page 52
$\texttt{Comp}_{\sigma}(\mathcal{AF})$	The compact enumeration representation of $\mathcal{AF}$ according to a semantics σ. See Definition 40 on page 64
$\mathcal{MAF}$	A "Meta-Argumentation Framework" (MAF). They are basically Argumentation Framework. See [46] for more information
$Cred_{\sigma}$, $Skep_{\sigma}$, Ver_{σ}, $Exists_{\sigma}$, $Exists_{\sigma}^{\neg\varnothing}$, $Unique_{\sigma}$	Respectively the *Credulous Acceptance*, *Skeptical Acceptance*, *Verification of a labelling*, *Existence of an extension/labelling/structure*, *Existence of a "non-empty" extension/labelling/structure* and the *Uniqueness of a solution* AF decision problems. See Definition 28 on page 23

Table 18.5: RAF related Symbols

Symbol	Meaning
Φ_{raf}	The set of all possible RAF

Continued on next page ...

Table 18.5: RAF related Symbols (continued)

Symbol	Meaning
$\mathcal{RAF} = \langle A, K, s, t \rangle$	A RAF where: • A is a set of arguments • K is a set of attacks • s is a mapping function that associates an attack with its source (*i.e.* $s : K \to A$) • t is a mapping function that associates an attack with its target (*i.e.* $t : K \to A \cup K$)
$\mathcal{U} = \langle S, Q \rangle$	A structure. S is a set of arguments and Q is a set of attacks
RAF-$Def(\mathcal{U})$	Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and $\mathcal{U} = \langle S, Q \rangle$ be a structure of $\mathcal{RAF}$. RAF-$Def(\mathcal{U})$ is the set of arguments defeated by $\mathcal{U}$, that is: $$\text{RAF-}Def(\mathcal{U}) = \{a \in A \mid \exists \alpha \in Q \text{ s.t. } s(\alpha) \in S \text{ and } t(\alpha) = a\}$$ See Definition 61 on page 92
RAF-$Inh(\mathcal{U})$	Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and $\mathcal{U} = \langle S, Q \rangle$ be a structure of $\mathcal{RAF}$. RAF-$Inh(\mathcal{U})$ is the set of attacks inhibited by $\mathcal{U}$, that is: $$\text{RAF-}Inh(\mathcal{U}) = \{\alpha \in K \mid \exists \beta \in Q \text{ s.t. } s(\beta) \in S \text{ and } t(\beta) = \alpha\}$$ See Definition 61 on page 92
RAF-$Acc(\mathcal{U})$	Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and $\mathcal{U} = \langle S, Q \rangle$ be a structure of $\mathcal{RAF}$. RAF-$Acc(\mathcal{U})$ is the set of elements acceptable *w.r.t.* $\mathcal{U}$, that is: $\{e \in (A \cup K) \mid (\exists \alpha \in K \text{ s.t. } t(\alpha) = e) \implies (s(\alpha) \in \text{RAF-}Def(\mathcal{U}) \text{ or } \alpha \in \text{RAF-}Inh(\mathcal{U}))\}$. See Definition 62 on page 92
$\sqsubseteq$ $\sqsubseteq_{ar}$	Let $\mathcal{U} = \langle S, Q \rangle$ and $\mathcal{U}' = \langle S', Q' \rangle$ be any pair of structures. We write $\mathcal{U}' \sqsubseteq \mathcal{U}'$ *iff* $(S \cup Q) \subseteq (S' \cup Q')$ and $\mathcal{U} \sqsubseteq_{ar} \mathcal{U}'$ *iff* $S \subseteq S'$.
$\mathcal{L} = \langle \ell_A, \ell_K \rangle$	A structure labelling
$\mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF})$	$\mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF})$ is the set of structure labellings of $\mathcal{RAF}$ under the semantics σ
`Lab2Struct`(ℓ)	Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF. The function `Lab2Struct` transforms a labelling ℓ of $\mathcal{RAF}$ into its corresponding structure $\mathcal{U}$ of $\mathcal{RAF}$. See Definition 75 on page 107

Continued on next page ...

Table 18.5: RAF related Symbols (continued)

Symbol	Meaning
`Struct2Lab`$(\mathcal{U})$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF. The function `Struct2Lab` transforms a structure $\mathcal{U}$ of $\mathcal{RAF}$ into its corresponding labelling ℓ of $\mathcal{RAF}$. See Definition 75 on page 107
$SCCS_{raf}(\mathcal{RAF})$	The set of SCC_{raf} of $\mathcal{RAF}$. See Definition 87 on page 125
$Paths_{raf}(\mathcal{RAF})$	The set of RAF-paths of $\mathcal{RAF}$. See Definition 82 on page 123
$Cycles_{raf}(\mathcal{RAF})$	The set of RAF-cycles of $\mathcal{RAF}$. See Definition 83 on page 124
$ClosedWalk_{raf}(\mathcal{RAF})$	The set of RAF-closed-walks of $\mathcal{RAF}$. See Definition 84 on page 124
$PE_{raf}(\mathcal{RAF})$	The path-equivalence relation over $\mathcal{RAF}$. See Definition 87 on page 125
$\underset{\mathcal{RAF}}{\equiv}$ $\underset{\mathcal{RAF}}{\not\equiv}$	Other notations for the path-equivalence relation over given $\mathcal{RAF}$. "$x \underset{\mathcal{RAF}}{\equiv} y$" means "$(x,y) \in PE_{raf}(\mathcal{RAF})$" and "$x \underset{\mathcal{RAF}}{\not\equiv} y$" means "$(x,y) \notin PE_{raf}(\mathcal{RAF})$"
$\widetilde{\mathcal{RAF}} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t\rangle$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF. $\widetilde{\mathcal{RAF}}$ is a partial RAF of $\mathcal{RAF}$ where: • $\tilde{A} \subseteq A$ is a set representing arguments • $\tilde{K} \subseteq K$ is a set representing attacks • $\tilde{s} : \tilde{K} \to \{\texttt{true}, \texttt{false}\}$ is a boolean function that indicates whether or not an attack in $\tilde{K}$ has its source in $\tilde{A}$ defined as following: $$\forall \alpha \in \tilde{K}, \tilde{s}(\alpha) = \texttt{true} \text{ if } s(\alpha) \in \tilde{A} \text{ otherwise } \texttt{false}$$ • $\tilde{t} : \tilde{K} \to \{\texttt{true}, \texttt{false}\}$ is a boolean function that indicates whether or not an attack in $\tilde{K}$ has its target in $\tilde{A}$ $$\forall \alpha \in \tilde{K}, \tilde{t}(\alpha) = \texttt{true} \text{ if } t(\alpha) \in \tilde{A} \cup \tilde{K} \text{ otherwise } \texttt{false}$$ See Definition 92 on page 134
$\mathfrak{I} = \langle S^{inp}, Q^{inp}\rangle$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\widetilde{\mathcal{RAF}}$ is a partial RAF of $\mathcal{RAF}$. We denote by $\mathfrak{I}$ the input of $\widetilde{\mathcal{RAF}}$. See Definition 96 on page 136
$\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$	Let $\widetilde{\mathcal{RAF}}$ be a partial RAF and $\mathfrak{I}$ be an the input of $\widetilde{\mathcal{RAF}}$. The tuple $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ is called a "partial RAF with input", where $\mathcal{L}^{inp}$ is a structure labelling of the elements in S^{inp} and Q^{inp}. See Definition 96 on page 136

Continued on next page ...

Table 18.5: RAF related Symbols (continued)

Symbol	Meaning
$\widetilde{\mathcal{RAF}}_s = \langle A_s, K_s, s_s, t_s \rangle$	Let $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ be a partial RAF with input. $\widetilde{\mathcal{RAF}}_s$ is the standard RAF corresponding to it. See Definition 97 on page 138
ζ υ ρ θ $And_{\tilde{A}_s,\tilde{K}_s}$ $Not_{\tilde{A}_s}$ $Not_{\tilde{K}_s}$	Let $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ be a partial RAF with input. In the process of flattening created the standard RAF $\widetilde{\mathcal{RAF}}_s$ corresponding to it, several elements and sets of elements are created. See Definition 97 on page 138
$\mathscr{F}^{raf}(\widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp})$	Let $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ be a partial RAF with input, the canonical function $\mathscr{F}^{raf}(\widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp})$ computes the structure labellings corresponding to it. See Definition 99 on page 138
$\preccurlyeq$	A partial order relation between SCC_{raf}. See Definition 89 on page 132
$Dag_{scc}(\mathcal{RAF})$	A function that creates a the directed graph corresponding to the $\preccurlyeq$ relation over $\mathcal{RAF}$. See Definition 91 on page 133
$\mathscr{S}(\mathcal{RAF})$	A partition selector $\mathscr{S}$ is a function receiving as input a RAF $\mathcal{RAF} = \langle A, K, s, t \rangle$ and returning a set of partitions of $A \cup K$
$\mathscr{S}_{D\text{-}raf}$	The "*default RAF partition selector*" is the partition selector that produces all possible partitions of a RAF. See Definition 108 on page 159
$\mathscr{S}_{raf\text{-}USCC}$	A RAF partition selector that does not split SCC_{raf}. See Definition 109 on page 161
$\mathscr{S}_{raf\text{-}c\text{-}USCC}$	The AF partition selector that corresponds to $\mathscr{S}_{raf\text{-}USCC}$. See Definition 109 on page 161
$RAF\text{-}Cred_\sigma$, $RAF\text{-}Skep_\sigma$, $RAF\text{-}Ver_\sigma$, $RAF\text{-}Exists_\sigma$, $RAF\text{-}Exists_\sigma^{\neg\varnothing}$, $RAF\text{-}Unique_\sigma$	Respectively the *Credulous Acceptance*, *Skeptical Acceptance*, *Verification of a labelling*, *Existence of an extension/labelling/structure*, *Existence of a "non-empty" extension/labelling/structure* and the *Uniqueness of a solution* RAF decision problems. See Definition 79 on page 119

Table 18.6: Transformation related symbols: RAF and AF

Symbol	Meaning
$\texttt{Af2Raf}(\mathcal{AF})$	Let $\mathcal{AF} = \langle A, K\rangle$ be an AF. The function `Af2Raf` transforms an AF into a RAF by naming its attacks. See Definition 80 on page 121
$\texttt{Raf2Af}(\mathcal{RAF})$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF. The function `Raf2Af` transforms a RAF into an AF. See Definition 76 on page 115
Not_A Not_K $And_{A,K}$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF. In the process of flattening $\mathcal{RAF}$ with `Raf2Af`, several sets of arguments are created. Among those are Not_A, Not_K and $And_{A,K}$. See Definition 76 on page 115
$\neg x$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be the flattened version of $\mathcal{RAF}$. Let $x \in (A \cup K)$ be an element of $\mathcal{RAF}$. $\neg x$ is the created argument that represents the “negation” of element x. See Definition 76 on page 115 (Notice that $\neg x \in (Not_A \cup Not_K)$)
$s(\alpha).\alpha$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be the flattened version of $\mathcal{RAF}$. Let $\alpha \in K$ be an attack of $\mathcal{RAF}$. $s(\alpha).\alpha$ is the created argument that represents the “conjunction” of attack α with its source $s(\alpha)$. See Definition 76 on page 115 (Notice that $s(\alpha).\alpha \in And_{A,K}$)
$\varepsilon_{\mathcal{U}}$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be the flattened version of $\mathcal{RAF}$. Let $\mathcal{U}$ be a structure of $\mathcal{RAF}$. $\varepsilon_{\mathcal{U}}$ is the extension of $\mathcal{AF}$ corresponding to $\mathcal{U}$. See Definition 77 on page 116
$\texttt{rafLab2AfLab}(\mathcal{L})$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be the flattened version of $\mathcal{RAF}$. `rafLab2AfLab` is a function that transforms a structure labelling $\mathcal{L}$ of $\mathcal{RAF}$ into the labelling of $\mathcal{AF}$ corresponding to it. See Definition 78 on page 118
$\texttt{afLab2RafLab}(\ell)$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be the flattened version of $\mathcal{RAF}$. `afLab2RafLab` is a function that transforms a labelling ℓ of $\mathcal{AF}$ into the structure labelling of $\mathcal{AF}$ corresponding to it. See Definition 78 on page 118
$\texttt{Raf2Af}_{\texttt{maf}}(\mathcal{RAF})$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF. The function $\texttt{Raf2Af}_{\texttt{maf}}$ flattens a RAF into a MAF/AF. See Definition 68 on page 98
$\texttt{str2MafExt}(\mathcal{U})$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{MAF} = \texttt{Raf2Af}_{\texttt{maf}}(\mathcal{RAF})$. Let $\mathcal{U}$ be a structure labelling of $\mathcal{RAF}$. $\texttt{str2MafExt}(\mathcal{U})$ is the extension of $\mathcal{MAF}$ corresponding to $\mathcal{U}$. See Definition 69 on page 98

Continued on next page ...

Table 18.6: Transformation related symbols: RAF and AF (continued)

Symbol	Meaning
$\mathscr{S}_{raf\text{-}c}(\mathcal{AF})$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be the flattened version of $\mathcal{RAF}$. $\mathscr{S}_{raf\text{-}c}$ is a partition selector that selects partitions of $\mathcal{AF}$ that are "compliant" with $\mathcal{RAF}$. See Definition 101 on page 139
$\mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{AF})$	Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be the flattened version of $\mathcal{RAF}$. $\mathscr{S}_{D\text{-}raf\text{-}c}$, so-called "*default RAF-compliant partition selector*", is the partition selector that produces all the RAF-compliant partitions of $\mathcal{AF}$. See Definition 102 on page 140

Table 18.7: Complexity Symbols

Symbol	Meaning
$\mathscr{P}$	A decision problem. See Sections 16.4.1 and 16.4.2 on page 178 and on page 179
$\mathscr{C}$	A complexity class. See Section 16.4.1 on page 178
$\leq_{\texttt{P}}$, $\leq_{\texttt{L}}$	Respectively polynomial reduction and logarithmic in space reduction. See Section 16.4.4.1 on page 182
$\mathscr{C}$*-hard*	Let $\mathscr{P}$ be a problem and $\mathscr{C}$ be a complexity classes. $\mathscr{P} \in \mathscr{C}$*-hard* if and only if $\mathscr{P}$ is at least as hard as the other problems of the class $\mathscr{C}$. See Section 16.4.4.2 on page 183
$\mathscr{C}$*-c*	Let $\mathscr{P}$ be a problem and $\mathscr{C}$ be a complexity classes. $\mathscr{P} \in \mathscr{C}$*-c* if an only if $\mathscr{P} \in \mathscr{C}$*-hard* and $\mathscr{P} \in \mathscr{C}$. See Section 16.4.4.2 on page 183
$\texttt{L}$, $\texttt{P}$, $\texttt{NP}$, $\texttt{DP}$, $\texttt{DP}_2$, $\texttt{coNP}$, $\Sigma_2^{\texttt{P}}$, $\Pi_2^{\texttt{P}}$, $\Theta_2^{\texttt{P}}$, $\texttt{NP}^{\texttt{NP}}$, $\texttt{P}^{\texttt{NP}}$, $\texttt{NP}^{\texttt{coNP}}$	Several time complexity classes. See Section 16.4.3 on page 180

Table of figures

Table 18.8: Table of figures

Continued on next page ...

Table 18.8: Table of figures (continued)

Continued on next page ...

Table 18.8: Table of figures (continued)

Figure	Description
Figures 14.4 to 14.5 on pages 128–129	Different cases of RAF flattening that illustrate some interesting properties leading to Propositions 35 and 36 on page 129
Figures 14.7 to 14.8 on pages 131–132	Illustrations of the correspondance between SCC_{af} and SCC_{raf}
Figure 14.9 on page 133	$Dag_{scc}(\mathcal{RAF})$ corresponding to the RAF in Figure 14.7(a) on page 131
Figure 14.10 on page 134	A RAF example having attacks in cascade
Figure 14.11 on page 135	Illustration of partial RAFs
Figure 14.12 on page 137	Illustration of partial RAFs with inputs
Figure 14.13 on page 139	An example of standard RAFs
Figure 14.14 on page 141	An example of RAF-compliant partition
Figures 14.15 to 14.23 on pages 143–150	An illustration of the decomposability RAF semantics
Figures 14.24 and 14.25 on page 153 and on page 154	Schemas showing the demonstration overview of RAF semantics decomposability properties
Figures 14.26 to 14.29 on pages 156–158	These figures illustrate that given a RAF and its flattened version, there is a correspondence between its RAFs with input and AFs with input, between their respective standard RAFs and standard AFs, and finally, between the flattened version of a standard RAF and its corresponding standard AF
Figure 16.1 on page 171	Examples of graph
Figure 16.2 on page 172	Examples of weighted graph
Figure 16.3 on page 176	A weighted non-directed graph
Figure 16.4 on page 182	A schema of the *Polynomial Hierarchy*

Continued on next page ...

Table 18.8: Table of figures (continued)

Table of tables

Table 18.9: Table of tables

Continued on next page ...

Table 18.9: Table of tables (continued)

Glossary

Abstract Argumentation Abstract argumentation theory proposes methods to represent and deal with contentious information, and to draw conclusions or take decision from it. Such an abstract approach focuses on how arguments affect each other. Arguments are seen as generic entities which interact positively (support relation) or negatively (attack relation) with each other. 2, 70, 166, 210

adjacency See Definition 115 on page 171. 210

adjacency matrix See Definition 134. 53, 175, 176, 189, 210

AF trivial part Part of an AF that has a unique and fixed labelling that can be computed in linear time. See Section 4.2.1. i, 30, 210

AF with input See Definition 16. 17, 18, 48, 49, 153, 155, 157–159, 210

AF-extension A set of arguments of an AF. 210, 213

AFRA Argumentation Framework with Recursive Attacks. See Definition 49. 83, 210

AFRA-extension A set of arguments and attacks of an AFRA. 210, 213

algorithm A finite sequence of well-defined, computer-implementable instructions, typically to solve a class of specific problems or to perform a computation. 3, 5, 27, 70, 104, 123, 165, 210

application A mathematical application is a relationship between two sets in which each element of the former is related to a single element of the latter. 175, 177, 210

Argument Mining A sub research field of *Argumentation* interested in extracting arguments and their relations with each others, from natural language speeches (oral or written), in order to create a formal model to reason with. 2, 28, 210

argumentation A research field of *Artificial Intelligence* interested in managing contentious information. 2, 210

Argumentation Framework An argumentation framework, in the general sense, is a particular formalism to express argumentation problems. The expression has also come to mean an argumentation problem instance modeled after a particular formalism (by the way, a particular *argumentation framework*). 2, 3, 83, 210

Argumentation Reasoning A sub research field of *Argumentation* interested in reasoning over some argumentation model. It is useful to conclude, decide, convince, persuade or explain some issue. 2, 210

asymptotic Given a mathematical function f, the "asymptotic" behavior of f is its limiting behavior, that is for a large input (*w.r.t.* its domain definition). 179, 184, 185, 210

bijection A mathematical application establishing a relation between two sets such that any element of one is the image of a single element of the other. 12, 23, 167, 210

border argument Argument of a cluster structure being the source of an outward attack or the target of an inward attack. See Definition 30. 32, 36, 37, 42–45, 143, 145, 210

bottom-up decomposability See *glossary*: semantics decomposability. 210

bottom-up decomposable See *glossary*: semantics decomposability. 19, 20, 47, 142, 159–161, 210

cardinality A notion of size for sets. When a set is finite its cardinal is the number of elements it contains. 9, 12, 87, 94, 210

cluster structure See Definition 30. 32, 34–37, 42, 43, 48, 49, 210

clustering A method of data analysis consisting in dividing a data set into different homogeneous "clusters". The data in each subset share common characteristics, which usually correspond to similarity criteria that are defined by introducing distance measures between the data set elements. 3, 29, 30, 32, 34, 42, 51, 52, 54–57, 60, 79, 165, 166, 178, 186, 188–191, 210

compact enumeration representation See Chapter 6. 60, 64–66, 68, 210

complete A problem is said to be complete for a given complexity class $\mathscr{C}$ if it belongs to $\mathscr{C}$ and is $\mathscr{C}$-hard. See Section 16.4.4.2. 183, 210

complete An algorithm is said to be complete for a given problem if and only if it produces all the solutions of the wanted problem. See Section 4.3 as an example. 50, 165, 168, 210, 212

complete-based See Definition 38. 49, 157–161, 210

complete-compatible See Definition 21. 19, 20, 49, 210

completeness See *glossary*: complete. 32, 47, 48, 210

complexity class A theoretical group computational problems of similar hardness. See Section 16.4.1. 24, 120, 178, 179, 181, 183, 184, 210

computational complexity theory Computational complexity theory is a field of computer science whose purpose is to cluster computational problems into "complexity classes". Problems are gathered according to some criterion on the resources required to solve them. See Section 16.4. 170, 178, 179, 183, 184, 210

concavity A function is called concave if the line segment between any two points on the graph of the function lies below the graph between the two points. Some functions can have concave and convex (the opposite of concave) portions. 191, 210

configuration See Definition 34. 31, 36, 38–43, 46, 210

conjecture In mathematics, a conjecture is an assertion for which a proof is not yet known, but which is strongly believed to be true, in the absence of a counterexample. More generally speaking, its an opinion based on probabilities, appearances. 29, 210

connected component See Definition 127. 30–32, 34, 47, 64, 68, 174, 210

CPU time Amount of time for which a CPU was used for processing instructions of a computer program, as opposed to, for example, waiting for input/output operations. The CPU time is often measured in clock ticks or as a percentage of the CPU capacity. 57, 210, 216

CSP A Contraint Satisfaction Problem is a mathematical problem that looks for a configuration of object states (*i.e.* a mapping where each object has a particular state) that satisfies a certain number of constraints. See Section 17.1. 43–45, 186, 187, 210

datapoint Discrete unit of information. In a statistical or analytical context, a datapoint is usually derived from a measurement or research and can be represented numerically and/or graphically. 188, 190, 192, 210

decision problem A decision problem is a type of computational problem that has for output a boolean. That is, given an input the solution of the problem is whether "yes" (equivalently true or 1) or "no" (equivalently false or 0). We say that the problem "accepts" or "rejects" the input. See Section 16.4.2. 7, 23, 24, 55, 64–66, 120–122, 163, 167, 168, 210

degree matrix See Definition 135 on page 175. 176, 189, 210

directed graph See Definition 112. 210, 214

Dung's Argumentation Framework Dung introduced in [39] the seminal abstract argumentation framework. See Definition 1. 2, 23, 27, 86, 210

eigenvalue See Definition 133. 175, 178, 188, 190, 191, 210

eigenvector See Definition 133. 53, 175, 177, 178, 188, 190, 192, 210

enumeration problem Functional problem consisting in finding all the possible solutions of a computational problem. See Definition 29 for the enumeration problem in Argumentation. 27, 28, 51, 68, 165, 210

extension See *glossary*: AF-extension and AFRA-extension. 3, 8, 9, 85, 102, 103, 111, 166, 210

extension-based See *glossary*: extension-based semantics. 8, 12, 28, 210

extension-based semantics A semantics that produces extensions. 3, 8, 12, 16, 23, 85, 91, 97, 210, 213

field A field is one of the fundamental algebraic structures of general algebra. It is a set with two binary operations making possible addition, multiplication and the calculation of opposites and inverses, allowing the definition of subtraction and division operators. 210, 215, 217

flattening Process by which an argumentation framework is transformed into another one (often using a less complex formalism) while keeping some properties of interest. 3, 94, 98, 99, 101, 114, 115, 157, 158, 163, 167, 210

full decomposability See *glossary*: semantics decomposability. 210

fully decomposable See *glossary*: semantics decomposability. 18–20, 47, 48, 142, 159–161, 210

function problem Type of computational problem that has for output a more complex output than decision problems. See Definition 137. 28, 168, 183, 210

graph See *glossary*: non-directed graph, directed graph, weighted graph. 210

hard We consider that a computational problem is *hard* for a certain complexity class if an efficient algorithm solving it could be used to efficiently solve, by mean of reductions, all the problems in that given class. See Section 16.4.4.2. 121, 122, 183, 210

heuristic Any approach to problem solving or self-discovery that employs a practical method that is not guaranteed to be optimal, perfect, or rational, but is nevertheless sufficient for reaching an immediate, short-term goal or approximation. 3, 28, 188, 191, 210

Higher-Order Attack Argumentation Framework A argumentation framework that allow attacks to have as target an attack. See Parts IV and V. ii, 82, 83, 100, 210

incidence See Definition 114 on page 171. 210

independant partial RAF See Definition 95. 136, 210

induced AF See Definition 32. 34, 35, 49, 145, 210

inhibited Basically, the "inhibition" is the notion of defeat but for attacks. Given a structure $\mathcal{U}$, an attack α is said to be inhibited by $\mathcal{U}$ if there exists $\beta \in \mathcal{U}$ such that β has its source in $\mathcal{U}$ and its target is α. See Definition 61. 92, 121, 210, 217

input argument See Definition 15. 16–18, 80, 157, 210

inward attack Attack going *into* a cluster. See Definition 30. 32, 34, 42, 51, 54, 68, 143, 210

labelling Generally speaking, a labelling is an acceptance value mapping of a set of elements. See Definition 6 for AF, Definition 72 for RAF. 3, 11, 29, 101, 104, 165, 167, 210

labelling-based See *glossary*: labelling-based semantics. 8, 12, 28, 210

labelling-based semantics A semantics that produces labellings. 12, 23, 29, 167, 210, 214

laplacian matrix See Definition 136. 175, 176, 188, 190, 210

legally labelled See Definition 7 for AF and Definition 74 for RAF. 11, 104, 210, 216

linear application See *glossary*: linear transformation. 210

linear combination expression constructed from a set of terms by multiplying each term by a constant and adding the results. 210, 215

linear transformation A linear application (also called *linear transformation*) is an application between two vector spaces over a field which respects vector addition and scalar multiplication, and thus more generally preserves linear combinations. See *glossary*: field, linear combination, vector space. 175, 177, 210, 214

log-space function A polynomial time function that can be executed using at most a memory space logarithmic *w.r.t.* to the size of the input. Section 16.4.3.1. 121, 122, 210

matrix See Definition 132. 52, 53, 170, 175, 177, 188–192, 210

merge configuration See Definition 36. 39–44, 210

multi-threaded See *glossary*: multi-threading. 55, 57, 210

multi-threading a form of parallelization or division of work to enable simultaneous processing. Instead of giving a large workload to a single CPU core, threaded programs divide the work into several software tasks (threads). These tasks are processed in parallel by different CPU cores to save time. 210, 215

non-directed graph See Definition 112. 210, 214

order of magnitude See Definition 138. 184, 210

outward attack Attack coming out of a cluster. See Definition 30. 32, 143, 210

partial RAF See Definition 92. 134, 210

partial RAF with input See Definition 96. 136, 210

partition See Definition 111. An AF partition is a partition of its arguments. A RAF partition is a partition of its elements (arguments and attacks). 18–20, 32, 47, 53, 54, 56, 64, 135, 139, 140, 210

partition selector An application that produces some set of partitions from a given argumentation framework (whether AF or RAF). See Definition 23 for AF and Definition 100 for RAF. 20, 21, 139, 140, 159, 161, 210

path See Definition 121. 210

path-equivalence relation See Definition 128 for AF and Definition 86 for RAF. 20, 210, 216

polynomial reduction A problem reduction that is polynomial in time. See *glossary*: problem reduction. 120, 183, 210

problem reduction Basically, a procedure that transforms a given computational problem into another one. See Section 16.4.4.1 for details. 210, 215

RAF Recursive Argumentation Framework. See Definition 48. 83, 210

RAF path See Definition 82. 210

RAF-compliant See Definition 101. 139, 140, 155, 160, 161, 210

real time As opposed to CPU time (See *glossary*: CPU time), Real Time is the actual, real world, time that a process takes to run. 51, 210

reinstatement labelling A labelling is in which all elements are legally labelled. 102, 111, 167, 210

reunified labelling profile See Definition 37. 31, 42–44, 46, 210

SAT The decision problem consisting in deciding if a given propositional formula is satisfiable or not. 51, 70–73, 80, 179–181, 183, 210

SCC Given an AF (resp. a RAF), an SCC is a set of arguments (resp. elements) that are equivalent *w.r.t.* the PE_{af} (resp. *w.r.t.* the PE_{raf}) relation. See *glossary*: path-equivalence relation. 20, 54, 73, 78–80, 123, 174, 210

SCC decomposition See Section 7.2.2. ii, 78, 210

semantics Given an *argumentation framework*, a *semantics* corresponds to a formal way to say how the solution of the argumentation should be decided. 2–4, 8, 28, 83, 85, 101, 165, 210

semantics decomposability Properties of a semantics stating if the latter is computable in a distributed way, that is by considering sub-parts of an argumentation framework (whatever the formalism and for all instances). The top-down (resp. bottom-up) decomposability property ensures that the distributed computation made is complete (resp. sound). The fully decomposability property is the intersection of both properties. 3, 7, 16, 48, 101, 114, 123, 210, 212, 214, 217

set A set is a collection of distinct elements. 210

similarity criterion Criterion used in the spectral clustering method. See Section 17.2. 53, 210

sound An algorithm is said to be sound for a given problem if and only if it produces only valid solutions for the wanted problem. See Section 4.3 as an example. 50, 165, 168, 210, 216

soundness See *glossary*: sound. 32, 47, 48, 210

space In mathematics, a space is a set with additional structures, allowing to define objects analogous to those of usual geometry. The elements can be called points, vectors, functions, etc., depending on the context. See for an example *glossary*: vector space. 188, 190, 192, 210

sparse A graph is said to be sparse when its density is low. See Definition 131. 29, 53, 210, 216

sparsity See *glossary*: sparse. 210

spectral clustering A clustering method. See Section 17.2. 52, 53, 55, 57, 60, 165, 188, 192, 210

standard AF See Definition 17. 17, 18, 49, 153, 156–158, 210

standard RAF See Definition 97. 210

structure Given a RAF, a structure is a pair whose first element is a set of arguments and the second a set of attacks. 3, 91, 103, 167, 210

structure labelling Given a RAF, a structure labelling is a pair whose first element is a labelling of arguments and the second a labelling of attacks. 3, 101, 117, 118, 136, 138, 151, 155, 167, 210

structure-based See *glossary*: structure-based semantics. 167, 210

structure-based semantics A semantics that produces RAF structures. 3, 91, 92, 97, 210, 217

top-down decomposability See *glossary*: semantics decomposability. 210

top-down decomposable See *glossary*: semantics decomposability. 19, 20, 47, 142, 159–161, 210

USCC Union of SCCs. See Definition 27 for AF and Definition 109 for RAF. 21, 54–58, 60, 210

USCC Chain A clustering method. See Section 5.2.2. 54, 57, 210

USCC Tree A clustering method. See Section 5.2.2. 54, 57, 210

valid attack Given a RAF and a structure $\mathcal{U}$, an attack α is said to be valid if $\mathcal{U}$ does not inhibit α. See *glossary*: inhibited. 92, 210

variance In probability theory and statistics, variance is the expectation of the squared deviation of a random variable from its population mean or sample mean. 190, 191, 210

vector space Also called a linear space, a vector space is a set of objects called vectors, which may be added together and multiplied ("scaled") by numbers, called scalars. Scalars are often taken to be real numbers, but there are also vector spaces with scalar multiplication by complex numbers, rational numbers, or generally any field. 175, 210, 215, 216

walk See Definition 119. 210

weighted graph See Definition 113. 210, 214

well-founded partial RAF See Definition 94. 136, 210

Appendix 3: Proofs

Proofs of Part III: Contributions about AF

Proofs of Section 4.3: *AFDivider* soundness and completeness

In all the following proofs, by $\mathscr{L}_\sigma()$ we mean "the set of labellings under the semantics σ according to the mathematical definition of σ" whereas by $\mathscr{L}^*_\sigma()$ we mean "the set of labellings under the semantics σ computed with our algorithm". Thus, proving completeness is proving that $\mathscr{L}_\sigma() \subseteq \mathscr{L}^*_\sigma()$ and proving soundness is proving $\mathscr{L}^*_\sigma() \subseteq \mathscr{L}_\sigma()$.

We assume, in the following proofs, that the external existing solver used to compute the labellings of the induced AFs from the different cluster structures is sound and complete for the grounded, complete, stable and preferred semantics.

Proofs of Section 4.3.1: Relation between AFs with input and cluster structures

Proof of Proposition 9 on page 49. Let af' be the induced AF of κ under the context μ. Let $\langle \mathit{af}, \omega^{inp}, \mu, \omega^K \rangle$ be an AF with input (See Definition 16 on page 17) and $std\text{-}\mathcal{AF}$ be its standard argumentation framework (See Definition 17 on page 17). Let prove that:

$$\mathscr{L}_\sigma^{\mu(\kappa)} = \mathscr{F}_\sigma(\mathit{af}, \omega^{inp}, \mu, \omega^K)$$

By definition of the induced AF (Definition 32 on page 35), we have:

$$\mathit{af}' = \langle \omega', K' \rangle$$

Where:

- $D = \{a | a \in \omega \text{ and } (s,a) \in \omega^K \text{ and } s \in \mathtt{in}(\mu)\}$ being the set of arguments attacked by an $\mathtt{in}$-labelled argument in μ.
- $\omega' = \omega \setminus D$
- $K' = (K \cap (\omega' \times \omega')) \cup \{(a,a) | (s,a) \in \omega^K \text{ and } s \in \mathtt{und}(\mu)\}$

af' is so the AF obtained from af after the removal of the arguments attacked by an $\mathtt{in}$-labelled argument of the context and after the adding of self-attacks on each argument attacked by an $\mathtt{und}$-labelled argument of the context.

By definition of the standard argumentation framework (Definition 17 on page 17), we have:

$$std\text{-}\mathcal{AF} = \langle \omega \cup \mathcal{J}', (K \cap (\omega \times \omega)) \cup K'_{\mathcal{J}} \rangle$$

Where:

- $\mathcal{J}' = \omega^{inp} \cup \{a' | a \in \omega^{inp} \cap \mathtt{out}(\mu)\}$. See footnote.[1]
- $K'_{\mathcal{J}} = \omega^K \cup \{(a', a) | a \in \omega^{inp} \cap \mathtt{out}(\mu)\} \cup \{(a,a) | a \in \omega^{inp} \cap \mathtt{und}(\mu)\}$

Let $std\text{-}\mathcal{AF}_1$ be the AF corresponding to $std\text{-}\mathcal{AF} \downarrow_{\omega \cup \{a | a \in \mathtt{in}(\mu)\} \cup \{a | a \in \mathtt{und}(\mu)\}}$. Given that to obtain $std\text{-}\mathcal{AF}_1$ from $std\text{-}\mathcal{AF}$ we just have to remove the arguments labelled $\mathtt{out}$ in μ and those attacking them,[2] we have then:

$$\{\ell \downarrow_\omega | \ell \in \mathcal{L}_\sigma(std\text{-}\mathcal{AF}_1)\} = \{\ell \downarrow_\omega | \ell \in \mathcal{L}_\sigma(std\text{-}\mathcal{AF})\} \tag{18.1}$$

Let ω' be the set of arguments such that $\omega' = \omega \setminus D$ (as defined above). Let $std\text{-}\mathcal{AF}_2$ be the AF corresponding to $std\text{-}\mathcal{AF}_1 \downarrow_{\omega' \cup \{a | a \in \mathtt{und}(\mu)\}}$. Given that to obtain $std\text{-}\mathcal{AF}_2$ from $std\text{-}\mathcal{AF}_1$ we just have to remove the arguments labelled $\mathtt{in}$ in μ and those they attack,[3] we have then:

$$\{\ell \downarrow_{\omega'} | \ell \in \mathcal{L}_\sigma(std\text{-}\mathcal{AF}_2)\} = \{\ell \downarrow_{\omega'} | \ell \in \mathcal{L}_\sigma(std\text{-}\mathcal{AF}_1)\} \tag{18.2}$$

Considering the AF $std\text{-}\mathcal{AF}_2$, let $U = \{a | a \in \omega' \text{ and } (b,a) \in \omega^K \text{ and } b \in \omega^{inp} \cap \mathtt{und}(\mu)\}$ be the set of arguments of ω' attacked by an argument labelled $\mathtt{und}$ in μ. Let $u \in U$ be one of these arguments.

Given that u is attacked by an $\mathtt{und}$-labelled argument, u must be labelled $\mathtt{und}$ or $\mathtt{out}$. Nevertheless having an argument labelled $\mathtt{und}$ cannot have as consequence an argument labelled $\mathtt{in}$ or $\mathtt{out}$. And so, if u is labelled $\mathtt{out}$ in some labelling of $std\text{-}\mathcal{AF}_2$, it is not due to the set of arguments labelled $\mathtt{und}$ in μ.

Knowing this, we have:

$$\mathcal{L}_\sigma(af') = \{\ell \downarrow_{\omega'} | \ell \in \mathcal{L}_\sigma(std\text{-}\mathcal{AF}_2)\} \tag{18.3}$$

From Equation (18.3) and Equation (18.2), we have:

$$\mathcal{L}_\sigma(af') = \{\ell \downarrow_{\omega'} | \ell \in \mathcal{L}_\sigma(std\text{-}\mathcal{AF}_1)\} \tag{18.4}$$

Let ℓ^D be the labelling of the set of arguments D defined as following: $\ell^D = \{(a, \mathtt{out}) | a \in D\}$.
From Equation (18.4) and Equation (18.1), we have:

$$\{\ell \cup \ell^D | \ell \in \mathcal{L}_\sigma(af')\} = \{\ell \downarrow_\omega | \ell \in \mathcal{L}_\sigma(std\text{-}\mathcal{AF})\} \tag{18.5}$$

By definition of an induced labelling set (Definition 33 on page 35), we have:

$$\mathcal{L}_\sigma^{\mu(\kappa)} = \{\ell \cup \ell^D | \ell \in \mathcal{L}_\sigma(af')\} \tag{18.6}$$

By definition of a canonical local function (Definition 18 on page 18), we have:

$$\mathcal{F}_\sigma(af, \omega^{inp}, \mu, \omega^K) = \{\ell \downarrow_\omega | \ell \in \mathcal{L}_\sigma(std\text{-}\mathcal{AF})\} \tag{18.7}$$

[1] For each $a \in \omega^{inp} \cap \mathtt{out}(\mu)$ an argument a' is created.
[2] All these arguments are not in ω.
[3] All these arguments are not in ω'.

From Equations (18.5) to (18.7) on the previous page, we prove thus that:

$$\mathscr{L}_\sigma^{\mu(\kappa)} = \mathscr{F}_\sigma(af, \omega^{inp}, \mu, \omega^K)$$

■

Proofs of Section 4.3.2: Soundness and completeness

Proof of Proposition 10 on page 50. Let $af = \langle A, K\rangle$ be an AF, $\Omega = \{\omega_1, ..., \omega_n\}$ be a partition of A and $\{\kappa_1, ..., \kappa_n\}$ be the set of cluster structures corresponding to Ω, with each κ_i being defined as:

$$\kappa_i = \left\langle af{\downarrow_{\omega_i}}, I = \omega_i^K, O = K \cap (\omega_i \times (A \setminus \omega_i)), B = \{a | (a,b) \in O \text{ or } (b,a) \in I\}\right\rangle$$

Let σ be a top-down decomposable semantics.
Let $\mathscr{L}_\mathscr{D}^{\kappa_i}$ be the set of distinct labellings of κ_i according to the semantics σ.
Let $\mathscr{L}_\sigma^*(af)$ be the set of labellings of af according to σ obtained by Algorithm 2.
Let $\mathscr{L}_\sigma^{*\mu(\kappa_i)}$ be the set of labellings of κ_i under the context μ.
By definition we have (Definition 22 on page 19):

$$\mathscr{L}_\sigma(af) \subseteq \{\ell^{\omega_1} \cup ... \cup \ell^{\omega_n} | \ell^{\omega_i} \in \mathscr{F}_\sigma(af\downarrow_{\omega_i}, \omega_i^{inp}, (\bigcup_{j\in\{1,...,n\} \text{ s.t. } j\neq i} \ell^{\omega_j})\downarrow_{\omega_i^{inp}}, \omega_i^K)\} \tag{18.8}$$

Given that the labellings of all cluster structures are computed for every possible context, we have, by definition of the context and of the input arguments:

$$\forall i, \forall \ell^{inp} = (\bigcup_{j\in\{1,...,n\} \text{ s.t. } j\neq i} \ell^{\omega_j})\downarrow_{\omega_i^{inp}}, \quad \exists \mu^{\kappa_i} \text{ s.t. } \mu^{\kappa_i} = \ell^{inp} \tag{18.9}$$

Given that the external solver that computes the labellings of $af{\downarrow_{\omega_i}}$ according to the semantics σ is sound and complete, and considering $std\text{-}\mathcal{AF}$ being the standard AF w.r.t to the AF with input $\left\langle af{\downarrow_{\omega_i}}, \omega_i^{inp}, \mu^{\kappa_i}, \omega_i^K\right\rangle$, we have:

$$\forall i, \forall \mu^{\kappa_i}, \forall \ell^{std\text{-}\mathcal{AF}} \in \mathscr{L}_\sigma(std\text{-}\mathcal{AF}), \exists \ell \in \mathscr{L}_\sigma^{*\mu(\kappa_i)} \text{ s.t. } \ell = \ell^{std\text{-}\mathcal{AF}}\downarrow_{\omega_i} \tag{18.10}$$

So we have:

$$\forall i, \forall \mu^{\kappa_i}, \forall \ell^{std\text{-}\mathcal{AF}} \in \mathscr{L}_\sigma(std\text{-}\mathcal{AF}), \ell^{std\text{-}\mathcal{AF}}\downarrow_{\omega_i} \in \mathscr{L}_\mathscr{D}^{\kappa_i} \tag{18.11}$$

And so (following Definition 18 on page 18):

$$\forall \omega_i, \mathscr{F}_\sigma(af\downarrow_{\omega_i}, \omega_i^{inp}, (\bigcup_{j\in\{1,...,n\} \text{ s.t. } j\neq i} \ell^{\omega_j})\downarrow_{\omega_i^{inp}}, \omega_i^K) \subseteq \mathscr{L}_\mathscr{D}^{\kappa_i} \tag{18.12}$$

As a consequence and because of Equation (18.8) we have ($\prod$ denoting the cartesian product):

$$\mathscr{L}_\sigma(af) \subseteq \prod_{\kappa_i} \mathscr{L}_\mathscr{D}^{\kappa_i} \tag{18.13}$$

Let $\chi = \{\ell | \ell \in \prod_{\kappa_i} \mathscr{L}_\mathscr{D}^{\kappa_i}$ and $\exists a \in A$ s.t. a is illegally labelled in $\ell\}$ be the set of all possible incorrect labellings (*i.e.* the set of labellings in which there exists an argument that is not legally labelled).

We have, by definition of σ:

$$\mathscr{L}_\sigma(af) \subseteq (\prod_{\kappa_i} \mathscr{L}_{\mathscr{D}}^{\kappa_i}) \setminus \chi \tag{18.14}$$

Given that, for all computed labellings, we keep only the merged configuration, that is the most flexible possible configuration, our CSP modelisation does not add extra constraints. The proposed reunification removes, thus, only the labellings belonging to χ.

As a consequence, we have:

$$\mathscr{L}_\sigma(af) \subseteq \mathscr{L}_\sigma^*(af) \tag{18.15}$$

We prove so that for any top-down decomposable semantics σ our algorithm is complete, and so for the complete, stable and preferred semantics following Proposition 8 on page 21. ■

Proof of Proposition 11 on page 50.

- **Assertion 1:** Let $af = \langle A, K\rangle$ be an AF and $\Omega = \{\omega_1, ..., \omega_n\}$ be a partition of A corresponding to the clustering of af. Let σ be a fully decomposable and complete-based semantics and let ℓ^* be a labelling of af according to σ obtained by Algorithm 2.

  Let suppose that $\ell^* \notin \mathscr{L}_\sigma(af)$. We will prove that it is impossible with a reductio ad absurdum.

  As σ is a complete-based and fully decomposable semantics we can say that (Definition 19 on page 19):

  $$\ell^* \notin \{\ell^{\omega_1} \cup ... \cup \ell^{\omega_n} | \ell^{\omega_i} \in \mathscr{F}_\sigma(af{\downarrow_{\omega_i}}, \omega_i^{inp}, (\bigcup_{j \in \{1,...,n\} \text{ s.t. } j \neq i} \ell^{\omega_j}) \downarrow_{\omega_i^{inp}}, \omega_i^K)\} \tag{18.16}$$

  And so:

  $$\exists \omega_i \in \Omega \text{ s.t. } \ell^* \downarrow_{\omega_i} \notin \mathscr{F}_\sigma(af{\downarrow_{\omega_i}}, \omega_i^{inp}, (\bigcup_{j \in \{1,...,n\} \text{ s.t. } j \neq i} \ell^{\omega_j}) \downarrow_{\omega_i^{inp}}, \omega_i^K) \tag{18.17}$$

  In the following we denote by ω the particular ω_i for which Equation (18.17) holds in order to simplify the notation.

  Let $\kappa = \langle af{\downarrow_\omega}, I = \omega^K, O = K \cap (\omega \times (A \setminus \omega)), B = \{a | (a,b) \in O \text{ or } (b,a) \in I\}\rangle$ be the cluster structure corresponding to ω.

  Let μ be a context of κ such that $\mu = (\bigcup_{j \in \{1,...,n\} \text{ s.t. } \omega_j \neq \omega} \ell^{\omega_j}) \downarrow_{\omega^{inp}}$.

  Let $\mathscr{L}_\sigma^{*\mu(\kappa)}$ be the set of labellings of κ under the context μ produced by Algorithm 2.

  Let $\ell'^* \in \mathscr{L}_\sigma^{*\mu(\kappa)}$ be the labelling coinciding with $\ell^* \downarrow_\omega$ (*i.e.* $\ell'^* = \ell^* \downarrow_\omega$).

  We have so:

  $$\ell'^* \in \mathscr{L}_\sigma^{*\mu(\kappa)} \tag{18.18}$$

  Whereas:

  $$\ell'^* \notin \mathscr{F}_\sigma(af{\downarrow_\omega}, \omega^{inp}, \mu, \omega^K) \tag{18.19}$$

  And so:

  $$\mathscr{L}_\sigma^{*\mu(\kappa)} \neq \mathscr{F}_\sigma(af{\downarrow_\omega}, \omega^{inp}, \mu, \omega^K) \tag{18.20}$$

Nevertheless, according to Proposition 9 on page 49 we must have:

$$\mathscr{L}_\sigma^{*\mu(\kappa)} = \mathscr{F}_\sigma(af{\downarrow_\omega}, \omega^{inp}, \mu, \omega^K) \tag{18.21}$$

Thus, there is a contradiction between Equation (18.20) on the previous page and Equation (18.21).

From this contradiction we can conclude that:

$$\mathscr{L}_\sigma^*(af) \subseteq \mathscr{L}_\sigma(af) \tag{18.22}$$

We prove so that for any fully decomposable and complete-based semantics σ our algorithm is sound, and so for the *complete* and *stable* semantics, following Proposition 8 on page 21.

- **Assertion 2:** Let $af = \langle A, K\rangle$ be an AF. Given that Algorithm 2 is *complete* for the *preferred* semantics (see Proposition 10 on page 50), $\mathscr{L}_\sigma^*$, the set of all labellings reunified from the different clusters obtained in Algorithm 2 line 7, contains all the preferred labellings of af.

  In Algorithm 2 line 8, we keep from $\mathscr{L}_\sigma^*$ only the maximal (w.r.t $\subseteq$ of $\mathtt{in}$-labelled arguments) labellings, that are by definition the preferred labellings. As a consequence, $\mathscr{L}_{pr}$ contains only and all the *preferred* labellings of af.

  Algorithm 2 is, thus, sound and complete for the *preferred* semantics.

■

Proof of Proposition 12 on page 50. (Completeness of Algorithm 1 + Algorithm 2). Let $\mathcal{AF} = \langle A, K\rangle$ be an AF, ℓ_{gr} be its grounded labelling, $\mathcal{AF}_{hard} = \mathcal{AF}{\downarrow_{\{a|a\in A, \ell_{gr}(a)=\mathtt{und}\}}}$ be the hard part of $\mathcal{AF}$ and $\{af_1 = \langle A_1, K_1\rangle, ..., af_n = \langle A_n, K_n\rangle\}$ be the set of AFs obtained from $\mathcal{AF}_{hard}$ components.

Let σ be the *complete*, *stable* or *preferred* semantics.

Let $\mathscr{L}_\sigma^*(\mathcal{AF})$ be the set of labellings obtained from Algorithm 1.

Let $\mathscr{L}_\sigma^*(af_i)$ be the set of labellings obtained from Algorithm 2 for the component af_i.

Let $\mathscr{L}_\sigma(\mathcal{AF})$ be the set of labellings of $\mathcal{AF}$.

Let $\Omega = \{\omega_{gr}, A_1, ..., A_n\}$ be a partition of A with $\omega_{gr} = \{a | a \in \mathtt{in}(\ell_{gr})$ or $a \in \mathtt{out}(\ell_{gr})\}$.

Let $\ell \in \mathscr{L}_\sigma(\mathcal{AF})$ be a labelling of $\mathcal{AF}$ according to σ.

Given that (following Definition 18 on page 18):

$$\mathscr{F}_\sigma(\mathcal{AF}{\downarrow_{\omega_{gr}}}, \omega_{gr}^{inp}, (\bigcup_{i\in\{1,...,n\}} \ell^{A_i}){\downarrow_{\omega_{gr}^{inp}}}, \omega_{gr}^K) = \{\ell_{gr}\} \tag{18.23}$$

We have by definition of top-down decomposable semantics (following Definition 22 on page 19):

$$\mathscr{L}_\sigma(\mathcal{AF}) \subseteq \{\ell_{gr} \cup \bigcup_{A_i} \ell^{A_i}\} \text{ with } \ell^{A_i} \in \mathscr{F}_\sigma(\mathcal{AF}{\downarrow_{A_i}}, A_i^{inp}, (\bigcup_{j\in\{1,...,n\} \text{ s.t. } j\neq i} \ell^{A_j}){\downarrow_{A_i^{inp}}}, A_i^K) \tag{18.24}$$

Given that Algorithm 2 is complete for top-down decomposable semantics (*i.e.* $\forall p \in \Omega, \mathscr{L}_\sigma(\mathcal{AF}{\downarrow_p}) \subseteq \mathscr{L}_\sigma^*(\mathcal{AF}{\downarrow_p})$),

$$\forall A_i, \ell^{A_i} \in \mathscr{L}_\sigma^*(af_i) \tag{18.25}$$

Furthermore:

$$\forall \ell^* \in \mathscr{L}^*_\sigma(\mathcal{AF}), \ell^* = \ell_{gr} \cup \bigcup \ell^*_i, \text{ with } \ell^*_i \in \mathscr{L}^*_\sigma(af_i) \tag{18.26}$$

We have so:

$$\{\ell_{gr} \cup \bigcup_{A_i} \ell^{A_i}\} = \mathscr{L}^*_\sigma(\mathcal{AF}) \tag{18.27}$$

Finally, we have:

$$\mathscr{L}_\sigma(\mathcal{AF}) \subseteq \mathscr{L}^*_\sigma(\mathcal{AF}) \tag{18.28}$$

We prove so that our algorithm is complete for the *complete*, *stable* and *preferred* semantics. ■

Proof of Proposition 13 on page 50.

- **Assertion 1:** Algorithm 1 is sound for the *stable* and *complete* semantics.

  Let $\mathcal{AF} = \langle A, K\rangle$ be an AF, ℓ_{gr} be its grounded labelling, $\mathcal{AF}_{hard} = \mathcal{AF}\downarrow_{\{a|a\in A, \ell_{gr}(a)=\mathtt{und}\}}$ be the hard part of $\mathcal{AF}$ and $\{af_1 = \langle A_1, K_1\rangle, ..., af_n = \langle A_n, K_n\rangle\}$ be the set of AFs obtained from $\mathcal{AF}_{hard}$ components.

  Let σ be the *complete* or *stable* semantics.

  Let $\mathscr{L}^*_\sigma(\mathcal{AF})$ be the set of labellings of $\mathcal{AF}$ obtained from Algorithm 1.

  Let $\mathscr{L}_\sigma(\mathcal{AF})$ be the set of labellings of $\mathcal{AF}$.

  Let $\ell^* \in \mathscr{L}^*_\sigma(\mathcal{AF})$ be a labelling of $\mathcal{AF}$ computed by Algorithm 1.

  Let $\mathscr{L}^*_\sigma(af_i)$ be the set of labellings of af_i obtained from Algorithm 2.

  Following Algorithm 1, we have:

  $$\ell^* = \ell_{gr} \cup \bigcup \ell^*_i, \text{ with } \ell^*_i \in \mathscr{L}^*_\sigma(af_i) \tag{18.29}$$

  Let $\Omega = \{\omega_{gr}, A_1, ..., A_n\}$ be a partition of A with $\omega_{gr} = \{a | a \in \mathtt{in}(\ell_{gr}) \text{ or } a \in \mathtt{out}(\ell_{gr})\}$.

  We have (following Definition 18 on page 18):

  $$\mathscr{F}_\sigma(\mathcal{AF}\downarrow_{\omega_{gr}}, \omega^{inp}_{gr}, (\bigcup_{i\in\{1,...,n\}} \ell^{A_i})\downarrow_{\omega^{inp}_{gr}}, \omega^K_{gr}) = \{\ell_{gr}\} \tag{18.30}$$

  Because σ is a fully decomposable semantics we have so (Definition 19 on page 19):

  $$\mathscr{L}_\sigma(\mathcal{AF}) = \{\ell_{gr} \cup \bigcup_{A_i} \ell^{A_i}\} \text{ with } \ell^{A_i} \in \mathscr{F}_\sigma(af\downarrow_{A_i}, A^{inp}_i, (\bigcup_{j\in\{1,...,n\} \text{ s.t. } j\neq i} \ell^{A_j})\downarrow_{A^{inp}_i}, A^K_i) \tag{18.31}$$

  Given that Equation (18.31) holds and that Algorithm 2 is sound for fully decomposable semantics (*i.e.* $\forall p \in \Omega$, $\mathscr{L}^*_\sigma(\mathcal{AF}\downarrow_p) \subseteq \mathscr{L}_\sigma(\mathcal{AF}\downarrow_p)$), we have:

  $$\ell^* \in \mathscr{L}_\sigma(\mathcal{AF}) \tag{18.32}$$

  And thus:

  $$\mathscr{L}^*_\sigma(\mathcal{AF}) \subseteq \mathscr{L}_\sigma(\mathcal{AF}) \tag{18.33}$$

  We prove so that for the *complete* and *stable* semantics our algorithm is sound.

- **Assertion 2:** Algorithm 1 is sound for the *preferred* semantics.

  Let $\mathcal{AF} = \langle A, K \rangle$ be an AF, ℓ_{gr} be its grounded labelling, $\mathcal{AF}_{hard} = \mathcal{AF}\downarrow_{\{a|a\in A, \ell_{gr}(a)=\mathtt{und}\}}$ be the hard part of $\mathcal{AF}$ and $\{af_1 = \langle A_1, K_1\rangle, ..., af_n = \langle A_n, K_n\rangle\}$ be the set of AFs obtained from $\mathcal{AF}_{hard}$ components.

  Let $\mathscr{L}^*_{pr}(\mathcal{AF})$ be the set of labellings of $\mathcal{AF}$ obtained from Algorithm 1.

  Let $\mathscr{L}_{pr}(\mathcal{AF})$ be the set of labellings of $\mathcal{AF}$.

  Let $\mathscr{L}^*_{pr}(af_i)$ be the set of labellings of af_i obtained from Algorithm 2.

  Following Algorithm 1, we have:

  $$\mathscr{L}^*_{pr}(\mathcal{AF}) = \{\ell_{gr} \cup \ell^{A_1} \cup ... \cup \ell^{A_n} | \ell^{A_i} \in \mathscr{L}^*_{pr}(af_i)\} \tag{18.34}$$

  Let $A_0 = \{a | a \in \mathtt{in}(\ell_{gr})$ or $a \in \mathtt{out}(\ell_{gr})\}$ be the fixed part of $\mathcal{AF}$. The set of argument sets $\Omega = \{A_0, A_1, ..., A_n\}$ is then a partition of A.

  By definition of the grounded labelling, we have:

  $$\exists a \in A \text{ s.t. } \ell_{gr}(a) = \mathtt{und} \implies (\forall a' \in A \text{ s.t. } (a', a) \in K, \ell_{gr}(a) \neq \mathtt{in}) \tag{18.35}$$

  Given that:

  $$\mathtt{und}(\ell_{gr}) \cap A_0 = \varnothing \tag{18.36}$$

  And that by construction of A_0:

  $$\forall i \in \{1, ..., n\}, \forall a \in A_i, \ell_{gr}(a) = \mathtt{und} \tag{18.37}$$

  The consequence of Equation (18.35) is:

  $$\forall i \in \{1, ..., n\}, \forall (a', a) \in K \text{ s.t. } a' \in A_0 \text{ and } a \in A_i, \ell_{gr}(a') = \mathtt{out} \tag{18.38}$$

  Let $\mathcal{AF}' = \langle A, K \setminus \{(a', a) | (a' \in A_0$ and $a \notin A_0)$ or $(a' \notin A_0$ and $a \in A_0)\}\rangle$ be the AF constructed by removing from $\mathcal{AF}$ the attacks between its fixed part and its non fixed part. As in $\mathcal{AF}$ all arguments in the fixed part attacking arguments outside the fixed part is labelled $\mathtt{out}$ (Equation (18.38)) their attacks have no effect. The consequence is the following:

  $$\mathscr{L}_{pr}(\mathcal{AF}') = \mathscr{L}_{pr}(\mathcal{AF}) \tag{18.39}$$

  Notice that $\mathcal{AF}'$ has $n+1$ connected components corresponding to the partition Ω. Given that there is no connection (attack) between those connected components, each $A_i \in \Omega$ is an $USCC_{af}$ (see Definition 27 on page 21). As a consequence, following the definition of $\mathscr{S}_{USCC}$ (Definition 27 on page 21), we have:

  $$\Omega \in \mathscr{S}_{USCC}(\mathcal{AF}') \tag{18.40}$$

  As the preferred semantics is fully decomposable w.r.t. $\mathscr{S}_{USCC}$ (Definition 27 on page 21), we have (following Definition 24 on page 20):

  $$\mathscr{L}_{\sigma}(\mathcal{AF}') = \{\ell^{A_0} \cup ... \cup \ell^{A_n} | \ell^{A_i} \in \mathscr{F}_{pr}(\mathcal{AF}\downarrow_{A_i}, A_i^{inp}, (\bigcup_{j\in\{0,...,n\} \text{ s.t. } j\neq i} \ell^{A_j})\downarrow_{A_i^{inp}}, A_i^K)\} \tag{18.41}$$

Notice that:

$$\mathscr{F}_{pr}(\mathcal{AF}' \downarrow_{A_0}, A_0^{inp}, (\bigcup_{i \in \{1,\dots,n\}} \ell^{A_i}) \downarrow_{A_0^{inp}}, A_0^K) = \{\ell_{gr}\} \tag{18.42}$$

Notice also that, given Algorithm 2 is sound and complete for the *preferred* semantics (Propositions 10 and 11 on page 50), we have:

$$\forall i \in \{1,\dots,n\}, \mathscr{F}_{pr}(\mathcal{AF}' \downarrow_{A_i}, A_i^{inp}, (\bigcup_{j \in \{1,\dots,n\} \text{ s.t. } j \neq i} \ell^{A_j}) \downarrow_{A_i^{inp}}, A_i^K) = \mathscr{L}^*_{pr}(af_i) \tag{18.43}$$

From the Equations (18.41) to (18.43) on pages 225–226, we have:

$$\mathscr{L}_{pr}(\mathcal{AF}') = \{\ell_{gr} \cup \ell^{A_1} \cup \dots \cup \ell^{A_n} | \ell^{A_i} \in \mathscr{L}^*_{pr}(af_i)\} \tag{18.44}$$

From Equations (18.39) and (18.44) on the previous page and on this page, we have:

$$\mathscr{L}_{pr}(\mathcal{AF}) = \{\ell_{gr} \cup \ell^{A_1} \cup \dots \cup \ell^{A_n} | \ell^{A_i} \in \mathscr{L}^*_{pr}(af_i)\} \tag{18.45}$$

Finally, from Equations (18.34) and (18.45) on the previous page and on this page we have:

$$\mathscr{L}^*_{pr}(\mathcal{AF}) = \mathscr{L}_{pr}(\mathcal{AF}) \tag{18.46}$$

We prove so that Algorithm 1, when using Algorithm 2 to compute the component labellings, is sound and complete for the *preferred* semantics.

■

Proofs of Chapter 6: Compact representation

Proof of Proposition 14 on page 66. Following Algorithm 1, let $\mathcal{AF}$ be decomposed into n components and let $\texttt{Comp}_\sigma(\mathcal{AF}) = \{\mathscr{L}_\sigma(af_0), \dots, \mathscr{L}_\sigma(af_n)\}$ be the compact enumeration representation corresponding to $\sigma(\mathcal{AF})$.

Assertion 1: $Cred_\sigma(\mathcal{AF}, a) \equiv Comp\text{-}Cred_\sigma(\mathcal{AF}, a)$.

- Case 1: If $Cred_\sigma(\mathcal{AF}, a)$ is true, then $Comp\text{-}Cred_\sigma(\mathcal{AF}, a)$ is true.

  If $Cred_\sigma(\mathcal{AF}, a)$ is true, then there exists a labelling $\ell \in \sigma(\mathcal{AF})$ such that $a \in \mathtt{in}(\ell)$.

  Given that $\sigma(\mathcal{AF}) \neq \varnothing$ and that Algorithm 1 is complete for σ then $\forall \mathscr{L}_\sigma(af_i) \in \texttt{Comp}_\sigma(\mathcal{AF})$, we have: $\mathscr{L}_\sigma(af_i) \neq \varnothing$.

  Moreover, as Algorithm 1 is complete for σ, there exists a combination of component labellings $\ell_0, \dots, \ell_n$, with $\ell_i \in \mathscr{L}_\sigma(af_i)$ and $\mathscr{L}_\sigma(af_i) \in \texttt{Comp}_\sigma(\mathcal{AF})$, such that: $\ell = \bigcup_{i=0}^n \ell_i$. As a consequence, there exists $i \in \{0, \dots, n\}$ such that $a \in \mathtt{in}(\ell_i)$.

  We have so: $Comp\text{-}Cred_\sigma(\mathcal{AF}, a)$ being true.

- Case 2: If $Comp\text{-}Cred_\sigma(\mathcal{AF}, a)$ is true, then $Cred_\sigma(\mathcal{AF}, a)$ is true.

  If $Comp\text{-}Cred_\sigma(\mathcal{AF}, a)$ is true, then $\forall \mathscr{L}_\sigma(af_i) \in \texttt{Comp}_\sigma(\mathcal{AF})$, $\mathscr{L}_\sigma(af_i) \neq \varnothing$, and there exists a set $\mathscr{L}_\sigma(af) \in \texttt{Comp}_\sigma(\mathcal{AF})$ such that $\exists \ell_j \in \mathscr{L}_\sigma(af)$ s.t. $\ell_j(a) = \mathtt{in}$. As no component labelling set is empty, following Algorithm 1, there is thus a combination of component labellings $\ell_0, ..., \ell_n$, with $\ell_i \in \mathscr{L}_\sigma(af_i)$ and $\mathscr{L}_\sigma(af_i) \in \texttt{Comp}_\sigma(\mathcal{AF})$, including that particular ℓ_j. Let $\ell = \bigcup_{i=0}^{n} \ell_i$. Given that Algorithm 1 is sound for σ, then we have: $\ell \in \sigma(\mathcal{AF})$. As $a \in in(\ell)$, then we have: $Cred_\sigma(\mathcal{AF}, a)$ being true.

We prove so that: $Cred_\sigma(\mathcal{AF}, a) \equiv Comp\text{-}Cred_\sigma(\mathcal{AF}, a)$.

Assertion 2: $Skep_\sigma(\mathcal{AF}, a) \equiv Comp\text{-}Skep_\sigma(\mathcal{AF}, a)$.

- Case 1: If $Skep_\sigma(\mathcal{AF}, a)$ is true, then $Comp\text{-}Skep_\sigma(\mathcal{AF}, a)$ is true.

  If $Skep_\sigma(\mathcal{AF}, a)$ is true, then $\sigma(\mathcal{AF}) \neq \varnothing$ and $\forall \ell \in \sigma(\mathcal{AF})$, we have: $a \in in(\ell)$.

  Given that $\sigma(\mathcal{AF}) \neq \varnothing$ and that Algorithm 1 is complete for σ then $\forall \mathscr{L}_\sigma(af_i) \in \texttt{Comp}_\sigma(\mathcal{AF})$, we have: $\mathscr{L}_\sigma(af_i) \neq \varnothing$.

  Let $\mathscr{L}_\sigma(af_j)$ be the labelling set of the particular component to which a belongs and let, for $i \in (\{0, ..., n\} \setminus \{j\})$, $\mathscr{L}_\sigma(af_i) \in \texttt{Comp}_\sigma(\mathcal{AF})\}$ be a set of labellings of the compact enumeration representation different from $\mathscr{L}_\sigma(af_j)$. As Algorithm 1 is sound for σ, then $\forall \ell_j \in \mathscr{L}_\sigma(af_j)$, $\forall \ell_0 \in \mathscr{L}_\sigma(af_0)$, ..., $\forall \mathscr{L}_\sigma(\ell_n) \in af_n$, $(\ell_j \bigcup_{i=0}^{n} \ell_i) \in \sigma(\mathcal{AF})$. As $\forall \ell \in \sigma(\mathcal{AF})$, $a \in in(\ell)$, then $a \in in(\ell_j \bigcup_{i=0}^{n} \ell_i)$. As a consequence: $a \in in(\ell_j)$.

  We have so: $Comp\text{-}Skep_\sigma(\mathcal{AF}, a)$ being true.

- Case 2: If $Comp\text{-}Skep_\sigma(\mathcal{AF}, a)$ is true, then $Skep_\sigma(\mathcal{AF}, a)$ is true.

  If $Comp\text{-}Skep_\sigma(\mathcal{AF}, a)$ is true, then $\forall \mathscr{L}_\sigma(af_i) \in \texttt{Comp}_\sigma(\mathcal{AF})$, $\mathscr{L}_\sigma(af_i) \neq \varnothing$, and there exists a set $\mathscr{L}_\sigma(af_j) \in \texttt{Comp}_\sigma(\mathcal{AF})$ such that $\forall \ell_j \in \mathscr{L}_\sigma(af_j), \ell_j(a) = in$.

  As no component labelling set is empty and as Algorithm 1 is sound for σ, we have, with $\mathscr{L}_\sigma(af_i) \in \texttt{Comp}_\sigma(\mathcal{AF})$ for $i \in (\{0, ..., n\} \setminus \{j\})$, the following assertions: $\forall \ell_j \in \mathscr{L}_\sigma(af_j)$, $\forall \ell_0 \in \mathscr{L}_\sigma(af_0)$, ..., $\forall \mathscr{L}_\sigma(\ell_n) \in af_n$, $(\ell_j \bigcup_{i=0}^{n} \ell_i) \in \sigma(\mathcal{AF})$ and $a \in in(\ell_j \bigcup_{i=0}^{n} \ell_i)$.

  Given that Algorithm 1 is complete for σ and that $\forall \ell = \ell_j \bigcup_{i=0}^{n} \ell_i$, we have $a \in in(\ell)$, then we have: $Skep_\sigma(\mathcal{AF}, a)$ being true.

We prove so that: $Skep_\sigma(\mathcal{AF}, a) \equiv Comp\text{-}Skep_\sigma(\mathcal{AF}, a)$.

Assertion 3: $Ver_\sigma(\mathcal{AF}, \ell) \equiv Comp\text{-}Ver_\sigma(\mathcal{AF}, \ell)$

- Case 1: If $Ver_\sigma(\mathcal{AF}, \ell)$ is true, then $Comp\text{-}Ver_\sigma(\mathcal{AF}, a)$ is true.

  If $Ver_\sigma(\mathcal{AF}, \ell)$ is true, then $\ell \in \sigma(\mathcal{AF})$. Given that Algorithm 1 is complete for σ, then there exists a combination of component labellings $\ell_0, ..., \ell_n$, with $\ell_i \in \mathscr{L}_\sigma(af_i)$ and $\mathscr{L}_\sigma(af_i) \in \texttt{Comp}_\sigma(\mathcal{AF})$, such that: $\ell = \bigcup_{i=0}^{n} \ell_i$. As a consequence we have: $Comp\text{-}Ver_\sigma(\mathcal{AF}, a)$ is true.

- Case 2: If $Comp\text{-}Ver_\sigma(\mathcal{AF}, a)$ is true, then $Ver_\sigma(\mathcal{AF}, \ell)$ is true.

  If $Comp\text{-}Ver_\sigma(\mathcal{AF}, \ell)$ is true, there exists a combination of component labellings $\ell_0, ..., \ell_n$, with $\ell_i \in \mathscr{L}_\sigma(af_i)$ and $\mathscr{L}_\sigma(af_i) \in \texttt{Comp}_\sigma(\mathcal{AF})$, such that: $\ell = \bigcup_{i=0}^{n} \ell_i$. Given that Algorithm 1 is sound for σ then $\ell \in \sigma(\mathcal{AF})$. As a consequence we have: $Ver_\sigma(\mathcal{AF}, a)$ is true.

We prove so that: $Ver_\sigma(\mathcal{AF}, \ell) \equiv Comp\text{-}Ver_\sigma(\mathcal{AF}, \ell)$.

Assertion 4: $Exists_\sigma(\mathcal{AF}) \equiv Comp\text{-}Exists_\sigma(\mathcal{AF})$

- Case 1: If $\mathit{Exists}_\sigma(\mathcal{AF})$ is true, then $\mathit{Comp\text{-}Exists}_\sigma(\mathcal{AF})$ is true.

  If $\mathit{Exists}_\sigma(\mathcal{AF})$ is true, then $\sigma(\mathcal{AF}) \neq \varnothing$. Let $\ell \in \sigma(\mathcal{AF})$ be a labelling. As Algorithm 1 is complete for σ, there exists thus a combination of component labellings $\ell_0,...,\ell_n$, with $\ell_i \in \mathscr{L}_\sigma(\mathit{af}_i)$ and $\mathscr{L}_\sigma(\mathit{af}_i) \in \mathtt{Comp}_\sigma(\mathcal{AF})$ such that $\ell = \bigcup_{i=0}^n \ell_i$. As a consequence, we have: $\forall \mathscr{L}_\sigma(\mathit{af}_i) \in \mathtt{Comp}_\sigma(\mathcal{AF})$, $\mathscr{L}_\sigma(\mathit{af}_i) \neq \varnothing$.

  We prove so that if $\mathit{Exists}_\sigma(\mathcal{AF})$ is true, then $\mathit{Comp\text{-}Exists}_\sigma(\mathcal{AF})$ is true.

- Case 2: If $\mathit{Comp\text{-}Exists}_\sigma(\mathcal{AF})$ is true, then $\mathit{Exists}_\sigma(\mathcal{AF})$ is true.

  If $\mathit{Comp\text{-}Exists}_\sigma(\mathcal{AF})$ is true, then $\forall \mathscr{L}_\sigma(\mathit{af}_i) \in \mathtt{Comp}_\sigma(\mathcal{AF})$, $\mathscr{L}_\sigma(\mathit{af}_i) \neq \varnothing$. Let $\ell = \bigcup_{i=0}^n \ell_i$ be a labelling with $\ell_{i\in\{0,...,n\}} \in \mathscr{L}_\sigma(\mathit{af}_i)$. As Algorithm 1 is sound for σ, then $\ell \in \sigma(\mathcal{AF})$.

  We prove so that if $\mathit{Comp\text{-}Exists}_\sigma(\mathcal{AF})$ is true, then $\mathit{Exists}_\sigma(\mathcal{AF})$ is true.

We prove so that: $\mathit{Exists}_\sigma(\mathcal{AF}) \equiv \mathit{Comp\text{-}Exists}_\sigma(\mathcal{AF})$.

Assertion 5: $\mathit{Exists}_\sigma^{\neg\varnothing}(\mathcal{AF}) \equiv \mathit{Comp\text{-}Exists}_\sigma^{\neg\varnothing}(\mathcal{AF})$

- Case 1: If $\mathit{Exists}_\sigma^{\neg\varnothing}(\mathcal{AF})$ is true, then $\mathit{Comp\text{-}Exists}_\sigma^{\neg\varnothing}(\mathcal{AF})$ is true.

  If $\mathit{Exists}_\sigma^{\neg\varnothing}(\mathcal{AF})$ then $\exists \ell \in \sigma(\mathcal{AF})$ s.t. $\mathtt{in}(\ell) \neq \varnothing$. As Algorithm 1 is complete for σ, there exists thus a combination of component labellings $\ell_0,...,\ell_n$, with $\ell_i \in \mathscr{L}_\sigma(\mathit{af}_i)$ and $\mathscr{L}_\sigma(\mathit{af}_i) \in \mathtt{Comp}_\sigma(\mathcal{AF})$ such that $\ell = \bigcup_{i=0}^n \ell_i$. As a consequence, we have: $\forall \mathscr{L}_\sigma(\mathit{af}_i) \in \mathtt{Comp}_\sigma(\mathcal{AF})$, $\mathscr{L}_\sigma(\mathit{af}_i) \neq \varnothing$ and $\exists i \in \{0,...,n\}$ s.t. $\mathtt{in}(\ell_i) \neq \varnothing$.

  We prove so that if $\mathit{Exists}_\sigma^{\neg\varnothing}(\mathcal{AF})$ is true, then $\mathit{Comp\text{-}Exists}_\sigma^{\neg\varnothing}(\mathcal{AF})$ is true.

- Case 2: If $\mathit{Comp\text{-}Exists}_\sigma^{\neg\varnothing}(\mathcal{AF})$ is true, then $\mathit{Exists}_\sigma^{\neg\varnothing}(\mathcal{AF})$ is true.

  If $\mathit{Comp\text{-}Exists}_\sigma^{\neg\varnothing}(\mathcal{AF})$ is true, then $\forall \mathscr{L}_\sigma(\mathit{af}_i) \in \mathtt{Comp}_\sigma(\mathcal{AF})$, $\mathscr{L}_\sigma(\mathit{af}_i) \neq \varnothing$ and $\exists \mathscr{L}_\sigma(\mathit{af}_i)$ s.t. $\ell_i \in \mathscr{L}_\sigma(\mathit{af}_i)$ and $\mathtt{in}(\ell_i) \neq \varnothing$. As Algorithm 1 is sound for σ then there exists a labelling $\ell = \bigcup_{i=0}^n \ell_i$, with $\ell_i \in \mathscr{L}_\sigma(\mathit{af}_i)$, such that $\ell \in \sigma(\mathcal{AF})$ and $\mathtt{in}(\ell) \neq \varnothing$.

  We prove so that if $\mathit{Comp\text{-}Exists}_\sigma^{\neg\varnothing}(\mathcal{AF})$ is true, then $\mathit{Exists}_\sigma^{\neg\varnothing}(\mathcal{AF})$ is true.

We prove so that: $\mathit{Exists}_\sigma^{\neg\varnothing}(\mathcal{AF}) \equiv \mathit{Comp\text{-}Exists}_\sigma^{\neg\varnothing}(\mathcal{AF})$.

Assertion 6: $\mathit{Unique}_\sigma(\mathcal{AF}) \equiv \mathit{Comp\text{-}Unique}_\sigma(\mathcal{AF})$

- Case 1: If $\mathit{Unique}_\sigma(\mathcal{AF})$ is true, then $\mathit{Comp\text{-}Unique}_\sigma(\mathcal{AF})$ is true.

  If $\mathit{Unique}_\sigma(\mathcal{AF})$ is true, then $\sigma(\mathcal{AF}) = \{\ell\}$.

  As Algorithm 1 is complete for σ, there exists thus a combination of component labellings $\ell_0,...,\ell_n$, with $\ell_i \in \mathscr{L}_\sigma(\mathit{af}_i)$ and $\mathscr{L}_\sigma(\mathit{af}_i) \in \mathtt{Comp}_\sigma(\mathcal{AF})$ such that $\ell = \bigcup_{i=0}^n \ell_i$. As a consequence, $\forall \mathscr{L}_\sigma(\mathit{af}_i) \in \mathtt{Comp}_\sigma(\mathcal{AF})$, $\mathscr{L}_\sigma(\mathit{af}_i) \neq \varnothing$.

  As Algorithm 1 is sound for σ and as $|\sigma(\mathcal{AF})| = 1$, then $\forall \mathscr{L}_\sigma(\mathit{af}_i) \in \mathtt{Comp}_\sigma(\mathcal{AF})$, $|\mathscr{L}_\sigma(\mathit{af}_i)| = 1$.

  We prove so that if $\mathit{Unique}_\sigma(\mathcal{AF})$ is true, then $\mathit{Comp\text{-}Unique}_\sigma(\mathcal{AF})$ is true.

- Case 2: If $\mathit{Comp\text{-}Unique}_\sigma(\mathcal{AF})$ is true, then $\mathit{Unique}_\sigma(\mathcal{AF})$ is true.

  If $\mathit{Comp\text{-}Unique}_\sigma(\mathcal{AF})$ is true, then $\forall \mathscr{L}_\sigma(\mathit{af}_i) \in \mathtt{Comp}_\sigma(\mathcal{AF})$, $|\mathscr{L}_\sigma(\mathit{af}_i)| = 1$. For $i \in \{0,...,n\}$, let ℓ_i be the unique labelling of the component af_i.

Let $\ell = \bigcup_{i=0}^{n} \ell_i$ be a labelling. Given that Algorithm 1 is sound for σ, then $\ell \in \sigma(\mathcal{AF})$.

Given that Algorithm 1 is complete for σ, then $|\sigma(\mathcal{AF})| = 1$.

We prove so that if $\mathit{Comp}\text{-}\mathit{Unique}_\sigma(\mathcal{AF})$ is true, then $\mathit{Unique}_\sigma(\mathcal{AF})$ is true.

We prove so that: $\mathit{Unique}_\sigma(\mathcal{AF}) \equiv \mathit{Comp}\text{-}\mathit{Unique}_\sigma(\mathcal{AF})$ ■

Proofs of Part V: Contributions about RAF

Proofs of Chapter 10: New RAF semantics

Proof of Theorem 6 on page 103.

1. (*Stable* structures are *semi-stable* ones). Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and $\mathcal{U} = \langle S, Q \rangle$ be a *stable* structure. According to the definition of a *stable* structure (Definition 63 on page 92), we have:

   $$S = A \setminus \text{RAF-}Def(\mathcal{U}) \text{ and } Q = K \setminus \text{RAF-}Inh(\mathcal{U})$$

   For any $x \in (A \cup K)$, x is either in $\mathcal{U}$ or is defeated or inhibited by $\mathcal{U}$. As a consequence, $(S \cup Q \cup \text{RAF-}Def(\mathcal{U}) \cup \text{RAF-}Inh(\mathcal{U}))$ is maximal *w.r.t.* to inclusion.

   We prove so that every *stable* structure is a *semi-stable* structure.

2. (*Semi-stable* structures are *preferred* ones). Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and $\mathcal{U} = \langle S, Q \rangle$ be a *semi-stable* structure. Let suppose that $\mathcal{U}$ is not a *preferred* structure. $\mathcal{U}$ being by definition a *complete* structure (Definition 71 on page 103), there exists thus a *preferred* structure $\mathcal{U}' = \langle S', Q' \rangle$ such that $\mathcal{U} \sqsubset \mathcal{U}'$. We have thus by definition of $\sqsubseteq$-inclusion:

   $$S \subseteq S' \text{ and } Q \subseteq Q' \tag{18.47}$$

   From the strict inclusion, we also have:

   $$(S \cup Q) \subset (S' \cup Q') \tag{18.48}$$

   It follows, from Equations (18.47) and (18.48) and from Definition 61 on page 92 that:

   $$(\text{RAF-}Def(\mathcal{U}) \cup \text{RAF-}Inh(\mathcal{U})) \subset (\text{RAF-}Def(\mathcal{U}') \cup \text{RAF-}Inh(\mathcal{U}')) \tag{18.49}$$

   Combining Equations (18.48) and (18.49), we have:

   $$(S \cup Q \cup \text{RAF-}Def(\mathcal{U}) \cup \text{RAF-}Inh(\mathcal{U})) \subset (S' \cup Q' \cup \text{RAF-}Def(\mathcal{U}') \cup \text{RAF-}Inh(\mathcal{U}')) \tag{18.50}$$

   Given that $\mathcal{U}'$ is also a *complete* structure, the consequence of Equation (18.50) is that $\mathcal{U}$ is not a *semi-stable* structure, as $(S \cup Q \cup \text{RAF-}Def(\mathcal{U}) \cup \text{RAF-}Inh(\mathcal{U}))$ is not maximal. There is thus a contradiction.

We prove so that every *semi-stable* structure is also a *preferred* structure.

■

Proof of Theorem 7 on page 103. Let suppose that there exists a *stable* structure $\mathcal{U} = \langle S, Q \rangle$. Following the definition of *stable* structures (Definition 63 on page 92), we have: $S = A \setminus \text{RAF-}Def(\mathcal{U})$ and $Q = K \setminus \text{RAF-}Inh(\mathcal{U})$. As a consequence, we have $(S \cup \text{RAF-}Def(\mathcal{U}) \cup Q \cup \text{RAF-}Inh(\mathcal{U}))$ including all the arguments and attacks of $\mathcal{RAF}$.

According to Theorem 6 on page 103, $\mathcal{U}$ is also a *semi-stable* structure. As any *semi-stable* structure $\mathcal{U}' = \langle S', Q' \rangle$ maximizes the set $(S' \cup \text{RAF-}Def(\mathcal{U}') \cup Q' \cup \text{RAF-}Inh(\mathcal{U}'))$ and as there exists $\mathcal{U}$, a structure such that $(S \cup \text{RAF-}Def(\mathcal{U}) \cup Q \cup \text{RAF-}Inh(\mathcal{U}))$ is maximized to point that it includes all the arguments and attacks of $\mathcal{RAF}$, then for $\mathcal{U}'$ to be maximal we necessarily have $(S' \cup \text{RAF-}Def(\mathcal{U}') \cup Q' \cup \text{RAF-}Inh(\mathcal{U}'))$ also including all the arguments and attacks of $\mathcal{RAF}$. $\mathcal{U}'$ is then a *stable* structure.

We prove thus that if there exists a *stable* structure, then the *semi-stable* structures coincide with the *stable* structures. ■

Proof of Proposition 23 on page 103. Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF such that $\forall \alpha \in K$, $t(\alpha) \in A$. $\mathcal{RAF}$ can thus be considered as a simple AF. Let $\mathcal{AF} = \langle A, K \rangle$ be the AF version of $\mathcal{RAF}$.

Step 1: Let prove in a first place that if $\mathcal{U} = \langle S, K \rangle$ is a *semi-stable* structure of $\mathcal{RAF}$ then S is a *semi-stable* extension of $\mathcal{AF}$.

Let $\mathcal{U} = \langle S, K \rangle$ be a *semi-stable* structure over $\mathcal{RAF}$. Notice that the set of attacks of $\mathcal{U}$ is K as attacks are always valid in $\mathcal{RAF}$, and so that $\text{RAF-}Inh(\mathcal{U}) = \varnothing$. Let suppose that S is not a *semi-stable* extension of $\mathcal{AF}$. There exists thus an extension S' of $\mathcal{AF}$ such that:

$$(S \cup \text{RAF-}Def(S)) \subset (S' \cup \text{RAF-}Def(S')) \tag{18.51}$$

We have thus :

$$(S \cup \text{RAF-}Def(S) \cup K) \subset (S' \cup \text{RAF-}Def(S') \cup K) \tag{18.52}$$

Let $\mathcal{U}' = \langle S', K \rangle$ be the structure over $\mathcal{RAF}$ whose set of arguments is the extension S'. For the same reason as $\mathcal{U}$, the set of attacks of $\mathcal{U}'$ is K and $\text{RAF-}Inh(\mathcal{U}') = \varnothing$.

As $\text{RAF-}Inh(\mathcal{U}) = \varnothing$ and $\text{RAF-}Inh(\mathcal{U}') = \varnothing$, we can thus say from Equation (18.52) that:

$$(S \cup \text{RAF-}Def(S) \cup K \cup \text{RAF-}Inh(\mathcal{U})) \subset (S' \cup \text{RAF-}Def(S') \cup K \cup \text{RAF-}Inh(\mathcal{U}')) \tag{18.53}$$

Given that all attacks are valid in $\mathcal{RAF}$, we have: $\text{RAF-}Def(S) = \text{RAF-}Def(\mathcal{U})$ and $\text{RAF-}Def(S') = \text{RAF-}Def(\mathcal{U}')$. We have thus from Equation (18.53):

$$(S \cup \text{RAF-}Def(\mathcal{U}) \cup K \cup \text{RAF-}Inh(\mathcal{U})) \subset (S' \cup \text{RAF-}Def(\mathcal{U}') \cup K \cup \text{RAF-}Inh(\mathcal{U}')) \tag{18.54}$$

As stated by Equation (18.54), $(S \cup \text{RAF-}Def(S) \cup K \cup \text{RAF-}Inh(\mathcal{U}))$ is not maximal. It follows that $\mathcal{U}$ is not a *semi-stable* structure, which is a contradiction.

We prove so that if $\mathcal{U} = \langle S, K \rangle$ is a *semi-stable* structure of $\mathcal{RAF}$ then S is a *semi-stable* extension of $\mathcal{AF}$.

Step 2: Let now prove that if S is a *semi-stable* extension of $\mathcal{AF}$ then $\mathcal{U} = \langle S, K \rangle$ is a *semi-stable* structure of $\mathcal{RAF}$.

Let S be a *semi-stable* extension of $\mathcal{AF}$ and let $\mathcal{U} = \langle S, K \rangle$ be a structure over $\mathcal{RAF}$ whose set of arguments is S. Notice that the set of attacks of $\mathcal{U}$ is K as attacks are always valid in $\mathcal{RAF}$.

Let suppose that $\mathcal{U}$ is not a *semi-stable* structure. There exists thus a *semi-stable* structure $\mathcal{U}' = \langle S', K \rangle$ such that:

$$(S \cup \text{RAF-}Def(\mathcal{U}) \cup K \cup \text{RAF-}Inh(\mathcal{U})) \subset (S' \cup \text{RAF-}Def(\mathcal{U}') \cup K \cup \text{RAF-}Inh(\mathcal{U}')) \tag{18.55}$$

Given that all attacks are valid in $\mathcal{RAF}$, we have: RAF-$Inh(\mathcal{U}) = \varnothing$ and RAF-$Inh(\mathcal{U}') = \varnothing$. We have thus from Equation (18.55):

$$(S \cup \text{RAF-}Def(\mathcal{U})) \subset (S' \cup \text{RAF-}Def(\mathcal{U}')) \tag{18.56}$$

Furthermore, as all attacks are valid in $\mathcal{RAF}$, we have: RAF-$Def(S) =$ RAF-$Def(\mathcal{U})$ and RAF-$Def(S') =$ RAF-$Def(\mathcal{U}')$. We have thus from Equation (18.56):

$$(S \cup \text{RAF-}Def(S)) \subset (S' \cup \text{RAF-}Def(S')) \tag{18.57}$$

As stated by Equation (18.57), $(S \cup \text{RAF-}Def(S))$ is not maximal. It follows that S is not a *semi-stable* extension, which is a contradiction.

We prove so that if S is a *semi-stable* extension of $\mathcal{AF}$ then $\mathcal{U} = \langle S, K \rangle$ is a *semi-stable* structure of $\mathcal{RAF}$.

With steps 1 and 2, we have thus proven that:

$\mathcal{U} = \langle S, K \rangle$ is a *semi-stable* structure of $\mathcal{RAF}$ *iff* S is a *semi-stable* extension of $\mathcal{AF}$

■

Proofs of Chapter 11: Semantics and Labellings

Proofs of Section 11.1: Complete semantics

Proof of Theorem 8 on page 107. Let $\mathcal{U} = \texttt{Lab2Struct}(\mathcal{L})$. According to Definition 63 on page 92, $\mathcal{U}$ being a *complete* structure (with $\mathcal{U} = \langle S, Q \rangle$) means that $(S \cup Q) = Acc(\mathcal{U})$. In a first step, let us prove that $(S \cup Q) \subseteq Acc(\mathcal{U})$ and then that $(S \cup Q) \supseteq Acc(\mathcal{U})$.

Step 1: $(S \cup Q) \subseteq Acc(\mathcal{U})$

Let $x \in (S \cup Q)$. By definition of $\texttt{Lab2Struct}(\mathcal{L})$, we have $\mathcal{L}(x) = \textit{in}$. Given that $\mathcal{L}$ is a reinstatement RAF labelling, we have:

$$\forall \alpha \in K \text{ s.t. } t(\alpha) = x, \ell_K(\alpha) = \textit{out} \text{ or } \ell_A(s(\alpha)) = \textit{out} \tag{18.58}$$

So two cases must be considered: $\ell_K(\alpha) = \textit{out}$ or $\ell_A(s(\alpha)) = \textit{out}$.

1. $\ell_K(\alpha) = \textit{out}$.

   Given $\mathcal{L}$ is a reinstatement RAF labelling there exists an attack β such that $t(\beta) = \alpha$, $\ell_K(\beta) = \textit{in}$ and $\ell_A(s(\beta)) = \textit{in}$. As a consequence, $\beta \in Q$ and $s(\beta) \in S$. According to Definition 61 on page 92, we have so: $\alpha \in$ RAF-$Inh(\mathcal{U})$.

2. $\ell_A(s(\alpha)) = \textit{out}$.

   Given $\mathcal{L}$ is a reinstatement RAF labelling there exists an attack γ such that $t(\gamma) = s(\alpha)$, $\ell_K(\gamma) = \textit{in}$ and $\ell_A(s(\gamma)) = \textit{in}$. As a consequence, $\gamma \in Q$ and $s(\gamma) \in S$. According to Definition 61 on page 92, we have so: $s(\alpha) \in$ RAF-$Def(\mathcal{U})$.

As a consequence and following Definition 62 on page 92, we have: $x \in Acc(\mathcal{U})$.

We prove so that:

$$(S \cup Q) \subseteq Acc(\mathcal{U}) \tag{18.59}$$

Step 2: $(S \cup Q) \supseteq Acc(\mathcal{U})$

Let $x \in Acc(\mathcal{U})$, x being an argument or an attack. According to Definition 62 on page 92, for all $\alpha \in K$ such that $t(\alpha) = x$, we have: $s(\alpha) \in$ RAF-*Def*$(\mathcal{U})$ or $\alpha \in$ RAF-*Inh*$(\mathcal{U})$.

Let y be $s(\alpha)$ or α. Given that $y \in$ (RAF-*Def*$(\mathcal{U}) \cup$ RAF-*Inh*$(\mathcal{U})$), there exists an attack β such that $s(\beta) \in S$, $\beta \in Q$ and $t(\beta) = y$. By definition of `Lab2Struct`$(\mathcal{L})$, we have $\ell_A(s(\beta)) = \mathit{in}$ and $\ell_K(\beta) = \mathit{in}$. Given $\mathcal{L}$ is a reinstatement RAF labelling, we have $\mathcal{L}(y) = \mathit{out}$.

As a consequence, we have:

$$\forall \alpha \in K \text{ s.t. } t(\alpha) = x,\ \ell_A(s(\alpha)) = \mathit{out} \text{ or } \ell_K(\alpha) = \mathit{out} \tag{18.60}$$

Then, given that $\mathcal{L}$ is a reinstatement RAF labelling, we have : $\mathcal{L}(x) = \mathit{in}$. By definition of `Lab2Struct`$(\mathcal{L})$ we have so $x \in (S \cup Q)$.

We prove so that :

$$(S \cup Q) \supseteq Acc(\mathcal{U}) \tag{18.61}$$

Finally, because of Equations (18.59) and (18.61) we have:

$$(S \cup Q) = Acc(\mathcal{U}) \tag{18.62}$$

We prove thus that `Lab2Struct`$(\mathcal{L})$ is a *complete* structure. ■

Proof of Theorem 9 on page 108. Let $\mathcal{L} =$ `Struct2Lab`$(\mathcal{U})$. In order to prove that $\mathcal{L}$ is a reinstatement RAF labelling (with $\mathcal{L} = \langle \ell_A, \ell_K \rangle$) we have to prove that, for all $x \in (A \cup K)$:

1. $(\mathcal{L}(x) = \mathit{out}) \Longrightarrow (\exists \alpha \in K \text{ s.t. } t(\alpha) = x,\ \ell_K(\alpha) = \mathit{in} \text{ and } \ell_A(s(\alpha)) = \mathit{in})$
2. $(\mathcal{L}(x) = \mathit{out}) \Longleftarrow (\exists \alpha \in K \text{ s.t. } t(\alpha) = x,\ \ell_K(\alpha) = \mathit{in} \text{ and } \ell_A(s(\alpha)) = \mathit{in})$
3. $(\mathcal{L}(x) = \mathit{in}) \Longrightarrow (\forall \alpha \in K \text{ s.t. } t(\alpha) = x,\ \ell_K(\alpha) = \mathit{out} \text{ or } \ell_A(s(\alpha)) = \mathit{out})$
4. $(\mathcal{L}(x) = \mathit{in}) \Longleftarrow (\forall \alpha \in K \text{ s.t. } t(\alpha) = x,\ \ell_K(\alpha) = \mathit{out} \text{ or } \ell_A(s(\alpha)) = \mathit{out})$

Step 1: $(\mathcal{L}(x) = \mathit{out}) \Longrightarrow (\exists \alpha \in K \text{ s.t. } t(\alpha) = x,\ \ell_K(\alpha) = \mathit{in} \text{ and } \ell_A(s(\alpha)) = \mathit{in})$

Let $x \in (A \cup K)$ be an argument or an attack such that $\mathcal{L}(x) = \mathit{out}$. According to the definition of `Struct2Lab`$(\mathcal{U})$, we have $x \in$ (RAF-*Def*$(\mathcal{U}) \cup$ RAF-*Inh*$(\mathcal{U})$). Following the definitions of RAF-*Def*$(\mathcal{U})$ and RAF-*Inh*$(\mathcal{U})$, we can state that there exists an attack α such that $\alpha \in Q$, $s(\alpha) \in S$ and $t(\alpha) = x$. According to the definition of `Struct2Lab`$(\mathcal{U})$, we have so $\ell_K(\alpha) = \mathit{in}$ and $\ell_A(s(\alpha)) = \mathit{in}$.

We prove so that for all $x \in (A \cup K)$:

$$(\mathcal{L}(x) = \mathit{out}) \Longrightarrow (\exists \alpha \in K \text{ s.t. } t(\alpha) = x,\ \ell_K(\alpha) = \mathit{in} \text{ and } \ell_A(s(\alpha)) = \mathit{in}) \tag{18.63}$$

Step 2: $(\mathcal{L}(x) = \mathit{out}) \Longleftarrow (\exists \alpha \in K \text{ s.t. } t(\alpha) = x,\ \ell_K(\alpha) = \mathit{in} \text{ and } \ell_A(s(\alpha)) = \mathit{in})$

Let $x \in (A \cup K)$ be an argument or an attack. If there exists an attack $\alpha \in K$ such that $t(\alpha) = x$, $\ell_K(\alpha) = \mathit{in}$ and $\ell_A(s(\alpha)) = \mathit{in}$, then according to the definition of `Struct2Lab`$(\mathcal{U})$, we have $\alpha \in Q$ and $s(\alpha) \in S$.

As a consequence, we have $x \in (\text{RAF-}Def(\mathcal{U}) \cup \text{RAF-}Inh(\mathcal{U}))$. We have thus, according to the definition of $\texttt{Struct2Lab}(\mathcal{U})$: $\mathcal{L}(x) = \mathit{out}$.

We prove so that for all $x \in (A \cup K)$:

$$(\mathcal{L}(x) = \mathit{out}) \Longleftarrow (\exists \alpha \in K \text{ s.t. } t(\alpha) = x, \ell_K(\alpha) = \mathit{in} \text{ and } \ell_A(s(\alpha)) = \mathit{in}) \tag{18.64}$$

Step 3: $(\mathcal{L}(x) = \mathit{in}) \Longrightarrow (\forall \alpha \in K \text{ s.t. } t(\alpha) = x, \ell_K(\alpha) = \mathit{out} \text{ or } \ell_A(s(\alpha)) = \mathit{out})$

Let $x \in (A \cup K)$ be an argument or an attack such that $\mathcal{L}(x) = \mathit{in}$. According to the definition of $\texttt{Struct2Lab}(\mathcal{U})$, we have then $x \in \mathcal{U}$ and as $\mathcal{U}$ is a complete structure we have $x \in Acc(\mathcal{U})$. As a consequence, for all $\alpha \in K$ such that $t(\alpha) = x$, we have: $s(\alpha) \in \text{RAF-}Def(\mathcal{U})$ or $\alpha \in \text{RAF-}Inh(\mathcal{U})$. According to the definition of $\texttt{Struct2Lab}(\mathcal{U})$, we have then: $\ell_A(s(\alpha)) = \mathit{out}$ or $\ell_K(\alpha) = \mathit{out}$.

We prove so that for all $x \in (A \cup K)$:

$$(\mathcal{L}(x) = \mathit{in}) \Longrightarrow (\forall \alpha \in K \text{ s.t. } t(\alpha) = x, \ell_K(\alpha) = \mathit{out} \text{ or } \ell_A(s(\alpha)) = \mathit{out}) \tag{18.65}$$

Step 4: $(\mathcal{L}(x) = \mathit{in}) \Longleftarrow (\forall \alpha \in K \text{ s.t. } t(\alpha) = x, \ell_K(\alpha) = \mathit{out} \text{ or } \ell_A(s(\alpha)) = \mathit{out})$

Let $x \in (A \cup K)$ be an argument or an attack such that for all attacks $\alpha \in K$ s.t. $t(\alpha) = x$, $\ell_K(\alpha) = \mathit{out}$ or $\ell_A(s(\alpha)) = \mathit{out}$. For any such attack α, we have then, according to the definition of $\texttt{Struct2Lab}(\mathcal{U})$: $\alpha \in \text{RAF-}Inh(\mathcal{U})$ or $s(\alpha) \in \text{RAF-}Def(\mathcal{U})$. As a consequence, we have $x \in Acc(\mathcal{U})$ and so $x \in (S \cup Q)$, $\mathcal{U}$ being a *complete* structure. According to the definition of $\texttt{Struct2Lab}(\mathcal{U})$, we have then: $\mathcal{L}(x) = \mathit{in}$.

We prove so that for all $x \in (A \cup K)$:

$$(\mathcal{L}(x) = \mathit{in}) \Longleftarrow (\forall \alpha \in K \text{ s.t. } t(\alpha) = x, \ell_K(\alpha) = \mathit{out} \text{ or } \ell_A(s(\alpha)) = \mathit{out}) \tag{18.66}$$

Equations (18.63) to (18.66) on pages 233–234 being stated, we prove thus that $\mathcal{L}$ is a reinstatement RAF labelling. ■

Proofs of Section 11.2: Preferred semantics

Proof of Theorem 10 on page 108. Let $\mathcal{L}$ be a reinstatement RAF labelling such that $\mathit{in}(\mathcal{L})$ is maximal. Let suppose that $\mathcal{U} = \texttt{Lab2Struct}(\mathcal{L})$ is not a *preferred* structure. According to Definition 63 on page 92, Proposition 19 on page 94 and Theorem 2 on page 94, there exists then a *complete* structure $\mathcal{U}'$ such that $\mathcal{U} \sqsubset \mathcal{U}'$ (strict inclusion). Let $\mathcal{L}' = \texttt{Struct2Lab}(\mathcal{U}')$. Then $\mathit{in}(\mathcal{L}') \subset \mathit{in}(\mathcal{L})$. As a consequence $\mathcal{L}$ is not a reinstatement RAF labelling such that $\mathit{in}(\mathcal{L})$ is maximal, which is a contradiction. ■

Proof of Theorem 11 on page 108. Let $\mathcal{U}$ be a *preferred* structure and $\mathcal{L} = \texttt{Struct2Lab}(\mathcal{U})$. Let us suppose that $\mathcal{L}$ is not a reinstatement RAF labelling such that $\mathit{in}(\mathcal{L})$ is maximal. Then there exists a reinstatement RAF labelling $\mathcal{L}'$ such that $\mathit{in}(\mathcal{L}) \subset \mathit{in}(\mathcal{L}')$. Let $\mathcal{U}' = \texttt{Lab2Struct}(\mathcal{L}')$. Then $\mathcal{U}'$ is a *complete* structure such that $\mathcal{U} \sqsubset \mathcal{U}'$ (strict inclusion). As a consequence, $\mathcal{U}$ is not a *preferred* structure, which is a contradiction. ■

Lemma 1. *Let $\mathcal{L}$ and $\mathcal{L}'$ be two reinstatement RAF labellings. If $\mathit{in}(\mathcal{L}) \subset \mathit{in}(\mathcal{L}')$ then $\mathit{out}(\mathcal{L}) \subset \mathit{out}(\mathcal{L}')$.*

Proof of Lemma 1. Let $\mathcal{L}$ and $\mathcal{L}'$ be two reinstatement RAF labellings such that $\mathit{in}(\mathcal{L}) \subset \mathit{in}(\mathcal{L}')$, meaning that:

$$\forall w \in \mathit{in}(\mathcal{L}), w \in \mathit{in}(\mathcal{L}') \tag{18.67}$$

and

$$\exists x \in \mathtt{in}(\mathcal{L}'), x \notin \mathtt{in}(\mathcal{L}) \tag{18.68}$$

Let prove that $\mathtt{out}(\mathcal{L}) \subset \mathtt{out}(\mathcal{L}')$, and so that :

1. $\forall y \in \mathtt{out}(\mathcal{L}), y \in \mathtt{out}(\mathcal{L}')$
2. $\exists z \in \mathtt{out}(\mathcal{L}'), z \notin \mathtt{out}(\mathcal{L})$

Step 1: $\forall y \in \mathtt{out}(\mathcal{L}), y \in \mathtt{out}(\mathcal{L}')$

Let y be an attack or an argument such that $y \in \mathtt{out}(\mathcal{L})$. Given $\mathcal{L}$ is a reinstatement RAF labelling, we have by definition:

$$(\mathcal{L}(y) = \mathtt{out}) \Longrightarrow (\exists \alpha \in K \text{ s.t. } t(\alpha) = y, \ell_K(\alpha) = \mathtt{in} \text{ and } \ell_A(s(\alpha)) = \mathtt{in})$$

Then according to Equation (18.67) on the previous page, $\alpha \in \mathtt{in}(\mathcal{L}')$ and $s(\alpha) \in \mathtt{in}(\mathcal{L}')$. As $\mathcal{L}'$ is also a reinstatement RAF labelling, we have so $y \in \mathtt{out}(\mathcal{L}')$.

Step 2: $\exists z \in \mathtt{out}(\mathcal{L}'), z \notin \mathtt{out}(\mathcal{L})$

Let x be an attack or an argument such that $x \in \mathtt{in}(\mathcal{L}')$ and $x \notin \mathtt{in}(\mathcal{L})$. Given $\mathcal{L}$ and $\mathcal{L}'$ are reinstatement RAF labellings, we have by definition:

$$(\mathcal{L}'(x) = \mathtt{in}) \iff (\forall \alpha \in K \text{ s.t. } t(\alpha) = x, {\ell_K}'(\alpha) = \mathtt{out} \text{ or } {\ell_A}'(s(\alpha)) = \mathtt{out}) \tag{18.69}$$

$$(\mathcal{L}(x) \neq \mathtt{in}) \iff (\exists \alpha \in K \text{ s.t. } t(\alpha) = x, \ell_K(\alpha) \neq \mathtt{out} \text{ and } \ell_A(s(\alpha)) \neq \mathtt{out}) \tag{18.70}$$

Let α be such an attack with $t(\alpha) = x$, $\ell_K(\alpha) \neq \mathtt{out}$ and $\ell_A(s(\alpha)) \neq \mathtt{out}$.

By definition of α we have, $\alpha \notin \mathtt{out}(\mathcal{L})$ and $s(\alpha) \notin \mathtt{out}(\mathcal{L})$. Furthermore, given that $\mathcal{L}'(x) = \mathtt{in}$, we have following Equation (18.69), $\alpha \in \mathtt{out}(\mathcal{L}')$ or $s(\alpha) \in \mathtt{out}(\mathcal{L}')$.

We prove thus that there exists z such that, $z \in \mathtt{out}(\mathcal{L}')$ and $z \notin \mathtt{out}(\mathcal{L})$. ■

Lemma 2. *Let $\mathcal{L}$ and $\mathcal{L}'$ be two reinstatement RAF labellings. If $\mathtt{out}(\mathcal{L}) \subset \mathtt{out}(\mathcal{L}')$ then $\mathtt{in}(\mathcal{L}) \subset \mathtt{in}(\mathcal{L}')$.*

Proof of Lemma 2. Let $\mathcal{L}$ and $\mathcal{L}'$ be two reinstatement RAF labellings such that $\mathtt{out}(\mathcal{L}) \subset \mathtt{out}(\mathcal{L}')$, meaning that:

$$\forall w \in \mathtt{out}(\mathcal{L}), w \in \mathtt{out}(\mathcal{L}') \tag{18.71}$$

and

$$\exists x \in \mathtt{out}(\mathcal{L}'), x \notin \mathtt{out}(\mathcal{L}) \tag{18.72}$$

Let prove that $\mathtt{in}(\mathcal{L}) \subset \mathtt{in}(\mathcal{L}')$, and so that :

1. $\forall y \in \mathtt{in}(\mathcal{L}), y \in \mathtt{in}(\mathcal{L}')$
2. $\exists z \in \mathtt{in}(\mathcal{L}'), z \notin \mathtt{in}(\mathcal{L})$

Step 1: $\forall y \in \mathtt{in}(\mathcal{L}), y \in \mathtt{in}(\mathcal{L}')$

Let y be an attack or an argument such that $y \in \mathtt{in}(\mathcal{L})$. Given $\mathcal{L}$ is a reinstatement RAF labelling, we have by definition:

$$(\mathcal{L}(y) = \mathtt{in}) \Longrightarrow (\forall \alpha \in K \text{ s.t. } t(\alpha) = y, \ell_K(\alpha) = \mathtt{out} \text{ or } \ell_A(s(\alpha)) = \mathtt{out})$$

Then according to Equation (18.71) on the previous page, we have:

$$(\mathcal{L}(y) = \mathtt{in}) \Longrightarrow (\forall \alpha \in K \text{ s.t. } t(\alpha) = y, {\ell_K}'(\alpha) = \mathtt{out} \text{ or } {\ell_A}'(s(\alpha)) = \mathtt{out})$$

As $\mathcal{L}'$ is also a reinstatement RAF labelling, we have then $y \in \mathtt{in}(\mathcal{L}')$.

Step 2: $\exists z \in \mathtt{in}(\mathcal{L}'), z \notin \mathtt{in}(\mathcal{L})$

Let x be an attack or an argument such that $x \in \mathtt{out}(\mathcal{L}')$ and $x \notin \mathtt{out}(\mathcal{L})$. Given $\mathcal{L}$ and $\mathcal{L}'$ are reinstatement RAF labellings, we have by definition:

$$(\mathcal{L}'(x) = \mathtt{out}) \iff (\exists \alpha \in K \text{ s.t. } t(\alpha) = x, {\ell_K}'(\alpha) = \mathtt{in} \text{ and } {\ell_A}'(s(\alpha)) = \mathtt{in}) \tag{18.73}$$

$$(\mathcal{L}(x) \neq \mathtt{out}) \iff (\forall \alpha \in K \text{ s.t. } t(\alpha) = x, \ell_K(\alpha) \neq \mathtt{in} \text{ or } \ell_A(s(\alpha)) \neq \mathtt{in}) \tag{18.74}$$

According to Equation (18.74), for any attack α such that $t(\alpha) = x$, we have: $\alpha \notin \mathtt{in}(\mathcal{L})$ or $s(\alpha) \notin \mathtt{in}(\mathcal{L})$. However, we have following Equation (18.73), there exists at least one attack $\alpha \in K$ s.t. $\alpha \in \mathtt{in}(\mathcal{L}')$ and $s(\alpha) \in \mathtt{in}(\mathcal{L}')$.

We prove thus that there exists z such that, $z \in \mathtt{in}(\mathcal{L}')$ and $z \notin \mathtt{in}(\mathcal{L})$. ■

Proof of Theorem 12 on page 108. Let $\mathcal{L}$ be a reinstatement RAF labelling such that $\mathtt{out}(\mathcal{L})$ is maximal. Let suppose that $\mathtt{Lab2Struct}(\mathcal{L})$ is not a *preferred* structure. Then according to Theorem 10 on page 108, $\mathtt{in}(\mathcal{L})$ is not maximal. There exists thus a reinstatement RAF labelling $\mathcal{L}'$ such that $\mathtt{in}(\mathcal{L}) \subset \mathtt{in}(\mathcal{L}')$. We have then, following Lemma 1 on page 234, $\mathtt{out}(\mathcal{L}) \subset \mathtt{out}(\mathcal{L}')$, which is a contradiction. ■

Proof of Theorem 13 on page 108. Let $\mathcal{U}$ be a *preferred* structure. According to Theorem 11 on page 108, $\mathcal{L} = \mathtt{Struct2Lab}(\mathcal{U})$ is a reinstatement RAF labelling such that $\mathtt{in}(\mathcal{L})$ is maximal. Let suppose that $\mathtt{out}(\mathcal{L})$ is not maximal. There exist thus a reinstatement RAF labelling $\mathcal{L}'$ such that $\mathtt{out}(\mathcal{L}) \subset \mathtt{out}(\mathcal{L}')$. We have then, following Lemma 2 on the previous page, $\mathtt{in}(\mathcal{L}) \subset \mathtt{in}(\mathcal{L}')$, which is a contradiction. ■

Proofs of Section 11.3: Stable semantics

Proof of Theorem 14 on page 109. Let $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ be a reinstatement RAF labelling such that $\mathtt{und}(\mathcal{L}) = \varnothing$. Let $\mathcal{U} = \mathtt{Lab2Struct}(\mathcal{L})$. Let x be any attack or argument such that $x \notin \mathcal{U}$.

Given that $\mathtt{und}(\mathcal{L}) = \varnothing$, we have according to Definition 75 on page 107: $\mathcal{L}(x) = \mathtt{out}$. There exists then an attack α such that: $\ell_A(s(\alpha)) = \mathtt{in} \cap \ell_K(\alpha) = \mathtt{in}$. We have then $s(\alpha) \in \mathcal{U}$ and $\alpha \in \mathcal{U}$.

Therefore, according to Definition 61 on page 92, we have: $x \in$ RAF-$\mathit{Def}(\mathcal{U})$ or $x \in$ RAF-$\mathit{Inh}(\mathcal{U})$, following the nature of x. This means that $\mathcal{U}$ defeats or inhibits any argument and attack which is not in it.

We prove so that $\mathcal{U}$ is, thus, a *stable* structure. ■

Proof of Theorem 15 on page 109. Let $\mathcal{U}$ be a *stable* structure and let x be an argument or an attack.

If $x \in \mathcal{U}$ then $\mathcal{L}(x) = \mathtt{in}$.

Let consider the case when $x \notin \mathcal{U}$. Given $\mathcal{U}$ is a *stable* structure then there exists an attack α in $\mathcal{U}$ that defeats or inhibits x. We have then, according to Definition 75 on page 107: $\mathcal{L}(x) = \mathtt{out}$.

In both cases $\mathcal{L}(x) \neq \mathtt{und}$. We prove so that $\mathtt{und}(\mathcal{L}) = \varnothing$. ■

Proofs of Section 11.4: Grounded semantics

Proof of Theorem 16 on page 109. Let $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ be a reinstatement RAF labelling such that $\mathtt{und}(\mathcal{L})$ is maximal. Let suppose that $\mathcal{U} = \mathtt{Lab2Struct}(\mathcal{L})$ is not the *grounded* structure. According to Theorem 8 on page 107, $\mathcal{U}$ is a *complete* structure. By definition of the *grounded* structure (Definition 63 on page 92), we can thus say that there exists a structure $\mathcal{U}'$ that is the grounded structure and such that $\mathcal{U}' \sqsubset \mathcal{U}$ (strict inclusion). Let $\mathcal{L}' = \mathtt{Struct2Lab}(\mathcal{U}')$ be the reinstatement RAF labelling corresponding with the *grounded* structure.

As $\mathcal{U}' \sqsubset \mathcal{U}$ we have, by definition of Struct2Lab: $\mathtt{in}(\mathcal{L}') \subset \mathtt{in}(\mathcal{L})$

Following Lemma 1 on page 234, we thus also have: $\mathtt{out}(\mathcal{L}') \subset \mathtt{out}(\mathcal{L})$

As a consequence, we can say that: $\mathtt{und}(\mathcal{L}) \subset \mathtt{und}(\mathcal{L}')$

There is a contradiction.

We prove so that $\mathcal{U}$ is, thus, the *grounded* structure. ■

Proof of Theorem 17 on page 109. Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF, let $\mathcal{U} = \langle S, Q \rangle$ be the *grounded* structure and $\mathcal{U}'$ be any *complete* structure that is not *grounded*. Let $\mathcal{L} = \mathtt{Struct2Lab}(\mathcal{U})$ and $\mathcal{L}' = \mathtt{Struct2Lab}(\mathcal{U}')$.

According to Definition 63 on page 92, we have: $\mathcal{U} \sqsubset \mathcal{U}'$.

By definition of Struct2Lab, we thus have: $\mathtt{in}(\mathcal{L}) \subset \mathtt{in}(\mathcal{L}')$.

Following Lemma 1 on page 234, we thus also have: $\mathtt{out}(\mathcal{L}) \subset \mathtt{out}(\mathcal{L}')$.

As a consequence, we have: $\mathtt{und}(\mathcal{L}') \subset \mathtt{und}(\mathcal{L})$.

We prove so that $\mathcal{L} = \mathtt{Struct2Lab}(\mathcal{U})$ is a reinstatement RAF labelling such that $\mathtt{und}(\mathcal{L})$ is maximal. ■

Proof of Theorem 18 on page 109. $\mathcal{L}$ be a reinstatement RAF labelling such that $\mathtt{in}(\mathcal{L})$ is minimal. Let suppose that $\mathcal{U} = \mathtt{Lab2Struct}(\mathcal{L})$ is not the *grounded* structure. By definition of the *grounded* structure (Definition 63 on page 92), we can thus say that there exists a structure $\mathcal{U}'$ that is the grounded structure and such that $\mathcal{U}' \sqsubset \mathcal{U}$ (strict inclusion). Let $\mathcal{L}' = \mathtt{Struct2Lab}(\mathcal{U}')$ be the reinstatement RAF labelling corresponding with the *grounded* structure. As $\mathcal{U}' \sqsubset \mathcal{U}$ we have, by definition of Struct2Lab: $\mathtt{in}(\mathcal{L}') \subset \mathtt{in}(\mathcal{L})$. We have then a contradiction.

We prove so that $\mathcal{U} = \mathtt{Lab2Struct}(\mathcal{L})$ is the *grounded* structure. ■

Proof of Theorem 19 on page 109. Let $\mathcal{U}$ be the *grounded* structure and $\mathcal{L} = \mathtt{Struct2Lab}(\mathcal{U})$. Let suppose that $\mathtt{in}(\mathcal{L})$ is not minimal. There exists then a reinstatement RAF labelling $\mathcal{L}'$ such that $\mathtt{in}(\mathcal{L}') \subset \mathtt{in}(\mathcal{L})$. Let $\mathcal{U}' = \mathtt{Lab2Struct}(\mathcal{L}')$. From the definition of Lab2Struct, we can say that: $\mathcal{U}' \sqsubset \mathcal{U}$. This contradicts the definition of the grounded structure (Definition 63 on page 92).

We prove so that $\mathcal{L} = \mathtt{Struct2Lab}(\mathcal{U})$ is a reinstatement RAF labelling such that $\mathtt{in}(\mathcal{L})$ is minimal. ■

Proof of Theorem 20 on page 110. $\mathcal{L}$ be a reinstatement RAF labelling such that $\mathtt{out}(\mathcal{L})$ is minimal. Following Lemma 2 on page 235, $\mathtt{in}(\mathcal{L})$ is also minimal. Therefore, according to Theorem 18 on page 109, $\mathtt{Lab2Struct}(\mathcal{L})$ is the *grounded* structure. ■

Proof of Theorem 21 on page 110. Let $\mathcal{U}$ be the *grounded* structure and $\mathcal{L} = \mathtt{Struct2Lab}(\mathcal{U})$. According to Theorem 19 on page 109, $\mathtt{in}(\mathcal{L})$ is minimal. Following Lemma 1 on page 234, $\mathtt{out}(\mathcal{L})$ is also minimal. $\mathcal{L}$ is thus a reinstatement RAF labelling such that $\mathtt{out}(\mathcal{L})$ is minimal. ■

Proofs of Section 11.5: Semi-stable semantics

Proof of Theorem 22 on page 110. Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF. Let $\mathcal{L} = \langle \ell_A, \ell_K \rangle$ be a reinstatement RAF labelling such that $\mathtt{und}(\mathcal{L})$ is minimal. Let suppose that $\mathcal{U} = \mathtt{Lab2Struct}(\mathcal{L})$ is not a *semi-stable* structure (with $\mathcal{U} = \langle S, Q\rangle$). There exists thus a *semi-stable* structure $\mathcal{U}' = \langle S', Q'\rangle$ such that:

$$(S \cup Q \cup \text{RAF-}\mathit{Def}(\mathcal{U}) \cup \text{RAF-}\mathit{Inh}(\mathcal{U})) \subset (S' \cup Q' \cup \text{RAF-}\mathit{Def}(\mathcal{U}') \cup \text{RAF-}\mathit{Inh}(\mathcal{U}')) \tag{18.75}$$

As a consequence, we have:

$$(A \cup K) \setminus (S \cup Q \cup \text{RAF-}\mathit{Def}(\mathcal{U}) \cup \text{RAF-}\mathit{Inh}(\mathcal{U})) \supset (A \cup K) \setminus (S' \cup Q' \cup \text{RAF-}\mathit{Def}(\mathcal{U}') \cup \text{RAF-}\mathit{Inh}(\mathcal{U}')) \tag{18.76}$$

Let $\mathcal{L}' = \mathtt{Struct2Lab}(\mathcal{U}')$. Following Equation (18.76), we have according to the definition of Struct2Lab:

$$\mathtt{und}(\mathcal{L}) \supset \mathtt{und}(\mathcal{L}')$$

Then $\mathtt{und}(\mathcal{L})$ is not minimal, which is a contradiction.
We prove so that $\mathcal{U} = \mathtt{Lab2Struct}(\mathcal{L})$ is a *semi-stable* structure. ■

Proof of Theorem 23 on page 110. Let $\mathcal{U} = \langle S, Q\rangle$ be a *semi-stable* structure. By definition, we have thus:

$$(S \cup Q \cup \text{RAF-}\mathit{Def}(\mathcal{U}) \cup \text{RAF-}\mathit{Inh}(\mathcal{U})) \text{ being maximal}$$

As a consequence, $(A \cup K) \setminus (S \cup Q \cup \text{RAF-}\mathit{Def}(\mathcal{U}) \cup \text{RAF-}\mathit{Inh}(\mathcal{U}))$ is minimal. Let $\mathcal{L} = \mathtt{Struct2Lab}(\mathcal{U})$. According to the definition of Struct2Lab and following the previous statement, we have thus $\mathtt{und}(\mathcal{L})$ being minimal.

We prove so that $\mathcal{L} = \mathtt{Struct2Lab}(\mathcal{U})$ is a reinstatement RAF labelling such that $\mathtt{und}(\mathcal{L})$ is minimal. ■

Proofs of Chapter 12: Flattening

Proof of Proposition 24 on page 117. **Assertion 1:** RAF-$\mathit{Def}(\mathcal{U}) \cup$ RAF-$\mathit{Inh}(\mathcal{U}) =$

$$\mathit{Def}(\varepsilon_{\mathcal{U}}) \setminus \left(\begin{array}{l} \{\neg a \in \mathit{Not}_A | a \in \varepsilon_{\mathcal{U}}\} \\ \cup \{\neg\beta \in \mathit{Not}_K | \beta \in \varepsilon_{\mathcal{U}}\} \\ \cup \{s(\beta).\beta \in \mathit{And}_{A,K} | \beta \in \mathit{Def}(\varepsilon_{\mathcal{U}}) \text{ or } s(\beta) \in \mathit{Def}(\varepsilon_{\mathcal{U}})\} \end{array} \right)$$

- **Step 1:** RAF-$\mathit{Def}(\mathcal{U}) \cup$ RAF-$\mathit{Inh}(\mathcal{U}) \subseteq$

$$\mathit{Def}(\varepsilon_{\mathcal{U}}) \setminus \left(\begin{array}{l} \{\neg a \in \mathit{Not}_A | a \in \varepsilon_{\mathcal{U}}\} \\ \cup \{\neg\beta \in \mathit{Not}_K | \beta \in \varepsilon_{\mathcal{U}}\} \\ \cup \{s(\beta).\beta \in \mathit{And}_{A,K} | \beta \in \mathit{Def}(\varepsilon_{\mathcal{U}}) \text{ or } s(\beta) \in \mathit{Def}(\varepsilon_{\mathcal{U}})\} \end{array} \right)$$

  Let $x \in$ RAF-$\mathit{Def}(\mathcal{U}) \cup$ RAF-$\mathit{Inh}(\mathcal{U})$. There exists thus an attack $\alpha \in Q$ such that $s(\alpha) \in S$ and $t(\alpha) = x$. We have thus:

$$\alpha \in \varepsilon_{\mathcal{U}} \text{ and } s(\alpha) \in \varepsilon_{\mathcal{U}}$$

As a consequence, following the definition of `Raf2Af`, we have:

$$\neg\alpha \in Def(\varepsilon_{\mathfrak{U}}) \text{ and } \neg s(\alpha) \in Def(\varepsilon_{\mathfrak{U}})$$

And so:

$$s(\alpha).\alpha \in \varepsilon_{\mathfrak{U}}$$

Given that $s(\alpha).\alpha \in \varepsilon_{\mathfrak{U}}$, we have so:

$$x \in Def(\varepsilon_{\mathfrak{U}})$$

Given that $x \in A \cup K$, we have, following the definition of `Raf2Af`: $x \notin (Not_A \cup Not_K \cup And_{A,K})$. As a consequence, we have:

$$x \in Def(\varepsilon_{\mathfrak{U}}) \setminus (Not_A \cup Not_K \cup And_{A,K})$$

We prove so that: RAF-$Def(\mathfrak{U}) \cup$ RAF-$Inh(\mathfrak{U}) \subseteq$

$$Def(\varepsilon_{\mathfrak{U}}) \setminus \left(\begin{array}{l} \{\neg a \in Not_A | a \in \varepsilon_{\mathfrak{U}}\} \\ \\ \cup\{\neg\beta \in Not_K | \beta \in \varepsilon_{\mathfrak{U}}\} \\ \\ \cup\{s(\beta).\beta \in And_{A,K} | \beta \in Def(\varepsilon_{\mathfrak{U}}) \text{ or } s(\beta) \in Def(\varepsilon_{\mathfrak{U}})\} \end{array} \right)$$

- **Step 2:** RAF-$Def(\mathfrak{U}) \cup$ RAF-$Inh(\mathfrak{U}) \supseteq$

$$Def(\varepsilon_{\mathfrak{U}}) \setminus \left(\begin{array}{l} \{\neg a \in Not_A | a \in \varepsilon_{\mathfrak{U}}\} \\ \\ \cup\{\neg\beta \in Not_K | \beta \in \varepsilon_{\mathfrak{U}}\} \\ \\ \cup\{s(\beta).\beta \in And_{A,K} | \beta \in Def(\varepsilon_{\mathfrak{U}}) \text{ or } s(\beta) \in Def(\varepsilon_{\mathfrak{U}})\} \end{array} \right)$$

Let $x \in Def(\varepsilon_{\mathfrak{U}}) \setminus \left(\begin{array}{l} \{\neg a \in Not_A | a \in \varepsilon_{\mathfrak{U}}\} \\ \\ \cup\{\neg\beta \in Not_K | \beta \in \varepsilon_{\mathfrak{U}}\} \\ \\ \cup\{s(\beta).\beta \in And_{A,K} | \beta \in Def(\varepsilon_{\mathfrak{U}}) \text{ or } s(\beta) \in Def(\varepsilon_{\mathfrak{U}})\} \end{array} \right)$.

Let consider four cases: $x \in Not_A$, $x \in Not_K$, $x \in And_{A,K}$ and $x \in A \cup K$. Let show that the three first cases are impossible.

- Let suppose that $x \in Not_A$ with $x = \neg b$. Given that $\neg b \in Def(\varepsilon_{\mathfrak{U}})$, according to the definition of `Raf2Af`, we have $b \in \varepsilon_{\mathfrak{U}}$. As a consequence, we have: $x \in \{\neg a \in Not_A | a \in \varepsilon_{\mathfrak{U}}\}$, which is a contradiction.
- Let suppose that $x \in Not_K$ with $x = \neg\alpha$. Given that $\neg\alpha \in Def(\varepsilon_{\mathfrak{U}})$, according to the definition of `Raf2Af`, we have $\alpha \in \varepsilon_{\mathfrak{U}}$. As a consequence, we have: $x \in \{\neg\beta \in Not_K | \beta \in \varepsilon_{\mathfrak{U}}\}$, which is a contradiction.
- Let suppose that $x \in And_{A,K}$ with $x = s(\alpha).\alpha$. Given that $s(\alpha).\alpha \in Def(\varepsilon_{\mathfrak{U}})$, according to the definition of `Raf2Af`, we have thus: $\neg s(\alpha) \in \varepsilon_{\mathfrak{U}}$ or $\neg\alpha \in \varepsilon_{\mathfrak{U}}$. And so we have: $s(\alpha) \in Def(\varepsilon_{\mathfrak{U}})$ or $\alpha \in Def(\varepsilon_{\mathfrak{U}})$. As a consequence, we have: $x \in \{s(\beta).\beta \in And_{A,K} | \beta \in Def(\varepsilon_{\mathfrak{U}}) \text{ or } s(\beta) \in Def(\varepsilon_{\mathfrak{U}})\}$, which is a contradiction.

We prove so that: $x \in A \cup K$.

Given that $x \in Def(\varepsilon_{\mathcal{U}})$, following the definition of `Raf2Af`, there exists thus an argument $s(\alpha).\alpha \in (\varepsilon_{\mathcal{U}} \cap And_{A,K})$ attacking x. Given that $s(\alpha).\alpha \in (\varepsilon_{\mathcal{U}} \cap And_{A,K})$, we have following the definition of $\varepsilon_{\mathcal{U}}$: $s(\alpha) \in S$ and $\alpha \in Q$. As a consequence we have: $x \in$ RAF-$Def(\mathcal{U}) \cup$ RAF-$Inh(\mathcal{U})$.

We prove so that: RAF-$Def(\mathcal{U}) \cup$ RAF-$Inh(\mathcal{U}) \supseteq$

$$Def(\varepsilon_{\mathcal{U}}) \setminus \left(\begin{array}{l} \{\neg a \in Not_A | a \in \varepsilon_{\mathcal{U}}\} \\ \cup \{\neg \beta \in Not_K | \beta \in \varepsilon_{\mathcal{U}}\} \\ \cup \{s(\beta).\beta \in And_{A,K} | \beta \in Def(\varepsilon_{\mathcal{U}}) \text{ or } s(\beta) \in Def(\varepsilon_{\mathcal{U}})\} \end{array} \right)$$

Assertion 2: $\text{RAF-}Acc(\mathcal{U}) = Acc(\varepsilon_{\mathcal{U}}) \setminus \left(\begin{array}{l} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup \{\neg \beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup \{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \end{array} \right)$

- **Step 1:** $\text{RAF-}Acc(\mathcal{U}) \subseteq Acc(\varepsilon_{\mathcal{U}}) \setminus \left(\begin{array}{l} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup \{\neg \beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup \{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \end{array} \right)$

  Let $x \in$ RAF-$Acc(\mathcal{U})$. For any attack $\alpha \in K$ such that $t(\alpha) = x$, we have thus:

$$\alpha \in \text{RAF-}Inh(\mathcal{U}) \text{ or } s(\alpha) \in \text{RAF-}Def(\mathcal{U})$$

  As Assertion 1 holds, we have so:

$$\alpha \in Def(\varepsilon_{\mathcal{U}}) \setminus \left(\begin{array}{l} \{\neg a \in Not_A | a \in \varepsilon_{\mathcal{U}}\} \\ \cup \{\neg \beta \in Not_K | \beta \in \varepsilon_{\mathcal{U}}\} \\ \cup \{s(\beta).\beta \in And_{A,K} | \beta \in Def(\varepsilon_{\mathcal{U}}) \text{ or } s(\beta) \in Def(\varepsilon_{\mathcal{U}})\} \end{array} \right)$$

or

$$s(\alpha) \in Def(\varepsilon_{\mathcal{U}}) \setminus \left(\begin{array}{l} \{\neg a \in Not_A | a \in \varepsilon_{\mathcal{U}}\} \\ \cup \{\neg \beta \in Not_K | \beta \in \varepsilon_{\mathcal{U}}\} \\ \cup \{s(\beta).\beta \in And_{A,K} | \beta \in Def(\varepsilon_{\mathcal{U}}) \text{ or } s(\beta) \in Def(\varepsilon_{\mathcal{U}})\} \end{array} \right)$$

As a consequence, following the definition of `Raf2Af`, we have:

$$\neg\alpha \in \varepsilon_{\mathcal{U}} \text{ or } \neg s(\alpha) \in \varepsilon_{\mathcal{U}}$$

Furthermore, since $s(\alpha).\alpha \in And_{A,K}$, we have:

$$s(\alpha).\alpha \in Def(\varepsilon_{\mathcal{U}}) \cap And_{A,K}$$

As a consequence, as it is the case of any attack α attacking x, we have:

$$x \in Acc(\varepsilon_{\mathcal{U}})$$

Given that $x \in A \cup K$, we have, following the definition of `Raf2Af`: $x \notin (Not_A \cup Not_K \cup And_{A,K})$. As a consequence, we have:

$$x \in Acc(\varepsilon_{\mathcal{U}}) \setminus (Not_A \cup Not_K \cup And_{A,K})$$

We prove so that:

$$\text{RAF-}Acc(\mathcal{U}) \subseteq Acc(\varepsilon_{\mathcal{U}}) \setminus \begin{pmatrix} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{\neg\beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \end{pmatrix}$$

- **Step 2:** $\text{RAF-}Acc(\mathcal{U}) \supseteq Acc(\varepsilon_{\mathcal{U}}) \setminus \begin{pmatrix} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{\neg\beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \end{pmatrix}$

Let $x \in Acc(\varepsilon_{\mathcal{U}}) \setminus \begin{pmatrix} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{\neg\beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \end{pmatrix}$

Let consider four cases: $x \in Not_A$, $x \in Not_K$, $x \in And_{A,K}$ and $x \in A \cup K$. Let show that the three first cases are impossible.

  – Let suppose that $x \in Not_A$ with $x = \neg b$. Given that $\neg b \in Acc(\varepsilon_{\mathcal{U}})$, according to the definition of `Raf2Af`, we have $b \in Def(\varepsilon_{\mathcal{U}})$. As a consequence, we have: $x \in \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\}$, which is a contradiction.
  – Let suppose that $x \in Not_K$ with $x = \neg\alpha$. Given that $\neg\alpha \in Acc(\varepsilon_{\mathcal{U}})$, according to the definition of `Raf2Af`, we have $\alpha \in Def(\varepsilon_{\mathcal{U}})$. As a consequence, we have: $x \in \{\neg\beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\}$, which is a contradiction.
  – Let suppose that $x \in And_{A,K}$ with $x = s(\alpha).\alpha$. Given that $s(\alpha).\alpha \in Acc(\varepsilon_{\mathcal{U}})$, according to the definition of `Raf2Af`, we have thus: $\neg s(\alpha) \in Def(\varepsilon_{\mathcal{U}})$ and $\neg\alpha \in Def(\varepsilon_{\mathcal{U}})$. And so we have:

$s(\alpha) \in \varepsilon_{\mathcal{U}}$ and $\alpha \in \varepsilon_{\mathcal{U}}$. As a consequence, we have: $x \in \{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\}$, which is a contradiction.

We prove so that: $x \in A \cup K$.

Given that $x \in Acc(\varepsilon_{\mathcal{U}})$, then following the definition of `Raf2Af`, for any argument $s(\alpha).\alpha$ attacking x, we have: $s(\alpha).\alpha \in Def(\varepsilon_{\mathcal{U}})$. As a consequence, following the definition of `Raf2Af`, we have:

$$\neg s(\alpha) \in \varepsilon_{\mathcal{U}} \text{ or } \neg\alpha \in \varepsilon_{\mathcal{U}}$$

And so:

$$s(\alpha) \in Def(\varepsilon_{\mathcal{U}}) \text{ or } \alpha \in Def(\varepsilon_{\mathcal{U}})$$

As Assertion 1 holds and as $s(\alpha) \in A$ and $\alpha \in K$, we have:

$$s(\alpha) \in \text{RAF-}Def(\mathcal{U}) \text{ or } \alpha \in \text{RAF-}Inh(\mathcal{U})$$

as it is the case of any attack α attacking x, we have:

$$x \in \text{RAF-}Acc(\mathcal{U})$$

We prove so that:

$$\text{RAF-}Acc(\mathcal{U}) \supseteq Acc(\varepsilon_{\mathcal{U}}) \setminus \left(\begin{array}{c} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{\neg\beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \end{array} \right)$$

■

Proof of Proposition 25 on page 117.
Assertion 1: $\mathcal{U} = \langle S, Q\rangle$ is a RAF-*complete* structure in $\mathcal{RAF}$ *iff* $\varepsilon_{\mathcal{U}}$ is a *complete* extension in $\mathcal{AF}$.

$\mathcal{U} = \langle S, Q\rangle$ is a RAF-*complete* structure in $\mathcal{RAF}$ *iff* $(S \cup Q) = \text{RAF-}Acc(\mathcal{U})$.
Following Proposition 24 on page 117, we have thus:

$$\mathcal{U} = \langle S, Q\rangle \text{ is a RAF-}complete \text{ structure in } \mathcal{RAF}$$
$$iff$$
$$(S \cup Q) = Acc(\varepsilon_{\mathcal{U}}) \setminus \left(\begin{array}{c} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{\neg\beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \end{array} \right)$$

And so:

$$\mathcal{U} = \langle S, Q\rangle \text{ is a RAF-\textit{complete} structure in } \mathcal{RAF}$$
$$\textit{iff}$$
$$(S \cup Q) \cup \begin{pmatrix} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{\neg\beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \end{pmatrix} = Acc(\varepsilon_{\mathcal{U}}) \tag{18.77}$$

Given that, for all $s(\beta).\beta \in And_{A,K}$ such that $s(\beta).\beta \in \varepsilon_{\mathcal{U}}$ we have following the definition of $\varepsilon_{\mathcal{U}}$: $\beta \in Q, s(\beta) \in S$, from Equation (18.77), we have then:

$$\mathcal{U} = \langle S, Q\rangle \text{ is a RAF-\textit{complete} structure in } \mathcal{RAF}$$
$$\textit{iff}$$
$$(S \cup Q) \cup \begin{pmatrix} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{\neg\beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | \beta \in Q, s(\beta) \in S\} \end{pmatrix} = Acc(\varepsilon_{\mathcal{U}}) \tag{18.78}$$

Following the definition of `Raf2Af`:

$$\begin{gathered} \neg a \in Not_A \textit{ iff } a \in A \\ \text{and} \\ \neg\beta \in Not_K \textit{ iff } \beta \in K \end{gathered} \tag{18.79}$$

Furthermore, following Proposition 24 on page 117, we have:

$$Def(\varepsilon_{\mathcal{U}}) = \begin{pmatrix} \text{RAF-}\textit{Def}(\mathcal{U}) \cup \text{RAF-}\textit{Inh}(\mathcal{U}) \\ \cup\{\neg a \in Not_A | a \in \varepsilon_{\mathcal{U}}\} \cup \{\neg\beta \in Not_K | \beta \in \varepsilon_{\mathcal{U}}\} \\ \cup\{s(\beta).\beta \in And_{A,K} | \beta \in Def(\varepsilon_{\mathcal{U}}) \text{ or } s(\beta) \in Def(\varepsilon_{\mathcal{U}})\} \end{pmatrix} \tag{18.80}$$

From Equations (18.79) and (18.80) we have so:

$$\begin{gathered} a \in Def(\varepsilon_{\mathcal{U}}) \textit{ iff } a \in \text{RAF-}\textit{Def}(\mathcal{U}) \\ \text{and} \\ \beta \in Def(\varepsilon_{\mathcal{U}}) \textit{ iff } \beta \in \text{RAF-}\textit{Inh}(\mathcal{U}) \end{gathered} \tag{18.81}$$

As a consequence, we have from Equations (18.78) and (18.81) on the previous page:

$$\mathcal{U} = \langle S, Q \rangle \text{ is a RAF-}\textit{complete} \text{ structure in } \mathcal{RAF}$$
$$\textit{iff}$$
$$(S \cup Q) \cup \left(\begin{array}{c} \{\neg a \in Not_A | a \in \text{RAF-}Def(\mathcal{U})\} \\ \cup\{\neg \beta \in Not_K | \beta \in \text{RAF-}Inh(\mathcal{U})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | \beta \in Q, s(\beta) \in S\} \end{array} \right) = Acc(\varepsilon_{\mathcal{U}})$$

Following the definition of $\varepsilon_{\mathcal{U}}$, we have thus:

$$\mathcal{U} = \langle S, Q \rangle \text{ is a RAF-}\textit{complete} \text{ structure in } \mathcal{RAF}$$
$$\textit{iff}$$
$$\varepsilon_{\mathcal{U}} = Acc(\varepsilon_{\mathcal{U}})$$

We prove so that:

$$\mathcal{U} = \langle S, Q \rangle \text{ is a RAF-}\textit{complete} \text{ structure in } \mathcal{RAF}$$
$$\textit{iff}$$
$$\varepsilon_{\mathcal{U}} \text{ is a } \textit{complete} \text{ extension in } \mathcal{AF}$$

Assertion 2: $\mathcal{U} = \langle S, Q \rangle$ is a RAF-*grounded* structure in $\mathcal{RAF}$ *iff* $\varepsilon_{\mathcal{U}}$ is a *grounded* extension in $\mathcal{AF}$.

$\varepsilon_{\mathcal{U}}$ is a *grounded* extension in $\mathcal{AF}$ *iff* there is no *complete* extension $\varepsilon_{\mathcal{U}'}$ in $\mathcal{AF}$ (with $\mathcal{U}' = \langle S', Q' \rangle$) such that: $\varepsilon_{\mathcal{U}'} \subset \varepsilon_{\mathcal{U}}$.

We have so:

$$\varepsilon_{\mathcal{U}} \in \sigma_{gr}(\mathcal{AF})$$
$$\textit{iff}$$
$$\nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t. } Acc(\varepsilon_{\mathcal{U}'}) \subset Acc(\varepsilon_{\mathcal{U}})$$

Following Proposition 24 on page 117, we have:

$$\varepsilon_{\mathcal{U}} \in \sigma_{gr}(\mathcal{AF})$$
$$\textit{iff}$$
$$\nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t.}$$
$$\text{RAF-}Acc(\mathcal{U}') \cup \left(\begin{array}{c} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}'})\} \\ \cup\{\neg \beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}'})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}'}\} \end{array} \right)$$
$$\subset$$
$$\text{RAF-}Acc(\mathcal{U}) \cup \left(\begin{array}{c} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{\neg \beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \end{array} \right)$$

Removing $(Not_A \cup Not_K \cup And_{A,K})$ from both sides gives us a $\subseteq$-inclusion:

$$\varepsilon_{\mathcal{U}} \in \sigma_{gr}(\mathcal{AF})$$
$$\textit{iff}$$
$$\nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t.}$$
$$\left(\text{RAF-}Acc(\mathcal{U}') \cup \left(\begin{array}{c} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}'})\} \\ \cup\{\neg \beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}'})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}'}\} \end{array} \right) \right) \setminus (Not_A \cup Not_K \cup And_{A,K})$$
$$\subseteq$$
$$\left(\text{RAF-}Acc(\mathcal{U}) \cup \left(\begin{array}{c} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{\neg \beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \end{array} \right) \right) \setminus (Not_A \cup Not_K \cup And_{A,K})$$

We have thus:

$$\varepsilon_{\mathcal{U}} \in \sigma_{gr}(\mathcal{AF})$$
$$\textit{iff}$$
$$\nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t. RAF-}Acc(\mathcal{U}') \subseteq \text{RAF-}Acc(\mathcal{U})$$

Given that, following Assertion 1, $\varepsilon_{\mathcal{U}'}$ and $\varepsilon_{\mathcal{U}}$ are *complete iff* $\mathcal{U}'$ and $\mathcal{U}$ are RAF-*complete*, we have thus:

$$\varepsilon_{\mathcal{U}} \in \sigma_{gr}(\mathcal{AF}) \\ \textit{iff} \\ \nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t. } \mathcal{U}' \subseteq \mathcal{U}$$

Given that $\varepsilon_{\mathcal{U}'} \neq \varepsilon_{\mathcal{U}}$ *iff* $\mathcal{U}' \neq \mathcal{U}$, we have thus:

$$\varepsilon_{\mathcal{U}} \in \sigma_{gr}(\mathcal{AF}) \\ \textit{iff} \\ \nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t. } \mathcal{U}' \subset \mathcal{U}$$

We prove so that $\varepsilon_{\mathcal{U}}$ is a *grounded* extension in $\mathcal{AF}$ *iff* $\mathcal{U} = \langle S, Q \rangle$ is a RAF-*grounded* structure in $\mathcal{RAF}$.

Assertion 3: $\mathcal{U} = \langle S, Q \rangle$ is a RAF-*preferred* structure in $\mathcal{RAF}$ *iff* $\varepsilon_{\mathcal{U}}$ is a *preferred* extension in $\mathcal{AF}$.

$\varepsilon_{\mathcal{U}}$ is a *preferred* extension in $\mathcal{AF}$ *iff* there is no *complete* extension $\varepsilon_{\mathcal{U}'}$ in $\mathcal{AF}$ (with $\mathcal{U}' = \langle S', Q' \rangle$) such that: $\varepsilon_{\mathcal{U}} \subset \varepsilon_{\mathcal{U}'}$.

We have so:

$$\varepsilon_{\mathcal{U}} \in \sigma_{pr}(\mathcal{AF}) \\ \textit{iff} \\ \nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t. } Acc(\varepsilon_{\mathcal{U}}) \subset Acc(\varepsilon_{\mathcal{U}'})$$

Following Proposition 24 on page 117, we have:

$$\varepsilon_{\mathcal{U}} \in \sigma_{pr}(\mathcal{AF}) \\ \textit{iff} \\ \nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t.}$$

$$\text{RAF-}Acc(\mathcal{U}) \cup \begin{pmatrix} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{\neg \beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \end{pmatrix}$$

$$\subset$$

$$\text{RAF-}Acc(\mathcal{U}') \cup \begin{pmatrix} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}'})\} \\ \cup\{\neg \beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}'})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}'}\} \end{pmatrix}$$

Removing $(Not_A \cup Not_K \cup And_{A,K})$ from both sides gives us a $\subseteq$-inclusion:

$$\varepsilon_{\mathcal{U}} \in \sigma_{pr}(\mathcal{AF})$$
$$\textit{iff}$$
$$\nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t.}$$
$$\left(\text{RAF-}Acc(\mathcal{U}) \cup \left(\begin{array}{c} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{\neg\beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \end{array} \right) \right) \setminus (Not_A \cup Not_K \cup And_{A,K})$$
$$\subseteq$$
$$\left(\text{RAF-}Acc(\mathcal{U}') \cup \left(\begin{array}{c} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}'})\} \\ \cup\{\neg\beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}'})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}'}\} \end{array} \right) \right) \setminus (Not_A \cup Not_K \cup And_{A,K})$$

We have thus:

$$\varepsilon_{\mathcal{U}} \in \sigma_{pr}(\mathcal{AF})$$
$$\textit{iff}$$
$$\nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t. RAF-}Acc(\mathcal{U}) \subseteq \text{RAF-}Acc(\mathcal{U}')$$

Given that, following Assertion 1, $\varepsilon_{\mathcal{U}'}$ and $\varepsilon_{\mathcal{U}}$ are *complete iff* $\mathcal{U}'$ and $\mathcal{U}$ are RAF-*complete*, we have thus:

$$\varepsilon_{\mathcal{U}} \in \sigma_{pr}(\mathcal{AF})$$
$$\textit{iff}$$
$$\nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t. } \mathcal{U} \subseteq \mathcal{U}'$$

Given that $\varepsilon_{\mathcal{U}'} \neq \varepsilon_{\mathcal{U}}$ *iff* $\mathcal{U}' \neq \mathcal{U}$, we have thus:

$$\varepsilon_{\mathcal{U}} \in \sigma_{pr}(\mathcal{AF})$$
$$\textit{iff}$$
$$\nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t. } \mathcal{U} \subset \mathcal{U}'$$

We prove so that $\varepsilon_{\mathcal{U}}$ is a *preferred* extension in $\mathcal{AF}$ *iff* $\mathcal{U} = \langle S, Q \rangle$ is a RAF-*preferred* structure in $\mathcal{RAF}$.

Assertion 4: $\mathcal{U} = \langle S, Q \rangle$ is a RAF-*stable* structure in $\mathcal{RAF}$ *iff* $\varepsilon_{\mathcal{U}}$ is a *stable* extension in $\mathcal{AF}$.

- **Step 1:** If $\mathcal{U} = \langle S, Q \rangle$ is a RAF-*stable* structure in $\mathcal{RAF}$ then $\varepsilon_{\mathcal{U}}$ is a *stable* extension in $\mathcal{AF}$.

  If $\mathcal{U} = \langle S, Q \rangle$ is a RAF-*stable* structure in $\mathcal{RAF}$ then $\nexists x \in (A \cup K)$ such that $x \notin \mathcal{U}$ and $x \notin$ (RAF-$Def(\mathcal{U}) \cup$ RAF-$Inh(\mathcal{U})$). If $\mathcal{U}$ is a RAF-*stable* structure in $\mathcal{RAF}$ then $\mathcal{U}$ is also RAF-*complete*. Following Assertion 1, $\varepsilon_{\mathcal{U}}$ is thus a *complete* extension in $\mathcal{AF}$. Let suppose that $\varepsilon_{\mathcal{U}}$ is not *stable*. There exists thus $x \in (A' \cup K')$ such that $x \notin \varepsilon_{\mathcal{U}}$ and $x \notin Def(\varepsilon_{\mathcal{U}})$.

Let consider two cases: $x \in A \cup K$ and $x \notin (A \cup K)$.

Case 1: Given that $\mathcal{U}$ is RAF-*stable*, if $x \in A \cup K$, we have so: $x \in (S \cup Q \cup \text{RAF-}Def(\mathcal{U}) \cup \text{RAF-}Inh(\mathcal{U}))$.
As shown in Proof of Assertion 1 (Equation (18.81) on page 243):

$$\begin{gathered} a \in A \cap Def(\varepsilon_{\mathcal{U}}) \textit{ iff } a \in \text{RAF-}Def(\mathcal{U}) \\ \text{and} \\ \beta \in K \cap Def(\varepsilon_{\mathcal{U}}) \textit{ iff } \beta \in \text{RAF-}Inh(\mathcal{U}) \end{gathered} \tag{18.82}$$

As a consequence if $x \in A \cup K$ then:

$$x \in (S \cup Q \cup (A \cap Def(\varepsilon_{\mathcal{U}})) \cup (K \cap Def(\varepsilon_{\mathcal{U}})))$$

As $x \notin \varepsilon_{\mathcal{U}}$ and $x \notin Def(\varepsilon_{\mathcal{U}})$ there is a contradiction.

Case 2: If $x \notin (A \cup K)$ then: $x \in (Not_A \cup Not_K \cup And_{A,K})$.
Given that:

$$x \notin \varepsilon_{\mathcal{U}} \text{ and } x \notin Def(\varepsilon_{\mathcal{U}})$$

We have thus three possible cases:

- $x \in Not_A \setminus \{\neg a \in Not_A | a \in (Def(\varepsilon_{\mathcal{U}}) \cup \varepsilon_{\mathcal{U}})\}$
- $x \in Not_K \setminus \{\neg \beta \in Not_K | \beta \in (Def(\varepsilon_{\mathcal{U}}) \cup \varepsilon_{\mathcal{U}})\}$
- $x \in And_{A,K} \setminus \{s(\beta).\beta \in And_{A,K} | \beta \in (Def(\varepsilon_{\mathcal{U}}) \cup \varepsilon_{\mathcal{U}})\}$

Given that following the definition of `Raf2Af`:

$$\begin{gathered} \neg a \in Not_A \textit{ iff } a \in A \\ \text{and} \\ \neg \beta \in Not_K \textit{ iff } \beta \in K \\ \text{and} \\ s(\beta).\beta \in And_{A,K} \textit{ iff } \beta \in K \end{gathered} \tag{18.83}$$

We have thus:

- $x \in Not_A \setminus \{\neg a \in Not_A | a \in (A \cap Def(\varepsilon_{\mathcal{U}})) \cup (A \cap \varepsilon_{\mathcal{U}})\}$
- $x \in Not_K \setminus \{\neg \beta \in Not_K | \beta \in (K \cap Def(\varepsilon_{\mathcal{U}})) \cup (K \cap \varepsilon_{\mathcal{U}})\}$
- $x \in And_{A,K} \setminus \{s(\beta).\beta \in And_{A,K} | \beta \in (K \cap Def(\varepsilon_{\mathcal{U}})) \cup (K \cap \varepsilon_{\mathcal{U}})\}$

As shown in Proof of Assertion 1 (Equation (18.81) on page 243):

$$\begin{gathered} A \cap Def(\varepsilon_{\mathcal{U}}) = \text{RAF-}Def(\mathcal{U}) \\ \text{and} \\ K \cap Def(\varepsilon_{\mathcal{U}}) = \text{RAF-}Inh(\mathcal{U}) \end{gathered} \tag{18.84}$$

We have thus:

- $x \in Not_A \setminus \{\neg a \in Not_A | a \in (\text{RAF-}Def(\mathcal{U}) \cup S)\}$
- $x \in Not_K \setminus \{\neg \beta \in Not_K | \beta \in (\text{RAF-}Inh(\mathcal{U}) \cup Q)\}$
- $x \in And_{A,K} \setminus \{s(\beta).\beta \in And_{A,K} | \beta \in (\text{RAF-}Inh(\mathcal{U}) \cup Q)\}$

Given that $\mathcal{U}$ is RAF-*stable*, we have thus:

- $x \in Not_A \setminus Not_A = \varnothing$
- $x \in Not_K \setminus Not_K = \varnothing$
- $x \in And_{A,K} \setminus And_{A,K} = \varnothing$

There is thus a contradiction.

We prove so that if $\mathcal{U} = \langle S, Q\rangle$ is a RAF-*stable* structure in $\mathcal{RAF}$ then $\varepsilon_{\mathcal{U}}$ is a *stable* extension in $\mathcal{AF}$.

- **Step 2:** If $\varepsilon_{\mathcal{U}}$ is a *stable* extension in $\mathcal{AF}$ then $\mathcal{U} = \langle S, Q\rangle$ is a RAF-*stable* structure in $\mathcal{RAF}$.

  If $\varepsilon_{\mathcal{U}}$ is a *stable* extension in $\mathcal{AF}$ then $\nexists x \in (A' \cup K')$ such that $x \notin \varepsilon_{\mathcal{U}}$ and $x \notin Def(\varepsilon_{\mathcal{U}})$. If $\varepsilon_{\mathcal{U}}$ is a *stable* extension in $\mathcal{AF}$ then $\varepsilon_{\mathcal{U}}$ is also *complete*. As Assertion 1 holds, then $\mathcal{U}$ is RAF-*complete*. Let suppose that $\mathcal{U}$ is not RAF-*stable*. There exists thus $x \in (A \cup K)$ such that $x \notin \mathcal{U}$ and $x \notin (\text{RAF-}Def(\mathcal{U}) \cup \text{RAF-}Inh(\mathcal{U}))$.

  As shown in Proof of Assertion 1 (Equation (18.81) on page 243):
  $$A \cap Def(\varepsilon_{\mathcal{U}}) = \text{RAF-}Def(\mathcal{U})$$
  and
  $$K \cap Def(\varepsilon_{\mathcal{U}}) = \text{RAF-}Inh(\mathcal{U})$$

  We have thus:
  $$x \notin \Big((A \cap Def(\varepsilon_{\mathcal{U}})) \cup (K \cap Def(\varepsilon_{\mathcal{U}})) \cup S \cup Q \Big)$$

  Given that $\varepsilon_{\mathcal{U}}$ is *stable*, we have thus: $x \notin (A \cup K)$, which is a contradiction.

  We prove so that if $\varepsilon_{\mathcal{U}}$ is a *stable* extension in $\mathcal{AF}$ then $\mathcal{U} = \langle S, Q\rangle$ is a RAF-*stable* structure in $\mathcal{RAF}$.

Assertion 5: $\mathcal{U} = \langle S, Q\rangle$ is a RAF-*semi-stable* structure in $\mathcal{RAF}$ *iff* $\varepsilon_{\mathcal{U}}$ is a *semi-stable* extension in $\mathcal{AF}$.

$\varepsilon_{\mathcal{U}}$ is a *semi-stable* extension in $\mathcal{AF}$ *iff* there is no *complete* extension $\varepsilon_{\mathcal{U}'}$ in $\mathcal{AF}$ (with $\mathcal{U}' = \langle S', Q'\rangle$) such that: $(\varepsilon_{\mathcal{U}} \cup Def(\varepsilon_{\mathcal{U}})) \subset (\varepsilon_{\mathcal{U}'} \cup Def(\varepsilon_{\mathcal{U}'}))$.

We have so:
$$\varepsilon_{\mathcal{U}} \in \sigma_{sst}(\mathcal{AF})$$
iff
$$\nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t. } (Acc(\varepsilon_{\mathcal{U}}) \cup Def(\varepsilon_{\mathcal{U}})) \subset (Acc(\varepsilon_{\mathcal{U}'}) \cup Def(\varepsilon_{\mathcal{U}'}))$$

Following Proposition 24 on page 117, we have:

$$\varepsilon_{\mathcal{U}} \in \sigma_{sst}(\mathcal{AF})$$
$$\textit{iff}$$
$$\nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t.}$$
$$\begin{pmatrix} \text{RAF-}Acc(\mathcal{U}) \\ \cup\text{RAF-}Def(\mathcal{U}) \\ \cup\text{RAF-}Inh(\mathcal{U}) \end{pmatrix} \cup \begin{pmatrix} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{\neg\beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}}\} \\ \cup\{\neg a \in Not_A | a \in \varepsilon_{\mathcal{U}}\} \\ \cup\{\neg\beta \in Not_K | \beta \in \varepsilon_{\mathcal{U}}\} \\ \cup\{s(\beta).\beta \in And_{A,K} | \beta \in Def(\varepsilon_{\mathcal{U}}) \text{ or } s(\beta) \in Def(\varepsilon_{\mathcal{U}})\} \end{pmatrix}$$
$$\subset$$
$$\begin{pmatrix} \text{RAF-}Acc(\mathcal{U}') \\ \cup\text{RAF-}Def(\mathcal{U}') \\ \cup\text{RAF-}Inh(\mathcal{U}') \end{pmatrix} \cup \begin{pmatrix} \{\neg a \in Not_A | a \in Def(\varepsilon_{\mathcal{U}'})\} \\ \cup\{\neg\beta \in Not_K | \beta \in Def(\varepsilon_{\mathcal{U}'})\} \\ \cup\{s(\beta).\beta \in And_{A,K} | s(\beta).\beta \in \varepsilon_{\mathcal{U}'}\} \\ \cup\{\neg a \in Not_A | a \in \varepsilon_{\mathcal{U}'}\} \\ \cup\{\neg\beta \in Not_K | \beta \in \varepsilon_{\mathcal{U}'}\} \\ \cup\{s(\beta).\beta \in And_{A,K} | \beta \in Def(\varepsilon_{\mathcal{U}'}) \text{ or } s(\beta) \in Def(\varepsilon_{\mathcal{U}'})\} \end{pmatrix}$$

Removing $(Not_A \cup Not_K \cup And_{A,K})$ from both sides gives us a $\subseteq$-inclusion:

$$\varepsilon_{\mathcal{U}} \in \sigma_{sst}(\mathcal{AF})$$
$$\textit{iff}$$
$$\nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t. } \begin{pmatrix} \text{RAF-}Acc(\mathcal{U}) \\ \cup\text{RAF-}Def(\mathcal{U}) \\ \cup\text{RAF-}Inh(\mathcal{U}) \end{pmatrix} \subseteq \begin{pmatrix} \text{RAF-}Acc(\mathcal{U}') \\ \cup\text{RAF-}Def(\mathcal{U}') \\ \cup\text{RAF-}Inh(\mathcal{U}') \end{pmatrix}$$

Given that, following Assertion 1, $\varepsilon_{\mathcal{U}'}$ and $\varepsilon_{\mathcal{U}}$ are *complete iff* $\mathcal{U}'$ and $\mathcal{U}$ are RAF-*complete*, we have thus:

$$\varepsilon_{\mathcal{U}} \in \sigma_{sst}(\mathcal{AF})$$
$$\textit{iff}$$
$$\nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t. } \mathcal{U} \cup \begin{pmatrix} \text{RAF-}Def(\mathcal{U}) \\ \cup\text{RAF-}Inh(\mathcal{U}) \end{pmatrix} \subseteq \mathcal{U}' \cup \begin{pmatrix} \text{RAF-}Def(\mathcal{U}') \\ \cup\text{RAF-}Inh(\mathcal{U}') \end{pmatrix}$$

Given that $\varepsilon_{\mathcal{U}'} \neq \varepsilon_{\mathcal{U}}$ *iff* $\mathcal{U}' \neq \mathcal{U}$, we have thus:

$$\varepsilon_{\mathcal{U}} \in \sigma_{sst}(\mathcal{AF})$$
$$\textit{iff}$$
$$\nexists \varepsilon_{\mathcal{U}'} \in \sigma_{co}(\mathcal{AF}) \text{ s.t. } \mathcal{U} \cup \begin{pmatrix} \text{RAF-}Def(\mathcal{U}) \\ \cup\text{RAF-}Inh(\mathcal{U}) \end{pmatrix} \subset \mathcal{U}' \cup \begin{pmatrix} \text{RAF-}Def(\mathcal{U}') \\ \cup\text{RAF-}Inh(\mathcal{U}') \end{pmatrix}$$

We prove so that $\varepsilon_{\mathcal{U}}$ is a *semi-stable* extension in $\mathcal{AF}$ *iff* $\mathcal{U} = \langle S, Q \rangle$ is a RAF-*semi-stable* structure in $\mathcal{RAF}$. ■

Proofs of Chapter 13: Complexity

Proof of Proposition 28 on page 121.

Assertion 1: *RAF-Cred*$_\sigma$ accepts $(\mathcal{RAF}, a)$ *iff* *Cred*$_\sigma$ accepts $(\mathcal{AF}, a)$.

$$\begin{aligned} RAF\text{-}Cred_\sigma \text{ accepts } (\mathcal{RAF}, a) &\ \textit{iff}\ \exists \mathcal{U} \in \sigma(\mathcal{RAF}) \text{ s.t. } a \in \mathcal{U} \\ &\ \textit{iff}\ \exists \varepsilon_{\mathcal{U}} \in \sigma(\mathcal{AF}) \text{ s.t. } a \in \varepsilon_{\mathcal{U}} \quad \text{(following Proposition 25 on page 117)} \\ &\ \textit{iff}\ Cred_\sigma \text{ accepts } (\mathcal{AF}, a) \end{aligned}$$

Assertion 2: *RAF-Skep*$_\sigma$ accepts $(\mathcal{RAF}, a)$ *iff* *Skep*$_\sigma$ accepts $(\mathcal{AF}, a)$.

$$\begin{aligned} RAF\text{-}Skep_\sigma \text{ accepts } (\mathcal{RAF}, a) &\ \textit{iff}\ \forall \mathcal{U} \in \sigma(\mathcal{RAF}),\ a \in \mathcal{U} \\ &\ \textit{iff}\ \forall \varepsilon_{\mathcal{U}} \in \sigma(\mathcal{AF}),\ a \in \varepsilon_{\mathcal{U}} \quad \text{(following Proposition 25 on page 117)} \\ &\ \textit{iff}\ Skep_\sigma \text{ accepts } (\mathcal{AF}, a) \end{aligned}$$

Assertion 3: *RAF-Ver*$_\sigma$ accepts $(\mathcal{RAF}, \mathcal{U})$ *iff* *Ver*$_\sigma$ accepts $(\mathcal{AF}, \varepsilon_{\mathcal{U}})$.

$$\begin{aligned} RAF\text{-}Ver_\sigma \text{ accepts } (\mathcal{RAF}, \mathcal{U}) &\ \textit{iff}\ \mathcal{U} \in \sigma(\mathcal{RAF}) \\ &\ \textit{iff}\ \varepsilon_{\mathcal{U}} \in \sigma(\mathcal{AF}) \quad \text{(following Proposition 25 on page 117)} \\ &\ \textit{iff}\ Ver_\sigma \text{ accepts } (\mathcal{AF}, \varepsilon_{\mathcal{U}}) \end{aligned}$$

Assertion 4: *RAF-Exists*$_\sigma$ accepts $\mathcal{RAF}$ *iff* *Exists*$_\sigma$ accepts $\mathcal{AF}$.

$$\begin{aligned} \textit{RAF-Exists}_\sigma \text{ accepts } \mathcal{RAF} \; & \textit{iff } \exists \mathcal{U} \in \sigma(\mathcal{RAF}) \\ & \textit{iff } \exists \varepsilon_{\mathcal{U}} \in \sigma(\mathcal{AF}) \quad \text{(following Proposition 25 on page 117)} \\ & \textit{iff } \textit{Exists}_\sigma \text{ accepts } \mathcal{AF} \end{aligned}$$

Assertion 5: *RAF-Exists*$_\sigma^{\neg\varnothing}$ accepts $\mathcal{RAF}$ *iff* *Exists*$_\sigma^{\neg\varnothing}$ accepts $\mathcal{AF}$.

$$\begin{aligned} \textit{RAF-Exists}_\sigma^{\neg\varnothing} \text{ accepts } \mathcal{RAF} \; & \textit{iff } \exists (\mathcal{U} = \langle S, Q\rangle) \in \sigma(\mathcal{RAF}) \text{ s.t. } (S \cup Q) \neq \varnothing \\ & \textit{iff } \exists \varepsilon_{\mathcal{U}} \in \sigma(\mathcal{AF}) \text{ s.t. } \varepsilon_{\mathcal{U}} \neq \varnothing \quad \text{(following Proposition 25 on page 117)} \\ & \textit{iff } \textit{Exists}_\sigma^{\neg\varnothing} \text{ accepts } \mathcal{AF} \end{aligned}$$

Assertion 6: *RAF-Unique*$_\sigma$ accepts $\mathcal{RAF}$ *iff* *Unique*$_\sigma$ accepts $\mathcal{AF}$.

$$\begin{aligned} \textit{RAF-Unique}_\sigma \text{ accepts } \mathcal{RAF} \; & \textit{iff } \exists ! \mathcal{U} \in \sigma(\mathcal{RAF}) \\ & \textit{iff } \exists ! \varepsilon_{\mathcal{U}} \in \sigma(\mathcal{AF}) \quad \text{(following Proposition 25 on page 117)} \\ & \textit{iff } \textit{Unique}_\sigma \text{ accepts } \mathcal{AF} \end{aligned}$$

■

Proof of Proposition 29 on page 121. Given that `Raf2Af` is a polynomial time, log-space function (See Sections 16.4.3.1 and 16.4.4.1 on page 180 and on page 182), then according to Proposition 28 on page 121, for each semantics $\sigma \in \{$*complete*, *semi-stable*, *stable*, *preferred*, *grounded*$\}$ we have:

- *RAF-Cred*$_\sigma \leq_{\mathrm{L}}^{\texttt{Raf2Af}}$ *Cred*$_\sigma$
- *RAF-Skep*$_\sigma \leq_{\mathrm{L}}^{\texttt{Raf2Af}}$ *Skep*$_\sigma$
- *RAF-Ver*$_\sigma \leq_{\mathrm{L}}^{\texttt{Raf2Af}}$ *Ver*$_\sigma$
- *RAF-Exists*$_\sigma \leq_{\mathrm{L}}^{\texttt{Raf2Af}}$ *Exists*$_\sigma$
- *RAF-Exists*$_\sigma^{\neg\varnothing} \leq_{\mathrm{L}}^{\texttt{Raf2Af}}$ *Exists*$_\sigma^{\neg\varnothing}$
- *RAF-Unique*$_\sigma \leq_{\mathrm{L}}^{\texttt{Raf2Af}}$ *Unique*$_\sigma$

■

Proof of Proposition 30 on page 122. This proof is trivial considering Theorem 3 on page 97 and Proposition 23 on page 103. ■

Proof of Proposition 31 on page 122. Given that `Af2Raf` is a polynomial time, log-space function, then according to Proposition 30 on page 122, for each semantics $\sigma \in \{$*complete*, *semi-stable*, *stable*, *preferred*, *grounded*$\}$ we have:

- *Cred*$_\sigma \leq_{\mathrm{L}}^{\texttt{Af2Raf}}$ *RAF-Cred*$_\sigma$
- *Skep*$_\sigma \leq_{\mathrm{L}}^{\texttt{Af2Raf}}$ *RAF-Skep*$_\sigma$

- $Ver_\sigma \leq_{\mathtt{L}}^{\mathtt{Af2Raf}} RAF\text{-}Ver_\sigma$
- $Exists_\sigma \leq_{\mathtt{L}}^{\mathtt{Af2Raf}} RAF\text{-}Exists_\sigma$
- $Exists_\sigma^{\neg\emptyset} \leq_{\mathtt{L}}^{\mathtt{Af2Raf}} RAF\text{-}Exists_\sigma^{\neg\emptyset}$
- $Unique_\sigma \leq_{\mathtt{L}}^{\mathtt{Af2Raf}} RAF\text{-}Unique_\sigma$

■

Proof of Proposition 32 on page 122. Given that `Raf2Af` and `Af2Raf` are polynomial time procedures and that Propositions 29 and 31 on page 121 and on page 122 holds, then all the complexities are the same. ■

Proofs of Chapter 14: Decomposability and Hierarchy

Proofs of Section 14.1: SCC_{raf}

Proof of Proposition 33 on page 126. Given that $x \neq y$, $x \underset{\mathcal{RAF}}{\equiv} y$ *iff* we have the following facts:

- $\exists p \in Paths_{raf}(\mathcal{RAF})$ such that $p = (x, ..., e_{n-1}, y)$ and $y = t(e_{n-1})$.
- $\exists p' \in Paths_{raf}(\mathcal{RAF})$ such that $p' = (y, ..., o_{m-1}, x)$ and $x = t(o_{m-1})$.

Let c be the sequence formed by the concatenation of p and p' as follows:

$$c = (e_1 = x, ..., e_{n-1}, e_n = y = o_1, ..., o_m = x)$$

Given that p and p' are RAF-paths and given that $e_1 = o_m = x$ then c is a RAF-closed-walk.
For all $i \in \{1, ..., n\}$, let $u_i = e_i$. For all $i \in \{2, ..., m\}$, let $u_{i+n-1} = o_i$.
Let $l = n + m - 1$. We have thus:

$$c = (u_1 = x, ..., u_n = y, ..., u_l = x) \in ClosedWalk_{raf}(\mathcal{RAF})$$

With this re-indexation of c, we can easily see that:

- $x \in c$ and $l \in \{2, ..., l\}$ s.t. $u_l = x$
- $y \in c$ and $n \in \{2, ..., l\}$ s.t. $u_n = y$
- $e_{l-1} \in K$ as $x = t(o_{m-1}) = t(u_{l-1})$
- $e_{n-1} \in K$ as $y = t(e_{n-1}) = t(u_{n-1})$

■

Proof of Proposition 34 on page 126. For any $c \in ClosedWalk_{raf}(\mathcal{RAF})$, we have by definition: $|U| >= 1$. If $|U| = 1$ then trivially U is included in some $S \in SCCS_{raf}(\mathcal{RAF})$. Let consider that $|U| > 1$. Let $e_i \in U$ and $e_j \in U$ such that $e_i \neq e_j$. Given that $c \in ClosedWalk_{raf}(\mathcal{RAF})$, then following Proposition 33 on page 126 we have: $e_i \underset{\mathcal{RAF}}{\equiv} e_j$. As a consequence, e_i and e_j belong to the same $S \in SCCS_{raf}(\mathcal{RAF})$. We prove so that: U is included in some $S \in SCCS_{raf}(\mathcal{RAF})$. ■

The following lemma states that the source and the target of an attack (whose source is different from its attack) belong to the same SCC_{raf} *iff* there exists a RAF-path from the target to the source and whose before last element is an attack:

Lemma 3. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and let $\alpha \in K$. The following proposition holds:*

$$\exists(e_1 = t(\alpha), ..., e_n = s(\alpha)) \in Paths_{raf}(\mathcal{RAF}) \text{ s.t. } e_{n-1} \in K$$
$$\Longleftrightarrow$$
$$t(\alpha) \neq s(\alpha) \text{ and } (t(\alpha) \text{ and } s(\alpha) \text{ are in the same } S \in SCCS_{raf}(\mathcal{RAF}))$$

Proof of Lemma 3. In two steps:

- **Step 1:**

$$\exists(e_1 = t(\alpha), ..., e_n = s(\alpha)) \in Paths_{raf}(\mathcal{RAF}) \text{ s.t. } e_{n-1} \in K$$
$$\Longrightarrow$$
$$t(\alpha) \neq s(\alpha) \text{ and } (t(\alpha) \text{ and } s(\alpha) \text{ are in the same } S \in SCCS_{raf}(\mathcal{RAF}))$$

  If there exists a RAF-path $p = (e_1 = t(\alpha), ..., e_n = s(\alpha)) \in Paths_{raf}(\mathcal{RAF})$ such that $e_{n-1} \in K$, then (by Definition 82 on page 123) $t(\alpha) \neq s(\alpha)$. Furthermore, $c = (s(\alpha), \alpha, e_1 = t(\alpha), ..., e_n = s(\alpha)) \in Cycles_{raf}(\mathcal{RAF})$ and c attacks $s(\alpha)$ and $t(\alpha)$. Following Proposition 34 on page 126, we have so: $t(\alpha) \underset{\mathcal{RAF}}{\equiv} s(\alpha)$.

- **Step 2:**

$$t(\alpha) \neq s(\alpha) \text{ and } (t(\alpha) \text{ and } s(\alpha) \text{ are in the same } S \in SCCS_{raf}(\mathcal{RAF}))$$
$$\Longrightarrow$$
$$\exists(e_1 = t(\alpha), ..., e_n = s(\alpha)) \in Paths_{raf}(\mathcal{RAF}) \text{ s.t. } e_{n-1} \in K$$

  By definition, $t(\alpha) \underset{\mathcal{RAF}}{\equiv} s(\alpha) \implies \exists(e_1 = t(\alpha), ..., e_n = s(\alpha)) \in Paths_{raf}(\mathcal{RAF})$ s.t. $e_{n-1} \in K$.

From Steps 1 and 2, we prove Lemma 3. ■

The following lemma states implications over the different categories of attacks:

1. Self-attacking attacks (*i.e.* the target is the attack itself)
2. Attacks whose source is also their target
3. Non self-attacking attacks (including thus attacks in (2.))
4. Attacks whose source is different from their target (including thus attacks in (1.))
5. Non self-attacking attacks whose source is different from their target (intersection of (3.) and (4.))

Lemma 4. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be its corresponding AF. Let $(e_1, ..., e_n) \in Paths_{raf}(\mathcal{RAF})$ be a RAF-path. For any $i \in \{1, ..., n\}$:*

1. *$e_i \in K$ s.t. $t(e_i) = e_i \implies \exists! c = (e_i, ..., e_i) \in Cycles_{af}(\mathcal{AF})$ and $\exists(s(e_i), ..., t(e_i)) \in Paths_{af}(\mathcal{AF})$*

2. $e_i \in K$ *s.t.* $t(e_i) = s(e_i) \implies \exists c = (s(e_i), ..., s(e_i)) \in Cycles_{af}(\mathcal{AF})$ *and* $\exists!(e_i, ..., t(e_i)) \in Paths_{af}(\mathcal{AF})$

3. $e_i \in K$ *s.t.* $t(e_i) \neq e_i \implies \left(\begin{array}{c} \exists!(e_i, ..., t(e_i)) \in Paths_{af}(\mathcal{AF}) \\ \\ \textit{and} \\ \\ (\exists(s(e_i), ..., t(e_i)) \in Paths_{af}(\mathcal{AF}) \textit{ or } \exists(s(e_i), ..., t(e_i)) \in Cycles_{af}(\mathcal{AF})) \end{array}\right)$

4. $e_i \in K$ *s.t.* $t(e_i) \neq s(e_i) \implies \left(\begin{array}{c} (\exists!(e_i, ..., t(e_i)) \in Paths_{af}(\mathcal{AF}) \textit{ or } \exists!(e_i, ..., t(e_i)) \in Cycles_{af}(\mathcal{AF})) \\ \\ \textit{and} \\ \\ \exists(s(e_i), ..., t(e_i)) \in Paths_{af}(\mathcal{AF}) \end{array}\right)$

5. $e_i \in K$ *s.t.* $t(e_i) \neq s(e_i)$ *and* $t(e_i) \neq e_i \iff \exists!(e_i, ..., t(e_i)) \in Paths_{af}(\mathcal{AF})$ *and* $\exists(s(e_i), ..., t(e_i)) \in Paths_{af}(\mathcal{AF})$

Proof of Lemma 4.

Assertion 1: $e_i \in K$ s.t. $t(e_i) = e_i \implies \exists! c = (e_i, ..., e_i) \in Cycles_{af}(\mathcal{AF})$ and $\exists(s(e_i), ..., t(e_i)) \in Paths_{af}(\mathcal{AF})$
Let $\alpha \in K$ be an attack of $\mathcal{RAF}$ such that $t(\alpha) = \alpha$.

- According to the definition of `Raf2Af` (Definition 76 on page 115, see rules Not_K, $And_{A,K}$, K_2', K_4' and K_5'), there is a walk in `Raf2Af`$(\mathcal{AF})$ from α to $t(\alpha)$ which corresponds to the sequence of arguments: $(\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha))$. As $t(\alpha) = \alpha$ then we have: $(\alpha, \neg\alpha, s(\alpha).\alpha, \alpha) \in Cycles_{af}(\mathcal{AF})$. Furthermore, there exists an unique cycle whose first element is α. Indeed, by definition, an attack only has one target.

- Likewise (see rules Not_A, $And_{A,K}$, K_1', K_3' and K_5'), there is a walk in `Raf2Af`$(\mathcal{AF})$ from $s(\alpha)$ to $t(\alpha)$ which corresponds the sequence of arguments: $(s(\alpha), \neg s(\alpha), s(\alpha).\alpha, t(\alpha))$. Given that $t(\alpha) \neq s(\alpha)$, this walk is a path. Notice that this path may not be the unique going from $s(\alpha)$ to $t(\alpha)$. Indeed it could exists another attack whose source is $s(\alpha)$ and from which there exists a path to $t(\alpha)$.

Assertion 2: $e_i \in K$ s.t. $t(e_i) = s(e_i) \implies \exists c = (s(e_i), ..., s(e_i)) \in Cycles_{af}(\mathcal{AF})$ and $\exists!(e_i, ..., t(e_i)) \in Paths_{af}(\mathcal{AF})$
Let $\alpha \in K$ be an attack of $\mathcal{RAF}$ such that $t(\alpha) = s(\alpha)$.

- According to the definition of `Raf2Af` (Definition 76 on page 115, see rules Not_K, $And_{A,K}$, K_2', K_4' and K_5'), there is a walk in `Raf2Af`$(\mathcal{AF})$ from α to $t(\alpha)$ which corresponds to the sequence of arguments: $(\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha))$. Given that $t(e_i) \neq e_i$, this walk is a path. Furthermore, this path is the unique going from α to $t(\alpha)$ as, by definition, an attack only has one target.

- Likewise (see rules Not_A, $And_{A,K}$, K_1', K_3' and K_5'), there is a walk in `Raf2Af`$(\mathcal{AF})$ from $s(\alpha)$ to $t(\alpha)$ which corresponds the sequence of arguments: $(s(\alpha), \neg s(\alpha), s(\alpha).\alpha, t(\alpha))$. Given that $t(\alpha) = s(\alpha)$, this walk is a cycle. Notice that this cycle may not be the unique cycle whose first element is $s(\alpha)$. Indeed it could exists another attack whose source is $s(\alpha)$ and from which there exists a cycle to $s(\alpha)$.

Assertion 3:

$$e_i \in K \text{ s.t. } t(e_i) \neq e_i \implies \left(\begin{array}{c} \exists!(e_i, ..., t(e_i)) \in Paths_{af}(\mathcal{AF}) \\ \\ \text{and} \\ \\ (\exists(s(e_i), ..., t(e_i)) \in Paths_{af}(\mathcal{AF}) \text{ or } \exists(s(e_i), ..., t(e_i)) \in Cycles_{af}(\mathcal{AF})) \end{array} \right)$$

Let $\alpha \in K$ such that $t(\alpha) \neq \alpha$.

- According to the definition of `Raf2Af` (Definition 76 on page 115, see rules Not_K, $And_{A,K}$, K'_2, K'_4 and K'_5), there is a walk in $\texttt{Raf2Af}(\mathcal{AF})$ from α to $t(\alpha)$ which corresponds to the sequence of arguments: $(\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha))$. Given that $t(\alpha) \neq \alpha$, this walk is a path. Furthermore, this path is the unique path going from α to $t(\alpha)$ as, by definition, an attack only has one target.

  We have thus:
$$\alpha \in K \text{ s.t. } t(\alpha) \neq \alpha \implies \exists!(\alpha, ..., t(\alpha)) \in Paths_{af}(\mathcal{AF}) \tag{18.85}$$

- Likewise (see rules Not_A, $And_{A,K}$, K'_1, K'_3 and K'_5), there is a walk in $\texttt{Raf2Af}(\mathcal{AF})$ from $s(\alpha)$ to $t(\alpha)$ which corresponds the sequence of arguments: $(s(\alpha), \neg s(\alpha), s(\alpha).\alpha, t(\alpha))$. If $t(\alpha) \neq s(\alpha)$, then this walk is a path. Otherwise it is a cycle. Notice that, in both cases, it may not be the unique path or cycle going from $s(\alpha)$ to $t(\alpha)$. Indeed it could exists another attack whose source is $s(\alpha)$ and from which there exists a path to $t(\alpha)$.

  We have thus:
$$\alpha \in K \text{ s.t. } t(\alpha) \neq \alpha \implies \exists(s(\alpha), ..., t(\alpha)) \in Paths_{af}(\mathcal{AF}) \text{ or } \exists(s(\alpha), ..., t(\alpha)) \in Cycles_{af}(\mathcal{AF}) \tag{18.86}$$

From Equations (18.85) and (18.86) we prove Assertion 3.

Assertion 4:

$$e_i \in K \text{ s.t. } t(e_i) \neq s(e_i) \implies \left(\begin{array}{c} (\exists!(e_i, ..., t(e_i)) \in Paths_{af}(\mathcal{AF}) \text{ or } \exists!(e_i, ..., t(e_i)) \in Cycles_{af}(\mathcal{AF})) \\ \\ \text{and} \\ \\ \exists(s(e_i), ..., t(e_i)) \in Paths_{af}(\mathcal{AF}) \end{array} \right)$$

Let $\alpha \in K$ such that $t(\alpha) \neq s(\alpha)$.

- According to the definition of `Raf2Af` (Definition 76 on page 115, see rules Not_K, $And_{A,K}$, K'_2, K'_4 and K'_5), there is a walk in $\texttt{Raf2Af}(\mathcal{AF})$ from α to $t(\alpha)$ which corresponds to the sequence of arguments: $(\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha))$. If $t(\alpha) \neq \alpha$, then this walk is a path. Otherwise, it is a cycle. Furthermore, this path or cycle is the unique one going from α to $t(\alpha)$ as, by definition, an attack only has one target.

  We have thus:
$$\alpha \in K \text{ s.t. } t(\alpha) \neq \alpha \implies \exists!(\alpha, ..., t(\alpha)) \in Paths_{af}(\mathcal{AF}) \text{ or } \exists!(\alpha, ..., t(\alpha)) \in Cycles_{af}(\mathcal{AF}) \tag{18.87}$$

- Likewise (see rules Not_A, $And_{A,K}$, K_1', K_3' and K_5'), there is a walk in $\texttt{Raf2Af}(\mathcal{AF})$ from $s(\alpha)$ to $t(\alpha)$ which corresponds the sequence of arguments: $(s(\alpha), \neg s(\alpha), s(\alpha).\alpha, t(\alpha))$. Given that $t(\alpha) \neq s(\alpha)$, this walk is a path. Notice that it may not be the unique path going from $s(\alpha)$ to $t(\alpha)$. Indeed it could exists another attack whose source is $s(\alpha)$ and from which there exists a path to $t(\alpha)$.

  We have thus:
$$\alpha \in K \text{ s.t. } t(\alpha) \neq \alpha \implies \exists (s(\alpha),...,t(\alpha)) \in Paths_{af}(\mathcal{AF}) \tag{18.88}$$

From Equations (18.87) and (18.88) on the previous page and on this page we prove Assertion 4.

Assertion 5: $e_i \in K$ s.t. $t(e_i) \neq s(e_i)$ and $t(e_i) \neq e_i \iff \exists!(e_i,...,t(e_i)) \in Paths_{af}(\mathcal{AF})$ and $\exists(s(e_i),..., t(e_i)) \in Paths_{af}(\mathcal{AF})$

Trivially Assertion 5 is deduced from Assertions 3 and 4. ■

The following lemma establishes the elementary link between RAF-paths and paths in the AF version of a RAF:

Lemma 5. *Let $(e_1,...,e_n) \in Paths_{raf}(\mathcal{RAF})$.*

1. *For $i \in \{1,...,n-2\}$ (so $n \geq 3$), if $e_i \in A$ then $e_{i+1} \in K$ and then there is a path in $\texttt{Raf2Af}(\mathcal{AF})$ from e_i to e_{i+2}.*
2. *For $i \in \{1,...,n-1\}$ (so $n \geq 2$), if $e_i \in K$ then there is a unique path in $\texttt{Raf2Af}(\mathcal{AF})$ from e_i to e_{i+1}.*

Proof of Lemma 5.
Assertion 1: For $i \in \{1,...,n-2\}$, if $e_i \in A$ then $e_{i+1} \in K$ and then there is a path in $\texttt{Raf2Af}(\mathcal{AF})$ from e_i to e_{i+2}.

Given that $(e_1,...,e_n) \in Paths_{raf}(\mathcal{RAF})$ then $\forall(i,j) \in \{1,...,n\}^2$ s.t. $i \neq j, e_i \neq e_j$. Furthermore, we have: $e_i \in A \implies e_{i+1} \in K$. As $s(e_{i+1}) = e_i$, we have: $t(e_{i+1}) \neq s(e_{i+1})$. We also have: $t(e_{i+1}) \neq e_{i+1}$. According to Assertion 5 of Lemma 4 on page 255, we have then: $\exists!(e_{i+1},...,t(e_{i+1})) \in Paths_{af}(\mathcal{AF})$ and $\exists(s(e_{i+1}),..., t(e_{i+1})) \in Paths_{af}(\mathcal{AF})$. As $s(e_{i+1}) = e_i$ and $t(e_{i+1}) = e_{i+2}$, we have thus: $(e_i,...,e_{i+2}) \in Paths_{af}(\mathcal{AF})$.

Assertion 2: For $i \in \{1,...,n-1\}$, if $e_i \in K$ then there is a unique path in $\texttt{Raf2Af}(\mathcal{AF})$ from e_i to e_{i+1}.

Given that $(e_1,...,e_n) \in Paths_{raf}(\mathcal{RAF})$ then $\forall(i,j) \in \{1,...,n\}^2$ s.t. $i \neq j$, $e_i \neq e_j$. Furthermore, we have: $e_i \in K \implies t(e_i) = e_{i+1}$. As $t(e_i) \neq e_i$, we have following Assertion 3 of Lemma 4 on page 255: $\exists!(e_i,...,e_{i+1}) \in Paths_{af}(\mathcal{AF})$. ■

Following Lemma 5, Lemma 6 deepens the link between RAF-paths and paths in the AF version of a RAF:

Lemma 6. *Let $\{e_1,...,e_n\} \in Paths_{raf}(\mathcal{RAF})$ with $n > 1$. If $e_{n-1} \in K$ then there is a path in $\texttt{Raf2Af}(\mathcal{RAF})$ from e_1 to e_n.*

Proof of Lemma 6. If $e_{n-1} \in K$, then according to Lemma 5 there is a path in $\texttt{Raf2Af}(\mathcal{RAF})$ from e_{n-1} to e_n. Let prove this property for $n > 2$. Let $e_j \in A$ be the last argument of the RAF-path such that $j \leq n-2$. There are three cases to consider:

- **Case 1:** If there is no such e_j then for $i \in \{1,...,n-2\}$, all e_i are attacks. According to Lemma 5 (Item 2) there is thus a walk in $\texttt{Raf2Af}(\mathcal{RAF})$ from e_1 to e_n (*i.e.* the concatenation of the unique RAF-paths from each e_i to e_{i+1}, for $i = \{1,\ldots,n-1\}$). Furthermore, given that $\forall(l,k) \in \{1,...,n\}^2$ s.t. $l \neq k$, $e_l \neq e_k$, this walk is thus a path.

- **Case 2:** Else if $n = 3$ and $e_j = e_1$, then according to Lemma 5 on the previous page (Item 1) there is thus a path in $\texttt{Raf2Af}(\mathcal{RAF})$ from e_1 to e_3.
- **Case 3:** Else (*i.e.* $n > 3$ and there exists such e_j), then according to Lemma 5 on the previous page (Item 1) there is a path in $\texttt{Raf2Af}(\mathcal{RAF})$ from e_j to e_{j+2}. Furthermore given that for $i \in \{j+1, ..., n-1\}$, all e_i are attacks and given that $\forall (l,k) \in \{1, ..., n\}^2$ s.t. $l \neq k$, $e_l \neq e_k$, there is thus a path in $\texttt{Raf2Af}(\mathcal{RAF})$ from e_j to e_n.

If we are in Case 3, then $e_{j-1} \in K$ and according to Lemma 5 on the previous page (Item 2) there is a path in $\texttt{Raf2Af}(\mathcal{RAF})$ from e_{j-1} to e_j. By replacing n by j, by applying iteratively Case 1, 2 or 3, and by considering that $\forall (l,k) \in \{1, ..., n\}^2$ s.t. $l \neq k$, $e_l \neq e_k$, we end up with the conclusion that: if $e_{n-1} \in K$ then there is a path in $\texttt{Raf2Af}(\mathcal{RAF})$ from e_1 to e_n. ■

Note: We cannot apply this lemma for RAF-path of length 2 with an argument as first element (a, α) due to the condition "if $e_{n-1} \in K$".

Following Lemmas 5 and 6 on the previous page, Lemma 7 establishes the general relation between RAF-paths and paths in the AF version of a RAF with the following equivalence:

Lemma 7. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and let $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be its corresponding AF. The following property holds:*

$$\exists p' = (x, ..., y) \in Paths_{af}(\mathcal{AF}) \text{ s.t. } (x,y) \in (A \cup K)^2$$
$$\Longleftrightarrow$$
$$\exists p = (x = e_1, ..., y = e_n) \in Paths_{raf}(\mathcal{RAF}) \text{ s.t. } e_{n-1} \in K$$

Proof of Lemma 7. In two steps:

Step 1: Let prove that:

$$\exists p' = (x, ..., y) \in Paths_{af}(\mathcal{AF}) \text{ s.t. } (x,y) \in (A \cup K)^2$$
$$\Longrightarrow$$
$$\exists p = (x = e_1, ..., y = e_n) \in Paths_{raf}(\mathcal{RAF}) \text{ s.t. } e_{n-1} \in K$$

Let $p' = (x = o_1, ..., y = o_m) \in Paths_{af}(\mathcal{AF})$ s.t. $(x,y) \in (A \cup K)^2$. Let consider two cases :

- **Case 1:** If $o_1 \in A$ then according to the definition of $\texttt{Raf2Af}$, we have:
  - $o_2 \in Not_A$ and $o_2 = \neg o_1$
  - $o_3 \in And_{A,K}$ and $o_3 = o_1.\alpha$ with $\alpha \in K$ and $s(\alpha) = o_1$
  - $o_4 \in (A \cup K)$ and $o_4 = t(\alpha)$
  - $(\alpha, \neg\alpha) \in K'$
  - $(\neg\alpha, o_1.\alpha) \in K'$

  As a consequence, in $\mathcal{RAF}$, we have: $\alpha \in K$ s.t. $s(\alpha) = o_1$ and $t(\alpha) = o_4$.
- **Case 2:** Now, if $o_1 \in K$ then according to the definition of $\texttt{Raf2Af}$, we have:

- $o_2 \in Not_K$ and $o_2 = \neg o_1$
- $o_3 \in And_{A,K}$ and $o_3 = s(o_1).o_1$
- $o_4 \in (A \cup K)$ and $o_4 = t(o_1)$
- $(s(o_1), \neg s(o_1)) \in K'$
- $(\neg s(o_1), s(o_1).o_1) \in K'$

As a consequence, in $\mathcal{RAF}$, we have: $o_1 \in K$ s.t. $t(o_1) = o_4$.

In both cases $o_4 \in (A \cup K)$ and $o_1 \neq o_4$ and there is a RAF-path in $\mathcal{RAF}$ from o_1 to o_4, that is whether $\{o_1, \alpha, o_4\}$ or $\{o_1, o_4\}$. Let l be the length of the one or the other path, that is 3 in the first case and 2 in the second one. We have so $e_{l-1} = \alpha$ or $e_{l-1} = o_1$. In both cases we have: $e_{l-1} \in K$.

If $o_4 = e_n$ then the property holds: there exists a RAF-Path from x to y attacking y. That is, the before last element of this RAF-Path is an attack whose target is y (*i.e.* e_{l-1}).

Otherwise, by replacing o_1 by o_4 in the two cases studied above and by applying them iteratively until $o_4 = e_n$, we can obtain a well formed RAF-path[4] $p = \{x = e_1, ..., y = e_n\}$ from x to y. Furthermore, $e_{n-1} \in K$.

We prove so that:

$$\exists p' = (x, ..., y) \in Paths_{af}(\mathcal{AF}) \text{ s.t. } (x,y) \in (A \cup K)^2$$
$$\Longrightarrow$$
$$\exists p = (x = e_1, ..., y = e_n) \in Paths_{raf}(\mathcal{RAF}) \text{ s.t. } e_{n-1} \in K$$

Step 2: Let prove that:

$$\exists p = (x = e_1, ..., y = e_n) \in Paths_{raf}(\mathcal{RAF}) \text{ s.t. } e_{n-1} \in K$$
$$\Longrightarrow$$
$$\exists p' = (x, ..., y) \in Paths_{af}(\mathcal{AF}) \text{ s.t. } (x,y) \in (A \cup K)^2$$

Let $p = (x = e_1, ..., y = e_n) \in Paths_{raf}(\mathcal{RAF})$ s.t. $e_{n-1} \in K$. According to Lemma 6 on page 257: $\exists p' = (x, ..., y) \in Paths_{af}(\mathcal{AF})$. Furthermore, $(x,y) \in (A \cup K)^2$.

We prove from Step 1 and Step 2 that Lemma 7 on the previous page holds. ■

Lemma 8. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and $\mathcal{AF} =$ `Raf2Af`$(\mathcal{RAF})$ be the AF corresponding to $\mathcal{RAF}$. Let $S \in SCCS_{af}(\mathcal{AF})$ and $x \in A \cup K$. The following property holds:*

$$|S| > 1 \textit{ and } x \in S \iff |S| > 1 \textit{ and } \neg x \in S$$

Proof of Lemma 8. In two steps:

- **Step 1:** $|S| > 1$ and $x \in S \implies |S| > 1$ and $\neg x \in S$

  If $|S| > 1$ and $x \in S$ then there exists $x' \in S$ such that $x \neq x'$. There is thus a path from x to x' and a path from x' to x. It follows that there is a cycle from x to x. Let $c \in Cycles_{af}(\mathcal{AF})$ be such a cycle. According to the definition of `Raf2Af`, for any $x \in A \cup K$, the only argument attacked by x is $\neg x$. As a consequence, we have: $\neg x \in c$ and so: $\neg x \in S$.

[4]The RAF-path p is indeed well formed because at each iteration we obtain whether: $\{o_1, \alpha, o_4\}$ s.t. $o_1 \in A, \alpha \in K$ and $o_4 \in (A \cup K)$ or: $\{o_1, o_4\}$ s.t. $o_1 \in K$ and $o_4 \in (A \cup K)$. This implies so that there cannot be two consecutive arguments in p.

- **Step 2:** $|S| > 1$ and $\neg x \in S \implies |S| > 1$ and $x \in S$

  If $|S| > 1$ and $\neg x \in S$ then there exists $x' \in S$ such that $\neg x \neq x'$. There is thus a path from $\neg x$ to x' and a path from x' to $\neg x$. It follows that there is a cycle from $\neg x$ to $\neg x$. Let $c \in Cycles_{af}(\mathcal{AF})$ be such a cycle. According to the definition of `Raf2Af`, for any $\neg x \in Not_A \cup Not_K$, the only argument attacking $\neg x$ is x. As a consequence, we have: $x \in c$ and so: $x \in S$.

From Steps 1 and 2, we prove that: $|S| > 1$ and $x \in S \iff |S| > 1$ and $\neg x \in S$ ■

Lemma 9. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} =$ `Raf2Af`$(\mathcal{RAF})$ be the AF corresponding to $\mathcal{RAF}$. Let $S \in SCCS_{af}(\mathcal{AF})$ and $\alpha \in K$. The following property holds:*

$$s(\alpha) \in S \text{ and } t(\alpha) \in S \iff \neg s(\alpha) \in S \text{ and } s(\alpha).\alpha \in S \text{ and } \neg t(\alpha) \in S$$

Proof of Lemma 9. In two steps:

- **Step 1:** $s(\alpha) \in S$ and $t(\alpha) \in S \implies \neg s(\alpha) \in S$ and $s(\alpha).\alpha \in S$ and $\neg t(\alpha) \in S$

  If $s(\alpha) \in S$ and $t(\alpha) \in S$ then according to the definition of `Raf2Af`, there exists a path from $t(\alpha)$ to $s(\alpha)$ whose second element is $\neg t(\alpha)$ and a path from $s(\alpha)$ to $t(\alpha)$ which is $(s(\alpha), \neg s(\alpha), s(\alpha).\alpha, t(\alpha))$. Let $c \in Cycles_{af}(\mathcal{AF})$ be the cycle formed by merging both paths. Given that: $\neg s(\alpha) \in c$ and $s(\alpha).\alpha \in c$ and $\neg t(\alpha) \in c$, we also have: $\neg s(\alpha) \in S$ and $s(\alpha).\alpha \in S$ and $\neg t(\alpha) \in S$.

- **Step 2:** $\neg s(\alpha) \in S$ and $s(\alpha).\alpha \in S$ and $\neg t(\alpha) \in S \implies s(\alpha) \in S$ and $t(\alpha) \in S$

  Given that $|S| > 1$, then according to Lemma 8 on the previous page, we have:

  $$\neg t(\alpha) \in S \implies t(\alpha) \in S \text{ and } \neg s(\alpha) \in S \implies s(\alpha) \in S$$

From Steps 1 and 2, we prove that Lemma 9 holds. ■

Lemma 10. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} =$ `Raf2Af`$(\mathcal{RAF})$ be the AF corresponding to $\mathcal{RAF}$. Let $S \in SCCS_{af}(\mathcal{AF})$ and $x \in A \cup K$. The following properties holds:*

1. $|S| > 1$ *and* $x \in S \iff |S| > 1$ *and* $\exists \alpha \in K$ *s.t.* $t(\alpha) = x$ *and* $s(\alpha).\alpha \in S$
2. $|S| > 1$ *and* $x \in S \implies |S| > 1$ *and* $\exists \alpha \in K$ *s.t.* $t(\alpha) = x$ *and* $(s(\alpha) \in S$ *or* $\alpha \in S)$

Proof of Lemma 10.

Assertion 1: $|S| > 1$ and $x \in S \iff |S| > 1$ and $\exists \alpha \in K$ s.t. $t(\alpha) = x$ and $s(\alpha).\alpha \in S$

- **Step 1:** $|S| > 1$ and $x \in S \implies |S| > 1$ and $\exists \alpha \in K$ s.t. $t(\alpha) = x$ and $s(\alpha).\alpha \in S$

  If $|S| > 1$ and $x \in S$ then there exists $x' \in S$ such that $x \neq x'$. There is thus a path from x to x' and a path from x' to x. It follows that there is a cycle from x to x. Let $c \in Cycles_{af}(\mathcal{AF})$ be such a cycle. According to the definition of `Raf2Af`, x can only be attacked by arguments in $And_{A,K}$. There exists thus an attack $\alpha \in K$ such that $t(\alpha) = x$ and $s(\alpha).\alpha \in c$. As a consequence, we have: $s(\alpha).\alpha \in S$.

- **Step 2:** $|S| > 1$ and $\exists \alpha \in K$ s.t. $t(\alpha) = x$ and $s(\alpha).\alpha \in S \implies |S| > 1$ and $x \in S$

  If $|S| > 1$ and $s(\alpha).\alpha \in S$ then there exists $x' \in S$ such that $s(\alpha).\alpha \neq x'$. There is thus a path from $s(\alpha).\alpha$ to x' and a path from x' to $s(\alpha).\alpha$. It follows that there is a cycle from $s(\alpha).\alpha$ to $s(\alpha).\alpha$. Let $c \in Cycles_{af}(\mathcal{AF})$ be such a cycle. According to the definition of `Raf2Af`, the only argument attacked by $s(\alpha).\alpha$ is $t(\alpha) = x$. As a consequence, we have: $x \in S$.

Assertion 2: $|S| > 1$ and $x \in S \implies |S| > 1$ and $\exists \alpha \in K$ s.t. $t(\alpha) = x$ and $(s(\alpha) \in S$ or $\alpha \in S)$

According to the proof of Assertion 1, given that $|S| > 1$ and $x \in S$, $\exists \alpha \in K$ s.t. $t(\alpha) = x$ and there exists a cycle $c = (x, ..., x) \in Cycles_{af}(\mathcal{AF})$ such that $s(\alpha).\alpha \in c$. Given that $s(\alpha).\alpha$ is attacked by only two arguments: $\neg\alpha$ and $\neg s(\alpha)$, and so, defended by: α and $s(\alpha)$, we have:

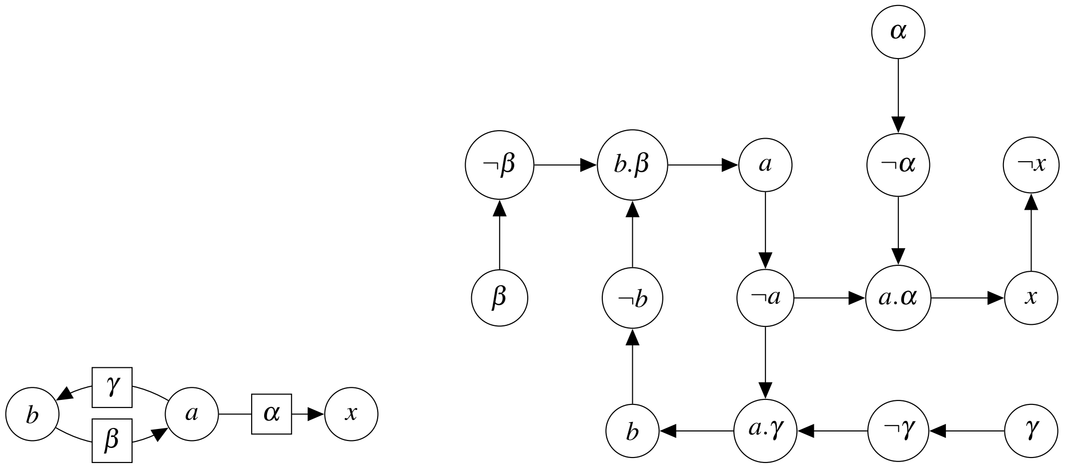
$$(s(\alpha) \text{ and } \neg s(\alpha) \in c) \text{ or } (\alpha \text{ and } \neg\alpha \in c)$$

We have so :

$$s(\alpha) \in S \text{ or } \alpha \in S \tag{18.89}$$

■

Note: Assertion 2 of Lemma 10 on the previous page is only an implication and not an equivalence. See Example 91 for a counter example.



(a) A RAF (b) Flattened version of the RAF in Figure 18.1(a)

Figure 18.1: Counter example illustration

Example 91. *Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be respectively the RAF and the AF illustrated in Figure 18.1. Let $S = \{a, \neg a, a.\gamma, b, \neg b, b.\beta\}$. We have: $S \in SCCS_{af}(\mathcal{AF})$, $|S| > 1$ and $\exists \alpha \in K$ s.t. $t(\alpha) = x$ and $(s(\alpha) \in S$ or $\alpha \in S)$ however $x \notin S$.*

***Proof of Proposition 35 on page 129*.** In two steps:

- **Step 1:**

$$\text{If } U \text{ is included is some } S \in SCCS_{raf}(\mathcal{RAF})$$
$$\text{then}$$
$$U \text{ is included in some } S' \in SCCS_{af}(\texttt{Raf2Af}(\mathcal{RAF}))$$

Let $S \in SCCS_{raf}(\mathcal{RAF})$ and $U \subseteq S$. If $|S| = 1$ then trivially $x \in S$ belongs to a unique SCC of $\texttt{Raf2Af}(\mathcal{RAF})$. Otherwise (*i.e.* if $|S| > 1$), according to the definition of SCC_{raf} (Definition 87 on page 125), for any couple $(x,y) \in S^2$ s.t. $x \neq y$ there is a RAF-path from x to y in which y is attacked and a RAF-path from y to x in which x is attacked.

Following Lemma 6 on page 257, there is thus a path from x to y and a path from y to x in $\texttt{Raf2Af}(\mathcal{RAF})$. As a consequence x and y are in the same SCC of $\texttt{Raf2Af}(\mathcal{RAF})$. It follows that any element of S belongs to the same SCC.

We prove so that if U is included is some $S \in SCCS_{raf}(\mathcal{RAF})$ then U is included in some $S' \in SCCS_{af}(\texttt{Raf2Af}(\mathcal{RAF}))$.

- **Step 2:**

$$\text{If } U \text{ is included is some } S' \in SCCS_{af}(\texttt{Raf2Af}(\mathcal{RAF}))$$
$$\text{then}$$
$$U \text{ is included in some } S \in SCCS_{raf}(\mathcal{RAF})$$

Let $S' \in SCCS_{af}(\texttt{Raf2Af}(\mathcal{RAF}))$ and $U \subseteq S'$. If $|S'| = 1$ then trivially $x \in S'$ belongs to an unique SCC_{raf} of $\mathcal{RAF}$. Otherwise (*i.e.* if $|S'| > 1$), according to the definition of SCC_{af}, for any couple $(x,y) \in S'^2$ s.t. $x \neq y$ there is a path from x to y and a path from y to x. It is particularly the case for any couple $(x,y) \in U^2$.

Following Lemma 7 on page 258, there is thus a RAF-path: $(x = e_1, ..., y = e_n) \in Paths_{raf}(\mathcal{RAF})$ s.t. $e_{n-1} \in K$ and another one: $(y = o_1, ..., x = o_m) \in Paths_{raf}(\mathcal{RAF})$ s.t. $o_{m-1} \in K$. As a consequence: $(x,y) \in PE_{raf}$.

We prove so that $U \subseteq S$ with S being some SCC_{raf} of $\mathcal{RAF}$.

We prove from Case 1 and Case 2 that Proposition 35 on page 129 holds. ■

Proof of Proposition 36 on page 129. Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF, $S \subseteq A \cup K$ be a subset of elements of $\mathcal{RAF}$ and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be the corresponding AF of $\mathcal{RAF}$.

Step 1: Let prove that:

$$S \in SCCS_{raf}(\mathcal{RAF})$$
$$\Longrightarrow$$
$$\left(\begin{array}{l} S \ \cup \\ \{\neg a \in Not_A | a \in S \text{ and } (|S| > 1 \text{ or } (\exists \alpha \in K \text{ s.t. } s(\alpha) = a \text{ and } t(\alpha) = a))\} \ \cup \\ \{\neg \alpha \in Not_K | \alpha \in S \text{ and } (|S| > 1 \text{ or } t(\alpha) = \alpha)\} \ \cup \\ \{s(\alpha).\alpha \in And_{A,K} | \alpha \in S \text{ and } (|S| > 1 \text{ or } t(\alpha) = \alpha)\} \ \cup \\ \{s(\alpha).\alpha \in And_{A,K} | s(\alpha) \in S \text{ and } t(\alpha) \in S\} \end{array} \right) \in SCCS_{af}(\texttt{Raf2Af}(\mathcal{RAF}))$$

If $S \in SCCS_{raf}(\mathcal{RAF})$ then, according to Proposition 35 on page 129, S is included in some $S' \in SCCS_{af}(\mathcal{AF})$. If $S' \in SCCS_{af}(\mathcal{AF})$ then, according to Proposition 35 on page 129, $S' \cap (A \cup K) \subseteq S$. We have so: $S' \setminus S \subseteq (Not_A \cup Not_K \cup And_{A,K})$.

Let $S \in SCCS_{raf}(\mathcal{RAF})$ and let $S' \in SCCS_{af}(\mathcal{AF})$ be the SCC such that $S \subseteq S'$ and $S' \setminus S \subseteq (Not_A \cup Not_K \cup And_{A,K})$.

1. Let consider the case where $|S| = 1$. Let $x \in S$.

   - Let $x \in A$.

     – Let suppose that there exists a RAF-closed-walk $p = (x, ..., e_{n-1}, x) \in ClosedWalk_{raf}(\mathcal{RAF})$. Given that $x \in A$ then we have: $e_{n-1} \in K$ and $t(e_{n-1}) = x$, $e_2 \in K$ and $s(e_2) = x$. If the length of p is greater than three this means that there exists $e_i \in p$ for some $i \in [\![3, n-1]\!]$ such that $e_{i-1} \in K$ and so $t(e_{i-1}) = e_i$. Given that p is a RAF-closed-walk and that both x and e_i are attacked in p then following Proposition 34 on page 126, we have: $e_i \in S$, which contradicts: $|S| = 1$. As a consequence, if such RAF-closed-walk exists it must be the case that its length equals three, as $x \in A$. We have so : $p = (x, \alpha, x)$. There exists thus $\alpha \in K$ such that $s(\alpha) = x$ and $t(\alpha) = x$. As $s(\alpha) \in S$ and $t(\alpha) \in S$, we have according to Lemma 9 on page 260: $\neg x \in S'$ and $x.\alpha \in S'$. Furthermore, given that $|S| = 1$, there is no $y \in A \cup K$ such that $x \neq y$ and $x \underset{\mathcal{RAF}}{\equiv} y$. As a consequence, we have: $S' = \{x, \neg x, x.\alpha\}$. We finally have:

$$S' = \left(\begin{array}{l} S \ \cup \\ \\ \{\neg a \in Not_A | a \in S \text{ and } (|S| > 1 \text{ or } (\exists \alpha \in K \text{ s.t. } s(\alpha) = a \text{ and } t(\alpha) = a))\} \ \cup \\ \\ \{\neg \alpha \in Not_K | \alpha \in S \text{ and } (|S| > 1 \text{ or } t(\alpha) = \alpha)\} \ \cup \\ \\ \{s(\alpha).\alpha \in And_{A,K} | \alpha \in S \text{ and } (|S| > 1 \text{ or } t(\alpha) = \alpha)\} \ \cup \\ \\ \{s(\alpha).\alpha \in And_{A,K} | s(\alpha) \in S \text{ and } t(\alpha) \in S\} \end{array}\right) \tag{18.90}$$

     – If there is no RAF-closed-walk $p \in ClosedWalk_{raf}(\mathcal{RAF})$ from x to x then, there is no attack $\alpha \in K$ such that $s(\alpha) = x$ and $t(\alpha) = x$. We have thus: $S' = S = \{x\}$ and Equation (18.90) holds.

   - Let $x \in K$.

     – Let suppose that there exists a RAF-closed-walk $p = (x, ..., e_{n-1}, x) \in ClosedWalk_{raf}(\mathcal{RAF})$ from x to x such that $e_{n-1} \in K$. As x in K, e_2 and x are attacked by p. As a consequence, we have $x \underset{\mathcal{RAF}}{\equiv} e_2$. This equivalence contradicts $|S| = 1$, except if $n = 2$ and $x = e_2$. As a consequence, we have: $p = (x, x)$. We have so: $t(x) = x$. According to the definition of `Raf2Af`, there exists a cycle in $\mathcal{AF}$ from x to x which is $(x, \neg x, s(x).x, x)$. We have thus: $\neg x \in S'$ and $s(x).x \in S'$. As there is no $y \in A \cup K$ such that $x \neq y$ and $x \underset{\mathcal{RAF}}{\equiv} y$ (*i.e.* $|S| = 1$) and as $x \in S$ and $t(x) = x$, we finally have: $S' = \{x, \neg x, s(x).x\}$ and Equation (18.90) holds.

     – If there is no RAF-closed-walk $p \in ClosedWalk_{raf}(\mathcal{RAF})$ from x to x then: $t(x) \neq x$. We have thus: $S' = S = \{x\}$ and Equation (18.90) holds.

2. Let consider the case where $|S| > 1$.

   - Let $a \in A$ be an argument in $\mathcal{RAF}$.

Given that $S \subseteq S'$, we have, according Lemma 8 on page 259, the following property:

$$|S| > 1 \text{ and } a \in S \iff |S| > 1 \text{ and } \neg a \in S' \tag{18.91}$$

- Let $\alpha \in K$ be an attack in $\mathcal{RAF}$. We have to cases to consider: $\alpha \in S$ and $\alpha \notin S$.
  - Let $\alpha \in S$. As $|S| > 1$, then α is equivalent to another element of $\mathcal{RAF}$ *w.r.t.* PE_{raf}, which is also in S. There is thus, following Proposition 33 on page 126, a RAF-closed-walk $c = (e_1 = \alpha, ..., e_n = \alpha) \in ClosedWalk_{raf}(\mathcal{RAF})$ such that $n > 2$. Following Proposition 34 on page 126, all elements of $U = \{e_i \in c | i \in \{2, ..., n\}$ s.t. $e_{i-1} \in K\}$ belong to the same SCC_{raf}. We have thus: $e_2 = t(\alpha) \in S$. Given that $\alpha \in S$ and $t(\alpha) \in S$, we also have: $\alpha \in S'$ and $t(\alpha) \in S'$. Furthermore, according to the definition of `Raf2Af`, we have: $(\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha)) \in Walks_{af}(\mathcal{AF})$. Notice that if $t(\alpha) = \alpha$ then it must be the case that: $c = (\alpha, \alpha)$ according to the definition of RAF-closed-walk. This contradicts $n > 2$. As a consequence, we have: $t(\alpha) \neq \alpha$ and so: $(\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha)) \in Paths_{af}(\mathcal{AF})$. α and $t(\alpha)$ being in the same SCC_{af} of $\mathcal{AF}$, there is also a path from $t(\alpha)$ to α in $\mathcal{AF}$. Given that there is a path from $t(\alpha)$ to α in $\mathcal{AF}$ and that $(\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha)) \in Paths_{af}(\mathcal{AF})$, we have: $\neg\alpha$, $s(\alpha).\alpha$ and α being equivalent *w.r.t.* PE_{af}. As a consequence we have: $\neg\alpha \in S'$ and $s(\alpha).\alpha \in S'$. The following property holds then:

$$|S| > 1 \text{ and } \alpha \in S \cap K \implies \neg\alpha \in S' \text{ and } s(\alpha).\alpha \in S' \tag{18.92}$$

  - Let $\alpha \notin S$. Let consider two cases: ($s(\alpha) \in S$ and $t(\alpha) \in S$), ($s(\alpha) \notin S$ or $t(\alpha) \notin S$).
    - ∗ Let suppose that $s(\alpha) \in S$ and $t(\alpha) \in S$. We have thus $s(\alpha) \in S'$ and $t(\alpha) \in S'$. Following to Lemma 9 on page 260, the following property then holds:

$$\alpha \in K \setminus S \text{ s.t. } s(\alpha) \in S \text{ and } t(\alpha) \in S \implies \neg s(\alpha) \in S' \text{ and } s(\alpha).\alpha \in S' \tag{18.93}$$

    - ∗ Let suppose that $s(\alpha) \notin S$ or $t(\alpha) \notin S$. We have thus: $s(\alpha) \notin S'$ or $t(\alpha) \notin S'$.
Let x be whether $s(\alpha)$ or $t(\alpha)$ and let suppose that $x \notin S'$. As $|S| > 1$ and $S \subseteq S'$ then we have: $|S'| > 1$. As $x \notin S'$, we also have according to Lemma 8 on page 259: $\neg x \notin S'$. As consequence, the two following properties hold:

$$|S| > 1 \text{ and } \alpha \in K \setminus S \text{ s.t. } s(\alpha) \notin S \implies \neg s(\alpha) \notin S' \tag{18.94}$$

$$|S| > 1 \text{ and } \alpha \in K \setminus S \text{ s.t. } t(\alpha) \notin S \implies \neg t(\alpha) \notin S' \tag{18.95}$$

Let consider two cases: $s(\alpha) \notin S'$ and $t(\alpha) \notin S'$.

      · Let $t(\alpha) \notin S'$. According to Lemma 10 on page 260 (Assertion 1), we have:

$$|S'| > 1 \text{ and } t(\alpha) \in S' \iff |S'| > 1 \text{ and } \exists \beta \in K \text{ s.t. } t(\beta) = t(\alpha) \text{ and } s(\beta).\beta \in S'$$

The negation of this equivalence is the following:

$$|S'| \leq 1 \text{ or } t(\alpha) \notin S' \iff |S'| \leq 1 \text{ or } (\nexists \beta \in K \text{ s.t. } t(\beta) = t(\alpha) \text{ and } s(\beta).\beta \in S')$$

Given that $|S'| > 1$, we have:

$$t(\alpha) \notin S' \iff \nexists \beta \in K \text{ s.t. } t(\beta) = t(\alpha) \text{ and } s(\beta).\beta \in S'$$

Given that $t(\alpha) \notin S'$, we have thus:

$$\nexists \beta \in K \text{ s.t. } t(\beta) = t(\alpha) \text{ and } s(\beta).\beta \in S'$$

That particularly holds for $\beta = \alpha$ and so, as a consequence, we have:

$$s(\alpha).\alpha \notin S'$$

The following property holds then:

$$|S| > 1 \text{ and } \alpha \notin S \text{ and } t(\alpha) \notin S \implies s(\alpha).\alpha \notin S' \tag{18.96}$$

· Let $s(\alpha) \notin S'$. We have: $|S'| > 1$, $\alpha \notin S'$ and $s(\alpha) \notin S'$. Given that the case where $t(\alpha) \notin S$ has already been treated with Equation (18.96), let consider the case where $t(\alpha) \in S$ and so $t(\alpha) \in S'$.
If $t(\alpha)$ is not in the same SCC_{af} as $s(\alpha)$, then there is no cycle $c \in Cycles_{af}(\mathcal{AF})$ going from $t(\alpha)$ to $t(\alpha)$ such that $s(\alpha) \in c$. Likewise, if $t(\alpha)$ is not in the same SCC_{af} as α means that there is no cycle $c \in Cycles_{af}(\mathcal{AF})$ going from $t(\alpha)$ to $t(\alpha)$ such that $\alpha \in c$.
Following the definition of `Raf2Af`, there exist a walk from $s(\alpha)$ to $t(\alpha)$ which is $(s(\alpha), \neg s(\alpha), s(\alpha).\alpha,\ t(\alpha))$ and a walk from α to $t(\alpha)$ which is $(\alpha, \neg\alpha, s(\alpha).\alpha,$ $t(\alpha))$.
As a consequence, there is no cycle $c = (t(\alpha), ..., t(\alpha)) \in Cycles_{af}(\mathcal{AF})$ such that $s(\alpha).\alpha \in c$. We have thus: $s(\alpha).\alpha \notin S'$.
The following property holds then:

$$|S| > 1 \text{ and } \alpha \notin S \text{ and } s(\alpha) \notin S \implies s(\alpha).\alpha \notin S' \tag{18.97}$$

From Equations (18.96) and (18.97), we have thus:

$$|S| > 1 \text{ and } \alpha \in K \setminus S \text{ and } (s(\alpha) \notin S \text{ or } t(\alpha) \notin S) \implies s(\alpha).\alpha \notin S' \tag{18.98}$$

From Proposition 35 and Equations (18.91) to (18.95) and (18.98) on page 129 and on pages 264–265, we prove to that if $|S| > 1$ then Equation (18.90) on page 263 holds.

From Cases 1 and 2, we prove so that:

$$S \in SCCS_{raf}(\mathcal{RAF})$$
$$\Longrightarrow$$
$$\left(\begin{array}{l} S \ \cup \\ \{\neg a \in Not_A | a \in S \text{ and } (|S| > 1 \text{ or } (\exists \alpha \in K \text{ s.t. } s(\alpha) = a \text{ and } t(\alpha) = a))\} \ \cup \\ \{\neg\alpha \in Not_K | \alpha \in S \text{ and } (|S| > 1 \text{ or } t(\alpha) = \alpha)\} \ \cup \\ \{s(\alpha).\alpha \in And_{A,K} | \alpha \in S \text{ and } (|S| > 1 \text{ or } t(\alpha) = \alpha)\} \ \cup \\ \{s(\alpha).\alpha \in And_{A,K} | s(\alpha) \in S \text{ and } t(\alpha) \in S\} \end{array}\right) \in SCCS_{af}(\texttt{Raf2Af}(\mathcal{RAF}))$$

Step 2: Let $S' = \left(\begin{array}{l} S \ \cup \\ \{\neg a \in Not_A | a \in S \text{ and } (|S| > 1 \text{ or } (\exists \alpha \in K \text{ s.t. } s(\alpha) = a \text{ and } t(\alpha) = a))\} \ \cup \\ \{\neg \alpha \in Not_K | \alpha \in S \text{ and } (|S| > 1 \text{ or } t(\alpha) = \alpha)\} \ \cup \\ \{s(\alpha).\alpha \in And_{A,K} | \alpha \in S \text{ and } (|S| > 1 \text{ or } t(\alpha) = \alpha)\} \ \cup \\ \{s(\alpha).\alpha \in And_{A,K} | s(\alpha) \in S \text{ and } t(\alpha) \in S\} \end{array} \right)$ and let prove that:

$$S' \in SCCS_{af}(\texttt{Raf2Af}(\mathcal{RAF})) \implies S \in SCCS_{raf}(\mathcal{RAF})$$

If $S' \in SCCS_{af}(\mathcal{AF})$ then, according to Proposition 35 on page 129, $S' \cap (A \cup K)$ is included in the same SCC_{raf}. We have thus: $S' \cap (A \cup K) \subseteq V \in SCCS_{raf}(\mathcal{RAF})$. Let suppose that there exists an element x such that $x \in V \setminus (S' \cap (A \cup K))$. As $V \in SCCS_{raf}(\mathcal{RAF})$ then according to Proposition 35 on page 129, V is included in some SCC_{af} of $\mathcal{AF}$. As $\exists z \in V$ s.t. $z \in S'$ then the SCC_{af} in which V is included is S'. We have thus: $x \in S'$. Furthermore given that $S' \cap (A \cup K) \subseteq V$, we have so: $x \in S' \cap (Not_A \cup Not_K \cup And_{A,K})$. Given that $x \in V$ then we also have: $x \in (A \cup K)$. Contradiction.

As a consequence such an element x does not exist. We have so: $S = V$.

We prove so that:

$$S' \in SCCS_{af}(\texttt{Raf2Af}(\mathcal{RAF})) \implies S \in SCCS_{raf}(\mathcal{RAF})$$

From Steps 1 and 2, we prove that Proposition 36 on page 129 holds. ■

Proofs of Section 14.2: RAF hierarchy

Proof of Proposition 37 on page 132.

1. By definition, $\preccurlyeq$ is reflexive.
2. Let prove that $\preccurlyeq$ is antisymmetric.

   Let suppose that $S \neq S'$, $S \preccurlyeq S'$ and $S' \preccurlyeq S$. If $S \preccurlyeq S'$ then, as $S \neq S'$, we have: $\exists (e_1, ..., e_{n-1}, e_n) \in Paths_{raf}(\mathcal{RAF})$ s.t. $e_1 \in S$ and $e_n \in S'$ and $e_{n-1} \in K$. Likewise, if $S' \preccurlyeq S$ then, as $S \neq S'$, we have: $\exists (o_1, ..., o_{m-1}, o_m) \in Paths_{raf}(\mathcal{RAF})$ s.t. $o_1 \in S'$ and $o_m \in S$ and $o_{m-1} \in K$. As all elements of S (resp. S') are equivalent *w.r.t.* PE_{raf}, we have as a consequence: $\forall x \in S, \forall x' \in S'$, $(x, x') \in PE_{raf}(\mathcal{RAF})$. Given that S and S' are SCC_{raf} and thus are equivalence class of elements under the relation PE_{raf}, then we have: $S = S'$ which is a contradiction.

   We prove so that:

   $$(S \preccurlyeq S' \text{ and } S' \preccurlyeq S) \implies S = S'$$

3. Let prove that $\preccurlyeq$ is transitive.

   Let S_1, S_2, S_3 be three SCC_{raf} and let $S_1 \preccurlyeq S_2$ and $S_2 \preccurlyeq S_3$. Let prove that $S_1 \preccurlyeq S_3$.

   Trivially, if $S_1 = S_2$ or $S_2 = S_3$ then $S_1 \preccurlyeq S_3$. Let consider the case where $S_1 \neq S_2$ and $S_2 \neq S_3$.

   If $S_1 \preccurlyeq S_2$ then, as $S_1 \neq S_2$, we have: $\exists (e_1, ..., e_{n-1}, e_n) \in Paths_{raf}(\mathcal{RAF})$ s.t. $e_1 \in S_1$ and $e_n \in S_2$ and $e_{n-1} \in K$. Likewise, if $S_2 \preccurlyeq S_3$ then, as $S_2 \neq S_3$, we have: $\exists (o_1, ..., o_{m-1}, o_m) \in Paths_{raf}(\mathcal{RAF})$ s.t. $o_1 \in S_2$ and $o_m \in S_3$ and $o_{m-1} \in K$.

- If $e_n = o_1$ then we have: $(e_1,...,e_{n-1},e_n = o_1,...,o_{m-1},o_m) \in Paths_{raf}(\mathcal{RAF})$. As $e_1 \in S_1$ and $o_m \in S_3$, we have: $S_1 \preccurlyeq S_3$.
- Otherwise, if $e_n \neq o_1$ then we have: $|S_2| > 1$. Given that $(e_n,o_1) \in PE_{raf}(\mathcal{RAF})$ then: $\exists(e_n = x_1,...,x_{p-1},x_p = o_1) \in Paths_{raf}(\mathcal{RAF})$ s.t. $x_{p-1} \in K$ and $x_p = o_1$. As a consequence: $(e_1,...,e_{n-1}, e_n = x_1,...,x_{p-1},x_p = o_1,...,o_{m-1},o_m) \in Paths_{raf}(\mathcal{RAF})$. We have thus: $S_1 \preccurlyeq S_3$.

We prove so that:

$$(S_1 \preccurlyeq S_2 \text{ and } S_2 \preccurlyeq S_3) \implies S_1 \preccurlyeq S_3$$

From Steps 1, 2 and 3, we prove so that $\preccurlyeq$ is a partial order. ■

Proof of Proposition 38 on page 132. Let $p = (e_1,...,e_{n-1},e_n) \in Paths_{raf}(\mathcal{RAF})$ such that $e_1 \in S$ and $e_n \in S'$ and $e_{n-1} \in K$. Let suppose that there exists $e_i \in p \cap K$ such that $t(e_i) \notin (S \cup S')$. If $t(e_i) \notin (S \cup S')$, we have thus $t(e_i) \neq e_n$. Let $V \in SCCS_{raf}(\mathcal{RAF})$ be the SCC_{raf} such that $t(e_i) \in V$. Given that $(e_1,...,e_i,e_{i+1} = t(e_i)) \in Paths_{raf}(\mathcal{RAF})$, $e_1 \in S$ and $t(e_i) \in V$, we have then: $S \preccurlyeq V$. Furthermore, given that $t(e_i) \neq e_n$ we have: $(e_{i+1} = t(e_i),...,e_n) \in Paths_{raf}(\mathcal{RAF})$. As $t(e_i) \in V$, $e_n \in S'$, $e_{n-1} \in K$ then we have: $V \preccurlyeq S'$. Given that $S \preccurlyeq V$ and $V \preccurlyeq S'$ then S is not a predecessor of S', which is a contradiction. We prove so that:

$$\forall i \in \{1,...,n\}, e_i \in p \cap K \implies t(e_i) \in (S \cup S')$$

■

Proof of Proposition 39 on page 133. Given that the notion of predecessor, as defined, proceed from the relation $\preccurlyeq$, which is a partial order, then trivially $Dag_{scc}(\mathcal{RAF})$ is acyclic. ■

Proofs of Section 14.3: RAF semantics decomposability

Lemma 11. *Let $\mathcal{RAF} = \langle A,K,s,t\rangle$ be a RAF and $\mathcal{AF} = \mathtt{Raf2Af}(\mathcal{RAF})$ be the corresponding AF of $\mathcal{RAF}$ (with $\mathcal{AF} = \langle A',K'\rangle$). Let Ω be a partition of $(A \cup K)$ and $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{AF})$ be the RAF-compliant partition of A' corresponding to Ω, i.e. $\Omega' = \{\omega' = \omega \cup \{\neg x | x \in \omega\} \cup \{s(\alpha).\alpha \in And_{A,K} | \alpha \in \omega\} | \omega \in \Omega\}$. Let $\omega \in \Omega$ and $\omega' \in \Omega'$ be its counterpart in $\mathcal{AF}$. Let $\widetilde{\mathcal{RAF}} = \langle \tilde{A},\tilde{K},\tilde{s},\tilde{t},s,t\rangle$ be the partial RAF corresponding to ω. Let $\mathfrak{I} = \langle S^{inp},Q^{inp}\rangle$ be the input elements of $\widetilde{\mathcal{RAF}}$ and $\mathfrak{L}^{inp}$ be a structure labelling of them. Let $\left\langle \widetilde{\mathcal{RAF}},\mathfrak{I},\mathfrak{L}^{inp}\right\rangle$ be a RAF with input and $\left\langle \mathcal{AF}\downarrow_{\omega'},\mathcal{I},\ell^{\mathcal{I}},K_{\mathcal{I}}\right\rangle$ be its corresponding AF with input, as defined in Definition 105 on page 155. Let $\widetilde{\mathcal{RAF}}_s = \langle \tilde{A}_s,\tilde{K}_s,s_s,t_s\rangle$ be the standard of $\left\langle \widetilde{\mathcal{RAF}},\mathfrak{I},\mathfrak{L}^{inp}\right\rangle$ and let std-$\mathcal{AF} = \langle A'_s,K'_s\rangle$ be the standard AF corresponding to $\left\langle \mathcal{AF}\downarrow_{\omega'},\mathcal{I},\ell^{\mathcal{I}},K_{\mathcal{I}}\right\rangle$. Let $\widetilde{\mathcal{RAF}}'_s = \mathtt{Raf2Af}(\widetilde{\mathcal{RAF}}_s)$ be the corresponding AF of $\widetilde{\mathcal{RAF}}_s$ (with $\widetilde{\mathcal{RAF}}'_s = \langle \tilde{A}'_s,\tilde{K}'_s\rangle$). The following assertions hold:*

1. *$\mathcal{I} \subseteq (Not_A \cup And_{A,K})$*
2. *$(\neg s(\alpha),s(\alpha).\alpha) \in \tilde{K}'_s \cup K'_s$ s.t. $s(\alpha).\alpha \in \omega'$ and $\neg s(\alpha) \notin \omega' \implies s(\alpha) \in S^{inp}$*
3. *$(s(\alpha).\alpha,t(\alpha)) \in \tilde{K}'_s \cup K'_s$ s.t. $s(\alpha).\alpha \notin \omega'$ and $t(\alpha) \in \omega' \implies \alpha \in Q^{inp}$*
4. *$(x,y) \in \tilde{K}'_s \cup K'_s$ s.t. $y \in \omega' \implies x \in \omega' \cup \mathcal{I}$*

`Note:` *In order to follow easily the following proofs, here is a reminder of some set relations:*

- *Considering the definition of* $\widetilde{\mathcal{RAF}} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t \rangle$, $\widetilde{\mathcal{RAF}}_s = \langle \tilde{A}_s, \tilde{K}_s, s_s, t_s \rangle$, $\widetilde{\mathcal{RAF}}'_s = \langle \tilde{A}'_s, \tilde{K}'_s \rangle$ *and the definition of* `Raf2Af`*, we have:*
  - $\tilde{A} = A \cap \omega = \tilde{A}_s \cap \omega$
  - $\tilde{K} = K \cap \omega = \tilde{K}_s \cap \omega$
  - $\tilde{A}_s \cup \tilde{K}_s \cup Not_{\tilde{A}_s} \cup Not_{\tilde{K}_s} \cup And_{\tilde{A}_s,\tilde{K}_s} = \tilde{A}'_s$
- *Considering the definitions of* $\mathcal{AF} = \langle A', K' \rangle$, $\left\langle \mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$, $std\text{-}\mathcal{AF} = \langle A'_s, K'_s \rangle$ *and the definition of* `Raf2Af`*, we have:*
  - $A \cup K \cup Not_A \cup Not_K \cup And_{A,K} = A'$
  - $\omega' = A' \cap \omega' = A'_s \cap \omega'$

***Proof of Lemma 11 on the previous page*.**
Assertion 1: $\mathcal{I} \subseteq (Not_A \cup And_{A,K})$
Trivial considering Definition 105 on page 155.

Assertion 2: $(\neg s(\alpha), s(\alpha).\alpha) \in \tilde{K}'_s \cup K'_s$ s.t. $s(\alpha).\alpha \in \omega'$ and $\neg s(\alpha) \notin \omega' \implies s(\alpha) \in S^{inp}$
Let consider two cases: $(\neg s(\alpha), s(\alpha).\alpha) \in \tilde{K}'_s$ and $(\neg s(\alpha), s(\alpha).\alpha) \in K'_s$.

- **Case 1:** Let consider $\widetilde{\mathcal{RAF}}'_s$ and let $(\neg s(\alpha), s(\alpha).\alpha) \in \tilde{K}'_s$ s.t. $s(\alpha).\alpha \in \omega'$ and $\neg s(\alpha) \notin \omega'$.

  Given that $s(\alpha).\alpha \in \tilde{A}'$, we have following the definition of `Raf2Af`: $\alpha \in \tilde{K}_s$. α is an attack of the standard RAF $\widetilde{\mathcal{RAF}}'_s$. Furthermore, given that $s(\alpha).\alpha \in \omega'$, that $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{RAF})$ and that $\omega' \in \Omega'$, we have so: $\alpha \in \omega'$ and $\neg\alpha \in \omega'$. As $\alpha \in \tilde{K}_s \cap \omega'$, then we have: $\alpha \in \tilde{K}$. α is thus an element of $\widetilde{\mathcal{RAF}}$. Furthermore, as $\neg s(\alpha) \notin \omega'$, we have: $s(\alpha) \notin \omega'$. As a consequence, $s(\alpha)$ is not an element of $\widetilde{\mathcal{RAF}}$. Given that $s(\alpha) \notin \widetilde{\mathcal{RAF}}$ and $\alpha \in \widetilde{\mathcal{RAF}}$, we have so: $s(\alpha) \in S^{inp}$.
- **Case 2:** Let consider $std\text{-}\mathcal{AF}$ and let $(\neg s(\alpha), s(\alpha).\alpha) \in K'_s$ s.t. $s(\alpha).\alpha \in \omega'$ and $\neg s(\alpha) \notin \omega'$.

  Given that $s(\alpha).\alpha \in A'_s$, we have following the definition of `Raf2Af`: $\alpha \in K$. α is an attack of the original RAF $\mathcal{RAF}$. Given that $s(\alpha).\alpha \in \omega'$, that $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{RAF})$ and that $\omega' \in \Omega'$, we have so: $\alpha \in \omega'$ and $\neg\alpha \in \omega'$. As $\alpha \in K \cap \omega'$, we have: $\alpha \in \widetilde{\mathcal{RAF}}$. Furthermore, as $\neg s(\alpha) \notin \omega'$, we have: $s(\alpha) \notin \omega'$. As a consequence, $s(\alpha)$ is not an element of $\widetilde{\mathcal{RAF}}$. Given that $s(\alpha) \notin \widetilde{\mathcal{RAF}}$ and $\alpha \in \widetilde{\mathcal{RAF}}$, we have so: $s(\alpha) \in S^{inp}$.

From Cases 1 and 2, we prove so that Assertion 2 holds.

Assertion 3: $(s(\alpha).\alpha, t(\alpha)) \in \tilde{K}'_s \cup K'_s$ s.t. $s(\alpha).\alpha \notin \omega'$ and $t(\alpha) \in \omega' \implies \alpha \in Q^{inp}$

- **Case 1:** Let consider $\widetilde{\mathcal{RAF}}'_s$ and let $(s(\alpha).\alpha, t(\alpha)) \in \tilde{K}'_s$ s.t. $s(\alpha).\alpha \notin \omega'$ and $t(\alpha) \in \omega'$.

  Given that $s(\alpha).\alpha \in \tilde{A}'$, we have following the definition of `Raf2Af`: $\alpha \in \tilde{K}_s$. α is an attack of the standard RAF $\widetilde{\mathcal{RAF}}'_s$. Given that $t(\alpha) \in \omega'$ and that $\alpha \in \tilde{K}_s$, we have so: $t(\alpha) \in \tilde{A}_s \cup \tilde{K}_s$. As a consequence and as $t(\alpha) \in \omega'$, we have: $t(\alpha) \in \tilde{A} \cup \tilde{K}$. Given that $s(\alpha).\alpha \notin \omega'$, that $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{RAF})$ and that $\omega' \in \Omega'$, we have so: $\alpha \notin \omega'$ and $\neg\alpha \notin \omega'$. As a consequence, we have $\alpha \notin \tilde{K}$. As $t(\alpha)$ belongs to $\widetilde{\mathcal{RAF}}$ and α does not belong to $\widetilde{\mathcal{RAF}}$, We have: $\alpha \in Q^{inp}$.

- **Case 2:** Let consider $std\text{-}\mathcal{AF}$ and let $(s(\alpha).\alpha, t(\alpha)) \in K'_s$ s.t. $s(\alpha).\alpha \notin \omega'$ and $t(\alpha) \in \omega'$.

  Given that $s(\alpha).\alpha \in A'_s$, we have following the definition of `Raf2Af`: $\alpha \in K$. α is an attack of the original RAF $\mathcal{RAF}$. As a consequence, we have: $t(\alpha) \in A \cup K$. Given that $t(\alpha) \in \omega'$ and that $t(\alpha) \in A \cup K$, we have: $t(\alpha) \in \tilde{A} \cup \tilde{K}$. Given that $s(\alpha).\alpha \notin \omega'$, that $\Omega' \in \mathcal{S}_{D\text{-}raf\text{-}c}(\mathcal{RAF})$ and that $\omega' \in \Omega'$, we have so: $\alpha \notin \omega'$ and $\neg\alpha \notin \omega'$. As a consequence, we have $\alpha \notin \tilde{K}$. As $t(\alpha)$ belongs to $\widetilde{\mathcal{RAF}}$ and α does not belong to $\widetilde{\mathcal{RAF}}$, We have: $\alpha \in Q^{inp}$.

From Cases 1 and 2, we prove so that Assertion 3 holds.

Assertion 4: $(x,y) \in K'_s \cup \tilde{K}'_s$ s.t. $y \in \omega' \implies x \in \omega' \cup \mathcal{I}$.
Let consider two cases: $(x,y) \in \tilde{K}'_s$ and $(x,y) \in K'_s$.

- **Case 1:** $(x,y) \in K'_s$

  Let $(x,y) \in K'_s$ s.t. $y \in \omega'$. Let suppose that $x \notin (\omega' \cup \mathcal{I})$. Following the definition of standard AF, we have thus: $x \in \mathcal{I}'$, $\mathcal{I}'$ being the set of added arguments to fit with the labelling of the input arguments (see Definition 17 on page 17). Also, following the definition of standard AF: $\nexists(x,y) \in K'_s$ s.t. $x \in \mathcal{I}'$ and $y \in \omega'$. We have a contradiction. We prove so that:

$$(x,y) \in K'_s \text{ s.t. } y \in \omega' \implies x \in \omega' \cup \mathcal{I}$$

- **Case 2:** $(x,y) \in \tilde{K}'_s$

  Let $(x,y) \in \tilde{K}'_s$ s.t. $y \in \omega'$. Trivially, x may belong to ω'. Let thus suppose that $x \notin \omega'$ and prove that $x \in \mathcal{I}$. Let consider three cases: $y \in A \cup K$, $y \in Not_A \cup Not_K$ and $y \in And_{A,K}$.

  - **Case 2.1:** $y \in A \cup K$

    Following the definition of `Raf2Af`, we have: $x \in And_{\tilde{A}_s,\tilde{K}_s}$. Let assume that $x = s(\alpha).\alpha$ with $\alpha \in \tilde{K}_s$ and $t(\alpha) = y$. If $\alpha \in \tilde{K}_s \cap \tilde{K}$ then we have: $\alpha \in \omega$. Given that $\omega \subseteq \omega'$, that $\omega' \in \Omega'$, that $\Omega' \in \mathcal{S}_{D\text{-}raf\text{-}c}(\mathcal{AF})$, we have: $s(\alpha).\alpha \in \omega'$, which contradicts: $x \notin \omega'$. We have so: $\alpha \in \tilde{K}_s \setminus \tilde{K}$. Thus α does not belong to $\widetilde{\mathcal{RAF}}$. As $t(\alpha) \in A \cup K$ and $t(\alpha) \in \omega'$, $t(\alpha)$ belongs to $\widetilde{\mathcal{RAF}}$. Given that $\alpha \notin \tilde{K}$ and $t(\alpha) \in \tilde{A} \cup \tilde{K}$, we have: $\alpha \in Q^{inp}$ and so $\alpha \in K$.

    In the flattening process of $\mathcal{RAF}$ the RAF-walk $(s(\alpha), \alpha, t(\alpha))$ will produce the following walks: $(s(\alpha), \neg s(\alpha), s(\alpha).\alpha, t(\alpha))$ and $(\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha))$. Given that $(s(\alpha).\alpha, t(\alpha)) \in K'$, that $s(\alpha).\alpha \notin \omega'$ and that $t(\alpha) \in \omega'$, we have: $(s(\alpha).\alpha, t(\alpha)) \in K_{\mathcal{I}}$. As a consequence we have: $x \in \mathcal{I}$.

  - **Case 2.2:** $y \in Not_A \cup Not_K$

    Following the definition of `Raf2Af`, we have: $x \in \tilde{A}_s \cup \tilde{K}_s$. Let assume that $y = \neg x$. If $x \in (\tilde{A}_s \cup \tilde{K}_s) \cap (\tilde{A} \cup \tilde{K})$, then we have: $x \in \omega$. Given that $\omega \subseteq \omega'$, we have: $x \in \omega'$, which contradicts: $x \notin \omega'$. We have so: $x \in (\tilde{A}_s \cup \tilde{K}_s) \setminus (\tilde{A} \cup \tilde{K})$. We have so: $x \in S^{inp} \cup Q^{inp}$. x is thus an element of $\mathcal{RAF}$ and we have: $x \in A \cup K$.

    Given that $\Omega' \in \mathcal{S}_{D\text{-}raf\text{-}c}(\mathcal{RAF})$ and that $\omega' \in \Omega'$, as $y = \neg x \in \omega'$, we have: $x \in \omega'$, which contradicts $x \notin \omega'$. As a consequence:

$$\nexists(x,y) \in \tilde{K}'_s \text{ s.t. } x \notin \omega' \text{ and } y \in Not_A \cup Not_K \tag{18.99}$$

- **Case 2.3:** $y \in And_{A,K}$

  Let assume that $y = s(\alpha).\alpha$ with $\alpha \in \tilde{K}_s$.

  Following the definition of `Raf2Af`, we have so: $x \in Not_{\tilde{A}_s} \cup Not_{\tilde{K}_s}$. Let assume that $x = \neg z$ with $z \in \tilde{A}_s \cup \tilde{K}_s$. We have so: $z = s(\alpha)$ or $z = \alpha$. If $x \in Not_{\tilde{K}_s}$ then: $z = \alpha$. Given that $s(\alpha).\alpha \in \omega'$, that $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{RAF})$ and that $\omega' \in \Omega'$, we have so: $\alpha \in \omega'$ and $\neg\alpha \in \omega'$, which contradicts $x \notin \omega'$. We have so: $x \in Not_{\tilde{A}_s}$ and so: $z = s(\alpha)$ and $z \in \tilde{A}_s$.

  Given that $\alpha \in \omega'$ and $\alpha \in \tilde{K}_s$, we have so: $\alpha \in \tilde{K}$. α is thus an element of $\widetilde{\mathcal{RAF}}$ and so of $\mathcal{RAF}$. Given that $\neg s(\alpha) \notin \omega'$, that $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{RAF})$ and that $\omega' \in \Omega'$, we have so: $s(\alpha) \notin \omega'$. As a consequence, $z = s(\alpha) \in \tilde{A}_s \setminus \tilde{A}$. $s(\alpha)$ is not an element of $\widetilde{\mathcal{RAF}}$. As $\alpha \in \tilde{K}$, we have so: $s(\alpha) \in S^{inp}$ and so $s(\alpha) \in A$.

  In the flattening process of $\mathcal{RAF}$ the RAF-walk $(s(\alpha), \alpha, t(\alpha))$ will produce the following walks: $(s(\alpha), \neg s(\alpha), s(\alpha).\alpha, t(\alpha))$ and $(\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha))$. Given that $(\neg s(\alpha), s(\alpha).\alpha) \in K'$, that $\neg s(\alpha) \notin \omega'$ and that $s(\alpha).\alpha \in \omega'$, we have: $(\neg s(\alpha), s(\alpha).\alpha) \in K_{\mathcal{I}}$. As a consequence we have: $x \in \mathcal{I}$.

From Cases 2.1, 2.2, 2.3, we prove that:

$$(x,y) \in \tilde{K}'_s \text{ s.t. } y \in \omega' \implies x \in \omega' \cup \mathcal{I}$$

From Cases 1 and 2, we prove so that Assertion 4 holds. ∎

Lemma 12. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and $\mathcal{AF} =$ `Raf2Af`$(\mathcal{RAF})$ (with $\mathcal{AF} = \langle A', K' \rangle$) be the corresponding AF of $\mathcal{RAF}$ (with $\mathcal{AF} = \langle A', K' \rangle$). Let Ω be a partition of $(A \cup K)$ and $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{AF})$ be the RAF-compliant partition of A' corresponding to Ω, i.e. $\Omega' = \{\omega' = \omega \cup \{\neg x | x \in \omega\} \cup \{s(\alpha).\alpha \in And_{A,K} | \alpha \in \omega\} | \omega \in \Omega\}$. Let $\omega \in \Omega$ and $\omega' \in \Omega'$ be its counterpart in $\mathcal{AF}$. Let $\widetilde{\mathcal{RAF}} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t \rangle$ be the partial RAF corresponding to ω. Let $\mathfrak{I} = \langle S^{inp}, Q^{inp} \rangle$ be the input elements of $\widetilde{\mathcal{RAF}}$ and $\mathcal{L}^{inp}$ be a structure labelling of them. Let $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ be a RAF with input of $\mathcal{RAF}$ and $\left\langle \mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$ be its corresponding AF with input, as defined in Definition 105 on page 155. Let $\widetilde{\mathcal{RAF}}_s = \langle \tilde{A}_s, \tilde{K}_s, s_s, t_s \rangle$ be the standard RAF of $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ and let std-$\mathcal{AF} = \langle A'_s, K'_s \rangle$ be the standard AF corresponding to $\left\langle \mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$. Let $\widetilde{\mathcal{RAF}}'_s =$ `Raf2Af`$(\widetilde{\mathcal{RAF}}_s)$ be the corresponding AF of $\widetilde{\mathcal{RAF}}_s$ (with $\widetilde{\mathcal{RAF}}'_s = \langle \tilde{A}'_s, \tilde{K}'_s \rangle$). The following assertions hold:*

1. $A'_s \cap \tilde{A}'_s = \omega' \cup \mathcal{I}$
2. $K'_s \cap (\omega' \times \omega') = \tilde{K}'_s \cap (\omega' \times \omega')$
3. $K'_s \cap (\mathcal{I} \times \omega') = \tilde{K}'_s \cap (\mathcal{I} \times \omega')$

Proof of Lemma 12. **Assertion 1:** $A'_s \cap \tilde{A}'_s = \omega' \cup \mathcal{I}$

- **Step 1:** Let prove that $\omega' \subseteq A'_s \cap \tilde{A}'_s$.

  First of all, notice that if $x \in \omega'$ then $x \in \mathcal{AF} \downarrow_{\omega'}$, and so by definition of the standard AF, we have $x \in std\text{-}\mathcal{AF}$. More precisely we have: $x \in A'_s$. The following property holds then:

$$x \in \omega' \implies x \in A'_s \tag{18.100}$$

Secondly, notice that if $x \in \omega$ then x belongs to $\widetilde{\mathcal{RAF}}$. We have thus, by definition of the standard RAF: $x \in \tilde{A}_s \cup \tilde{K}_s$. As a consequence, x and $\neg x$ belong to $\widetilde{\mathcal{RAF}}'_s$ and we have: $x \in \tilde{A}'_s$ and $\neg x \in \tilde{A}'_s$. The following property holds then:

$$x \in \omega \implies x \in \tilde{A}'_s \text{ and } \neg x \in \tilde{A}'_s \tag{18.101}$$

Let prove that $x \in \omega' \setminus \omega$ belongs to $\tilde{A}'_s$. We have two cases to consider: $x \in (\omega' \cap (Not_A \cup Not_K))$ and $x \in (\omega' \cap And_{A,K})$.

- **Case 1:** $x \in (\omega' \cap (Not_A \cup Not_K))$
  Let assume that $x = \neg y$ as it belongs whether to Not_A or Not_K. Given that $\neg y \in \omega'$, that $\omega' \in \Omega'$ and that $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{AF})$, we have then: $y \in \omega' \cap (A \cup K)$ and so: $y \in \omega$. According to Equation (18.101), we have thus: $y \in \tilde{A}'_s$ and $\neg y \in \tilde{A}'_s$. As a consequence, we have: $x \in \tilde{A}'_s$. The following property holds then:

$$x \in (\omega' \cap (Not_A \cup Not_K)) \implies x \in \tilde{A}'_s \tag{18.102}$$

- **Case 2:** $x \in (\omega' \cap And_{A,K})$
  Let assume that $x = s(\alpha).\alpha$ as it belongs to $And_{A,K}$. Given that $s(\alpha).\alpha \in \omega'$, that $\omega' \in \Omega'$ and that $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{AF})$, we have then: $\neg\alpha \in \omega'$ and $\alpha \in \omega'$. Furthermore, given that $\alpha \in K$, we have: $\alpha \in \omega$. As $\omega \cap K = \tilde{K}$, we have: $\alpha \in \tilde{K}$ (*i.e.* α is an attack belonging to $\widetilde{\mathcal{RAF}}$). By definition of the standard RAF, we have thus: $\alpha \in \tilde{A}_s$. Now, given that $\alpha \in \tilde{A}_s$, by definition of `Raf2Af`, we have thus: $s(\alpha) \in \tilde{A}'_s$, $\alpha \in \tilde{A}'_s$ and $s(\alpha).\alpha \in \tilde{A}'_s$. As a consequence we have: $x \in \tilde{A}'_s$. The following property holds then:

$$x \in (\omega' \cap And_{A,K}) \implies x \in \tilde{A}'_s \tag{18.103}$$

From Equations (18.101) to (18.103), we prove the following property:

$$x \in \omega' \implies x \in \tilde{A}'_s \tag{18.104}$$

From Equations (18.100) and (18.104) on the previous page and on this page we prove that:

$$\omega' \subseteq A'_s \cap \tilde{A}'_s$$

- **Step 2:** Let prove that $\mathcal{I} \subseteq A'_s \cap \tilde{A}'_s$.
  By definition of the standard AF, we have:

$$\mathcal{I} \subseteq A'_s \tag{18.105}$$

  Let prove that: $\mathcal{I} \subseteq \tilde{A}'_s$. Let $x \in \mathcal{I}$. Following Assertion 1 of Lemma 11 on page 267, we have: $x \in Not_A$ or $x \in And_{A,K}$. Let consider two cases.

  - **Case 1:** $x \in Not_A$.
    Let assume that $x = \neg y$ with $y \in A$. Following the definition of `Raf2Af`, we have:

$$(\neg y, z) \in K' \implies z \in And_{A,K}$$

    We know that such an attack exists as $\neg y \in \mathcal{I}$. Let $s(\alpha).\alpha$ be an argument such that $(\neg y, s(\alpha).\alpha) \in K'$. Following the definition of `Raf2Af`, we have: $s(\alpha) = y$. We have so: $s(\alpha) \notin \omega'$ and $\neg s(\alpha) \notin \omega'$. Furthermore, we have: $s(\alpha).\alpha \in \omega'$.

Following Assertion 2 of Lemma 11 on page 267, we have thus: $y \in S^{inp}$ and so $y \in \tilde{A}_s$. Then, following the definition of `Raf2Af`, we have thus: $y \in \tilde{A}'_s$ and $\neg y \in \tilde{A}'_s$, and so $x \in \tilde{A}'_s$. The following equation holds then:

$$x \in \mathcal{J} \cap Not_A \implies x \in \tilde{A}'_s \tag{18.106}$$

- **Case 2:** $x \in And_{A,K}$.

  Let assume that $x = s(\alpha).\alpha$ with $s(\alpha) \in A$ and $\alpha \in K$. Following the definition of `Raf2Af`, we have:

  $$(s(\alpha).\alpha, z) \in K' \implies z \in A \cup K$$

  Let z be an element such that $(s(\alpha).\alpha, z) \in K'$ s.t. $z \in \omega'$. We know that such an attack exists as $s(\alpha).\alpha \in \mathcal{J}$. Following Assertion 3 of Lemma 11 on page 267, we have thus: $\alpha \in Q^{inp}$. Following the definition of standard RAF, we have so: $\alpha \in \tilde{K}_s$. Then, following the definition of `Raf2Af`, we have thus: $\alpha \in \tilde{A}'_s$, $\neg\alpha \in \tilde{A}'_s$ and $s(\alpha).\alpha \in \tilde{A}'_s$, and so $x \in \tilde{A}'_s$. The following equation holds then:

  $$x \in \mathcal{J} \cap And_{A,K} \implies x \in \tilde{A}'_s \tag{18.107}$$

From Equations (18.105) to (18.107) on pages 271–272 we prove so that:

$$\mathcal{J} \subseteq A'_s \cap \tilde{A}'_s$$

- **Step 3:** Let prove that $A'_s \cap \tilde{A}'_s \subseteq \omega' \cup \mathcal{J}$.

  Let $x \in A'_s \cap \tilde{A}'_s$. Notice that following the definition of standard AF (Definition 17 on page 17 used to create $std\text{-}\mathcal{AF}$), we have: $\tilde{A}'_s \cap \mathcal{J}' = \varnothing$, $\mathcal{J}'$ being the set of added arguments to fit with the labelling of the input arguments in $std\text{-}\mathcal{AF}$.[5] Given that $x \in \tilde{A}'_s$, we have so: $x \notin \mathcal{J}'$. As a consequence, we have: $x \in \omega' \cup \mathcal{J}$. We prove so that:

  $$A'_s \cap \tilde{A}'_s \subseteq \omega' \cup \mathcal{J}$$

From Steps 1, 2 and 3, we prove that Assertion 1 holds.

Assertion 2: $K'_s \cap (\omega' \times \omega') = \tilde{K}'_s \cap (\omega' \times \omega')$

- **Step 1:** Let prove that $K'_s \cap (\omega' \times \omega') \subseteq \tilde{K}'_s \cap (\omega' \times \omega')$

  Let $(x,y) \in K'_s \cap (\omega' \times \omega')$. Given that $(x,y) \in (\omega' \times \omega')$ implies that: $x \notin \mathcal{J}'$ and $y \notin \mathcal{J}'$, we have thus, following Assertion 1: $x \in A'_s \cap \tilde{A}'_s$ and $y \in A'_s \cap \tilde{A}'_s$.

  Following the definition of `Raf2Af`, we have the following four cases to consider: ($x \in A \cup K$ and $y \in Not_A \cup Not_K$), ($x \in Not_A$ and $y \in And_{A,K}$), ($x \in Not_K$ and $y \in And_{A,K}$) and ($x \in And_{A,K}$ and $y \in A \cup K$)

  - **Case 1:** $x \in A \cup K$ and $y \in Not_A \cup Not_K$

[5] Remind that $\mathcal{J} \cap \mathcal{J}' = \varnothing$ and that the arguments that are in $\mathcal{J}'$ are different from those created in $\widetilde{\mathcal{RAF}}_s$ to fit input labelling of $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$.

According to the definition of `Raf2Af`, we have: $y = \neg x$. If $x \in A \cup K$, then we have $x \in \omega$. As a consequence we have: $x \in \tilde{A} \cup \tilde{K}$ and then $x \in \tilde{A}_s \cup \tilde{K}_s$. The flattening of $\widetilde{\mathcal{RAF}}_s$ will thus produced an attack $(x, \neg x) \in \tilde{K}'_s$, with $x \in \tilde{A}'_s$ and $\neg x \in \tilde{A}'_s$. The following property holds then:

$$(x,y) \in K'_s \cap (\omega' \times \omega') \text{ s.t. } x \in A \cup K \text{ and } y \in Not_A \cup Not_K$$
$$\Longrightarrow \tag{18.108}$$
$$(x,y) \in \tilde{K}'_s \cap (\omega' \times \omega')$$

– **Case 2:** $x \in Not_A$ and $y \in And_{A,K}$
As $x \in Not_A$, let assume that $x = \neg z$ with $z \in A$.
Given that $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{RAF})$, that $\omega' \in \Omega'$ and that $\neg z \in \omega'$, we have: $z \in \omega'$. Moreover, as $z \in A$, we have: $z \in \omega$. z is thus an element of $\widetilde{\mathcal{RAF}}$.
Let assume that $y = s(\alpha).\alpha$ with $\alpha \in K$. We have thus: $s(\alpha) = z$.
Given that $s(\alpha).\alpha \in \omega'$, that $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{RAF})$ and that $\omega' \in \Omega'$, we have so: $\alpha \in \omega'$. As $\alpha \in K$, we have: $\alpha \in \omega$. α is thus an element of $\widetilde{\mathcal{RAF}}$.
As a consequence, we have: $\alpha \in \tilde{K}_s$ and $s(\alpha) \in \tilde{A}_s$. Following the definition of `Raf2Af`, we have so: $\{s(\alpha), \neg s(\alpha), s(\alpha).\alpha\} \in Walks_{af}(\widetilde{\mathcal{RAF}}'_s)$.
The following property then holds:

$$s(\alpha) \in \omega' \text{ and } s(\alpha).\alpha \in \omega' \implies \{s(\alpha), \neg s(\alpha), s(\alpha).\alpha\} \in Walks_{af}(\widetilde{\mathcal{RAF}}'_s) \tag{18.109}$$

As a consequence we have: $(\neg z, z.\alpha) \in \tilde{K}'_s$ and so: $(x,y) \in \tilde{K}'_s$.
The following property then holds:

$$(x,y) \in K'_s \cap (\omega' \times \omega') \text{ s.t. } x \in Not_A \text{ and } y \in And_{A,K}$$
$$\Longrightarrow \tag{18.110}$$
$$(x,y) \in \tilde{K}'_s \cap (\omega' \times \omega')$$

– **Case 3:** $x \in Not_K$ and $y \in And_{A,K}$
As $x \in Not_K$, let assume that $x = \neg\alpha$ with $\alpha \in K$ and that $y = s(\alpha).\alpha$.
Given that $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{RAF})$, that $\omega' \in \Omega'$ and that $\neg\alpha \in \omega'$, we have: $\alpha \in \omega'$. Moreover, as $\alpha \in K$, we have: $\alpha \in \omega$. α is thus an element of $\widetilde{\mathcal{RAF}}$.
As a consequence, we have: $\alpha \in \tilde{K}_s$. Following the definition of `Raf2Af`, we have so:
$\{\alpha, \neg\alpha, s(\alpha).\alpha\} \in Walks_{af}(\widetilde{\mathcal{RAF}}'_s)$.
The following property then holds:

$$\alpha \in \omega' \implies \{\alpha, \neg\alpha, s(\alpha).\alpha\} \in Walks_{af}(\widetilde{\mathcal{RAF}}'_s) \tag{18.111}$$

As a consequence we have: $(\neg\alpha, s(\alpha).\alpha) \in \tilde{K}'_s$ and so: $(x,y) \in \tilde{K}'_s$.

The following property then holds:

$$(x,y) \in K_s' \cap (\omega' \times \omega') \text{ s.t. } x \in Not_K \text{ and } y \in And_{A,K}$$
$$\Longrightarrow \tag{18.112}$$
$$(x,y) \in \tilde{K}_s' \cap (\omega' \times \omega')$$

- **Case 4:** $x \in And_{A,K}$ and $y \in A \cup K$
  As $x \in And_{A,K}$, let assume that $x = s(\alpha).\alpha$ such that $y = t(\alpha)$. As $s(\alpha).\alpha \in \omega'$ and as $y \in \omega$, we have following Equation (18.111) on the previous page: $\{\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha)\} \in Walks_{af}(\widetilde{\mathcal{RAF}}'_s)$. As a consequence we have: $(s(\alpha).\alpha, t(\alpha)) \in \tilde{K}_s'$ and so: $(x,y) \in \tilde{K}_s'$.
  The following property then holds:

$$(x,y) \in K_s' \cap (\omega' \times \omega') \text{ s.t. } x \in And_{A,K} \text{ and } y \in A \cup K$$
$$\Longrightarrow \tag{18.113}$$
$$(x,y) \in \tilde{K}_s' \cap (\omega' \times \omega')$$

From Cases 1, 2, 3 and 4 we prove so that:

$$(x,y) \in K_s' \cap (\omega' \times \omega') \implies (x,y) \in \tilde{K}_s' \cap (\omega' \times \omega') \tag{18.114}$$

- **Step 2:** Let prove that $\tilde{K}_s' \cap (\omega' \times \omega') \subseteq K_s' \cap (\omega' \times \omega')$
  Let $(x,y) \in \tilde{K}_s' \cap (\omega' \times \omega')$. According to the definition of `Raf2Af`, we have whether: ($x \in A \cup K$ and $y \in Not_A \cup Not_K$ s.t. $y = \neg x$), ($x \in Not_A$ and $y \in And_{A,K}$), ($x \in Not_K$ and $y \in And_{A,K}$) or ($x \in And_{A,K}$ and $y \in A \cup K$).
  Let consider those cases.
  - **Case 1**: $x \in A \cup K$ and $y \in Not_A \cup Not_K$ s.t. $y = \neg x$
    Given that $x \in A \cup K$ then, following the definition of `Raf2Af`, we have: $(x, \neg x)$ in K'. As $x \in \omega'$, $\Omega' \in \mathcal{S}_{D\text{-}raf\text{-}c}(\mathcal{RAF})$ and $\omega' \in \Omega'$, we have so: $\neg x \in \omega'$. Both x and $\neg x$ are thus arguments of $\mathcal{AF} \downarrow_{\omega'}$. As $x \in \omega' \cap (A \cup K)$, we have following the definition of `Raf2Af` and the definition of standard AF the following property:

$$x \in \omega' \cap (A \cup K) \implies (x, \neg x) \in K_s' \tag{18.115}$$

    We have so: $(x,y) \in K_s'$.
  - **Case 2**: $x \in Not_A$ and $y \in And_{A,K}$
    As $x \in Not_A$, let assume that $x = \neg z$, with $z \in A$. As $y \in And_{A,K}$, let assume that $y = s(\alpha).\alpha$ with $\alpha \in K$ such that $s(\alpha) = z$.
    Given that $s(\alpha).\alpha \in \omega'$, that $\Omega' \in \mathcal{S}_{D\text{-}raf\text{-}c}(\mathcal{RAF})$ and that $\omega' \in \Omega'$, we have: $\alpha \in \omega'$. α is thus an argument of $\mathcal{AF} \downarrow_{\omega'}$. Following the definition of `Raf2Af`, and the definition of standard AF, we have thus the following property:

$$\alpha \in \omega' \cap K \implies \{\alpha, \neg\alpha, s(\alpha).\alpha\} \in Walks_{af}(std\text{-}\mathcal{AF}) \tag{18.116}$$

Furthermore, we have also the following property:

$$\alpha \in \omega' \cap K \text{ and } s(\alpha) \in \omega' \implies \{s(\alpha), \neg s(\alpha), s(\alpha).\alpha\} \in Walks_{af}(std\text{-}\mathcal{AF}) \tag{18.117}$$

As a consequence, we have: $(\neg s(\alpha), s(\alpha).\alpha) \in K'_s$, and so: $(x,y) \in K'_s$.

- **Case 3**: $x \in Not_K$ and $y \in And_{A,K}$

  As $x \in Not_K$, let assume that $x = \neg\alpha$ with $\alpha \in K$ and $y = s(\alpha).\alpha$. Given that $s(\alpha).\alpha \in \omega'$, that $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{RAF})$ and that $\omega' \in \Omega'$, we have: $\alpha \in \omega'$. Following Equation (18.116), we have thus: $\{\alpha, \neg\alpha, s(\alpha).\alpha\} \in Walks_{af}(std\text{-}\mathcal{AF})$.

  The following property holds then:

$$s(\alpha).\alpha \in \omega' \implies \{\alpha, \neg\alpha, s(\alpha).\alpha\} \in Walks_{af}(std\text{-}\mathcal{AF}) \tag{18.118}$$

  As a consequence, we have: $(\neg\alpha, s(\alpha).\alpha) \in K'_s$ and so: $(x,y) \in K'_s$

- **Case 4**: $x \in And_{A,K}$ and $y \in A \cup K$

  As $x \in And_{A,K}$, let assume that $x = s(\alpha).\alpha$ with $\alpha \in K$ and $y = t(\alpha)$. Following Equation (18.118) and the fact that $y \in \omega'$, we have: $\{\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha)\} \in Walks_{af}(std\text{-}\mathcal{AF})$.

  As a consequence, we have: $(s(\alpha).\alpha, t(\alpha)) \in K'_s$ and so: $(x,y) \in K'_s$.

From Cases 1, 2, 3 and 4, the following property then holds:

$$(x,y) \in \tilde{K}'_s \text{ s.t. } x \in \omega' \text{ and } y \in \omega' \implies (x,y) \in K'_s \tag{18.119}$$

From Steps 1 and 2, we prove that Assertion 2 holds.

Assertion 3: $K'_s \cap (\mathcal{J} \times \omega') = \tilde{K}'_s \cap (\mathcal{J} \times \omega')$

- **Step 1:** $K'_s \cap (\mathcal{J} \times \omega') \subseteq \tilde{K}'_s \cap (\mathcal{J} \times \omega')$

  Let $(x,y) \in K'_s \cap (\mathcal{J} \times \omega')$. Following Assertion 1 of Lemma 11 on page 267, we have whether: $x \in Not_A$ or $x \in And_{A,K}$. Let consider those two cases.

  - **Case 1**: $x \in \mathcal{J} \cap Not_A$ and $y \in \omega'$.

    As $x \in Not_A$, let assume that $x = \neg z$ with $z \in A$. Following the definition of `Raf2Af`, we have: $y \in And_{A,K}$. Let thus assume that $y = s(\alpha).\alpha$ with $\alpha \in K$ such that $s(\alpha) = z$.

    As $s(\alpha).\alpha \in \omega'$, following Equation (18.111) on page 273, we have:

$$\{\alpha, \neg\alpha, s(\alpha).\alpha\} \in Walks_{af}(\widetilde{\mathcal{RAF}}'_s)$$

    As a consequence we have: $(\neg\alpha, s(\alpha).\alpha) \in \tilde{K}'_s$ and so: $(x,y) \in \tilde{K}'_s$.

    The following property then holds:

$$(x,y) \in K'_s \cap (\mathcal{J} \times \omega') \text{ s.t. } x \in Not_A \implies (x,y) \in \tilde{K}'_s \cap (\mathcal{J} \times \omega') \tag{18.120}$$

- **Case 2**: $x \in \mathcal{J} \cap And_{A,K}$ and $y \in \omega'$.
  As $x \in And_{A,K}$, let assume that $x = s(\alpha).\alpha$ with $\alpha \in K$ such that $t(\alpha) = y$ with $y \in A \cup K$. Following Assertion 3 of Lemma 11 on page 267, we have: $\alpha \in Q^{inp}$. Following the definition of the standard RAF, we have so: $\alpha \in \tilde{K}_s$. Following the definition of `Raf2Af` and the fact that $y \in \omega'$, we have then: $\{\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha)\} \in Walks_{af}(\widetilde{\mathcal{RAF}}'_s)\}$.
  The following property then holds:

$$\alpha \in \mathcal{J} \implies \{s(\alpha), \neg s(\alpha), s(\alpha).\alpha, t(\alpha)\} \in Walks_{af}(\widetilde{\mathcal{RAF}}'_s) \tag{18.121}$$

  As a consequence: $(s(\alpha).\alpha, t(\alpha)) \in \tilde{K}'_s$ and so: $(x,y) \in \tilde{K}'_s$.
  The following property then holds:

$$(x,y) \in K'_s \cap (\mathcal{J} \times \omega') \text{ s.t. } x \in And_{A,K} \implies (x,y) \in \tilde{K}'_s \cap (\mathcal{J} \times \omega') \tag{18.122}$$

From Cases 1 and 2, we prove so that:

$$(x,y) \in K'_s \cap (\mathcal{J} \times \omega') \implies (x,y) \in \tilde{K}'_s \cap (\mathcal{J} \times \omega') \tag{18.123}$$

- **Step 2: $\tilde{K}'_s \cap (\mathcal{J} \times \omega') \subseteq K'_s \cap (\mathcal{J} \times \omega')$**
  Let $(x,y) \in \tilde{K}'_s \cap (\mathcal{J} \times \omega')$. Following Assertion 1 of Lemma 11 on page 267, we have whether: $x \in Not_A$ or $x \in And_{A,K}$. Let consider those two cases.

  - **Case 1**: $x \in Not_A \setminus \omega'$ and $y \in \omega'$
    As $x \in Not_A$, let assume that $x = \neg z$ with $z \in A$. We have so: $y \in And_{A,K}$. Let assume that $y = s(\alpha).\alpha$ with $\alpha \in K$ such that $s(\alpha) = z$. In the flattening process of $\mathcal{RAF}$ the RAF-walk $(s(\alpha), \alpha, t(\alpha))$ will produce the following walks: $(s(\alpha), \neg s(\alpha), s(\alpha).\alpha, t(\alpha))$ and $(\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha))$. Given that $(\neg s(\alpha), s(\alpha).\alpha) \in K'$, that $\neg s(\alpha) \notin \omega'$ and that $s(\alpha).\alpha \in \omega'$, we have: $(\neg s(\alpha), s(\alpha).\alpha) \in K_{\mathcal{J}}$. Following the definition of standard AF, we have thus: $(\neg s(\alpha), s(\alpha).\alpha) \in K'_s$, and so: $(x,y) \in K'_s$.
  - **Case 2**: $x \in And_{A,K} \setminus \omega'$ and $y \in \omega'$
    As $x \in And_{A,K}$, let assume that $x = s(\alpha).\alpha$ with $\alpha \in K$. We have so: $y = t(\alpha)$ and $y \in (A \cup K)$, following the definition of `Raf2Af`. In the flattening process of $\mathcal{RAF}$ the RAF-walk $(s(\alpha), \alpha, t(\alpha))$ will produce the following walks: $(s(\alpha), \neg s(\alpha), s(\alpha).\alpha, t(\alpha))$ and $(\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha))$. Given that $(s(\alpha).\alpha, t(\alpha)) \in K'$, that $\neg s(\alpha).\alpha \notin \omega'$ and that $t(\alpha) \in \omega'$, we have: $(s(\alpha).\alpha, t(\alpha)) \in K_{\mathcal{J}}$. Following the definition of standard AF, we have thus: $(s(\alpha).\alpha, t(\alpha)) \in K'_s$, and so: $(x,y) \in K'_s$.

  From Cases 1 and 2, the following property then holds:

$$(x,y) \in \tilde{K}'_s \text{ s.t. } x \in \mathcal{J} \text{ and } y \in \omega' \implies (x,y) \in K'_s \tag{18.124}$$

From Steps 1 and 2, we prove that Assertion 3 holds. ■

Lemma 13. *Let $\mathcal{RAF} = \langle A, K, s, t \rangle$ be a RAF and $\mathcal{AF} =$ `Raf2Af`$(\mathcal{RAF})$ be the corresponding AF of $\mathcal{RAF}$ (with $\mathcal{AF} = \langle A', K' \rangle$). Let Ω be a partition of $(A \cup K)$ and $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{AF})$ be the RAF-compliant partition of A' corresponding to Ω, i.e. $\Omega' = \{\omega' = \omega \cup \{\neg x | x \in \omega\} \cup \{s(\alpha).\alpha \in And_{A,K} | \alpha \in \omega\} | \omega \in \Omega\}$. Let $\omega \in \Omega$ and $\omega' \in \Omega'$ be its counterpart in $\mathcal{AF}$. Let $\widetilde{\mathcal{RAF}} = \langle \tilde{A}, \tilde{K}, \tilde{s}, \tilde{t}, s, t \rangle$ be the partial RAF corresponding to*

ω. Let $\mathfrak{I} = \left\langle S^{inp}, Q^{inp} \right\rangle$ be the input elements of $\widetilde{\mathcal{RAF}}$ and $\mathcal{L}^{inp}$ be a structure labelling of them. Let $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ be a RAF with input of $\mathcal{RAF}$ and $\left\langle \mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$ be its corresponding AF with input, as defined in Definition 105 on page 155. Let $\widetilde{\mathcal{RAF}}_s = \langle \tilde{A}_s, \tilde{K}_s, s_s, t_s \rangle$ be the standard of $\left\langle \widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp} \right\rangle$ and let std-$\mathcal{AF} = \langle A'_s, K'_s \rangle$ be the standard AF corresponding to $\left\langle \mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}} \right\rangle$. Let $\widetilde{\mathcal{RAF}}'_s = \texttt{Raf2Af}(\widetilde{\mathcal{RAF}}_s)$ be the corresponding AF of $\widetilde{\mathcal{RAF}}_s$ (with $\widetilde{\mathcal{RAF}}'_s = \langle \tilde{A}'_s, \tilde{K}'_s \rangle$). Let σ be a complete-based AF semantics (See Definition 38 on page 49) and let σ-raf be the RAF semantics corresponding to σ.[6] The following assertions hold:

1. $\forall \neg a \in \mathcal{I} \cap Not_A$, $\forall \ell \in \mathscr{L}_\sigma(\widetilde{\mathcal{RAF}}'_s)$,

$$\ell(\neg a) = \begin{cases} \texttt{in} \iff \mathcal{L}^{inp}(a) = \texttt{out} \\ \texttt{out} \iff \mathcal{L}^{inp}(a) = \texttt{in} \\ \texttt{und} \iff \mathcal{L}^{inp}(a) = \texttt{und} \end{cases}$$

2. $\forall s(\alpha).\alpha \in \mathcal{I} \cap And_{A,K}$, $\forall \ell \in \mathscr{L}_\sigma(\widetilde{\mathcal{RAF}}'_s)$,

$$\ell(s(\alpha).\alpha) = \begin{cases} \texttt{in} \iff \mathcal{L}^{inp}(\alpha) = \texttt{in}\ and\ \mathcal{L}^{inp}(s(\alpha)) = \texttt{in} \\ \texttt{out} \iff \mathcal{L}^{inp}(\alpha) = \texttt{out}\ or\ \mathcal{L}^{inp}(s(\alpha)) = \texttt{out} \\ \texttt{und} \iff \left(\begin{array}{l} \mathcal{L}^{inp}(\alpha) \neq \texttt{out}\ and\ \mathcal{L}^{inp}(s(\alpha)) \neq \texttt{out}\ and \\ (\mathcal{L}^{inp}(\alpha) = \texttt{und}\ or\ \mathcal{L}^{inp}(s(\alpha)) = \texttt{und}) \end{array} \right) \end{cases}$$

3. $\forall a \in \mathcal{I}$, $\forall \ell_1 \in \mathscr{L}_\sigma(\widetilde{\mathcal{RAF}}'_s)$, $\forall \ell_2 \in \mathscr{L}_\sigma(std\text{-}\mathcal{AF})$,

$$\ell_1(a) = \ell_2(a)$$

***Proof of Lemma 13*.** **Assertion 1**: $\forall \neg a \in \mathcal{I} \cap Not_A$, $\forall \ell \in \mathscr{L}_\sigma(\widetilde{\mathcal{RAF}}'_s)$,

$$\ell(\neg a) = \begin{cases} \texttt{in} \iff \mathcal{L}^{inp}(a) = \texttt{out} \\ \texttt{out} \iff \mathcal{L}^{inp}(a) = \texttt{in} \\ \texttt{und} \iff \mathcal{L}^{inp}(a) = \texttt{und} \end{cases}$$

Let $\ell \in \mathscr{L}_\sigma(\widetilde{\mathcal{RAF}}'_s)$ and $\neg a \in \mathcal{I} \cap Not_A$. Given that $\neg a \in \mathcal{I} \cap Not_A$ we have following Definition 105 on page 155: $a \in S^{inp}$. As a consequence, following the definition of standard RAF we have: $a \in \tilde{A}_s$. Following the definition of `Raf2Af`, we have so: $a \in \tilde{A}'_s$. As a consequence $\ell(a)$ must be defined. Let $\mathcal{L} =$ `afLab2RafLab`(ℓ). Given that σ-raf is the RAF semantics corresponding to σ, we have: $\mathcal{L} \in \mathscr{L}_{\sigma\text{-}raf}(\widetilde{\mathcal{RAF}}_s)$. Moreover, according to Definition 78 on page 118 we have:

$$\forall x \in (\tilde{A}_s \cup \tilde{K}_s), \ell(x) = \mathcal{L}(x)$$

Given that $\mathcal{L} \in \mathscr{L}_{\sigma\text{-}raf}(\widetilde{\mathcal{RAF}}'_s)$, we have following the definition of standard RAF:

$$\forall x \in S^{inp}, \mathcal{L}(x) = \mathcal{L}^{inp}(x)$$

[6]See Definition 106 on page 159 for RAF and AF semantics correspondence.

As a consequence, we have:

$$\forall x \in S^{inp},\, \ell(x) = \mathcal{L}^{inp}(x) \tag{18.125}$$

As σ is complete-based, we have: $\ell(a) = \texttt{in} \iff \ell(\neg a) = \texttt{out}$, $\ell(a) = \texttt{out} \iff \ell(\neg a) = \texttt{in}$ and $\ell(a) = \texttt{und} \iff \ell(\neg a) = \texttt{und}$. From those equivalences and from Equation (18.125), we prove so that Assertion 1 holds.

Assertion 2: $\forall s(\alpha).\alpha \in \mathcal{I} \cap And_{A,K}$, $\forall \ell \in \mathscr{L}_\sigma(\widetilde{\mathcal{RAF}}'_s)$,

$$\ell(s(\alpha).\alpha) = \begin{cases} \texttt{in} \iff \mathcal{L}^{inp}(\alpha) = \texttt{in} \text{ and } \mathcal{L}^{inp}(s(\alpha)) = \texttt{in} \\ \texttt{out} \iff \mathcal{L}^{inp}(\alpha) = \texttt{out} \text{ or } \mathcal{L}^{inp}(s(\alpha)) = \texttt{out} \\ \texttt{und} \iff \left(\begin{array}{l} \mathcal{L}^{inp}(\alpha) \neq \texttt{out} \text{ and } \mathcal{L}^{inp}(s(\alpha)) \neq \texttt{out} \text{ and} \\ (\mathcal{L}^{inp}(\alpha) = \texttt{und} \text{ or } \mathcal{L}^{inp}(s(\alpha)) = \texttt{und}) \end{array} \right) \end{cases}$$

Let $\ell \in \mathscr{L}_\sigma(\widetilde{\mathcal{RAF}}'_s)$ and $s(\alpha).\alpha \in \mathcal{I} \cap And_{A,K}$. Given that $s(\alpha).\alpha \in \mathcal{I} \cap And_{A,K}$, we have following Definition 105 on page 155: $\alpha \in Q^{inp}$ and $s(\alpha) \in S^{inp}$. As a consequence, following the definition of standard RAF we have: $\alpha \in \tilde{K}_s$ and $s(\alpha) \in \tilde{A}_s$. The flattening process of $\widetilde{\mathcal{RAF}}_s$ will thus produce the following walks: $(s(\alpha), \neg s(\alpha), s(\alpha).\alpha, t(\alpha))$ and $(\alpha, \neg\alpha, s(\alpha).\alpha, t(\alpha))$. As a consequence $\ell(s(\alpha))$, $\ell(\neg s(\alpha))$, $\ell(\alpha)$ and $\ell(\neg\alpha)$ must be defined.

Let $\mathcal{L} = \texttt{afLab2RafLab}(\ell)$. Given that σ-raf is the RAF semantics corresponding to σ, we have: $\mathcal{L} \in \mathscr{L}_{\sigma\text{-}raf}(\widetilde{\mathcal{RAF}}_s)$. Moreover, according to Definition 78 on page 118 we have:

$$\forall x \in (\tilde{A}_s \cup \tilde{K}_s),\, \ell(x) = \mathcal{L}(x)$$

Given that $\mathcal{L} \in \mathscr{L}_{\sigma\text{-}raf}(\widetilde{\mathcal{RAF}}'_s)$, we have following the definition of standard RAF:

$$\forall x \in (S^{inp} \cup Q^{inp}),\, \mathcal{L}(x) = \mathcal{L}^{inp}(x)$$

As a consequence, we have:

$$\forall x \in (S^{inp} \cup Q^{inp}),\, \ell(x) = \mathcal{L}^{inp}(x) \tag{18.126}$$

As for all $s(\alpha).\alpha \in And_{\tilde{A}_s,\tilde{K}_s}$ (σ being complete-based), we have:

$$\ell(s(\alpha).\alpha) = \begin{cases} \texttt{in} \iff \ell(\alpha) = \texttt{in} \text{ and } \ell(s(\alpha)) = \texttt{in} \\ \texttt{out} \iff \ell(\alpha) = \texttt{out} \text{ or } \ell(s(\alpha)) = \texttt{out} \\ \texttt{und} \iff \left(\begin{array}{l} \ell(\alpha) \neq \texttt{out} \text{ and } \ell(s(\alpha)) \neq \texttt{out} \text{ and} \\ (\ell(\alpha) = \texttt{und} \text{ or } \ell(s(\alpha)) = \texttt{und}) \end{array} \right) \end{cases}$$

And as Equation (18.126) holds, we prove so that Assertion 2 holds.

Assertion 3: $\forall a \in \mathcal{I}$, $\forall \ell_1 \in \mathscr{L}_\sigma(\widetilde{\mathcal{RAF}}'_s)$, $\forall \ell_2 \in \mathscr{L}_\sigma(std\text{-}\mathcal{AF})$,

$$\ell_1(a) = \ell_2(a)$$

Let $\ell_1 \in \mathscr{L}_\sigma(\widetilde{\mathcal{RAF}}'_s)$ be any σ-labelling of $\widetilde{\mathcal{RAF}}'_s$. Let $\ell_2 \in \mathscr{L}_\sigma(std\text{-}\mathcal{AF})$ be any σ-labelling of $std\text{-}\mathcal{AF}$. By Definition of the standard AF we have:

$$\forall \ell \in \mathscr{L}_\sigma(std\text{-}\mathcal{AF}), \forall x \in \mathcal{I},\, \ell(x) = \ell^{\mathcal{I}}(x)$$

We have so:

$$\forall x \in \mathcal{J}, \ell_2(x) = \ell^{\mathcal{J}}(x) \tag{18.127}$$

Following Definition 105 on page 155 (σ being complete-based), we have:

- $\forall (s(\alpha).\alpha) \in \mathcal{J}$ s.t. $\alpha \in Q^{inp}$,

$$\ell^{\mathcal{J}}(s(\alpha).\alpha) = \begin{cases} \texttt{in} \iff \mathcal{L}^{inp}(\alpha) = \texttt{in} \text{ and } \mathcal{L}^{inp}(s(\alpha)) = \texttt{in} \\ \texttt{out} \iff \mathcal{L}^{inp}(\alpha) = \texttt{out} \text{ or } \mathcal{L}^{inp}(s(\alpha)) = \texttt{out} \\ \texttt{und} \iff \left(\begin{array}{l} \mathcal{L}^{inp}(\alpha) \neq \texttt{out} \text{ and } \mathcal{L}^{inp}(s(\alpha)) \neq \texttt{out} \text{ and} \\ (\mathcal{L}^{inp}(\alpha) = \texttt{und} \text{ or } \mathcal{L}^{inp}(s(\alpha)) = \texttt{und}) \end{array} \right) \end{cases}$$

- $\forall \neg a \in \mathcal{J}$ s.t. $a \in S^{inp}$,

$$\ell^{\mathcal{J}}(\neg a) = \begin{cases} \texttt{in} \iff \mathcal{L}^{inp}(a) = \texttt{out} \\ \texttt{out} \iff \mathcal{L}^{inp}(a) = \texttt{in} \\ \texttt{und} \iff \mathcal{L}^{inp}(a) = \texttt{und} \end{cases}$$

Following Equation (18.127), we have so:

- $\forall (s(\alpha).\alpha) \in \mathcal{J}$ s.t. $\alpha \in Q^{inp}$,

$$\ell_2(s(\alpha).\alpha) = \begin{cases} \texttt{in} \iff \mathcal{L}^{inp}(\alpha) = \texttt{in} \text{ and } \mathcal{L}^{inp}(s(\alpha)) = \texttt{in} \\ \texttt{out} \iff \mathcal{L}^{inp}(\alpha) = \texttt{out} \text{ or } \mathcal{L}^{inp}(s(\alpha)) = \texttt{out} \\ \texttt{und} \iff \left(\begin{array}{l} \mathcal{L}^{inp}(\alpha) \neq \texttt{out} \text{ and } \mathcal{L}^{inp}(s(\alpha)) \neq \texttt{out} \text{ and} \\ (\mathcal{L}^{inp}(\alpha) = \texttt{und} \text{ or } \mathcal{L}^{inp}(s(\alpha)) = \texttt{und}) \end{array} \right) \end{cases}$$

- $\forall \neg a \in \mathcal{J}$ s.t. $a \in S^{inp}$,

$$\ell_2(\neg a) = \begin{cases} \texttt{in} \iff \mathcal{L}^{inp}(a) = \texttt{out} \\ \texttt{out} \iff \mathcal{L}^{inp}(a) = \texttt{in} \\ \texttt{und} \iff \mathcal{L}^{inp}(a) = \texttt{und} \end{cases}$$

Given that by definition $\mathcal{J} = \{s(\alpha).\alpha | \alpha \in Q^{inp}\} \cup \{\neg a | a \in S^{inp}\}$, we prove, following Assertions 1 and 2, that: $\forall a \in \mathcal{J}, \forall \ell_1 \in \mathscr{L}_\sigma(\widetilde{\mathcal{RAF}}'_s), \forall \ell_2 \in \mathscr{L}_\sigma(std\text{-}\mathcal{AF}), \ell_1(a) = \ell_2(a)$. ■

Proof of Proposition 40 on page 158. According to Assertions 1 and 2 of Lemma 12 on page 270, we have:

$$std\text{-}\mathcal{AF} \downarrow_{\omega'} = \widetilde{\mathcal{RAF}}'_s \downarrow_{\omega'} \tag{18.128}$$

According to Assertion 3 of Lemma 12 on page 270, we have:

$$K'_s \cap (\mathcal{J} \times \omega') = \tilde{K}'_s \cap (\mathcal{J} \times \omega') \tag{18.129}$$

According to Assertion 4 of Lemma 11 on page 267, we have:

$$(x,y) \in \tilde{K}'_s \cup K'_s \text{ s.t. } y \in \omega' \implies x \in \omega' \cup \mathcal{J} \tag{18.130}$$

According to Assertion 3 of Lemma 13 on page 277, we have:

$$\forall a \in \mathcal{J}, \forall \ell_1 \in \mathscr{L}_\sigma(\widetilde{\mathcal{RAF}}'_s), \forall \ell_2 \in \mathscr{L}_\sigma(std\text{-}\mathcal{AF}), \ell_1(a) = \ell_2(a) \tag{18.131}$$

Given that:

1. Following Equation (18.128) on the previous page, both AFs are identical restricted to ω'
2. Following Equations (18.129) and (18.130) on the previous page, this common subAF is attacked in both AFs identically (by same arguments and attack relations)
3. Following Equation (18.131) on the previous page, all arguments attacking this common subAF is labelled identically in both AFs and for all possible labellings
4. The semantics σ is complete-based

We have so:

$$\{\ell \downarrow_{\omega' \cup \mathcal{I}} | \ell \in \mathscr{L}_{\sigma}(\widetilde{\mathcal{RAF}}'_s)\} = \{\ell \downarrow_{\omega' \cup \mathcal{I}} | \ell \in \mathscr{L}_{\sigma}(std\text{-}\mathcal{AF})\}$$

■

Proof of Proposition 41 on page 159. Given that $\omega' \subseteq (\omega' \cup \mathcal{I})$, we have, following Equations (18.128) and (18.129) on the previous page given in the proof of Proposition 40 on page 158:

$$\{\ell \downarrow_{\omega'} | \ell \in \mathscr{L}_{\sigma}(\widetilde{\mathcal{RAF}}'_s)\} = \{\ell \downarrow_{\omega'} | \ell \in \mathscr{L}_{\sigma}(std\text{-}\mathcal{AF})\}$$

Equivalently, we have so:

$$\{\ell \downarrow_{\omega'} | \ell \in \mathscr{L}_{\sigma}(\widetilde{\mathcal{RAF}}'_s)\} = \mathscr{F}^{af}_{\sigma}(\mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}}) \tag{18.132}$$

Now, given that σ-raf is the RAF semantics corresponding to σ, we have:

$$\mathscr{L}_{\sigma\text{-}raf}(\widetilde{\mathcal{RAF}}_s) = \{\texttt{afLab2RafLab}(\ell) | \ell \in \mathscr{L}_{\sigma}(\widetilde{\mathcal{RAF}}'_s)\}$$

Or equivalently:

$$\mathscr{L}_{\sigma\text{-}raf}(\widetilde{\mathcal{RAF}}_s) = \{\left\langle \ell \downarrow_{\tilde{A}_s}, \ell \downarrow_{\tilde{K}_s} \right\rangle | \ell \in \mathscr{L}_{\sigma}(\widetilde{\mathcal{RAF}}'_s)\} \tag{18.133}$$

Given that $\tilde{A} \subseteq \tilde{A}_s$ and $\tilde{K} \subseteq \tilde{K}_s$, we have following Equation (18.133):

$$\{\mathcal{L} \downarrow_{\langle \tilde{A}, \tilde{K} \rangle} | \mathcal{L} \in \mathscr{L}_{\sigma\text{-}raf}(\widetilde{\mathcal{RAF}}_s)\} = \{\left\langle \ell \downarrow_{\tilde{A}}, \ell \downarrow_{\tilde{K}} \right\rangle | \ell \in \mathscr{L}_{\sigma}(\widetilde{\mathcal{RAF}}'_s)\}$$

Given that $\tilde{A} \cup \tilde{K} \subseteq \omega'$, we have following Equation (18.132):

$$\{\mathcal{L} \downarrow_{\langle \tilde{A}, \tilde{K} \rangle} | \mathcal{L} \in \mathscr{L}_{\sigma\text{-}raf}(\widetilde{\mathcal{RAF}}_s)\} = \{\left\langle \ell \downarrow_{\tilde{A}}, \ell \downarrow_{\tilde{K}} \right\rangle | \ell \in \mathscr{F}^{af}_{\sigma}(\mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}})\} \tag{18.134}$$

Given that $(\tilde{A} \cup \tilde{K}) \subseteq (A \cup K)$, that $\omega' \cap ((A \cup K) \setminus (\tilde{A} \cup \tilde{K})) \cap (A \cup K) = \varnothing$ and that $\mathscr{F}^{af}_{\sigma}(\mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}})$ produces labellings of arguments that are in ω', we have following Equation (18.134):

$$\{\mathcal{L} \downarrow_{\langle \tilde{A}, \tilde{K} \rangle} | \mathcal{L} \in \mathscr{L}_{\sigma\text{-}raf}(\widetilde{\mathcal{RAF}}_s)\} = \{\left\langle \ell \downarrow_{A}, \ell \downarrow_{K} \right\rangle | \ell \in \mathscr{F}^{af}_{\sigma}(\mathcal{AF} \downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}})\} \tag{18.135}$$

By Definition we have:

$$\mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp}) = \{\mathcal{L} \downarrow_{\langle \tilde{A}, \tilde{K} \rangle} | \mathcal{L} \in \mathscr{L}_{\sigma\text{-}raf}(\widetilde{\mathcal{RAF}}_s)\} \tag{18.136}$$

Thus, from Equations (18.135) and (18.136) on the previous page, we prove that:

$$\mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}, \mathfrak{I}, \mathcal{L}^{inp}) = \left\{ \left\langle \ell\downarrow_A, \ell\downarrow_K \right\rangle \middle| \ell \in \mathscr{F}^{af}_{\sigma}(\mathcal{AF}\downarrow_{\omega'}, \mathcal{I}, \ell^{\mathcal{I}}, K_{\mathcal{I}}) \right\}$$

■

Proof of Proposition 42 on page 160. Let $\mathcal{RAF} = \langle A, K, s, t\rangle$ be a RAF and $\mathcal{AF} = \texttt{Raf2Af}(\mathcal{RAF})$ be the corresponding AF of $\mathcal{RAF}$ (with $\mathcal{AF} = \langle A', K'\rangle$). Let $\Omega' \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{AF})$ be any RAF-compliant partition of A', with $\Omega' = \{\omega'_1, ..., \omega'_n\}$. Let $\Omega \in \mathscr{S}(\mathcal{RAF})$ be the partition of $\mathcal{RAF}$ corresponding to Ω', with $\Omega = \{\omega_i = \omega'_i \cap (A \cup K) | \omega'_i \in \Omega'\}$. Let $\{\widetilde{\mathcal{RAF}}_1, ..., \widetilde{\mathcal{RAF}}_n\}$ be the partition of $\mathcal{RAF}$ corresponding to Ω, with $\widetilde{\mathcal{RAF}}_i = \langle \tilde{A}_i, \tilde{K}_i, \tilde{s}_i, \tilde{t}_i, s, t\rangle$ being the partial RAF corresponding to $\omega_i \in \Omega$. Let $\mathfrak{I}_i = \left\langle S^{inp}_i, Q^{inp}_i \right\rangle$ be the input elements of $\widetilde{\mathcal{RAF}}_i$ and $\mathcal{L}^{inp}_i$ be a structure labelling of them. For $i \in \{1, ..., n\}$, let $\left\langle \widetilde{\mathcal{RAF}}_i, \mathfrak{I}_i, \mathcal{L}^{inp}_i \right\rangle$ be a RAF with input and $\left\langle \mathcal{AF}\downarrow_{\omega'_i}, \mathcal{I}_i, \ell^{\mathcal{I}_i}, K_{\mathcal{I}_i} \right\rangle$ be its corresponding AF with input, as defined in Definition 105 on page 155.

Assertion 1: σ-raf is top-down decomposable *w.r.t.* $\mathscr{S}$ *iff* σ is top-down decomposable *w.r.t.* $\mathscr{S}_{raf\text{-}c}$.

As σ-raf is the RAF semantics corresponding to σ, we have:

$$\mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF}) = \{\left\langle \ell\downarrow_A, \ell\downarrow_K \right\rangle | \ell \in \mathscr{L}_{\sigma}(\mathcal{AF})\} \tag{18.137}$$

σ is top-down decomposable *w.r.t.* $\mathscr{S}_{raf\text{-}c}$ *iff* :

$$\begin{gathered}\forall\Omega' \in \mathscr{S}_{raf\text{-}c}(\mathcal{AF}), \mathscr{L}_{\sigma}(\mathcal{AF}) \subseteq \left\{ \ell^1 \cup ... \cup \ell^n \middle| \ell^i \in \mathscr{F}^{af}_{\sigma}(\mathcal{AF}\downarrow_{\omega'_i}, \mathcal{I}_i, \ell^{\mathcal{I}_i}, K_{\mathcal{I}_i}) \right\} \\ \text{With: } \ell^{\mathcal{I}_i} = (\textstyle\bigcup_{j \in \{1,...,n\} \text{ s.t. } j \neq i} \ell^j)\downarrow_{\mathcal{I}_i}\end{gathered} \tag{18.138}$$

We have so, from Equations (18.137) and (18.138), σ being top-down decomposable *w.r.t.* $\mathscr{S}_{raf\text{-}c}$ *iff* :

$$\begin{gathered}\forall\Omega' \in \mathscr{S}_{raf\text{-}c}(\mathcal{AF}), \\ \mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF}) \subseteq \left\{ \left\langle \ell^1\downarrow_A, \ell^1\downarrow_K \right\rangle \cup ... \cup \langle \ell^n\downarrow_A, \ell^n\downarrow_K\rangle \middle| \ell^i \in \mathscr{F}^{af}_{\sigma}(\mathcal{AF}\downarrow_{\omega'_i}, \mathcal{I}_i, \ell^{\mathcal{I}_i}, K_{\mathcal{I}_i}) \right\} \\ \text{With: } \ell^{\mathcal{I}_i} = (\textstyle\bigcup_{j \in \{1,...,n\} \text{ s.t. } j \neq i} \ell^j)\downarrow_{\mathcal{I}_i}\end{gathered} \tag{18.139}$$

Following Proposition 41 on page 159, for all $i \in \{1, ..., n\}$, we have :

$$\mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_i, \mathfrak{I}_i, \mathcal{L}^{inp}_i) = \left\{ \left\langle \ell\downarrow_A, \ell\downarrow_K \right\rangle \middle| \ell \in \mathscr{F}^{af}_{\sigma}(\mathcal{AF}\downarrow_{\omega'_i}, \mathcal{I}_i, \ell^{\mathcal{I}_i}, K_{\mathcal{I}_i}) \right\} \tag{18.140}$$

Let denote $\left\langle \ell^i\downarrow_A, \ell^i\downarrow_K \right\rangle$ by $\mathcal{L}_i$, for $i \in \{1, ..., n\}$. Then, following Equations (18.139) and (18.140), we have σ being top-down decomposable *w.r.t.* $\mathscr{S}_{raf\text{-}c}$ *iff* :

$$\begin{gathered}\forall\Omega' \in \mathscr{S}_{raf\text{-}c}(\mathcal{AF}), \mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF}) \subseteq \left\{ \mathcal{L}_1 \cup ... \cup \mathcal{L}_n \middle| \mathcal{L}_i \in \mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_i, \mathfrak{I}_i, \mathcal{L}^{inp}_i) \right\} \\ \text{With: } \mathcal{L}^{inp}_i = (\textstyle\bigcup_{j \in \{1,...,n\} \text{ s.t. } j \neq i} \mathcal{L}_j)\downarrow_{\mathfrak{I}_i}\end{gathered} \tag{18.141}$$

And so:

$$\forall \Omega \in \mathscr{S}(\mathcal{RAF}),\, \mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF}) \subseteq \left\{ \mathcal{L}_1 \cup \ldots \cup \mathcal{L}_n \middle| \mathcal{L}_i \in \mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_i, \mathfrak{I}_i, \mathcal{L}^{inp}_i) \right\}$$
$$\text{With: } \mathcal{L}^{inp}_i = (\textstyle\bigcup_{j\in\{1,\ldots,n\} \text{ s.t. } j\neq i} \mathcal{L}_j) \downarrow_{\mathfrak{I}_i} \tag{18.142}$$

As a consequence, we have σ being top-down decomposable *w.r.t.* $\mathscr{S}_{raf\text{-}c}$ *iff* σ-raf is top-down decomposable *w.r.t.* $\mathscr{S}$.

Assertion 2: σ-raf is bottom-up decomposable *w.r.t.* $\mathscr{S}$ *iff* σ is bottom-up decomposable *w.r.t.* $\mathscr{S}_{raf\text{-}c}$.

As σ-raf is the RAF semantics corresponding to σ, we have:

$$\mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF}) = \{\left\langle \ell\downarrow_A, \ell\downarrow_K \right\rangle | \ell \in \mathscr{L}_{\sigma}(\mathcal{AF})\} \tag{18.143}$$

σ is bottom-up decomposable *w.r.t.* $\mathscr{S}_{raf\text{-}c}$ *iff* :

$$\forall \Omega' \in \mathscr{S}_{raf\text{-}c}(\mathcal{AF}),\, \mathscr{L}_{\sigma}(\mathcal{AF}) \supseteq \left\{ \ell^1 \cup \ldots \cup \ell^n \middle| \ell^i \in \mathscr{F}^{af}_{\sigma}(\mathcal{AF}\downarrow_{\omega'_i}, \mathcal{I}_i, \ell^{\mathcal{I}_i}, K_{\mathcal{I}_i}) \right\}$$
$$\text{With: } \ell^{\mathcal{I}_i} = (\textstyle\bigcup_{j\in\{1,\ldots,n\} \text{ s.t. } j\neq i} \ell^j) \downarrow_{\mathcal{I}_i} \tag{18.144}$$

We have so, from Equations (18.143) and (18.144) on the next page, σ being bottom-up decomposable *w.r.t.* $\mathscr{S}_{raf\text{-}c}$ *iff* :

$$\forall \Omega' \in \mathscr{S}_{raf\text{-}c}(\mathcal{AF}),$$
$$\mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF}) \supseteq \left\{ \left\langle \ell^1\downarrow_A, \ell^1\downarrow_K \right\rangle \cup \ldots \cup \langle \ell^n\downarrow_A, \ell^n\downarrow_K \rangle \middle| \ell^i \in \mathscr{F}^{af}_{\sigma}(\mathcal{AF}\downarrow_{\omega'_i}, \mathcal{I}_i, \ell^{\mathcal{I}_i}, K_{\mathcal{I}_i}) \right\} \tag{18.145}$$
$$\text{With: } \ell^{\mathcal{I}_i} = (\textstyle\bigcup_{j\in\{1,\ldots,n\} \text{ s.t. } j\neq i} \ell^j) \downarrow_{\mathcal{I}_i}$$

Following Proposition 41 on page 159, for all $i \in \{1,\ldots,n\}$, we have :

$$\mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_i, \mathfrak{I}_i, \mathcal{L}^{inp}_i) = \left\{ \left\langle \ell\downarrow_A, \ell\downarrow_K \right\rangle \middle| \ell \in \mathscr{F}^{af}_{\sigma}(\mathcal{AF}\downarrow_{\omega'_i}, \mathcal{I}_i, \ell^{\mathcal{I}_i}, K_{\mathcal{I}_i}) \right\} \tag{18.146}$$

Let denote $\left\langle \ell^i\downarrow_A, \ell^i\downarrow_K \right\rangle$ by $\mathcal{L}_i$, for $i \in \{1,\ldots,n\}$. Then, following Equations (18.145) and (18.146), we have σ being bottom-up decomposable *w.r.t.* $\mathscr{S}_{raf\text{-}c}$ *iff* :

$$\forall \Omega' \in \mathscr{S}_{raf\text{-}c}(\mathcal{AF}),\, \mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF}) \supseteq \left\{ \mathcal{L}_1 \cup \ldots \cup \mathcal{L}_n \middle| \mathcal{L}_i \in \mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_i, \mathfrak{I}_i, \mathcal{L}^{inp}_i) \right\}$$
$$\text{With: } \mathcal{L}^{inp}_i = (\textstyle\bigcup_{j\in\{1,\ldots,n\} \text{ s.t. } j\neq i} \mathcal{L}_j) \downarrow_{\mathfrak{I}_i} \tag{18.147}$$

And so:

$$\forall \Omega \in \mathscr{S}(\mathcal{RAF}),\, \mathscr{L}_{\sigma\text{-}raf}(\mathcal{RAF}) \supseteq \left\{ \mathcal{L}_1 \cup \ldots \cup \mathcal{L}_n \middle| \mathcal{L}_i \in \mathscr{F}^{raf}_{\sigma}(\widetilde{\mathcal{RAF}}_i, \mathfrak{I}_i, \mathcal{L}^{inp}_i) \right\}$$
$$\text{With: } \mathcal{L}^{inp}_i = (\textstyle\bigcup_{j\in\{1,\ldots,n\} \text{ s.t. } j\neq i} \mathcal{L}_j) \downarrow_{\mathfrak{I}_i} \tag{18.148}$$

As a consequence, we have σ being bottom-up decomposable *w.r.t.* $\mathscr{S}_{raf\text{-}c}$ *iff* σ-raf is bottom-up decomposable *w.r.t.* $\mathscr{S}$.

Assertion 3: σ-raf is fully decomposable *w.r.t.* $\mathscr{S}$ *iff* σ is fully decomposable *w.r.t.* $\mathscr{S}_{raf\text{-}c}$.

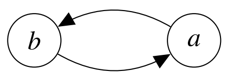
Trivial considering Assertions 1 and 2. ■

Lemmas 14 and 15 on page 285 and on page 287 pave the road for the demonstration of Propositions 43 and 45 on page 160 and on page 161.

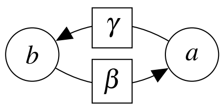
The hard demonstration part of Proposition 43 on page 160 is to show that if an AF semantics σ is *not* fully (resp. top-down, bottom-up) decomposable then σ is *not* fully (resp. top-down, bottom-up) decomposable *w.r.t.* $\mathscr{S}_{D\text{-}raf\text{-}c}$. We prove this property by choosing an AF and a partition for which σ is *not* fully (resp. top-down, bottom-up) decomposable. Then, we transform this AF into a RAF by naming its attacks. Next, we flatten this RAF into a new AF. Finally, we show with this new AF and the partition corresponding that σ is *not* fully (resp. top-down, bottom-up) decomposable *w.r.t.* $\mathscr{S}_{D\text{-}raf\text{-}c}$.

Likewise, the hard demonstration part of Proposition 45 on page 161 is to show that if an AF semantics σ is *not* fully (resp. top-down, bottom-up) decomposable *w.r.t.* $\mathscr{S}_{USCC}$ then σ is *not* fully (resp. top-down, bottom-up) decomposable *w.r.t.* $\mathscr{S}_{raf\text{-}c\text{-}USCC}$. We prove this property by the very same process, except that this time it is *w.r.t.* $\mathscr{S}_{USCC}$ and $\mathscr{S}_{raf\text{-}c\text{-}USCC}$.

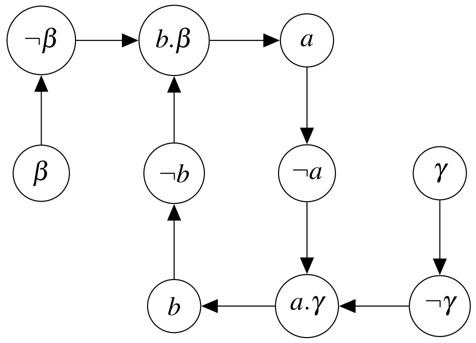
The following example illustrates what have been said above:



(a) An AF: $\mathcal{AF}_1$



(b) Its corresponding RAF: $\mathcal{RAF}_2$



(c) The flattened version of the RAF: $\mathcal{AF}_3$

Figure 18.2: Example giving an intuition for the proof of Propositions 43 and 45

Example 92. *Let consider Figure 18.2. Let* $\mathcal{AF}_1 = \langle A_1, K_1\rangle$ *be the AF illustrated in Figure 18.2(a),* $\mathcal{RAF}_2 = \langle A_2, K_2, s_2, t_2\rangle$ *be the RAF illustrated in Figure 18.2(b) such that* $\mathcal{RAF}_2 = \texttt{Af2Raf}(\mathcal{AF}_1)$ *and* $\mathcal{AF}_3 = \langle A_3, K_3\rangle$ *be the AF illustrated in Figure 18.2(c) such that* $\mathcal{AF}_3 = \texttt{Raf2Af}(\mathcal{RAF}_2)$.

Let consider the semantics preferred *, the bottom-up decomposability property and the following partition of* $\mathcal{AF}_1$*:* $\Omega_1 = \left\{\omega_1^1 = \{a\}, \omega_2^1 = \{b\}\right\}$. *We have:*

$$\mathscr{L}_{pr}(\mathcal{AF}_1) \not\supseteq \left\{\ell^1 \cup \ldots \cup \ell^n \,\middle|\, \ell^j \in \mathscr{F}^{af}_{pr}(\mathcal{AF}\downarrow_{\omega_j^1}, \mathcal{I}_j^1, \ell^{\mathcal{I}_j^1}, K_{\mathcal{I}_j^1})\right\}$$

Indeed:

$$\mathscr{L}_{pr}(\mathcal{AF}_1) = \{\{(a, in), (b, out)\}, \{(b, in), (a, out)\}\}$$

And:

$$\left\{\ell^1 \cup \ldots \cup \ell^n \middle| \ell^j \in \mathscr{F}^{af}_{pr}(\mathcal{AF}\downarrow_{\omega^1_j}, \mathcal{I}^1_j, \ell^{\mathcal{I}^1_j}, K_{\mathcal{I}^1_j})\right\}$$

$$=$$

$$\{\{(a, \mathtt{in}), (b, \mathtt{out})\}, \{(b, \mathtt{in}), (a, \mathtt{out})\}, \{(b, \mathtt{und}), (a, \mathtt{und})\}\}$$

Let consider Ω_2 the partition of $\mathcal{RAF}_2$ corresponding to Ω_1 in which all attacks are in the same part as their sources. We have: $\Omega_2 = \left\{\omega^2_1 = \{a, \gamma\}, \omega^2_2 = \{b, \beta\}\right\}$.

Now let consider $\Omega_3 \in \mathscr{S}_{raf\text{-}c\text{-}USCC}(\mathcal{AF}_3)$, the RAF-compliant partition of $\mathcal{AF}_3$ corresponding to Ω_2. We have: $\Omega_3 = \left\{\omega^3_1 = \{a, \neg a, \gamma, \neg\gamma, a.\gamma\}, \omega^3_2 = \{b, \neg b, \beta, \neg\beta, b.\beta\}\right\}$.

We can observe that:

$$\mathscr{L}_{pr}(\mathcal{AF}_3) \not\supseteq \left\{\ell^1 \cup \ldots \cup \ell^n \middle| \ell^j \in \mathscr{F}^{af}_{pr}(\mathcal{AF}\downarrow_{\omega^3_j}, \mathcal{I}^3_j, \ell^{\mathcal{I}^3_j}, K_{\mathcal{I}^3_j})\right\}$$

Indeed:

$$\mathscr{L}_{pr}(\mathcal{AF}_3) = \left\{ \begin{array}{c} \left\{ \begin{array}{c} (a, \mathtt{in}), (\neg a, \mathtt{out}), (\gamma, \mathtt{in}), (\neg\gamma, \mathtt{out}), (a.\gamma, \mathtt{in}), \\ (b, \mathtt{out}), (\neg b, \mathtt{in}), (\beta, \mathtt{in}), (\neg\beta, \mathtt{out}), (b.\beta, \mathtt{out}) \end{array} \right\}, \\ \left\{ \begin{array}{c} (a, \mathtt{out}), (\neg a, \mathtt{in}), (\gamma, \mathtt{in}), (\neg\gamma, \mathtt{out}), (a.\gamma, \mathtt{out}), \\ (b, \mathtt{in}), (\neg b, \mathtt{out}), (\beta, \mathtt{in}), (\neg\beta, \mathtt{out}), (b.\beta, \mathtt{in}) \end{array} \right\} \end{array} \right\}$$

And:

$$\left\{\ell^1 \cup \ldots \cup \ell^n \middle| \ell^j \in \mathscr{F}^{af}_{pr}(\mathcal{AF}\downarrow_{\omega^3_j}, \mathcal{I}^3_j, \ell^{\mathcal{I}^3_j}, K_{\mathcal{I}^3_j})\right\}$$

$$=$$

$$\left\{ \begin{array}{c} \left\{ \begin{array}{c} (a, \mathtt{in}), (\neg a, \mathtt{out}), (\gamma, \mathtt{in}), (\neg\gamma, \mathtt{out}), (a.\gamma, \mathtt{in}), \\ (b, \mathtt{out}), (\neg b, \mathtt{in}), (\beta, \mathtt{in}), (\neg\beta, \mathtt{out}), (b.\beta, \mathtt{out}) \end{array} \right\}, \\ \left\{ \begin{array}{c} (a, \mathtt{out}), (\neg a, \mathtt{in}), (\gamma, \mathtt{in}), (\neg\gamma, \mathtt{out}), (a.\gamma, \mathtt{out}), \\ (b, \mathtt{in}), (\neg b, \mathtt{out}), (\beta, \mathtt{in}), (\neg\beta, \mathtt{out}), (b.\beta, \mathtt{in}) \end{array} \right\}, \\ \left\{ \begin{array}{c} (a, \mathtt{und}), (\neg a, \mathtt{und}), (\gamma, \mathtt{in}), (\neg\gamma, \mathtt{out}), (a.\gamma, \mathtt{und}), \\ (b, \mathtt{und}), (\neg b, \mathtt{und}), (\beta, \mathtt{in}), (\neg\beta, \mathtt{out}), (b.\beta, \mathtt{und}) \end{array} \right\}, \end{array} \right\}$$

As a consequence the preferred *semantics is* not *bottom-up decomposable w.r.t. $\mathscr{S}_{raf\text{-}c\text{-}USCC}$.*

`Note:` *Other examples than Example 92 on the next page could be constructed for any wanted semantics, any selector, any AF and partition. Example 92 just highlights the idea behind Lemmas 14 and 15 and Propositions 43 and 45 on page 160, on page 161, on the following page and on page 287.*

Lemma 14. *Let σ be a complete-based AF semantics. Let $\mathcal{AF}_1 = \langle A_1, K_1 \rangle$ be an AF and $\Omega_1 = \left\{\omega_1^1, ..., \omega_n^1\right\}$ be any partition of $\mathcal{AF}_1$. Let $\mathcal{RAF}_2 = \texttt{Af2Raf}(\mathcal{AF}_1)$ be the non recursive RAF corresponding to $\mathcal{AF}_1$, with $\mathcal{RAF}_2 = \langle A_2, K_2, s_2, t_2 \rangle$ (see Definition 80 on page 121). Let $\Omega_2 = \left\{\omega_1^2, ..., \omega_n^2\right\}$ be the partition of $\mathcal{RAF}_2$ such that: $\forall \omega_j^2 \in \Omega_2, \omega_j^2 = \omega_j^1 \cup \left\{\alpha | \alpha \in K_2 \text{ s.t. } s(\alpha) \in \omega_j^1\right\}$. Let $\mathcal{AF}_3 = \texttt{Raf2Af}(\mathcal{RAF}_2)$ (with $\mathcal{AF}_3 = \langle A_3, K_3 \rangle$) be the AF corresponding to the flattening of $\mathcal{RAF}_2$. Let $\Omega_3 = \left\{\omega_1^3, ..., \omega_n^3\right\}$ be the partition of $\mathcal{AF}_3$ such that: $\forall \omega_j^3 \in \Omega_3, \omega_j^3 = \omega_j^2 \cup \left\{\neg x | x \in (A_2 \cup K_2) \cap \omega_j^2\right\} \cup \left\{s(\alpha).\alpha | \alpha \in K_2 \cap \omega_j^2\right\}$. The following property holds:*

$$\forall \omega_j^1 \in \Omega_1,$$

$$\left\{ \begin{array}{l|l} \ell & \\ \cup\left\{(\neg a, \mathtt{in}) | a \in \mathtt{out}(\ell)\right\} & \\ \cup\left\{(\neg a, \mathtt{out}) | a \in \mathtt{in}(\ell)\right\} & \\ \cup\left\{(\neg a, \mathtt{und}) | a \in \mathtt{und}(\ell)\right\} & \ell \in \mathscr{F}_\sigma^{af}(\mathcal{AF}_1 \downarrow_{\omega_j^1}, \mathcal{I}_j^1, \ell^{\mathcal{I}_j^1}, K_{\mathcal{I}_j^1}) \\ \cup\left\{(\alpha, \mathtt{in}) | \alpha \in K_2 \cap \omega_j^3\right\} & \\ \cup\left\{(\neg\alpha, \mathtt{out}) | \alpha \in K_2 \cap \omega_j^3\right\} & \\ \cup\left\{(s(\alpha).\alpha, \ell(s(\alpha))) | \alpha \in K_2 \cap \omega_j^3\right\} & \end{array} \right\} = \mathscr{F}_\sigma^{af}(\mathcal{AF}_3 \downarrow_{\omega_j^3}, \mathcal{I}_j^3, \ell^{\mathcal{I}_j^3}, K_{\mathcal{I}_j^3})$$

With:

- $\mathcal{I}_j^3 = \left\{s(\alpha).\alpha | \alpha \in K_2 \text{ s.t. } t(\alpha) \in \omega_j^3 \text{ and } s(\alpha).\alpha \notin \omega_j^3\right\}$
- $K_{\mathcal{I}_j^3} = \left\{(s(\alpha).\alpha, t(\alpha)) | \alpha \in K_2 \text{ s.t. } t(\alpha) \in \omega_j^3 \text{ and } s(\alpha).\alpha \notin \omega_j^3\right\}$
- $\ell^{\mathcal{I}_j^3}$ *being a labelling of* $\mathcal{I}_j^3$ *defined as:* $\forall \alpha = (a, b) \in K_{\mathcal{I}_j^1}, \ell^{\mathcal{I}_j^3}(s(\alpha).\alpha) = \ell^{\mathcal{I}_j^1}(a)$

Proof of Lemma 14. Given that all attacks in $\mathcal{RAF}_2$ are valid and that σ is complete-based, we have:

$$\forall \ell' \in \mathscr{F}_\sigma^{af}(\mathcal{AF}_3 \downarrow_{\omega_j^3}, \mathcal{I}_j^3, \ell^{\mathcal{I}_j^3}, K_{\mathcal{I}_j^3}), \forall \alpha \in (K_2 \cap \omega_j^3), \begin{cases} \ell'(\alpha) = \mathtt{in} \\ \ell'(\neg\alpha) = \mathtt{out} \\ \ell'(s(\alpha).\alpha) = \ell(s(\alpha)) \end{cases} \tag{18.149}$$

Following the definition of Raf2Af and given that σ is complete-based, we have:

$$\forall \ell' \in \mathscr{F}^{af}_{\sigma}(\mathcal{AF}_3 \downarrow_{\omega^3_j}, \mathcal{I}^3_j, \ell^{\mathcal{I}^3_j}, K_{\mathcal{I}^3_j}), \forall a \in (A_2 \cap \omega^3_j) \begin{cases} \ell'(\neg a) = \mathtt{in} \iff \ell'(a) = \mathtt{out} \\ \ell'(\neg a) = \mathtt{out} \iff \ell'(a) = \mathtt{in} \\ \ell'(\neg a) = \mathtt{und} \iff \ell'(a) = \mathtt{und} \end{cases} \tag{18.150}$$

Given that $\forall \alpha = (a,b) \in K_{\mathcal{I}^1_j}, \ell^{\mathcal{I}^3_j}(s(\alpha).\alpha) = \ell^{\mathcal{I}^1_j}(a)$, we have so:

$$\forall \omega_j \in \Omega,$$

$$\left\{ \begin{array}{l|l} \ell & \\ \cup \left\{(\neg a, \mathtt{in}) | a \in \mathtt{out}(\ell)\right\} & \\ \cup \left\{(\neg a, \mathtt{out}) | a \in \mathtt{in}(\ell)\right\} & \\ \cup \left\{(\neg a, \mathtt{und}) | a \in \mathtt{und}(\ell)\right\} & \ell \in \mathscr{F}^{af}_{\sigma}(\mathcal{AF}_1 \downarrow_{\omega^1_j}, \mathcal{I}^1_j, \ell^{\mathcal{I}^1_j}, K_{\mathcal{I}^1_j}) \\ \cup \left\{(\alpha, \mathtt{in}) | \alpha \in K_2 \cap \omega^3_j\right\} & \\ \cup \left\{(\neg\alpha, \mathtt{out}) | \alpha \in K_2 \cap \omega^3_j\right\} & \\ \cup \left\{(s(\alpha).\alpha, \ell(s(\alpha))) | \alpha \in K_2 \cap \omega^3_j\right\} & \end{array} \right\} = \mathscr{F}^{af}_{\sigma}(\mathcal{AF}_3 \downarrow_{\omega^3_j}, \mathcal{I}^3_j, \ell^{\mathcal{I}^3_j}, K_{\mathcal{I}^3_j})$$

■

Lemma 15. *Let σ be a complete-based AF semantics. Let $\mathcal{AF}_1 = \langle A_1, K_1 \rangle$ be an AF and $\Omega_1 = \left\{\omega^1_1, ..., \omega^1_n\right\}$ be any partition of $\mathcal{AF}_1$. Let $\mathcal{RAF}_2 =$ Af2Raf$(\mathcal{AF}_1)$ be the non recursive RAF corresponding to $\mathcal{AF}_1$, with $\mathcal{RAF}_2 = \langle A_2, K_2, s_2, t_2 \rangle$ (see Definition 80 on page 121). Let $\Omega_2 = \left\{\omega^2_1, ..., \omega^2_n\right\}$ be the partition of $\mathcal{RAF}_2$ such that: $\forall \omega^2_j \in \Omega_2, \omega^2_j = \omega^1_j \cup \left\{\alpha | \alpha \in K_2 \text{ s.t. } s(\alpha) \in \omega^1_j\right\}$. Let $\mathcal{AF}_3 =$ Raf2Af$(\mathcal{RAF}_2)$ (with $\mathcal{AF}_3 = \langle A_3, K_3 \rangle$) be the AF corresponding to the flattening of $\mathcal{RAF}_2$. Let $\Omega_3 = \left\{\omega^3_1, ..., \omega^3_n\right\}$ be the partition of $\mathcal{AF}_3$ such that: $\forall \omega^3_j \in \Omega_3, \omega^3_j = \omega^2_j \cup \left\{\neg x | x \in (A_2 \cup K_2) \cap \omega^2_j\right\} \cup \left\{s(\alpha).\alpha | \alpha \in K_2 \cap \omega^2_j\right\}$. Let $\mathcal{R} \in \{\subseteq, \supseteq, =\}$ be a binary relation over sets. The following property holds:*

$$\mathscr{L}_{\sigma}(\mathcal{AF}_1)\ \mathcal{R}\ \left\{\ell^1 \cup ... \cup \ell^n \middle| \ell^j \in \mathscr{F}^{af}_{\sigma}(\mathcal{AF}_1 \downarrow_{\omega^1_j}, \mathcal{I}^1_j, \ell^{\mathcal{I}^1_j}, K_{\mathcal{I}^1_j})\right\}$$

$$\iff$$

$$\mathscr{L}_{\sigma}(\mathcal{AF}_3)\ \mathcal{R}\ \left\{\ell^1 \cup ... \cup \ell^n \middle| \ell^j \in \mathscr{F}^{af}_{\sigma}(\mathcal{AF}_3 \downarrow_{\omega^3_j}, \mathcal{I}^3_j, \ell^{\mathcal{I}^3_j}, K_{\mathcal{I}^3_j})\right\}$$

With:

- $\mathcal{I}^3_j = \left\{s(\alpha).\alpha | \alpha \in K_2 \text{ s.t. } t(\alpha) \in \omega^3_j \text{ and } s(\alpha).\alpha \notin \omega^3_j\right\}$

- $K_{\mathcal{J}_j^3} = \left\{(s(\alpha).\alpha, t(\alpha)) | \alpha \in K_2 \textit{ s.t. } t(\alpha) \in \omega_j^3 \textit{ and } s(\alpha).\alpha \notin \omega_j^3\right\}$
- $\ell^{\mathcal{J}_j^3}$ *being a labelling of* $\mathcal{J}_j^3$

`Note:` Unlike in Lemma 14 on the previous page, there is no constraint on $\ell^{\mathcal{J}_j^3}$ following the definition of the decomposability of an AF semantics. See Definitions 19 and 22 on page 19.

Proof of Lemma 15. Given that σ is complete-based, we have:

$$\forall \ell' \in \mathcal{L}_\sigma(\mathcal{AF}_3), \forall x \in (A_2 \cup K_2), \begin{cases} \ell'(\neg x) = \mathtt{in} \iff \ell'(x) = \mathtt{out} \\ \ell'(\neg x) = \mathtt{out} \iff \ell'(x) = \mathtt{in} \\ \ell'(\neg x) = \mathtt{und} \iff \ell'(x) = \mathtt{und} \end{cases} \tag{18.151}$$

Furthermore, given that all attacks in $\mathcal{RAF}_2$ are valid and that σ is complete-based, we have:

$$\forall \ell' \in \mathcal{L}_\sigma(\mathcal{AF}_3), \forall \alpha \in K_2, \begin{cases} \ell'(\alpha) = \mathtt{in} \\ \ell'(\neg\alpha) = \mathtt{out} \\ \ell'(s(\alpha).\alpha) = \ell(s(\alpha)) \end{cases} \tag{18.152}$$

From Equations (18.151) and (18.152), from the definition of `Raf2Af` and from the fact that σ is complete-based, we have:

$$\left\{ \begin{array}{l|l} \ell & \\ \cup\{(\neg a, \mathtt{in}) | a \in \mathtt{out}(\ell)\} & \\ \cup\{(\neg a, \mathtt{out}) | a \in \mathtt{in}(\ell)\} & \\ \cup\{(\neg a, \mathtt{und}) | a \in \mathtt{und}(\ell)\} & \ell \in \mathcal{L}_\sigma(\mathcal{AF}_1) \\ \cup\{(\alpha, \mathtt{in}) | \alpha \in K_2\} & \\ \cup\{(\neg\alpha, \mathtt{out}) | \alpha \in K_2\} & \\ \cup\{(s(\alpha).\alpha, \ell(s(\alpha))) | \alpha \in K_2\} & \end{array} \right\} = \mathcal{L}_\sigma(\mathcal{AF}_3) \tag{18.153}$$

Now, from Lemma 14 on page 285, we have:

$$\left\{ \begin{array}{l|l} \ell & \\ \cup\left\{(\neg a, \mathtt{in})|a \in \mathtt{out}(\ell)\right\} & \\ \cup\left\{(\neg a, \mathtt{out})|a \in \mathtt{in}(\ell)\right\} & \\ \cup\left\{(\neg a, \mathtt{und})|a \in \mathtt{und}(\ell)\right\} & \ell \in \left\{\ell^1 \cup \ldots \cup \ell^n \middle| \ell^j \in \mathscr{F}_{\sigma}^{af}(\mathcal{AF}_1 \downarrow_{\omega_j^1}, \mathcal{I}_j^1, \ell^{\mathcal{I}_j^1}, K_{\mathcal{I}_j^1})\right\} \\ \cup\{(\alpha, \mathtt{in})|\alpha \in K_2\} & \\ \cup\{(\neg\alpha, \mathtt{out})|\alpha \in K_2\} & \\ \cup\left\{(s(\alpha).\alpha, \ell(s(\alpha)))|\alpha \in K_2\right\} & \end{array} \right\} \tag{18.154}$$

$$=$$

$$\left\{\ell^1 \cup \ldots \cup \ell^n \middle| \ell^j \in \mathscr{F}_{\sigma}^{af}(\mathcal{AF}_3 \downarrow_{\omega_j^3}, \mathcal{I}_j^3, \ell^{\mathcal{I}_j^3}, K_{\mathcal{I}_j^3})\right\}$$

Finally, from Equations (18.153) and (18.154) on the previous page and on this page, we prove so that:

$$\mathscr{L}_{\sigma}(\mathcal{AF}_1)\ \mathfrak{R}\ \left\{\ell^1 \cup \ldots \cup \ell^n \middle| \ell^j \in \mathscr{F}_{\sigma}^{af}(\mathcal{AF}_1 \downarrow_{\omega_j^1}, \mathcal{I}_j^1, \ell^{\mathcal{I}_j^1}, K_{\mathcal{I}_j^1})\right\}$$

$$\Longleftrightarrow$$

$$\mathscr{L}_{\sigma}(\mathcal{AF}_3)\ \mathfrak{R}\ \left\{\ell^1 \cup \ldots \cup \ell^n \middle| \ell^j \in \mathscr{F}_{\sigma}^{af}(\mathcal{AF}_3 \downarrow_{\omega_j^3}, \mathcal{I}_j^3, \ell^{\mathcal{I}_j^3}, K_{\mathcal{I}_j^3})\right\}$$

■

Proof of Proposition 43 on page 160. Trivially we have:

$$\sigma \text{ is top-down (resp. bottom-up, fully) decomposable}$$

$$\Longrightarrow \tag{18.155}$$

$$\sigma \text{ is top-down (resp. bottom-up, fully) decomposable } \textit{w.r.t. } \mathscr{S}_{D\text{-}raf\text{-}c}$$

Let prove that the reciprocal proposition is also true.

Let σ be an AF complete-based semantics that is *not* top-down (resp. bottom-up, fully) decomposable. With $\mathfrak{R}$ a binary relation over sets being respectively "$\subseteq$", "$\supseteq$", "$=$", we have so:

$$\begin{gathered} \exists \mathcal{AF} \in \Phi_{af} \text{ and } \Omega, \text{ a partition of } \mathcal{AF} \text{ s.t.} \\ \mathscr{L}_{\sigma}(\mathcal{AF})\, \not{\mathfrak{R}}\, \left\{\ell^1 \cup \ldots \cup \ell^n \middle| \ell^i \in \mathscr{F}_{\sigma}^{af}(\mathcal{AF} \downarrow_{\omega_i}, \mathcal{I}_i, \ell^{\mathcal{I}_i}, K_{\mathcal{I}_i})\right\} \end{gathered} \tag{18.156}$$

Let $\mathcal{AF}_1 = \langle A_1, K_1 \rangle$ be an AF and $\Omega_1 = \left\{\omega_1^1, ..., \omega_n^1\right\}$ be any partition of $\mathcal{AF}_1$ such that they satisfy Equation (18.156) on the previous page. Let $\mathcal{RAF}_2 = \texttt{Af2Raf}(\mathcal{AF}_1)$ be the non recursive RAF corresponding to $\mathcal{AF}_1$, with $\mathcal{RAF}_2 = \langle A_2, K_2, s_2, t_2 \rangle$ (see Definition 80 on page 121). Let $\Omega_2 = \left\{\omega_1^2, ..., \omega_n^2\right\}$ be the partition of $\mathcal{RAF}_2$ such that: $\forall \omega_j^2 \in \Omega_2, \omega_j^2 = \omega_j^1 \cup \left\{\alpha | \alpha \in K_2 \text{ s.t. } s(\alpha) \in \omega_j^1\right\}$. Let $\mathcal{AF}_3 = \texttt{Raf2Af}(\mathcal{RAF}_2)$ (with $\mathcal{AF}_3 = \langle A_3, K_3 \rangle$) be the AF corresponding to the flattening of $\mathcal{RAF}_2$. Let $\Omega_3 = \left\{\omega_1^3, ..., \omega_n^3\right\}$ be the partition of $\mathcal{AF}_3$ such that: $\forall \omega_j^3 \in \Omega_3, \omega_j^3 = \omega_j^2 \cup \left\{\neg x | x \in (A_2 \cup K_2) \cap \omega_j^2\right\} \cup \left\{s(\alpha).\alpha | \alpha \in K_2 \cap \omega_j^2\right\}$.

Following Lemma 15 on page 287, we have:

$$\mathcal{L}_\sigma(\mathcal{AF}_3) \mathcal{R} \left\{\ell^1 \cup ... \cup \ell^n \middle| \ell^i \in \mathcal{F}_\sigma^{af}(\mathcal{AF}_3 \downarrow_{\omega_j^3}, \mathcal{I}_j^3, \ell^{\mathcal{I}_j^3}, K_{\mathcal{I}_j^3})\right\} \tag{18.157}$$

Notice that Ω_3 is a RAF-compliant partition of $\mathcal{AF}_3$ (*i.e.* $\Omega_3 \in \mathcal{S}_{D\text{-}raf\text{-}c}(\mathcal{AF}_3)$). As a consequence, if σ is *not* top-down (resp. bottom-up, fully) decomposable then the following statement holds:

$$\begin{gathered} \exists \mathcal{AF} \in \{\texttt{Raf2Af}(\mathcal{RAF}) | \mathcal{RAF} \in \Phi_{raf}\} \text{ and } \Omega \in \mathcal{S}_{D\text{-}raf\text{-}c}(\mathcal{AF}) \text{ s.t.} \\ \mathcal{L}_\sigma(\mathcal{AF}) \mathcal{R} \left\{\ell^1 \cup ... \cup \ell^n \middle| \ell^i \in \mathcal{F}_\sigma^{af}(\mathcal{AF} \downarrow_{\omega_i}, \mathcal{I}_i, \ell^{\mathcal{I}_i}, K_{\mathcal{I}_i})\right\} \end{gathered} \tag{18.158}$$

Considering the contrapositive of the previous implication, we prove so that:

$$\begin{gathered} \sigma \text{ is top-down (resp. bottom-up, fully) decomposable } \textit{w.r.t. } \mathcal{S}_{D\text{-}raf\text{-}c} \\ \implies \\ \sigma \text{ is top-down (resp. bottom-up, fully) decomposable} \end{gathered} \tag{18.159}$$

■

Proof of Proposition 44 on page 160. Trivial considering Proposition 43 on page 160 and the decomposability properties of AF semantics shown in Table 2.1 on page 22. ■

Proof of Proposition 45 on page 161. Following the definitions of $\mathcal{S}_{USCC}$ (See Definition 27 on page 21) and of $\mathcal{S}_{raf\text{-}c\text{-}USCC}$ (See Definition 109 on page 161), we have:

$$\forall \mathcal{RAF} \in \Phi_{raf}, \mathcal{S}_{raf\text{-}c\text{-}USCC}(\texttt{Raf2Af}(\mathcal{RAF})) \subseteq \mathcal{S}_{USCC}(\texttt{Raf2Af}(\mathcal{RAF})) \tag{18.160}$$

That is, for any RAF $\mathcal{RAF} = \langle A, K, s, t \rangle$, each partition of $\texttt{Raf2Af}(\mathcal{RAF})$ produced by $\mathcal{S}_{raf\text{-}c\text{-}USCC}$ is also a partition of $\texttt{Raf2Af}(\mathcal{RAF})$ produced by $\mathcal{S}_{USCC}$.

It follows that:

$$\begin{gathered} \sigma \text{ is top-down (resp. bottom-up, fully) decomposable } \textit{w.r.t. } \mathcal{S}_{USCC} \\ \implies \\ \sigma \text{ is top-down (resp. bottom-up, fully) decomposable } \textit{w.r.t. } \mathcal{S}_{raf\text{-}c\text{-}USCC} \end{gathered} \tag{18.161}$$

Let prove that the reciprocal proposition is also true.

Let σ be an AF complete-based semantics that is *not* top-down (resp. bottom-up, fully) decomposable *w.r.t.* $\mathscr{S}_{USCC}$. With $\mathcal{R}$ a binary relation over sets being respectively "$\subseteq$", "$\supseteq$", "$=$", we have so:

$$\begin{gathered}\exists \mathcal{AF} \in \Phi_{af} \text{ and } \Omega \in \mathscr{S}_{USCC}(\mathcal{AF}) \text{ s.t.} \\ \mathscr{L}_\sigma(\mathcal{AF})\,\cancel{\mathcal{R}}\left\{\ell^1 \cup \ldots \cup \ell^n \middle| \ell^i \in \mathscr{F}^{af}_\sigma(\mathcal{AF}\downarrow_{\omega_i}, \mathcal{I}_i, \ell^{\mathcal{I}_i}, K_{\mathcal{I}_i})\right\}\end{gathered} \tag{18.162}$$

Let $\mathcal{AF}_1 = \langle A_1, K_1 \rangle$ be an AF and $\Omega_1 \in \mathscr{S}_{USCC}(\mathcal{AF}_1)$ (with $\Omega_1 = \left\{\omega_1^1, \ldots, \omega_n^1\right\}$) be a partition of $\mathcal{AF}_1$ such that they satisfy Equation (18.162). Let $\mathcal{RAF}_2 = \mathtt{Af2Raf}(\mathcal{AF}_1)$ be the non recursive RAF corresponding to $\mathcal{AF}_1$, with $\mathcal{RAF}_2 = \langle A_2, K_2, s_2, t_2 \rangle$ (see Definition 80 on page 121). Let $\Omega_2 = \left\{\omega_1^2, \ldots, \omega_n^2\right\}$ be the partition of $\mathcal{RAF}_2$ such that: $\forall \omega_j^2 \in \Omega_2, \omega_j^2 = \omega_j^1 \cup \left\{\alpha | \alpha \in K_2 \text{ s.t. } s(\alpha) \in \omega_j^1\right\}$. Let $\mathcal{AF}_3 = \mathtt{Raf2Af}(\mathcal{RAF}_2)$ (with $\mathcal{AF}_3 = \langle A_3, K_3 \rangle$) be the AF corresponding to the flattening of $\mathcal{RAF}_2$. Let $\Omega_3 = \left\{\omega_1^3, \ldots, \omega_n^3\right\}$ be the partition of $\mathcal{AF}_3$ such that: $\forall \omega_j^3 \in \Omega_3, \omega_j^3 = \omega_j^2 \cup \left\{\neg x | x \in (A_2 \cup K_2) \cap \omega_j^2\right\} \cup \left\{s(\alpha).\alpha | \alpha \in K_2 \cap \omega_j^2\right\}$.

Following Lemma 15 on page 287, we have:

$$\mathscr{L}_\sigma(\mathcal{AF}_3)\,\cancel{\mathcal{R}}\left\{\ell^1 \cup \ldots \cup \ell^n \middle| \ell^i \in \mathscr{F}^{af}_\sigma(\mathcal{AF}_3\downarrow_{\omega_j^3}, \mathcal{I}_j^3, \ell^{\mathcal{I}_j^3}, K_{\mathcal{I}_j^3})\right\} \tag{18.163}$$

Notice that Ω_3 is a RAF-compliant partition of $\mathcal{AF}_3$ (*i.e.* $\Omega_3 \in \mathscr{S}_{D\text{-}raf\text{-}c}(\mathcal{AF}_3)$). As a consequence, following the definition of Ω_1, of Ω_2 and of Ω_3, we have: $\Omega_3 \in \mathscr{S}_{raf\text{-}c\text{-}USCC}(\mathcal{AF}_3)$.

It follows then that, if σ is *not* top-down (resp. bottom-up, fully) decomposable *w.r.t.* $\mathscr{S}_{USCC}$, the following statement holds:

$$\begin{gathered}\exists \mathcal{AF} \in \{\mathtt{Raf2Af}(\mathcal{RAF}) | \mathcal{RAF} \in \Phi_{raf}\} \text{ and } \Omega \in \mathscr{S}_{raf\text{-}c\text{-}USCC}(\mathcal{AF}) \text{ s.t.} \\ \mathscr{L}_\sigma(\mathcal{AF})\,\cancel{\mathcal{R}}\left\{\ell^1 \cup \ldots \cup \ell^n \middle| \ell^i \in \mathscr{F}^{af}_\sigma(\mathcal{AF}\downarrow_{\omega_i}, \mathcal{I}_i, \ell^{\mathcal{I}_i}, K_{\mathcal{I}_i})\right\}\end{gathered} \tag{18.164}$$

Considering the contrapositive of the previous implication, we prove so that:

$$\begin{gathered}\sigma \text{ is top-down (resp. bottom-up, fully) decomposable } \textit{w.r.t. } \mathscr{S}_{raf\text{-}c\text{-}USCC} \\ \Longrightarrow \\ \sigma \text{ is top-down (resp. bottom-up, fully) decomposable } \textit{w.r.t. } \mathscr{S}_{USCC}\end{gathered} \tag{18.165}$$

■

Proof of Proposition 46 on page 161. Trivial considering Proposition 45 on page 161 and the decomposability properties of AF semantics shown in Table 2.1 on page 22. ■

Proof of Proposition 47 on page 162. Trivial considering Propositions 43 and 46 on page 160 and on page 161 and the decomposability properties of AF semantics shown in Table 2.1 on page 22. ■

Bibliography

[1] Gianvincenzo Alfano, Sergio Greco, and Francesco Parisi. Efficient computation of extensions for dynamic abstract argumentation frameworks: An incremental approach. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, pages 49–55, 2017.

[2] Mario Alviano. Ingredients of the argumentation reasoner pyglaf: Python, circumscription, and glucose to taste. In *RCRA@ AI* IA*, pages 1–16, 2017.

[3] Mario Alviano. The pyglaf argumentation reasoner. In *OASIcs-OpenAccess Series in Informatics*, volume 58. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[4] Leila Amgoud, Jonathan Ben-Naim, Dragan Doder, and Srdjan Vesic. Acceptability semantics for weighted argumentation frameworks. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, volume 2017, 2017.

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

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

[7] P. Baroni and M. Giacomin. Refining SCC decomposition in argumentation semantics : a first investigation. Citeseer, 2006.

[8] Pietro Baroni, Guido Boella, Federico Cerutti, Massimiliano Giacomin, Leendert Van Der Torre, and Serena Villata. On the input/output behavior of argumentation frameworks. *Artificial Intelligence*, 217:144–197, 2014.

[9] Pietro Baroni, Guido Boella, Federico Cerutti, Massimiliano Giacomin, Leendert W. N. van der Torre, and Serena Villata. On input/output argumentation frameworks. In *COMMA*, pages 358–365, 2012.

[10] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *Knowledge Eng. Review*, 26(4):365–410, 2011.

[11] Pietro Baroni, Massimiliano Giacomin, and Giovanni Guida. SCC-recursiveness: a general schema for argumentation semantics. *Artificial Intelligence*, 168(1-2):162–210, 2005.

[12] H. Barringer, D. M. Gabbay, and J. Woods. Temporal dynamics of support and attack networks : From argumentation to zoology. In D. Hutter and W. Stephan, editors, *Mechanizing Mathematical Reasoning, Essays in Honor of J. H. Siekmann on the Occasion of His 60th Birthday. LNAI 2605*, pages 59–98. Springer Verlag, 2005.

[13] Philippe Besnard, Claudette Cayrol, and Marie-Christine Lagasquie-Schiex. Logical theories and abstract argumentation: A survey of existing works. *Argument and Computation*, 11(1-2):41–102, May 2020.

[14] Martin Caminada. On the issue of reinstatement in argumentation. In *JELIA*, pages 111–123, 2006.

[15] Álvaro Carrera and Carlos A Iglesias. A systematic review of argumentation techniques for multi-agent systems research. *Artificial Intelligence Review*, 44(4):509–535, 2015.

[16] C. Cayrol, A. Cohen, and M-C. Lagasquie-Schiex. Towards a new framework for recursive interactions in abstract bipolar argumentation. In *Proc. of COMMA*, pages 191–198. IOS Press, 2016.

[17] C. Cayrol, J. Fandinno, L. Fariñas del Cerro, and M.-C. Lagasquie-Schiex. Valid attacks in Argumentation Frameworks with Recursive Attacks (long version). Rapport de recherche IRIT/RR–2019–02–FR, IRIT, 2019.

[18] Claudette Cayrol, Jorge Fandinno, Luis Fariñas del Cerro, and Marie-Christine Lagasquie-Schiex. Valid attacks in argumentation frameworks with recursive attacks. *Annals of Mathematics and Artificial Intelligence*, 89(1):53–101, November 2020.

[19] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. Logical Encoding of Argumentation Frameworks with Higher-order Attacks and Evidential Supports. *International Journal on Artificial Intelligence Tools*, 29(03n04):2060003, June 2020.

[20] F. Cerutti, M. Giacomin, and M. Vallati. Exploiting planning problems for generating challenging abstract Arg. Frameworks. *Second ICCMA edition (2017). http://argumentationcompetition.org/2017/Planning2AF.pdf.*

[21] F. Cerutti, M. Giacomin, and M. Vallati. Generating structured argumentation frameworks: Afbenchgen2. In *COMMA*, pages 467–468, 2016.

[22] F. Cerutti, M. Vallati, M. Giacomin, and T. Zanetti. ArgSemSAT-2017. Second ICCMA edition (2017). http://argumentationcompetition.org/2017/ArgSemSAT.pdf.

[23] Federico Cerutti, Paul E Dunne, Massimiliano Giacomin, and Mauro Vallati. A SAT-based approach for computing extensions in abstract argumentation. In *Second International Workshop on Theory and Applications of Formal Argumentation (TAFA-13)*, 2013.

[24] Federico Cerutti, Massimiliano Giacomin, Mauro Vallati, and Marina Zanella. An SCC recursive meta-algorithm for computing preferred labellings in abstract argumentation. In Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press, 2014.

[25] Federico Cerutti, Ilias Tachmazidis, Mauro Vallati, Sotirios Batsakis, Massimiliano Giacomin, and Grigoris Antoniou. Exploiting parallelism for hard problems in abstract argumentation. In *AAAI*, pages 1475–1481, 2015.

[26] Günther Charwat, Wolfgang Dvořák, Sarah A Gaggl, Johannes P Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation–a survey. *Artificial intelligence*, 220:28–63, 2015.

[27] OHAAI Collaboration, Federico Castagna, Timotheus Kampik, Atefeh Keshavarzi Zafarghandi, Mickaël Lafages, Jack Mumford, Christos T. Rodosthenous, Samy Sá, Stefan Sarkadi, Joseph Singleton, Kenneth Skiba, and Andreas Xydis. Online handbook of argumentation for ai: Volume 1, 2020.

[28] Stephen A Cook. An overview of computational complexity. *Communications of the ACM*, 26(6):400–408, 1983.

[29] Sylvie Coste-Marquis, Sébastien Konieczny, Pierre Marquis, and Mohand Akli Ouali. Weighted attacks in argumentation frameworks. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*. AAAI Press, 2012.

[30] M. Diller. Traffic networks become argumentation frameworks. *Second ICCMA edition (2017). http://argumentationcompetition.org/2017/Traffic.pdf.*

[31] S. Doutre, M. Lafages, and M-C. Lagasquie-Schiex. A distributed and clustering-based algorithm for the enumeration problem in abstract argumentation. In *Proc. of PRIMA*, pages 87–105. Springer, 2019.

[32] Sylvie Doutre, Mickaël Lafages, and Marie-Christine Lagasquie-Schiex. A Distributed and Clustering-based Algorithm for the Enumeration Problem in Abstract Argumentation (JIAF 2020). In *14èmes Journées d'Intelligence Artificielle Fondamentale (JIAF 2020)*, Actes des JIAF 2020, pages 99–108, Angers, France, 2020. AFIA.

[33] Sylvie Doutre, Mickaël Lafages, and Marie-Christine Lagasquie-Schiex. Argumentation Frameworks with Higher-Order Attacks: Complexity results. Research Report IRIT/RR–2020–03–FR, Institut recherche en informatique de toulouse (IRIT), 2020.

[34] Sylvie Doutre, Mickaël Lafages, and Marie-Christine Lagasquie-Schiex. Argumentation Frameworks with Higher-Order Attacks: Labelling Semantics. Research Report IRIT/RR–2020–01–FR, IRIT - Institut de recherche en informatique de Toulouse, 2020.

[35] Sylvie Doutre, Mickaël Lafages, and Marie-Christine Lagasquie-Schiex. Argumentation Frameworks with Higher-Order Attacks: Labellings and Complexity. In Miltos Alamaniotis and Shimei Pan, editors, *32nd International Conference on Tools with Artificial Intelligence (ICTAI 2020)*, Proceedings of ICTAI 2020, pages 1210–1217, Chicago (virtual conference), United States, November 2020. IEEE, IEEE.

[36] Sylvie Doutre, Mickaël Lafages, and Marie-Christine Lagasquie-Schiex. Argumentation Frameworks with Higher-Order Attacks: Semantics and Complexity. 17th International Conference on Principles of Knowledge Representation and Reasoning, September 2020. Poster.

[37] Sylvie Doutre, Mickaël Lafages, and Marie-Christine Lagasquie-Schiex. AFDivider: Manual and Documentation. Research Report IRIT/RR–2022–02–FR, IRIT - Institut de recherche en informatique de Toulouse, 2022.

[38] Sylvie Doutre and Jérôme Mengin. Preferred extensions of argumentation frameworks: Query, answering, and computation. In *International Joint Conference on Automated Reasoning*, pages 272–288. Springer, 2001.

[39] Phan Minh Dung. 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.

[40] Wolfgang Dvořák and Paul E Dunne. Computational problems in formal argumentation and their complexity. *Journal of Logics and their Applications*, 4(8):2557–2622, 2017.

[41] Wolfgang Dvorak and Paul E. Dunne. Computational problems in formal argumentation and their complexity. In *Handbook of formal argumentation*, pages 631–688. College publication, 2018.

[42] Wolfgang Dvorák, Matti Järvisalo, and Johannes P Wallner. Cegartix v2017-3-13: A SAT-based counter-example guided argumentation reasoning tool. *Second ICCMA*, 2017.

[43] Wolfgang Dvořák, Reinhard Pichler, and Stefan Woltran. Towards fixed-parameter tractable algorithms for abstract argumentation. *Artificial Intelligence*, 186:1–37, 2012.

[44] Wolfgang Dvorák, Anna Rapberger, Johannes P Wallner, and Stefan Woltran. Aspartix-v19-system description for iccma'19. *Third ICCMA edition (2019). http://argumentationcompetition.org/2019/papers/ICCMA19_paper_7.pdf.*

[45] Matthew Foreman and Akihiro Kanamori. *Handbook Set Theory*. Springer, 2006.

[46] D. M. Gabbay. Semantics for higher level attacks in extended argumentation frames. *Studia Logica*, 93:357–381, 2009.

[47] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, 1996.

[48] Jonathan L. Gross, Jay Yellen, and Ping Zhang. *Handbook of Graph Theory, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2013.

[49] Mickaël Lafages, Sylvie Doutre, and Marie-Christine Lagasquie-Schiex. Clustering and distributed computing in abstract argumentation. Rapport de recherche IRIT/RR–2018–11–FR, IRIT, Université Paul Sabatier, Toulouse, décembre 2018.

[50] Jean-Marie Lagniez, Emmanuel Lonca, and Jean-Guy Mailly. Coquiaas v3. 0 iccma 2019 solver description. *System descriptions of the Third International Competition on Computational Models of Argumentation (ICCMA'19)*, 2019.

[51] Daniel Le Berre and Stéphanie Roussel. Sat4j 2.3.2: on the fly solver configuration. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(3-4):197–202, 2012.

[52] Beishui Liao. Toward incremental computation of argumentation semantics: A decomposition-based approach. *Annals of Mathematics and Artificial Intelligence*, 67(3-4):319–358, 2013.

[53] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[54] Fragkiskos D Malliaros and Michalis Vazirgiannis. Clustering and community detection in directed networks: A survey. *Physics Reports*, 533(4):95–142, 2013.

[55] Lars Malmqvist. Yonas: An experimental neural argumentation solver. *International Competition on Computational Models of Argumentation (ICCMA)*, 2019.

[56] Keith Robert Matthews. *Elementary linear algebra*. University of Queenland, 2013.

[57] S. Modgil. An abstract theory of argumentation that accommodates defeasible reasoning about preferences. In *Proc. of ECSQARU*, pages 648–659, 2007.

[58] S. Modgil. Reasoning about preferences in argumentation frameworks. *Artificial Intelligence*, 173:901–934, 2009.

[59] A. Niskanen and M. Järvisalo. μ-toksia (version 2019-10-31): SAT-based solver for static and dynamic argumentation frameworks. Third ICCMA edition (2019). `http://argumentationcompetition.org/2019/papers/ICCMA19_paper_11.pdf`.

[60] Fuan Pu, Hang Ya, and Guiming Luo. argmat-dvisat: A division-based algorithm framework for solving argumentation problems using SAT. *The Second International Competition on Computational Models of Argumentation (ICCMA'17)*, 2017.

[61] Fuan Pu, Hang Ya, and Guiming Luo. argmat-sat: Applying SAT solvers for argumentation problems based on boolean matrix algebra. *The Second International Competition on Computational Models of Argumentation (ICCMA'17)*, 2017.

[62] Patrick Saint-Dizier. Challenges of argument mining: generating an argument synthesis based on the qualia structure. In *9th International Conference on Natural Language Generation (INLG 2016)*, pages pp–79, 2016.

[63] Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.

[64] Edward Tsang. *Foundations of constraint satisfaction: the classic text*. BoD–Books on Demand, 2014.

[65] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[66] Jan Van Leeuwen. *Handbook of theoretical computer science (vol. A) algorithms and complexity*. Mit Press, 1991.

[67] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[68] Johannes Peter Wallner, Georg Weissenbacher, and Stefan Woltran. Advanced SAT techniques for abstract argumentation. In *International Workshop on Computational Logic in Multi-Agent Systems*, pages 138–154. Springer, 2013.