Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# Advancements in Equivalence Checking for Abstract Argumentation Frameworks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Logic and Computation**

eingereicht von

**Patrick Bellositz, BSc**
Matrikelnummer 01027108

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Ass. Dipl.-Ing. Dipl.-Ing. Dr.techn. Wolfgang Dvořák
Mitwirkung: Univ.Prof. Dipl.-Ing. Dr.techn. Stefan Woltran

Wien, 24. August 2018

Patrick Bellositz | Wolfgang Dvořák

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# Advancements in Equivalence Checking for Abstract Argumentation Frameworks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Logic and Computation**

by

**Patrick Bellositz, BSc**
Registration Number 01027108

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Ass. Dipl.-Ing. Dipl.-Ing. Dr.techn. Wolfgang Dvořák
Assistance: Univ.Prof. Dipl.-Ing. Dr.techn. Stefan Woltran

Vienna, 24th August, 2018

Patrick Bellositz

Wolfgang Dvořák

# Erklärung zur Verfassung der Arbeit

Patrick Bellositz, BSc
Eichkogelstraße 10/10/9
2353 Guntramsdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 24. August 2018

Patrick Bellositz

# Danksagung

Ich danke meinen geduldigen Betreuern Wolfgang Dvořák und Stefan Woltran.

Dank spreche ich auch meiner Familie: Denise, Martha und Harald sowie meinen Großeltern Johann und Johanna.

Desweiteren danke ich Christoph, Martina und Konstantin für ihre Meinungen zu dieser Arbeit.

# Acknowledgements

I thank my patient advisors Wolfgang Dvořák and Stefan Woltran.

I also extend thanks to my family Denise, Martha and Harald as well as to my grandparents Johann and Johanna.

Furthermore, I thank Christoph, Martina and Konstantin for their input on this thesis.

# Kurzfassung

Formale Argumentationstheorie ist ein aktives Forschungsfeld innerhalb der Künstlichen Intelligenz. *Argumentation Frameworks* (AFs) wurden erstmals 1995 von Dung beschrieben. Sie stellen einen Ansatz zur Betrachtung inkonsistenten Wissens dar, welches in Form von Argumenten und Angriffen zwischen ihnen beschrieben wird. Argumente können anhand von *Semantiken* zu Mengen, sogenannten *Extensionen*, zusammengefasst werden, solange diese nicht widersprüchlich sind. Seit deren Erstbeschreibung wurden AFs um zahlreiche weitere Semantiken erweitert.

Beim Einsatz von AFs kann es zu Änderungen an Argumenten und Angriffen, und daher Änderungen an deren Informationsgehalt, kommen. Abstrakte Argumentation kann daher als dynamischer Vorgang betrachtet werden. Solche Veränderungen können mithilfe von Äquivalenzrelationen untersucht werden. Es kann somit festgestellt werden, ob eine Änderung Extensionen beeinflusst. Unter den bisher beschriebenen Äquivalenzrelationen finden sich zum Beispiel *standard*, *expansion*, *strong* und *C-relativised equivalence*.

Das Konzept der C-relativised equivalence ist für die ursprünglich von Dung vorgestellten Semantiken *stable*, *admissible*, *preferred*, *complete* und *grounded* definiert. Diese sind hierbei verändert – *C-restricted* – sodass nur eine beschränkte Menge an Argumenten untersucht wird, die sogenannten *core* Argumente. Da nicht alle Argumente betrachtet werden, ist ein lokal beschränktes Ersetzen von Teil-AFs möglich.

In dieser Arbeit wird C-relativised equivalence zur automatisierten Ersetzung einer großen Anzahl an lokal äquivalenten AFs durch vereinfachte Alternativen verwendet. Diese Ersetzung soll zur Reduktion von Rechenaufwand beim Berechnen von Extensionen führen. Dieses Ziel wird erreicht, indem *C-relativised equivalency patterns* eingeführt werden, welche die Äquivalenz eines AFs vor und nach einer Teilersetzung – bezüglich einer gegebenen Semantik – garantieren.

Zur Findung solcher patterns, wird in dieser Arbeit ein System zur Beschreibung von C-restricted semantics, sowie von Charakterisierungen von C-relativised equivalence als Programmcode präsentiert. Dieses System stellt für eine beliebige Anzahl von AFs paarweise fest, ob diese äquivalent sind. Weiters wird eine große Anzahl solcher Äquivalenzberechnungen analysiert, um C-relativised equivalency patterns zu finden und derartig gefundene patterns werden formal bewiesen. Darauf folgt eine Diskussion der Auswirkungen einer Anwendung solcher patterns auf generierte AFs, welche in Experimenten untersucht werden.

# Abstract

Formal Argumentation is an actively studied field within Artificial Intelligence. *Abstract Argumentation Frameworks* (AFs), introduced by Dung in 1995, are an approach to reason over inconsistent knowledge. This knowledge is represented by arguments which may attack one another. Sets of non-conflicting arguments may then be grouped into *extensions* according to some *argumentation semantics* of which many different ones have been introduced over the years.

When working with AFs, changes in arguments and attacks may occur and therefore change the information contained within the framework. Hence abstract argumentation can be described as dynamic reasoning. To study the effects of such changes to AFs and whether they affect changes in extensions, different notions of equivalency have been proposed. These include *standard*, *expansion*, *strong* and *C-relativised equivalence*.

C-relativised equivalence is defined for the *stable*, *admissible*, *preferred*, *complete* and *grounded* semantics originally introduced by Dung. These semantics are modified (i.e. *C-restricted*) to only consider the relations of a set of *core* arguments. By not reasoning over all arguments, this concept enables local replacements of sub-AFs.

This thesis examines how to automate C-relativised equivalence checking to replace large numbers of locally equivalent frameworks with simpler alternatives and therefore reduce computational strain on AF solver programs. This goal is achieved by formally defining *C-relativised equivalency patterns* that guarantee equivalence of an AF before and after replacement of a sub-AF therein with respect to some semantics and under some constraints for attack relations.

To find C-relativised equivalence patterns, this work presents a system of encodings of C-restricted semantics as well as C-relativised equivalence characterisations. This system accepts arbitrary numbers of AFs and determines whether they are pairwise equivalent. For this work, the results of large numbers of equivalency computations are systematically analysed to find C-relativised equivalency patterns. Found patterns are then formally proven and the effects of applying them to generated frameworks are discussed, based on experiments that are conducted.

# Contents

**Kurzfassung** **xi**

**Abstract** **xiii**

**1 Introduction** **1**
1.1 Argumentation Theory . . . . . 1
1.2 Abstract Argumentation Frameworks . . . . . 2
1.3 The Problem in Question . . . . . 2
1.4 Methodology . . . . . 3
1.5 Contributions . . . . . 5
1.6 Organisation . . . . . 6

**2 Preliminaries** **7**
2.1 Abstract Argumentation Frameworks . . . . . 7
2.2 Semantics . . . . . 8
2.3 Notions of Equivalence for Argumentation Frameworks . . . . . 11

**3 Implementing C-relativised Equivalence** **15**
3.1 Encoding C-restricted Semantics . . . . . 15
3.2 Encoding C-relativised Equivalency . . . . . 29
3.3 Summary . . . . . 43

**4 C-relativised Equivalency Patterns** **45**
4.1 Pattern Definition . . . . . 45
4.2 Patterns for Stable Semantics . . . . . 46
4.3 Patterns for Admissible Semantics . . . . . 55
4.4 Patterns for Preferred Semantics . . . . . 61
4.5 Patterns for Complete Semantics . . . . . 62
4.6 Summary . . . . . 66

**5 C-relativised Equivalency Patterns for Pre-Processing** **67**
5.1 Problem Statement . . . . . 67
5.2 Replacing Pattern Occurrences . . . . . 68
5.3 Experimental Evaluation . . . . . 72

**6 Conclusion** **79**
6.1 Summary . . . . . 79
6.2 Future Work . . . . . 80

**Bibliography** **81**

CHAPTER 1

# Introduction

Humans argue in many facets of life. Such arguments may concern political beliefs, planning a trip, business decision making, and countless other situations. Often, the goal of people participating in arguments is making the best points to sway another person's opinion or to win the argument.

## 1.1 Argumentation Theory

The higher the number of different arguments in a conversation, the more likely it is that people might contradict each other or even themselves, since with greater amounts of information, the difficulty of maintaining coherent sets of arguments rises. Thus it is beneficial to be able to formalise or even automate the concept of argumentation and its processes. The field of Argumentation Theory is concerned with such formalisations as well as the examination of how one may arrive at the best decisions or the most accurate conclusions.

There are different perspectives to Argumentation Theory that include such fields as sociology, philosophy, mathematics and computer science [SR09]. In this thesis we are concerned with formal systems that may be used to gain knowledge via an automated or even autonomous system.

Arguments can be formalised in a number of different ways. They may be expressed by Abstract Argumentation Frameworks that look at the relations between sets of arguments [Dun95], as dialogue games with the goal of persuading an opponent [MHP07] or be expressed as first order logic formulas using trees to give argument relations structure [BH05]. In [GS04] argumentation is formalised as logic programs. Further fomalisations include assumption-based argumentation [DKT09] and coherence frameworks for finding consensus [JP09]. For more information about formal argumentation one may consult

the *Handbook of Formal Argumentation* [BGGvdT18].

The approach considered in this thesis, is the notion of Abstract Argumentation Frameworks which we describe in more detail in the next section.

## 1.2 Abstract Argumentation Frameworks

Abstract Argumentation Frameworks, introduced in 1995 by Dung in [Dun95], are a way of non-monotonic reasoning. Different variations of these frameworks can be used for tasks like AI learning [BD07], preparing for legal arguments [BM09] or to analyse political opinions [CA08].

Abstract Argumentation Frameworks (also Argumentation Frameworks, abbreviated AFs) are a way to consider arguments and how they may conflict with each other. Here, arguments are thought of exclusively in the abstract, disregarding their content and instead focussing on the relationships between them.

AFs consist of arguments and attacks, where each attack is a relationship between two arguments. To derive meaning from these frameworks, semantics are applied to the arguments to be able to find self-consistent and defensible sets of arguments with relation to the whole AF. There exist different semantics for different purposes, sceptical or credulous, that approach defensibility of arguments from different angles. In his seminal work, Dung originally introduces *admissible, preferred, complete, grounded and stable* semantics, which are the semantics discussed herein. Later literature expands on these in different ways, for instance by introducing *cf2* [BGG05], *prudent* [CDM05], *ideal* [DMT07] and *semi-stable* [Cam06] semantics. A survey of these and additional semantics can be found in [BGGvdT18].

Other approaches for AFs introduce argument preferences [Ben03] and probabilities [LON11], fuzzy AFs [TC14], modelling them as constraint satisfaction [AD13] or satisfiability problems [CDGV13].

## 1.3 The Problem in Question

In addition to developing new semantics, there has been research on the question of whether two (or more) AFs are equivalent. This helps to understand how they may behave differently when presented with new information. Understanding AF-equivalence may enable the efficient and dynamic removal or addition of specific arguments and attacks from an AF without losing any crucial properties of the framework.

Kernel-based expansion equivalence [Bau12] examines how a given framework behaves, when additional arguments or attacks are added to it. A similar notion is explored, when

examining whether two AFs are equivalent. There are different concepts of argumentation framework equivalency, such as standard or strong equivalency [OW11]. Furthermore, the question of how equivalency is affected by local replacements. This notion is further explored in [BDLW17], is called *C-relativised equivalence* and makes use of *C-restricted semantics* for local extensions.

This thesis takes the concept of *C-relativised equivalence* for AFs and seeks to give a formal definition of *C-relativised equivalency patterns*. With these patterns, consisting of two AFs, we want to be able to replace one of the AFs with the other in any larger AF, as long as one of the frameworks provided by the pattern is contained within the larger graph, without losing the larger AF's properties.

This means, that given two AFs $F'$ and $G'$ and a semantics $\sigma$, a pattern has to ensure that in any AF $F$, where $F' \subseteq F$ one can replace $F'$ with $G'$ (or vice versa) without changing the overall behaviour of $F$ with regard to a given semantics, by making sure no arguments violate certain attack restrictions.

This thesis defines such patterns formally and seeks a way to find them. Furthermore, we determine with what frequency they appear in realistic instances of argumentation frameworks and what that might mean for computational effort in AF solving by using the patterns for pre-processing before employing any other methods or algorithms.

## 1.4 Methodology

This section presents the tools we use to solve the problem described in section 1.3. First, we discuss the basic concepts of Answer-set programming and how it can be used to describe AFs. Subsequently, we describe the subgraph isomorphism problem and how it is related to finding C-relativised equvalency patterns within larger AFs.

### 1.4.1 Answer-set programming

Answer-set programming (ASP), a form of declarative programming [Lif08], is a programming paradigm that is closely related to the logic programming paradigm, but can be distinguished from approaches like PROLOG in that they always terminate, are less general purpose and give models as solutions instead of deriving solutions by looking for proofs [BET11]. ASP is used to solve complex search problems [Lif08].

ASP also deviates from other programming paradigms in that it does not resolve top to bottom [BET11], but instead declaratively searches the rules of a program from which knowledge is gained, i.e. instead of starting program execution with the first command and working downwards in the program document, the order of rules in any specific problem definition is not relevant to the solution model.

Like with other programming paradigms for which there are multiple languages, there exist multiple tools with different dialects of ASP like clasp [GKNS07], DLV [LPF+06] or Smodels [NS97]. The basic structure of rules of any such programming language is as follows:

$$a \leftarrow b_1, b_2, ..., b_k.$$

In general this means, that if all $b_i$ are evaluated as being true, we derive that $a$ is true. Otherwise we can derive *not* $a$, which means $a$ could not be proven and thus is assumed false. This concept is called *negation-as-failure* [Lif08].

To derive any knowledge from rules, there needs to be a set of facts (a knowledge base). For example if we wanted the program to assume $b_1$ as true, we would include the statement

$$b_1.$$

within our program code.

Apart from these basic concepts, ASP languages may include different forms of disjunction to get different options for solutions or constraints to filter out certain models. More expansive enhancements of ASP include aggregate functions to enable concepts like summation and counting [DFI+03], or the inclusion of sets and lists [CCIL08].

ASP is connected with AFs as it is used as a means of encoding frameworks in [EGW10], which enables the automated evaluation of AFs, including the deriving of extensions to gain knowledge about optimal subsets of arguments. This approach is based on the ASPARTIX [EGW08] tool, while further advancements for some semantics have been achieved in [GMR+15].

This thesis introduces ASP encodings of C-restricted semantics, that can be used to compare multiple AFs. These encodings produce solutions that contain all pairs of C-relativised equivalent AFs from a given set of input frameworks with regards to a chosen semantics.

### 1.4.2 Subgraph isomorphism

Finding patterns in AFs is closely related to the subgraph isomorphism problem because AFs can be expressed as graphs and therefore sub-frameworks can be mapped onto subgraphs. The subgraph isomorphism problem asks the question of whether a graph (the *query graph*) is isomorphic to some subgraph within a larger *database* graph (i.e.

it is subgraph-isomorphic) [AY17]. This problem can be extended into the subgraph matching problem, where instead of only detecting whether the given graph is included in a lager one, the goal is to find all occurrences of the query graph and match them to specific parts of the database graph [AY17].

More specialised variants of these problems are concerned with labelled graphs (e.g. [ZYY07]), weighted graphs (e.g. [BFGV03]) or directed graphs [Ull76]. In the case of AFs, attacks are modelled as directed edges without weights or special labels and therefore we are concerned with subgraph matching of directed graphs.

There are many proposed solutions to efficiently find out whether a query graph can in fact be found in database graph. Generally, there are two approaches to this problem ([AY17]): (1) the *filtering-and-verification* approach (employed in [MNG$^+$10] and [ZHY07]) uses an index of graph features (represented by small graphs) to check whether the database graph includes the features of the query graph and then filters feature occurrences for exact matches, whereas (2) *branch-and-bound* algorithms (used in [SZLY08] and [Ull10]) construct matches vertex by vertex for as long as a local occurrence can still become an exact match to find all matches.

Herein we modify the *BB-Graph* algorithm (based on the branch-and-bound approach) presented in [AY17] to aide in enabling the replacement of pattern instances.

## 1.5 Contributions

The contributions of this thesis are twofold. Firstly, theoretical contributions include a formalisation of the concept of *C-relativised equivalency patterns* such that for any pair of AFs, we can define patterns where a sub-framework can be replaced by a simpler version without changing the extensions of a given semantics. Additionally, we use this formalisation to give formal proofs of concrete patterns and properties useful for local replacement during pre-processing. The patterns as well as the properties are found through systematic investigation of equivalent argumentation frameworks.

Secondly, there are two kinds of practical contributions. First, we implement a tool ([Bel18a]) for C-relativised equivalencies of arbitrary numbers of argumentation frameworks using answer-set programming to obtain data for such systematic investigation. This program takes user-defined or automatically generated AFs and outputs pairs of C-relativised equivalent AFs for any given semantics in the scope of the thesis, enabling the checking of large numbers of AFs needed for such investigation and is used to find all patterns and properties proven herein. Subsequently, we implement an algorithm ([Bel18b]) based on the BB-Graph algorithm in [AY17] and expanded so that any proper occurrence of a pattern AF in a given larger AF is replaced by its alternate AF. Equipped

with this tool, we then study empirical results obtained by evaluation of the frequency of patterns within greater amounts of large, generated AFs as well as an examination of possible effects of local replacements on the execution time of AF solvers.

## 1.6 Organisation

The thesis is structured as follows:

- chapter 2 formally introduces AFs, their extensions and AF equivalence,
- chapter 3 introduces *C-restricted* extensions and the notion of *C-relativised equivalency* and presents the implementation of an ASP-encoding thereof,
- chapter 4 describes *C-relativised equivalency* patterns found with the help of encoding from chapter 3 and formally proves their correctness. More general pattern properties are also introduced,
- chapter 5 evaluates the practical effect of using patterns to locally replace parts of real AF instances by analysing the results of experiments,
- finally, chapter 6 concludes the thesis and discusses further possible research connected to this thesis.

CHAPTER 2

# Preliminaries

In this section we define abstract Argumentation Frameworks (AFs) as originally introduced by Dung in [Dun95]. Furthermore the relationships between arguments (i.e. attacks and defences) are defined and, based on these relationships, the semantics for selecting coherent sets (i.e. admissible, stable, preferred, complete and grounded) of arguments are presented and elaborated upon.
After defining these essential notions regarding Argumentation Frameworks, it is discussed which conditions must be met by AFs to be seen as equivalent. Here, three different notions of equivalence (standard, strong and C-relativised) are defined and explained.

## 2.1 Abstract Argumentation Frameworks

Argumentation Frameworks consist of a set of arguments which may be related to each other via attacks. All definitions in this section were originally introduced in [Dun95] or [BCG11], unless otherwise stated.

**Definition 1.** *An* (abstract) Argumentation Framework *is a pair* $F = (A, R)$.

- $A \subseteq D$ *is a finite set of arguments, whereby* $D$ *is fixed as a countable infinite set of arguments (the domain).*
- $R \subseteq A \times A$ *is a set of attacks, where for arguments* $a, b \in A$*,* $a$ *attacks* $b$ *when there exists an attack* $(a, b) \in R$.

*Where needed* $A$ *is denoted as* $A(F)$*, and* $R$ *is denoted as* $R(F)$.

Each AF $F = (A, R)$ can be represented as a directed graph $G_F = (V, E)$, where $V = A$ and $E = R$. This is shown in Figure 2.1.

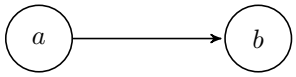



Figure 2.1: Graphical representation of an AF with $A = \{a, b\}$ and $R = \{(a, b)\}$.

To relate different sets of arguments we introduce $S^-, S^+$ and $S^\oplus$, that express the sets of arguments they attack or are attacked by.

**Definition 2.** *For a set $S \subseteq A$:*

- $S^- = \{a \in A | \exists b \in S : (a, b) \in R\}$ *is the set of arguments attacking $S$.*
- $S^+ = \{a \in A | \exists b \in S : (b, a) \in R\}$ *is the set of arguments attacked by $S$.*
- $S^\oplus = S^+ \cup (S \cap A)$ *is the* range *of $S$.*

Knowledge about which sets attack each other is used to define the notion of *defence*.

**Definition 3.** *A set $S \subseteq A$* defends *an argument $a \in A$ if $\{a\}^- \subseteq S^+$. Furthermore, the* characteristic function *$\mathcal{F}_F(S) = \{a \in A | a$ is defended by $S$ in $F\}$ computes the set of all arguments defended by $S$.*

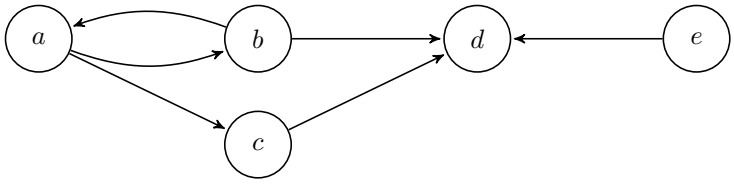



Figure 2.2: A more expansive argumentation framework.

**Example 1.** *Figure 2.2 shows the graph representation of $F$, where $A(F) = \{a, b, c, d, e\}$, and $A(R) = \{(a, b), (a, c), (b, a), (b, d), (c, d), (e, d)\}$. It can be seen that for $S = \{a, d\}$, $S^+ = \{b, c\}$, since $a$ attacks $b$ and $c$, but $d$ does not attack any argument. It follows that $S^\oplus = \{a, b, c, d\}$. On the other hand $S^- = \{b, c, e\}$ because $b, c$ and $e$ each attack $d$. Furthermore it can be seen that $\{a, d\}$ defends $a$, since $\{a\}^- = \{b\}$ and $b \in \{a, d\}^+$, but not $d$ because $e \notin \{a, d\}^+$. $\{a, d\}$ also defends $e$, since it is not attacked, but does neither defend $b$ nor $c$ because $\{a\}^+ = \{b, c\}$.*

## 2.2 Semantics

To reason with AFs, certain semantics are used that describe different properties of sets of arguments. In this section we first define the basic notion of *conflict-free* sets, then introduce the most fundamental semantics and finally look at how the different semantics relate to each other.

*Conflict-free* sets are used as a basis for all such semantics. If a set is *conflict-free* it is certain that it does not contradict itself.

**Definition 4.** *A set $S \subseteq A$ is called* conflict-free *if there is no attack $(a, b) \in R$ where $a, b \in S$, i.e. $S \cap S^+ = \emptyset$. The set of all conflict-free sets in $F$ is denoted by $cf(F)$.*

### 2.2.1 Semantics definitions

The following semantics are the most fundamental ones as introduced in [Dun95]. Note that since then additional semantics have been defined. Some important ones can be found in [BCG11].

A further step from conflict-free sets towards rational reasoning is reached if a set of arguments can defend itself from outside attacks. Such a set is called an *admissible set.*

**Definition 5.** *A set $S \in cf(F)$ is called an* admissible extension *if it defends all arguments $a \in S$, i.e. $S^- \subseteq S^+$. The set of all admissible extensions in $F$ is denoted by $adm(F)$.*

*Preferred semantics* are credulous, not only accepting arguments that provide defences, but also accepting other arguments as long as they are not contradictory. Preferred extensions are maximal admissible extensions.

**Definition 6.** *A set $S \in adm(F)$ is called a* preferred extension *if for all $T \in adm(F)$, $S \not\subset T$. The set of all preferred extensions in $F$ is denoted by $prf(F)$.*

**Example 2.** *Recalling $F$ from Example 1, the set of conflict-free sets $cf(F) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{a, d\}, \{a, e\}, \{b, c\}, \{b, e\}, \{c, e\}, \{b, c, e\}\}$. This set includes every combination of arguments in $A(F)$ that do not attack each other.*
*Since only sets within $cf(F)$ can be extensions, only conflict-free sets are checked for inclusion in these sets.*
*The set of admissible extensions $adm(F) = cf(F) \backslash \{\{c\}, \{d\}, \{a, d\}, \{c, e\}\}$ does not retain sets that do not defend themselves from outside attacks. For instance $c$ is only defended against $a$ by $b$ and neither $\{c\}$ nor $\{c, e\}$ include $b$, which makes them not admissible. Additionally $d$ is not defended from $e$ by any argument and therefore $\{d\}$ and $\{a, d\}$ are also not admissible extensions.*
*Preferred extensions are subset maximal admissible extensions. Neither $\{a\}, \{b\}, \{c\}, \{d\}$ nor $\{e\}$ are maximal, because they are either subsets of $\{a, e\}, \{b, c\}, \{b, e\}$ or of $\{b, c, e\}$. The sets $\{b, c\}$ and $\{b, e\}$ are themselves subsets of $\{b, c, e\}$, resulting in $prf(F) = \{\{a, e\}, \{b, c, e\}\}$.*

A stricter semantics than admissible and preferred extensions is the *stable extension.* Instead of merely defending itself, a set that is said to be a stable extension attacks all other arguments, even if they do not attack it.

**Definition 7.** *A set $S \in cf(F)$ is called a* stable extension *if for every argument $a \notin S$ there is an attack $(b, a) \in R$, where $b \in S$, i.e. $S^{\oplus} = A(F)$. The set of all stable extensions in $F$ is denoted by $stb(F)$.*

**Example 3.** *Continuing Example 2, conflict-free sets that attack all arguments not included within themselves constitute $stb(F) = \{\{a, e\}, \{b, c, e\}\}$. Because there exists no attack on $e$, it must be included in any stable extension; it also provides an attack on $d$. Adding $a$ to $e$ provides attacks on $b$ and $c$, and therefore $\{a, e\} \in stb(F)$. On the other hand $\{b, e\}$ and $\{c, e\}$ both do not attack all arguments they do not include, leaving only $\{b, c, e\}$ as a stable extension.*

*Complete extensions* include all the arguments that they defend.

**Definition 8.** *A set $S \in cf(F)$ is called a* complete extension *if $S$ defends all arguments $a \in S$ and there exists no argument $b \notin S$ that is defended by $S$, i.e. $S = \mathcal{F}_F(S)$. The set of all complete extensions in $F$ is denoted by $com(F)$.*

Opposed to the credulous semantics of the preferred extensions, the *grounded extension* represents skeptical reasoning by including only those arguments that are included in each complete extension.

**Definition 9.** *The set $grd(F) = \bigcap_{T \in com(F)} T$ is called the (unique)* grounded extension *of $F$.*

**Example 4.** *Complete extensions for $F$ from Example 2 include all arguments that they defend. This means that $e$ is part of every complete extension for $F$, since it is not attacked and it only attack $d$ which does not attack another argument. The argument $a$ defends itself from $b$ and defends no other argument from all its attackers. On the other hand $b$ defends itself and $c$ from $a$, while also attacking $d$. Therefore $com(F) = \{\{e\}, \{a, e\}, \{b, c, e\}\}$. The unique grounded extension is the intersection $grd(F) = \{e\}$ of all complete extensions. Since only $e$ is included in every complete extension, it is the only argument included in $grd(F)$.*

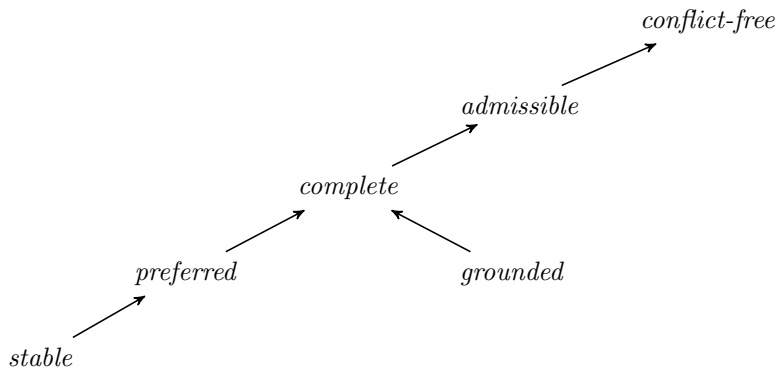



Figure 2.3: Relations between the extensions of an argumentation framework. An arrow from X to Y means that, if a set is an extension w.r.t. semantics X, then it is also an extension w.r.t semantics Y.

As we have seen a set often satisfies multiple semantics. In [BCG11] it is shown that the different semantics are not isolated from each other, but are related. Figure 2.3 shows exactly these relations for the semantics examined in this thesis.

## 2.3 Notions of Equivalence for Argumentation Frameworks

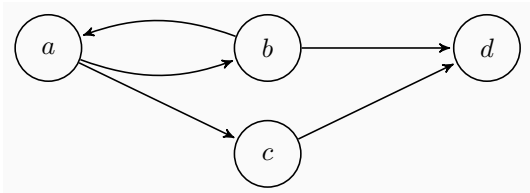
To compare AFs it is useful to know whether they are equivalent. In this section three ways of determining which AFs are equivalent are presented. Firstly, standard equivalence is defined, secondly strong equivalence is discussed. Afterwards we look at C-relativised equivalency, which is the main focus of this thesis.

### 2.3.1 Standard equivalence

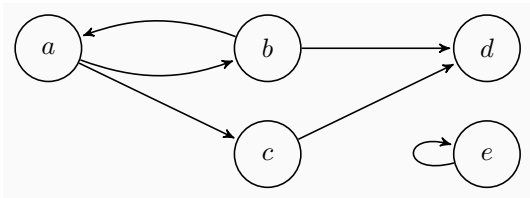
*Standard equivalence* is the naive approach to equivalence, wherein two AFs are said to be equivalent w.r.t. some semantics if they possess the exact same extensions.

**Definition 10** ([OW11])**.** *AFs $F$ and $G$ are* standard equivalent*, i.e. $F \equiv^{\sigma} G$, iff $\sigma(F) = \sigma(G)$, where $\sigma \in \{stb, adm, prf, com, grd\}$.*

**Example 5.** *Figure 2.4 shows two AFs $F_1$ and $F_2$. Two AFs are standard equivalent w.r.t. a given semantic if they have the exact same extensions for that semantic.*
*The only difference between the AFs is the argument $e$. It follows that $cf(F_1) = cf(F_2)$, since $e$ can not be part of any conflict-free set because it attacks itself.*
*$e$ does not attack other arguments, hence $adm(F_1) = adm(F_2)$ and therefore $F_1 \equiv^{adm} F_2$. It also follows that $F_1 \equiv^{prf} F_2$.*
*Additionally, since $e$ can not be defended by and be included in the same conflict-free set we know $F_1 \equiv^{com} F_2$ and therefore also $F_1 \equiv^{grd} F_2$.*




(a)




(b)

Figure 2.4: Argumentation frameworks $F_1$ (a) and $F_2$ (b) are standard equivalent w.r.t. some semantics.

*Contrary to the other semantics* $F_1 \not\equiv^{stb} F_2$. *While* $stb(F_1) = \{\{a, d\}, \{b, c\}\}$, *e is not attacked by any other argument resulting in* $stb(F_2) = \emptyset$.

### 2.3.2 Strong equivalence

*Strong equivalence* is far more restrictive than standard equivalence. Two AFs are only strongly equivalent w.r.t. a given semantics if they are standard equivalent and any addition of arguments or attacks, that is applied to both AFs results in the exact same changes to their extensions.

**Definition 11** ([BDLW17])**.** *The* union *of AFs* $F, G$ *is defined as* $F \cup G = (A(F) \cup A(G), R(F) \cup R(G))$.

**Definition 12** ([OW11, BDLW17])**.** *AFs* $F$ *and* $G$ *are* strongly equivalent *w.r.t.* $\sigma \in \{stb, adm, prf, com, grd\}$, *formally* $F \equiv^{\sigma}_{S} G$, *iff* $F \cup H \equiv^{\sigma} G \cup H$ *for all AFs* $H$.

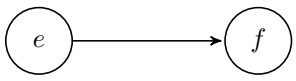



Figure 2.5: An expansion to the AFs in Figure 2.4

**Example 6.** *Provided the AFs* $F_1$ *and* $F_2$ *from Example 5 and the expansion* $H$ *given in Figure 2.5, it can be shown whether* $F_1 \cup H \equiv^{\sigma} F_2 \cup H$ *for* $\sigma \in \{adm, stb, prf, com, grd\}$.

- $F_1 \not\equiv^{stb}_{S} F_2$ *is already shown in Example 3, since* $F_1 \cup \emptyset \not\equiv^{stb} F_2 \cup \emptyset$.

- $F_1 \not\equiv_S^{adm} F_2$, *since* $\{e\} \in adm(F_1 \cup H)$, *but* $\{e\} \notin adm(F_2 \cup H)$, *since* $e$ *only attacks itself within* $F_2 \cup H$.
- $F_1 \not\equiv_S^{prf} F_2$ *follows from* adm, *because* $e \in E, E \in prf\,(F_1 \cup H)$, *but* $E \notin F_2 \cup H$.
- $F_1 \not\equiv_S^{com} F_2$, *since, similar to* prf, $\{\{a, d, e\}\} = com(F_1 \cup H)$ *and* $\{\{a, d\}\} = com(F_2 \cup H)$, *since* $e \notin F_1$.
- $F_1 \not\equiv_S^{grd} F_2$ *directly follows from* com.

### 2.3.3 C-relativised equivalence

Between standard and strong equivalence there exists the notion of *C-relativised equivalency*. Here, equivalence is parameterized by a set of arguments $C \subseteq D$, called *core arguments*.

**Definition 13** ([BDLW17])**.** *Two AFs* $F, G$ *are* C-relativised equivalent *w.r.t. a semantics* $\sigma$, *formally* $F \equiv_C^\sigma G$, *iff for each AF* $H \subseteq D \backslash C, F \cup H \equiv^\sigma G \cup H$.

Note that if $C = D$, C-relativised equivalency functions as standard equivalency, since for each $H \subseteq D \backslash C = \emptyset, F \cup H = F$ and $G \cup H = G$ and therefore $F \equiv_C^\sigma H$ iff $\sigma(F) = \sigma(G)$, which is the definition of standard equivalency. On the other hand, with $C = \emptyset$, C-relativised equivalency behaves like strong equivalency, since in that case each $H \subseteq D \backslash C = H$ and therefore $F \equiv_C^\sigma H$ iff $F \equiv_S^\sigma G$.

CHAPTER 3

# Implementing C-relativised Equivalence

In this chapter ASP-encodings for testing several AFs for *C-relativised equivalence* are given. Before encoding the rules and conditions for equivalency itself, encodings of argumentation frameworks and locally resticted versions of the considered semantics (called *C-restricted semantics*) are also defined, encoded and explained.

## 3.1 Encoding C-restricted Semantics

*C-restricted semantics* are needed to evaluate whether some AFs are *C-relativised equivalent*. In this section C-restricted versions for stable, admissible, preferred, complete and grounded semantics are examined and it is shown how to encode them using answer-set programming.

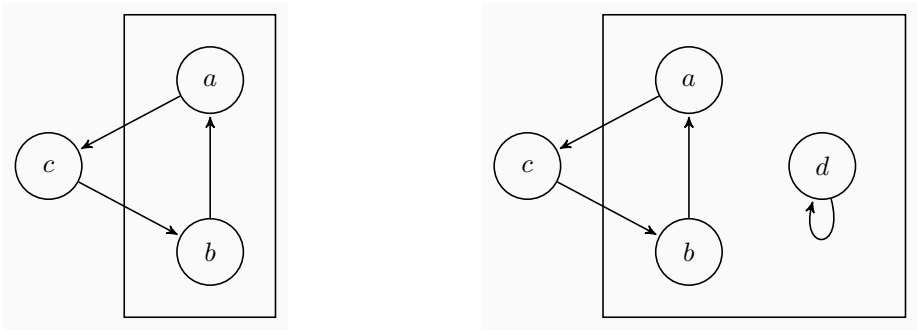
### 3.1.1 Encoding Argumentation Frameworks

To be able to computationally determine any kind of equivalence relation between AFs it is neccessary to determine a format for defining them. This includes fixing the domain of arguments that can be used, identifiers for all AFs, their arguments and attacks as well as a set of core arguments. In answer-set programming this is done by defining facts.

Firstly, we write for each argument $a \in D$, where $D$ is the domain:

$$\text{domain}(a).$$

Then it is possible to define the core set $C \subseteq D$ by fixing a set of core arguments $\{x_1, x_2, ...\}$ where sets of elements are denoted by using square brackets:




(a) (b)

Figure 3.1: The AFs $f_1$ (a) and $f_2$ (b) are represented as graphs; the set of core arguments inside a rectangle.

$$\text{core}([x1, x2, ...]).$$

Lastly we must set for each AF $f_i$ the set of arguments $A(f_i) = \{x_1, x_2, ...\}$ and the set of attacks $R(f_i) \subseteq A(f_i) \times A(f_i)$ as follows:

$$\text{arguments}(fi, [x1, x2, ...]).$$

$$\text{attacks}(fi, [[x1, x2], [x2, x3], ...]).$$

**Example 7.** *For the AFs in Figure 3.1, the following encoding $\pi_{facts}$ is defined:*

$$\begin{aligned}\pi_{facts} = & \mathit{domain}(a).\\ & \mathit{domain}(b).\\ & \mathit{domain}(c).\\ & \mathit{domain}(d).\\ & \mathit{core}([a, b, d]).\\ & \mathit{arguments}(f1, [a, b, c]).\\ & \mathit{attacks}(f1, [[b, a], [c, b], [a, c]]).\\ & \mathit{arguments}(f2, [a, b, c, d]).\\ & \mathit{attacks}(f2, [[b, a], [c, b], [a, c], [d, d]]).\end{aligned}$$

### 3.1.2 Basic Encodings

Given the above facts, we can use rules to deduct more information. Some of the basic rules that are needed for computing multiple semantics are given here.

Before any actual computation starts we introduce some rules to better represent the raw information given by the facts.

In 3.1, three rules are introduced: (1) $af(F)$ holds for any $F$ that is given as an AF in an *arguments* fact; (2) $argument(F, A)$ is derived for each argument $A$ in the domain, that is part of AF $F$; and (3) the $attack(F, A, B)$ predicate is true for arguments $A, B$, where $A$ attacks $B$ in $F$. These rules are together denoted by $\pi_{\text{args}}$.

$$
\begin{aligned}
\pi_{\text{args}} = \quad & \text{af}(F) \leftarrow \text{arguments}(F, X). \\
& \text{argument}(F, A) \leftarrow \text{domain}(A), \text{af}(F), \text{arguments}(F, X), \#\text{member}(A, X). \\
& \text{attack}(F, A, B) \leftarrow \text{argument}(F, A), \text{argument}(F, B), \#\text{member}([A, B], X), \\
& \qquad \text{attacks}(F, X).
\end{aligned} \tag{3.1}
$$

Here, the rule *af* only holds, if each AF is correctly defined by use of an *arguments* fact, even if it does not contain any arguments (i.e. by giving it the empty list of arguments []). Also note that *argument* and *attack* use the aggregate function *#member*, which determines inclusion in a list, in this case inclusion within a set of arguments and attacks respectively.

The program described here approaches solving argumentation frameworks by first computing all $S \subseteq A$, and subsequently filtering out contradictory or otherwise unqualified sets. The computation of powersets is achieved by using the rules described in 3.2.

$argSet(F, X)$ holds iff $X$ is a subset of the arguments of the AF $F$. Notice that all sets of arguments encoded are represented by lists, where elements are ordered increasingly, i.e. each set is represented by exactly one list.

$$
\begin{aligned}
\pi_{\text{argSet}} = \quad & \text{argSet}((F, []) \leftarrow \text{arguments}(F, X). \\
& \text{argSet}((F, X) \leftarrow \text{argument}(F, A), \text{argSet}((F, Y), \#\text{append}(Y, [A], X), \\
& \qquad \#\text{length}(Y, 0). \\
& \text{argSet}((F, X) \leftarrow \text{argument}(F, A), \text{argSet}((F, Y), \#\text{append}(Y, [A], X), \\
& \qquad \text{not } \#\text{member}(A, Y), \#\text{last}(Y, B), B < A.
\end{aligned} \tag{3.2}
$$

To understand how the rules described in $\pi_{\text{argSet}}$ work, it is also important to know that $\#last(Y, B)$ holds if $B$ is the rightmost element of list $Y$ and $\#length(Y, A)$ is true only if the list $Y$ has $A$ elements, i.e. $\#length(Y, 0)$ only holds if list $Y$ is empty. Additional, $\#append(Y, A, X)$ describes three lists $A, X, Y$, where $Y$ concatenated with $A$ equals $X$.

Knowing that, we can see that using $argSet(F, X)$, all sets of arguments are built by starting with the empty set. From this point we iteratively build all sets by appending arguments of $F$ that are greater than the set's current greatest element to the current $X$, creating new sets for each such argument and repeat the process with each of them until there are no greater arguments left to add.

Once, all possible sets of arguments are computed, it is possible to start filtering out unwanted options, i.e. only conflict-free sets are used for further computation of extensions. Equation 3.3 describes a set of rules to obtain these.

Conflict-free sets are obtained by use of the rule $cf(F, E)$, which holds for all sets $E$, which are *argSets*, that do not have attacks between their own arguments.

$$\begin{aligned} \pi_{\text{cf}} = \; & \text{notCf}(F, E) \leftarrow \text{argSet}(F, E), \text{argument}(F, A), \text{argument}(F, B), \\ & \qquad \text{attack}(F, A, B), \#\text{member}(A, E), \#\text{member}(B, E). \\ & \text{cf}(F, E) \leftarrow \text{argSet}(F, E), \text{ not notCf}(F, E). \end{aligned} \tag{3.3}$$

More detailed, for each *argSet* $E$ it is first evaluated whether it is not conflict-free within $F$ and, subsequently all $E$ which are not not conflict-free (i.e. not *notCf*) are then derived as being conflict-free. Hence we could also write $\text{cf}(F, E) \Leftrightarrow \text{argSet}(F, E) \wedge \neg\text{notCf}(F, E)$.

Such uses of double-negation are a common occurence in the encodings presented in this thesis.

For ease of use, from now on we summarise these rules for further reference by saying $\pi_{\text{basic}} = \pi_{\text{args}} \cup \pi_{\text{argSet}} \cup \pi_{\text{cf}}$.

Since conflict-free sets are encoded, we are able to formulate encodings of *C-restricted semantics*, which are based on the semantics introduced in chapter 2 and are needed to compute *C-relativised equivalency*. Further notice that the set of all C-restricted extensions of some semantics $\sigma$ is denoted by $\sigma_C(F)$, whereby $\sigma \in \{\text{adm,prf,stb,com,grd}\}$. All definitions for C-restricted semantics were originally introduced in [BDLW17].

### 3.1.3 C-restricted Stable Semantics

*C-restricted stable semantics* function like *stable semantics*, with the difference that the requirement to attack all arguments is restricted to core arguments.

**Definition 14.** *[BDLW17] A set $S \in cf(F)$ is called a* C-restricted stable extension *if for every argument $a \notin S, a \in C \cap A(F)$ there is an attack $(b, a) \in R$, where $b \in S$, formally $S \in stb_C(F)$ if $A(F) \cap C \subseteq S_F^{\oplus}$.*

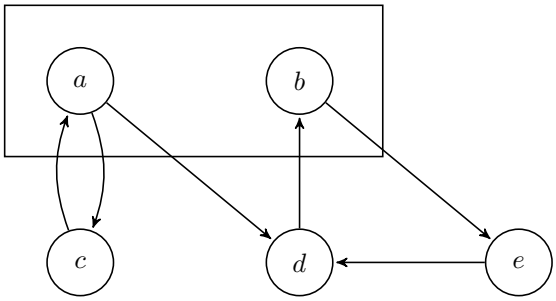



Figure 3.2: This argumentation framework has a difference between stable and C-restricted stable extensions (see Example 8).

Since C-restricted semantics are less strict than the standard ones, there may be more C-restricted sets that qualify as extensions within the same AF than for non-restricted semantics.

**Example 8.** *Figure 3.2 shows the argumentation framework $F$ with $C = \{a, b\}$. The set of stable extensions $stb(F) = \{\{a, b\}\}$ contains the only conflict-free set that attacks all arguments it does not contain. When this restriction applies only to core arguments we see that $\{b, c\}$ attacks $a$ and contains $b$, while $\{c, d\}$ attacks both core arguments and $\{a, b\} = C$. This leads to a larger set than that of the stable extensions: $stb_C(F) = \{\{a, b\}, \{b, c\}, \{c, d\}\}$.*

To be able to compute answer-sets (e.g. as in Example 8) we encode *C-restricted stable semantics.*

First, we introduce in $\pi_{\text{stbc}}$ (3.4) the shorthand rule *isAttacked*, which checks whether an argument $A$ is attacked by a set of arguments. That means $isAttacked(F, A, Y)$ holds iff there exists an argument $B \in Y$, which attacks the argument $A \in A(F)$.

Further, to compute C-restricted stable extensions, the predicate $notAncSub(F, E)$ is introduced, which is true only for sets $E \in cf(F)$ for which there is an argument $A \in C, A \notin E$, where $isAttacked(F, E, Y)$ does not hold.

Finally, the rule $stbc(F, E)$ returns all conflict-free sets $E$ in $F$, for which $notAncSub(F, E)$ does not hold.

$$
\begin{aligned}
\pi_{\text{stbc}} = \text{isAttacked}(F, A, Y) \leftarrow & \text{argument}(F, A), \text{argument}(F, B), \text{cf}(F, Y), \\
& \#\text{member}(B, Y), \text{attack}(F, B, A). \\
\text{notAncSub}(F, E) \leftarrow & \text{argument}(F, A), \text{cf}(F, E), \text{core}(C), \#\text{member}(A, C), \\
& \text{not } \#\text{member}(A, E), \text{not isAttacked}(F, A, E). \\
\text{stbc}(F, E) \leftarrow & \text{cf}(F, E), \text{not notAncSub}(F, E).
\end{aligned} \tag{3.4}
$$

As can be seen above $\pi_{\text{stbc}}$ again makes use of double negation by computing all sets $E$ for which there can be no argument found in the core, which $E$ does not attack.

The computation of C-restricted stable extensions of an AF $F_i$ therefore requires all the rules and facts included in $\pi_{\text{basic}} \cup \pi_{\text{stbc}} \cup \pi_{\text{facts}_i}$, where $\pi_{\text{facts}_i}$ describes $F_i$ and the set of core arguments $C$.

**Example 9.** *The encoding given above therefore returns the following answer-set for the AF from Example 8:*

$$\begin{aligned}\pi_{stbc_{AS}} = & stbc(f, [a, b]).\\ & stbc(f, [b, c]).\\ & stbc(f, [c, d]).\end{aligned}$$

### 3.1.4 C-restricted Admissible Semantics

Like C-restricted stable extensions, *C-restricted admissible extensions* are similar to their non-restricted counterpart. The difference is that they only need to defend themselves from core arguments, but not from arguments, that are not in $C$.

**Definition 15.** *[BDLW17] A set $S \in cf(F)$ is called a* C-restricted admissible extension *if $S$ attacks every argument $a \in C$ with $\{a\}^+ \subseteq S$, i.e. $S \in adm_C(F)$ if $S^- \cap C \subseteq S^+$.*

**Example 10.** *Recalling $F$ from Example 8 it can be seen that $adm(F) = \{\emptyset, \{a\}, \{c\}, \{a, b\}\}$ since $a$ is only attacked by $c$, against which it defends itself, $c$ has no other interactions, the empty set is always admissible and $b$ is defended against its only attacker $d$ by $a$.*
*If one now disregards undefended attacks from non-core arguments the following* C-restricted admissible extensions *follow: $adm_C(F) = adm(F) \cup \{\{b\}, \{b, c\}, \{c, d\}\}$. This is because $b$ is only attacked by $d \notin C$ and the attack $(e, d)$ is disregarded since $e \notin C$.*

Analoguous to the approach for C-restricted stable semantics, a rule for sets attacking single arguments is introduced for *C-restricted admissible semantics* in $\pi_{\text{admc}}$ (see Equation 3.5). This *isAttacking* rule determines whether some set $E$ is attacked by an argument $A$. This means that $isAttacking(F, A, E)$ holds for every set $E \in cf(F)$, where there exists an argument $B \in E$ that is attacked by $A \in A(F)$.

This rule can be used in conjunction with the *isAttacked* rule to evaluate for a set, if any of its arguments is not defended against an attacker and as a consequence whether it can be deduced to be *C-restricted admissible.* For this purpose the predicate $undef(F, E)$ exists, which is true for each set $E \in cf(F)$ iff there is an argument $A \in E$, such that $A$ attacks $E$, but $A$ is not attacked by $E$.

Finally, we derive $admc(F, E)$ for all $E \in cf(F)$ if they are not undefended against any attacking arguments.

$$
\begin{aligned}
\pi_{\text{admc}} = \text{isAttacking}(F, A, E) \leftarrow & \text{argument}(F, A), \text{argument}(F, B), \text{cf}(F, E), \\
& \#\text{member}(B, E), \text{attack}(F, A, B). \\
\text{undef}(F, E) \leftarrow & \text{argument}(F, A), \text{cf}(F, E), \text{core}(C), \#\text{member}(A, C), \\
& \text{isAttacking}(F, A, E), \text{not isAttacked}(F, A, E). \\
\text{admc}(F, E) \leftarrow & \text{cf}(F, E), \text{not undef}(F, E).
\end{aligned} \tag{3.5}
$$

To be able to compute the *C-restricted admissible extensions* of an AF $F_i$, we need $\pi_{\text{basic}} \cup \pi_{\text{admc}} \cup \pi_{\text{facts}_i}$ as well as the *isAttacked* predicate from $\pi_{\text{stbc}}$.

### 3.1.5 C-restricted Preferred Semantics

*C-restricted preferred semantics* concern only C-restricted admissible extensions. To compute *C-restricted preferred extensions*, each C-restricted admissible extension is compared to all others to determine whether it is maximal w.r.t. the core argument set $C$.

**Definition 16.** *[BDLW17] A set $S \in adm_C(F)$ is called a* C-restricted preferred extension *if for all $E \in adm_C(F)$ the following holds: if (1) $E \backslash C = S \backslash C$, (2) $S^+ \backslash C \subseteq E^+ \backslash C$ and (3) $E^- \backslash E^+ \subseteq S^- \backslash S^+$, then (4) $S \cap C \not\subset E \cap C$.*

**Example 11.** *Continuing from Example 10, all $S \in adm_C(F)$ are evaluated regarding* C-restricted preferred semantics*. This means each $S$ is compared to each $E \in adm_C(F)$:*

> *$\emptyset \notin prf_C(F)$, since $\emptyset \backslash C = \{a\} \backslash C$, $\emptyset^+ \backslash C \subseteq \{a\}^+ \backslash C$ and $\{a\}^- \backslash \{a\}^+ \subseteq \emptyset^- \backslash \emptyset^+$, but $\emptyset \cap C \subset \{a\} \cap C$, i.e. (1)-(3) hold for $\emptyset$ regarding $\{a\}$, but (4) does not.*
>
> *$\{a\} \notin prf_C(F)$, since while (1)-(3) hold regarding $\{a, b\}$, (4) does not.*
>
> *$\{b\} \notin prf_C(F)$, since while (1)-(3) hold regarding $\{a, b\}$, (4) does not.*

*Notice that $\{c\}$ does only fulfill (1)-(3) regarding itself, but not regarding $\{b, c\}$. Since $c \notin C$ and $\{c\}$ is not a proper subset of itself, $\{c\} \in prf_C(F)$. There are no other C-restricted admissible violating (4), but satisfying (1)-(3) regarding any $E$, and therefore $prf_C(F) = \{\{c\}, \{a, b\}, \{b, c\}, \{c, d\}\}$.*

Preferred semantics require the encoding of multiple distinct properties. Firstly, we examine whether an admissible extension $E$ contains exactly the same arguments as another set $D$ outside the set of core arguments. To be able to compute this, the rule *diffArgsOut*$(F, E, D)$ is introduced in $\pi_{\text{prf}_A}$ (3.6) and holds for two sets $E, D \in adm_C(F)$ iff there is an argument $A \notin C$, which either occurs in $E$ or in $D$ but not in both, where the auxiliary rule *outside*$(F, A)$ is true only for arguments $A \notin C$.

$$
\begin{aligned}
\pi_{\mathrm{prf}_A} = \quad & \mathrm{outside}(F,A) \leftarrow \mathrm{argument}(F,A), \mathrm{core}(C), \mathrm{not}\ \#\mathrm{member}(A,C). \\
& \mathrm{diffArgsOut}(F,E,D) \leftarrow \mathrm{argument}(F,A), \mathrm{admc}(F,E), \mathrm{admc}(F,D), \\
& \qquad \#\mathrm{member}(A,E), \mathrm{not}\ \#\mathrm{member}(A,D), \mathrm{outside}(F,A). \\
& \mathrm{diffArgsOut}(F,E,D) \leftarrow \mathrm{argument}(F,A), \mathrm{admc}(F,E), \mathrm{admc}(F,D), \\
& \qquad \mathrm{not}\ \#\mathrm{member}(A,E), \#\mathrm{member}(A,D), \mathrm{outside}(F,A).
\end{aligned} \tag{3.6}
$$

Notice that $\pi_{\mathrm{prf}_A}$ contains two *diffArgsOut* rules. From now on if there are multiple occurrences of a rule with only minor differences (henceforth called a *double rule*), only one of them will be shown. In each of these cases a short descripiton of the differences is given.

Next, we check $E_F^+ \backslash C \subseteq D_F^+ \backslash C$. For this purpose the rule *plusOutNotCommon*(F,E,D) is defined in Equation 3.7. It holds for extensions $E, D \in adm_C(F)$ iff there is an argument $A \notin C$ that $D$ attacks, but $E$ does not.

$$
\begin{aligned}
\pi_{\mathrm{prf}_B} = \mathrm{plusOutNotCommon}(F,E,D) \leftarrow & \mathrm{argument}(F,A), \mathrm{core}(C), \mathrm{admc}(F,E), \\
& \mathrm{admc}(F,D), \mathrm{not}\ \#\mathrm{member}(A,C), \\
& \mathrm{isAttacked}(F,A,E), \mathrm{not}\ \mathrm{isAttacked}(F,A,D).
\end{aligned} \tag{3.7}
$$

Furthermore, we evaluate $D_F^- \backslash D_F^+ \subseteq E_F^- \backslash E_F^+$ in Equation 3.8. This property can be checked with the predicate *PMNotInc*$(F,E,D)$ which holds for $E, D \in adm_C(F)$ if there is an argument $A$, that (1) attacks $D$ and (2) is not attacked by $D$ and (3) is either (a) not attacking $E$ or (b) attacked by $E$.

For (3) we have another double rule. This means the rule in $\pi_{\mathrm{prf}_C}$ can be changed from *not isAttacking*$(F,A,E)$ to *isAttacked*$(F,A,E)$.

$$
\begin{aligned}
\pi_{\mathrm{prf}_C} = \mathrm{PMNotInc}(F,E,D) \leftarrow & \mathrm{argument}(F,A), \mathrm{admc}(F,E), \mathrm{admc}(F,D), \\
& \mathrm{not}\ \mathrm{isAttacking}(F,A,E), \mathrm{isAttacking}(F,A,D), \\
& \mathrm{not}\ \mathrm{isAttacked}(F,A,D).
\end{aligned} \tag{3.8}
$$

In 3.9 we combinedly use $\pi_{\mathrm{prf}_A}$, $\pi_{\mathrm{prf}_B}$ and $\pi_{\mathrm{prf}_C}$. Here, the rule *prfPremis* determines whether $E, D$ need to be checked for subset inclusion.

*prfPremis*$(F,E,D)$ holds if $E, D \in adm_C(F)$ and neither *diffArgsOut*, *plusOutNotCommon* nor *PMNotInc* hold.

$$\pi_{\text{prf}_D} = \begin{aligned} \text{prfPremis}(F,E,D) \leftarrow & \text{admc}(F,E), \text{admc}(F,D), \text{not diffArgsOut}(F,E,D), \\ & \text{not plusOutNotCommon}(F,E,D), \\ & \text{not PMNotInc}(F,E,D). \end{aligned} \tag{3.9}$$

If *prfPremis* holds for two C-restricted admissible extensions $S, D$, $S \cap C \not\subset D \cap C$ is checked. To check this, first the sets $S \cap C$ and $D \cap C$ are computed.

For that purpose the predicate $enc(F, E, X)$ (see encoding $\pi_{\text{enc}}$ in 3.10) is computed. To be able to do this, we need three auxiliary rules: *notCommon*, *common* and *notMaxCommon*. For all three rules $X \in cf(F), E \in adm_C(F)$ must be the case:

> $notCommon(F, E, X)$ can be derived if there is an argument $A$ that is either (1) in both $X$ and $E$, but not in $C$, (2) is in $X$ and $C$, but not in $E$ or (3) is in $X$ but neither in $E$ nor $C$.
>
> $common(F, E, X)$ holds in case *notCommon* does not hold.
>
> $notMaxCommon(F, E, X)$ can be derived if there is a set $Y$ for which $common(F, E, Y)$ holds and $|Y| > |X|$.

$$\pi_{\text{enc}} = \begin{aligned} \text{notCommon}(F,E,X) \leftarrow & \text{argument}(F,A), \text{cf}(F,X), \text{admc}(F,E), \text{core}(C), \\ & \#\text{member}(A,X), \#\text{member}(A,E), \\ & \text{not } \#\text{member}(A,C). \\ \text{notCommon}(F,E,X) \leftarrow & \text{argument}(F,A), \text{cf}(F,X), \text{admc}(F,E), \text{core}(C), \\ & \#\text{member}(A,X), \text{not } \#\text{member}(A,E), \\ & \#\text{member}(A,C). \\ \text{notCommon}(F,E,X) \leftarrow & \text{argument}(F,A), \text{cf}(F,X), \text{admc}(F,E), \text{core}(C), \\ & \#\text{member}(A,X), \text{not } \#\text{member}(A,E), \\ & \text{not } \#\text{member}(A,C). \\ \text{common}(F,E,X) \leftarrow & \text{cf}(F,X), \text{admc}(F,E), \text{not notCommon}(F,E,X). \\ \text{notMaxCommon}(F,E,X) \leftarrow & \text{common}(F,E,X), \text{common}(F,E,Y), \\ & \#\text{length}(X,A), \#\text{length}(Y,B), B > A, X \neq Y. \\ \text{enc}(F,E,X) \leftarrow & \text{common}(F,E,X), \text{not notMaxCommon}(F,E,X). \end{aligned} \tag{3.10}$$

As we can see, $enc(F, E, X)$ holds for maximal $X$ (w.r.t. set inclusion) that only contain core arguments that are in $E$, i.e. $X = E \cap C$.

Once we know $S \cap C$ and $D \cap C$, we can check whether the former is included in the latter. This is done using the rules that are defined within $\pi_{\text{prf}_E}$ (see 3.11).

Firstly, *encNotSubset* holds for $S, D \in adm_C(F)$ if either (1) there is an argument $A$ such that $A \in S \cap C$ and $A \notin D \cap C$ or (2) $S \cap C = D \cap C$.

Secondly, *encSubset* is only derivable for $S, D \in adm_C(F)$ for which *encNotSubset* does not hold, i.e. if $S \cap C \subset D \cap C$.

$$\begin{aligned}
\pi_{\text{prf}_E} = \text{encNotSubset}(F, S, D) \leftarrow & \text{argument}(F, A), \text{admc}(F, S), \text{admc}(F, D), \\
& \#\text{member}(A, X), \text{not } \#\text{member}(A, Y), \text{enc}(F, S, X), \\
& \text{enc}(F, D, Y). \\
\text{encNotSubset}(F, S, D) \leftarrow & \text{admc}(F, S), \text{admc}(F, D), \text{enc}(F, S, X), \text{enc}(F, D, Y), \\
& X = Y. \\
\text{encSubset}(F, S, D) \leftarrow & \text{admc}(F, S), \text{admc}(F, D), \text{not encNotSubset}(F, S, D).
\end{aligned} \tag{3.11}$$

Using the rules expressed in $\bigcup_{j \in \{A,B,C,D,E\}} \pi_{\text{prf}_j} \cup \pi_{\text{basic}} \cup \pi_{\text{admc}} \cup \pi_{\text{facts}_i} \cup \pi_{\text{enc}}$ as well as the *isAttacked* rule, allows for computation of the *C-restricted preferred extensions* of an AF $F_i$.

These are represented by the rules in $\pi_{\text{prfc}}$ (3.12): *prfc* and its auxiliary rule *prfNotImplied*.

*prfNotImplied*$(F, E)$ holds for $E \in adm_C(F)$ iff there is a $D \in adm_C(F)$ for which all three premisses in the definition hold, but the conclusion does not. *prfc* conversely holds iff *prfNotImplied* does not.

$$\begin{aligned}
\pi_{\text{prfc}} = \text{prfNotImplied}(F, E) \leftarrow & \text{admc}(F, E), \text{admc}(F, D), \text{prfPremis}(F, E, D), \\
& \text{encSubset}(F, E, D), E \neq D. \\
\text{prfc}(F, E) \leftarrow & \text{admc}(F, E), \text{not prfNotImplied}(F, E).
\end{aligned} \tag{3.12}$$

### 3.1.6 C-restricted Complete Semantics

As with non-restricted complete semantics, *C-restricted complete semantics* are calculated using a characteristic function, in this case adapted for C-restricted semantics. The *C-restricted characterstic function* [BDLW17] is given by:

$$\mathcal{F}_{F,C,E}(S) = \left\{ a \in E \;\middle|\; \forall c \in C : (c,a) \in R(F) \rightarrow c \in S_F^+ \right\} \cup \\ \left\{ c \in C \cap A(F) \;\middle|\; \forall (b,c) \in R(F) : b \in S_F^+ \cup (S_F^- \backslash C) \right\}$$

This function collects for a set $S$ all arguments $a \in E$ that $S$ defends against core arguments as well as all arguments $c \in C \cap A(F)$ that are either defended by $S$ or that share attackers with $S$. This characteristic function is split in two for encoding purposes, the left (see 3.13) as well as the right (see 3.14) part of the union.

We define the left part of the union in $\pi_{\text{cleft}}$, containing the rule *notInLeft*$(F, E, S, A)$ which holds iff for sets $E, S \in cf(F)$ there are arguments $A \in E, B \in C$ and $B$ attacks $A$, but is not attacked by set $S$.

$$\begin{aligned} \pi_{\text{cleft}} = \text{notInLeft}(F,E,S,A) \leftarrow & \text{argument}(F,A), \text{argument}(F,B), \text{cf}(F,E), \text{cf}(F,S), \\ & \text{core}(C), \text{attack}(F,B,A), \text{not isAttacked}(F,B,S), \\ & \#\text{member}(A,E), \#\text{member}(B,C). \end{aligned} \tag{3.13}$$

The right part (defined in $\pi_{\text{cright}}$) contains the auxiliary rule *inSubUnion*$(F, E, B)$, which holds if an argument $B$ is either attacked by $E$ or is attacked by it (except if $B \in C$).

The main rule in $\pi_{\text{cright}}$ is *notInRight*$(F, S, A)$, that is true for some set of arguments $S$ iff (1) $A \in C$ is attacked by another argument $B$, which is neither attacked by $S$ nor is attacking non-core parts of $S$ or (2) $A \notin C, A \in S$.

$$\begin{aligned} \pi_{\text{cright}} = \quad \text{notInRight}(F,S,A) \leftarrow & \text{argument}(F,A), \text{argument}(F,B), \text{argSet}(F,S), \\ & \text{core}(C), \text{attack}(F,B,A), \#\text{member}(A,C), \\ & \text{not inSubUnion}(F,S,B). \\ \text{notInRight}(F,S,A) \leftarrow & \text{argument}(F,A), \text{argSet}(F,S), \text{core}(C), \\ & \#\text{member}(A,S), \text{not } \#\text{member}(A,C). \\ \text{inSubUnion}(F,E,B) \leftarrow & \text{argument}(F,B), \text{core}(C), \text{isAttacking}(F,B,E), \\ & \text{not } \#\text{member}(B,C). \\ \text{inSubUnion}(F,E,B) \leftarrow & \text{cf}(F,E), \text{argument}(F,B), \text{isAttacked}(F,B,E). \end{aligned} \tag{3.14}$$

The characteristic function is needed to define *C-restricted complete semantics* and its encoding is used to compute $com_C(F)$:

**Definition 17.** *[BDLW17] A set $E \in cf(F)$ is called a* C-restricted complete extension *if $E = \mathcal{F}_{F,C,E}(E)$.*

**Example 12.** *Expanding further upon Example 8 we compare regular complete extensions to* C-restricted complete extensions*.*

*When examining $F$, it can be seen that $com(F) = \{\emptyset, \{c\}, \{a, b\}\}$. On the other hand,* C-restricted complete semantics *result in more extensions. Firstly, the empty set is trivially C-restricted complete, since it does not defend anything. Other extensions include $\{c\}$, since it defends itself from a and does not relate to b as well as $\{b\}$ because it does not interact with a, which is the only other core argument. Furthermore, $\{a, b\}, \{b, c\}$ and $\{c, d\}$ are also C-restricted complete.*

The ASP program determines whether a set $E \in comc(F)$ by evaluating if $E$ equals the result of the characteristic function, using $E$ as parameter. As before, a double negation approach is used in Equation 3.15.

Here, the predicate $notCom(F, E)$ holds iff (1) there is an argument $A \in E$ that is not in the characteristic function or (2) an argument $A \notin E$ is part of the result of the characteristic function.

$$
\begin{aligned}
\pi_{\text{comc}} = \; & \text{notCom}(F, E) \leftarrow \text{argument}(F, A), \text{cf}(F, E), \#\text{member}(A, E), \\
& \qquad \text{notInLeft}(F, E, E, A), \text{notInRight}(F, E, A). \\
& \text{notCom}(F, E) \leftarrow \text{argument}(F, A), \text{cf}(F, E), \text{core}(C), \text{not } \#\text{member}(A, E), \\
& \qquad \#\text{member}(A, C), \text{not notInRight}(F, E, A). \\
& \text{comc}(F, E) \leftarrow \text{cf}(F, E), \text{not notCom}(F, E).
\end{aligned} \tag{3.15}
$$

Finally, the rule $comc(F, E)$ holds for every $E \in cf(F)$ for which computing the characteristic function again results in $E$. For this computation (regarding AF $F_i$) we use $\pi_{\text{comc}} \cup \pi_{\text{cright}} \cup \pi_{\text{cleft}} \cup \pi_{\text{basic}} \cup \pi_{\text{facts}_i}$ as well as the *isAttacked* and *isAttacking* predicates as defined in $\pi_{\text{stbc}}$ and $\pi_{\text{admc}}$.

### 3.1.7 C-restricted Grounded Semantics

We also define *C-restricted grounded semantics* using the C-restricted characteristic function:

**Definition 18.** *[BDLW17] A set $S \in cf(F)$ is called a* C-restricted grounded extension *if $S = \mathcal{F}^{\infty}_{F,C,S}(\emptyset)$.*

Notice, the fixed point of the characterstic function is computed (denoted by $^{\infty}$), using the empty set as a starting parameter. Also note that *C-restricted grounded extensions*

are not unique w.r.t. $F$, i.e. an AF can have more than one C-restricted grounded extension.

**Example 13.** *Continuing from Example 12, we know that there is only one unique grounded extension $\emptyset$. To the contrary, there are multiple* C-restricted grounded extensions. *It can easily be seen that the right part of the characteristic function always results in $\emptyset$ given $\emptyset$ as its parameter. Therefore, any conflict-free sets that also have an empty left side can not be C-restricted grounded (except for $\emptyset$, which is grounded). Sets without an empty left side are $\{a\}, \{b\}, \{a, b\}, \{a, e\}$ and $\{b, c\}$. $\{a\}$ and $\{a, e\}$ are not C-restricted complete since they also defend $b$ against $c$ and $\{b, c\}$ can not be C-restricted grounded either because $b$ does not defend $c$. $\{b\}$ and $\{a, b\}$ equal their fixpoint and hence are C-restricted grounded.*

To compute the fixpoint of the characteristic function, the set of rules $\pi_{\text{grd}_A}$ (in 3.16) is used to collect for each *C-restricted complete extension* $E$ all the arguments that are in $\mathcal{F}_{F,C,S}(E)$.

The *fixing*$(F, E, S, X)$ rules collect in set $X$ iteratively all arguments that are in the set produced by the characteristic function, given set $E$ as input. In the case of the C-restricted grounded extension we start by setting $E = \emptyset$. There are multiple rules to seperately consider whether a given argument $A$ is the first argument added or not and additionally whether it is either in the right or in the left part of the union computed by the characteristic function.

$$
\begin{aligned}
\pi_{\text{grd}_A} = \ & \text{fixing}(F, E, S, []) \leftarrow \text{comc}(F, E), \text{argSet}(F, S). \\
& \text{fixing}(F, E, S, X) \leftarrow \text{argument}(F, A), \#\text{member}(A, E), \\
& \qquad \#\text{append}(Pr, [A], X), \#\text{length}(Pr, 0), \\
& \qquad \text{fixing}(F, E, S, Pr), \text{not notInLeft}(F, E, S, A). \\
& \text{fixing}(F, E, S, X) \leftarrow \text{argument}(F, A), \text{core}(C), \#\text{member}(A, C), \\
& \qquad \#\text{append}(Pr, [A], X), \#\text{length}(Pr, 0), \\
& \qquad \text{fixing}(F, E, S, Pr), \text{not notInRight}(F, S, A). \\
& \text{fixing}(F, E, S, X) \leftarrow \text{argument}(F, A), \#\text{member}(A, E), \\
& \qquad \#\text{append}(Pr, [A], X), \#\text{last}(Pr, B), \text{fixing}(F, E, S, Pr), \\
& \qquad \text{not notInLeft}(F, E, S, A), B < A. \\
& \text{fixing}(F, E, S, X) \leftarrow \text{argument}(F, A), \text{core}(C), \#\text{member}(A, C), \\
& \qquad \#\text{append}(Pr, [A], X), \#\text{last}(Pr, B), \text{fixing}(F, E, S, Pr), \\
& \qquad \text{not notInRight}(F, S, A), B < A.
\end{aligned}
\tag{3.16}
$$

Since the ASP program retains each iteration of *fixing*, we need to find the maximum set produced by the rules in $\pi_{\mathrm{grd}_A}$. This maximum set is determined by the rules defined in 3.17. In addition to that, $\pi_{\mathrm{grd}_{B_2}}$ (see 3.18) contains further rules used to determine the fixed point.

The rules *notMaxFixing* and *maxFixing* are used to find the maximum set $X$ (regarding some $F$ with sets $E, S$) computed by the *fixing* rules.

*notMaxFixing*$(F, E, S, Pr)$ holds for set $Rs$ iff there is some set $As$ produced by the iterative process described above, in which some argument $A$ occurs, that is not in $Pr$. Conversely, *maxFixing* holds exactly for the sets that do contain all arguments collected by the *fixing* rules.

$$
\begin{aligned}
\pi_{\mathrm{grd}_{B_1}} = \mathrm{notMaxFixing}(F, E, S, Rs) \leftarrow & \mathrm{argument}(F, A), \mathrm{argSet}(F, S), \mathrm{comc}(F, E), \\
& \#\mathrm{member}(A, As), \mathrm{not}\ \#\mathrm{member}(A, Rs), \\
& \mathrm{fixing}(F, E, S, Rs), \mathrm{fixing}(F, E, S, As). \\
\mathrm{maxFixing}(F, E, S, As) \leftarrow & \mathrm{argSet}(F, S), \mathrm{comc}(F, E), \mathrm{fixing}(F, E, S, As), \\
& \mathrm{not}\ \mathrm{notMaxFixing}(F, E, S, As).
\end{aligned} \tag{3.17}
$$

Once we have computed the characteristic function using parameter $\emptyset$, the result of this computation is used as the new parameter. This process is repeated until the new parameter equals the previous one, i.e. $\mathcal{F}^{\infty}_{F,C,S}(\emptyset)$ is determined as the set which does not change anymore after some iteration.

Each computation step stores the number of iterations that were done to get to this iteration as the variable $N$, starting at 0.

The rule *grdIteration*$(F, E, Pr, Rs, 0)$ is used to repeat computation, each time with a new parameter. This parameter $Pr$ can also be seen as the result $Rs$ of the previous computation step. The predicate *eqIteration* checks whether the previous parameter equals the new one by expecting $Pr = Rs$. Finally, *notFirstEq* discards all iterations that are equal, but are not the first occurrence of equal parameters, whereas *firstEq*$(F, E, Rs, N)$ holds for the iteration with the lowest $N$ that has equal parameters.

$$
\begin{aligned}
\pi_{\mathrm{grd}_{B_2}} = \quad & \mathrm{grdIteration}(F, E, [], Rs, 0) \leftarrow \mathrm{comc}(F, E), \mathrm{maxFixing}(F, E, [], Rs). \\
& \mathrm{grdIteration}(F, E, Pr, Rs, N) \leftarrow \mathrm{comc}(F, E), \mathrm{maxFixing}(F, E, Pr, Rs), \\
& \qquad \mathrm{grdIteration}(F, E, S, Pr, M), \\
& \qquad \#\mathrm{count}\{As : \mathrm{argument}(F, As)\} = C, \#\mathrm{int}(N), \\
& \qquad \#\mathrm{int}(M), M = N - 1, N \leq C. \\
& \mathrm{eqIteration}(F, E, Rs, N) \leftarrow \mathrm{grdIteration}(F, E, Rs, Rs, N). \\
& \mathrm{notFirstEq}(F, E, Rs, N) \leftarrow \mathrm{eqIteration}(F, E, Rs, M), M < N, \#\mathrm{int}(N), \\
& \qquad \#\mathrm{int}(M). \\
& \mathrm{firstEq}(F, E, Rs, N) \leftarrow \mathrm{comc}(F, E), \mathrm{not\ notFirstEq}(F, E, Rs, N), \\
& \qquad \mathrm{eqIteration}(F, E, Rs, N), \#\mathrm{int}(N).
\end{aligned} \tag{3.18}
$$

To not compute an infinite number of iterations, the rule execution is bounded. We use the aggregate function $\#count\{As : \mathrm{argument}(F, As)\} = C$, which counts the number of arguments in the current set of parameters, with bound $N \leq C$ to at most iterate $|A(F)|$ times.

Finally, we evaluate whether $\mathcal{F}^{\infty}_{F,C,S}(\emptyset) = E$ (see 3.19), which would make $E$ a *C-restricted grounded extension.*

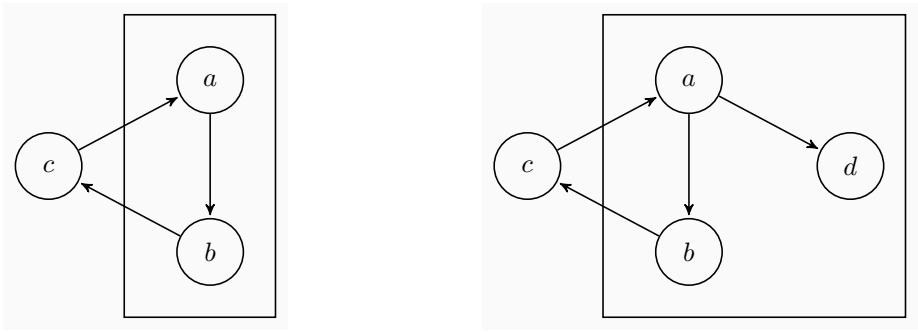
$grdc(F, E)$ holds for $E \in com_C(F)$ if we can find an iteration of the characteristic function, for which the previous parameter equals the current one.

$$
\pi_{\mathrm{grdc}} = \mathrm{grdc}(F, E) \leftarrow \mathrm{comc}(F, E), \mathrm{firstEq}(F, E, E, N), \#\mathrm{int}(N). \tag{3.19}
$$

To compute the C-restricted grounded extensions for an AF $F_i$, we therefore use $\pi_{\mathrm{grdc}} \cup \pi_{\mathrm{grd}_A} \cup \pi_{\mathrm{grd}_{B_1}} \cup \pi_{\mathrm{grd}_{B_2}} \cup \pi_{\mathrm{cright}} \cup \pi_{\mathrm{cleft}} \cup \pi_{\mathrm{comc}} \cup \pi_{\mathrm{basic}} \cup \pi_{\mathrm{facts}_i}$ as well as the *isAttacked* predicate.

## 3.2 Encoding C-relativised Equivalency

*C-relativised equivalency* is a notion of equivalence for argumentation frameworks that is parameterised by the set of core arguments $C$. This allows to check for local equivalency, where only a limited set of arguments are evaluated. This section shows how C-relativised equivalence between two AFs can be determined regarding stable, admissible, preferred, complete and grounded semantics. Figure 3.3 shows two AFs $F_1$ and $F_2$ which are not strongly equivalent, but are C-relativised equivalent for all of these semantics, given $C = \{a, b, c\}$. This section also includes encodings for automated computation of these equivalences.




(a) (b)

Figure 3.3: The AFs $F_1$ (a) and $F_2$ (b) are C-relativised equivalent for all semantics with $C = \{a, b, d\}$.

### 3.2.1 C-relativised Stable Equivalency

For two AFs $F, G$ to be *C-relativised stable equivalent*, it must hold that for every other AF $H \subseteq D \backslash C, F \cup H \equiv^{\sigma} G \cup H$. Since there are infinitely many AFs, it is impossible to check each $H$. We instead use the properties of *C-restricted stable semantics* to formulate conditions to be able find out whether $F$ and $G$ are indeed equivalent.

**Theorem 1.** *[BDLW17] Two AFs $F, G$ are* C-relativised stable equivalent *(i.e. $F \equiv_C^{stb} G$) iff the following conditions hold:*

*(1) If $stb_C(F) \neq \emptyset$, then $A(F) \backslash C = A(G) \backslash C$.*

*(2) $stb_C(F) = stb_C(G)$.*

*(3) $\forall E \in stb_C(F), E_F^+ \backslash C = E_G^+ \backslash C$.*

**Example 14.** *For the AFs $F_1 = (\{a, b, c\}, \{(a, b), (b, c), (c, a)\})$ and $F_2 = (\{a, b, c, d\}, \{(a, b), (b, c), (c, a), (d, a)\})$ with $C = \{a, b, d\}$ in Figure 3.3 we compute that $stb_C(F_1) = stb_C(F_2) = \{\{a\}\}$, directly satisfying condition (2). Since there is a C-restricted stable extension for $F_1$ and only $c \notin C$ for both AFs, condition (1) is satisfied and since a only attacks within C condition (3) also holds. Thus $F_1 \equiv_C^{stb} F_2$ holds.*

*Now assume instead $C = \{a, b\}$. While there is no change in C-restricted stable extensions, condition (1) does not hold anymore, since $d \in A(F_2) \backslash C$ and therefore $A(F_1) \backslash C \neq A(F_2) \backslash C$. Additionally (3) also does not hold anymore since a now attacks another argument (namely d) in $F_2 \backslash C$, but not in $F_1 \backslash C$. Hence $F_1 \not\equiv_C^{stb} F_2$.*

We state auxiliary rules for all three conditions for C-relativised stable equivalency below. The predicates for condition are stated in Equation 3.20.

To evaluate the first condition three rules are needed. While *nonempty*$(F)$ holds iff there is a C-restricted stable extension, *outside*(F,A) holds for each argument $A \notin C$.

Furthermore *outDiff*$(F, G, A)$ compares whether there are any arguments $A \notin C$ which are not both in $A(F)$ and $A(G)$.

$$\pi_{\text{stbdiff}_A} = \begin{aligned} \text{nonempty}(F) &\leftarrow \text{stbc}(F, E). \\ \text{outside}(F, A) &\leftarrow \text{argument}(F, A), \text{core}(C), \text{not } \#\text{member}(A, C). \\ \text{outDiff}(F, G, A) &\leftarrow \text{af}(F), \text{af}(G), \text{outside}(F, A), \text{not outside}(G, A). \end{aligned} \tag{3.20}$$

To evaluate property (2) we use the predicate *stbDifference*$(F, G, E)$ (3.21), which holds iff $E$ is not a C-restricted stable extension either in $F$ or in $G$.

Also in 3.21, two rules for ensuring condition (3) are encoded.

Firstly, *outsidePlus*$(F, E, A)$ holds iff for $E \in stb_C(F)$ there exists $A \in A(F), A \notin C$ such that $A$ is attacked by $E$. Secondly, *notBothOutPlus*$(F, G, E, A)$ is true if for some argument $A$ *outsidePlus* holds for $F$, but not for $G$.

$$\pi_{\text{stbdiff}_B} = \begin{aligned} \text{stbDifference}(F, G, E) &\leftarrow \text{arguments}(F, X), \text{arguments}(G, Y), \text{stbc}(F, E), \\ &\quad \text{not stbc}(G, E). \\ \text{outsidePlus}(F, E, A) &\leftarrow \text{af}(F), \text{argument}(F, A), \text{cf}(F, E), \text{core}(C), \\ &\quad \text{not } \#\text{member}(A, C), \text{isAttacked}(F, A, E). \\ \text{notBothOutPlus}(F, G, E, A) &\leftarrow \text{af}(F), \text{af}(G), \text{cf}(F, E), \text{cf}(G, E), \\ &\quad \text{outsidePlus}(F, E, A), \text{not outsidePlus}(G, E, A). \end{aligned} \tag{3.21}$$

To evaluate whether two AFs are equivalent we use these auxiliary rules to check for each condition whether it is violated. For that purpose new rules are introduced in Equation 3.22. Such rules which decide the inequivalency of AFs are called *inequivalency rules* from now on, since they exist for all semantics examined. Note that each of these rules is a double rule, with one rule determining $F > G$ and its counterpart checking $F < G$.

$$\begin{aligned}\pi_{\text{stbnot}} = \; & \text{notStEquiv}(F,G) \leftarrow \text{nonempty}(F), \text{outDiff}(F,G,A), F < G. \\ & \text{notStEquiv}(F,G) \leftarrow \text{nonempty}(F), \text{outDiff}(F,G,A), F > G. \\ & \text{notStEquiv}(F,G) \leftarrow \text{stbDifference}(F,G,E), F < G. \\ & \text{notStEquiv}(F,G) \leftarrow \text{stbDifference}(F,G,E), F > G. \\ & \text{notStEquiv}(F,G) \leftarrow \text{stbc}(F,E), \text{stbc}(G,E), \text{notBothOutPlus}(F,G,E,A), \\ & \qquad F < G. \\ & \text{notStEquiv}(F,G) \leftarrow \text{stbc}(F,E), \text{stbc}(G,E), \text{notBothOutPlus}(F,G,E,A), \\ & \qquad F > G.\end{aligned} \tag{3.22}$$

The final predicate, which holds only if no inequivalency rule for C-relativised stable semantics holds is called *stbeq* (3.23).

$$\begin{aligned}\pi_{\text{stbeq}} = & \\ & \text{stbEquiv}(F,G) \leftarrow \text{af}(F), \text{af}(G), \text{not notStEquiv}(F,G), F < G.\end{aligned} \tag{3.23}$$

If we want to compute the C-relativised stable equivalency for some set of AFs $\{F_i, F_j, ...\}$ the following set of rules needs to be used: $\pi_{\text{stbdiff}_A} \cup \pi_{\text{stbdiff}_B} \cup \pi_{\text{stbnot}} \cup \pi_{\text{stbeq}} \cup \pi_{\text{facts}_i} \cup \pi_{\text{facts}_j} \cup ...$ as well as all the rules needed for C-restricted stable semantics.

**Example 15.** *Continuing from Example 14 we show the answer-set produced by $\pi_{stbeq}$:*

$$\pi_{stbeq_{AS}} = stbEquiv(f1, f2).$$

### 3.2.2 C-relativised Admissible Equivalency

*C-relativised admissible equivalency* adds another condition compared to its stable counterpart.

**Theorem 2.** *[BDLW17] Two AFs $F, G$ are* C-relativised admissible equivalent *(i.e. $F \equiv_C^{adm} G$) iff the following conditions hold:*

*(1)* $A(F) \backslash C = A(G) \backslash C$.

*(2)* $adm_C(F) = adm_C(G)$.

*(3)* $\forall E \in adm_C(F)$ *the following conditions hold:*

- *(a)* $E_F^+ \backslash C = E_G^+ \backslash C$.
- *(b)* $E_F^- \backslash E_F^+ = E_G^- \backslash E_G^+$.

As with C-relativised stable equivalency, *C-relativised admissible equivalency* possesses a *difference* rule, which essentially works the same:

$$\pi_{\text{adm}_{\text{diff}}} = \text{adDifference}(F, G, E) \leftarrow \text{arguments}(F, X), \text{arguments}(G, Y), \text{admc}(F, E), \text{not admc}(G, E). \tag{3.24}$$

*C-relativised admissible equivalency* uses almost the same *inequivalency rules* as in $\pi_{\text{stbnot}}$, except here *nonempty* is not checked in the first rule and there is one additional inequivalency rule (again with $F > G$, $F < G$ as a double rule) using the auxiliary rules *eout* and *notBothEOut* (3.25).

*eout*$(F, E, A)$ holds for $E \in cf(F)$ iff there is some argument $A \in A(F)$ that attacks $E$, and $E$ does not defend itself against $A$.

*notBothEOut*$(F, G, E, A)$ can be derived for sets $E \in cf(F), E \in cf(G)$ for which *eout* holds for $F$, but not for $G$.

$$\begin{aligned}
\pi_{\text{admnot}} = \quad & \\
\text{eout}(F, E, A) &\leftarrow \text{argument}(F, A), \text{cf}(F, E), \\
&\quad \text{not isAttacked}(F, A, E), \text{isAttacking}(F, A, E). \\
\text{notBothEOut}(F, G, E, A) &\leftarrow \text{af}(F), \text{af}(G), \text{cf}(F, E), \text{cf}(G, E), \text{eout}(F, E, A), \\
&\quad \text{not eout}(G, E, A). \\
\text{notAdEquiv}(F, G) &\leftarrow \text{admc}(F, E), \text{admc}(G, E), \\
&\quad \text{notBothEOut}(F, G, E, A), F < G.
\end{aligned} \tag{3.25}$$

Like with 3.23, the final fact *admEquiv* is derived for equivalent AFs:

$$\pi_{\text{admeq}} = \text{admEquiv}(F, G) \leftarrow \text{af}(F), \text{af}(G), \text{not notAdEquiv}(F, G), F < G. \tag{3.26}$$

This means that computing C-relativised admissible equivalencies requires the rules in $\pi_{\text{adm}_{\text{diff}}} \cup \pi_{\text{admnot}} \cup \pi_{\text{admeq}}$, C-relativised admissible inequivalency rules, all rules needed to compute C-restricted admissible extensions as well as facts for the AFs to be compared.

### 3.2.3 C-relativised Preferred Equivalency

Since *C-restricted preferred extensions* are maximal *C-restricted admissible extensions* the notion of *C-relativised preferred equivalency* is defined very similarly to *C-relativised admissible equivalency.*

**Theorem 3.** *[BDLW17] Two AFs $F, G$ are* C-relativised preferred equivalent *(i.e. $F \equiv_C^{prf} G$) iff the following conditions hold:*

*(1)* $A(F)\backslash C = A(G)\backslash C$

*(2)* $prf_C(F) = prf_C(G)$.

*(3)* $\forall E \in prf_C(F)$ *the following conditions hold:*

*(a)* $E_F^+\backslash C = E_G^+\backslash C$.

*(b)* $E_F^-\backslash E_F^+ = E_G^-\backslash E_G^+$.

Encodings for *C-restricted preferred equivalency* are the same as for *C-restricted admissible equivalency*, except that all references to C-restricted admissible extensions are changed to references to their C-restricted preferred counterparts, e.g. all $admc(F, E)$ checks are replaced by $prfc(F, E)$ checks.

### 3.2.4 C-relativised Complete Equivalency

*C-relativised complete equivalency* needs further restrictions compared to *C-relativised admissible equivalency* and regarding the relationship between the results of the characteristic functions for both AFs $F$ and $G$ that we want to compare.

**Theorem 4.** *[BDLW17] Two AFs $F, G$ are* C-relativised complete equivalent *(i.e. $F \equiv_C^{com} G$) iff the following conditions hold:*

*(1)* $A(F)\backslash C = A(G)\backslash C$.

*(2)* $com_C(F) = com_C(G)$.

*(3)* $\forall E \in com_C(F)$ *the following conditions hold:*

*(a)* $E_F^+\backslash C = E_G^+\backslash C$.

*(b)* $E_F^-\backslash E_F^+ = E_G^-\backslash E_G^+$.

*(c) For all $S$ with $E_F^-\backslash E_F^+ \subseteq S \subseteq A(F)\backslash C \cup E$ the following holds: if $\mathcal{F}_{F\backslash S}(E) \cap C = E \cap C$ or $\mathcal{F}_{G\backslash S}(E) \cap C = E \cap C$ then $\mathcal{F}_{F\backslash S}(E) = \mathcal{F}_{G\backslash S}(E)$.*

*comDifference* is the exact analog of the previous rules introduced for other semantics. Beyond that, we introduce multiple new rules for *C-restricted complete equivalency.*

In Equation 3.27 it is ensured for a set $S$ that is to be used in further computations and an extension $E \in com_C(F)$, that $E_F^-\backslash E_F^+ \subseteq S \subseteq A(F)$ holds. This is achieved by formulating multiple auxiliary rules:

> $eoutNonS(F, E, S)$ holds iff there is an argument $A \notin S$ that is not attacked by $E$, but attacks $E$.

*eoutS*$(F, E, S)$ holds for all $E, S$ where *eoutNonS* does not hold.

*SinC*$(F, S)$ holds iff there is an argument $A \in S, A \in C$.

*SinE*$(F, E, S)$ holds iff there is an argument $A \in S, A \in E$.

*SinA*$(F, E, S)$ holds iff there is no argument $A \in S, A \in C$ and $A \in E$.

*legalS*$(F, E, S)$ holds iff $E_F^- \backslash E_F^+ \subseteq S \subseteq A(F) \backslash C \cup E$ holds.

*legalS* ensures the above-mentioned condition, since *eoutS* ensures that $E_F^- \backslash E_F^+ \subseteq S$ holds, while *SinA* ensures that $S \subseteq A(F) \backslash C \cup E$ holds.

$$
\begin{aligned}
\pi_{\mathrm{com}_A} = \mathrm{eoutNonS}(F, E, S) &\leftarrow \mathrm{argument}(F, A), \mathrm{argSet}(F, S), \mathrm{comc}(F, E), \\
&\quad \#\mathrm{not\ member}(A, S), \mathrm{eout}(F, E, A). \\
\mathrm{eoutS}(F, E, S) &\leftarrow \mathrm{argSet}(F, S), \mathrm{comc}(F, E), \mathrm{not\ eoutNonS}(F, E, S). \\
\mathrm{SinC}(F, S) &\leftarrow \mathrm{argument}(F, A), \mathrm{argSet}(F, S), \mathrm{core}(C), \\
&\quad \#\mathrm{member}(A, S), \#\mathrm{member}(A, C). \\
\mathrm{SinE}(F, E, S) &\leftarrow \mathrm{argument}(F, A), \mathrm{argSet}(F, S), \mathrm{comc}(F, E), \\
&\quad \#\mathrm{member}(A, S), \#\mathrm{member}(A, E). \\
\mathrm{SinA}(F, E, S) &\leftarrow \mathrm{argSet}(F, S), \mathrm{comc}(F, E), \mathrm{not\ SinC}(F, S), \\
&\quad \mathrm{not\ SinE}(F, E, S). \\
\mathrm{legalS}(F, E, S) &\leftarrow \mathrm{eoutS}(F, E, S), \mathrm{SinA}(F, E, S).
\end{aligned}
\tag{3.27}
$$

If the condition ensured by *legalS* holds, the original characteristic function about attacks is employed to check whether property (3c) holds. In this case though, we restrict its use to an AF without some specific set of arguments $S$. This means we also ignore attacks regarding arguments in $S$.

For this purpose, $\pi_{\mathrm{com}_{B_1}}$ (in Equation 3.28) introduces the rule *attackX*$(F, S, X, A)$ which is derived for all arguments $X \in A(F)$, which are attacked by $A \in A(F)$ (i.e. $A \in \{X\}^-$), where both $X$ and $A$ are not in the argument set $S$.

In the next step *notXinEplus* uses *attackX* to calculate the arguments not contained in the intersection of $C$ with the characteristic function for some $E \in com_C(F)$, while ignoring $S$. This means that *notXinEplus*$(F, S, X, E)$ holds for all $X \notin S$ that are attacked by some argument $A$ for which *attackX*$(F, S, X, A)$ holds and which is not attacked by $E$.

$$
\begin{aligned}
\pi_{\mathrm{com}_{B_1}} = \quad & \mathrm{attackX}(F,S,X,A) \leftarrow \mathrm{argument}(F,A), \mathrm{argument}(F,X), \mathrm{argSet}(F,S), \\
& \qquad \mathrm{attack}(F,A,X), \mathrm{not}\ \#\mathrm{member}(X,S), \\
& \qquad \mathrm{not}\ \#\mathrm{member}(A,S). \\
& \mathrm{notXinEplus}(F,S,X,E) \leftarrow \mathrm{argument}(F,A), \mathrm{argument}(F,X), \mathrm{argSet}(F,S), \\
& \qquad \mathrm{comc}(F,E), \mathrm{not}\ \#\mathrm{member}(X,S), \\
& \qquad \mathrm{not}\ \#\mathrm{member}(A,S), \mathrm{not}\ \mathrm{isAttacked}(F,A,E), \\
& \qquad \mathrm{attackX}(F,S,X,A).
\end{aligned} \tag{3.28}
$$

This again is a rule, where we want to compute the converse function to filter out unwanted results, which is shown in Equation 3.29 as $\pi_{\mathrm{com}_{B_2}}$.

In this case *xinEplus*$(F,S,X,E)$ holds iff *notXinEplus*$(F,S,X,E)$ does not, i.e. *xinEplus* holds for each argument $X \in \mathcal{F}_{F \setminus S}(E)$.

Lastly, we check whether the intersections $\mathcal{F}_{F \setminus S}(E) \cap C$ and $E \cap C$ are equal by using the predicate *unequalWithoutS*$(F,S,E)$. Notice, that *unequalWithoutS* is a double rule, the second version exchanging the two conditions *not #member*$(X,E)$ and *xinEplus*$(F,S,E,X)$ with *#member*$(X,E)$ and *not xinEplus*$(F,S,E,X)$.

$$
\begin{aligned}
\pi_{\mathrm{com}_{B_2}} = \quad & \mathrm{xinEplus}(F,S,X,E) \leftarrow \mathrm{argument}(F,X), \mathrm{argSet}(F,S), \mathrm{comc}(F,E), \\
& \qquad \mathrm{not}\ \#\mathrm{member}(X,S), \\
& \qquad \mathrm{not}\ \mathrm{notXinEplus}(F,S,X,E). \\
& \mathrm{unequalWithoutS}(F,S,E) \leftarrow \mathrm{argument}(F,X), \mathrm{argSet}(F,S), \mathrm{comc}(F,E), \\
& \qquad \mathrm{core}(C), \#\mathrm{member}(X,C), \mathrm{not}\ \#\mathrm{member}(X,E), \\
& \qquad \mathrm{xinEplus}(F,S,E,X).
\end{aligned} \tag{3.29}
$$

For two AFs $F, G$, if *unequalWithoutS* holds for at least one of them and *legalS* also holds, it is necessary to check whether the characteristic function produces the same result for both.

To satisfy this condition, the predicate *unequalFWWithoutS* (defined in 3.30) can not hold (meaning the predicate *equalFWWithoutS* does indeed hold).

*unequalFWWithoutS*$(F,G,S,E)$ does hold for AFs $F, G$, argument set $S$ and C-restricted complete extension $E$, iff there is an argument $X$ for which *xinEplus* holds for either $F$ or for $G$ and *unequalWithoutS* does not hold. *unequalFWWithoutS* is a double rule, where the conditions *xinEplus*$(F,S,X,E)$, *not xinEplus*$(G,S,X,E)$ can be replaced with *not xinEplus*$(F,S,X,E)$, *xinEplus*$(G,S,X,E)$.

$$\begin{aligned}\pi_{\mathrm{com}_C} = \mathrm{unequalFWWithoutS}(F,G,S,E) \leftarrow & \mathrm{argument}(F,X), \mathrm{argument}(G,X),\\ & \text{not } \mathrm{unequalWithoutS}(F,S,E),\\ & \mathrm{xinEplus}(F,S,X,E),\\ & \text{not } \mathrm{xinEplus}(G,S,X,E).\\ \mathrm{equalFWWithoutS}(F,G,S,E) \leftarrow & \mathrm{argSet}(F,S), \mathrm{argSet}(G,S), \mathrm{comc}(F,E),\\ & \mathrm{comc}(G,E), F \neq G,\\ & \text{not } \mathrm{unequalFWWithoutS}(F,G,S,E).\end{aligned} \tag{3.30}$$

In $\pi_{\mathrm{com}_A} \cup \pi_{\mathrm{com}_{B_1}} \cup \pi_{\mathrm{com}_{B_2}} \cup \pi_{\mathrm{com}_C}$ all rules needed to check condition (3c) are contained. In $\pi_{\mathrm{com}_{\mathrm{diff}}}$ (3.31) we combine all these rules, meaning the predicate *equalityNotImplied* holds iff (3c) does.

$$\begin{aligned}\pi_{\mathrm{com}_{\mathrm{diff}}} = \mathrm{equalityNotImplied}(F,G) \leftarrow & \mathrm{comc}(F,E), \mathrm{comc}(G,E), \mathrm{legalS}(F,E,S),\\ & \text{not } \mathrm{unequalWithoutS}(F,S,E),\\ & \text{not } \mathrm{equalFWWithoutS}(F,G,S,E), F \neq G.\end{aligned} \tag{3.31}$$

*C-restricted complete equivalency* uses the same inequality rules as does the admissible one, except for adding the following rule:

$$\begin{aligned}\pi_{\mathrm{com}_{\mathrm{not}}} = & \\ & \mathrm{notComEquiv}(F,G) \leftarrow \mathrm{equalityNotImplied}(F,G), F \neq G.\end{aligned} \tag{3.32}$$

The *comEquiv* rule, that holds for all C-relativised complete equivalent AFs, again does not differ from the rules for other semantics, except for referring to complete semantics instead.

In conclusion, C-relativised complete equivalencies can be calculated by use of all the rules required for calculation of C-restricted complete extensions as well as the rules in $\pi_{\mathrm{com}_A} \cup \pi_{\mathrm{com}_B} \cup \pi_{\mathrm{com}_C} \cup \pi_{\mathrm{com}_{\mathrm{diff}}}$, the inequivalency rules (including $\pi_{\mathrm{com}_{\mathrm{not}}}$), the above-mentioned *comEquiv* rule and facts for AFs.

### 3.2.5 C-relativised Grounded Equivalency

To compute *C-relativised grounded equivalency* another version of the characteristic function (i.e. the *C-relativised grounded characteristic function*) is needed. It is given by:

$$\mathcal{F}_{F,E}(S) = \{\, a \in E \mid S \text{ defends } a \text{ in } F \,\}\text{, where } E \subseteq A(F)$$

Here, we only consider defended arguments restricted to some set $E$ instead of all arguments in $A(F)$.

*C-relativised grounded equivalency* shares conditions (1) and (2) with complete equivalency, but replaces condition (3), making use of this revised characteristic function:

**Theorem 5.** *[BDLW17] Two AFs $F, G$ are* C-relativised grounded equivalent *(i.e. $F \equiv_C^{grd} G$) iff the following conditions hold:*

*(1)* $A(F)\backslash C = A(G)\backslash C$.

*(2)* $grd_C(F) = grd_C(G)$.

*(3)* $\forall E \in grd_C(F), \forall X \subseteq A(F)\backslash(C \cup E)$ *the following conditions hold:*

- *(a)* $\mathcal{F}^{\infty}_{F\backslash X, E\cup C}(\emptyset) = \mathcal{F}^{\infty}_{G\backslash X, E\cup C}(\emptyset)$.
- *(b) If* $\mathcal{F}^{\infty}_{F\backslash X, E\cup C}(\emptyset) = E$ *then* $E^+_F\backslash(C \cup X) = E^+_G\backslash(C \cup X)$.
- *(c) If* $\mathcal{F}^{\infty}_{F\backslash X, E\cup C}(\emptyset) = E$ *then* $\mathcal{F}_{F\backslash X}(E) = \mathcal{F}_{G\backslash X}(E)$.

Note that we again, like with *C-restricted grounded extensions*, compute the fixed point of the new characteristic function starting with the empty set as its initial parameter.

**Example 16.** *We continue with the AFs introduced in Example 14: Firstly, we compute that condition (1) and (2) hold, since $grd_C(F_1) = grd_C(F_2) = \{\emptyset, \{a\}\}$ and these extensions only contain core arguments. Condition (3) is checked for all C-restricted grounded extensions against all sets of arguments $X \subseteq A(F)\backslash(C \cup E)$. So we check against $\{c\}$ and $\emptyset$, which are the only such $X$ regarding $F_1$.*

*This step entails computation of the fixed point of the C-relativised grounded characteristic function. For $E = X = \emptyset$ we therefore start by calculating $\mathcal{F}_{F\backslash\emptyset,\emptyset\cup C}(\emptyset) = \emptyset$. Now $\emptyset$ would be the new parameter for the fixed point calculation, but this is the same calculation, so $\mathcal{F}^{\infty}_{F\backslash\emptyset,\emptyset\cup C}(\emptyset) = \emptyset$. For $G$ we get $\mathcal{F}_{G\backslash\emptyset,\emptyset\cup C}(\emptyset) = \emptyset$ for which we know already that it is the fixed point. Computation of the other combinations of such $E$ and $X$ leads to the following results:*

$$\mathcal{F}^{\infty}_{F_1\backslash\{c\},\emptyset\cup C}(\emptyset) = \mathcal{F}^{\infty}_{F_2\backslash\{c\},\emptyset\cup C}(\emptyset) = \{a\}$$
$$\mathcal{F}^{\infty}_{F_1\backslash\emptyset,\{a\}\cup C}(\emptyset) = \mathcal{F}^{\infty}_{F_2\backslash\emptyset,\{a\}\cup C}(\emptyset) = \emptyset$$
$$\mathcal{F}^{\infty}_{F_1\backslash\{c\},\{a\}\cup C}(\emptyset) = \mathcal{F}^{\infty}_{F_2\backslash\{c\},\{a\}\cup C}(\emptyset) = \{a\}$$

*This means (3a) holds.*

*Conditions (3b) and (3c) share a precondition that must hold. This is the case for $\mathcal{F}^{\infty}_{F_1\backslash\emptyset,\emptyset\cup C}(\emptyset) = \emptyset$ and $\mathcal{F}_{\infty}{}^{\infty}_{F\backslash\{c\},\{a\}\cup C}(\emptyset) = \{a\}$.*

*Since $\{a\}$ does not attack any arguments outside of $C$ and $\emptyset$ does not attack anything in either AF (3b) holds.*

*Condition (3c) also holds because neither* $\{a\}$ *nor* $\emptyset$ *defend any arguments. Hence condition (3) holds and* $F_1 \equiv_C^{grd} F_2$.

We introduce the encoding of this fixed point characteristic function in three parts: (1) $\pi_{\mathrm{grd}_D} = \pi_{\mathrm{grd}_{D_a}} \cup \pi_{\mathrm{grd}_{D_b}}$, where all arguments that are defended according to the characteristic function are found, (2) $\pi_{\mathrm{grd}_E}$ which finds the results of the characteristic function and (3) $\pi_{\mathrm{grd}_F}$, in which the rules for finding the fixed point are introduced.

To compute the properly defended arguments w.r.t. to the C-relativised grounded characterstic function, first the rule *argDef* together with its auxiliary rule *hasAttack* is introduced in Equation 3.33.

*hasAttack*$(F, X, A)$ is true for all arguments $A$ which are attacked by some $B \in A(F)\backslash X$, whereas *argDef*$(F, E, X, S, A, B)$ holds iff some set $S$ defends some argument (1) $A \in E$ or (2) $A \in C$, against some argument $B \in A(F)\backslash X$, where $X \subseteq A(F)\backslash(C \cup E)$ or if $A$ is not attacked in $F\backslash X$. Recall, that a rule to determine whether $X \subseteq A(F)\backslash(C \cup E)$ holds is given by *SinA* (see 3.27) and does not need to be separately implemented.

For the rules to work that way, there are two double rules that are both called *argDef*, one considering only arguments that are attacked, the other only those which are not attacked. The second version of both *argDef* rules is obtained by replacing *#member*$(A, E)$ with *core*$(R)$ and *#member*$(A, R)$.

$$
\begin{aligned}
\pi_{\mathrm{grd}_{D_a}} = \quad & \mathrm{hasAttack}(F, X, A) \leftarrow \mathrm{argument}(F, A), \mathrm{argument}(F, B), \\
& \qquad \mathrm{attack}(F, B, A), \mathrm{not}\ \#\mathrm{member}(A, X), \\
& \qquad \mathrm{not}\ \#\mathrm{member}(B, X), \mathrm{SinA}(F, E, X). \\
& \mathrm{argDef}(F, E, X, S, A, B) \leftarrow \mathrm{argument}(F, A), \mathrm{argument}(F, B), \\
& \qquad \mathrm{argument}(F, C), \mathrm{argSet}(F, S), \mathrm{grdc}(F, E), \\
& \qquad \mathrm{attack}(F, B, A), \mathrm{attack}(F, C, B), \\
& \qquad \#\mathrm{member}(A, E), \mathrm{not}\ \#\mathrm{member}(A, X), \\
& \qquad \mathrm{not}\ \#\mathrm{member}(B, X), \mathrm{not}\ \#\mathrm{member}(C, X), \\
& \qquad \#\mathrm{member}(C, S), \mathrm{SinA}(F, E, X). \\
& \mathrm{argDef}(F, E, X, S, A, B) \leftarrow \mathrm{argument}(F, A), \mathrm{argument}(F, B), \\
& \qquad \mathrm{argSet}(F, S), \mathrm{grdc}(F, E), \#\mathrm{member}(A, E), \\
& \qquad \mathrm{not}\ \#\mathrm{member}(A, X), \mathrm{not}\ \#\mathrm{member}(B, X), \\
& \qquad \mathrm{SinA}(F, E, X), \mathrm{not}\ \mathrm{hasAttack}(F, X, A).
\end{aligned}
\tag{3.33}
$$

The other two rules (3.34) needed to collect the arguments that are the result of the C-relativised grounded characteristic function, are *notFullDef* and *fullDef*. On one hand

*notFullDef*$(F, E, X, S, A)$ holds iff $A$ is not defended against all $B \in A(F) \backslash X$. On the other hand *fullDef*$(F, E, X, S, A)$ holds iff *notFullDef* does not.

The same rule replacement as with *argDef* also results in the double rules for *notFullDef* and *fullDef*.

$$
\begin{aligned}
\pi_{\mathrm{grd}_{D_b}} = \mathrm{notFullDef}(F, E, X, S, A) \leftarrow & \mathrm{argument}(F, A), \mathrm{argument}(F, B), \mathrm{argSet}(F, S), \\
& \mathrm{grdc}(F, E), \mathrm{attack}(F, B, A), \#\mathrm{member}(A, E), \\
& \mathrm{not}\ \#\mathrm{member}(A, X), \mathrm{not}\ \#\mathrm{member}(B, X), \\
& \mathrm{SinA}(F, E, X), \mathrm{not}\ \mathrm{argDef}(F, E, X, S, A, B). \\
\mathrm{fullDef}(F, E, X, S, A) \leftarrow & \mathrm{argument}(F, A), \mathrm{argSet}(F, S), \mathrm{grdc}(F, E), \\
& \#\mathrm{member}(A, E), \mathrm{not}\ \#\mathrm{member}(A, X), \\
& \mathrm{SinA}(F, E, X), \mathrm{not}\ \mathrm{notFullDef}(F, E, X, S, A).
\end{aligned} \tag{3.34}
$$

Once all properly defended arguments are found, they are collected iteratively, by starting with $\emptyset$ and step-by-step adding new arguments, which fulfill the necessary conditions and subsequently choosing the maximum such set computed for further computation.

*defList*$(F, E, X, S, N)$ holds for all argument sets $N \subseteq \{\, a \in E \cup C \mid S$ defends $a$ in $F \backslash X \,\}$. Its encoding is shown in $\pi_{\mathrm{grd}_{E_1}}$ (see Equation 3.35).

$$
\begin{aligned}
\pi_{\mathrm{grd}_{E_1}} = \mathrm{defList}(F, E, X, S, []) \leftarrow & \mathrm{argSet}(F, S), \mathrm{grdc}(F, E), \mathrm{SinA}(F, E, X). \\
\mathrm{defList}(F, E, X, S, N) \leftarrow & \mathrm{argument}(F, A), \mathrm{argSet}(F, S), \mathrm{grdc}(F, E), \\
& \#\mathit{append}([], [A], N), \mathrm{SinA}(F, E, X), \\
& \mathrm{fullDef}(F, E, X, S, A). \\
\mathrm{defList}(F, E, X, S, N) \leftarrow & \mathrm{argument}(F, A), \mathrm{argSet}(F, S), \mathrm{grdc}(F, E), \\
& \#\mathit{append}(O, [A], N), \mathrm{SinA}(F, E, X), \\
& \mathrm{fullDef}(F, E, X, S, A), \mathrm{defList}(F, E, X, S, O), \\
& \#\mathit{last}(O, B), B < A.
\end{aligned} \tag{3.35}
$$

To only get the subset maximal sets produced by *defList*, we first define *subDefLists*, which holds for such $N$ that are not maximal and use those as intermediate results. Subsequently, *maxDefList* holds for maximal $N$ by filtering out all those lists returned by *subDefLists*. Both these predicates are defined in Equation 3.36 as $\pi_{\mathrm{grd}_{E_2}}$. We combine these with the previous rules and say $\pi_{\mathrm{grd}_E} = \pi_{\mathrm{grd}_{E_1}} \cup \pi_{\mathrm{grd}_{E_2}}$.

$$
\begin{aligned}
\pi_{\mathrm{grd}_{E_2}} = \quad & \mathrm{subDefList}(F,E,X,S,L) \leftarrow \mathrm{argSet}(F,S), \mathrm{grdc}(F,E), \mathrm{SinA}(F,E,X), \\
& \qquad \mathrm{defList}(F,E,X,S,L), \mathrm{defList}(F,E,X,S,G), \\
& \qquad \#\mathrm{subList}(L,G), L \neq G. \\
& \mathrm{maxDefList}(F,E,X,S,L) \leftarrow \mathrm{argSet}(F,S), \mathrm{grdc}(F,E), \mathrm{SinA}(F,E,X), \\
& \qquad \mathrm{defList}(F,E,X,S,L), \\
& \qquad \mathrm{not\ subDefList}(F,E,X,S,L).
\end{aligned} \tag{3.36}
$$

The predicates for finding the fixed point of this characteristic function work in a similar fashion as the fixed point computation for *C-restricted grounded extensions* (3.17), again using the number of arguments as an upper bound for the number of computation steps.

While *defIteration* computes all steps, *eqDefIteration* checks if the result of some step equals that of another one, *notFirstDefEq* finds later steps fulfilling this condition and *firstDefEq* holds for the first iteration step that did not add any new arguments to the input parameter.

$$
\begin{aligned}
\pi_{\mathrm{grd}_F} = \quad & \mathrm{defIteration}(F,E,X,[],Rs,0) \leftarrow \mathrm{grdc}(F,E), \mathrm{maxDefList}(F,E,X,[],Rs). \\
& \mathrm{defIteration}(F,E,X,Pr,Rs,N) \leftarrow \mathrm{grdc}(F,E), \mathrm{SinA}(F,E,X), \\
& \qquad \mathrm{maxDefList}(F,E,X,Pr,Rs), \\
& \qquad \mathrm{defIteration}(F,E,X,S,Pr,M), \\
& \qquad \#\mathrm{count}\{As : \mathrm{argument}(F,As)\} = C, \\
& \qquad \#\mathrm{int}(M), \#\mathrm{int}(N), M = N-1, N \leq C. \\
& \mathrm{eqDefIteration}(F,E,X,Rs,N) \leftarrow \mathrm{defIteration}(F,E,X,Rs,Rs,N). \\
& \mathrm{notFirstDefEq}(F,E,X,Rs,N) \leftarrow \mathrm{eqDefIteration}(F,E,X,Rs,M), \\
& \qquad M < N, \#\mathrm{int}(N), \#\mathrm{int}(M). \\
& \mathrm{firstDefEq}(F,E,X,Rs,N) \leftarrow \mathrm{SinA}(F,E,X), \\
& \qquad \mathrm{eqDefIteration}(F,E,X,Rs,N), \\
& \qquad \mathrm{not\ notfirstDefEq}(F,E,X,Rs,N), \#\mathrm{int}(N).
\end{aligned} \tag{3.37}
$$

The rules included in $\pi_{\mathrm{grd}_G}$ use the results of the fixed point calculation to evaluate equivalence conditions (3a)-(c).

*equalDefFix*$(F, G, E, X)$ holds for AFs $F, G$ iff $\mathcal{F}^{\infty}_{F\setminus X, E\cup C}(\emptyset) = \mathcal{F}^{\infty}_{G\setminus X, E\cup C}(\emptyset)$ and thus encodes condition (3a).

Conditions (3b) and (3c) both neccessitate for $\mathcal{F}^{\infty}_{F\setminus X, E\cup C}(\emptyset) = E$ to be satisfied. *correctDef*$(F, E, X)$ holds iff this is the case.

The predicate *excludedNonEq*$(F, G, E, X)$ is the encoding of the right side of (3b) and holds iff $E^{+}_{F}\setminus(C \cup X) = E^{+}_{G}\setminus(C \cup X)$.

Finally, to encode the right side of (3c) two rules are introduced: *withoutXDiff* and *withoutXEq*.

*withoutXDiff*$(F, G, S, E)$ holds iff $F, G$ have different fixed points, while *withoutXEq* is derived iff $F, G$ have equal fixed points. To implement this, the rule *xinEplus* introduced in 3.28 is reused and must hold for $F$, but can not hold for $G$ for *withoutXDiff* to be true.

$$
\begin{aligned}
\pi_{\mathrm{grd}_G} = \quad & \mathrm{equalDefFix}(F, G, E, X) \leftarrow \mathrm{SinA}(F, E, X), \mathrm{firstDefEq}(F, E, X, Res, N), \\
& \qquad \mathrm{firstDefEq}(G, E, X, Res, M), \#\mathrm{int}(N), \#\mathrm{int}(M), \\
& \qquad F \neq G. \\
& \mathrm{correctDef}(F, E, X) \leftarrow \mathrm{firstDefEq}(F, E, X, E, N), \#\mathrm{int}(N). \\
& \mathrm{excludedNonEq}(F, G, E, X) \leftarrow \mathrm{argument}(F, A), \mathrm{grdc}(F, E), \mathrm{grdc}(G, E), \mathrm{core}(C), \\
& \qquad \mathrm{isAttacked}(F, A, E), \mathrm{not\ isAttacked}(G, A, E), \\
& \qquad \mathrm{not}\ \#\mathrm{member}(A, C), \mathrm{not}\ \#\mathrm{member}(A, X), \\
& \qquad \mathrm{SinA}(F, E, X). \\
& \mathrm{withoutXDiff}(F, G, S, E) \leftarrow \mathrm{argument}(F, X), \mathrm{argument}(G, X), \\
& \qquad \mathrm{xinEplus}(F, S, X, E), \mathrm{not\ xinEplus}(G, S, X, E). \\
& \mathrm{withoutXEq}(F, G, S, E) \leftarrow \mathrm{grdc}(F, E), \mathrm{grdc}(G, E), \mathrm{SinA}(F, E, S), \\
& \qquad \mathrm{not\ withoutXDiff}(F, G, S, E), F \neq G.
\end{aligned}
\tag{3.38}
$$

To find all C-restricted grounded extensions $E$ that at least violate one of the conditions stated in (3) with some set $X \subseteq A(F)\setminus(C \cup E)$, the predicate *EXFalse* is introduced (3.39). Here all the rules in $\pi_{\mathrm{grd}_D} \cup \pi_{\mathrm{grd}_E} \cup \pi_{\mathrm{grd}_F} \cup \pi_{\mathrm{grd}_G}$ are combined.

$$
\begin{aligned}
\pi_{\mathrm{grd}_H} = \mathrm{EXFalse}(F,G,E,X) \leftarrow & \mathrm{grdc}(F,E), \mathrm{grdc}(G,E), \mathrm{SinA}(F,E,X), \\
& \text{not } \mathrm{equalDefFix}(F,G,E,X). \\
\mathrm{EXFalse}(F,G,E,X) \leftarrow & \mathrm{grdc}(F,E), \mathrm{SinA}(F,E,X), \mathrm{correctDef}(F,E,X), \\
& \mathrm{excludedNonEq}(F,G,E,X). \\
\mathrm{EXFalse}(F,G,E,X) \leftarrow & \mathrm{grdc}(F,E), \mathrm{grdc}(G,E), \mathrm{SinA}(F,E,X), \\
& \mathrm{correctDef}(F,E,X), \text{not } \mathrm{withoutXEq}(F,G,X,E).
\end{aligned} \tag{3.39}
$$

To compute C-relativised grounded equivalency, we use the analog of the *outDiff* and *difference* rules as for C-relativised admissible equivalency. Additionally the following inequivalency double rule is introduced (3.40):

$$
\begin{aligned}
\pi_{\mathrm{grd}_{\mathrm{ineq}}} = \mathrm{notGrdEquiv}(F,G) \leftarrow & \mathrm{grdc}(F,E), \mathrm{grdc}(G,E), \mathrm{SinA}(F,E,X), \\
& \mathrm{EXFalse}(F,G,E,X), F < G.
\end{aligned} \tag{3.40}
$$

This rule holds iff for two AFs $F, G$ there is some $X \subseteq A(F) \backslash (C \cup E)$ for which *EXFalse* holds.

Notice that the double version of this rule again only exchanges $F < G$ for $F > G$.

C-relativised grounded equivalence also possesses a *grdEquiv* rule (3.41) that works exactly as for the other semantics:

$$
\pi_{\mathrm{grd}_{eq}} = \mathrm{grdEquiv}(F,G) \leftarrow \text{not } \mathrm{notGrdEquiv}(F,G), \mathrm{af}(F), \mathrm{af}(G), F < G. \tag{3.41}
$$

This rule is used in conjunction with facts for AFs that we want to compare, the rules in $\pi_{\mathrm{grd}_H} \cup \pi_{\mathrm{grd}_{\mathrm{ineq}}} \cup \pi_{\mathrm{grd}_D} \cup \pi_{\mathrm{grd}_E} \cup \pi_{\mathrm{grd}_F} \cup \pi_{\mathrm{grd}_G}$ and the rules needed to compute C-restricted grounded extensions to finally compute whether these AFs are C-relativised grounded equivalent.

## 3.3 Summary

In this chapter we examined *C-restricted semantics* ([BDLW17]), which are local versions of the semantics originally introduced in [Dun95]. For each of these semantics an encoding as an answer-set program was given and analysed. These programs are able to compute all *C-restricted extensions* of arbitrary argumentation frameworks given some domain and a set of core arguments.

For that purpose, we first defined a format for representing AFs, domains and cores. Then some basic encodings that are used for most of the further computations were given, followed finally by the encodings for the C-restricted semantics themselves.

Further, we analysed how to compare AFs to evaluate whether they are *C-relativised equivalent* for any of the considered semantics. Again, ASP-encodings expanding upon the encodings of the C-restricted semantics were provided to compare an arbitrary number of AFs and detailed step-by-step descriptions of how they work were given. All encodings are availiable at [Bel18a].

In the following chapters we describe the results obtained by systematically analysing the semantics of and equivalencies between large numbers of AFs as well as how these results can be used for simplification of large AFs.

CHAPTER 4

# C-relativised Equivalency Patterns

In this chapter the formal definition for *C-relativised equivalency patterns* is introduced. This notion can then be used to describe pairs of frameworks that are freely interchangeable if found within a larger AF. Furthermore, we present concrete patterns as well as general observations about C-relativised equivalence that have been found by generating AFs and evaluating them for equivalence (w.r.t. the semantics discussed herein) by using an implementation of the encodings described in chapter 3. Notice, that we are omitting grounded semantics since grounded extensions are computed using the same characteristic function as complete extensions and hence it suffices to apply pre-processing for complete semantics. Moreover, grounded extensions can be computed very efficiently without pre-processing.

The implementation of the encodings can be found at [Bel18a].

## 4.1 Pattern Definition

As shown in [BDLW17] it is possible to locally replace parts of an AF without losing any information regarding the semantics, i.e. the replacement patterns are given by two graphs such that if we find one of them as a subgraph of the AF we can replace it with the other without affecting the extensions. If such a replacement can be generalised to be not only applicable to a single graph but a class of similar graphs, we call it a *C-relativised equivalency pattern*. Before examining individual patterns though, a formal definition for what such a pattern actually is, is provided.

**Definition 19.** *A 6-tuple of form* $(\sigma, F', G', P^+, P^-, C)$ *is called a* C-relativised equivalency pattern *with regards to some semantics* $\sigma \in$ {*stb,adm,prf,com,grd*} *(short:* $\sigma$*-pattern), if it satisfies the following:*

*Let* $F' = (A_{F'}, R_{F'})$ *and* $G' = (A_{G'}, R_{G'})$*. For any* $H = (A_H, R_H)$ *such that for* $A_H \cap C = \emptyset$*:*

- *there is no* $(x, y) \in R_H$ *with* $x \in P^+$ *and*
- *there is no* $(x, y) \in R_H$ *with* $y \in P^-$*,*

*we then have* $F' \cup H \equiv^{\sigma}_{C} G' \cup H$*.*

Notice that $H$ is not necessarily an AF itself. It might therefore contain attacks involving arguments in $C$, though it never contains arguments from $C$.

We denote as complements the sets of arguments that are not prohibited from attack relations, i.e. $\overline{P^+} = C \backslash P^+$ and $\overline{P^-} = C \backslash P^-$.

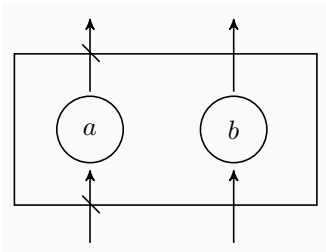



Figure 4.1: $P^+ = P^- = \{a\}$, i.e. $a$ can not attack or be attacked by non-core arguments, whereas $b$ is allowed to (i.e. $b \in \overline{P^+}, b \in \overline{P^-}$).

Prohibited attacks can be depicted as a crossed-out arrow, whereas attacks that are not prohibited can be shown as just an arrow, both not connected to any specific argument. Figure 4.1 shows examples of such attacks.

Note, that from here on instead of saying $E$ is a C-restricted $\sigma$ extension, $E$ may be simply called C-$\sigma$ for $\sigma = \{\text{stb}, \text{adm}, \text{prf}, \text{com}, \text{grd}\}$ (e.g. $E$ is C-stable means that $E$ is a C-restricted stable extension). Analoguously, AFs $F$ and $G$ being C-$\sigma$ equivalent means they are C-relativised $\sigma$ equivalent.

## 4.2 Patterns for Stable Semantics

In this section we look at properties of C-stable equivalency and present some C-stable patterns.

C-stable semantics are only concerned with attacks on core arguments. Also recall Figure 2.3, which shows that stable semanctis are the most specialised semantics amongst those that are discussed in this thesis in the sense that the stable extensions are also extensions w.r.t. the other semantics but not vice versa. This means that there may be properties of C-stable patterns, which are not applicable to other semantics, since they take into account more relations. It follows an analysis of such a property:

**Proposition 1.** *Let AFs $F \equiv_C^{stb} G$, where $A(F)\backslash C = A(G)\backslash C$ and $R(F)\backslash C^2 = R(G)\backslash C^2$ and let $R'_F \subseteq \{(x,y)|x \in C \cap A(F), y \notin C\}, R'_G = \{(x,y)|(x,y) \in R'_F, x \in A(G)\}$ with arbitrary fixed $C$. It holds that $F' \equiv_C^{stb} G'$ for $F' = F \cup (\emptyset, R'_F), G' = G \cup (\emptyset, R'_G)$.*

*Proof.* To prove Proposition 1, the three conditions for C-relativised stable equivalency must hold for all pairs of AFs $F', G'$, where $F \equiv_C^{stb} G$.

(1) If $stb_C(F') \neq \emptyset, A(F')\backslash C = A(G')\backslash C$. This condition holds, since $A(F) = A(F')$, $A(G) = A(G')$ and $A(F)\backslash C = A(G)\backslash C$.

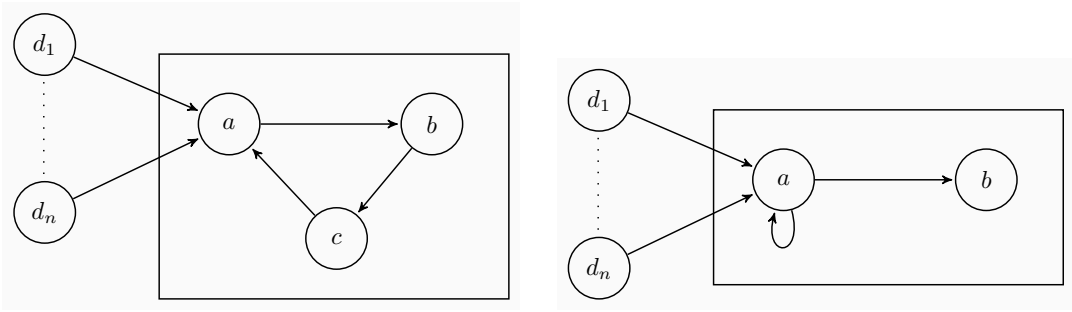
(2) $stb_C(F') = stb_C(G')$. We have that if $E \in stb_C(F')$ then also $E \in stb_C(F)$ as well as if $E \in stb_C(G')$ then also $E \in stb_C(G)$. This means it suffices to show that for every $E \subseteq A(F)$, if $E \in stb_C(F)$:

- $E \notin cf(F')$ iff $E \notin cf(G')$. Assume $E \in cf(F')$ and $E \notin cf(G')$, then there exists an attack $(x,y) \in R(G')$ s.t. $x, y \in E$ and $(x,y) \notin R(F')$. We know $y \in A(G)\backslash C = A(F)\backslash C$, which means $x \in A(G), x \notin A(F)$ must hold, which it can not, since $E \in cf(F), E \in cf(G)$.
- $E \notin stb_C(F')$ iff $E \notin stb_C(G')$. Assume $E \in stb_C(F')$ and $E \notin stb_C(G')$, it can mean two things. Firstly, that there is an attack $(x,y)$ s.t. $E \in cf(F), E \notin cf(G)$, which is not possible as is shown above. Secondly, that there is an attack $(x,y)$ s.t. $A(F') \cap C \subseteq E^{\oplus}_{F'}, A(G') \cap C \nsubseteq E^{\oplus}_{G'}$. Since $R'_F$ and $R'_G$ do not contain attacks on arguments in $C$ and we know $A(F) \cap C \subseteq E^{\oplus}_F$ and $A(G) \cap C \subseteq E^{\oplus}_G$, this can not be the case.
- $E \in stb_C(F')$ iff $E \in stb_C(G')$. To not be true, one must assume $E \in stb_C(F')$ and $E \notin stb_C(G')$, which can not happen, as is shown above.

(3) For all $E \in stb_C(F'), E^+_{F'}\backslash C = E^+_{G'}\backslash C$. This means for all $y \in A(F')\backslash C$ one of the following cases must hold:

- $y \in E^+_{F'}\backslash C, y \in E^+_{G'}\backslash C$.
- $y \notin E^+_{F'}\backslash C, y \notin E^+_{G'}\backslash C$.

Assume that $y \in E^+_{F'}\backslash C, y \notin E^+_{G'}\backslash C$. This means there is an argument $x \in E$ with an attack $(x,y) \in R(F'), (x,y) \notin R(G')$ and there is no argument $x' \in E$ s.t. $(x',y) \in R(G')$. As $x \in E$ we know $x \in A(F)$ and $x \in A(G)$. This means




(a) (b)

Figure 4.2: Argumentation frameworks $F'$ (a) and $G'$ (b), are C-relativised equivalent w.r.t stable semantics.

$(x, y) \notin R'_F$, as $x \in A(G)$ and thus if $(x, y) \in R'_F$ then also $(x, y) \in R'_G$. It follows that $(x, y) \in R(F)$. Additionally, as there is no $x' \in E$ s.t. $(x', y) \in R(G')$ it also holds that $\nexists x' \in E : (x', y) \in R(G)$, which would mean that $y \in E^+_F \backslash C$, but $y \notin E^+_G \backslash C$ and thus $F \not\equiv^{stb}_C G$.

Since all three conditions hold, Proposition 1 holds. □

We can now use Proposition 1 to prove concrete patterns more easily. One instance where this is useful is depicted in Figure 4.2 and its pattern is described in Proposition 2.

**Proposition 2.** *The pattern* $(stb, F', G', \emptyset, \{b, c\}, \{a, b, c\})$ *with* $F' = (\{a, b, c\}, \{(a, b), (b, c), (c, a)\})$ *and* $G' = (\{a, b\}, \{(a, a), (a, b)\})$ *holds.*

*Proof.* To prove $F \equiv^{stb}_C G$ three conditions are shown to hold. For this purpose let $F := F' \cup H, G := G' \cup H$, where $H = (\{d_1, ..., d_n\}, \{(d_1, a), ..., (d_n, a)\})$. Notice that by Proposition 1 there is no need to consider attacks $(a, x)$ or $(b, x)$ for $x \notin C$. Also, it suffices to show C-relativised equivalence for this specific $H$, since $P^-$ only allows $H$ to attack $a$ and any additional arguments and attacks in $H$ do not interact with $C$ and therefore behave the same for both $F$ and $G$.

(1) If $stb_C(F) \neq \emptyset, A(F)\backslash C = A(G)\backslash C$. This holds since $A(F) = A(G) \cup \{c\}$ and $c \in C$.

(2) $stb_C(F) = stb_C(G)$. Assume $stb_C(F) \neq stb_C(G)$, then that would mean there exists a set $E$ s.t. either:

- $A(F) \cap C \subseteq E^{\oplus}_F, A(G) \cap C \not\subseteq E^{\oplus}_G$. This means there is an argument $x \in C, x \in A(F), x \in A(G), x \notin E$ and there is an argument $y \in E$ with $(y, x) \in R(F)$, but there is no argument $y' \in E$ with $(y', x) \in R(G)$. This must hold for all $x \in \{a, b\}$, since $c \notin A(G)$.

Notice that $b \in E$ for all $E \in stb_C(F)$ and $E \in stb_C(G)$. In the former case, since there are no attacks on $c$ except by $b$, and in the latter case because $(a, a) \in R(G)$ and $\{b\}^- = \{a\}$. Moreover each $E$ contains at least one $d_i$ s.t. $(d_i, a) \in R(F), (d_i, a) \in R(G)$. Thus $\{a, b\} \subseteq E_G^{\oplus}$.

– $A(F) \cap C \not\subseteq E_F^{\oplus}, A(G) \cap C \subseteq E_G^{\oplus}$. This means there is an argument $x \in C, x \in A(F), x \in A(G), x \notin E$ and there is an argument $y \in E$ with $(y, x) \in R(G)$, but there is no argument $y' \in E$ with $(y', x) \in R(F)$. This must hold for $x \in \{a, b\}$, since $c \notin A(G)$.

  Again, notice that $b \in E$ for all $E \in stb_C(F)$ and $E \in stb_C(G)$. In the former case, since there are no attacks on $c$ except by $b$, and in the latter case because $(a, a) \in R(G)$ and $\{b\}^- = \{a\}$. Moreover each $E$ contains at least one $d_i$ s.t. $(d_i, a) \in R(F), (d_i, a) \in R(G)$. Therefore $\{a, b\} \subseteq E_F^{\oplus}$.

(3) For all $E \in stb_C(F), E_F^+ \backslash C = E_G^+ \backslash C$. This holds since $R(F) \backslash C = R(G) \backslash C$ and (2) holds.

□

Related to the fact that additional attacks by core arguments onto non-core arguments do not change C-stable equivalencies, in Proposition 3 it is examined whether the reverse (i.e. non-core argument attacks onto core arguments not influencing the results of the equivalence relation) also holds:

**Proposition 3.** *Given the definitions of Proposition 2 it can not be guaranteed that* $F \cup (\emptyset, \{(x, y)\}) \equiv_C^{stb} G \cup (\emptyset, \{(x, y)\})$ *holds for attacks* $(x, y)$*, where* $x \in A(F) \backslash C$ *and* $y \in C \backslash \{a\}$*.*

*Proof.* This is proven by a case analysis for all arguments $C \backslash \{a\}$ separately.

Adding an attack $(x, b)$ may affect $F, G$ s.t. $F \not\equiv_C^{stb} G$. Let $A(F) = \{a, b, c, d_1\}$ (and therefore $(d_1, a) \in R_F$) and $x = d_1$, then $\{d_1, c\} \in stb_C(F)$, but $c \notin A(G)$ and $\{d_1, c\} \notin stb_C(G)$.

Adding an attack $(x, c)$ may also affect $F, G$ s.t. $F \not\equiv_C^{stb} G$. Let $A(F) = \{a, b, c, e_1\}$ and $x = e_1$, then $\{e_1, a\} \in stb_C(F)$, but $\{e_1, a\} \notin stb_C(G)$, since $(a, a) \in R(G)$ and $\{e_1, a\} \notin cf(G)$. □

As a result of Proposition 1, in figures depicting C-stable patterns, out-arrows for arguments in $\overline{P^+}$ can be assumed, but visually omitted, since C-stable patterns always have $P^+ = \emptyset$. For similar reasons (see Proposition 3) crossed-out in-arrows can also be generally assumed and visually omitted. Also notice that depictions of pattern *instances* (i.e. $F'$ and $G'$ are shown) do not show the restrictions on $H$.

Further building upon Proposition 2, we show that a similar reduction is also possible for larger cycles of core arguments, as long as they contain an odd amount of arguments:

**Proposition 4.** *The pattern* $(stb, F', G', \emptyset, \{x_2, ..., x_n\}, \{x_1, ..., x_n\})$ *with* $F' = (\{x_1, ..., x_n\}, \{(x_1), (x_1, x_2), ..., (x_{n-1}, x_n), (x_n, x_1)\})$*, where* $n$ *is odd, and* $G' = (A_{F'} \backslash \{x_n\}, (R_{F'} \cup \{(x_1, x_1)\}) \backslash \{x_n, x_1\})$ *holds.*

*Proof.* To prove Proposition 4 let $F := F' \cup H, G := G' \cup H$ where $\{a_1, ..., a_k\} \subseteq A_H$ and $\{(a_1, x_1), ..., (a_k, x_1)\} \subseteq R_H$. As for Proposition 2, by Proposition 1 it is not necessary to examine any attacks $(x_i, y)$ where $y \notin C$ and by definition of $P^-$ only attacks on $x_1$ need to be considered. Furthermore all other possible arguments in $H$ do not interact with $C$ and therefore behave the same in $F$ and $G$. Then:

(1) $A(F) \backslash C = A(G) \backslash C$ as per defintion of this pattern.

(2) We check whether both AFs have the same C-stable extensions. In $F$ every $x_i$, where $i$ is even, attacks a $x_j$, $j$ being odd, and thus defends $x_{i+2}$. This means, since any $a_l$ attacks $x_1$ and therefore defends $x_2$, we know that all C-stable extensions contain $X \cup \{x_2, x_4, ..., x_{n-1}\}$, where $X \subseteq \{a_1, ..., a_k\}, X \neq \emptyset$. For $G$ the same applies, since $x_2$ is defended by any $a_l$ and therefore an additional defense is not needed.

    Now, if there exists any attack $(y, a_l)$, either (1) $y \in H$ and therefore $y \notin C$, which means defence against $y$ is not needed as per definition of C-stable extensions (see 3.1.3) in neither $F$ nor $G$, or (2) $y \in C$. If the latter is the case $y$ is either $x_i$ with odd or with even $i$. In case $i$ is odd, $y$ is attacked by $E$ in both AFs. If $i$ instead is even, $E$ is not C-stable in either framework. Hence $\text{stb}_C(F) = \text{stb}_C(G)$.

(3) $\forall E \in \text{stb}_C(F) : E_F^+ \backslash C = E_G^+ \backslash C$ can only be falsified by attacks from $x_n$ onto some $y \notin C, y \in A_F$, since all other arguments possess the same attacks and are both in $F$ and $G$, but $n$ is odd and so $x_n$ can not be in $E$. Hence the condition is satisfied.

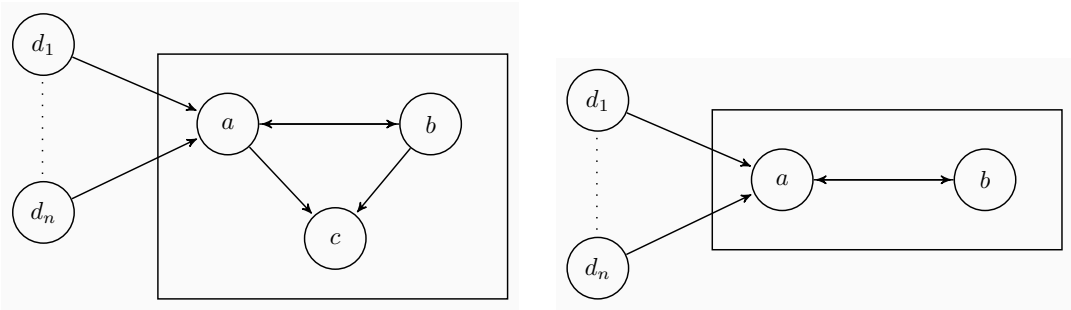
□

Another observation (see Proposition 5) follows directly from the definition of C-relativised stable equivalence in conjunction with Proposition 1. In certain cases we can remove from an AF core arguments that are not defended against another core argument (provided this argument does not attack itself). Figure 4.3 depicts two AFs $F', G'$, where the removal of such arguments from $F'$ results in $G'$.

**Proposition 5.** *Let* $F, G$ *be two AFs where* $F = (A(F), R(F))$*. Now let* $X_A = \{a \in C | \exists b \in C : (b, a) \in R(F), (a, b) \notin R(F), \forall (c, b) \in R_F : (c, a) \in R(F)\}$ *and* $X_R = \{(a, b) | a \in X_A \vee b \in X_A\}$*. For* $G = (A(F) \backslash X_A, R(F) \backslash X_R)$ *we then have* $G \equiv_C^{stb} F$*.*

*Proof.* We prove Proposition 5:

(1) Since $A(G) = A(F) \backslash X_A$ and $X_A$ only contains core arguments condition (1) holds.




(a) (b)

Figure 4.3: Argumentation frameworks $F'$ (a) and $G'$ (b), are also C-relativised equivalent w.r.t stable semantics.

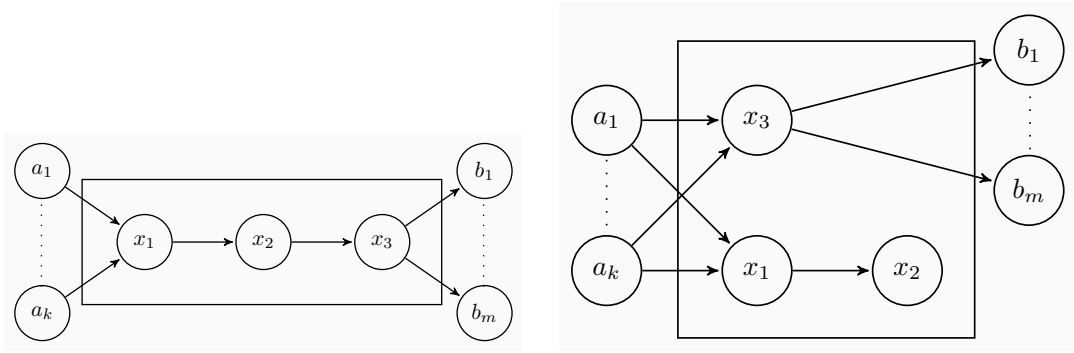
(2) It must be proven that $stb_C(F) = stb_C(G)$.

- Assume there exists $E \in stb_C(F), E \notin cf(G)$. This is impossible, since $R(G) \subseteq R(F)$ and $E \in cf(F)$.
- Assume there exists $E \in stb_C(G), E \notin cf(F)$. This means there is an attack $(a, b) \in X_R$ and $a, b \in E$, but since for all $(a, b) \in X_R$ either $a \in X_A$ or $b \in X_A$ and $A(G) \cap X_A = \emptyset$, this can not be the case.
- Assume there exists $E \in stb_C(F)$ and $E \notin stb_C(G)$. This means there exists an argument $x \in A(G) \cap C$ that is not in $E_G^{\oplus}$. Since $E \in stb_C(F)$, we know that $x \notin E_G^{+}$, but this can not be the case since only attacks involving arguments in $X_A$ are in $X_R$.
- Assume there exists $E \in stb_C(G)$ and $E \notin stb_C(F)$. This means there exists an argument $x \in A(F) \cap C$ that is not in $E_F^{\oplus}$, but (1) is in $E_G^{\oplus}$ or (2) $x \notin A_G$. In the first case this means either $x \in E$ and hence $x \in E_F^{\oplus}$ or $x \in E_G^{+}, x \notin E_F^{+}$, but since $E_G^{+} \subseteq E_F^{+}$ this can not happen. For the second case the only possibility would be $x = c$, but $c \notin E$ since it is attacked by all other arguments in $C$.

(3) Since $A(F) \backslash C = A(G) \backslash C$ and we only remove core arguments, for any $E \in stb_C(F), E_F^{+} \backslash C$ can not change by removing arguments in $X_A$ since these are not defended against some argument within the core and could therefore not be in any $E \in \mathrm{stb}_C(F)$.

□

There are also patterns that instead of removing arguments only restructure attacks. An instance of such a pattern is depicted in Figure 4.4, where an odd-length path is broken up. The basis for its C-stable equivalence is given by:




(a) (b)

Figure 4.4: Argumentation frameworks $F'$ (a) and $G'$ (b), are also C-relativised equivalent w.r.t stable semantics.

**Proposition 6.** *The pattern* $(stb, F', G', \emptyset, \{x_1, ..., x_n\}, \{x_1, ..., x_n\})$ *with* $F' = (\{a_1, ..., a_k, b_1, ..., b_m\} \cup C, \{(a_1, x_1), ..., (a_k, x_1), (x_1, x_2), (x_2, x_3), ..., (x_{n-1}, x_n), (x_n, b_1), ..., (x_n, b_m)\})$, $G' = (A_{F'}, \{(x_1, x_2), ..., (x_{n-2}, x_{n-1}), (x_n, b_1), ..., (x_n, b_m)\} \cup \{(a_i, x_j) | 1 \leq i \leq k, 1 \leq j \leq n, j$ *is odd* $\})$ *and odd* $n \geq 3$ *holds.*

*Proof.* We prove all conditions of C-stable equivalency to show that the pattern in Proposition 6 holds. For this purpose let $F := F' \cup H, G := G' \cup H$ for arbitrary AFs $H$. Notice that by Proposition 1 there is no need to consider attacks $(x_i, y)$ for $y \notin C$.

(1) $A(F) = A(G)$ by definition of the pattern.

(2) We know that there are two types of C-stable extensions for this pattern. Firstly, any extension of form $E_1 = \{x_i | i \text{ is odd}\} \cup X$, where $X$ contains arguments in $H$ that are not attacked by odd $x$, is C-stable. Furthermore any $E_2 = Y \cup \{x_i | i \text{ is even}\} \cup Z$ where $Y \subseteq \{a_1, ..., a_k\}, |Y| \geq 1$ (resulting in $Y$ attacking all odd $x$ and defending all even ones) and $Z$ contains arguments in $H$ that are not attacked by any even $x_j$ or any combination of arguments $b_l$, are also C-stable as long as no $x_j$ attacks any of the arguments in $Y$.

Now we check the following cases to prove $stb_C(F) = stb_C(G)$:

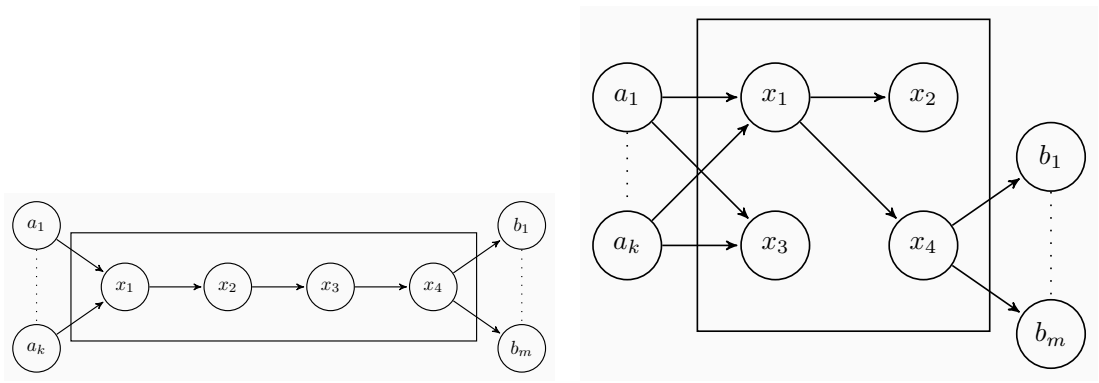
- Assume there exists an $E \in \text{stb}_C(F), E \notin cf(G)$. Since $E$ contains all odd or all even indexed $x$ and there are no attacks between any $x_i, x_j$, where $i$ and $j$ have the same parity, this can not be the case.
- Assume there exists $E \in \text{stb}_C(G), E \notin cf(F)$. This is not possible for the same reasons as the previous point.
- Assume there exists $E \in \text{stb}_C(F), E \notin \text{stb}_C(G)$. This means there is a core argument $x_i$ such that $x_i \in E_F^{\oplus}$, but $x_i \notin E_G^{\oplus}$.

If $E$ is of form $E_1$, all odd $x$ are in $E$, all even ones are attacked within $F$ and the same holds for $G$ (i.e. $A_F \cap C \subseteq E_F^{\oplus}$ and $A_G \cap C \subseteq E_G^{\oplus}$).
In case $E$ is of form $E_2$, we know it contains all even $x$, then in $F$ it attacks all odd $x_i$ except for $x_1$, which means it must contain at least one $a$ (that is not attacked by any of the other arguments in $E$, to be conflict-free), since all $a_j$ attack $x_1$. In $G$, the set of all even $x_l$ does neither attack $x_1$ nor $x_n$, but all other odd $x$. Since there are no differences in attacks on non-core arguments between the AFs and $E$ contains some arguments $a_j$, it also attacks all odd $x$ and is conflict-free. This means $A_F \cap C \subseteq E_F^{\oplus}$ and $A_G \cap C \subseteq E_G^{\oplus}$ again holds.

– Assume there exists $E \in \text{stb}_C(G), E \notin \text{stb}_C(F)$. This does not hold for analogous reasons as for the previous point.

(3) For all three C-restricted stable extensions the exact same arguments are attacked.

□




(a) (b)

Figure 4.5: Argumentation frameworks $F'$ (a) and $G'$ (b), are also C-relativised equivalent w.r.t stable semantics.

Similarly, a restructuring for even-length paths looks like Figure 4.5 and is defined as follows:

**Proposition 7.** *The pattern* $(stb, F', G', \emptyset, \{x_1, ..., x_n\}, \{x_1, ..., x_n\})$ *with* $F' = (\{a_1, ..., a_k, b_1, ..., b_m\} \cup C, \{(a_1, x_1), ..., (a_k, x_1), (x_1, x_2), (x_2, x_3), ..., (x_{n-1}, x_n), (x_n, b_1), ..., (x_n, b_m)\})$, $G' = (A_{F'}, \{(x_1, x_2), (x_1, x_4), ..., (x_1, x_n), (x_n, b_1), ..., (x_n, b_m)\} \cup \{(a_i, x_j) | 1 \leq i \leq k, 1 \leq j \leq n-1, j \text{ is odd }\})$ *and even* $n \geq 2$ *holds.*

*Proof.* We let $F := F' \cup H, G := G' \cup H$ and prove the following conditions. Again, notice that by Proposition 1 there is no need to consider attacks $(x_i, y)$ for $y \notin C$.

(1) $A(F) = A(G)$ by definition of the pattern.

(2) We distinguish two kinds of C-stable extensions for both AFs. The first kind is of form $E_1 = \{x_i | i \text{ is odd}\} \cup X$, where $X$ may contain arguments from $H$ that are not attacked by any odd $x$ as well as any number of arguments $b_j$. Furthermore, sets of form $E_2 = X \cup \{x_i | i \text{ is even}\} \cup Y$ also are C-restricted stable extensions in both AFs, with $X \subseteq \{a_1, ..., a_k\}, |X| \geq 1$ (since any $a$ attacks all odd $x$ and therefore defends all even $x$) and where $Y$ may contain arguments from $H$ that do not conflict with the even arguments or any arguments in $X$, but can not contain any $b_j$.

Now, the following cases are examined to prove $\text{stb}_C(F) = \text{stb}_C(G)$:

- Assume there exists an $E \in \text{stb}_C(F), E \notin cf(G)$. This means $E$ must contain two arguments, one attacking the other, in $G$ that is not in $R_F$. Since the only differences between the AFs are regarding attacks on core arguments and in $F$ there are only attacks of form $(x_i, x_{i+1})$, and $E$ only contains either even or odd core arguments, there needs to be an attack between two arguments of the same parity in $R_G$. Such an attack does not exist.
- Assume there exists $E \in \text{stb}_C(G), E \notin cf(F)$. This means $E$ must contain two arguments, one attacking the other, in $F$ that is not in $R_G$. As established in the previous point, any attack between two core arguments in $R_G$ must involve arguments of different index parity and $E$ only contains arguments of the same parity. Since this is still the case in $F$, the assumption can not be true.
- Assume there exists $E \in \text{stb}_C(F), E \notin \text{stb}_C(G)$. Thus there must exist a core argument $y \in E_F^{\oplus}, y \notin E_G^{\oplus}$. If $E$ is of form $E_1$, it contains all $x_i$, where $i$ is odd and therefore attacks all $x_{i+1}$. The same holds for $G$, which means $y \in E_F^{\oplus}, y \in E_G^{\oplus}$ for all core arguments $y$. If $E$ is of form $E_2$ it contains all $x_i$ where $i$ is even and at least one $a_j$, where $a_j$ attacks $x_1$, whereas $(x_i, x_{i+1}) \in R_F$ for each even $i$, hence $C \subseteq E_F^{\oplus}$. Additionally and $a_j$ attacks all $x_{i-1}$ in $G$ and therefore also $C \subseteq E_G^{\oplus}$.
- Assume there exists $E \in \text{stb}_C(G), E \notin \text{stb}_C(F)$. Analoguosly to the previous point, we know that all core arguments are contained in $E_F^{\oplus}$ as well as in $E_G^{\oplus}$.

(3) We look at all 3 types of extensions. As $F$ and $G$ coincide on $H$ it suffices to argue about the arguments in $F'$ and $G'$:

- $E_1^+ \cap A_{F'}$ contains no arguments that are not in the core in either AF.
- $E_2^+ \cap A_{F'} = E_1^+ \cap A_{G'}$ in both AFs.
- $E_3 \cap A_{F'}$ contains only one argument $x_n$ that attacks a non-core argument (i.e. it attacks $b$), but this argument is attacked in both frameworks.

□

Stable semantics have the unique property of being the only semantics of the ones discussed herein, in that it does not neccessarily produce any extension (i.e. it is possible that $\mathrm{stb}_C(F) = \emptyset$). Hence it is possible to reduce AFs where there is no C-stable extension to a much smaller amount of arguments. One such pattern is given by Proposition 8 and an instance of it is shown in Figure 4.6

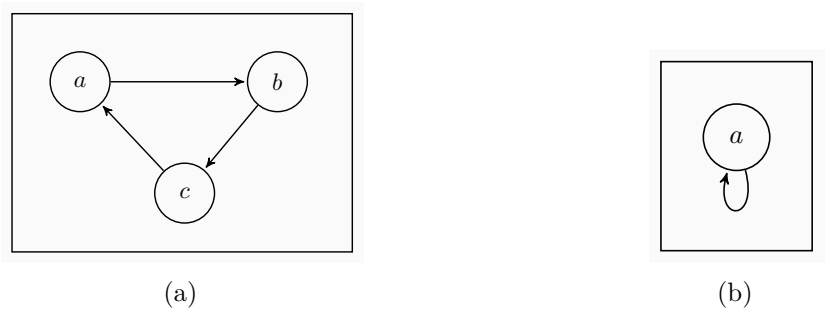



Figure 4.6: Argumentation frameworks $F'$ (a) and $G'$ (b), are also C-relativised equivalent w.r.t stable semantics.

**Proposition 8.** *The pattern* $(stb, F', G', \emptyset, \{a, b, c\}, \{a, b, c\})$ *with* $F' = (C, \{(a, b), (b, c), (c, a)\})$ *and* $G' = (\{a\}, \{(a, a)\}$ *holds.*

*Proof.* Since $stb_C(F') = \emptyset = stb_C(G')$, condition (1) need not be checked and conditions (2) and (3) are immediately satisfied. □

Note that as per Proposition 1, these arguments can freely attack any non-core arguments, but attacks onto any core argument may lead to the frameworks not being C-stable equivalent anymore.

## 4.3 Patterns for Admissible Semantics

This section examines C-admissible patterns and observations about them. First, we examine whether Proposition 1, which is applicable to C-stable equivalency also is suitable for C-admissible equivalency:

**Proposition 9.** *Let AFs* $F \equiv_C^{adm} G$*, where* $A(F)\backslash C = A(G)\backslash C$ *and* $R(F)\backslash C^2 = R(G)\backslash C^2$ *and let* $R'_F \subseteq \{(x, y) | x \in C \cap A(F), y \notin C\}, R'_G = \{(x, y) | (x, y) \in R'_F, x \in A(G)\}$ *with arbitrary fixed* $C$*. It does not hold that* $F' \equiv_C^{adm} G'$ *for* $F' = F \cup (\emptyset, R'_F), G' = G \cup (\emptyset, R'_G)$*.*

*Proof.* Recall the AFs in Figure 4.2. Now add the attack $(c, d_1)$ to both $F'$ and $G'$. This means that $\{d_1\} \in \mathrm{adm}_C(G')$, but $\{d_1\} \notin \mathrm{adm}_C(F')$ and therefore $\mathrm{adm}_C(G') \neq \mathrm{adm}_C(F')$ and $F' \not\equiv_C^{\mathrm{adm}} G'$, whereas $F' \equiv_C^{\mathrm{stb}} G'$ still holds. □

Similarly, it can be shown that adding an attack $(x_3, a_i)$ for all $i$ to the AFs depicted in Figure 4.5 results in differing C-admissible extensions.

This shows that the properties of C-stable equivalence can not be directly applied to other semantics. It follows an examination, whether a slightly modified Proposition 4 still holds for C-admissible equivalency regardless.

**Proposition 10.** *The pattern* $(adm, F', G', \{x_n\}, \{x_2, ..., x_n\}, \{x_1, ..., x_n\})$ *with* $F' = (\{x_1, ..., x_n\}, \{(x_1, x_2), ..., (x_{n-1}, x_n), (x_n, x_1)\})$*, where* $n$ *is odd, and* $G' = (A_{F'} \backslash \{x_n\}, (R_{F'} \cup \{(x_1, x_1)\}) \backslash \{x_n, x_1\})$ *holds.*

*Proof.* To prove Proposition 10, let $F := F' \cup H, G := G' \cup H$ where $\{a_1, ..., a_k\} \subseteq A_H$ and $\{(a_1, x_1), ..., (a_k, x_1)\} \subseteq R_H$. As with Proposition 2, this $H$ suffices to show the pattern holds for arbitrary $H$, since outgoing attacks from core arguments do not influence equivalence (see Proposition 1) and all attacks not prohibited by $P^-$ are already included in $H$.

(1) We know that $A_F \backslash C = A_G \backslash C$.

(2) It has to be evaluated whether $E^- \cap C \subseteq E^+$ for the exact same sets $E$ in $F$ and $G$. For this purpose let $S_i = Y \cup X_i$, where $X_i$ is a set of core arguments with even index, where if $x_j \in X$ then also $x_{j-2} \in X$ for $j > 2$ and $i$ is the maximum such index and $Y \subseteq \{a_1, ..., a_k\}$ and there are no attacks between arguments in $Y$. We know that all such $S$ are C-admissible extensions in both $F$ and $G$ (each argument $x_i$ is defended by $x_{i-2}$ or by any $a_j$ in case of defense against $x_1$). On the other hand, there is no set containing odd-indexed core arguments that is C-admissible, since even though each argument $x_i$ is again defended by $x_{i-2}$. However $x_1$ needs defence against $x_n$, but $n$ is odd itself (in $F$) or $x_1$ attacks itself (in $G$).

   - Now assume there exists an $E \in \text{adm}_C(F), E \notin \text{adm}_C(G)$. This means some argument $y \in E$ is defended against an attack by some core argument $c$ in $F$, but not in $G$. If $c = x_l$ and $l$ is odd, then $E \in \text{adm}_C(F)$ iff $S_{l-1} \subseteq E$ and then also $E \in \text{adm}_C(G)$. If $l$ is even, then $E$ is C-admissible neither in $F$ nor $G$.
   - If we assume $E$ is C-admissible in $G$, but not in $F$, the reasoning is the same as in the previous point.

(3) To prove the following conditions, we only need to look at $E \in \text{adm}_C$ for both $F$ and $G$.

   (a) Since no $E$ can contain $x_n$, $E_F^+ \backslash C = E_G^+ \backslash C$ holds.

   (b) $E_F^- \backslash E_F^+ = E_G^- \backslash E_G^+$, since there can only be a difference in attacks against or by core arguments, but we know $P^- = C$ and neither $x_1$ nor $x_n$ are in any $E$, which would be the only arguments with attack differences between $F$ and $G$.

□

The following Proposition (11) shows a property of C-admissible equivalency. It describes a way of removing undefended core arguments without violating C-admissible equivalence. It is a more restricted variant of Proposition 5 for C-admissible equivalency.

**Proposition 11.** *Let $F, G$ be two AFs where $F = (A(F), R(F))$. Now let $X_A = \{a \in C | \nexists d : (a, d) \in R(F), (d, a) \notin R(F) \wedge \exists b \in C : (b, a) \in R(F), (a, b) \notin R(F), \forall (c, b) : c \in C \wedge (c, a) \in R(F)\}$ and $X_R = \{(a, b) | a \in X_A \vee b \in X_A\}$. For $G = (A(F) \backslash X_A, R(F) \backslash X_R)$ we then have $G \equiv_C^{adm} F$.*

*Proof.* It follows a proof of Proposition 11:

(1) Since $A(G) = A(F) \backslash X_A$ and $X_A$ only contains core arguments condition (1) holds.

(2) It must be proven that $adm_C(F) = adm_C(G)$.

- Assume there exists $E \in adm_C(F), E \notin cf(G)$. This is impossible, since $E$ can not contain any argument from $X_A$ and $R(G) \subseteq R(F)$ and $E \in cf(F)$.
- Assume there exists $E \in adm_C(G), E \notin cf(F)$. This means there is an attack $(a, b) \in X_R$ and $a, b \in E$, but since for all $(a, b) \in X_R$ either $a \in X_A$ or $b \in X_A$ and $A(G) \cap X_A = \emptyset$, this can not be the case.
- Assume there exists $E \in adm_C(F)$ and $E \notin adm_C(G)$. This means $E_F^- \cap C \subseteq E_F^+$, but $E_G^- \cap C \nsubseteq E_G^+$. There are two ways for that to happen. Firstly, if there exists an argument $a \in E_G^- \cap C, a \notin E_F^-$. This is not possible since $E_G^- = E_F^- \backslash X_R$. Secondly, if there is an argument $a \in E_F^+ \cap C, a \notin E_G^+$, but $a \in E_G^-$. This can also not happen, since all attacks on and from arguments in $X_F$ are in $X_R$, i.e. any argument removed from $E^+$ must also have been removed from $E^-$.
- Assume there exists $E \in adm_C(G)$ and $E \notin adm_C(F)$. This means $E_F^- \cap C \nsubseteq E_F^+$, but $E_G^- \cap C \subseteq E_G^+$. This means there exists an argument $a \in C, a \in E_F^-, a \notin E_F^+$, whereas no such argument exists regarding $G$. This means that for all $e \in E_G^-$ also $e \in E_G^+$ and since $E_G^- \subseteq E_F^-, E_G^+ \subseteq E_F^+$ and $R(G) \subseteq R(F)$ it follows that for all such arguments $e \in E_F^- \cap C \Rightarrow e \in E_F^+ \cap C$. Therefore $a \neq e$ for any such $e$ and $a \in X_A$. Thus there can not be any argument $d$ that $a$ attacks, but that does not itself attack $a$, i.e. if $a \in E_F^-$ then also $a \in E_F^+$, which is a contradiction.

(3) (a) Since $A(F) \backslash C = A(G) \backslash C$ and we only remove core arguments, for any $E \in adm_C(F), E_F^+ \backslash C$ can not change by removing arguments in $X_A$ since these are not defended against some argument within the core and could therefore not be in any $E \in adm_C(F)$.

(b) We also prove that $E_F^- \backslash E_F^+ = E_G^- \backslash E_G^+$ holds. Since for every argument $a \in E_F^-, a \in X_A$ we know that there is at least one attack $(b, a)$, where $b \in E$ which $a$ is not defended from, i.e. $(E_F^- \cap X_A) \subseteq (E_F^+ \cap X_A)$. This means $(E_F^- \backslash E_F^+) \backslash X_A = E_F^- \backslash E_F^+$ and since $E_G^- = E_F^- \backslash X_A$ and $E_G^+ = E_F^+ \backslash X_A$ condition (3b) must hold.

□

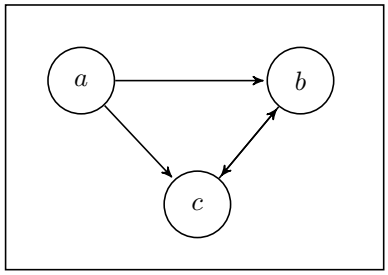



Figure 4.7: Removing arguments $b$ and $c$ and all related attacks, produces a C-relativised admissible equivalent AF, since neither argument is defended against $a$.

An example of Proposition 11 is shown in Figure 4.7.

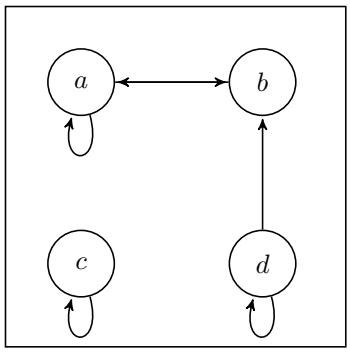



Figure 4.8: $a$ can be removed since it attacks itself and is defended against, while $d$ can not be removed since it attacks $b$. $c$ can be removed since it doesn't attack another core argument.

Furthermore, we can also remove arguments that attack themselves and against which the other arguments are properly defended (see Proposition 12). An instance of this is shown in Figure 4.8.

**Proposition 12.** *Let $F, G$ be two AFs where $F = (A(F), R(F))$. Now let $X_A \subseteq A(F) \cap C$ be the set of arguments that attack themselves and either not attack another argument or for each argument $b$, where $(a, b) \in R(F)$, also $(b, a) \in R(F)$. Additionally $X_R = \{(a, b) | a \in X_A \vee b \in X_A\}$ contains all attacks onto or by these arguments. For $G = (A(F) \backslash X_A, R(F) \backslash X_R)$ we then have that $G \equiv_C^{adm} F$.*

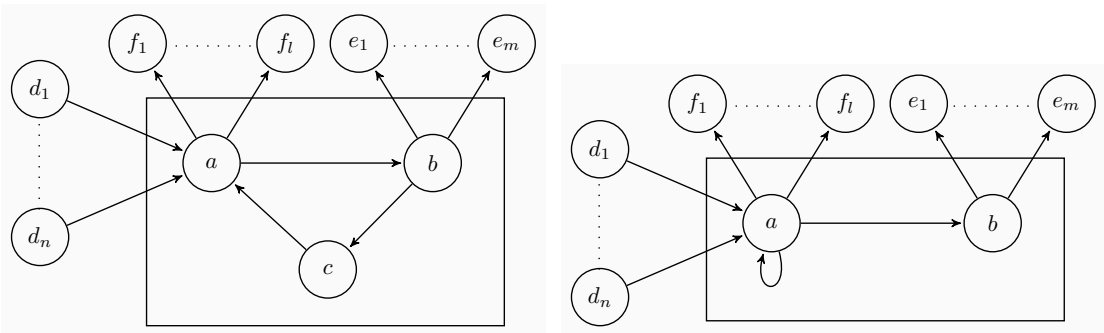
*Proof.* (1) Since $X_A \subseteq C$, $A(F)\backslash C = A(G)\backslash C$.

(2) Since for all $a \in X_A$, $\{a\} \cup S \notin cf(F)$ for any set $S \subseteq A(F)$ we know that $cf(F) = cf(G)$ and since all these $a$ do not attack any argument that does not defend itself against $a$, it follows that $adm_C(F) = adm_C(G)$.

(3) (a) $E_F^+\backslash C = E_G^+\backslash C$ holds because the only difference in attacks for any $E$ between $F$ and $G$ is only regarding arguments within the core.

(b) Each attack $(a, b)$ where $a \in E_F^-, a \notin E_G^-$ is an attack where $a \in X_A$. For each of these attacks there also exists an attack $(b, a)$ and therefore also $b \in E_F^+$. Thus we know for all $a \in E_F^-, a \notin E_G^-$ it also holds that $a \notin E_F^-\backslash E_F^+$ and therefore $E_F^-\backslash E_F^+ = E_G^-\backslash E_G^+$.

□




(a) (b)

Figure 4.9: Argumentation frameworks $F$ (a) and $G$ (b), are C-relativised equivalent w.r.t admissible semantics.

The following pattern is a variation of the pattern introduced in Proposition 2, only in this case regarding admissible semantics and with additional fixed attacks.

**Proposition 13.** *The pattern* $(adm, F', G', \{b, c\}, \{c\}, \{a, b, c\})$ *with* $A(F') = \{a, b, c\}$, $R(F') = \{(a, b), (b, c), (c, a)\}$ *and* $G' = (A(F')\backslash\{c\}, (R(F') \cup \{(a, a)\})\backslash\{(c, a), (b, c)\})$ *holds.*

We now give a proof for the C-relativised admissible pattern described in Proposition 13 and show an instance of it in Figure 4.9:

*Proof.* First, let $F := F' \cup H, G := G' \cup H$, where $H = (\{d_1, ..., d_n, e_1, ..., e_m, f_1, f_l\}, \{(d_1, a), ..., (d_n, a), (a, f_1), ..., (a, f_l), (b, e_1), ..., (b, e_m)\})$. It suffices to show the Proposition for $H$ since any other arguments that an arbitrary $H$ could contain do not interact with core arguments. Furthermore, let $\mathbf{D} \subseteq \{d_1, ..., d_n\}, \mathbf{E} \subseteq \{e_1, ..., e_m\}$ and $\mathbf{F} \subseteq \{f_1, ..., f_l\}$. Next, the conditions for C-relativised admissible equivalency are evaluated:

(1) It is obvious that $A(F)\backslash C = A(G)\backslash C$ holds.

(2) The C-restricted admissible extension for the AFs in question include $\emptyset$, all sets $X \subseteq \mathbf{D}$, as well as $\{\{b\} \cup X \neq \emptyset | X \subseteq \mathbf{D}\}, \{X \cup Z \neq \emptyset | X \subseteq \mathbf{D}, Z \subseteq \mathbf{F}\}$ and $\{\{b\} \cup X \cup Z \neq \emptyset | X \subseteq \mathbf{D}, Z \subseteq \mathbf{F}\}$, since $\mathbf{F}$ and $b$ are defended by $\mathbf{D}$ against $a$, while $\mathbf{E}$ is not defended by any set against $b$.

(3) (a) Since only $b$ and any $d_i$ or $f_k$ occur in any $E \in \mathrm{adm}_C(F)$ where no $d_i$ or $f_k$ attack any arguments not in $C$ and $b$ attacks only $\mathbf{E}$, of which all its elements are both in $F$ and $G$, condition (3a) must hold.

(b) Lastly, assume $E_F^- \backslash E_F^+ \neq E_G^- \backslash E_G^+$ holds. There are only three differences between $R(F)$ and $R(G)$: $(a, a)$, but since $a$ does not occur in any $E$ this does not make a difference, $(c, a)$ is not relevant since neither $c$ nor $a$ is in any $E$ and lastly $(b, c)$, which does not have an effect, since no $E$ needs to be defended against $c$.

□

Propositions 6 and 7 can also be re-examined w.r.t. C-admissible semantics.

**Proposition 14.** *$(adm, F', G', \emptyset, \{x_1, ..., x_n\}, \{x_1, ..., x_n\})$ with $F' = (\{a_1, ..., a_k, b_1, ..., b_m\} \cup C, \{(a_1, x_1), ..., (a_k, x_1), (x_1, x_2), (x_2, x_3), ..., (x_{n-1}, x_n), (x_n, b_1), ..., (x_n, b_m)\})$, $G' = (A_{F'}, \{(x_1, x_2), ..., (x_{n-2}, x_{n-1}), (x_n, b_1), ..., (x_n, b_m)\} \cup \{(a_i, x_j) | 1 \leq i \leq k, 1 \leq j \leq n, j$ is odd $\})$ and odd $n \geq 3$ does not hold and is not a C-admissible pattern.*

*Proof.* It can easily be seen that for a path of length three the exension $E = \{x_3\} \notin \mathrm{adm}_C(F)$ since $E^- \cap C = \{x_2\}$ and $x_3$ does not attack $x_2$, whereas $E \in \mathrm{adm}_C(G)$ because only $a_i$ attack $x_3$, which do not need to be defended since $a_i \notin C$. Thus $\mathrm{adm}_C(F) \neq \mathrm{adm}_C(G)$ and $F \not\equiv_C^{\mathrm{adm}} G$. □

This pattern can not be applied to C-admissible equivalency, since it is not required to consider all core arguments for each extension as it is the case for C-stable equivalence checks. For the same reason the we also can not restructure even paths described in 7 w.r.t. C-admissible semantics:

**Proposition 15.** *$(adm, F', G', \emptyset, \{x_1, ..., x_n\}, \{x_1, ..., x_n\})$ with $F' = (\{a_1, ..., a_k, b_1, ..., b_m\} \cup C, \{(a_1, x_1), ..., (a_k, x_1), (x_1, x_2), (x_2, x_3), ..., (x_{n-1}, x_n), (x_n, b_1), ..., (x_n, b_m)\})$, $G' = (A_{F'}, \{(x_1, x_2), (x_1, x_4), ..., (x_1, x_n), (x_n, b_1), ..., (x_n, b_m)\} \cup \{(a_i, x_j) | 1 \leq i \leq k, 1 \leq j \leq n-1, j$ is odd $\})$ and even $n \geq 2$ does not hold and is not a C-admissible pattern.*

*Proof.* Consider the set $E = \{x_1, b_1\}$. Then $E_F^- \cap C = \{x_4\}$ and neither $x_1$ nor $b_1$ attacks $x_4$ in $F$. On the other hand $E_G^- \cap C = \{x_4\}$, but $(x_1, x_4) \in R_G$ and hence $E \in \mathrm{adm}_C(G)$ and $F \not\equiv_C^{\mathrm{adm}} G$. □

## 4.4 Patterns for Preferred Semantics

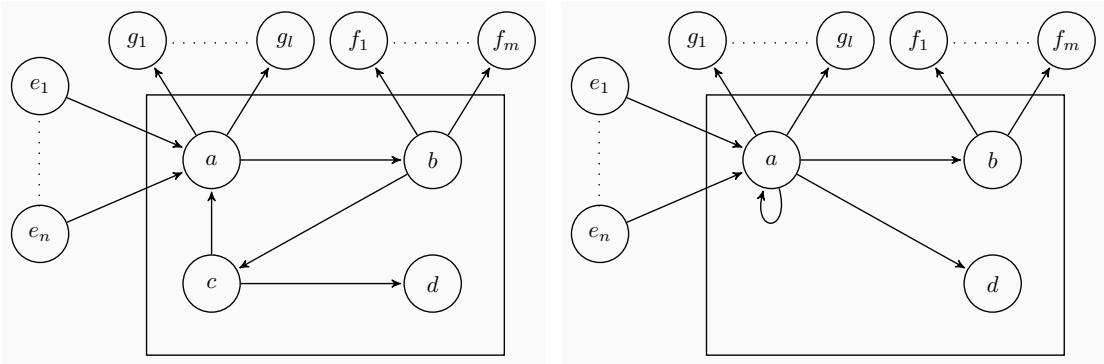



Figure 4.10: Argumentation frameworks $F'$ (a) and $G'$ (b), are C-relativised equivalent w.r.t preferred semantics.

In this section, C-preferred patterns are examined. For that purpose recall that $\mathrm{prf}_C(F) \subseteq \mathrm{adm}_C(F)$ for any AF $F$, since as per Definition 16 only C-admissible extensions are considered for computation in the first place. This indicates that if $F$ and $G$ have the same C-admissible extensions, they also have the same C-preferred extensions. A similar notion is expressed in Proposition 16, which is proven in [OW11, Proposition 1].

**Proposition 16** ([OW11])**.** *For any AFs $F, G$, we have $adm(F) = adm(G) \Rightarrow prf(F) = prf(G)$.*

Using this fact we want to show that any pattern or property that holds for C-admissible semantics also does hold for C-preferred semantics. For that reason we show Theorem 6:

**Theorem 6.** *For any AFs $F$ and $G$ with any core $C$, if $F \equiv_C^{adm} G$, then also $F \equiv_C^{prf} G$.*

*Proof.* We know from Definition 13 that we need to prove for any $H$ where $H \cap C = \emptyset$, that $F \cup H \equiv^{\mathrm{adm}} G \cup H \Rightarrow F \cup H \equiv^{\mathrm{prf}} G \cup H$ holds, which by Definition 10 is the same as proving $\mathrm{adm}(F \cup H) = \mathrm{adm}(G \cup H) \Rightarrow \mathrm{prf}(F \cup H) = \mathrm{prf}(G \cup H)$.

Assume now that we know for some AFs $F, G$ with $C$ and arbitrary $H$, where $H \cap C = \emptyset$, that $F \cup H \equiv_C^{\mathrm{adm}} G \cup H$ holds. Thus also $\mathrm{adm}(F \cup H) = \mathrm{adm}(G \cup H)$ and by Proposition 16 it follows that $\mathrm{prf}(F \cup H) = \mathrm{prf}(G \cup H)$.

This means Theorem 6 holds for any such $H$. □

Hence it follows that each pattern for C-admissible semantics can also be applied for C-preferred semantics:

**Corollary 1.** *Given a pattern $P_S = (adm, F', G', P^+, P^-, C)$, if $P_S$ holds, then the pattern $P_P = (prf, F', G', P^+, P^-, C)$ also holds.*

Apart from C-admissible patterns, an example for a C-preferred pattern is another version of a pattern where a 3-cycle is reduced, which is defined in Proposition 17 and an example framework is shown in Figure 4.10.

**Proposition 17.** *The pattern* $(adm, F', G', \{c,d\}, \{b,c,d\}, \{a,b,c,d\})$ *with* $A(F') = \{a, b, c, d, e_1\}$, $R(F') = \{(a,b),(b,c),(c,a),(c,d),(e_1,a)\}$ *and* $G' = (A(F')\backslash\{c\}, (R(F') \cup \{(a,a),(a,d)\})\backslash\{(b,c),(c,a),(c,d)\})$ *holds.*

*Proof.* First, let $F := F' \cup H, G := G' \cup H$, where $H = (\{e_2, ...e_n, f_1, ..., f_n\}, \{(e_2,a), ..., (e_n,a), (b,f_1), ..., (b,f_m), (a,g_1), ..., (a,g_l)\}$. Now three conditions are proven for Proposition 17 to hold:

(1) We know that $A(F)\backslash C = A(G)\backslash C$ holds since the only differences occur in the core.

(2) Condition (2) holds since $\text{prf}_C(F) = \text{prf}_C(G) = \{\{b,d\} \cup S | S \subseteq \{e_1, e_2, ..., e_n\}, S \neq \emptyset\} \cup \{\emptyset\}$.

(3) For all $E \in \text{prf}_C(F)$ we prove:

   (a) $E_F^+\backslash C = E_G^+\backslash C$ holds since the only attacks on non-core arguments concerning all $E$ are the attacks of $b$ which are the same in both AFs.

   (b) To prove $E_F^-\backslash E_F^+ = E_G^-\backslash E_G^+$, changes in attacks on $E$ need to be examined. $E^-$ may be influenced only by $(c,d)$ and $(a,d)$ since $d$ occurs in all $E$, but $c \notin E_F^-\backslash E_F^+$ since $b$ attacks $c$ and also $a \notin E_G^-\backslash E_G^+$ since any $e_i$ attacks $a$.

□

## 4.5 Patterns for Complete Semantics

For the analysis of C-complete patterns recall the characteristic functions introduced both in subsection 3.1.6 and Definition 3 which are needed to prove such patterns and describe sets of defended arguments.

First though, we show that as for C-admissible patterns, C-complete patterns are at the same time applicible to C-preferred semantics. Again, we use a fact shown within [OW11, Proposition 1]:

**Proposition 18** ([OW11])**.** *For any AFs* $F, G$*, we have* $com(F) = com(G) \Rightarrow prf(F) = prf(G)$*.*

Using this fact we can show that any pattern or property that holds for C-complete semantics also does hold for C-preferred semantics. For that reason the following Theorem is shown:

**Theorem 7.** *For any AFs $F$ and $G$ with any core $C$, if $F \equiv_C^{adm} G$, then also $F \equiv_C^{prf} G$.*

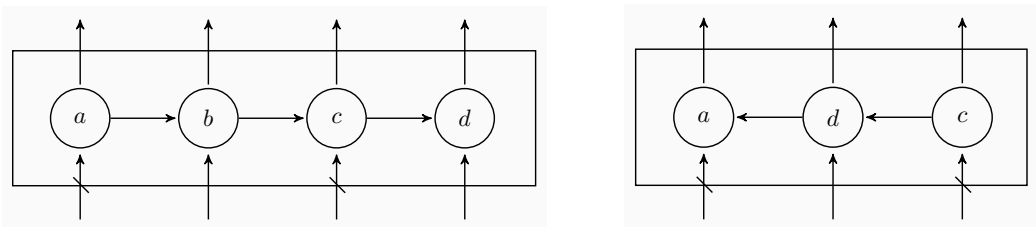
*Proof.* We know from Definition 13 that we need to prove for any $H$ where $H \cap C = \emptyset$, that $F \cup H \equiv^{\text{com}} G \cup H \Rightarrow F \cup H \equiv^{\text{prf}} G \cup H$ holds, which by Definition 10 is the same as proving $\text{com}(F \cup H) = \text{com}(G \cup H) \Rightarrow \text{prf}(F \cup H) = \text{prf}(G \cup H)$.

Assume now that we know for some AFs $F, G$ with $C$ and arbitrary $H$, where $H \cap C = \emptyset$, that $F \cup H \equiv_C^{\text{com}} G \cup H$ holds. Thus also $\text{com}(F \cup H) = \text{com}(G \cup H)$ and by Proposition 18 it follows that $\text{prf}(F \cup H) = \text{prf}(G \cup H)$.

This means Theorem 7 holds for any such $H$. □

Hence it follows that C-complete patterns are also C-preferred patterns:

**Corollary 2.** *Given a pattern $P_C = (com, F', G', P^+, P^-, C)$, if $P_C$ holds, then the pattern $P_P = (prf, F', G', P^+, P^-, C)$ also holds.*




(a) (b)

Figure 4.11: Argumentation frameworks $F$ (a) and $G$ (b), are C-relativised equivalent w.r.t complete semantics.

An example of a C-relativised complete pattern is described in Proposition 19 and shown in Figure 4.11.

**Proposition 19.** *The pattern $P = (com, (F' = (\{a, b, c, d\}, R_{F'}), (G' = (\{a, c, d\}, R_{G'}), \emptyset, \{a, c\}, \{a, b, c, d\})$, where $R_{F'} = \{(a, b), (b, c), (c, d)\}$ and $R_{G'} = \{(c, d), (d, a)\}$, holds.*

*Proof.* To prove Proposition 19 we must check all conditions for C-relativised complete equivalence. For this purpose assume $F := F' \cup H, G := G' \cup H$ for arbitrary AFs $H$ which fulfil all the criteria of Definition 19.

Also notice that for all $E \in \text{com}_C(F), E \in \text{com}_C(G)$ it must hold that $\{a, c\} \in E$, since both $a$ and $c$ are only defended from all attacking core arugments in both $F$ and $G$ if they both are part of the extension.

(1) $A_F \backslash C = A_G \backslash C$ as $F$ and $G$ only differ on core arguments.

(2) Assume $\text{com}_C(F) \neq \text{com}_C(G)$. This means there exists a set $E$ for which one of the following cases holds:

- $E \in \text{com}_C(F), E \notin \text{cf}(G)$. Since $E \in cf(F)$ and $R_G \subseteq R_F \cup \{(d,a)\}$ and there is no $E \in com_C(F)$ with $\{d,a\} \subseteq E$, it follows that $E \in cf(G)$.
- $E \in \text{com}_C(G), E \notin \text{cf}(F)$. Since $E \in cf(G)$ and $R_G \subseteq R_F \cup \{(d,a)\}$ it follows that there must be an attack $(x,y) \in R_F \backslash R_G$, where $x, y \in E$. Such an attack would need to involve $b$, but $b \notin E$, so this can not be the case.
- There exists an argument $x \in E, x \in \mathcal{F}_{F,C,E}(E), x \notin \mathcal{F}_{G,C,E}(E)$. Therefore $x \notin C$, since we know $C \subseteq E_F^{\oplus}$. This means there must be an attack $(y,x)$ s.t. $y \in C, y \in E_F^+, y \notin E_G^+$, but since $C \backslash \{b\} \subseteq E_G^{\oplus}$ this is impossible.
- There exists an argument $x \in E, x \notin \mathcal{F}_{F,C,E}(E), x \in \mathcal{F}_{G,C,E}(E)$. Therefore $x \notin C$, since we know $C \backslash \{b\} \subseteq E_G^{\oplus}$. This means there must be an attack $(y,x)$ s.t. $y \in C, y \notin E_F^+, y \in E_G^+$, but since $C \subseteq E_F^{\oplus}$ this is impossible.
- There exists an argument $x \notin E, x \in \mathcal{F}_{F,C,E}(E), x \notin \mathcal{F}_{G,C,E}(E)$. We know that $a, c \in E, b, d \in E_F^+, d \in E_G^+$. This means $a, c \in \mathcal{F}_{F,C,E}(E), a, c \in \mathcal{F}_{G,C,E}(E), b, d \notin \mathcal{F}_{F,C,E}(E), b, d \notin \mathcal{F}_{G,C,E}(E)$. Thus $x \notin C$. For $x \in \mathcal{F}_{F,C,E}(E)$ it is then necessary that $x \in E$.
- There exists an argument $x \notin E, x \notin \mathcal{F}_{F,C,E}(E), x \in \mathcal{F}_{G,C,E}(E)$. This is analogous to the previous point.

(3) We check for all $E \in com_C(F)$:

(a) Since $E \in com_C(F), com_C(F) = com_C(G)$, $b$ is not part of any C-complete extension and all other arguments do not have a difference in attacks on arguments not in $C$ between $F$ and $G$, we have $E_F^+ \backslash C = E_G^+ \backslash C$.

(b) Towards a contradiction assume $E_F^- \backslash E_F^+ \neq E_G^- \backslash E_G^+$. This means there exists an argument $x \in E_F^- \backslash E_F^+$ and $x \notin E_G^- \backslash E_G^+$. Since there is no difference in attacks between $F$ and $G$ regarding attacks on non-core arguments we know $E_F^- \backslash C = E_G^- \backslash C$ and $E_F^+ \backslash C = E_G^+ \backslash C$. Thus, $x \in C$ and as $\{a, c\} \subset E$ and $d \in E_F^+, d \in E_F^-$, it follows $x = b$, but $b \in E_F^+$.

Assuming $x \notin E_F^- \backslash E_F^+$ and $x \in E_G^- \backslash E_G^+$ has analogous results.

(c) We know that each extension $E$ contains $a, c$ and any non-core arguments that are defended against $b$ (in $F$) and $d$ (in either framework). This means there are a limited number of sets $S$ s.t. $E_F^- \backslash E_F^+ \subseteq S \subseteq A(F) \backslash C \cup E$. There are two ways for condition (3c) not to hold:

Firstly, assume $\mathcal{F}_{F \backslash S}(E) = E \cap C$, but $\mathcal{F}_{F \backslash S}(E) \neq \mathcal{F}_{G \backslash S}(E)$. This means exactly the arguments $x \in A_F \cap C$ that are defended by $E$ from arguments in $F \backslash S$ are in $E \cap C$. Since $b \notin E, d \notin E$, only $a$ and $c$ must be defended in such a way, which is the case. Furthermore one of the following holds:

- There is some argument $u \in \mathcal{F}_{F \backslash S}(E), u \notin \mathcal{F}_{G \backslash S}(E)$. This means $u$ is defended by $E$ against all arguments in $F \backslash S$ but not in $G \backslash S$. There are

only differences in attacks on or by $b$ and the attack $(d, a)$ between $F$ and $G$. Since any $E \in com_C(F)$ attacks $b$ and $d$ and any $E \in com_C(F)$ attacks $b$, all arguments are defended against $b$ and $d$ in $F$, and against $d$ in $G$ and therefore for any $u \in \mathcal{F}_{F\backslash S}(E)$ it holds that $u \in \mathcal{F}_{G\backslash S}(E)$.

– There is some argument $u \notin \mathcal{F}_{F\backslash S}(E), u \in \mathcal{F}_{G\backslash S}(E)$. This means there is an argument $u$ that is defended by $E$ against all arguments in $G\backslash S$, but there is an undefended attack on $u$ in $F\backslash S$. Such an attack can only exist from $b$, but since $E$ attacks $b$ – all other attacks being both in $R_F$ as well as $R_G$ – we know $u \in \mathcal{F}_{F\backslash S}(E)$.

Secondly, assume $\mathcal{F}_{G\backslash S}(E) = E \cap C$, but $\mathcal{F}_{F\backslash S}(E) \neq \mathcal{F}_{G\backslash S}(E)$. This means exactly the arguments $x \in A_G \cap C$ that are defended by $E$ from arguments in $G\backslash S$ are in $E \cap C$. Since $d \notin E$, only $a$ and $c$ must be defended. The inequality is the same as in the first case and hence has already been disproven.

□

Furthermore, we are again able to show that cycles of odd length, where only one is attacked by a non-core argument, can be reduced by one argument. This is the same property as shown for C-admissible semantics in Proposition 10.

**Proposition 20.** *The pattern* $(com, F', G', \{x_n\}, \{x_2, ..., x_n\}, \{x_1, ..., x_n\})$ *with* $F' = (\{a, x_1, ..., x_n\}, \{(a, x_1), (x_1, x_2), ..., (x_{n-1}, x_n), (x_n, x_1)\})$*, where* $n$ *is odd, and* $G' = (A'_F\backslash\{x_n\}, (R'_F \cup \{(x_1, x_1)\})\backslash\{x_n, x_1\})$ *holds.*

*Proof.* To prove Proposition 20, let $F := F' \cup H, G := G' \cup H$ for arbitrary $H$ that satisfy the pattern conditions.

(1) Per definition of the pattern $A_F\backslash C = A_G\backslash C$.

(2) To show that $com_C(F) = com_C(G)$ we assume first the contrary, that there exists an extension $E$ that is only C-complete in one of the AFs. But we know that all attacks in both AFs are the same, except that $(x_{n-1}, x_n), (x_n, x_1) \notin R_G, (x_1, x_1) \notin R_F$.

Additionally $x_2$ is defended by $a$ from $x_1$ and each even-indexed $x_i$ is defended by $x_{i-2}$ from $x_{i-1}$, thus $\mathcal{F}_{F,C,E} = E \cup \{x_i\}$ if $i-2$ is the highest index in $E$, as long as $a \in E$, except if $E$ contains all even $x_i$ and $E = \mathcal{F}_{F,C,E} = \mathcal{F}_{G,C,E}$. If $a \notin E$, then $E \cap C = \emptyset$ in both $F$ and $G$. So either $E \cap C = \emptyset$ or $E$ defends and contains all even $x_i$ in both $F$ and $G$ and therefore $com_C(F) = com_C(G)$.

(3) We prove three conditions concerning arguments $E$ attacks or is attacked by:

(a) Since neither $x_1$ nor $x_n$ are in any $E$, we know $E_F^+\backslash C = E_G^+\backslash C$.

(b) As there are no differences in attacks except for $x_1$ and $x_n$, which are not in any $E$, also $E_F^-\backslash E_F^+ = E_G^-\backslash E_G^+$.

(c) If follows an examination of any $E$, where $\mathcal{F}_{F\backslash S}(E) \neq \mathcal{F}_{G\backslash S}(E)$ for $E_F^- \backslash E_F^+ \subseteq S \subseteq A_F \backslash C \cup E$:

That means that there is an argument $u$ that is defended by $E$ in only one of the AFs. Since $S$ contains no core arguments $u \notin C$. Then, since $A_F \backslash C = A_G \backslash C$, $u$ must be attacked by some core argument. $E$ must contain all even $x_i$, which means all arguments are defended against odd $x_i$ in $F$ and $G$ both; or if $E \cap C = \emptyset$, then $E$ defends against no $x_i$ at all, both in $F$ and $G$. This means $\mathcal{F}_{F\backslash S}(E) = \mathcal{F}_{G\backslash S}(E)$ must always hold.

□

## 4.6 Summary

In this chapter we gave the definition for C-relativised equivalency patterns, which are 6-tuples comprising two C-relativised equivalent AFs w.r.t. to some semantics and some core set of arguments as long as any enlargement of these AFs does not contain certain prohibited attacks. Along with this we introduced a way of visualizing these prohibited attacks in graphical representations of patterns.

Furthermore we examined concrete patterns and pattern properties for stable, admissible, preferred and complete semantics, which were obtained by analysing the results of computations using the encodings presented in chapter 3.

We found that different semantics indeed posses different properties (some holding for multiple semantics, while others need modification to work) and following from that, also patterns changing the structure of an AF without changing its behaviour, in some cases even with a different amount of core arguments.

In the following chapters we will examine which effect these patterns have on concrete instances of larger AFs and how they may be useful in pre-processing these frameworks.

CHAPTER 5

# C-relativised Equivalency Patterns for Pre-Processing

This chapter considers the question of how pre-processing by replacing occurences of pattern graphs benefits the search for extensions. First we describe how to find instances of a given pattern in some graph and how to replace them. Following that, we conduct experiments as to how often certain patterns occur in example graphs and what computational impact their replacement has.

## 5.1 Problem Statement

The goal of this chapter is the analysis of practical useability of pattern pre-processing. To do this, firstly we need three graphs representing AFs: (1) an encoding of the *database graph* $G$ and (2) an encoding of the *query graph* $Q$, and (3) an encoding of a *replacement graph* $R$. Given these graphs we want to find all the occurrences of $Q$ within $G$ and replace them with $R$, where $Q$ and $R$ are provided by some C-relativised equivalency pattern.

Input data for this problem is provided in the ASPARTIX format [EGW08], which is used for Argumentation framework algorithms. Example 17 shows an example of such an encoding (i.e. an encoding of the pattern graph in Figure 4.2 with $n = 1$), where $arg(x)$ means $x \in A'_F$ and $att(x, y)$ specifies that $(x, y) \in R'_F$. It can also be seen $sem(\sigma)$ specifes the semantics that are used, $core(x)$ means that $x \in C$ and $pin(x)$ means $x \in P^-$. Additionally it is possible to express $x \in P^+$ as $pout(x)$. The *sem*, *core*, *pin* and *pout* information needs to be provided for the query and replacement graphs to ensure compatibility with the definition of C-relativised equivalency patterns, not for the database graph.

**Example 17.**

$$\begin{aligned}\pi(AF) = \; & arg(a).\\ & arg(b).\\ & arg(c).\\ & arg(d).\\ & att(a,b).\\ & att(b,c).\\ & att(c,a).\\ & att(d,a).\\ & sem(stb).\\ & core(a).\\ & core(b).\\ & core(c).\\ & pin(b).\\ & pin(c).\end{aligned}$$

These graphs together with the procedures described in section 5.2 ensure that the following two tasks are achieved:

**Task 1.** *Given a database graph $G$ and a query graph $Q$, return a list $M = \{m_1, ..., m_k\}$ of subgraphs of $G$ that each are isomorphic to $Q$.*

**Task 2.** *Given $G, Q$ and $M$ as well as a pattern $P = (\sigma, Q, R, P^+, P^-, C)$, where $R$ is a replacement graph, replace all $m_i$ with $R$ for if $G \equiv^{\sigma}_{C} G \backslash m_i \cup R$.*

## 5.2 Replacing Pattern Occurrences

To measure the effects of replacements it is first necessary to find occurrences of some pattern graph $Q := F'$ (i.e. the query graph) within some larger graph $G$ (i.e. $G = Q \cup H$, the database graph). If occurrences are found, it is possible to evaluate whether they adhere to the other conditions of a pattern and, in the affirmative case, replace $Q$ with a replacement graph $R := G'$.

### 5.2.1 Subgraph isomorphism

Determining whether an instance of $Q$ exists in $G$ is called the *subgraph isomorphism problem*, which is an NP-complete problem [ABH+16]. We consider a specialised instance of that problem: *subgraph matching*. This means instead of determining whether $Q$ exists in $G$, we want to find all instances of $Q$ within $G$ and enumerate them.

For that purpose an algorithm called *BB-Graph* is employed, which is a state-of-the-art subgraph matching algorithm introduced in 2017 [AY17] that uses a branch-and-bound approach on labelled graphs to find isomorphic subgraphs. To be applicable to argumentation frameworks it is modified so that arguments do not need to be labelled.

Given $Q$ and $G$, the algorithm then outputs a list of subgraphs of $G$ which could be matched with $Q$. Each of these subgraphs is itself a list of matched vertices and edges. We write $x \sim y$ for two vertices $x, y$, where $x \in V_Q$ and $y \in V_G$ to express they are matched. Analogously $x \sim y$ signifies matched edges if $x$ and $y$ are edges in $Q$ and $G$.

A JAVA 7 implementation of the modified algorithm is provided at [Bel18b]. This implementation randomises the order in which the vertices are processed, so that it is possible to get different results when only computing a sample of isomorphic subgraphs, rather than all occurrences.

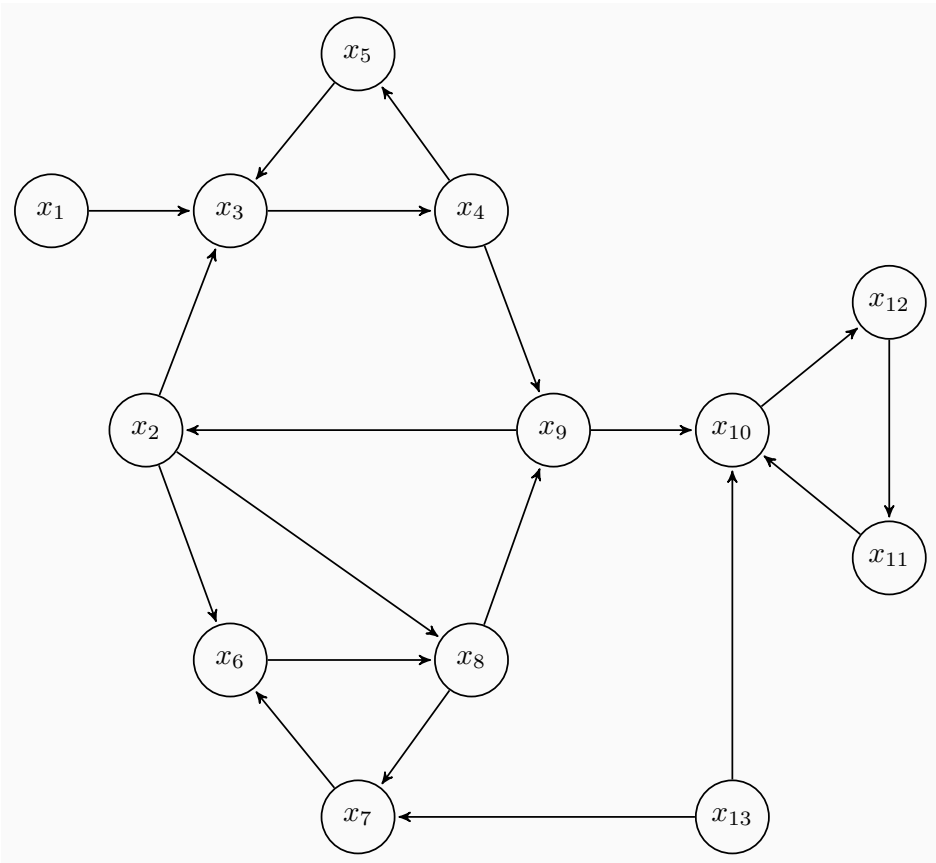



Figure 5.1: A database graph where the 3-length cycle pattern (Proposition 2) can be applied.

**Example 18.** *Given the AF depicted in Figure 5.1, we let the BB-Graph algorithm look for occurences of the graph $F'$ described in Proposition 2.*

*The AF contains four different cycles of length three, which occurs in $F'$. These cycles consist of the arguments in the following sets: $X_1 = \{x_3, x_4, x_5\}, X_2 = \{x_2, x_8, x_9\}, X_3 = \{x_6, x_7, x_8\}$ and $X_4 = \{x_{10}, x_{11}, x_{12}\}$.*

*For $X_1$ we find two possible matches for the argument d in the pattern ($x_1$ and $x_2$), which both attack $x_3$. $X_2$ also possesses two potential matches in $x_4$, which attacks $x_9$ and in $x_6$, that attacks $x_8$. For $X_3$ there are three options of matching to d: (1) $x_2$ attacking $x_6$, (2) $x_2$ instead attacking $x_8$ and lastly (3) $x_{13}$ with its attack on $x_7$. Furthermore, $X_4$ has $x_9$ and $x_{13}$ as matches, since they both attack $x_{10}$, resulting in an overall count of 9 isomorphic subgraphs, that partly intersect each other.*

Since some subgraphs isomorphic to the query graph do not have pairwise disjoint argument sets and the prohibited attacks fixed in $P^+$ and $P^-$ are not checked by BB-Graph, replacements can not be conducted by merely finding all matches.

### 5.2.2 Determining adherence to pattern conditions

Given the computation results of BB-Graph, to replace $Q$ with $R$ it is necessary to provide a file containing an encoding of $R$. We now describe in an abstract form a generalised version of an algorithm which replaces valid matches of the query graph with the replacement graph.

```
Data: List of matched isomorphic subgraphs: M, database graph: G, query
      graph: Q, replacement graph: R
Result: modified database graph: G'
G' := G;
if R fits Q then
    for m ∈ M do
        if m does not violate P+, P− for any argument in G' then
            if m still exists in G' then
                replace m in G' with R;
            end
        end
    end
end
```

First, the algorithm evaluates whether $R$ actually is a valid replacement for $Q$ in the sense that $A(R) \cap C \subseteq A(Q) \cap C$.

Once this condition is established, matches are processed one by one. This evaluation examines whether there exist prohibited edges for the matched vertices in $G'$. If there is no violation of this condition, it must be evaluated whether all arguments specified in a match still exist in $G'$, since multiple matches may have common arguments, i.e.

a match may have become invalid since some of its vertices or edges may have already been altered or removed.

If this is not the case, the matched instance of the query graph is replaced in $G'$ by an instance of the replacement graph. This is achieved by removing all vertices that $R$ does not contain, but $Q$ does as well as all incident edges. Furthermore, new edges may also be added if they are included in the replacement graph, but did not exist in the query graph. Also old edges that do not exist in the replacement graph are removed. This procedure is repeated until all potential matches have been either replaced with $R$ or have been deemed unfit for replacement.

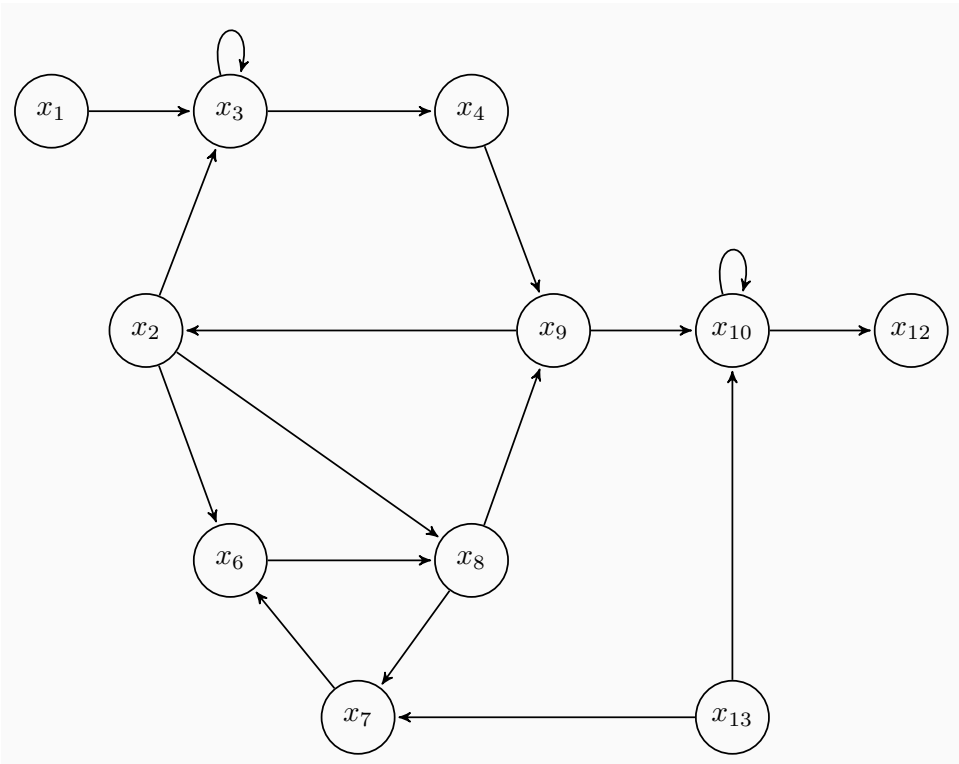



Figure 5.2: A possible result of applying the pattern of Proposition 2, BB-Graph and the replacement algorithm (10) to the graph in Figure 5.1.

**Example 19.** *Continuing from Example 18, we replace $F'$ with $G'$ as defined in Proposition 2, using line 10.*

*All nine options are examined:*

*(1) Let $C := X_1$ with $x_1$ matching $d$. This does not violate $P^-$ since $x_4$ and $x_5$ are not attacked. $P^+$ is empty and does not have to be checked for this pattern. This means we remove $x_5$ and all its relations and add $(x_3, x_3)$ as a self-attack to $E_G$.*

- *Since $x_5$ does not exist anymore, $X_1$ together with $x_2$ as $d$ can not be applied anymore.*

*(2) Let $C := X_2$. If $d \sim x_4$, then $x_6$ violates $P^+$ by attacking $x_8$. If, on the other hand we let $d \sim x_6$, then $x_4$ provides a violating attack on $x_9$. This means we can not replace $X_2$.*

*(3) Let $C := X_3$. Like in the previous point two vertices could be matched with $d$: $x_2$ and $x_{13}$, and violate $P^+$ in a similar way, by attacking different vertices matched to core arguments.*

*(4) Let $C := X_4$. Like with point (1), there are two attacks on the same core argument ($x_{10} \sim a$). Therefore one replacement can be executed. The resulting graph does not differ, regardless of whether match is replaced.*

*After the replacements in points (1) and (4), the graph has changed to look like depicted in Figure 5.2.*

This algorithm is available together with the modified BB-Graph implementation as well as some example query and replacement graphs, at the aforementioned location ([Bel18b]).

## 5.3 Experimental Evaluation

We use the procedure described in section 5.2 for replacing graphs with C-relativised equivalent graphs provided by C-relativised equivalency patterns to measure to what extent such replacements are capable of making the subsequent application of other algorithms (e.g. extension search algorithms) more efficient. For that purpose we describe the Data Sets and patterns which are used in the experiments, provide the results of these experiments and then discuss their ramifications.

### 5.3.1 Experimental Setup

We use two as yet unpublished Data Sets provided by the *Database and Artificial Intelligence Group* at the *TU Wien*[1] and three different patterns to evaluate the effects of pattern pre-processing. There are 5 repetitions of each experiment (i.e. 30 program runs) with a sample size of 50,000 randomised matches (or all matches, if that number is less than 50,000) from BB-Graph for each combination of query and database graph checked for pattern compliance. Such a sampling of results is necessary, since some combinations of query and database graphs have millions of isomorphic subgraphs, the checking of which is not feasible for this experiment.

[1] Private communication with Thomas Linsbichler, Wolfgang Dvořák and Stefan Woltran

### Patterns used

We evaluate three different patterns:

- *3cycle*: The odd-length-cycle pattern with $n = 3$ for C-stable equivalency (see Proposition 2),
- *evenpath*: The even-length path pattern described in Proposition 7 for C-stable equivalency (with $n = 4$),
- *compat*: The C-complete pattern proposed in Proposition 19.

### Data Sets

We use two distinct Data Sets of different origin. They both provide the combination of BB-Graph and the replacement algorithm with a set of database graphs.

Data Set 1 that we use includes two types of benchmark instances: (i) a total of 500 AFs based on the Watts-Strogatz model with the number of arguments $n \in \{500, 600, ..., 1500\}$ and parameters $k \in \{\lfloor \log_2(n) \rfloor - 1, \lfloor \log_2(n) \rfloor + 1\}$, $\beta \in \{0.1, 0.3, ..., 0.9\}$, and probCycles $\in \{0.1, 0.3, ..., 0.7\}$, using the generator AFBenchGen2 proposed in [CGV16] and employed in the 2017 ICCMA argumentation solver competition.

Data Set 2 includes 72 smaller instances specifically generated to include cycles of length three. They were generated as follows by using different random numbers introduced sub-AFs, arguments and attacks in each step:

1) An acyclic graph with $n$ numbered arguments $x_1, x_2, ..., x_n$ is generated. Then a number of attacks $(x_i, x_j)$ are added, where $i < j$ (i.e. they are forward edges) and each such attack has the same probability of being added.
2) 3cycle patterns and 4-cone sub-AFs are introduced into the instances at random positions in the AF, where for each attack added this way $i < j$ is no longer a condition, therefore producing cyclic graphs.
    a) 3cycle loops are introduced by choosing a random argument $x_i$ and introducing two new arguments $y$ and $z$ to form a cycle of length 3. Then new attacks $(z, x_j)$, where $j \neq i$ and $j$ is randomly chosen, are introduced to allow for backward edges.
    b) 4-cone sub-AFs are $K_3$ complete graphs where all of their arguments also attack an additional fourth argument. They are introduced by choosing a random argument $x_i$ and introducing three new arguments $u$, $v$ and $w$ to form a 4-cone where $x_i, u$ and $v$ constitute the $K_3$ and $w$ is the additional argument. Then new attacks $(w, x_j)$, where $j \neq i$ and $j$ is randomly chosen, are introduced. This may also introduce backward edges.

### 5.3.2 Results

The empirical data from the experimental program runs shows, that only a minority of files in Data Set 1 have matches that adhere to the pattern conditions. While BB-Graph always finds matches, in a majority of cases there are additional prohibited attacks, even when considering stable patterns which only have prohibited attacks in $P^-$, but not in $P^+$ (see Proposition 1). Data Set 2 on the other hand only provides for replaceable matches for the 3cycle pattern.

| | | 3cycle | evenpath | compat |
|---|---|---|---|---|
| Data Set 1 | graphs with repl. | 3 | 30 | 224 |
| | avg. repl. | 1 | 2.49 | 3.63 |
| | arg. change | -0.21% | 0% | -0.94% |
| | att. change | -0.77% | +0.3% | -4.36% |
| Data Set 2 | graphs with repl. | 72 | - | - |
| | avg. repl. | 30.38 | - | - |
| | arg. change | -7.67% | - | - |
| | att. change | -7.68% | - | - |

Table 5.1: Number of replaced matches and its effect.

Table 5.1 details the exact results obtained during empirical experimentation. It shows for both Data Sets the number of database graphs that contain at least one occurrence of $Q$ which could actually be replaced by $R$ (i.e. positive instances) as well as the average number of such successful replacements for these graph instances. Furthermore, the average percentage of change in arguments and attacks for graphs where local replacement happened, is given.

While instances of the 3cycle pattern can be found in all database graphs of Data Set 2 and there amounts to an average of about 30 replaced instances of the query graph per database graph, leading to a reduction of both arguments and attacks of more than seven percent on average. Conversely, the pattern occurs rarely in Data Set 1 and is only responsible for a change in argument and attack frequency of under one percent.

| Data Set 1 | | 3cycle | evenpath | compat |
|---|---|---|---|---|
| per instance | avg. repl. | 0.01 | 0.15 | 1.63 |
| avg. sum per run | removed arg. | 3 | 0 | 814 |
| | removed att. | 86 | 149.33 | 13545.2 |
| | added att. | 3 | 270.33 | 814 |

Table 5.2: Overall experiment results for Data Set 1.

On the other hand, the evenpath and compat patterns can only be found in Data Set 1, but did not occur in a single run with Data Set 2. In cases where the evenpath pattern was used for replacement, it lead to a change in the amount of attacks of only a third of a percent in 30 positive graph instances while not removing any arguments, since this pattern only changes attacks. compat had a higher rate of replacement with nearly half of instances having at least one replacement applied (224 out of 500 instances, an average of 3.63 replacements). This lead to a change in the attack number in positive instances of over 4 percent.

Since these results only concern positive results, the average number of replacements over all is considerably lower for Data Set 1, while staying the same for Data Set 2 (since there, either no or all instances are positive, depending on the pattern).

Table 5.2 shows the average number of replacement occurrences for Data Set 1 and their proportion of all, not just positive, instances. This means that for Data Set 1 the only three average replacable occurrences for the 3cycle pattern result in an average of 0.01 replacements per database graph. This number rises to 0.15 replacements for evenpath. Only compat, on average, achieves more than one replacement per database graph. To be able to better compare the impact of the patterns, the sums of removed arguments and attacks as well as added attacks per run of all 500 instances are also given.

Notice that the number of removed attacks far exceeds the number of removed arguments, suggesting a high vertex degree all over the data set, which might impede a higher number of replacements.

We now look more closely how the number of matches of the BB-Graph algorithm (recall that this is capped at a sample size of 50,000) and the number of positive replacement instances correlate. For that purpose we look at the experiments that have a larger number of replacements.

Figure 5.3 shows this relation for the compat pattern together with Data Set 1. We can see that the number of isomorphic subgraphs here is indirectly proportional to the number of replacments, that a higher number of replacements correlates with a lower number of isomorphic subgraphs.

On the other hand, Figure 5.4 shows a different, directly proportional correlation. Here, a higher number of BB-Graph matches directly correlates with a higher number of replacements. Also notice that these instances are not large enough to get to the limit of 50,000 matches.

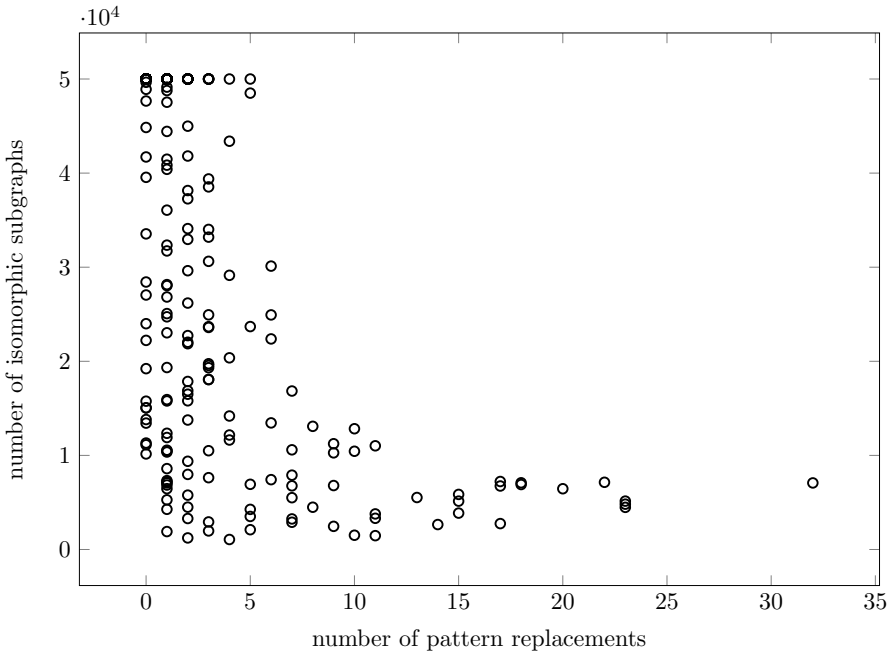



Figure 5.3: Relation of BB-Graph matches to actual replacements for Data Set 1 with compat.

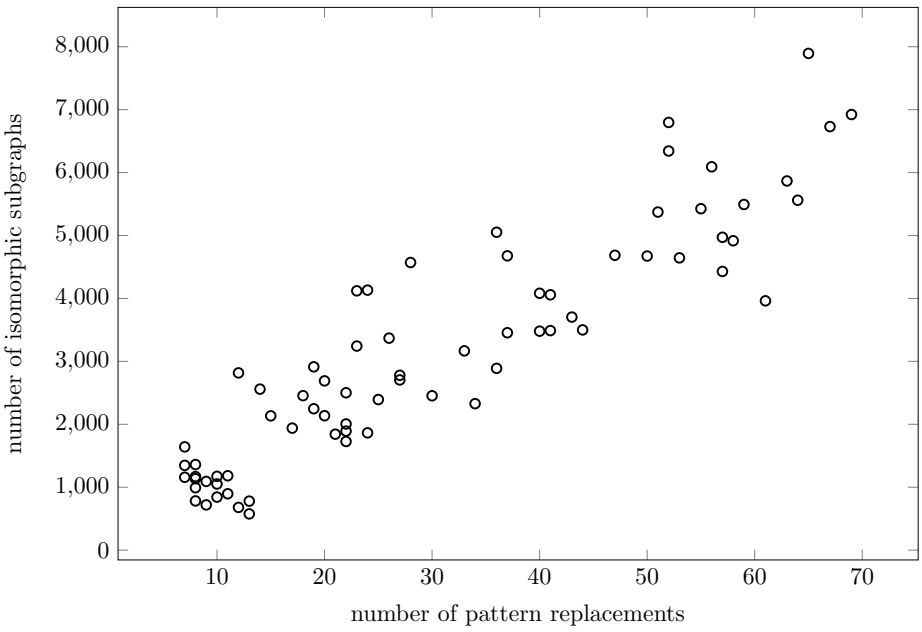



Figure 5.4: Relation of found isomorphic subgraphs to actual replacements for Data Set 2 with 3cycle.

### 5.3.3 Discussion of Results

The experiments show a high variance in replacement rates between the different combinations of data sets and patterns. This suggests, that it may be necessary to more closely analyse the impact of various generation methods on replacement rates. A first observation would be, that database graphs with high average vertex degrees do not lend themselves to pattern replacement since they make it unlikely that any given prohibited argument is in fact not related to another in an unwanted way.

Since there are combinations of data sets and patterns that achieve change rates in argument and attack number of multiple percent, it suggests that once a generation method is analysed, choosing fitting patterns and applying them, may lead to considerable computational advantages.

It might also be beneficial to find a way to avoid indirect proportionalities of isomorphic subgraphs found to actually replaced subgraphs like we see in Figure 5.3, but that may require a different algorithmic of handling isomorphic subgraphs.

CHAPTER 6

# Conclusion

This section provides a summary of the concepts presented within this thesis, how we used them and to what effect. Furthermore, possible future work on the topic of *C-relativised equivalency patterns* is discussed.

## 6.1 Summary

In this work we discussed *C-relativised equivalence* and its semantics. We introduced ASP encodings (provided at [Bel18a]) for these semantics and for their respective equivalence characterisations, that can be used to determine whether there are equivalencies between an arbitrary number of AFs.

We carefully analysed data about large numbers of small generated AFs using these encodings. To effectively express commonly occurring equivalencies, we formally introduced so-called *C-relativised equivalency patterns* of small sub-AFs that enable easy local replacements as long as some conditions are met. We found patterns for C-restricted stable, admissible, preferred an complete semantics as well as some generally applicable properties for these semantics.

Finally, a tool (provided at [Bel18b]) that finds pattern occurrences based on a branch-and-bound approach to find isomorphic subgraphs, combined with an evaluation of pattern conditions was provided to be able to conduct experiments about the patterns that were found. Using this program, we examined the effectiveness of pattern replacement by running experiments on two data sets of large AFs and analysing the frequency of pattern occurrences within them. The findings provided mixed results, showing that this initial approach could not yet provide a definitive method for automated pattern replacement.

## 6.2 Future Work

Direction for future work includes the search for more and larger C-relativised equivalency patterns for the semantics discussed herein.

An extension of C-relativised equivalency patterns to other semantics (see [BGGvdT18] for a discussion of further semantics) is another potential topic of research.

Furthermore, a different approach to pattern replacement is needed since some combinations of patterns and data produce an indirectly proportional relation between the number of isomorphic subgraphs and the number of applied replacements. For instance, one could prioritise non-intersecting isomorphic subgraphs or verify replacability as soon as a match is found instead of checking after collection of the full sample size. Another approach may be to use the filter-and-verification approach for subgraph isomorphism and integrating pattern features into such a filter.

It may also be beneficial to expand experiments to larger Data Sets such as those used at the *2017 International Competition on Computational Models of Argumentation* accessible at [ICC].

# Bibliography


[ABH+16] Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. Subtree isomorphism revisited. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1256–1271, 2016.

[AD13] Leila Amgoud and Caroline Devred. Argumentation frameworks as constraint satisfaction problems. *Ann. Math. Artif. Intell.*, 69(1):131–148, 2013.

[AY17] Merve Asiler and Adnan Yazici. Bb-graph: A new subgraph isomorphism algorithm for efficiently querying big graph databases. *CoRR*, abs/1706.06654, 2017.

[Bau12] Ringo Baumann. Normal and strong expansion equivalence for argumentation frameworks. *Artif. Intell.*, 193:18–44, 2012.

[BCG11] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *Knowledge Eng. Review*, 26(4):365–410, 2011.

[BD07] Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in artificial intelligence. *Artif. Intell.*, 171(10-15):619–641, 2007.

[BDLW17] Ringo Baumann, Wolfgang Dvorák, Thomas Linsbichler, and Stefan Woltran. A general notion of equivalence for abstract argumentation. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 800–806, 2017.

[Bel18a] Patrick Bellositz. C-relativised equivalence encodings. `https://github.com/TU-Wien-DBAI/argu-replacementpatterns/tree/master/asp`, 2018.

[Bel18b] Patrick Bellositz. C-relativised equivalency pattern pre-processing. `https://github.com/TU-Wien-DBAI/argu-replacementpatterns/tree/master/isosub_replace`, 2018.

[Ben03] Trevor J. M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *J. Log. Comput.*, 13(3):429–448, 2003.

[BET11] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[BFGV03] Horst Bunke, Pasquale Foggia, C. Guidobaldi, and Mario Vento. Graph clustering using the weighted minimum common supergraph. In *Graph Based Representations in Pattern Recognition, 4th IAPR International Workshop, GbRPR 2003, York, UK, June 30 - July 2, 2003, Proceedings*, pages 235–246, 2003.

[BGG05] Pietro Baroni, Massimiliano Giacomin, and Giovanni Guida. Scc-recursiveness: a general schema for argumentation semantics. *Artif. Intell.*, 168(1-2):162–210, 2005.

[BGGvdT18] Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors. *Handbook of Formal Argumentation.* College Publications, 2018.

[BH05] Philippe Besnard and Anthony Hunter. Practical first-order argumentation. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 590–595, 2005.

[BM09] Trevor J. M. Bench-Capon and Sanjay Modgil. Case law in extended argumentation frameworks. In *The 12th International Conference on Artificial Intelligence and Law, Proceedings of the Conference, June 8-12, 2009, Barcelona, Spain*, pages 118–127, 2009.

[CA08] Dan Cartwright and Katie Atkinson. Political engagement through tools for argumentation. In *Computational Models of Argument: Proceedings of COMMA 2008, Toulouse, France, May 28-30, 2008.*, pages 116–127, 2008.

[Cam06] Martin Caminada. Semi-stable semantics. In *Computational Models of Argument: Proceedings of COMMA 2006, September 11-12, 2006, Liverpool, UK*, pages 121–130, 2006.

[CCIL08] Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. Computable functions in ASP: theory and implementation. In *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, pages 407–424, 2008.

[CDGV13] Federico Cerutti, Paul E. Dunne, Massimiliano Giacomin, and Mauro Vallati. Computing preferred extensions in abstract argumentation: A sat-based approach. In *Theory and Applications of Formal Argumentation*

*- Second International Workshop, TAFA 2013, Beijing, China, August 3-5, 2013, Revised Selected papers*, pages 176–193, 2013.

[CDM05] Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Prudent semantics for argumentation frameworks. In *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2005), 14-16 November 2005, Hong Kong, China*, pages 568–572, 2005.

[CGV16] Federico Cerutti, Massimiliano Giacomin, and Mauro Vallati. Generating structured argumentation frameworks: Afbenchgen2. In *Computational Models of Argument - Proceedings of COMMA 2016, Potsdam, Germany, 12-16 September, 2016.*, pages 467–468, 2016.

[DFI+03] Tina Dell'Armi, Wolfgang Faber, Giuseppe Ielpa, Nicola Leone, and Gerald Pfeifer. Aggregate functions in disjunctive logic programming: Semantics, complexity, and implementation in DLV. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 847–852, 2003.

[DKT09] Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. Assumption-based argumentation. In *Argumentation in Artificial Intelligence*, pages 199–218. 2009.

[DMT07] Phan Minh Dung, Paolo Mancarella, and Francesca Toni. Computing ideal sceptical argumentation. *Artif. Intell.*, 171(10-15):642–674, 2007.

[Dun95] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.

[EGW08] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. ASPARTIX: implementing argumentation frameworks using answer-set programming. In *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, pages 734–738, 2008.

[EGW10] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. Answer-set programming encodings for argumentation frameworks. *Argument & Computation*, 1(2):147–177, 2010.

[GKNS07] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. *clasp* : A conflict-driven answer set solver. In *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, pages 260–265, 2007.

[GMR+15] Sarah Alice Gaggl, Norbert Manthey, Alessandro Ronca, Johannes Peter Wallner, and Stefan Woltran. Improved answer-set programming encodings for abstract argumentation. *TPLP*, 15(4-5):434–448, 2015.

[GS04] Alejandro Javier García and Guillermo Ricardo Simari. Defeasible logic programming: An argumentative approach. *TPLP*, 4(1-2):95–138, 2004.

[ICC] Webpage of the international competition on computational models of argumentation. `http://argumentationcompetition.org/2017/results.html`. Accessed: 2018-08-16.

[JP09] Sindhu Joseph and Henry Prakken. Coherence-driven argumentation to norm consensus. In *The 12th International Conference on Artificial Intelligence and Law, Proceedings of the Conference, June 8-12, 2009, Barcelona, Spain*, pages 58–67, 2009.

[Lif08] Vladimir Lifschitz. What is answer set programming? In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 1594–1597, 2008.

[LON11] Hengfei Li, Nir Oren, and Timothy J. Norman. Probabilistic argumentation frameworks. In *Theorie and Applications of Formal Argumentation - First International Workshop, TAFA 2011. Barcelona, Spain, July 16-17, 2011, Revised Selected Papers*, pages 1–16, 2011.

[LPF+06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[MHP07] Peter McBurney, David Hitchcock, and Simon Parsons. The eightfold way of deliberation dialogue. *Int. J. Intell. Syst.*, 22(1):95–132, 2007.

[MNG+10] Misael Mongiovì, Raffaele Di Natale, Rosalba Giugno, Alfredo Pulvirenti, Alfredo Ferro, and Roded Sharan. Sigma: a set-cover-based inexact graph matching algorithm. *J. Bioinformatics and Computational Biology*, 8(2):199–218, 2010.

[NS97] Ilkka Niemelä and Patrik Simons. Smodels - an implementation of the stable model and well-founded semantics for normal LP. In *Logic Programming and Nonmonotonic Reasoning, 4th International Conference, LPNMR'97, Dagstuhl Castle, Germany, July 28-31, 1997, Proceedings*, pages 421–430, 1997.

[OW11] Emilia Oikarinen and Stefan Woltran. Characterizing strong equivalence for argumentation frameworks. *Artif. Intell.*, 175(14-15):1985–2009, 2011.

[SR09] Guillermo Ricardo Simari and Iyad Rahwan, editors. *Argumentation in Artificial Intelligence.* Springer, 2009.

[SZLY08] Haichuan Shang, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *PVLDB*, 1(1):364–375, 2008.

[TC14] Nouredine Tamani and Madalina Croitoru. Fuzzy argumentation system for decision support. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems - 15th International Conference, IPMU 2014, Montpellier, France, July 15-19, 2014, Proceedings, Part I*, pages 77–86, 2014.

[Ull76] Julian R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

[Ull10] Julian R. Ullmann. Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. *ACM Journal of Experimental Algorithmics*, 15, 2010.

[ZHY07] Shijie Zhang, Meng Hu, and Jiong Yang. Treepi: A novel graph indexing method. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 966–975, 2007.

[ZYY07] Peixiang Zhao, Jeffrey Xu Yu, and Philip S. Yu. Graph indexing: Tree + delta >= graph. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 938–949, 2007.