FernUniversität in Hagen
Fakultät für Mathematik und Informatik

Artificial Intelligence Group

# Entwicklung und Implementierung von Rangbasierten Semantiken für SetAFs

## Masterarbeit

zur Erlangung des Grades eines Master of Science (M.Sc.)
im Studiengang Informatik

vorgelegt von
Dimitrij Pauls
dimitrij.pauls@outlook.de

| | |
|---|---|
| Erstgutachter: | Prof. Dr. Matthias Thimm<br>Artificial Intelligence Group |
| Betreuer: | Kenneth Skiba<br>Artificial Intelligence Group |

# Erklärung

Ich erkläre, dass ich die Masterarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Arbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

| | Ja | Nein |
|---|---|---|
| Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. | ☐ | ☐ |
| Der Veröffentlichung dieser Arbeit auf der Webseite des Lehrgebiets Künstliche Intelligenz stimme ich zu. | ☐ | ☐ |
| Der Text dieser Arbeit ist unter einer Creative Commons Lizenz (CC BY-SA 4.0) verfügbar. | ☐ | ☐ |
| Der Quellcode ist unter einer GNU General Public License (GPLv3) verfügbar. | ☐ | ☐ |
| Die erhobenen Daten sind unter einer Creative Commons Lizenz (CC BY-SA 4.0) verfügbar. | ☐ | ☐ |

.....

(Ort, Datum) (Unterschrift)

## Zusammenfassung

Ein Ansatz der Analyse von Konfliktsituationen liegt in der Untersuchung sogenannter *Argumentationsframeworks*. Diese erlauben eine Konfliktmodellierung anhand gerichteter Graphen, in denen jedes Argument durch einen Knoten repräsentiert wird. Die Kanten entsprechen den Widersprüchen (*Angriffen*) der Argumente untereinander.

Die *Mengen-Argumentationsframeworks* erweitern die normalen Frameworks. Mit ihrer Hilfe wird die Modellierung von Angriffen durch Mengen von Argumenten ermöglicht. Sowohl die normalen als auch die erweiterten Frameworks lassen sich durch *Rangsemantiken* analysieren. Über diese Funktionen können die Argumente relativ zueinander anhand unterschiedlicher Kriterien bewertet werden.

In der vorliegenden Arbeit wird eine für reguläre Argumentationsframeworks definierte Rangsemantik gewählt. Für diese wird eine zu den Mengen-Argumentationsframeworks kompatible Analogie eingeführt. Nach der Definition erfolgt eine Untersuchung der mathematischen Eigenschaften der neuen Funktion. Zusätzlich wird ein Algorithmus zur Berechnung der Semantik definiert. Dieser wird schließlich in einer Programmiersprache umgesetzt und die Implementierung auf konkreten Instanzen von Mengen-Argumentationsframeworks evaluiert.



## Abstract

*Argumentation frameworks* define one possible approach for the analysis of conflicts. They allow modelling of a conflict through directed graphs, where every argument is represented by a node. The edges define contradictions (often called *attacks*) between the arguments.

*Set argumentation frameworks* are an extension of the regular frameworks. In those an attack can be initiated by a set of multiple arguments. Both types of frameworks can be analysed by *ranking-based semantics* that sort all arguments relative to each other based on different criteria.

In this thesis one specific ranking-based semantics is taken that is defined for regular argumentation frameworks. An analogous semantics is then defined for the set argumentation frameworks. Afterwards the new semantics is analysed in regard to its mathematical properties. Additionally, an algorithm is defined that can compute the results of the semantics. Lastly, this algorithm is implemented and analysed on different specific instances of set argumentation frameworks.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 1. Einleitung

Gewisse Problemgebiete besitzen eine Komplexität, die eine begründete und vorurteilsfreie Meinungsbildung erschwert. Bei einem häufig genutzten Beispiel in diesem Zusammenhang geht es um die Wahl einer medizinischen Behandlung bei bestimmten Krankheitsbildern. Für jedes Medikament und jeden gerechtfertigten Eingriff gibt es Gegenargumente. Bei Abwägung der positiven Effekte gegen die möglichen Nebenwirkungen müssen mindestens das Alter und Geschlecht der behandelten Person berücksichtigt werden. Auch andere Krankheiten und Unverträglichkeiten, weitere eingenommene Medikamente und viele zusätzliche Aspekte dienen als Faktoren für oder gegen jede Entscheidung.

Ein anderes Beispiel ist die Wahl des Strafmaßes zu einem Vergehen. Auch hier ist jede Entscheidung von Pro- und Kontraargumenten umgeben. Jegliche mildernde und erschwerende Umstände müssen in Betracht gezogen werden. Diese können oft widersprüchlich sein oder auch zyklisch argumentieren. Der Zweck der Strafe ist ein Aspekt, der die Entscheidung zusätzlich erschwert. So kann eine Strafe mit General- und Spezialprävention oder aber Vergeltung motiviert werden.

Die Komplexität in den obigen Beispielen erlaubt viele Fehlermöglichkeiten in Bereichen, in denen Fehler schwerwiegende Konsequenzen haben können. Andere Domänen sind ebenfalls mit ähnlichen Problemen konfrontiert. Das legt nahe, die Zusammenhänge und Prozesse der Argumentation zu abstrahieren und zu formalisieren. Erst mit einer derartigen Grundlage können Verfahren entworfen werden, die aus einer Problemstellung und zugehörigen Argumenten eine Bewertung systematisch berechnen können.

Die Idee eines solchen Formalismus liegt dem Gebiet der *abstrakten Argumentation* nach [Dun95] zugrunde. Die Grundlagen davon werden im nachfolgenden Kapitel 2 „Abstrakte Argumentation“ formal eingeführt. Zunächst werden die sogenannten Argumentationsframeworks definiert, die der üblichen Modellierungsart von Konflikten entsprechen. Bei diesen besteht ein Konflikt aus der Menge seiner Argumente. Die Widersprüche zwischen Argumenten werden durch gerichtete Eins-zu-eins-Beziehungen dargestellt. Daneben werden die „Mengen-Argumentationsframeworks“ vorgestellt, bei denen mehrere Argumente im Zusammenschluss ein weiteres Argument „angreifen“ können (vgl. [NP06]). Für beide Modellierungswege werden zuletzt Methoden für eine Konfliktanalyse gezeigt. Als die bedeutendste Methode im Kontext dieser Arbeit können die „rangbasierten Semantiken“ genannt werden ([ABN13]). Diese definieren Kriterien für die Bewertung der Gültigkeit von Argumenten und ordnen die Argumente anhand dieser Kriterien untereinander ein.

Eine konkrete rangbasierte Semantik aus [ABN13] namens „Belastungssemantik“

wird in Kapitel 2 eingeführt. Diese ist im Original für normale Argumentationsframeworks nach [Dun95] definiert. Ihre Funktionsweise besteht darin, eine *Belastung* für die Argumente zu bestimmen. Diese Belastung wird für jedes Argument anhand seiner Angreifer sowie der Angreiferbelastungen berechnet. Eines der Ziele dieser Masterarbeit ist die Definition einer analogen, rangbasierten Semantik für Mengen-Argumentationsframeworks. In Kapitel 3 „Belastungssemantik in SetAFs“ folgt eine Auseinandersetzung mit dieser Aufgabe. Darin wird eine solche Semantik definiert und auf ihre mathematischen Eigenschaften untersucht.

Da es sich bei den Angreifern in regulären Argumentationsframeworks um einzelne Argumente handelt, sind ihre Belastungen klar definiert. Bei den Mengen-Argumentationsframeworks besteht eine Angreifermenge jedoch aus mehreren Argumenten. Somit existieren verschiedene Möglichkeiten, die Belastung des gesamten Angriffs zu bestimmen. In Kapitel 3 wird die konventionelle Methode im Kontext der Mengen-Argumentationsframeworks untersucht. Die Angriffskraft einer Argumentenmenge hängt bei dieser Vorgehensweise ausschließlich von dem schwächsten Mengenelement ab. In Kapitel 4 „Alternative Definitionen“ werden andere Optionen vorgestellt und analysiert.

Neben der formalen Definition der genannten Semantik wird in der vorliegenden Masterarbeit das Ziel verfolgt, die neu eingeführte Semantik auszuprogrammieren. Eine Voraussetzung dafür bildet die algorithmische Darstellung der Semantik. Die Durchführung dieser Teilaufgabe wird in Kapitel 5 „Algorithmus“ dokumentiert. Darin wird der Algorithmus in Form von Pseudocode beschrieben und seine Funktionsweise erläutert. Parallel erfolgt eine Untersuchung der Terminierung und der Laufzeit von dem Algorithmus.

Die konkrete Umsetzung des Verfahrens wird in Kapitel 6 „Implementierung“ beschrieben. Hier werden zunächst theoretische Überlegungen zur Vorbereitung der Implementierung erläutert. Dazu gehört eine begründete Strukturierung der Module und Klassen für die Umsetzung. Diesen Erläuterungen folgt eine Beschreibung der konkreten Programmierentscheidungen. Schließlich werden die für die Funktionsweise der Implementierung kritischen Codeausschnitte demonstriert.

Im nachfolgenden Kapitel 7 „Evaluation der Implementierung“ liegt der Fokus auf dem letzten Ziel der Masterarbeit – Evaluierung der implementierten Lösung. Zu diesem Zweck werden unterschiedliche Testinstanzen generiert. Ein Teil dieser wird manuell erstellt. Sie dienen nicht nur der Prüfung des normalen Ausführungsverlaufs, sondern auch diverser Sonderfälle. Auf dieser Basis wird die Korrektheit der Implementierung getestet. Ein anderer, automatisch generierter Teil der Testinstanzen wird zum Messen der Ausführungsgeschwindigkeit verwendet. Nach der Vorstellung der Testinstanzen und ihrer Generierungsmethoden werden die Ergebnisse der Ausführung gezeigt. Zuletzt werden Aussagen darüber getroffen, ob die Ergebnisse den Erwartungen aus dem Algorithmus-Kapitel entsprechen.

Im letzten Kapitel 8 „Fazit“ werden die Erkenntnisse der Arbeit zusammengefasst. Die Ergebnisse werden bewertet und den ursprünglichen Zielen gegenübergestellt. Abschließend wird ein Ausblick auf die Möglichkeiten zur Ergänzung und

Fortsetzung der Masterarbeit gegeben.

# 2. Abstrakte Argumentation

Als Basis der Modellierung von Konfliktsituationen nach [Dun95] dienen gerichtete Graphen. Ihre Knoten entsprechen den eigentlichen Argumenten. Diese Knoten sind von den qualitativen und inhaltlichen Eigenschaften abstrahiert, die ggf. Teil ihrer ursprünglichen Argumente sind. Der Fokus der Graphen liegt in der Aussage, welche Argumente sich untereinander widersprechen. In der Literatur werden solche Widersprüche meist *Angriffe* genannt und durch die Kanten der Graphen abgebildet. In den nachfolgenden Abschnitten dieses Kapitels werden unterschiedliche Arten der Modellierung und der Analyse dieser Graphen vorgestellt.

## 2.1. Argumentationsframeworks

Die mathematische Beschreibung der Argumentationsframework-Graphen wird in der Definition 2.1.1 gezeigt.

**Definition 2.1.1.** Ein *(abstraktes) Argumentationsframework* $AF$ nach [Dun95] ist ein Paar $(Ar, att)$, in dem $Ar$ einer endlichen Menge von Argumenten entspricht. $att \subseteq Ar \times Ar$ ist die Menge der gegenseitigen Angriffe der Argumente untereinander. $a$ *greift* $b$ *an* für $a, b \in Ar$, wenn $(a, b) \in att$.

Für eine kompakte Schreibweise werden zudem folgende Abkürzungen eingeführt (für ein Argumentationsframework $AF = (Ar, att)$, $a \in Ar, Args \subseteq Ar$):

$$a^+ := \{b \in Ar \mid (a, b) \in att\} \tag{2.1}$$

$$a^- := \{b \in Ar \mid (b, a) \in att\} \tag{2.2}$$

$$Args^+ := \{b \in Ar \mid \exists a \in Args : (a, b) \in att\} \tag{2.3}$$

$$Args^- := \{b \in Ar \mid \exists a \in Args : (b, a) \in att\} \tag{2.4}$$

Informell ausgedrückt bildet $a^+$ die Menge der von $a$ angegriffenen Argumente und $a^-$ die Menge der Angreifer von $a$. $Args^+$ beinhaltet Argumente, die von den Elementen aus $Args$ angegriffen werden. Umgekehrt greifen die Argumente in $Args^-$ die Elemente in $Args$ an.

Zur Illustration der Modellierung von Argumentationsframeworks diene die Situation von zwei Mitbewohnern – Alex und Ben. Ihr Konflikt besteht in der Uneinigkeit über die Zuständigkeit für das Geschirr.

**Beispiel 2.1.1.** Argument $A_1$ von Alex sei „Ich war gestern schon für das Geschirr zuständig“ und Bens Antwort $B_1$ „Ich habe heute schon das gesamte Badezimmer

geputzt“. Die Konfliktsituation sei durch ein Argumentationsframework $AF$ modelliert mit $AF = (Ar, att)$, $Ar = \{A_1, B_1\}$ und $att = \emptyset$.

Die Argumente $A_1$ und $B_1$ stellen die Gültigkeit des jeweils anderen nicht infrage. Das begründet die gewählte Modellierungsart, die keine Angriffe definiert. Daraus ergibt sich ein eher uninteressanter Graph wie in Abbildung 2.1 dargestellt.

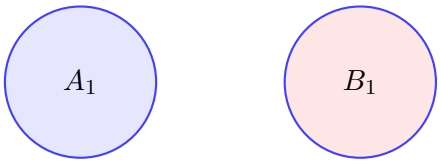



Abbildung 2.1.: Simples Argumentationsframework ohne Angriffe

**Beispiel 2.1.2.** Die Argumente $A_1$ und $B_1$ seien wie im Beispiel 2.1.1 definiert. Zusätzlich laute das Argument $A_2$ von Alex „Badezimmer- und Geschirr-Putzpläne haben nichts miteinander zu tun“. Die Situation wird mit $AF = (Ar, att)$, $Ar = \{A_1, A_2, B_1\}, att = \{(A_2, B_1)\}$ modelliert. Damit wird die Situation nun um einen Angriff erweitert, da das Argument $A_2$ dem Argument $B_1$ widerspricht. Die veränderte Konfliktsituation lässt sich durch Abbildung 2.2 grafisch darstellen.

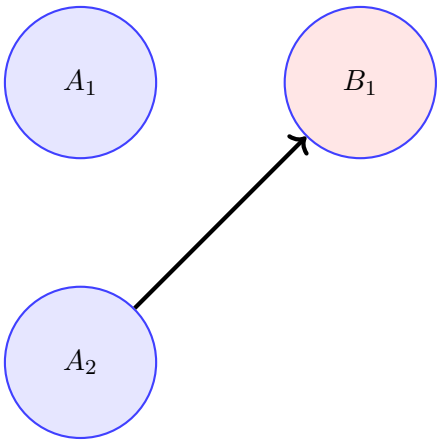



Abbildung 2.2.: Simples Argumentationsframework mit einem Angriff

Dabei ist zu erwähnen, dass die im Beispiel 2.1.2 gezeigte Modellierungsart lediglich einer Interpretation der dargestellten Situation entspricht. Das erzeugte Argumentationsframework beschreibt die Argumente zwar auf eine logische und konsistente Weise, dennoch sind auch andere Angriffsdefinitionen denkbar. Beispielsweise lässt sich der Zustand auch so abbilden, dass jedes Argument von Alex alle Argumente von Ben angreift und umgekehrt (s. Beispiel 2.1.3 und zugehörige Abbildung 2.3).

**Beispiel 2.1.3.** Die Argumente $A_1, A_2, B_1$ seien wie im Beispiel 2.1.2 definiert. Der Konflikt sei durch ein Argumentationsframework $AF$ modelliert mit $AF = (Ar, att)$, $Ar = \{A_1, A_2, B_1\}$, $att = \{(A_1, B_1), (A_2, B_1), (B_1, A_1), (B_1, A_2)\}$.

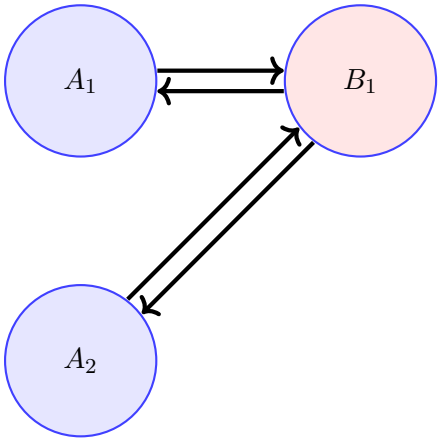



Abbildung 2.3.: Simples Argumentationsframework mit mehreren Angriffen

Trotz verschiedener theoretischer Möglichkeiten, den Sachverhalt zu modellieren, ist der beschriebene Zustand einfach. Realistische Szenarios besitzen eine weit höhere Komplexität und entsprechend kann es komplexer sein, einen Konsensus zu ihrer „richtigen“ Modellierung zu finden.

## 2.2. Semantiken

Die Erzeugung von Argumentationsframeworks erlaubt eine Analyse der zugrundeliegenden Argumente. In der Literatur sind für diesen Zweck zwei Ansätze am häufigsten anzutreffen – *Extension-basierte Semantiken* und *Labelling-basierte Semantiken*. Die nachfolgende kurze Einführung basiert auf [Dun95] und [BCG18] für Extension-basierte Semantiken. Für die Beschreibung der Labelling-basierten Semantiken wurde die Information aus [Cam06] sowie [BCG18] verwendet.

**Definition 2.2.1.** Eine *Extension-basierte Semantik* (kurz Extension-Semantik) $\sigma$ ist eine Funktion, die ein beliebiges Argumentationsframework $AF = (Ar, att)$ auf eine Menge $Exts \subseteq \mathcal{P}(Ar)$ abbildet[1].

Die Teilmengen $Ext \in Exts$ werden *Extensions* genannt. Für praktische Zwecke sollte jede Extension eine sinnvolle Position im Ausgangskonflikt repräsentieren. Die obige Definition schränkt die Art und Weise der Erzeugung von Extensions allerdings nicht ein. Aus diesem Grund existieren diverse Eigenschaften, mit denen

[1] Das Symbol $\mathcal{P}$ steht hier für die Potenzmenge.

die Semantiken bzw. ihre Extensions in Bezug auf unterschiedliche Merkmale bewertet werden können. Ein einfaches Beispiel ist die in Definition 2.2.2 beschriebene *Konfliktfreiheit*.

**Definition 2.2.2.** $AF = (Ar, att)$ sei ein beliebiges Argumentationsframework und $S \subseteq Ar$ eine Extension. $S$ heißt genau dann *konfliktfrei*, wenn kein Argumentenpaar $a, b \in S$ existiert, sodass $(a, b)$ in $att$ liegt.

Die *naive Semantik* wird häufig als ein konkretes Beispiel einer konfliktfreien Semantik demonstriert.

**Definition 2.2.3.** $AF = (Ar, att)$ sei ein beliebiges Argumentationsframework. Eine Menge $S \in Ar$ entspricht genau dann einer *naiven Extension*, wenn die folgenden zwei Bedingungen erfüllt sind:

- $S$ ist konfliktfrei.
- Es existiert keine konfliktfreie Menge $T \in Ar$, sodass $|T| > |S|$.

Als eine Verallgemeinerung der Extension-Semantiken werden manchmal Labelling-basierte Semantiken angesehen.

**Definition 2.2.4.** $AF = (Ar, att)$ sei ein Argumentationsframework und $\Lambda$ eine Menge von Kennzeichen (orig. Labels). Die Funktion $\Lambda$-*Labelling* $Lab : Ar \to \Lambda$ bildet jedes Argument aus $Ar$ auf ein Kennzeichen ab. Eine *Labelling-basierte Semantik* (kurz Labelling-Semantik) $\sigma$ bildet ein beliebiges Argumentationsframework $AF$ auf eine Menge von $\Lambda$-Labellings ab.

Für ein Argumentationsframework $AF = (Ar, att)$ können beliebige Elemente als Kennzeichen von Argumenten dienen. Häufig werden jedoch drei Kennzeichen gewählt – *in*, *out* und *undec* (vgl. bspw. [RL08]). Die Argumente aus $Ar$, die von einem entsprechenden Labelling mit *in* gekennzeichnet werden, gelten dann als klar akzeptiert ($in(Ar)$). Die mit *out* gekennzeichneten Argumente gelten als klar widerlegt ($out(Ar)$). Bei den übrigen ($undec(Ar)$) Argumenten lässt sich keine eindeutige Aussage treffen.

Die zuvor genannten Eigenschaften zur Bewertung von Extension-Semantiken existieren ebenso für die {*in*, *out*, *undec*}-Labelling-Semantiken. Ein Labelling, das einer konfliktfreien Extension $Ext$ entspricht, sieht bspw. so aus:

$$\begin{aligned} in(Ar) &= Ext \\ out(Ar) &= Ext^{+} \\ undec(Ar) &= Ar \setminus (Ext \cup Ext^{+}) \end{aligned}$$

Entsprechend kann die naive Semantik aus Definition 2.2.3 auch als eine {*in*, *out*, *undec*}-Labelling-Semantik beschrieben werden.

**Definition 2.2.5.** $AF = (Ar, att)$ sei ein beliebiges Argumentationsframework. Ein Labelling $\Lambda$ heißt genau dann *naiv*, wenn die folgenden zwei Bedingungen erfüllt sind:

- Das Labelling ist konfliktfrei.
- Die Menge $in(Ar)$ ist maximal (es existiert kein konfliktfreies Labelling mit einer größeren $in$-Menge).

## 2.3. Rangbasierte Semantiken

Ein Kritikpunkt bei den Extension- und Labelling-Semantiken bezieht sich auf ihre „alles-oder-nichts“ Funktionsweise. So ist es beispielsweise bei den Extension-Semantiken so, dass ein Argument entweder im Kontext einer Extension akzeptiert wird oder nicht. Eine ähnliche binäre Situation ergibt sich auch bei den Labelling-Semantiken. Es kann jedoch sinnvoll sein, die Bewertung der Argumente feiner zu gestalten und sie relativ zueinander einzuordnen.

Das Werk [ABN13] zeigt eine Möglichkeit für eine derartige Sortierung der Argumente durch die Einführung der *Rangrelation*, sowie der darauf aufbauenden *rangbasierten Semantiken*. Als Symbol der Relation wird nachfolgend „$\succcurlyeq$“ verwendet. $a \succcurlyeq b$ für beliebige Argumente $a, b \in Ar$ bedeutet, dass $a$ mindestens gleichwertig zu $b$ ist. Mit dieser Grundlage beschreibt [ABN13] dann eine rangbasierte Semantik analog zu der folgenden Definition 2.3.1.

**Definition 2.3.1.** Eine *rangbasierte Semantik* (kurz *Rangsemantik*) ist eine Funktion $S$, die jedes Argumentationsframework $AF = (Ar, att)$ auf eine Rangrelation von $Ar$ abbildet.

### 2.3.1. Belastungssemantik

In [ABN13] wird als Beispiel einer Rangsemantik die *Burden-based* Semantik eingeführt, wobei diese hier *Belastungssemantik* genannt wird. Die Idee basiert auf der Berechnung einer *Belastung* (Burden), die auf jedem Knoten liegt. Der Belastungswert auf einem Argument $a$ hängt von der Zahl sowie der Belastung der Angreifer von $a$ ab.

**Definition 2.3.2.** $AF = (Ar, att)$ sei ein Argumentationsframework und $a \in Ar$. Die *Belastung* $B_i(a)$ für $i \in \mathbb{N}_0$ ist wie folgt definiert:

$$B_i(a) = \begin{cases} 1, & \text{falls } i = 0 \\ 1, & \text{falls } a^- = \emptyset \\ 1 + \sum\limits_{b \in a^-} \frac{1}{B_{i-1}(b)}, & \text{sonst} \end{cases}$$

**Definition 2.3.3.** $AF = (Ar, att)$ sei ein Argumentationsframework und $a, b \in Ar$. Der Belastungsrang $\succcurlyeq_B$ wird über die Belastungswerte wie folgt definiert:
$a \succcurlyeq_B b$ gilt genau dann, wenn:

- $B_i(a) = B_i(b)\ \forall i \in \mathbb{N}_0$ **oder**
- $\exists i \in \mathbb{N}, B_i(a) < B_i(b)$ *und* $\forall j \in \{0, 1 \ldots i-1\}, B_j(a) = B_j(b)$

Die Berechnung vom Belastungsrang wird nachfolgend am Beispiel 2.3.1 demonstriert.

**Beispiel 2.3.1.** $AF = (Ar, att)$ sei ein abstraktes Argumentationsframework mit $Ar = \{A_1, A_2, A_3, A_4\}$ und $att = \{(A_1, A_2), (A_1, A_3), (A_2, A_3), (A_2, A_4), (A_3, A_4)\}$. Eine grafische Darstellung von $AF$ findet sich in Abbildung 2.4.

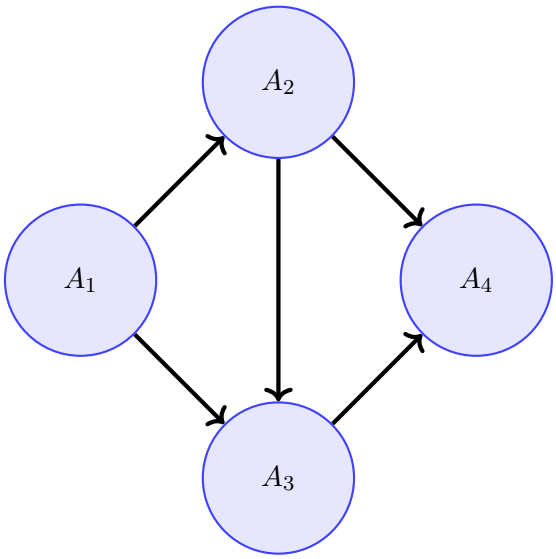



Abbildung 2.4.: Beispiel-Argumentationsframework zur Berechnung der Belastungssemantik

Der initiale Schritt ($i = 0$) ist trivial – $B_0(A_1) = B_0(A_2) = B_0(A_3) = B_0(A_4) = 1$. Die Anwendung der Formel ergibt für den nächsten Schritt ($i = 1$) folgende Werte:

$$B_1(A_1) = 1$$
$$B_1(A_2) = 2$$
$$B_1(A_3) = 3$$
$$B_1(A_4) = 3$$

Die Belastung von $A_3$ und $A_4$ bleibt weiter gleich. Damit ist es noch unbekannt, ob $A_3 \succcurlyeq_B A_4$ oder $A_4 \succcurlyeq_B A_3$ gilt. Sämtliche andere Relationen lassen sich allerdings bereits feststellen:

$$A_1 \succcurlyeq_B A_1, A_1 \succcurlyeq_B A_2, A_1 \succcurlyeq_B A_3, A_1 \succcurlyeq_B A_4$$
$$A_2 \succcurlyeq_B A_2, A_2 \succcurlyeq_B A_3, A_2 \succcurlyeq_B A_4$$
$$A_3 \succcurlyeq_B A_3$$
$$A_4 \succcurlyeq_B A_4$$

Die nächste Iteration ($i = 2$) ergibt:

$$B_2(A_1) = 1$$
$$B_2(A_2) = 2$$
$$B_2(A_3) = 2,5$$
$$B_2(A_4) \approx 1,83$$

Damit lässt sich nun die letzte fehlende Information – $A_4 \succcurlyeq_B A_3$ – bestimmen.

Dabei fällt auf, dass im letzten Schritt die Belastung von $A_4$ unter die Belastung von $A_2$ fällt. Da das Verhältnis zwischen den beiden Argumenten bereits im vorherigen Schritt fixiert wurde, hat das keine Auswirkung auf das Ergebnis.

## 2.4. Mengen-Argumentationsframeworks (SetAFs)

In der Realität existieren Konfliktsituationen, die mit den Modellierungswerkzeugen aus [Dun95] nur ungenau abgebildet werden können. Um das am Beispiel 2.4.1 zu demonstrieren, greifen wir die Situation der beiden Mitbewohner Alex und Ben auf.

**Beispiel 2.4.1.** Das Argument $A_1$ von Alex sei erneut „Ich war gestern schon für das Geschirr zuständig“. Diesem entgegnet Ben mit folgenden zwei Argumenten: „Ich habe gestern kein dreckiges Geschirr produziert“ ($B_1$) und „Der Geschirrputzplan gilt nur, wenn wir beide dreckiges Geschirr produzieren“ ($B_2$).

Das entsprechende Argumentationsframework $AF$ sei als $AF = (Ar, att)$ mit $Ar = \{A_1, B_1, B_2\}$ definiert. Mit den bisherigen Modellierungsmöglichkeiten ist es schwer, die Angriffsmenge $att$ sinnvoll abzubilden. Eine Option wäre, dass $B_1$ und $B_2$ jeweils einzeln $A_1$ angreifen ($att = \{(B_1, A_1), (B_2, A_1)\}$). Das gibt jedoch die Situation inkorrekt wieder, denn die Gültigkeit von nur einem dieser Argumente ohne jeweils das andere stellt die Gültigkeit von $A_1$ nicht infrage. Eine weitere Option besteht darin, dass weder $B_1$ noch $B_2$ das Argument von Alex angreifen ($att = \emptyset$). Diese Modellierung versagt allerdings bei der Wiedergabe der bedeutendsten Information zu der echten Situation. Die übrigen Möglichkeiten, nur von einem der beiden Argumente den Angriff abzubilden, vereinen beide Nachteile – sie verlieren Information und stellen gleichzeitig keine präzise Abbildung der Realität dar.

Am Beispiel 2.4.1 ist ersichtlich, dass bestimmte Argumente im Zusammenschluss einen Angriff hervorbringen, nicht jedoch einzeln. Es bietet sich an, für diese Tatsache auch Modellierungsmöglichkeiten zu schaffen. Darin besteht die Aufgabe der Arbeit [NP06], wobei derartige Argumentationsframeworks dort *Argumentationssysteme* genannt und wie in Definition 2.4.1 festgelegt werden. Im Artikel [FB19] wird diese Modellierung tiefer ausgeleuchtet und um mehrere Parallelen zu normalen Argumentationsframeworks erweitert.

Für die vorliegende Masterarbeit wird die Definition der Argumentationssysteme aus [NP06] übernommen, allerdings wird hier dafür aus Platzgründen die Abkürzung *SetAF* verwendet. Diese steht übersetzt und ausgeschrieben für *(Abstraktes) Mengen-Argumentationsframework*.

**Definition 2.4.1.** Ein *SetAF* ist ein Paar $AF = (Ar, att)$, wobei $Ar$ eine endliche, nichtleere Menge von Argumenten ist. $att \subseteq (\mathcal{P}(Ar) \setminus \{\emptyset\}) \times Ar$ ist die Menge der Angriffe. Ähnlich zum Unterkapitel 2.1 wird $a^-$ als die Menge von Mengen abgekürzt, die $a \in Ar$ angreifen.

Neben Angriffen wird in der Arbeit auch der Begriff *Verteidigung* häufig benutzt. Die *Verteidiger* von einem beliebigen Argument $a \in Ar$ sind die Argumentenmengen, die die Angreifer von $a$ angreifen.

Das Beispiel 2.4.1 kann nun mit Hilfe der Definition 2.4.1 wie folgt modelliert werden: $AF := (Ar, att)$ mit $Ar = \{A_1, B_1, B_2\}$ und $att = \{(\{B_1, B_2\}, A_1)\}$. Die Argumente $B_1$ und $B_2$ greifen somit $A_1$ gemeinsam an. Grafisch lässt sich $AF$ durch die Abbildung 2.5 darstellen.

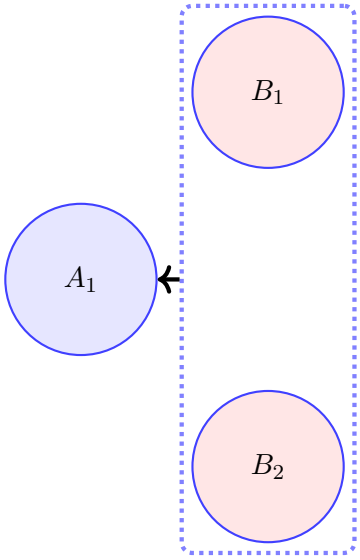



Abbildung 2.5.: Simples SetAF

## 2.5. Rangsemantiken in SetAFs

Sowohl Rangsemantiken als auch SetAFs stellen sinnvolle Erweiterungen von regulären Argumentationsframeworks dar. Nach getrennter Behandlung in den vorherigen Unterkapiteln werden sie nun kombiniert. In [YVC20] werden formale Grundlagen dafür gelegt. Die Anpassung der Definition von Rangsemantiken lautet dabei wie folgt:

**Definition 2.5.1.** Eine *Mengen-Rangsemantik* $\sigma$ ist eine Funktion, die jedes SetAF $AF = (Ar, att)$ auf eine Rangrelation $\succcurlyeq^{\sigma}_{AF}$ von $Ar$ abbildet. $\succcurlyeq^{\sigma}_{AF}$ ist eine Quasiordnung auf Argumenten. $a \succcurlyeq^{\sigma}_{AF} b$ bedeutet, dass $a$ mindestens so gültig ist wie $b$ für beliebige $a, b \in Ar$.
$\succ^{\sigma}_{AF}$ sei folgendermaßen definiert: $a \succ^{\sigma}_{AF} b :\Leftrightarrow a \succcurlyeq^{\sigma}_{AF} b \wedge b \not\succcurlyeq^{\sigma}_{AF} a$

### 2.5.1. nh-Categoriser

Als ein konkretes Beispiel einer Mengen-Rangsemantik wird in [YVC20] der sogenannte *nh-Categoriser* eingeführt. Dieser basiert auf der Rangsemantik *h-Categoriser*, die ursprünglich in [BH01] für normale Argumentationsframeworks beschrieben wurde.

**Definition 2.5.2.** $AF = (Ar, att)$ sei ein SetAF. Der *nh-Categoriser* ist eine Abbildung $C : Ar \to (0, 1]$, sodass für alle $a \in Ar$ gilt:

$$C(a) = \begin{cases} 1, & \text{wenn } a^- = \emptyset \\ \frac{1}{1+\sum_{S \in a^-} \min_{s \in S} C(s)}, & \text{sonst} \end{cases}$$

Der nh-Categoriser bildet somit Argumente aus $Ar$ für ein SetAF $AF = (Ar, att)$ auf numerische Werte zwischen 0 und 1 ab. Diese Werte entsprechen der „Wichtigkeit“ oder der Gültigkeit der Argumente. In [YVC20] wird zudem die Existenz und Eindeutigkeit der Lösung der nh-Categoriser Funktion für beliebige SetAFs bewiesen. Basierend auf diesen Voraussetzungen lässt sich nun eine Rangrelation $\succcurlyeq^{nh}_{AF}$ beschreiben – $\forall a, b \in Ar : a \succcurlyeq^{nh}_{AF} b \Leftrightarrow C(a) \geq C(b)$.

Die Berechnung der Rangbeziehungen wird am Beispiel 2.5.1 demonstriert.

**Beispiel 2.5.1.** $AF = (Ar, att)$ sei ein SetAF mit $Ar = \{A_1, A_2, A_3, B_1, B_2, B_3\}$ und $att = \{(\{A_1, A_2\}, B_3), (\{B_1, B_2, B_3\}, A_1), (\{B_1, B_2, B_3\}, A_3), (\{A_3\}, B_1)\}$.

$AF$ entspricht dem Graphen in Abbildung 2.6.

Die Auswertung vom nh-Categoriser ergibt folgende Werte:

$$C(A_2) = C(B_2) = 1,0$$
$$C(A_1) = C(A_3) = C(B_1) = C(B_3) \approx 0,618$$

Damit lassen sich folgende Rangbeziehungen feststellen:

$$A_2 \succcurlyeq^{nh}_{AF} a \text{ für alle } a \in Ar.$$
$$B_2 \succcurlyeq^{nh}_{AF} a \text{ für alle } a \in Ar.$$
$$a \succcurlyeq^{nh}_{AF} b \text{ für alle } a, b \in \{A_1, A_3, B_1, B_3\}$$

Abgesehen vom trivialen Fall von Argumenten ohne Angreifer ist die Berechnung der nh-Categoriser Werte nicht offensichtlich. Die Bewertung eines Arguments kann von den Bewertungen anderer abhängig sein, die ebenfalls unbekannt sind. Des

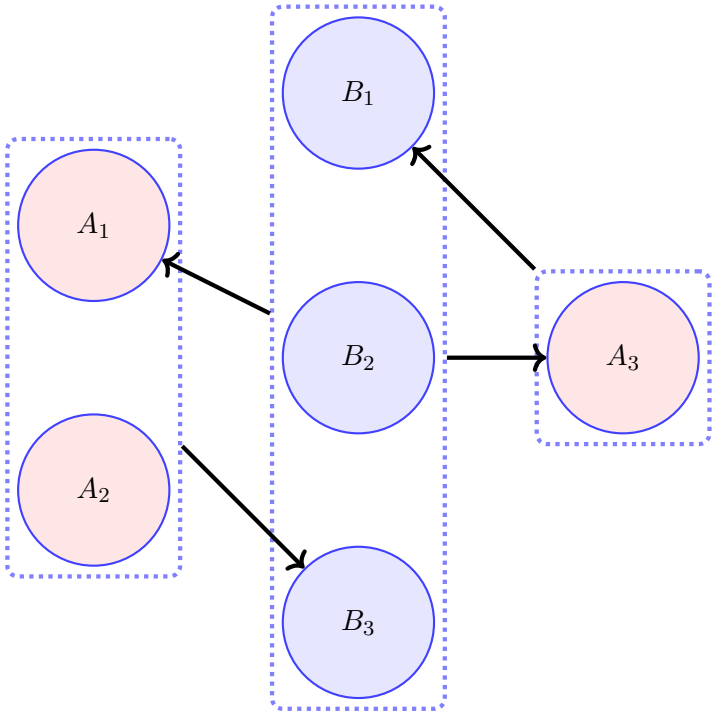



Abbildung 2.6.: Beispiel-SetAF zur nh-Categoriser Berechnung

Weiteren ist der Wert oft indirekt-rekursiv von sich selbst abhängig. So ist im Beispiel 2.5.1 $C(A_3)$ von $min(\{C(B_1), C(B_2), C(B_3)\})$ abhängig, während $C(B_1)$ von $C(A_3)$ abhängig ist.

Eine Möglichkeit, die Werte dennoch zu bestimmen, lässt sich aus dem Eindeutigkeitsbeweis der Lösung in [YVC20] ableiten. Für ein beliebiges SetAF $AF = (Ar, att)$ funktioniert das Verfahren nach dem folgenden Prinzip:

Die Bewertungen $C_0(a)$ werden mit 0 für alle Argumente $a \in Ar$ initialisiert. Auf der Basis dieser Werte werden iterativ neue Bewertungen über die nh-Categoriser Formel berechnet: $C_i(a) = \frac{1}{1+\sum_{S \in a^-} \min_{s \in S} C_{i-1}(s)}$. Laut den Autoren von [YVC20] konvergieren die Werte $C_i(a)$ gegen die Lösung $C(a)$ mit steigendem $i \in \mathbb{N}$. Im Codeausschnitt 1 wird das Verfahren in Form von Pseudocode dargestellt.

### 2.5.2. Eigenschaften von Rangsemantiken

Zur Bewertung von Rangsemantiken in regulären Argumentationsframeworks wird in der Literatur häufig auf verschiedene Eigenschaften zugegriffen. Eine Semantik kann auf die Erfüllung bzw. Nichterfüllung jeder dieser Charakteristiken geprüft werden. Dabei ist die Erfüllung eines möglichst breiten Spektrums an Eigenschaften aus praktischer Sicht erwünscht. Die im weiteren Verlauf verwendeten Eigenschaften wurden in ihrer ursprünglichen Form in den Werken [ABN13],

**Pseudocode 1** nh-Categoriser als Pseudocode

**for all** $a \in Ar$ **do**
    $C(a) \leftarrow 0$
**end for**
$stop \leftarrow false$
**while** $stop$ is $false$ **do**
    $stop \leftarrow true$
    **for all** $a \in Ar$ **do**
        $TempC(a) \leftarrow \frac{1}{1+\sum\limits_{S \in a^-} \min\limits_{s \in S} C(s)}$
        **if** $|C(a) - TempC(a)| > \epsilon$ **then**
            $stop \leftarrow false$
        **end if**
    **end for**
    $C \leftarrow TempC$
**end while**

[MT08], [CLS05] und [BDKM16] definiert.

Da die vorliegende Masterarbeit sich im Umfeld von SetAFs bewegt, sind hier Wege zur Bewertung von Mengen-Rangsemantiken von höherem Interesse. Für diese Zwecke wurde ein Satz von Merkmalen in [YVC20] eingeführt. Dieser bildet eine Analogie zu den oben genannten Eigenschaften, allerdings für die Domäne der SetAFs. Die Mengen-Rangsemantik Charakteristiken werden nachfolgend beschrieben – nach Einführung hilfreicher Begriffe und Notationen.

**Definition 2.5.3.** Die *Vereinigung* beliebiger SetAFs $AF_1 = (Ar_1, att_1)$ und $AF_2 = (Ar_2, att_2)$ wird durch das Symbol $\oplus$ ausgedrückt. $AF_1 \oplus AF_2$ wird dabei mit $(Ar_1 \cup Ar_2, att_1 \cup att_2)$ definiert.

**Definition 2.5.4.** $AF = (Ar, att)$ sei ein SetAF, $S \in \mathcal{P}(Ar) \setminus \emptyset$ und $\sigma$ eine Rangsemantik. Definiere $min$ und $max$ für Mengen von Argumenten als:

$$min(S) := \{s \in S | s' \succcurlyeq^{\sigma}_{AF} s \text{ für alle } s' \in S\}.$$
$$max(S) := \{s \in S | s \succcurlyeq^{\sigma}_{AF} s' \text{ für alle } s' \in S\}.$$

$min(S)$ ist somit das Argument mit dem kleinsten Rang in $S$. $max(S)$ besitzt entsprechend den höchsten Rang in $S$.

**Definition 2.5.5.** $AF = (Ar, att)$ sei ein SetAF, $S_1, S_2 \in \mathcal{P}(Ar) \setminus \emptyset$ und $\sigma$ eine Rangsemantik.
Definiere $\succcurlyeq^{\sigma}_{AF}$ für Argumentenmengen mit $S_1 \succcurlyeq^{\sigma}_{AF} S_2 :\Leftrightarrow s_1 \succcurlyeq^{\sigma}_{AF} s_2$ für alle $s_1 \in S_1, s_2 \in S_2$.
Analog $S_1 \succ^{\sigma}_{AF} S_2 :\Leftrightarrow s_1 \succ^{\sigma}_{AF} s_2$ für alle $s_1 \in S_1, s_2 \in S_2$.

**Definition 2.5.6.** $AF = (Ar, att)$ sei ein SetAF und $a \in Ar$. Eine Folge von Angriffen $((S_1, t_1), ..., (S_n, t_n))$ wird als *Pfad* von $S_1$ zu $t_n$ bezeichnet, wenn für alle $i \in \{1, ..., n\}$ gilt: $(S_i, t_i) \in att$ und für alle $i \in \{1, ..., n-1\}$ gilt: $t_i \in S_{i+1}$. Ein Pfad von $S$ zu $s$ wird *Zyklus* genannt, wenn $s \in S$. Ein SetAF $AF$ heißt *azyklisch*, wenn es keine zyklischen Pfade enthält.

**Definition 2.5.7.** $AF_1 = (Ar_1, att_1)$ und $AF_2 = (Ar_2, att_2)$ seien zwei SetAFs. Eine bijektive Abbildung $\gamma : Ar_1 \to Ar_2$ heißt *Isomorphismus*, wenn für alle $S \in \mathcal{P}(Ar_1), a \in Ar_1$ gilt: $(S, a) \in att_1 \Leftrightarrow (\{\gamma(s) \mid s \in S\}, \gamma(a)) \in att_2$.

Ein Isomorphismus existiert zwischen zwei SetAFs nur dann, wenn sie eine identische Knoten- und Angriffsstruktur besitzen.

**Definition 2.5.8.** $AF = (Ar, att)$ sei ein SetAF und $a \in Ar$. Die Verteidigung von $a$ heißt *simpel*, wenn jeder Verteidiger $V$ von $a$ genau einen Angreifer $A \in a^-$ angreift.

**Definition 2.5.9.** $AF = (Ar, att)$ sei ein SetAF und $a \in Ar$. Die Verteidigung von $a$ wird als *verteilt* bezeichnet, wenn für jeden Angreifer $A \in a^-$ maximal ein Verteidiger $V$ existiert, der $A$ angreift.

**Beispiel 2.5.2.** Zur Illustration der simplen und verteilten Verteidigung werden hier drei SetAFs gezeigt. $AF_1 = (Ar_1, att_1)$ sei ein SetAF mit $Ar_1 = \{A, B_1, B_2, C_1, C_2\}$ und $att = \{(\{B_1\}, A), (\{B_2\}, A), (\{C_1\}, B_1), (\{C_2\}, B_2)\}$ (S. Abb. 2.7a). Die Verteidiger von $A$ bilden $C_1$ und $C_2$. Beide Verteidiger greifen genau einen Angreifer von $A$ an. $A$ verfügt somit über eine simple Verteidigung. Sowohl $B_1$ als auch $B_2$ haben jeweils nur einen Angreifer. Das bedeutet, dass die Verteidigung von $A$ auch verteilt ist.

$AF_2 = (Ar_2, att_2)$ sei ein SetAF mit $Ar_2 = \{A, B_1, B_2, C\}$ und $att_2 = \{(\{B_1\}, A), (\{B_2\}, A), (\{C\}, B_1), (\{C\}, B_2)\}$ (S. Abb. 2.7b). Der Verteidiger von $A$ in $AF_2$ ist $C$, wobei $C$ zwei unterschiedliche Angreifer von $A$ angreift. Damit ist die Verteidigung von $A$ nicht länger simpel, allerdings immer noch verteilt, da $B_1$ und $B_2$ weiter nur jeweils einen Angreifer haben.

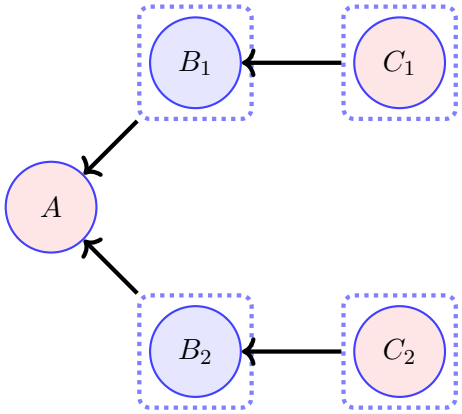
$AF_3 = (Ar_3, att_3)$ sei ein SetAF mit $Ar_3 = \{A, B_1, B_2, C_1, C_2\}$ und $att_3 = \{(\{B_1\}, A), (\{B_2\}, A), (\{C_1\}, B_1), (\{C_2\}, B_1)\}$ (S. Abb. 2.7c). Die Verteidigung von $A$ ist in $AF_3$ simpel, denn die Verteidiger greifen jeweils nur einen Angreifer von $A$ an. Sie ist jedoch nicht verteilt, denn $A$ wird vor $\{B_1\}$ durch zwei Argumente verteidigt – $C_1$ und $C_2$.

**Definition 2.5.10.** $AF = (Ar, att)$ sei ein SetAF und $a \in Ar$. $P_+(a) = (Ar_0, att_0)$ heißt *Verteidigungserweiterung* von $a$, wenn folgendes gilt:

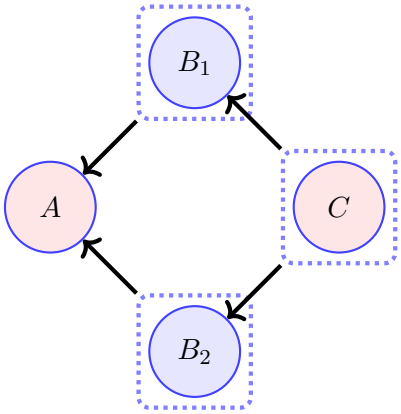
- $Ar_0 = \{a, x_1 \dots x_k\}$ mit $Ar \cap A_0 = a$.
- Es gibt eine gerade Zahl $n \in \mathbb{N}$ und ein $S_1 \in \mathcal{P}(Ar_0) \setminus \emptyset$ mit einem azyklischen Pfad $((S_1, t_1) \dots (S_n, t_n))$ der Länge $n$ von $S_1$ nach $t_n = a$.
- $att_0 = \{(S_1, t_1) \dots (S_n, t_n)\}$ und $S_1 \cup S_2 \cup \dots \cup S_n \cup \{a\} = A_0$.

In anderen Worten entsteht eine Verteidigungserweiterung von $a \in Ar$ durch die Vereinigung von $AF$ mit einem SetAF $P_+(a)$, wobei beide SetAFs abgesehen von $a$ disjunkt sind. $P_+(a)$ entspricht dabei lediglich einem azyklischen Pfad gerader Länge von $S \in \mathcal{P}(Ar_0)$ nach $a$.

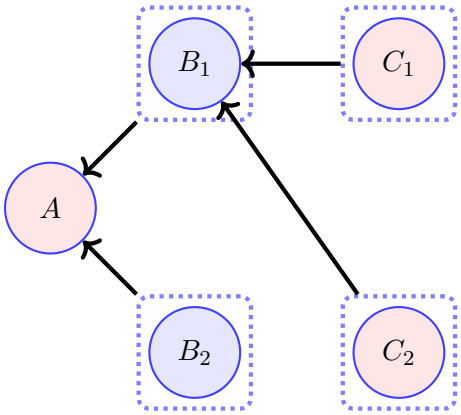
Die *Angriffserweiterung* $P_-(a)$ ist analog definiert mit dem Unterschied, dass die Pfadlänge $n$ ungerade ist.




(a) $A$ simpel und verteilt verteidigt




(b) $A$ nicht simpel und verteilt verteidigt




(c) $A$ simpel und nicht verteilt verteidigt

Abbildung 2.7.: Simple und verteilte Verteidigung

**Definition 2.5.11.** $AF = (Ar, att)$ sei ein SetAF, $S_1, S_2 \subseteq \mathcal{P}(Ar) \setminus \emptyset$ und $\sigma$ eine Rangsemantik. Die Relation $\geq$ für Mengen von Argumentenmengen wird definiert mit: $S_1 \geq S_2 :\Leftrightarrow$ es gibt eine injektive Abbildung $f : S_2 \to S_1$, sodass $min(f(s)) \succcurlyeq^{\sigma}_{AF} min(s)$ für alle $s \in S_2$.

Analog definiere $S_1 > S_2 :\Leftrightarrow$ es gibt eine injektive Abbildung $f : S_2 \to S_1$, sodass $min(f(s)) \succ^{\sigma}_{AF} min(s)$ für alle $s \in S_2$.

Beide Relationen werden weiter für den Vergleich von Angriffen auf unterschiedliche Argumente verwendet. Für beliebige Argumente $a$ und $b$ bedeutet $a^- \geq b^-$, dass für jeden Angreifer $B \in b^-$ paarweise ein $A \in a^-$ existiert, wobei das schwächste Argument in $A$ gleichen oder höheren Rang besitzt als das schwächste Argument in $B$.

Aufbauend auf den obigen Definitionen werden weiter die Eigenschaften für Rangsemantiken eingeführt.

- **Abstraktion** (orig. *Abstraction*)

  Eine Rangsemantik $\sigma$ erfüllt die Eigenschaft *Abstraktion*, wenn für beliebige SetAFs $AF_1 = (Ar_1, att_1)$, $AF_2 = (Ar_2, att_2)$ und beliebige Isomorphismen $\gamma$ mit $\gamma(Ar_1) = Ar_2$ gilt: $a \succcurlyeq^{\sigma}_{AF_1} b \Leftrightarrow \gamma(a) \succcurlyeq^{\sigma}_{AF_2} \gamma(b)$.

  Bei einer die Abstraktion erfüllenden Semantik ist sichergestellt, dass ein SetAF nur anhand der Angriffsrelationen ausgewertet und von dem Namen und Inhalt der Argumente abstrahiert wird.

- **Unabhängigkeit** (orig. *Independence*)

  Eine Rangsemantik $\sigma$ erfüllt *Unabhängigkeit*, wenn für beliebige SetAFs $AF_1 = (Ar_1, att_1), AF_2 = (Ar_2, att_2)$ mit $Ar_1 \cap Ar_2 = \emptyset$ und $a, b \in Ar_1$ gilt $a \succcurlyeq^{\sigma}_{AF_1} b \Leftrightarrow a \succcurlyeq^{\sigma}_{AF_1 \oplus AF_2} b$

  Die Erfüllung der Unabhängigkeit garantiert, dass der relative Rang zwischen zwei Argumenten $a$ und $b$ nicht von einem dritten Argument $c$ abhängt, wenn $c$ nicht über einen Pfad mit $a$ oder mit $b$ verbunden ist.

- **Void-Vorrang** (orig. *Void Precedence*)

  Eine Rangsemantik $\sigma$ erfüllt die Eigenschaft *Void-Vorrang*, wenn für jedes SetAF $AF = (Ar, att)$ und $a, b \in Ar$ mit $a^- = \emptyset, b^- \neq \emptyset$ gilt: $a \succ^{\sigma}_{AF} b$.

  Void-Vorrang gilt, wenn jedes Argument ohne Angreifer einen höheren Rang besitzt als jedes Argument mit mindestens einem Angreifer.

- **Widersprüchlichkeit** (orig. *Self-Contradiction*)

  $AF = (Ar, att)$ sei ein beliebiges SetAF. Eine Rangsemantik $\sigma$ erfüllt *Widersprüchlichkeit*, wenn für beliebige $a, b \in Ar$ folgendes gilt: $\exists S_1 \in a^-$ mit $a \in S_1$ und $\nexists S_2 \in b^-$ mit $b \in S_2 \Rightarrow b \succ^{\sigma}_{AF} a$.

  Die Widersprüchlichkeit erfüllende Semantiken bewerten sich selbt angreifende Argumente mit einem schwächeren Rang als Argumente ohne Selbstangriffe.

- **Kardinalität-Vorrang** (orig. *Cardinality Precedence*)

  $AF = (Ar, att)$ sei ein SetAF und $a, b \in Ar$. Eine Rangsemantik $\sigma$ erfüllt den *Kardinalität-Vorrang*, wenn gilt: $|a^-| < |b^-| \Rightarrow a \succ^{\sigma}_{AF} b$.

Diese Eigenschaft bezieht sich ausschließlich auf die Anzahl der Angreifer. Ein beliebiges Argument $a$ mit weniger Angreifern als ein Argument $b$ hat somit einen höheren Rang bei der Gültigkeit dieser Eigenschaft.

Dabei fällt auf, dass der Kardinalität-Vorrang eine Verallgemeinerung der zuvor eingeführten Eigenschaft „Void-Vorrang“ darstellt. Das bedeutet, dass bei Geltung des Kardinalität-Vorrangs auch der Void-Vorrang erfüllt ist. Umgekehrt impliziert die Nichterfüllung vom Void-Vorrang die Nichterfüllung vom Kardinalität-Vorrang.

- **Verteidigungsvorrang** (orig. *Defense Precedence)*

  $AF = (Ar, att)$ sei ein SetAF und $a, b \in Ar$ mit $|a^-| = |b^-|$. Weiter sei $A$ die Menge der Verteidiger von $a$ und $B$ die Menge der Verteidiger von $b$. Eine Rangsemantik $\sigma$ erfüllt *Verteidigungsvorrang*, falls folgendes gilt: $A = \emptyset, B \neq \emptyset \Rightarrow b \succ^{\sigma}_{AF} a$.

  Bei der Gültigkeit des Verteidigungsvorrangs verfügt ein Argument $a$ über einen höheren Rang als $b$, falls $a$ verteidigt ist und $b$ nicht, wenn die Anzahl ihrer Angreifer gleich ist.

- **Totalität** (orig. *Total*)

  $AF = (Ar, att)$ sei ein SetAF. Die *Totalität* gilt für eine Rangsemantik $\sigma$, wenn für alle $a, b \in Ar$ gilt: $a \succcurlyeq^{\sigma}_{AF} b$ oder $b \succ^{\sigma}_{AF} a$.

  Durch Totalität wird sichergestellt, dass die Rangrelation auf allen Argumenten eines SetAF anwendbar ist.

- **Kein-Angriff-Äquivalenz** (orig. *Non-Attacked Equivalence*)

  $AF = (Ar, att)$ sei ein SetAF und $a, b \in Ar$ beliebig mit $a^- = \emptyset$ und $b^- = \emptyset$. Die *Kein-Angriff-Äquivalenz* gilt für eine Rangsemantik $\sigma$ genau dann, wenn $a \succcurlyeq^{\sigma}_{AF} b \wedge b \succcurlyeq^{\sigma}_{AF} a$.

  Diese Eigenschaft gilt, wenn sämtliche Argumente ohne Angreifer den gleichen Rang besitzen.

- **Vorrang der verteilten Verteidigung** (orig. *Distributed-Defense Precedence*)

  $AF = (Ar, att)$ sei ein SetAF und $a, b \in Ar$ beliebig, wobei $a$ die gleiche Anzahl von Angreifern und Verteidigern hat wie $b$. Die Eigenschaft *Vorrang der verteilten Verteidigung* für eine Rangsemantik $\sigma$ ist genau dann erfüllt, wenn folgendes zutrifft:
  Hat $a$ eine simple und verteilte Verteidigung und $b$ eine simple und nicht verteilte Verteidigung, dann gilt $a \succ^{\sigma}_{AF} b$.

  Die Sinnhaftigkeit dieser Eigenschaft kann an dem Beispiel 2.5.2 verdeutlicht werden. Das Argument $A$ hat zwei Verteidiger sowohl in $AF_1$ als auch in $AF_3$. Dennoch wird $A$ in $AF_3$ gegen nur einen Angreifer – $B_1$ – verteidigt, während in $AF_1$ beide Angreifer „abgewehrt“ werden. Die obige Eigenschaft belohnt somit eine Verteidigung mit weniger Redundanz.

- **Vorrang strikter Verteidigungserweiterung** (orig. *Strict Addition of Defense Branch*)

  $AF_1 = (Ar_1, att_1), AF_2 = (Ar_2, att_2)$ seien beliebige SetAFs, sodass ein Isomorphismus $\gamma$ mit $\gamma(Ar_1) = Ar_2$ existiert. $a \in Ar_1$ sei ein beliebiges Argument und $P_+(a)$ eine beliebige Verteidigungserweiterung von $a$. Die Eigenschaft *Vorrang strikter Verteidigungserweiterung* gilt für eine Rangsemantik $\sigma$ genau dann, wenn $a \succ^{\sigma}_{AF_1 \oplus AF_2 \oplus P_+(a)} \gamma(a)$.

  Die Erfüllung dieser Eigenschaft impliziert, dass ein Argument mit einer Verteidigungserweiterung einen höheren Rang erhält als ein identisches Argument ohne die Verteidigungserweiterung.

- **Vorrang der Verteidigungserweiterung** (orig. *Addition of Defense Branch*)

  $AF_1 = (Ar_1, att_1), AF_2 = (Ar_2, att_2)$ seien beliebige SetAFs, sodass ein Isomorphismus $\gamma$ mit $\gamma(Ar_1) = Ar_2$ existiert. $a \in Ar_1$ mit $a^- \neq \emptyset$ sei ein beliebiges Argument und $P_+(a)$ eine beliebige Verteidigungserweiterung von $a$. Die Eigenschaft *Vorrang der Verteidigungserweiterung* gilt für eine Rangsemantik $\sigma$ genau dann, wenn $a \succ^{\sigma}_{AF_1 \oplus AF_2 \oplus P_+(a)} \gamma(a)$.

  Die Eigenschaft ähnelt dem Vorrang strikter Verteidigungserweiterung. Sie ist allerdings auf die angegriffenen Argumente beschränkt. Es ist leicht ersichtlich, dass der Vorrang der Verteidigungserweiterung durch die Gültigkeit vom Vorrang strikter Verteidigungserweiterung impliziert wird, jedoch nicht umgekehrt.

- **Abwertung der Angriffserweiterung** (orig. *Addition of Attack Branch*)

  $AF_1 = (Ar_1, att_1), AF_2 = (Ar_2, att_2)$ seien beliebige SetAFs, sodass ein Isomorphismus $\gamma$ mit $\gamma(Ar_1) = Ar_2$ existiert. $a \in Ar_1$ sei ein beliebiges Argument und $P_-(a)$ eine beliebige Angriffserweiterung von $a$. Die Eigenschaft *Abwertung der Angriffserweiterung* gilt für eine Rangsemantik $\sigma$ genau dann, wenn $a \succ^{\sigma}_{AF_1 \oplus AF_2 \oplus P_-(\gamma(a))} \gamma(a)$.

  Bei der Eigenschaft handelt es sich um die „Spiegelung“ vom Vorrang strikter Verteidigungserweiterung.

- **Verlängerte Angriffserweiterung** (orig. *Increase of Attack Branch*)

  $AF_1 = (Ar_1, att_1), AF_2 = (Ar_2, att_2)$ seien beliebige SetAFs, wobei ein Isomorphismus $\gamma$ mit $\gamma(Ar_1) = Ar_2$ existiert. Die *verlängerte Angriffserweiterung* gilt für eine Rangsemantik $\sigma$ genau dann, wenn für alle $a \in Ar_1, S \in \mathcal{P}(Ar_1) \setminus \emptyset$ beliebig, sodass:

  - $s^- = \emptyset$ für alle $s \in S$.
  - Es existiert ein Pfad der Länge $n > 0$ von $S$ nach $a$, wobei $n$ ungerade ist.
  - Es existiert kein Pfad der Länge $m > 0$ von $S$ nach $a$, wobei $m$ gerade ist.

folgendes gilt: $a \succ^{\sigma}_{AF_1 \oplus AF_2 \oplus P_+(s)} \gamma(a)$ für alle $s \in S$ mit beliebigen Verteidigungserweiterungen $P_+(s)$ von $s$.

Informell ausgedrückt erhält ein Argument $a$ eine höhere Bewertung als das dazu isomorphe Argument $\gamma(a)$, wenn ein Angriff auf $a$ verlängert wird. Diese Eigenschaft bewirkt somit eine Abschwächung des Angriffs bei dessen Verlängerung.

- **Verlängerte Verteidigungserweiterung** (orig. *Increase of Defense Branch*)

  $AF_1 = (Ar_1, att_1), AF_2 = (Ar_2, att_2)$ seien beliebige SetAFs, wobei ein Isomorphismus $\gamma$ mit $\gamma(Ar_1) = Ar_2$ existiert. Die *verlängerte Verteidigungserweiterung* gilt für eine Rangsemantik $\sigma$ genau dann, wenn für alle $a \in Ar_1, S \in \mathcal{P}(Ar_1) \setminus \emptyset$ beliebig, sodass:

  - $s^- = \emptyset$ für alle $s \in S$.
  - Es existiert ein Pfad der Länge $n > 0$ von $S$ nach $a$, wobei $n$ gerade ist.
  - Es existiert kein Pfad der Länge $m > 0$ von $S$ nach $a$, wobei $m$ ungerade ist.

  folgendes gilt: $a \succ^{\sigma}_{AF_1 \oplus AF_2 \oplus P_+(\gamma(s))} \gamma(a)$ für alle $s \in S$ mit beliebigen Verteidigungserweiterungen $P_+(\gamma(s))$ von $\gamma(s)$.

  Wenn die verlängerte Verteidigungserweiterung gilt, resultiert die Verlängerung eines Verteidigungspfades in einer Abschwächung der Verteidigungswirkung.

- **Vorrang der vollen Verteidigung** (orig. *Attack vs Full Defense*)

  $AF = (Ar, att)$ sei ein beliebiges azyklisches SetAF. Der *Vorrang der vollen Verteidigung* gilt für eine Rangsemantik $\sigma$ genau dann, wenn für alle $a, b \in Ar$ mit:

  - $b^- = \{B\}$ mit $B \in \mathcal{P}(Ar) \setminus \emptyset$.
  - $x^- = \emptyset$ für alle $x \in B$.
  - Es gibt keine Menge $S \in \mathcal{P}(Ar) \setminus \emptyset$ mit $s^- = \emptyset \; \forall s \in S$, sodass ein Pfad ungerader Länge von $S$ nach $a$ existiert.

  gilt: $a \succ^{\sigma}_{AF} b$.

  Durch diese Eigenschaft wird sichergestellt, dass Argumente mit nur abgewehrten Angriffen einen höheren Rang haben als Argumente mit einem nichtabgewehrten direkten Angreifer.

- **Qualitätsvorrang** (orig. *Quality Precedence*)

  $AF = (Ar, att)$ sei ein SetAF, $a, b \in Ar$. Weiter seien $A_1 \dots A_n$ die Angreifer von $a$ und $B_1 \dots B_m$ die Angreifer von $b$. Der *Qualitätsvorrang* gilt für eine Rangsemantik $\sigma$ genau dann, wenn folgendes gilt:
  $max(\{min(B_1) \dots min(B_m)\}) \succ^{\sigma}_{AF} max(\{min(A_1) \dots min(A_n)\}) \Rightarrow a \succ^{\sigma}_{AF} b$.

Hier wird die „Kraft“ der Angreifermengen mit dem Rang ihres schwächsten Arguments gleichgesetzt. Die Eigenschaft gilt somit, wenn ein Argument $a$ einen höheren Rang als ein Argument $b$ erhält, falls die stärkste Angreifermenge von $a$ schwächer ist als die stärkste Angreifermenge von $b$.

- **Gegentransitivität** (orig. *Counter-Transitivity*)

  $AF = (Ar, att)$ sei ein beliebiges SetAF und $\sigma$ eine Rangsemantik. $\sigma$ erfüllt die *Gegentransitivität*, wenn für alle $a, b \in Ar$ gilt: $a^- \geq b^- \Rightarrow b \succcurlyeq^{\sigma}_{AF} a$.

  Durch die obige Eigenschaft wird sichergestellt, dass Angriffe durch „stärkere“ Mengen sich negativer auf den Rang des angegriffenen Arguments auswirken als Angriffe durch schwächere Mengen.

- **Strikte Gegentransitivität** (orig. *Strict Counter-Transitivity*)

  $AF = (Ar, att)$ sei ein beliebiges SetAF und $\sigma$ eine Rangsemantik. $\sigma$ erfüllt die *strikte Gegentransitivität*, wenn für alle $a, b \in Ar$ gilt: $a^- > b^- \Rightarrow b \succ^{\sigma}_{AF} a$.

  Diese Eigenschaft ist analog zur Gegentransitivität, wobei hier eine strikte Ungleichheit der Angriffskraft in einem strikt unterschiedlichen Rang der angegriffenen Argumente resultiert.

Der beschriebene Satz von Merkmalen wird in den nachfolgenden Kapiteln 3 und 4 tiefer ausgeleuchtet und mit Bezug auf konkrete Mengen-Rangsemantiken untersucht.

# 3. Belastungssemantik in SetAFs

Die Arbeit [YVC20] liefert interessante Ergebnisse bei der Analyse von Rangsemantiken im Kontext von SetAFs. Die Autoren konzentrierten sich dabei auf die Erweiterung der h-Categoriser Semantik und bauten diese auf Kompatibilität zu SetAFs aus. Zusätzlich wurde die Erweiterung auf die Lösbarkeit sowie die Erfüllung der Semantikeigenschaften untersucht.

Als Fortsetzung der genannten Arbeit ist eine analoge Erweiterung und Analyse von anderen Rangsemantiken zielführend. Das ist eins der Ziele der vorliegenden Masterarbeit. Für die Wahl der zu erweiternden Semantik wird die Arbeit [BDKM16] verwendet. Darin werden mehrere häufig in der Literatur referenzierte Rangsemantiken vorgestellt und miteinander verglichen. Im Vergleich fällt die Belastungssemantik auf, die bereits in Unterkapitel 2.3.1 beschrieben wurde. Sie erfüllt mehr Semantikeigenschaften als die anderen verglichenen Semantiken, insbesondere erfüllt sie alle vom h-Categoriser erfüllten Eigenschaften sowie zusätzlich den Kardinalität-Vorrang und den Vorrang der verteilten Verteidigung.

Mit dieser Begründung wird die Belastungssemantik in diesem Kapitel für die Erweiterung auf SetAFs gewählt. Nach der Definitionsanpassung wird sie zusätzlich auf die Erfüllung der Semantik-Eigenschaften geprüft.

## 3.1. Definition

Bei der regulären Belastungssemantik ist die Belastung eines Arguments $a$ von der Anzahl und der Belastung der Angreifer von $a$ abhängig. Da die Angreifer hierbei einzelne Argumente sind, sind ihre Belastungen klar definiert. Im Bereich der SetAFs sind Angreifer allerdings Argumentenmengen. Jedes Element solcher Mengen hat eine eigene Belastung, wodurch für die Belastungsbestimmung der gesamten Menge unterschiedliche Methoden denkbar sind. Als Beispiel kann der Durchschnitt der Belastungen aller Mengenelemente als Belastungswert der Menge dienen. Auch die Summe oder das Produkt der Belastungen ist für die Berechnung denkbar.

Zu beachten ist allerdings, dass ein Angriff nur von der Gesamtheit der Angreifermenge ausgeht. Die Bewertung der stärkeren Argumente ist somit irrelevant, wenn das schwächste Argument nicht tragbar ist. Deswegen ist es im Kontext von SetAFs angebracht, eine angreifende Menge anhand ihres schwächsten Elements zu bewerten. Diese Vorgehensweise war ebenso am Beispiel des nh-Categorisers ersichtlich. Bei diesem wird die „Kraft“ einer Angreifermenge mit dem Element der niedrigsten Bewertung gleichgesetzt.

Mit dieser Begründung wird auch in der vorliegenden Arbeit auf diese Weise verfahren. Die Belastungssemantik wird so erweitert, dass die Belastung eines Arguments $a$ von den Belastungen der schwächsten Elemente der Angreifer in $a^-$ abhängt. Dabei ist zu beachten, dass eine hohe Belastung sich negativ auf den Rang eines Arguments auswirkt. Damit ist das schwächste Argument einer Menge das mit der höchsten Belastung.

**Definition 3.1.1.** $AF = (Ar, att)$ sei ein beliebiges SetAF. Die *Mengenbelastung* $\mathcal{B}_i(a)$ wird für alle $a \in Ar$ und $i \in \mathbb{N}_0$ wie folgt definiert:

$$\mathcal{B}_i(a) = \begin{cases} 1, & \text{falls } i = 0 \\ 1, & \text{falls } a^- = \emptyset \\ 1 + \sum\limits_{S \in a^-} \frac{1}{\max\limits_{s \in S}(\mathcal{B}_{i-1}(s))}, & \text{sonst} \end{cases}$$

Die Definition bestimmt für die Initialisierung somit $\mathcal{B}_0(a) = 1$ für beliebige Argumente $a$. Es sei an dieser Stelle erwähnt, dass sich daraus $\mathcal{B}_1(a) = 1 + |a^-|$ folgern lässt. Diese Vereinfachung wird in der späteren Analyse mehrfach benutzt.

Die obige Berechnung der Mengenbelastung ist im Vergleich zur normalen Belastung komplexer. Identisch bleibt dagegen die Definition des darauf basierenden Rangs.

**Definition 3.1.2.** $AF = (Ar, att)$ sei ein SetAF und $a, b \in Ar$. Der *Mengen-Belastungsrang* $\succcurlyeq_{\mathcal{B}}$ definiert sich über die Belastungswerte wie folgt:
$a \succcurlyeq_{\mathcal{B}} b$ gilt genau dann, wenn

- $\mathcal{B}_i(a) = \mathcal{B}_i(b)\ \forall i \in \mathbb{N}_0$ **oder**
- $\exists i \in \mathbb{N}, \mathcal{B}_i(a) < \mathcal{B}_i(b)$ *und* $\forall j \in \{0, 1 \dots i-1\}, \mathcal{B}_j(a) = \mathcal{B}_j(b)$.

Weiter gelte $a \succ_{\mathcal{B}} b$ genau dann, wenn $a \succcurlyeq_{\mathcal{B}} b$ gilt *und* $b \succcurlyeq_{\mathcal{B}} a$ nicht gilt.

**Definition 3.1.3.** Mit dem beschriebenen Rang $\succcurlyeq_{\mathcal{B}}$ lässt sich nun die eigentliche Mengen-Rangsemantik $\mathcal{B}$ einführen. Der Definition 2.5.1 folgend ist $\mathcal{B}$ eine Funktion, die jedes SetAF $AF$ auf $\succcurlyeq_{\mathcal{B}}$ abbildet.

## 3.2. Existenz und Eindeutigkeit der Lösung

Für die Lösung vom nh-Categoriser wird in [YVC20] die Existenz und die Eindeutigkeit gezeigt. Im Fall der Mengen-Belastungssemantik liegt die Gültigkeit von beidem wegen der iterativen Berechnungsart nahe.

Weiter in diesem Kapitel wird die Gültigkeit der Eigenschaft *Totalität* für $\mathcal{B}$ bewiesen. Die Totalität besagt, dass für ein beliebiges SetAF $AF = (Ar, att)$ und alle Argumente $a, b \in Ar$ entweder $a \succcurlyeq_{\mathcal{B}} b$ oder $b \succ_{\mathcal{B}} a$ gilt. Das impliziert direkt die Existenz einer Lösung für $\mathcal{B}$.

Die Eindeutigkeit lässt sich aus der Eindeutigkeit der Belastungen schlussfolgern.

## 3.3. Erfüllung von Semantikeigenschaften

Die oben eingeführte Modifikation der Belastungssemantik wird nun auf die Erfüllung der Eigenschaften aus Kapitel 2.5.2 analysiert. Die besprochenen Eigenschaften sind für SetAFs konzipiert. Wie zuvor im Grundlagenkapitel erwähnt, sind ihre Definitionen aus analogen Merkmalen für reguläre Rangsemantiken entstanden. Der Satz dieser ursprünglichen Eigenschaften wurde in der Literatur unter anderem in der Dissertation [Del17] untersucht. Darin werden mehrere Zusammenhänge zwischen den unterschiedlichen Eigenschaften erklärt. Es wird gezeigt, dass manche Eigenschaften die Erfüllung von bestimmten anderen implizieren. Umgekehrt sind mehrere Eigenschaften inkompatibel und können nicht gleichzeitig für eine Semantik zutreffen.

Auch wenn die SetAF-Eigenschaften als eine Analogie zu den Merkmalen für normale Argumentationsframeworks eingeführt wurden, unterscheidet sich ihre mathematische Definition. Aus diesem Grund ist die Gültigkeit der Schlussfolgerungen aus [Del17] nicht im SetAF-Kontext garantiert. Für die Analyse in diesem Kapitel werden diese Ergebnisse daher nicht verwendet. Die Eigenschaftenerfüllung wird unabhängig davon untersucht.

### 3.3.1. Übersicht

Die Tabelle 3.1 zeigt eine Zusammenfassung der Eigenschaftenerfüllung. Als Vergleich wird der nh-Categoriser daneben dargestellt, wobei die entsprechende Information zu den meisten Eigenschaften aus der Arbeit [YVC20] entnommen wurde. Für die Eigenschaften *verlängerte Angriffserweiterung* und *verlängerte Verteidigungserweiterung* wird in dem genannten Werk die Gültigkeit besagt. Diese Aussage ist nicht mit der Analyse in der vorliegenden Arbeit kompatibel. Das Gegenbeispiel 3.3.2, mit dem die Geltung der Eigenschaften für die Mengen-Belastungssemantik widerlegt wird, kann für die gleichen Zwecke beim nh-Categoriser benutzt werden. Eine ausführliche Begründung wird in dem entsprechenden Beweis in Unterkapitel 3.3.3 angegeben.

### 3.3.2. Diskussion der Ergebnisse

Wie in der Kapiteleinführung erwähnt, wurde auf die Nutzung der Eigenschaftenregeln aus [Del17] verzichtet, da ihre Gültigkeit nicht angenommen werden kann. Dennoch ist erwähnenswert, dass die gezeigten Endergebnisse mit diesen Regeln kompatibel sind.

In der Arbeit [BDKM16] werden mehrere Rangsemantiken in Bezug auf die Erfüllung von Semantik-Eigenschaften analysiert. Diese Arbeit bewegt sich ebenfalls im Umfeld von regulären Argumentationsframeworks. Dies erlaubt eine Gegenüberstellung der ursprünglichen Rangsemantiken (h-Categoriser, Belastungssemantik) und ihrer SetAF-Erweiterungen (nh-Categoriser, Mengen-Belastungsemantik).

| **Mengen-Rangsemantik Eigenschaft** | $\mathcal{B}$ | $nh$ |
|---|---|---|
| Abstraktion | ✓ | ✓ |
| Unabhängigkeit | ✓ | ✓ |
| Void-Vorrang | ✓ | ✓ |
| Widersprüchlichkeit | ✗ | ✗ |
| Kardinalität-Vorrang | ✓ | ✗ |
| Verteidigungsvorrang | ✓ | ✓ |
| Totalität | ✓ | ✓ |
| Kein-Angriff-Äquivalenz | ✓ | ✓ |
| Vorrang der verteilten Verteidigung | ✓ | ✗ |
| Vorrang strikter Verteidigungserweiterung | ✗ | ✗ |
| Vorrang der Verteidigungserweiterung | ✗ | ✗ |
| Abwertung der Angriffserweiterung | ✓ | ✓ |
| Verlängerte Angriffserweiterung | ✗ | ✗ |
| Verlängerte Verteidigungserweiterung | ✗ | ✗ |
| Vorrang der vollen Verteidigung | ✗ | ✗ |
| Qualitätsvorrang | ✗ | ✗ |
| Gegentransitivität | ✓ | ✓ |
| Strikte Gegentransitivität | ✓ | ✓ |

Tabelle 3.1.: Mengen-Belastungssemantik und nh-Categoriser – Eigenschaften

Die ursprüngliche Belastungssemantik erfüllt alle Eigenschaften, die von $\mathcal{B}$ erfüllt werden. Ein analoges Bild ergibt sich für den h-Categoriser. Zusätzlich erfüllen die beiden „Originale“ aber die verlängerte Angriffs- und Verteidigungserweiterung.

Diese Abweichung ist auf einen bedeutenden Unterschied bei dem Begriff vom Pfad zurückzuführen. Ein Pfad von $b$ nach $a$ in einem Argumentationsframework trägt stets zu einer Veränderung der Bewertung von $a$ bei. In einem SetAF ist das nicht zwangsweise der Fall. Da die Angreifer hier Argumentenmengen entsprechen, kann ein Pfad über ein Element $s$ einer Menge $S$ laufen, wobei $s$ nicht das schwächste Argument in $S$ ist. Sowohl beim nh-Categoriser als auch bei der Mengen-Belastungssemantik führt das dazu, dass dieser Pfad „ausgeschaltet“ wird und nicht zu der Berechnung der Bewertung von $a$ beiträgt. Ein konkretes Beispiel 3.3.2 wird weiter in der Widerlegung der verlängerten Angriffserweiterung demonstriert.

Der Berechnung der Mengen-Belastungssemantik liegt ein ähnliches Prinzip zur Berechnung der nh-Categoriser Bewertungen zugrunde. Für jedes Argument $a$ summieren beide iterativ das Inverse der Angreiferbewertungen von $a$ auf. Die Unterschiede können in drei Punkten zusammengefasst werden:

- Die Werte vom nh-Categoriser sind auf $(0, 1]$ normiert. Die Belastungswerte $\mathcal{B}_i$ für ein Argument $a$ liegen dagegen zwischen $1$ in der Initialisierungsphase und $1 + |a^-|$ in der Iteration $1$.

- Die Summe der Angreiferbewertungen steht beim nh-Categoriser vollständig im Nenner und verhält sich damit antiproportional zu der entsprechenden Summe von $\mathcal{B}_i$. Das erklärt die Tatsache, dass beim nh-Categoriser eine größere Bewertungszahl tendenziell in einem besseren Rang resultiert, gegensätzlich zu $\mathcal{B}_i$.

- Der nh-Categoriser (in seiner algorithmischen Darstellung) berechnet die Bewertungen aller Argumente bis zur Konvergenz und bestimmt die Ränge anhand der nh-Werte nach der letzten Iteration. Die Semantik $\mathcal{B}$ bestimmt dagegen die Relation zwischen zwei Argumenten $a$ und $b$ noch vor der Konvergenz sämtlicher Belastungen, wenn sich die Belastungen der beiden Argumente vorher unterscheiden.

Die ersten beiden Punkte sind oberflächlicher Natur und verursachen keine Unterschiede in dem letztendlichen Rang. Der dritte Punkt ist dagegen ausschlaggebend für die Abweichungen zwischen den beiden Semantiken und trägt dazu bei, dass $\mathcal{B}$ im Unterschied zum nh-Categoriser den Kardinalität-Vorrang und den Vorrang der verteilten Verteidigung erfüllt.

### 3.3.3. Beweise

**Abstraktion** Die Eigenschaft ist erfüllt.

Das ist naheliegend, da die Berechnung der Belastungswerte ausschließlich durch die Angriffsbeziehungen bestimmt wird. Zur Vollständigkeit wird dennoch ein Induktionsbeweis angegeben.

*Beweis.* $AF_1 = (Ar_1, att_1), AF_2 = (Ar_2, att_2)$ seien zwei SetAFs, sodass ein Isomorphismus $\gamma : Ar_1 \to Ar_2$ existiert. Zu zeigen ist, dass $\mathcal{B}_i(a) = \mathcal{B}_i(\gamma(a))$ für alle $i$ in $\mathbb{N}_0$ und alle $a \in Ar_1$.
**Induktionsanfang,** $i = 0$: Durch die Definition der Belastungszahlen ist gegeben, dass $\mathcal{B}_0(a) = \mathcal{B}_0(\gamma(a)) = 1$ für alle $a \in Ar_1$.
**Induktionsvoraussetzung**: Die Bedingung $\mathcal{B}_i(a) = \mathcal{B}_i(\gamma(a))$ gelte für alle $i \leq n$ für ein $n \in \mathbb{N}_0$ und alle $a \in Ar_1$.
**Induktionsschritt,** $\mathcal{B}_n \to \mathcal{B}_{n+1}$:

$$B_{n+1}(a) = 1 + \sum_{S \in a^-} \frac{1}{\max\limits_{s \in S}(\mathcal{B}_n(s))} \overset{IV}{=} 1 + \sum_{S \in \gamma(a)^-} \frac{1}{\max\limits_{s \in S}(\mathcal{B}_n(s))} = B_{n+1}(\gamma(a)). \qquad \square$$

**Unabhängigkeit** Die Eigenschaft ist erfüllt.

*Beweis.* $AF_1 = (Ar_1, att_1), AF_2 = (Ar_2, att_2)$ seien zwei SetAFs mit $Ar_1 \cap Ar_2 = \emptyset$. Wegen $Ar_1 \cap Ar_2 = \emptyset$ gilt $a^-$ in $AF_1$ ist gleich $a^-$ in $AF_1 \oplus AF_2$ für alle $a \in Ar_1$. Das bedeutet wiederum, $\mathcal{B}_i(a)$ in $AF_1$ ist gleich $\mathcal{B}_i(a)$ in $AF_1 \oplus AF_2$ für alle $a \in Ar_1$ und alle $i \in \mathbb{N}_0$. Daraus lässt sich nun die Unabhängigkeitsbedingung schlussfolgern: $a \succcurlyeq_{\mathcal{B}} b$ in $AF_1 \Leftrightarrow a \succcurlyeq_{\mathcal{B}} b$ in $AF_1 \oplus AF_2$ für alle $a, b \in Ar_1$. $\square$

**Widersprüchlichkeit** Die Eigenschaft ist *nicht* erfüllt.
Beweis erfolgt über das Gegenbeispiel 3.3.1.

*Beweis.*

**Beispiel 3.3.1.** $AF = (Ar, att)$ sei ein SetAF mit $Ar = \{A, B, C\}$ und $att = \{(\{A\}, A), (\{A\}, B)\}, (\{C\}, B)\}$. Abbildung 3.1 stellt $AF$ grafisch dar. $A$ greift sich selbst an, für die Erfüllung der Widersprüchlichkeit muss daher $B \succ_{\mathcal{B}} A$ gelten. Das trifft allerdings nicht zu, denn die Belastungszahlen $\mathcal{B}_i$ für $i = 1$ ergeben $\mathcal{B}_1(A) = 2$ und $\mathcal{B}_1(B) = 3$. Somit gilt $A \succ_{\mathcal{B}} B$.

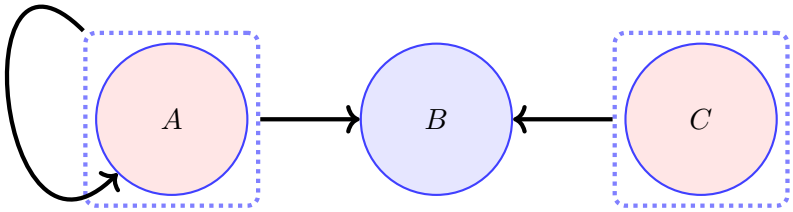



Abbildung 3.1.: Gegenbeispiel für die Widersprüchlichkeit

□

**Kardinalität-Vorrang** Die Eigenschaft ist erfüllt.

*Beweis.* $AF = (Ar, att)$ sei ein beliebiges SetAF und $a, b \in Ar$ seien zwei beliebige Argumente, sodass $|a^-| > |b^-|$. Dann gilt $\mathcal{B}_0(a) = \mathcal{B}_0(b) = 1$ und $\mathcal{B}_1(a) = 1 + |a^-| > 1 + |b^-| = \mathcal{B}_1(b)$. Daraus folgt $b \succ_{\mathcal{B}} a$. □

**Void-Vorrang** Die Eigenschaft ist erfüllt.

*Beweis.* $AF = (Ar, att)$ sei ein beliebiges SetAF und $a, b \in Ar$ seien zwei beliebige Argumente mit $a^- = \emptyset$ und $b^- \neq \emptyset$. Dann gilt $|a^-| = 0 < |b^-|$. Somit ist der Void-Vorrang ein Spezialfall des Kardinalität-Vorrangs und damit automatisch erfüllt. □

**Verteidigungsvorrang** Die Eigenschaft ist erfüllt.

*Beweis.* $AF = (Ar, att)$ sei ein SetAF und $a, b \in Ar$ seien zwei Argumente, wobei $|a^-| = |b^-|$ gilt. Weiter sei $A \neq \emptyset$ die Menge der Verteidiger von $a$ und $B = \emptyset$ die leere Verteidigermenge von $b$. Im folgenden wird nun $\mathcal{B}_i$ für $a$ und $b$ berechnet, bis

sie sich voneinander unterscheiden.
$\mathcal{B}_0(a) = 1 = \mathcal{B}_0(b)$.
$\mathcal{B}_1(a) = 1 + |a^-| = 1 + |b^-| = \mathcal{B}_1(b)$.
$\mathcal{B}_2(a) = 1 + \sum_{S \in a^-} \frac{1}{\max_{s \in S}(\mathcal{B}_1(s))} = 1 + \sum_{S \in a^-} \frac{1}{\max_{s \in S}(1+|s^-|)} \overset{A \neq \emptyset}{<} \mathcal{B}_1(a) = \mathcal{B}_1(b) = 1 + \sum_{S \in b^-} \frac{1}{(1+0)} \overset{B = \emptyset}{=} 1 + \sum_{S \in b^-} \frac{1}{\max_{s \in S}(1+|s^-|)} = 1 + \sum_{S \in b^-} \frac{1}{\max_{s \in S}(\mathcal{B}_1(s))} = \mathcal{B}_2(b)$.
Damit gilt $\mathcal{B}_2(a) < \mathcal{B}_2(b) \Rightarrow a \succ_{\mathcal{B}} b$. □

**Totalität** Die Eigenschaft ist erfüllt.

*Beweis.* Für ein SetAF $AF = (Ar, att)$ und beliebige Argumente $a, b \in Ar$ gilt genau eine der drei Möglichkeiten:

1. $\mathcal{B}_i(a) = \mathcal{B}_i(b)\ \forall i \in \mathbb{N}_0$
2. $\exists i \in \mathbb{N}_0, \mathcal{B}_i(a) < \mathcal{B}_i(b)$ *und* $\forall j \in \{0, 1 \dots i-1\}, \mathcal{B}_j(a) = \mathcal{B}_j(b)$
3. $\exists i \in \mathbb{N}_0, \mathcal{B}_i(a) > \mathcal{B}_i(b)$ *und* $\forall j \in \{0, 1 \dots i-1\}, \mathcal{B}_j(a) = \mathcal{B}_j(b)$

Durch die Definition von $\mathcal{B}$ gilt im Fall 1 $a \succcurlyeq_{\mathcal{B}} b$ und $b \succcurlyeq_{\mathcal{B}} a$. Im Fall 2 gilt $a \succ_{\mathcal{B}} b$, somit insbesondere auch $a \succcurlyeq_{\mathcal{B}} b$. Im Fall 3 gilt $b \succ_{\mathcal{B}} a$.

Zusammengefasst ist entweder $a \succcurlyeq_{\mathcal{B}} b$ oder $b \succ_{\mathcal{B}} a$ erfüllt, was dem Kriterium der Totalität entspricht. □

**Kein-Angriff-Äquivalenz** Die Eigenschaft ist erfüllt.

*Beweis.* $AF = (Ar, att)$ sei ein SetAF, $a, b \in Ar$ mit $a^- = b^- = \emptyset$. Dann gilt $\mathcal{B}_i(a) = 1 = \mathcal{B}_i(b)$ für alle $i \in \mathbb{N}_0$. Damit ist $a \succcurlyeq_{\mathcal{B}} b \wedge b \succcurlyeq_{\mathcal{B}} a$ erfüllt. □

**Vorrang der verteilten Verteidigung** Die Eigenschaft ist erfüllt.

*Beweis.* $AF = (Ar, att)$ sei ein SetAF, $a, b \in Ar$, wobei $a$ die gleiche Anzahl von Angreifern und Verteidigern besitzt wie $b$. Weiter gelte, dass $a$ simpel und verteilt verteidigt ist und $b$ ist simpel, aber nicht verteilt verteidigt. $A$ sei die Menge der Verteidiger von $a$ und entsprechend $B$ die Menge der Verteidiger von $b$.

Für $a$ gilt nun, dass $|A|$ Angreifer davon genau ein Mal angegriffen werden und $|a^-| - |A|$ nicht angegriffen werden (wg. simpler und verteilter Verteidigung). Das bedeutet, $|A|$ Summanden in $\sum_{S \in a^-} \frac{1}{\max_{s \in S}(\mathcal{B}_1(s))}$ gleichen $\frac{1}{2}$ und $|a^-| - |A|$ Summanden gleichen 1. Insgesamt ergibt sich damit die Formel:

$$\mathcal{B}_2(a) = 1 + \sum_{S \in a^-} \frac{1}{\max_{s \in S}(\mathcal{B}_1(s))} = 1 + |A| \times \frac{1}{2} + (|a^-| - |A|) \times 1 = 1 + |a^-| - \frac{|A|}{2}.$$

Für ein Argument, das von $i > 1$ Argumenten angegriffen wird, werden $i - 1$ Angreifer als *redundant* definiert. Da $b$ simpel und nicht verteilt verteidigt ist, hat $b$ eine Gesamtstumme $n > 0$ von redundanten Verteidigern.

Das bedeutet, die Summe $s1 = \sum_{S \in a^-} \frac{1}{\max_{s \in S}(\mathcal{B}_1(s))}$ weist folgende zwei Unterschiede gegenüber der Summe $s2 = \sum_{S \in b^-} \frac{1}{\max_{s \in S}(\mathcal{B}_1(s))}$ auf:

1. $n$ Summanden, die bei $s1$ $\frac{1}{2}$ gleichen, betragen bei $s2$ 1. Das liegt daran, dass jeder redundante Verteidiger das Argument $b$ gegen einen Angreifer mehr ungeschützt lässt, als das bei $a$ der Fall ist.

2. Maximal $n$ Summanden, die bei $s1$ $\frac{1}{2}$ betragen, liegen bei $s2$ zwischen $\frac{1}{n+1}$ und $\frac{1}{2}$. Der Fall $\frac{1}{n+1}$ tritt dann ein, wenn die $n$ redundanten Verteidiger von $b$ das gleiche Element angreifen, dessen Belastung dann auf $n+1$ wächst. Der gegensätzliche Fall $\frac{1}{2}$ gilt dann, wenn alle redundanten Verteidiger unterschiedliche Elemente angreifen.

Die anderen Summanden sind gleich. Der Punkt 1 bewirkt eine Erhöhung der Summe $s1$ gegenüber $s2$ um $n\frac{1}{2}$. Der Punkt 2 hat den gegenseitigen Effekt, die Summe $s1$ um maximal $n \times (\frac{1}{2} - \frac{1}{n+1}) = n\frac{1}{2} - \frac{n}{n+1}$ gegenüber $s2$ zu verringern. Für $n \geq 1$ überwiegt der Effekt von Punkt 1 oder anders gesagt $n\frac{1}{2} > n\frac{1}{2} - \frac{n}{n+1}$. Zusammengefasst gilt also $\mathcal{B}_0(a) = 1 = \mathcal{B}_0(b)$, $\mathcal{B}_1(a) = 1 + |A| = 1 + |B| = \mathcal{B}_1(b)$ und $\mathcal{B}_2(a) < \mathcal{B}_2(b)$. Daraus ergibt sich die Relation $a \succ_{\mathcal{B}} b$. □

**Vorrang strikter Verteidigungserweiterung** Die Eigenschaft ist *nicht* erfüllt.

*Beweis.* $AF = (Ar, att)$ sei ein SetAF, $a \in Ar$ und $P_+(a)$ eine Verteidigungserweiterung von $a$. Das Argument $a$ hat in $AF \oplus P_+(a)$ offensichtlich einen Angreifer mehr als in $AF$. Diese Tatsache kombiniert mit der Gültigkeit vom Kardinalität-Vorrang widerlegt den Vorrang strikter Verteidigungserweiterung. □

**Vorrang der Verteidigungserweiterung** Die Eigenschaft ist *nicht* erfüllt.

Der Beweis erfolgt über die gleiche Begründung wie beim Vorrang strikter Verteidigungserweiterung.

**Abwertung der Angriffserweiterung** Die Eigenschaft ist erfüllt.

Die Argumentation folgt erneut der vom Vorrang strikter Verteidigungserweiterung und basiert auf der Gültigkeit vom Kardinalität-Vorrang.

**Verlängerte Angriffserweiterung** Die Eigenschaft ist *nicht* erfüllt.

Der Beweis erfolgt über das Gegenbeispiel 3.3.2.

*Beweis.*

**Beispiel 3.3.2.** $AF = (Ar, att)$ sei ein SetAF mit $Ar = \{A, B_1, B_2, X_1, X_2, C, D\}$ und $att = \{(\{B_1, B_2\}, A), (\{X_1\}, B_1), (\{X_2\}, B_1), (\{C\}, B_2), (\{D\}, C)\}$. Weiter sei

$\widehat{AF} = (\hat{Ar}, \hat{att})$ ein zu $AF$ isomorphes SetAF. Der Isomorphismus sei durch die Abbildung $\gamma$ gegeben, die jedes Argument $a \in Ar$ auf ein entsprechendes $\gamma(a) \in \hat{Ar}$ abbildet. Zuletzt definiere mit $P_+(D) = (Ar_0, att_0)$ eine Verlängerung des (indirekten) Angriffs von $\{D\}$ auf $A$ mit $Ar_0 = \{E, F\}$ und $att_0 = \{(\{E\}, D), (\{F\}, E)\}$. Die Abbildung 3.2 dient der grafischen Darstellung von $AF \oplus P_+(D)$.

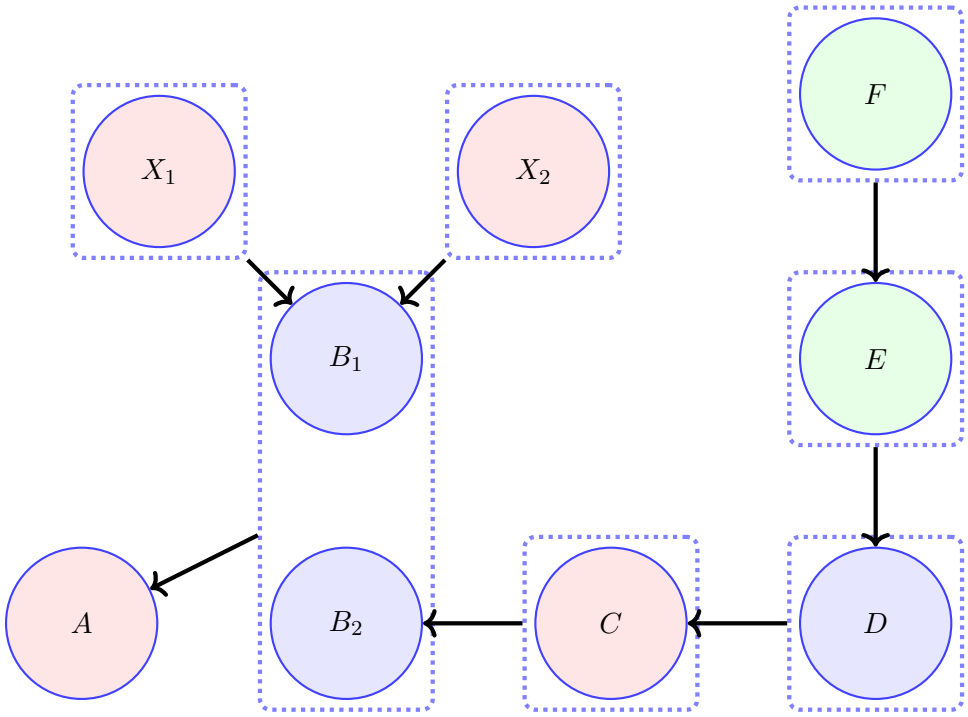



Abbildung 3.2.: Gegenbeispiel für die verlängerte Angriffserweiterung

Die Ausgangssituation für die Anwendbarkeit der Eigenschaft ist durch die Angriffsmenge $\{D\}$ gegeben, von der ein Angriffspfad der Länge 3 zum Argument $A$ führt. Es existiert kein Pfad einer geraden Länge von $\{D\}$ nach $A$ und das Argument $D$ ist (ohne die Angriffsverlängerung) nicht angegriffen.

Nachfolgend wird die Vereinigung der beiden isomorphen SetAFs mit der Angriffserweiterung $\widehat{AF} \oplus AF \oplus P_+(D)$ betrachtet. Dabei fällt auf, dass nach der Initialisierungsiteration die Belastung von $B_1$ und von $\gamma(B_1)$ konstant bei 3 liegt. Die Belastung von $B_2$ und $\gamma(B_2)$ kann dagegen maximal 2 erreichen. Das liegt daran, dass die beiden Argumente jeweils genau einen Angreifer besitzen und die maximale Belastung eines Arguments $a$ bei $1 + |a^-|$ liegt.

Damit wird die „Angriffskraft“ der Mengen $\{B_1, B_2\}$ und $\{\gamma(B_1), \gamma(B_2)\}$ vollständig durch $B_1$ bzw. $\gamma(B_1)$ dominiert. Das wiederum bedeutet, dass die Angriffserweiterung von $\{D\}$ sich nicht auf die Belastung von $A$ auswirkt. Somit ist die Belastung von $A$ über alle Iterationen identisch zu der Belastung von $\gamma(A)$, wodurch die Geltung von $A \succ_{\mathcal{B}} \gamma(A)$ widerlegt wird. □

Die Mengen-Belastungssemantik und der nh-Categoriser haben ähnliche Funktionsweisen. Daher bietet es sich an, das angegebene Beispiel mit dem nh-Categoriser ebenfalls auszuwerten. Die Argumente $X_1$ und $X_2$ sind nicht angegriffen. Sie erhalten damit – wie bei der Mengen-Belastungssemantik – eine Bewertung von 1. Anhand davon lässt sich die Bewertung $C(B_1)$ berechnen:

$$C(B_1) = \frac{1}{1 + \sum\limits_{S \in B_1^-} \min\limits_{s \in S} C(s)} = \frac{1}{1 + C(X_1) + C(X_2)} = \frac{1}{3}.$$

Die Bewertung $C(B_2)$ kann mit der folgenden Formel ausgedrückt werden:

$$C(B_2) = \frac{1}{1 + \sum\limits_{S \in B_2^-} \min\limits_{s \in S} C(s)} = \frac{1}{1 + C(C)} \geq \frac{1}{2}.$$

Die Relation $C(B_2) \geq \frac{1}{2}$ gilt, weil die Bewertung $C(C)$ im Intervall $(0, 1]$ liegt.

Somit gilt $C(B_1) < C(B_2)$ unabhängig von möglichen Angriffsverlängerungen für $\{D\}$. Damit wird der Pfad von $\{D\}$ nach $A$ „ausgeschaltet“, was der Gültigkeit der Eigenschaft widerspricht.

**Verlängerte Verteidigungserweiterung** Die Eigenschaft ist *nicht* erfüllt.

Der Beweis funktioniert mit einem fast identischen Gegenbeispiel zu 3.3.2. In dem Beispiel müssen nur das Argument $D$ und dessen Angriff auf $C$ entfernt werden. Damit lässt sich die Menge $\{C\}$ als die Verteidigung von $A$ analysieren, deren Erweiterung $P_+(C)$ keine Auswirkung auf die Belastung von $A$ hat.

Mit dieser Modifizierung lässt sich die Geltung der Eigenschaft ebenso für den nh-Categoriser widerlegen.

**Vorrang der vollen Verteidigung** Die Eigenschaft ist *nicht* erfüllt.

Der Beweis erfolgt über die Angabe des Gegenbeispiels 3.3.3.

*Beweis.*

**Beispiel 3.3.3.** $AF = (Ar, att)$ sei ein SetAF mit $Ar = \{A, B, C, D, E, X, Y\}$ und $att = \{(\{B\}, A), (\{D\}, A), (\{C\}, B), (\{E\}, D), (\{Y\}, X)\}$. $AF$ wird in Abbildung 3.3 dargestellt.

Es gilt:

- $X^- = \{\{Y\}\}$ mit $\{Y\} \in \mathcal{P}(Ar) \setminus \emptyset$.
- $Y^- = \emptyset$.
- Es gibt keine Menge $S \in \mathcal{P}(Ar) \setminus \emptyset$ mit $s^- = \emptyset \, \forall s \in S$, sodass ein Pfad ungerader Länge von $S$ nach $A$ existiert.

Damit ist $A$ vollständig verteidigt, während $X$ nicht verteidigt ist. Für die Gültigkeit vom Vorrang der vollen Verteidigung muss daher $A \succ_{\mathcal{B}} X$ gelten. Die Betrachtung der Belastungswerte von $A$ und $X$ zeigt jedoch, dass das nicht der Fall ist: $\mathcal{B}_0(A) = 1 = \mathcal{B}_0(X)$ und $\mathcal{B}_1(A) = 3 > 2 = \mathcal{B}_1(X)$.

□

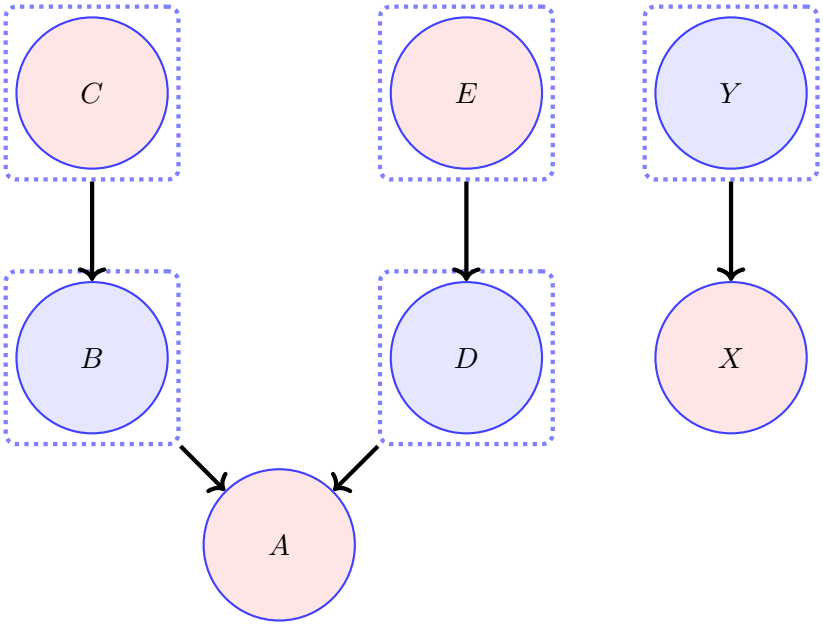



Abbildung 3.3.: Gegenbeispiel für den Vorrang der vollen Verteidigung

**Qualitätsvorrang** Die Eigenschaft ist *nicht* erfüllt.

*Beweis.* Das Beispiel 3.3.3 kann auch zum Widerlegen des Qualitätsvorrangs verwendet werden. Beide Angreifer $B$ und $D$ von $A$ sind gleich stark und zugleich schwächer als $Y$ – der einzige Angreifer von $X$. Damit der Qualitätsvorrang gilt, muss in diesem Fall $A \succ_{\mathcal{B}} X$ gelten. Es wurde jedoch in dem Beispiel bereits das Gegenteil gezeigt. □

**Gegentransitivität** Die Eigenschaft ist erfüllt.

*Beweis.* $AF = (Ar, att)$ sei ein SetAF, $a, b \in Ar$ mit $a^- \geq b^-$.

Der Fall $|a^-| < |b^-|$ ist aufgrund der Definition von $a^- \geq b^-$ nicht möglich (Existenz injektiver Abb. von $b^-$ nach $a^-$).

Im Fall $|a^-| > |b^-|$ gilt $b \succ_{\mathcal{B}} a$ trivialerweise wegen der Geltung vom Kardinalität-Vorrang.

Die Relation $b \succcurlyeq_{\mathcal{B}} a$ muss daher nur noch für den Fall $|a^-| = |b^-|$ bewiesen werden. Durch $a^- \geq b^-$ ist gegeben, dass eine injektive Abbildung $f : b^- \to a^-$ existiert, für die $\min(f(x)) \succcurlyeq_{\mathcal{B}} \min(x)$ für alle $x \in b^-$ gilt.

Für jedes $x \in b^-$ muss somit einer der beiden Fälle gelten:

1. $\mathcal{B}_i(\min(f(x))) = \mathcal{B}_i(\min(x))\ \forall i \in \mathbb{N}_0$ **oder**
2. Es existiert ein $n \in \mathbb{N}$, sodass $B_n(\min(f(x))) < B_n(\min(x))$ *und* $\forall j \in \{0, 1 \ldots n-1\}, \mathcal{B}_j(\min(f(x))) = \mathcal{B}_j(\min(x))$ gilt.

Im ersten Fall gilt für alle $i \in \mathbb{N}$:
$\mathcal{B}_i(b) = 1 + \sum_{S \in b^-} \frac{1}{\max_{s \in S}(\mathcal{B}_{i-1}(s))} = 1 + \sum_{S \in b^-} \frac{1}{\mathcal{B}_{i-1}(\min(s))} = 1 + \sum_{S \in b^-} \frac{1}{\mathcal{B}_{i-1}(\min(f(s)))} = 1 + \sum_{S \in a^-} \frac{1}{\mathcal{B}_{i-1}(\min(s))} = 1 + \sum_{S \in a^-} \frac{1}{\max_{s \in S}(\mathcal{B}_{i-1}(s))} = \mathcal{B}_i(a)$
$\Rightarrow b \succcurlyeq_{\mathcal{B}} a$ (und gleichzeitig $a \succcurlyeq_{\mathcal{B}} b$).

Im zweiten Fall wird das kleinste $n$ betrachtet, für das Argumente $x_1 \dots x_t \in b^-$ existieren, die die Bedingungen $\mathcal{B}_n(\min(f(x_i))) < \mathcal{B}_n(\min(x_i))$ und $\forall j \in \{0, 1 \dots n-1\}, \mathcal{B}_j(\min(f(x_i))) = \mathcal{B}_j(\min(x_i))$ erfüllen.

Der Ausdruck $\mathcal{B}_j(\min(f(x))) = \mathcal{B}_j(\min(x))$ für alle $j \in \{0, 1 \dots n-1\}$ gilt für sämtliche Angreifer $x \in b^-$. In der Iteration $n$ hingegen gilt $\mathcal{B}_n(\min(f(x))) = \mathcal{B}_n(\min(x))$ nur noch für $|b^-| - t$ Angreifer. Für die restlichen $t$ Angreifer gilt $\mathcal{B}_n(\min(f(x))) < \mathcal{B}_n(\min(x))$.

Daraus lässt sich $\mathcal{B}_i(b) = \mathcal{B}_i(a)$ für alle $i \in \{0 \dots n\}$ schlussfolgern. Für $n+1$ gilt dementsprechend $\mathcal{B}_{n+1}(b) = 1 + \sum_{S \in b^-} \frac{1}{\max_{s \in S}(\mathcal{B}_n(s))} = 1 + \sum_{S \in b^-} \frac{1}{\mathcal{B}_n(\min(s))} < 1 + \sum_{S \in b^-} \frac{1}{\mathcal{B}_n(\min(f(s)))} = 1 + \sum_{S \in a^-} \frac{1}{\mathcal{B}_n(\min(s))} = 1 + \sum_{S \in a^-} \frac{1}{\max_{s \in S}(\mathcal{B}_n(s))} = \mathcal{B}_{n+1}(a)$.

Zusammengefasst ergibt sich $\mathcal{B}_i(b) = \mathcal{B}_i(a)$ für $i \in \{0 \dots n\}$ und $\mathcal{B}_{n+1}(b) < \mathcal{B}_{n+1}(a)$. Daraus folgt $b \succ_{\mathcal{B}} a$. □

**Strikte Gegentransitivität** Die Eigenschaft ist erfüllt.

Der Fall 2 im Beweis der Gültigkeit der Gegentransitivität impliziert ebenfalls die Gültigkeit strikter Gegentransitivität.

# 4. Alternative Definitionen

Die im obigen Kapitel definierte Anpassung der Belastungssemantik basiert auf der Idee, dass ein Angriff durch eine Argumentenmenge nur so stark sein kann wie ihr schwächstes Element. Auch wenn diese Vorgehensweise begründet ist, können Teilaspekte davon hinterfragt werden. Ein möglicher Kritikpunkt besteht darin, dass die übrigen Argumente der angreifenden Menge teilweise keinen Effekt auf den Angriff oder die gesamte Rangberechnung haben. Daher werden nachfolgend mehrere abweichende Möglichkeiten der Berechnung der Angriffskraft (oder hier Angriffsbelastung) einer Argumentenmenge vorgestellt. Die darauf aufbauenden Definitionen der Rangrelation sowie der passenden Semantik sind analog zu $\mathcal{B}$ und werden zur Redundanzvermeidung ausgelassen.

## 4.1. Durchschnitt-Belastungssemantik

Ein Weg, mehrere Belastungen zusammenzufassen liegt in der Berechnung ihres Durchschnitts.

**Definition 4.1.1.** Die Durchschnittsabbildung $\varnothing : \mathcal{P}(\mathbb{R})\backslash\emptyset \to \mathbb{R}$ sei mit $\varnothing(S) \mapsto \frac{\sum_{s\in S} s}{|S|}$ definiert.

Mit Hilfe der Durchschnittsdefinition 4.1.1 lässt sich die Mengenbelastung nun wie in Definition 4.1.2 einführen.

**Definition 4.1.2.** $AF = (Ar, att)$ sei ein beliebiges SetAF. Die Mengenbelastung $\mathcal{B}_i^{\varnothing}(a)$ wird für alle $a \in Ar$ und $i \in \mathbb{N}_0$ wie folgt definiert:

$$\mathcal{B}_i^{\varnothing}(a) = \begin{cases} 1, & \text{falls } i = 0 \\ 1, & \text{falls } a^- = \emptyset \\ 1 + \sum\limits_{S\in a^-} \frac{1}{\varnothing(\{\mathcal{B}_{i-1}^{\varnothing}(s)|s\in S\})}, & \text{sonst} \end{cases}$$

## 4.2. Summen-Belastungssemantik

Die Schwäche der auf $\mathcal{B}_i^{\varnothing}$ basierenden Mengen-Belastungssemantik besteht darin, dass eine Angriffsmenge durch die Hinzunahme von anderen Argumenten potentiell stärker werden kann. Das widerspricht dem semantischen Gedanken einer Angriffsmenge. Es bieten sich Optionen der Berechnung von Mengenbelastungen an,

bei denen das Gegenteil passiert. Eine solche Möglichkeit liegt in der Summierung von Belastungen aller Argumente der Menge (s. Def. 4.2.1).

**Definition 4.2.1.** $AF = (Ar, att)$ sei ein beliebiges SetAF. Die Mengenbelastung $\mathcal{B}_i^+(a)$ wird für alle $a \in Ar$ und $i \in \mathbb{N}_0$ wie folgt definiert:

$$\mathcal{B}_i^+(a) = \begin{cases} 1, & \text{falls } i = 0 \\ 1, & \text{falls } a^- = \emptyset \\ 1 + \sum\limits_{S \in a^-} \frac{1}{\sum\limits_{s \in S} \mathcal{B}_{i-1}^+(s)}, & \text{sonst} \end{cases}$$

## 4.3. Produkt-Belastungssemantik

Die kleinste mögliche Belastung eines Arguments liegt bei 1. Das bedeutet, dass ein Produkt von mehreren Belastungen mindestens so groß ist wie sein größter Faktor. Somit besitzt eine auf dem Produkt basierende Berechnung der Mengenbelastung eine zu $\mathcal{B}^+$ ähnliche Funktion.

**Definition 4.3.1.** $AF = (Ar, att)$ sei ein beliebiges SetAF. Die Mengenbelastung $\mathcal{B}_i^*(a)$ wird für alle $a \in Ar$ und $i \in \mathbb{N}_0$ wie folgt definiert:

$$\mathcal{B}_i^*(a) = \begin{cases} 1, & \text{falls } i = 0 \\ 1, & \text{falls } a^- = \emptyset \\ 1 + \sum\limits_{S \in a^-} \frac{1}{\prod\limits_{s \in S} \mathcal{B}_{i-1}^*(s)}, & \text{sonst} \end{cases}$$

Im Unterschied zu der Summenbelastung tragen nichtangegriffene Argumente der Angreifermenge nicht zu einer Erhöhung der Gesamtbelastung bei.

## 4.4. Erfüllung von Semantikeigenschaften

Die Tabelle 4.1 zeigt eine Gegenüberstellung der unterschiedlichen Variationen in Bezug auf ihre Eigenschaftenerfüllung.

Für alle Eigenschaften, bei denen die Erfüllung bzw. Nichterfüllung bei allen vier Variationen übereinstimmt, funktionieren die Beweise aus dem Kapitel 3.3. Die Unterschiede bei den anderen Eigenschaften werden nachfolgend begründet. Da der Fokus der Arbeit hauptsächlich auf der Semantik $\mathcal{B}$ liegt, wird an dieser Stelle auf die detaillierte Beweisführung verzichtet.

**Kardinalität-Vorrang** Im Beweis für diese Eigenschaft bei der Semantik $\mathcal{B}$ wurde die Gleichung $\mathcal{B}_1(a) = 1 + |a^-|$ ausgenutzt. Diese Formel gilt, weil die initiale Belastung immer bei 1 liegt und das Maximum beliebig vieler Einsen immer Eins ist.

| **Mengen-Rangsemantik Eigenschaft** | $\mathcal{B}^{\varnothing}$ | $\mathcal{B}^{+}$ | $\mathcal{B}^{*}$ | $\mathcal{B}$ |
|---|---|---|---|---|
| Abstraktion | ✓ | ✓ | ✓ | ✓ |
| Unabhängigkeit | ✓ | ✓ | ✓ | ✓ |
| Void-Vorrang | ✓ | ✓ | ✓ | ✓ |
| Widersprüchlichkeit | ✗ | ✗ | ✗ | ✗ |
| Kardinalität-Vorrang | ✓ | ✗ | ✓ | ✓ |
| Verteidigungsvorrang | ✓ | ✗ | ✓ | ✓ |
| Totalität | ✓ | ✓ | ✓ | ✓ |
| Kein-Angriff-Äquivalenz | ✓ | ✓ | ✓ | ✓ |
| Vorrang der verteilten Verteidigung | ✗ | ✗ | ✓ | ✓ |
| Vorrang strikter Verteidigungserweiterung | ✗ | ✗ | ✗ | ✗ |
| Vorrang der Verteidigungserweiterung | ✗ | ✗ | ✗ | ✗ |
| Abwertung der Angriffserweiterung | ✓ | ✓ | ✓ | ✓ |
| Verlängerte Angriffserweiterung | ✓ | ✓ | ✓ | ✗ |
| Verlängerte Verteidigungserweiterung | ✓ | ✓ | ✓ | ✗ |
| Vorrang der vollen Verteidigung | ✗ | ✗ | ✗ | ✗ |
| Qualitätsvorrang | ✗ | ✗ | ✗ | ✗ |
| Gegentransitivität | ✗ | ✗ | ✗ | ✓ |
| Strikte Gegentransitivität | ✗ | ✗ | ✗ | ✓ |

Tabelle 4.1.: Variationen der Mengen-Belastungssemantik – Eigenschaften

Das Produkt und der Durchschnitt von beliebig vielen Einsen resultieren ebenfalls in einer Eins. Daher gilt eine analoge Formel gleichermaßen für $\mathcal{B}_1^{\varnothing}$ und $\mathcal{B}_1^{*}$.

Bei der Summenvariante gilt dagegen $\mathcal{B}_1^{+}(a) = 1 + \sum_{S \in a^-} \frac{1}{|S|}$. Bereits in der ersten Iteration sind die Angriffsmengen mit mehr Elementen somit schwächer. Mit diesem Wissen lässt sich leicht das Beispiel 4.4.1 konstruieren, bei dem das Argument $A$ über mehr, dafür jedoch schwächere Angreifer als $X$ verfügt.

**Beispiel 4.4.1.** $AF = (Ar, att)$ sei ein SetAF mit $Ar = \{A, B_1, B_2, B_3, C_1, C_2, C_3, X, Y\}$ und $att = \{(\{B_1, B_2, B_3\}, A), (\{C_1, C_2, C_3\}, A), (\{Y\}, X)\}$.

Ein Vergleich von Kardinalitäten zeigt $|A^-| = 2 > 1 = |X^-|$. Die beiden Angreifer von $A$ bestehen jedoch aus jeweils drei Elementen und die Belastung $\mathcal{B}_1^{+}(A)$ liegt damit bei $\frac{5}{3}$. $X$ besitzt zwar nur einen Angreifer, dieser greift jedoch mit der maximalen Angriffskraft an, was in der Belastung $\mathcal{B}_1^{+}(X) = 2 > \mathcal{B}_1^{+}(A)$ resultiert.

Die Schlussfolgerung $A \succ_{\mathcal{B}^{+}} X$ widerlegt die Gültigkeit der Eigenschaft für diese Semantik.

**Verteidigungsvorrang** Auch hier fällt nur die summenbasierte Semantik auf. Die Begründung für die Abweichung ähnelt dem Kardinalität-Vorrang. Ein Angriff kann bei $\mathcal{B}_1^{+}$ allein durch die Mengengröße geschwächt sein, ohne die Notwendigkeit einer Verteidigung. Das Gegenbeispiel 4.4.2 zeigt eine entsprechende Situation.

**Beispiel 4.4.2.** $AF = (Ar, att)$ sei ein SetAF mit $Ar = \{A, B_1, B_2, B_3, X, Y, Z\}$ und $att = \{(\{B_1, B_2, B_3\}, A), (\{Y\}, X), (\{Z\}, Y)\}$. Die Argumente $A$ und $X$ haben beide jeweils einen Angreifer, wobei nur $X$ vor seinem Angreifer durch $Z$ verteidigt wird. Dennoch ist die Belastung $\mathcal{B}_1^+(A) = \frac{4}{3}$ geringer als $\mathcal{B}_1^+(X) = 2$.

Damit gilt $A \succ_{\mathcal{B}^+} X$, was dem Verteidigungsvorrang widerspricht.

**Vorrang der verteilten Verteidigung** An dieser Eigenschaft scheitern nun zwei Variationen – $\mathcal{B}^\varnothing$ und $\mathcal{B}^+$. Die Begründung im Fall von $\mathcal{B}^+$ liegt erneut darin, dass eine angreifende Menge durch eine höhere Anzahl an Elementen stärker belastet wird. Ein Gegenbeispiel für diese Eigenschaft nimmt zu viel Platz in Anspruch und wird schnell unübersichtlich. Die Konstruktionsidee liegt jedoch bei zwei Argumenten $a$ und $b$, wobei die Anzahl der Angreifer und Verteidiger von $a$ und $b$ übereinstimmt. Weiter gelte, dass $a$ simpel und nichtverteilt verteidigt sei, während $b$ simpel und verteilt verteidigt sei.

Sollte die Eigenschaft gelten, wird der Angriff auf $b$ durch die verteilte Verteidigung so weit abgeschwächt, dass er schwächer als der Angriff auf $a$ wird. Somit wird erwartet, dass $b$ eine kleinere Belastung erhält und damit $b \succ_{\mathcal{B}^+} a$ gilt. Haben die Angreifer von $a$ aber mehr Elemente als die Angreifer von $b$, ist der Angriff auf $a$ bereits vor der Berücksichtigung der Verteidigung schwächer. In diesem Fall reicht der Unterschied zwischen verteilter und nichtverteilter Verteidigung nicht unbedingt aus, um den Angriff auf $b$ ausreichend abzuschwächen. Bei passend gewählten Größen gilt dann $a \succ_{\mathcal{B}^+} b$ und die Eigenschaft wird somit nicht erfüllt.

Bei $\mathcal{B}^\varnothing$ gilt das Gegenteil zu $\mathcal{B}^+$ – eine Angreifermenge wird stärker, wenn sie mehr nichtangegriffene Elemente enthält (oder bleibt zumindest gleich stark, wenn ihre sämtlichen Elemente nicht angegriffen sind). Die Konstruktion von dem Gegenbeispiel ist damit ähnlich, aber zu $\mathcal{B}^+$ gespiegelt. Bei $\mathcal{B}^+$ werden die Angriffsmengen von dem nichtverteilt verteidigten Argument vergrößert. Dadurch werden sie schwächer und das angegriffene Argument entsprechend stärker. Bei $\mathcal{B}^\varnothing$ dagegen wird die angreifende Menge des verteilt verteidigten Arguments durch Vergrößerung gestärkt. Mit einer ausreichenden Größe wird das verteilt verteidigte Argument schwächer gegenüber dem nichtverteilt verteidigten.

**Verlängerte Angriffs- und Verteidigungserweiterung** Alle drei alternativen Definitionen erfüllen diese beiden Eigenschaften. Bei der Semantik $\mathcal{B}$ gilt die Eigenschaft nicht, da bestimmte Pfade „ausgeschaltet“ werden können. Das geschieht dann, wenn ein Pfad über ein Element $s$ einer Angriffsmenge $S$ läuft, wobei $s$ in dieser Menge nie das Element mit der höchsten Belastung ist.

In den neuen Semantiken besteht der Grundgedanke gerade darin, alle Elemente der Angriffsmengen in der Berechnung zu berücksichtigen. Somit wirken sich alle Pfade auf die Belastung ihres Ziels aus.

**(Strikte) Gegentransitivität** Im Gegenteil zu $\mathcal{B}$ erfüllt keine der drei neuen Variationen die Eigenschaften. Das ist nicht überraschend, denn die Relation $\geq$ bzw.

$>$ für Mengen von angreifenden Mengen ist mit dem Grundgedanken konzipiert, die Angreifer anhand ihres schwächsten Elements zu vergleichen. Die Berechnungsart der Angriffskraft bei den alternativen Belastungssemantiken widerspricht direkt und bewusst dieser Idee.

Das Gegenbeispiel 4.4.3 widerlegt beide Eigenschaften für $\mathcal{B}^{\varnothing}$, Gegenbeispiel 4.4.4 für $\mathcal{B}^{+}$ und Gegenbeispiel 4.4.5 für $\mathcal{B}^{*}$.

**Beispiel 4.4.3.** $AF = (Ar, att)$ sei ein SetAF mit $Ar = \{A, B_1, B_2, B_3, B_4, C_1, C_2, C_3, X, Y, Z_1, Z_2\}$ und $att = \{(\{B_1, B_2, B_3, B_4\}, A), (\{C_1\}, B_4), (\{C_2\}, B_4), (\{C_3\}, B_4), (\{Y\}, X), (\{Z_1\}, Y), (\{Z_2\}, Y)\}$.

Im Folgenden werden die Angreifermengen von $A$ und von $X$ verglichen.

Das schwächste Element von $\{B_1, B_2, B_3, B_4\} \in A^-$ ist $B_4$ mit $\mathcal{B}_1^{\varnothing}(B_4) = 4$. Für $\{Y\} \in X^-$ gilt $\mathcal{B}_1^{\varnothing}(Y) = 2$. Damit ist $X^- > A^-$ erfüllt.

Dennoch gilt in der zweiten Iteration $\mathcal{B}_2^{\varnothing}(A) = \frac{11}{7} > \frac{4}{3} = \mathcal{B}_2^{\varnothing}(X)$. Daraus folgt $X \succ_{\mathcal{B}^{\varnothing}} A$.

**Beispiel 4.4.4.** $AF = (Ar, att)$ sei ein SetAF mit $Ar = \{A, B_1, B_2, B_3, X, Y, Z\}$ und $att = \{(\{B_1, B_2, B_3\}, A), (\{Y\}, X), (\{Z\}, Y)$.

Alle Elemente in $\{B_1, B_2, B_3\} \in A^-$ weisen die gleiche Belastung von 1 auf. $Y \in \{Y\} \in X^-$ wird mit 2 belastet. Somit gilt $A^- > X^-$.

Für $A$ und $X$ gilt allerdings $\mathcal{B}_1^{+}(A) = \frac{4}{3} < 2 = \mathcal{B}_1^{+}(X)$. Daraus folgt $A \succ_{\mathcal{B}^{+}} X$.

**Beispiel 4.4.5.** $AF = (Ar, att)$ sei ein SetAF mit $Ar = \{A, B, C_1, C_2, X, Y_1, Y_2, Z_1, Z_2\}$ und $att = \{(\{B\}, A), (\{C_1\}, B), (\{C_2\}, B), (\{Y_1, Y_2\}, X), (\{Z_1\}, Y_1), (\{Z_2\}, Y_2)$.

Ein Vergleich der Angreifermengen von $A$ und $X$ zeigt folgende Werte: $\mathcal{B}_1^{*}(B) = 3 > 2 = \mathcal{B}_1^{*}(Y_1) = \mathcal{B}_1^{*}(Y_2)$. Damit gilt $X^- > A^-$.

Bei den Belastungen von $A$ und $X$ wird jedoch in der zweiten Iteration $X \succ_{\mathcal{B}^{*}} A$ anhand der Belastungen $\mathcal{B}_2^{*}(A) = \frac{4}{3} > \frac{5}{4} = \mathcal{B}_2^{*}(X)$ festgestellt.

## 4.5. Vergleich

Die Semantik $\mathcal{B}^{\varnothing}$ verfügt über das bereits genannte unerwünschte Merkmal, dass die Angreifermengen durch Vergrößerung stärker werden können. Aus diesem Grund ist es wenig überraschend, dass sie bezüglich der Eigenschaftenerfüllung $\mathcal{B}$ nicht ebenbürtig ist.

Dass die Summensemantik $\mathcal{B}^{+}$ noch weniger Eigenschaften erfüllt als $\mathcal{B}^{\varnothing}$, kann dagegen im ersten Moment überraschen. Bei näherer Betrachtung wird die Ursache dafür jedoch deutlich. Die Angreifermengen werden bei dieser Semantik mit jedem neuen Element schwächer, auch wenn diese Elemente die maximale mögliche Angriffskraft besitzen. Diese Verhaltensweise steht im Konflikt zu Kardinalität- und Verteidigungsvorrang, spricht aber nicht unbedingt gegen den Grundgedanken von Angriffsmengen. Es kann daher sinnvoll sein, die Sinnhaftigkeit dieser Eigenschaften in der gegebenen Form zu hinterfragen. Alternative Definitionen können

potentiell für bestimmte konkrete Anwendungen oder Rangsemantiken besser geeignet sein.

Die besten Ergebnisse in Bezug auf die definierten Semantikeigenschaften zeigte die produktbasierte Semantik $\mathcal{B}^*$. Als einzige Kritik kann die Nichterfüllung von der (strikten) Gegentransitivität genannt werden. Auch hier kann jedoch zur Verteidigung festgehalten werden, dass für beide Eigenschaften andere – zu $\mathcal{B}^*$ kompatible Definitionen – denkbar sind. Solche Alternativen können erneut in Abhängigkeit von der konkreten Situation und ihrer Modellierung gewählt werden.

# 5. Algorithmus

Vor der konkreten Implementierung der Mengen-Belastungssemantik wird ihre algorithmische Berechnung in Pseudocode vorgestellt. Damit wird das Verfahren konzeptuell dargestellt und auf dieser Basis in Bezug auf die Terminierung sowie Laufzeit untersucht.

Als Eingabe für den Algorithmus dient die Argumentenmenge $Ar$ eines SetAF $AF = (Ar, att)$. Zu jedem Argument $a \in Ar$ sind dem Algorithmus zusätzlich folgende Variablen zugänglich:

- Die Angreifermenge $a^-$ von $a$
- Minimaler möglicher Rang nach dem Kenntnisstand aktueller Iteration ($a.MinRang$). Der Rang wird als eine Zahl zwischen 1 und $|Ar|$ ausgedrückt, wobei eine kleinere Zahl einen „besseren“ Rang bedeutet.
- Maximaler möglicher Rang nach dem Kenntnisstand aktueller Iteration ($a.MaxRang$)
- Die Belastung der aktuellen Iteration ($a.Belastung$)
- Die Belastung der nächsten Iteration ($a.TempBelastung$): Sie darf die aktuelle Belastung nicht direkt überschreiben, da diese ggf. noch für die Berechnung von Belastungen anderer Argumente benötigt wird.
- Eine Angabe, ob das Argument seine endgültige Belastung erreicht hat ($a.IstEndbelastung$).

Die minimalen und maximalen Ränge aller Argumente dienen gleichzeitig als Ausgabe des Algorithmus. Ein kleinerer Rang ist „besser“ oder formell ausgedrückt $a.MaxRang \leq b.MinRang \Rightarrow a \succcurlyeq_{\mathcal{B}} b$ für beliebige Argumente $a$ und $b$. Für ein Argument $a$ mit einem eindeutigen Rang gilt nach der Terminierung $a.MinRang = a.MaxRang$. Die Argumente, die aus der Sicht der Mengen-Belastungssemantik gleich bewertet sind, erhalten das gleiche Rangintervall $[MinRang, MaxRang]$.

Diese Ausgabeart basiert auf der Sortierung der Argumente und ist nur bei der Gültigkeit der Transitivität für die $\succcurlyeq_{\mathcal{B}}$-Relation möglich. Andernfalls lassen sich die Rangrelationen zwischen allen Argumenten nur über eine $|Ar| \times |Ar|$-große Matrix ausdrücken. Dass die Transitivität hier gilt, ist allerdings nicht offensichtlich. Im ersten Augenblick fällt für beliebige Argumente $a, b$ und $c$ auf, dass ihre paarweisen Relationen potentiell in unterschiedlichen Iterationen bestimmt werden. Die Belastungen zwischen den Iterationen können sich verschieben. Das führt allerdings nicht zum Verlust der Transitivität wie die nachfolgende Proposition 5.0.1 zeigt.

**Proposition 5.0.1.** Die Rangrelation $\succcurlyeq_{\mathcal{B}}$ ist transitiv.

*Beweis.* Es gelte $a \succcurlyeq_{\mathcal{B}} b$ und $b \succcurlyeq_{\mathcal{B}} c$ für beliebige Argumente $a, b$ und $c$. Das kann durch vier unterschiedliche Situationen entstehen:

1. $\mathcal{B}_i(a) = \mathcal{B}_i(b) = \mathcal{B}_i(c) \forall i \in \mathbb{N}_0$.

   In diesem Fall gilt $a \succcurlyeq_{\mathcal{B}} c$ trivialerweise und damit auch die Transitivität.

2. $\mathcal{B}_i(a) = \mathcal{B}_i(b) \forall i \in \mathbb{N}_0$ und es gibt ein $j \in \mathbb{N}$ mit $\mathcal{B}_j(b) < \mathcal{B}_j(c)$ mit $\mathcal{B}_i(b) = \mathcal{B}_i(c)$ für $i < j$.

   Dann gilt aufgrund der Gleichheit der Belastungen von $a$ und $b$ die Aussage $\mathcal{B}_j(a) < \mathcal{B}_j(c)$, wobei $\mathcal{B}_i(a) = \mathcal{B}_i(c)$ für $i < j$. Somit ist auch hier $a \succcurlyeq_{\mathcal{B}} c$ erfüllt.

3. Es gibt ein $j \in \mathbb{N}$ mit $\mathcal{B}_j(a) < \mathcal{B}_j(b)$, wobei $\mathcal{B}_i(a) = \mathcal{B}_i(b)$ für $i < j$ und $\mathcal{B}_i(b) = \mathcal{B}_i(c) \forall i \in \mathbb{N}_0$.

   Der Beweis erfolgt hier analog zum Fall 2.

4. Es gibt ein $j \in \mathbb{N}$ mit $\mathcal{B}_j(a) < \mathcal{B}_j(b)$, wobei $\mathcal{B}_i(a) = \mathcal{B}_i(b)$ für $i < j$ und es gibt ein $m \in \mathbb{N}$ mit $\mathcal{B}_m(b) < \mathcal{B}_m(c)$, wobei $\mathcal{B}_i(b) = \mathcal{B}_i(c)$ für $i < m$.

   Im Fall $j < m$ gilt dann $\mathcal{B}_j(a) < \mathcal{B}_j(b) = \mathcal{B}_j(c)$ und $\mathcal{B}_i(a) = \mathcal{B}_i(b) = \mathcal{B}_i(c)$ für $i < j$.

   Im Fall $j > m$ analog $\mathcal{B}_m(a) = \mathcal{B}_m(b) < \mathcal{B}_m(c)$ und $\mathcal{B}_i(a) = \mathcal{B}_i(b) = \mathcal{B}_i(c)$ für $i < m$.

   Für den Fall $j = m$ gilt schließlich $\mathcal{B}_j(a) < \mathcal{B}_j(b) < \mathcal{B}_j(c)$ und $\mathcal{B}_i(a) = \mathcal{B}_i(b) = \mathcal{B}_i(c)$ für $i < m$.

   In allen drei Fällen wird somit das Transitivitätskriterium $a \succcurlyeq_{\mathcal{B}} c$ erfüllt.

□

## 5.1. Initialisierung und Hauptschleife

Der Einstiegspunkt für den Algorithmus wird im Codeausschnitt 2 beschrieben.

Die erste Schleife in den Zeilen $1 - 5$ dient der Initialisierung. Alle Belastungen werden entsprechend der Definition von $\mathcal{B}_0$ hierbei auf 1 gesetzt. Da an dieser Stelle noch keine Ränge bekannt sind, wird für alle Argumente der minimale mögliche Rang auf 1 gesetzt und der maximale auf $|Ar|$.

Die Variable $Abbruch$ in der sechsten Zeile signalisiert, ob ein Abbruchkriterium für den gesamten Algorithmus erreicht wurde. Da dies am Anfang der Ausführung nicht zutrifft, wird die Variable mit $Falsch$ initialisiert.

Die $while$-Schleife in Zeile 7 wird nachfolgend *Hauptschleife* genannt. Sie wird bis zum Eintritt eines Abbruchkriteriums ausgeführt. Bei jeder Iteration $i$ werden im Inneren der Schleife die Funktionen zur Berechnung der Belastungen und Ränge

aufgerufen. Dabei ist anzumerken, dass der Algorithmus nicht die Belastungen aller vergangenen Iterationen speichert. Die Belastungen $\mathcal{B}_i$ der aktuellen Iteration $i$ werden pro Argument $a \in Ar$ in $a.Belastung$ gespeichert. Zusätzlich werden die Belastungen der Iteration $i+1$ in $a.TempBelastung$ gespeichert. Die Werte aus den Iterationen $j < i$ sind weder für die Berechnung der neuen Belastungen noch für die Berechnung der Ränge relevant.

**Pseudocode 2** Initialisierung und Hauptschleife

```
1:  for all a ∈ Ar do
2:      a.Belastung ← 1
3:      a.MinRang ← 1
4:      a.MaxRang ← |Ar|
5:  end for
6:  Abbruch ← Falsch
7:  while ¬Abbruch do
8:      Konvergiert ← BerechneBelastung(Ar)
9:      for all a ∈ Ar do
10:         a.Belastung ← a.TempBelastung
11:     end for
12:     AlleRaengeFertig ← BerechneRang(Ar)
13:     Abbruch ← AlleRaengeFertig ∨ Konvergiert
14: end while
```

Innerhalb der Hauptschleife wird zuletzt das Erreichen der Abbruchbedingung geprüft. Das Verfahren definiert zwei Abbruchkriterien:

1. **Rangeindeutigkeit** (Variable $AlleRaengeFertig$)

   Die Ränge für alle Argumente $a$ haben ihren endgültigen Wert erreicht. Damit ein Argument $a$ diese Voraussetzung erfüllt, muss mindestens einer der beiden nachstehenden Punkte gelten:

   - Der minimale Rang von $a$ ist gleich seinem maximalen Rang ($a.MinRang = a.MaxRang$).
   - Die Belastung von $a$ ist zwischen zwei aufeinanderfolgenden Iterationen identisch.

   **oder**

2. $\epsilon$**-Kriterium** (Variable $Konvergiert$)

   Die Differenz in den Belastungen zwischen zwei aufeinanderfolgenden Iterationen $i-1$ und $i$ ist kleiner als eine vordefinierte Abbruchschwelle $\epsilon > 0$ für alle Argumente $a$ ($|\mathcal{B}_i(a) - \mathcal{B}_{i-1}(a)| < \epsilon$).

Die erste Bedingung führt zur Terminierung bei allen azyklischen SetAFs sowie bei zyklischen SetAFs, bei denen alle Argumente ab einem Ausführungszeitpunkt

paarweise unterschiedliche Belastungen erhalten. Das zweite Kriterium tritt bei den übrigen zyklischen SetAFs auf, bei denen mindestens zwei Argumente im SetAF eine ständig wechselnde aber zueinander gleiche Belastung haben. Dieses Kriterium ist bedeutend, um in solchen Fällen Endlosschleifen zu vermeiden.

## 5.2. Belastungen

Die Berechnung der Belastungswerte $\mathcal{B}_i$ über $i \in \mathbb{N}_0$ ist aus der Definition 3.1.1 ableitbar. Sie lässt sich mit der Funktion $BerechneBelastung$ im Pseudocodeausschnitt 3 beschreiben, die in jeder Iteration der Hauptschleife aufgerufen wird.

**Pseudocode 3** Berechnung der Belastungen

```
1:  procedure BERECHNEBELASTUNG(Ar)
2:      Konvergiert ← Wahr
3:      for all a ∈ Ar do
4:          a.TempBelastung ← 1
5:          for all S ∈ a⁻ do
6:              MaxBelastung ← 1
7:              for all s ∈ S do
8:                  if s.Belastung > MaxBelastung then
9:                      MaxBelastung ← s.Belastung
10:                 end if
11:             end for
12:             a.TempBelastung ← a.TempBelastung + 1/MaxBelastung
13:         end for
14:         if |a.Belastung − a.TempBelastung| > ϵ then
15:             Konvergiert ← Falsch
16:         end if
17:         a.IstEndbelastung ← a.Belastung == a.TempBelastung
18:     end for
19:     return Konvergiert
20: end procedure
```

Die Schleife in Zeile 3 läuft über alle Argumente $a \in Ar$. Jedes $a$ erhält initial die Belastung von 1 in Zeile 4. Sollte $a$ keine Angreifer besitzen, bleibt es bei diesem initialen Wert, wie durch die Definition von $\mathcal{B}_i$ beschrieben.

Andernfalls wird die Belastung von $a$ in $a.TempBelastung$ über alle Angreifermengen $S \in a^-$ akkumuliert. Das geschieht in der Schleife in Zeile 5, die über alle Angreifer $S \in a^-$ iteriert. Zu jedem dieser Angreifer wird in den Zeilen $6 - 11$ die maximale Belastung bestimmt. Das Inverse der maximalen Belastung wird letztlich auf den akkumulierten Belastungswert von $a$ dazuaddiert. Das entspricht der Formel in der Definition von $\mathcal{B}_i(a)$ für $i > 0$ und Argumente $a$ mit nichtleerer Angreifermenge $a^-$.

Für jede neu berechnete Belastung wird geprüft, ob sie sich von der vorherigen um mehr als $\epsilon$ unterscheidet. Ist das für mindestens ein Argument der Fall, gibt die Funktion $Falsch$ zurück. Nur im gegensätzlichen Fall wird $Wahr$ zurückgegeben, was der aufrufenden Stelle das erreichte $\epsilon$-Abbruchkriterium signalisiert.

Zuletzt wird über die Zuweisung in der Zeile 17 die Variable $IstEndbelastung$ von entsprechenden Argumenten verwaltet. Sie wird mit $Wahr$ befüllt, falls ein Argument über zwei Iterationen eine identische Belastung aufweist, andernfalls mit $Falsch$.

## 5.3. Ränge

Die Ränge lassen sich über das Verfahren im Codeausschnitt 4 berechnen. Die Korrektheit davon hängt von der Funktionsweise der Sortierfunktion in Zeile 2 ab. Es handelt sich bei dieser um ein beliebiges, stabiles, vergleichsbasiertes Sortierverfahren mit einer linearithmischen Laufzeit. Für dieses Verfahren müssen beim Vergleich von zwei Argumenten $a$ und $b$ zuerst ihre $MinRang$- und $MaxRang$-Werte verglichen werden ($a.MaxRang < b.MinRang \Rightarrow a \succ_{\mathcal{B}} b$ und umgekehrt). Falls sich hier noch keine Aussage treffen lässt, erfolgt der Vergleich anhand der Belastungswerte. Die sortierten Elemente in $ArSrt$ seien mit $a_1 \dots a_{|Ar|}$ bezeichnet.

**Pseudocode 4** Berechnung der Ränge anhand der Belastungen

```
 1: procedure BERECHNERANG(Ar)
 2:     ArSrt ← sort(Ar ∪ Dummy)
 3:     AlleRaengeFertig ← Wahr
 4:     min ← 1
 5:     for all i ∈ 2 ... |ArSrt| do
 6:         IstVorigeBlstGleich ← a_i.Belastung == a_{i−1}.Belastung
 7:         if IstVorigeBlstGleich ∧ a_{i−1}.MaxRang ≥ a_i.MinRang then
 8:             if ¬a_{i−1}.IstEndbelastung then
 9:                 AlleRaengeFertig ← Falsch
10:             end if
11:             Continue
12:         end if
13:         for all j ∈ min ... i do
14:             a_j.MinRang ← min
15:             a_j.MaxRang ← i − 1
16:         end for
17:         min ← i
18:     end for
19:     return AlleRaengeFertig
20: end procedure
```

Vor der Sortierung der Argumente wird die Menge $Ar$ um das Element $Dummy$

erweitert. Hierbei handelt es sich um ein künstliches Argument mit einer maximalen Belastung, sodass es bei der Sortierung an das Ende der Liste gestellt wird. Das vereinfacht die Implementierung, da der Algorithmus in seiner Schleife nicht das aktuelle Argument bearbeitet, sondern nur diejenigen aus vorigen Iterationen. Ohne diesen Platzhalter wäre eine Sonderbehandlung für das letzte Argument notwendig.

Die Variable $AlleRaengeFertig$ in Zeile 3 speichert, ob das Abbruchkriterium der Rangeindeutigkeit aus Unterkapitel 5.1 erreicht wurde. Es wird mit $Wahr$ initialisiert und erst beim Entdecken von Argumenten mit einem noch nicht eindeutigen Rang auf $Falsch$ zurückgesetzt.

Die Idee des Algorithmus lässt sich wie folgt beschreiben: Mit $min = 1$ wird ein Anfang für das $[MinRang = min, MaxRang]$-Intervall festgelegt. Danach werden die Argumente in der sortierten Reihenfolge iteriert, angefangen bei dem $min +$ 1-ten Argument. In jeder Iteration $i > min$ wird geprüft, ob $a_{i-1} \succ_{\mathcal{B}} a_i$ gilt. An dieser Stelle sind zwei Möglichkeiten gegeben – entweder gilt die Relation oder die Information ist noch unbekannt.

Steht die Geltung von $a_{i-1} \succ_{\mathcal{B}} a_i$ *nicht* fest, wird $AlleRaengeFertig$ auf $Falsch$ gesetzt, da sich in diesem Fall noch keine Information zum Verhältnis zwischen $a_{i-1}$ und $a_i$ bestimmen lässt. Die Iteration wird in diesem Fall weiter mit dem $i + 1$-Element auf gleiche Weise fortgeführt ($Continue$-Befehl in Zeile 9).

Falls $a_{i-1} \succ_{\mathcal{B}} a_i$ in der $i$-ten Schleifeniteration gilt, haben sich zwischen $min$ und $i - 1$ Argumente akkumuliert, die über die gesamte bisherige Ausführung gleich belastet wurden. In der Schleife in den Zeilen $11 - 14$ erhalten sie entsprechend alle das gleiche Rangintervall $[MinRang = min, MaxRang = i-1]$. Der Fall $min = i-1$ entspricht dabei einem eindeutig bestimmten Rang für das $i - 1$-te Argument. Die Variable $min$ wird dann im letzten Schritt auf $i$ gesetzt. Damit wird weiter auf die gleiche Art verfahren und die nächste Folge von Argumenten mit (noch) gleichem Rang berechnet.

## 5.4. Terminierung

Ob der definierte Algorithmus terminiert, ist von den beiden Abbruchkriterien abhängig. Für jede SetAF-Instanz muss mindestens eines der Kriterien nach einer endlichen Iterationszahl zutreffen. Es können leicht Graphen erstellt werden, bei denen die Bedingung der Rangeindeutigkeit nie erfüllt wird. Das einfachste Beispiel ist das zyklische SetAF $AF_{zyklus} = (Ar, att)$ mit $Ar = \{A, B\}$ und $att = \{(\{A\}, B), (\{B\}, A)\}$.

Zu prüfen ist somit, ob das $\epsilon$-Kriterium für solche Instanzen die Ausführung abbricht. Das ist genau dann der Fall, wenn die Folge der Belastungen $\mathcal{B}_i(a)$ für $i \in \mathbb{N}_0$ und alle Argumente $a$ gegen einen Grenzwert konvergiert.

Die Folge beginnt mit $\mathcal{B}_0(a) = 1$, was gleichzeitig die minimale mögliche Belastung ist. Der nächste Wert $\mathcal{B}_1(a) = 1 + |a^-|$ entspricht der maximal möglichen Belastung von $a$. Im ungünstigsten Fall für die Konvergenz negiert jede Iteration

$i$ die vorherige Iteration $i-1$, sodass die Belastungen bei geraden Iterationen nah an 1 bleiben und bei ungeraden an $1+|a^-|$. Dieser Effekt wird durch Ketten von möglichst vielen Angreifern erreicht, die wiederum ihrerseits viele Angreifer haben usw.

Das lässt sich mit dem Beispiel verdeutlichen, in dem $a$ keine Angreifer hat. In diesem Fall tritt die Konvergenz bereits in der Iteration 0 ein. Besitzt $a$ eine unendliche Kette von stets einem Angreifer und Verteidiger (z. B. $AF_{zyklus}$), so entsprechen die Belastungen $\mathcal{B}_i(a)$ der Folge $x_i = 1 + \frac{1}{x_{i-1}}$ für $i \in \mathbb{N}$ und $x_0 = 1$. Für beide Beispiele lässt sich eine Verallgemeinerung herleiten. So wird angenommen, ein Argument $a$ wird von $c \in \mathbb{N}_0$ Argumentenmengen angegriffen. Diese Angreifer haben ihrerseits jeweils $c$ Angreifer. Die Angreiferkette wird auf diese Art unendlich fortgeführt. Ein solches SetAF lässt sich am unkompliziertesten aus zwei Argumenten $a, b$ konstruieren, die sich gegenseitig $c$ Mal angreifen. Die Belastungen von $\mathcal{B}_i(a)$ entsprechen dann der konvergenten Folge $x_i = 1 + \frac{c}{x_{i-1}}$ mit $i \in \mathbb{N}$ und $x_0 = 1$.

Für ein beliebiges SetAF $AF = (Ar, att)$ definiere nun $c := \max_{a \in Ar}(|a^-|)$. Dabei ist ersichtlich, dass für alle Argumente $a \in Ar$ die Belastungen $\mathcal{B}_i(a)$ dann mindestens so schnell konvergieren wie die genannte Folge $x_i = 1 + \frac{c}{x_{i-1}}$, $i \in \mathbb{N}$, $x_0 = 1$.

Die Konstante $\epsilon$ definiert das Konvergenzkriterium für die Folge. In anderen Worten wird die Ausführung des Algorithmus genau dann abgebrochen, wenn die Belastungsunterschiede zwischen zwei Iterationen $i$ und $i+1$ für alle Argumente unter $\epsilon$ liegen. Das impliziert eine steigende Laufzeit bei kleineren Werten für $\epsilon$. Gleichzeitig besteht bei größeren $\epsilon$-Werten die Gefahr, dass die Ausführung zu früh abgebrochen wird. In dem Fall können mehrere Argumente potentiell den gleichen Rang erhalten, obwohl bei einer längeren Ausführung Unterschiede in ihren Belastungen festgestellt werden könnten.

Die Wahl eines passenden Wertes für $\epsilon$ ist nicht trivial und hängt von den konkreten Anforderungen an die Genauigkeit und Ausführungsgeschwindigkeit von dem Algorithmus ab.

## 5.5. Laufzeit

Für die Bestimmung der Laufzeit des gesamten Algorithmus muss zunächst die Laufzeit der Funktionen $BerechneBelastung(Ar)$ und $BerechneRang(Ar)$ ermittelt werden.

### Berechnung der Belastungen

$BerechneBelastung(Ar)$ besteht aus drei verschachtelten Schleifen. Bei den anderen Anweisungen der Funktion handelt es sich um einfache Zuweisungen oder $if$-Verzweigungen, die sich konstant auf die Laufzeit auswirken. Die äußere Schleife iteriert über alle Argumente $a \in Ar$, die mittlere Schleife iteriert über alle Angreifer $S \in a^-$ und die innere Schleife läuft über alle Argumente $s \in S$. Die Angreifermen-

ge $a^-$ kann theoretisch so groß sein wie die gesamte Menge der Angriffe $att$ eines SetAF $AF = (Ar, att)$. Ebenso kann ein Angreifer $S \in a^-$ die gesamte Argumentenmenge $Ar$ enthalten. Eine naive Einschätzung zur Laufzeit der Funktion liegt damit bei $\mathcal{O}(|Ar| \times |att| \times |Ar|)$.

Diese Einschätzung ist allerdings zu pessimistisch. Ein Angreifer $S \in a^-$ wird nur in der Iteration der äußeren Schleife für das Argument $a$ besucht und in keiner anderen. Anhand dessen lässt sich schlussfolgern, dass die Verschachtelung der äußeren und mittleren Schleifen sich additiv und nicht multiplikativ auf die Laufzeit auswirkt. Insgesamt liegt die Laufzeit der Funktion somit in $\mathcal{O}(|Ar| + |att| \times |Ar|)$ $= \mathcal{O}(|att| \times |Ar|)$.

### Berechnung der Ränge

Eine naive Vorgehensweise für die Funktion $BerechneRang$ ist die Aufstellung einer Matrix $M$ der Größe $|Ar| \times |Ar|$. In dieser stellt jedes Element $m_{i,j} \in M$ die Relation zwischen $a_i$ und $a_j$ aus $Ar$ dar. Die möglichen Werte für $m_{i,j}$ wären $a_i \succcurlyeq_{\mathcal{B}} a_j$, $a_i \not\succcurlyeq_{\mathcal{B}} a_j$ oder *unbekannt*. In Kombination mit dem gegensätzlichen Element $m_{j,i}$ ist die vollständige Information zum Verhältnis zwischen $a_i$ und $a_j$ nach der Terminierung des Algorithmus gegeben. Dabei ist allerdings ersichtlich, dass dies für die Funktion $BerechneRang$ in einer Laufzeit von $\Omega(|Ar|^2)$ resultieren würde.

Aus diesem Grund wird für die Implementierung das Verfahren im Pseudocodeausschnitt 4 gewählt. Dieses nutzt für die Beschleunigung der Berechnung die Transitivität von $\succcurlyeq_{\mathcal{B}}$ und $\succ_{\mathcal{B}}$ aus. Für die Sortierung in der zweiten Zeile der Funktion werden $\mathcal{O}(|Ar| \times \log(|Ar|))$ Operationen benötigt. Die Zeilen 3 und 4 sind einfache Zuweisungen und laufen somit in konstanter Zeit. Die Schleife in Zeile 5 iteriert über $|Ar|$ Elemente. Innerhalb dieser Schleife besitzen die Befehle der Zeilen $6 - 10$ und 15 eine konstante Laufzeit. Die innere Schleife in den Zeilen $11 - 14$ dagegen ist ebenfalls linear zu $|Ar|$.

Eine erste Einschätzung zu dem gesamten Algorithmus wäre damit wieder $\mathcal{O}(|Ar|^2)$. Wie in der obigen Rechnung für die Belastungen kann auch hier begründet werden, dass diese Schätzung zu pessimistisch ist. Ein Argument, das in der Iteration $i$ der äußeren Schleife und Iteration $j$ der inneren Schleife besucht wird, kann bedingt durch das Verschieben der $min$ Variable in keiner anderen Iterationskombination auftauchen. Dadurch laufen beide Schleifen trotz Verschachtelung in insgesamt $\mathcal{O}(2 \times |Ar|) = \mathcal{O}(|Ar|)$ Zeit ab. Die Laufzeit von $BerechneRang$ wird somit durch die Sortierung bestimmt, die bei der Wahl von einem entsprechenden Sortieralgorithmus in $\mathcal{O}(|Ar| \times \log(|Ar|))$ liegt.

### Gesamter Algorithmus

Mit der Information zur Berechnung der Belastungen und Ränge kann nun der gesamte Algorithmus aus dem Codeausschnitt 2 analysiert werden.

Die Zeilen $1 - 6$ dienen der Initialisierung. Dabei ist ersichtlich, dass sie insgesamt eine zu $Ar$ lineare Laufzeit besitzen.

Anders verhält es sich mit der Hauptschleife. Falls das SetAF azyklisch ist und der längste Pfad darin die Länge $n$ besitzt, erreichen alle Belastungen spätestens ab der Iteration $n$ ihren endgültigen Wert. Somit terminiert der Algorithmus in $\mathcal{O}(n)$ Iterationen bedingt durch das Abbruchkriterium der Rangeindeutigkeit.

Bei zyklischen Graphen hängt die Anzahl der Iterationen von dem gewählten Wert für $\epsilon$ sowie vom Aufbau des Graphen ab. In diesem Fall liegt die Einschätzung zur Hauptschleife nicht im Umfang der vorliegenden Arbeit.

Das Innere der Hauptschleife beginnt mit der Berechnung der Belastungen mit der oben bestimmten Laufzeit von $\mathcal{O}(|att| \times |Ar|)$. Die Schleife in den Zeilen $9 - 11$ besitzt bezogen auf die Argumentenmenge $Ar$ eine lineare Laufzeit. In Zeile $12$ wird die Funktion $BerechneRang$ aufgerufen, die $\mathcal{O}(|Ar| \times \log(|Ar|))$ Anweisungen ausführt. Bei der $13$-ten Zeile handelt es sich um eine einfache Zuweisung mit einer konstanten Laufzeit.

Insgesamt lässt sich die Laufzeit des gesamten Algorithmus mit der folgenden Formel einschätzen (mit $n$ = Anzahl der Hauptschleifeniterationen):

$\mathcal{O}(|Ar| + n \times (\ |att| \times |Ar| \ + \ |Ar| \ + \ |Ar| \times \log(|Ar|)\ ))$
$= \mathcal{O}(n \times (\ |att| \times |Ar| \ + \ |Ar| \times \log(|Ar|)\ ))$.

# 6. Implementierung

Die Berechnung der Mengen-Belastungssemantik wurde bisher nur konzeptuell beschrieben. Den nächsten Schritt bildet eine funktionierende Implementierung in einer konkreten Programmiersprache. Mit einer praktischen Umsetzung lassen sich die theoretischen Überlegungen der vorigen Kapitel besser veranschaulichen, bestätigen oder widerlegen.

## 6.1. Architektur

Die wichtigste nichtfunktionale Anforderung (neben der offensichtlichen Korrektheitsanforderung) liegt in der Effizienz der Ausführung. Der Architekturentwurf muss sich daher hauptsächlich darauf fokussieren. Für den eigentlichen Algorithmus ist keine komplexe Architektur notwendig. Die einzige Herausforderung liegt in der Umsetzung der SetAF-Modellierung. Bei Betrachtung des Algorithmus wird ersichtlich, dass seitens der SetAF-Datenstruktur folgende Aktionen benötigt werden:

1. Zugriff auf alle Argumente des SetAF
2. Zugriff auf alle Angreifermengen eines gegebenen Arguments
3. Verwaltung von Metadaten zu jedem Argument (minimaler/maximaler Rang, Belastung der aktuellen Iteration, Belastung für die nächste Iteration usw.)

Eine konkrete Realisierung einer Datenstruktur für SetAFs existiert bereits in den Bibliotheken des Projekts *TweetyProject*[1]. Ihre Architektur besitzt viele Abhängigkeiten von anderen Modulen des Projekts. Aus diesem Grund ist deren vollständige Darstellung zu komplex und beinhaltet für die vorliegende Arbeit nicht relevante Elemente. In Abbildung 6.1 wird daher nur ein isoliertes und vereinfachtes Klassendiagramm dazu gezeigt.

Darin ist zu erkennen, dass die Klasse *SetAf* die zentrale Komponente darstellt. Sie beinhaltet alle Argumente und Angriffsmengen des Graphen. Die Klasse *Argument* ist für die Verwaltung der Argumentendaten zuständig. Die Daten von Angriffsmengen (insbesondere die Referenzen auf ihre Elemente) werden in den Objekten der Klasse *SetAttack* gehalten. Auch die Angriffsziele werden innerhalb der SetAttack-Objekte gespeichert.

Diese Modellierungsart muss auf die Eignung zu den obigen notwendigen Aktionen untersucht werden.

[1] `https://tweetyproject.org/index.html`

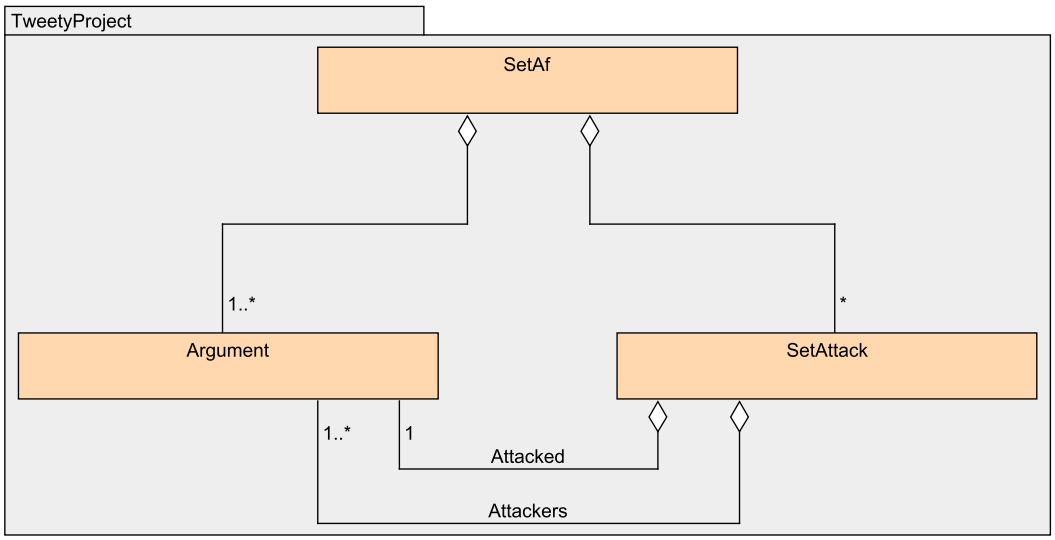



Abbildung 6.1.: Klassendiagramm – TweetyProject

1. Zugriff auf alle Argumente des SetAF:

   Der Zugriff und damit das Iterieren über die Argumente ist trivial möglich und läuft in der optimalen linearen Zeit.

2. Zugriff auf alle Angreifermengen eines gegebenen Arguments:

   In der vorhandenen TweetyProject-Implementierung wird der Zugriff auf die Angreifer von einem Argument durch eine lineare Suche über alle Angriffe des SetAF gelöst. Dabei wäre ein direkter Zugriff auf die Angreifer eines Arguments effizienter. Dieser ließe sich beispielsweise durch die Pflege von zusätzlichen Metadaten am SetAF lösen. Allerdings würde das den Speicherverbrauch erhöhen, sowie den initialen Aufbau des Graphen verlangsamen, da die Metadaten bei jedem neu eingefügten Angriff aktualisiert werden müssen.

3. Verwaltung von Metadaten zu jedem Argument:

   Die Realisierung von SetAFs in TweetyProject ist nicht auf die Belastungssemantik ausgelegt. Daher müssen die Datenstrukturen für die Verwaltung von Belastungen erweitert oder außerhalb der TweetyProject-Strukturen geführt werden.

Bei den Punkten 2 und 3 ist die Architektur offensichtlich suboptimal. Für die Modifikation besteht die Möglichkeit, von den Klassen *SetAf* und *Argument* zu erben. Die Subklassen lassen sich um die gewünschten Features anreichern. Trotzdem

bleiben die in Punkt 2 beschriebenen Schwächen – Speicherverbrauch und langsamer Graphenaufbau – bestehen. Die SetAF Realisierung von TweetyProject wurde nicht für die in dieser Arbeit gestellten Anforderungen entworfen. Eine Erweiterung würde die Architektur zweckentfremden und die resultierende Modellierung würde mehrere Kompromisse eingehen.

Die obigen Ausführungen legen die Sinnhaftigkeit einer neuen SetAF-Implementierung nahe. Sie sollte sich auf die Anforderungen an die Berechnung der Mengen-Belastungssemantik konzentrieren. Das Klassendiagramm in Abbildung 6.2 illustriert grob einen entsprechenden Entwurf.

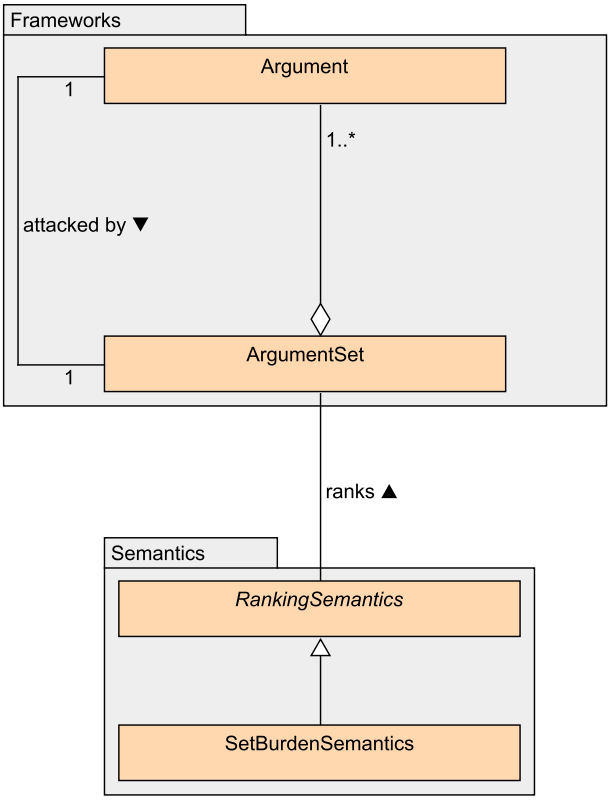



Abbildung 6.2.: Klassendiagramm – Vereinfachte Architektur

Bei dieser Modellierung entfällt die zentrale Verwaltungskomponente. Die Instanzen von *Argument* sind für die Verwaltung der Argument-spezifischen Information zuständig (Angreifer des Arguments eingeschlossen). Die Daten der Angreifermengen werden in den Objekten der Klasse *ArgumentSet* verwaltet. Eine Erkenntnis besteht darin, dass ein SetAF bei dieser Modellierungsart ebenfalls als eine Instanz von *ArgumentSet* ausgedrückt werden kann. Für andere Anwendungen ist es denkbar, dass ein SetAF-Objekt zusätzliche Eigenschaften benötigt, die *ArgumentSet* nicht anbietet. In dem Fall wäre die Erstellung einer *SetAF*-Klasse sinnvoll, die von *Argu-*

*mentSet* erbt und um die notwendigen Eigenschaften erweitert.

Für die drei notwendigen Aktionen lassen sich nun folgende Aussagen treffen:

1. Zugriff auf alle Argumente des SetAF:

   Zugriff und Iteration über alle Elemente eines SetAF sind trivial und schnell.

2. Zugriff auf alle Angreifermengen eines gegebenen Arguments:

   Die Angreifer von jedem Argument $a$ werden direkt in $a$ gespeichert. Damit ist der Zugriff auf sie direkt und ohne Suche möglich.

3. Verwaltung von Metadaten zu jedem Argument:

   Die neue Implementierung wird von Beginn an mit den notwendigen Metadaten ausgestattet.

Das vorgestellte Modell ist besser für die gegebene Aufgabe geeignet als das *SetAf* Modul in TweetyProject. Die Implementierung wird somit darauf basieren.

Bei der Skalierung der Aufgabe besteht keine Notwendigkeit für weitere komplexe Designentscheidungen. Der Planungsabschnitt wird daher mit diesem Entwurf abgeschlossen.

## 6.2. Implementierungsdetails

Die erste praktische Entscheidung liegt in der Wahl der Programmiersprache. Der obige Architekturentwurf ist an einer objektorientierten Umsetzung ausgerichtet. Diese Tatsache kombiniert mit den hohen Anforderungen an die Performance motiviert die Wahl von `C++` für die Implementierung. Durch die Hardwarenähe erlaubt `C++` eine hohe Ausführungsgeschwindigkeit sowie volle Ressourcenkontrolle.

Die eigentliche Implementierung wird in drei Komponenten geteilt:

- Umsetzung von SetAFs
- Implementierung der Mengen-Belastungssemantik
- Hilfsfunktionalitäten

Für die ersten beiden Punkte werden nachfolgend mehrere Quellcodeausschnitte vorgestellt und erläutert. Es handelt sich dabei lediglich um die Codestellen, die für die korrekte Funktion der Implementierung relevant sind. Zwecks einer kompakten Darstellung wurden sie auf die funktionsbezogenen Teile gekürzt.

Bei dem dritten Punkt Hilfsfunktionalitäten handelt es sich um Features zum Parsen von SetAF-Instanzen aus entsprechenden Dateien. Des Weiteren werden in dem Modul diverse Operationen auf Zeichenketten implementiert. Diese Themen sind für die Arbeit nebensächlich, daher wird weiter nur auf die Implementierung der SetAFs und der Belastungssemantik eingegangen.

### 6.2.1. Umsetzung der SetAFs

Die Umsetzung der SetAFs folgt vollständig dem Architekturentwurf aus dem vorherigen Unterkapitel. Der Kern der Implementierung besteht aus den beiden Klassen *Argument* und *ArgumentSet*.

Innerhalb der Klasse *Argument* befinden sich die Variablen für den minimalen und maximalen Rang des Arguments (*mMinRank*, *mMaxRank*). Zusätzlich speichern Objekte der Klasse die aktuelle Belastung sowie temporär die Belastung für die nächste Iteration (*mCurrentEval*, *mNextEval*). Anhand dieser beiden Variablen wird der Wert der Variable *mConverged* ermittelt. Sie entspricht der Variable $IstEndbelastung$ aus dem Algorithmus-Kapitel. Zuletzt findet sich die Variable für die Speicherung von Angreifern des Arguments in der Klassenimplementierung (*mAttackers*).

Die Methoden der Klasse erlauben die Verwaltung obiger Variablen. Daneben werden einfache Hilfsmethoden zur Verfügung gestellt, beispielsweise eine Methode zur Umschaltung auf die nächste Iteration (Codestück 6.1).

```
1 void nextIteration()
2 {
3   mConverged = mNextEval == mCurrentEval;
4   mCurrentEval = mNextEval;
5   mNextEval = 0;
6 }
```

Listing 6.1: Nächste Iteration

Die Klasse *ArgumentSet* dient der Speicherung von Argumentenmengen. Bei den Objekten kann es sich um eine Angreifermenge oder um das gesamte SetAF handeln. Die Mengenelemente werden in der Variable *mArguments* gespeichert. Eine Hilfsvariable *mNrArgs* speichert die Größe der Argumentenmenge. Bei den Methoden der Klasse handelt es sich auch hier hauptsächlich um die Verwaltung beider Variablen. Die einzige Hilfsmethode *getMaxEval* erlaubt die Bestimmung der Belastung der Argumentenmenge (Codestück 6.2). Dazu werden alle Argumente durchlaufen und in Bezug auf ihre Belastungen verglichen. Das Ergebnis wird über die Variable *result* verwaltet.

```
getMaxEval()
{
  double result = 1.0;
  for(auto iterAtt = mArguments.begin();
      iterAtt != mArguments.end();
      ++iterAtt)
  {
    SetAF::Argument * arg = *iterAtt;
    if(arg->getEval() > result) { result = arg->getEval(); }
  }
  return result;
}
```

Listing 6.2: Maximale Belastung einer Argumentenmenge

### 6.2.2. Umsetzung der Semantik

Die Implementierung der Mengen-Belastungssemantik hält sich eng an den Pseudocode aus dem Algorithmus-Kapitel. Bei der Aufrufstruktur gibt es geringe Abweichungen, die keinen funktionalen Unterschied in der Berechnung verursachen. Im Codestück 6.3 wird die Methode zur Bestimmung der Belastung eines Arguments vorgestellt. Im Unterschied zum Pseudocode ist diese Methode auf ein Argument beschränkt. Sie wird entsprechend in einer Schleife aufgerufen, die über alle Argumente läuft.

```
computeBurden(SetAF::Argument * arg)
{
  double sum = 1.0;
  auto attackers = arg->getAttackers();
  for(auto iterAttSet = attackers.begin();
      iterAttSet != attackers.end();
      ++iterAttSet)
  {
    SetAF::ArgumentSet * attacker = *iterAttSet;
    double maxBurden = attacker->getMaxEval();
    sum += 1.0 / maxBurden;
  }

  arg->setNextEval(sum);

  double diff = sum - arg->getEval();
  return diff > 0 ? diff : -diff;
}
```

Listing 6.3: Belastungsbestimmung

Hier ist erkennbar, dass die Belastung in der Variable *sum* über alle Angreifermengen akkumuliert wird. Für die Bestimmung der Angreiferbelastungen wird auf die oben beschriebene Methode *getMaxEval* zurückgegriffen. Die Methodenrückgabe entspricht der Differenz zwischen den Belastungen von aktueller und nächster Iteration. An der aufrufenden Stelle wird diese Differenz benutzt, um das Erreichen vom $\epsilon$-Abbruchkriterium zu prüfen.

Im Codestück 6.4 wird die Methode zur Berechnung der Ränge anhand der Belastungen vorgestellt. Ihre Umsetzung entspricht fast einer direkten Übersetzung vom Pseudocode aus der Algorithmusdefinition nach C++. Der genannte Pseudocode wurde bereits ausführlich in seinem Kapitel 5.3 beschrieben. Auf eine erneute detaillierte Erläuterung der Funktionalität wird daher verzichtet.

```
computeRankings(std::vector<SetAF::Argument *> & args)
{
  std::stable_sort(
    args.begin(), args.end(), SetAF::argCompare);
  SetAF::Argument* dummy = addDummy(args);

  bool allRanksComputed = true;
  size_t min = 0;
  for(size_t current = 1;
      current < args.size();
      ++current)
  {
    auto prev = args[current-1];
    auto arg = args[current];

    bool prevEqual = prev->getEval() == arg->getEval();
    if(prevEqual && prev->getMaxRank() >= arg->getMinRank())
    {
      if(!prev->isConverged())
      {
        allRanksComputed = false;
      }
      continue;
    }

    for(size_t i = min; i < current; ++i)
    {
      args[i]->setMinRank(min);
      args[i]->setMaxRank(current-1);
    }
    min = current;
  }

  args.pop_back();
  delete dummy;

  return allRanksComputed;
}
```

Listing 6.4: Rangbestimmung

Die für die Sortierung verwendete Funktion *SetAF::argCompare* ist im Codestück 6.5 abgebildet. Sie wirkt sich auf die Sortierung so aus, dass bei der booleschen

Rückgabe *true* für die Parameter $a1$ und $a2$ das Argument $a1$ vor $a2$ einsortiert wird. Entsprechend wird $a2$ bei der Rückgabe von *false* vor $a1$ einsortiert.

```
1  argCompare(Argument * a1, Argument * a2)
2  {
3    long min1 = a1->getMinRank();
4    long max1 = a1->getMaxRank();
5    long min2 = a2->getMinRank();
6    long max2 = a2->getMaxRank();
7    if(max1 != -1 && min2 != -1)
8    {
9      if(max1 < min2) { return true; }
10     if(min1 >= max2) { return false; }
11   }
12
13   return a1->getEval() < a2->getEval();
14 }
```

Listing 6.5: Argumentenvergleich für die Sortierung

# 7. Evaluation der Implementierung

Im vorigen Kapitel wurde die Implementierung der Belastungssemantik erläutert. Ihre Korrektheit wurde bereits in Kapitel 5 begründet. Auch zur asymptotischen Laufzeit wurden im gleichen Kapitel Aussagen gemacht.

Diese theoretische Analyse wird im Weiteren um eine praktische Evaluation ergänzt. Sowohl für die Prüfung der Korrektheit als auch für die Performancemessung werden unterschiedliche Testinstanzen erzeugt. Diese werden daraufhin von dem implementierten Programm verarbeitet und die Ergebnisse demonstriert.

## 7.1. Korrektheit

Auch wenn die Abwesenheit von Fehlern in einem Programm nicht durch Tests bewiesen werden kann, sind die Tests dennoch nicht überflüssig. Selbst wenn eine vollständige Korrektheit nicht garantiert werden kann, können Tests die Korrektheit in bestimmten bedeutenden Szenarien demonstrieren.

Aus diesem Grund werden nachfolgend mehrere SetAF-Testinstanzen manuell erzeugt. Sie repräsentieren verschiedene Typen von Graphen, wie bspw. zyklische, azyklische oder Graphen mit Argumenten, die sich selbst angreifen. Diese SetAFs werden in einer handhabbaren Größe erzeugt, um die Ergebnisse der Algorithmusausführung mit vertretbarem Aufwand manuell prüfen zu können.

### 7.1.1. Testinstanzen

**Testinstanz 7.1.1.** Als eine einfache Testinstanz zur Prüfung der Standardfälle dient das SetAF $AF_1 = (Ar, att)$ mit $Ar = \{A_1, A_2, B_1, B_2, B_3, C_1\}$ und $att = \{(\{A_1, A_2\}, B_2), (\{B_1, B_2, B_3\}, C_1)\}$. Die Abbildung 7.1 zeigt die grafische Darstellung von $AF_1$. Dieses Beispiel dient hauptsächlich der Sicherstellung von drei Aussagen:

- Die nichtangegriffenen Argumente $A_1, A_2, B_1, B_3$ erhalten den gleichen Rang.
- $C_1$ wird besser bewertet als $B_2$, da die Angreifermenge von $B_2$ stärker ist.
- Der Algorithmus terminiert durch das Kriterium der Rangeindeutigkeit.

Die erwarteten Ränge nach der Ausführung lauten wie folgt: $A_1, A_2, B_1, B_3$ teilen sich den ersten Rang. $C_1$ erhält den Rang $5$ und $B_2$ den Rang $6$.

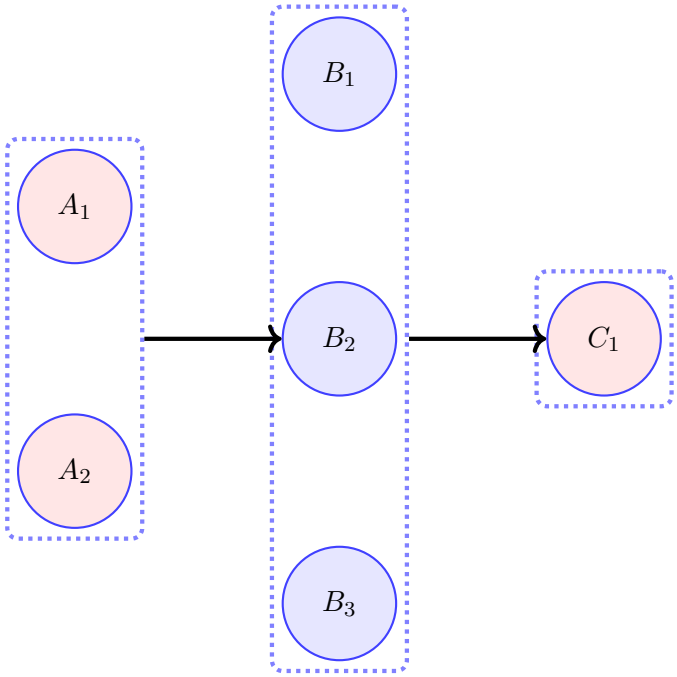



Abbildung 7.1.: Testinstanz $AF_1$

Das obige Beispiel zeigt ein azyklisches SetAF. Auch zyklische Variationen müssen getestet werden. Die einfachste Variante eines zyklischen SetAF besteht aus einem einzigen Argument $A$, das von der Menge $\{A\}$ angegriffen wird. Eine algorithmische Verarbeitung dessen entspricht jedoch einer Sortierung von einem Element und ist damit uninteressant. Das erste zyklische SetAF für die Testzwecke besteht daher aus zwei Argumenten und wird in der Testinstanz 7.1.2 definiert.

**Testinstanz 7.1.2.** $AF_2 = (Ar, att)$ sei ein SetAF mit $Ar = \{A_1, A_2\}$ und $att = \{(\{A_1\}, A_2), (\{A_2\}, A_1)\}$.

In diesem Beispiel werden das $\epsilon$-Abbruchkriterium des Algorithmus für zyklische Graphen sowie die korrekte Einordnung der beiden Argumente geprüft.

Im Ergebnis ist hier der gleiche Rang für beide Argumente zu erwarten.

Die Zyklen ungerader Länge sind ebenfalls interessant, da die Argumente innerhalb solcher Zyklen indirekt sich selbst angreifen. Damit wird die nachfolgende Testinstanz 7.1.3 begründet.

**Testinstanz 7.1.3.** $AF_3 = (Ar, att)$ sei ein SetAF mit $Ar = \{A_1, A_2, A_3\}$ und $att = \{(\{A_1\}, A_2), (\{A_2\}, A_3), (\{A_3\}, A_1)\}$.

Anhand dieses Beispiels werden das $\epsilon$-Abbruchkriterium des Algorithmus für zyklische Graphen mit Zyklen ungerader Länge und die korrekte Einordnung der Argumente geprüft.

Auch in dieser Situation wird für alle Argumente der gleiche Rang erwartet.

Ein Argument kann auch Teil mehrerer Zyklen sein. Das Beispiel 7.1.4 definiert ein SetAFs mit einem derartigen Argument.

**Testinstanz 7.1.4.** $AF_4 = (Ar, att)$ sei ein SetAF mit $Ar = \{A_1, A_2, A_3, A_4, A_5\}$ und $att = \{(\{A_1\}, A_2), (\{A_2\}, A_3), (\{A_3\}, A_1), (\{A_3\}, A_4), (\{A_4\}, A_5), (\{A_5\}, A_3), \}$. Das SetAF wird in der Abbildung 7.2 gezeigt.

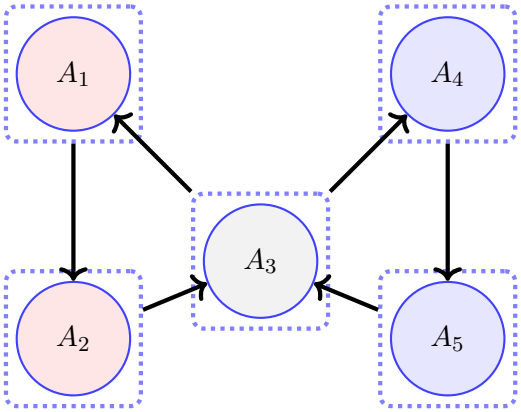



Abbildung 7.2.: Testinstanz $AF_4$

Das Argument $A_3$ ist Teil von zwei Zyklen. Von der Implementierung wird erwartet, dass $A_3$ den letzten Rang erhält, da es als einziges Argument zwei Angreifer besitzt. Das wiederum schwächt den Angriff von $A_3$ auf $A_1$ und $A_4$ ab. Damit sollten sie den ersten Rang teilen, was für $A_2$ und $A_5$ den gemeinsamen dritten Rang bedeutet. Da die Belastungen bei diesem SetAF aufgrund der Zyklen ständig geändert werden und zwei Argumentenpaare mit gleichem Rang existieren, wird ein Abbruch durch das $\epsilon$-Kriterium erwartet.

Die obigen zyklischen SetAFs bestehen lediglich aus dem eigentlichen Zyklus. In der Instanz 7.1.5 wird ein SetAF eingeführt, das neben dem Zyklus noch weitere Argumente beinhaltet.

**Testinstanz 7.1.5.** $AF_5 = (Ar, att)$ sei ein SetAF mit $Ar = \{A_1, A_2, A_3, A_4, A_5\}$ und $att = \{(\{A_1\}, A_2), (\{A_2\}, A_3), (\{A_3\}, A_1), (\{A_3\}, A_4), (\{A_5\}, A_1)\}$. Eine grafische Darstellung von $AF_5$ ist in der Abbildung 7.3 enthalten.
In diesem Beispiel wird die korrekte Einordnung der Argumente für komplexere zyklische Graphen mit Zyklen ungerader Länge überprüft. Die erwarteten Ränge nach der Ausführung lauten wie folgt: $A_5 \succ_{\mathcal{B}} A_2 \succ_{\mathcal{B}} A_4 \succ_{\mathcal{B}} A_3 \succ_{\mathcal{B}} A_1$. Bei einer korrekten Ausführung werden sie in der dritten Iteration – nach einem Abbruch durch das Rangeindeutigkeitskriterium – bestimmt.

Ein weiterer Sonderfall ist der von selbstangreifenden Argumenten. In der Testinstanz 7.1.6 wird ein entsprechendes SetAF eingeführt.

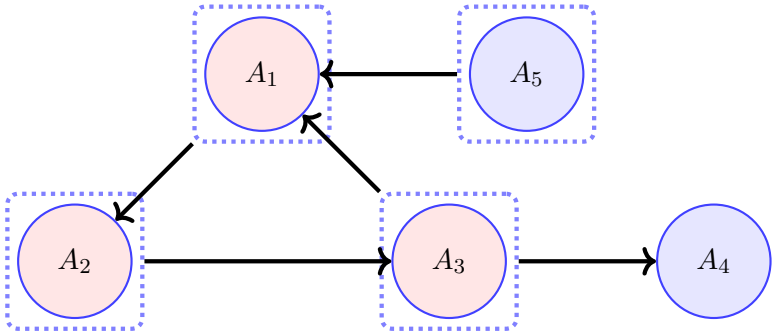



Abbildung 7.3.: Testinstanz $AF_5$

**Testinstanz 7.1.6.** $AF_6 = (Ar, att)$ sei ein SetAF mit $Ar = \{A_1, A_2, A_3, A_4\}$ und $att = \{(\{A_1\}, A_2), (\{A_2, A_3\}, A_3), (\{A_3\}, A_4)\}$. Das Beispiel ist in der Abbildung 7.4 dargestellt. Der Selbstangriff trifft für das Argument $A_3$ zu, das sich in seiner eigenen Angreifermenge $\{A_2, A_3\}$ befindet.

Wie im letzten Beispiel stehen auch hier alle Ränge nach der dritten Iteration fest und lauten folgendermaßen: $A_1 \succ_{\mathcal{B}} A_3 \succ_{\mathcal{B}} A_4 \succ_{\mathcal{B}} A_2$.

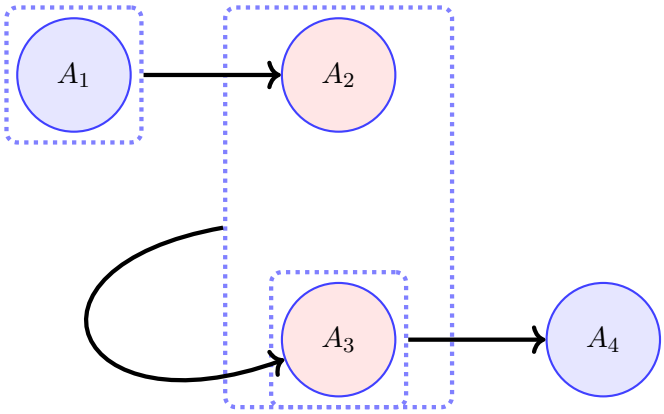



Abbildung 7.4.: Testinstanz $AF_6$

Zuletzt zeigt die Testinstanz 7.1.7 eine Vereinigung von zwei isomorphen SetAFs.

**Testinstanz 7.1.7.** $\widehat{AF_6} = (Ar, att)$ sei ein SetAF mit $Ar = \{\hat{A}_1, \hat{A}_2, \hat{A}_3, \hat{A}_4\}$ und $att = \{(\{\hat{A}_1\}, \hat{A}_2), (\{\hat{A}_2, \hat{A}_3\}, \hat{A}_3), (\{\hat{A}_3\}, \hat{A}_4)\}$. Dabei ist zu erkennen, dass $\widehat{AF_6}$ isomorph zu dem SetAF $AF_6$ aus dem vorherigen Beispiel ist. Das SetAF $AF_7$ sei nun definiert als $AF_6 \oplus \widehat{AF_6}$.

Diese Instanz unterscheidet sich von den vorherigen dadurch, dass hier Argumente existieren, die nicht durch einen Pfad miteinander verbunden sind. Von dem Algorithmus wird eine gleichgewichtete Wertung der isomorphen Argumente erwartet. Die korrekten Ränge, die von der Implementierung konkret erwartet wer-

den, sind nachfolgend aufgelistet:
$A_1$ und $\hat{A}_1$ teilen sich den ersten Rang.
$A_3$ und $\hat{A}_3$ teilen sich den dritten Rang.
$A_4$ und $\hat{A}_4$ teilen sich den fünften Rang.
$A_2$ und $\hat{A}_2$ teilen sich den letzten Rang.

### 7.1.2. Ergebnisse

Die nachfolgend dargestellten Ergebnisse wurden aus den Ausgaben des implementierten Programms extrahiert. Die Ausgaben umfassen Beginn und Ende jeder Iteration der Hauptschleife. Zusätzlich wird in jeder Iteration der aktuelle Kenntnisstand zu den Belastungen und Rangintervallen aller Argumente ausgegeben. Als $\epsilon$ wurde der Wert $0,0001$ gewählt. Diese Wahl reicht aus, um alle gegebenen Testinstanzen korrekt auszuwerten und den Algorithmusablauf zu demonstrieren.

**Testinstanz 7.1.1** Die Tabelle 7.1 zeigt den Zustand aller Argumente nach der Beendigung jeder Iteration.

| $i$ | Belastungen $\mathcal{B}_i$ | | | | | | Rangintervalle | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | $A_1$ | $A_2$ | $B_1$ | $B_2$ | $B_3$ | $C_1$ | $A_1$ | $A_2$ | $B_1$ | $B_2$ | $B_3$ | $C_1$ |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | $1-6$ | $1-6$ | $1-6$ | $1-6$ | $1-6$ | $1-6$ |
| 1 | 1 | 1 | 1 | 2 | 1 | 2 | $1-4$ | $1-4$ | $1-4$ | $5-6$ | $1-4$ | $5-6$ |
| 2 | 1 | 1 | 1 | 2 | 1 | 1,5 | $1-4$ | $1-4$ | $1-4$ | 6 | $1-4$ | 5 |

Tabelle 7.1.: Evaluierung der Testinstanz 7.1.1

In der ersten Iteration nach der Initialisierung werden die Argumente $A_1, A_2, B_1, B_3$ in das Rangintervall $1-4$ und die Argumente $B_2$ und $C_1$ in das Intervall $5-6$ eingeordnet. Beides entspricht der Erwartung, da die erste Iteration ausschließlich von der Anzahl der direkten Angreifer abhängt.

In der zweiten Iteration stellt die Implementierung fest, dass der Angriff auf $B_2$ stärker ist als auf $C_1$. Somit werden die Ränge von $C_1$ und $B_2$ entsprechend auf $5$ und $6$ gesetzt. Gleichzeitig erkennt der Algorithmus, dass die Belastungen sich bei den nichtangegriffenen Argumenten nicht geändert haben. Damit wird die Erfüllung des Rangeindeutigkeitskriteriums erkannt und die Verarbeitung endet mit dieser Iteration.

Im Endergebnis teilen sich die Argumente $A_1, A_2, B_1, B_3$ weiter das Intervall $1-4$. Das Argument $C_1$ nimmt Rang $5$ und $B_2$ Rang $6$ ein.

Diese Ränge wie auch der Ausführungsverlauf der Implementierung entsprechen vollständig den Erwartungen.

**Testinstanz 7.1.2** Über die gesamte Ausführungsdauer teilen sich die beiden Argumente des SetAF den gleichen Rangbereich $1-2$. Die Implementierung hat nach

der 11-ten Iteration das $\epsilon$-Kriterium erreicht. In der Tabelle 7.2 werden die Belastungen aller Iterationen gezeigt (auf fünf Nachkommastellen kaufmännisch gerundet). Eine manuelle Nachberechnung der Belastungen bestätigt die Korrektheit der Tabelle.

| $i$ | $\mathcal{B}_i$ von $A_1$ und $A_2$ | Unterschied zu $\mathcal{B}_{i-1}$ |
|---|---|---|
| 0 (Init.) | 1,00000 | - |
| 1 | 2,00000 | 1,00000 |
| 2 | 1,50000 | 0,50000 |
| 3 | 1,66667 | 0,16667 |
| 4 | 1,60000 | 0,06667 |
| 5 | 1,62500 | 0,02500 |
| 6 | 1,61538 | 0,00962 |
| 7 | 1,61905 | 0,00366 |
| 8 | 1,61765 | 0,00140 |
| 9 | 1,61818 | 0,00053 |
| 10 | 1,61798 | 0,00020 |
| 11 | 1,61806 | 0,00005 |

Tabelle 7.2.: Evaluierung der Testinstanz 7.1.2

Dabei ist zu erkennen, dass die Abweichung der Belastungen zwischen zwei aufeinanderfolgenden Iterationen zum ersten Mal in der elften Iteration unter $\epsilon = 0,0001$ fällt. Dieses Ergebnis zeigt somit auch eine erwartungsgemäße Funktion der Implementierung.

**Testinstanz 7.1.3** Die Ränge der drei Argumente sind über die gesamte Ausführung im Intervall $1 - 3$ geblieben. Die Belastungen und damit auch ihre Abweichungen zwischen den Iterationen sind identisch zu dem obigen Beispiel 7.1.2. Somit wird auch hier die $\epsilon$-Schwelle nach elf Iterationen erreicht. Dieses Ergebnis entspricht ebenfalls der Erwartung an den Algorithmus.

**Testinstanz 7.1.4** In der Tabelle 7.3 wird der Ausführungsverlauf des Algorithmus ausschnittsweise dargestellt. Auf die Auflistung aller Iterationen wird aus Platzgründen verzichtet.

Wie erwartet, wird $A_3$ unmittelbar in der ersten Iteration richtig eingeordnet. Die anderen Argumente erhalten in der darauffolgenden Iteration ihr endgültiges Rangintervall.

Das $\epsilon$-Kriterium wird in dieser Instanz nach dreizehn Iterationen erreicht, im Unterschied zu den vorigen zyklischen SetAFs. Für diese waren elf Iterationen notwendig, unabhängig von der Zykluslänge.

Daran wird deutlich, dass die Konvergenzgeschwindigkeit nicht durch die Zykluslänge bestimmt wird, sondern durch die Anzahl der sich überschneidenden

Zyklen.

| $i$ | Belastungen $\mathcal{B}_i$ | | | | | Rangintervalle | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| - | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ |
| 1 | 2 | 2 | 3 | 2 | 2 | $1-4$ | $1-4$ | 5 | $1-4$ | $1-4$ |
| 2 | $1,33$ | $1,5$ | 2 | $1,33$ | $1,5$ | $1-2$ | $3-4$ | 5 | $1-2$ | $3-4$ |
| | | | | | ... | | | | | |
| 13 | $1,46$ | $1,69$ | $2,19$ | $1,46$ | $1,69$ | $1-2$ | $3-4$ | 5 | $1-2$ | $3-4$ |

Tabelle 7.3.: Evaluierung der Testinstanz 7.1.4

**Testinstanz 7.1.5** Der Ausführungsverlauf zu diesem SetAF wird in Tabelle 7.4 gezeigt.

| $i$ | Belastungen $\mathcal{B}_i$ | | | | | Rangintervalle | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| - | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ |
| 0 | 1 | 1 | 1 | 1 | 1 | $1-5$ | $1-5$ | $1-5$ | $1-5$ | $1-5$ |
| 1 | 3 | 2 | 2 | 2 | 1 | 5 | $2-4$ | $2-4$ | $2-4$ | 1 |
| 2 | $2,5$ | $1,33$ | $1,5$ | $1,5$ | 1 | 5 | 2 | $3-4$ | $3-4$ | 1 |
| 3 | $2,67$ | $1,4$ | $1,75$ | $1,67$ | 1 | 5 | 2 | 4 | 3 | 1 |

Tabelle 7.4.: Evaluierung der Testinstanz 7.1.5

Die Berechnung wurde nach drei Iterationen beendet. Als Terminierungsgrund dient hier erneut das Rangeindeutigkeitskriterium.

Der erste Rang für das Argument $A_5$ sowie der letzte Rang für $A_1$ wurden bereits in der ersten Iteration nach der Initialisierung gesetzt. Das beruht darauf, dass die Argumente $A_2$, $A_3$ und $A_4$ jeweils einen Angreifer haben, während $A_1$ von zwei Argumentenmengen und $A_5$ nicht angegriffen wird.

Da $A_1$ als einziges Argument im Zyklus doppelt angegriffen wird, bricht es die „Symmetrie“ zwischen den Zykluselementen. Das wirkt sich in der zweiten Iteration auf den Rang von $A_2$ aus, das von $A_1$ angegriffen wird. Die Implementierung setzt den Rang von $A_2$ in diesem Schritt auf 2.

Die neue Bewertung von $A_2$ propagiert sich in der letzten Iteration weiter auf sein Angriffsziel $A_3$. Die Verstärkung dieses Angriffs bewirkt, dass $A_4$ höher als $A_3$ bewertet wird (Ränge 3 und 4 entsprechend).

Zusammengefasst ergibt sich $A_5 \succ_{\mathcal{B}} A_2 \succ_{\mathcal{B}} A_4 \succ_{\mathcal{B}} A_3 \succ_{\mathcal{B}} A_1$. Das entspricht der Erwartung, die bei Einführung dieser Testinstanz gestellt wurde.

**Testinstanz 7.1.6** In der Tabelle 7.5 werden die Evaluierungsergebnisse für die Selbstangriff-Instanz dargestellt.

In der ersten Iteration erhält $A_1$ den ersten Rang, da es das einzige nicht angegriffene Argument ist. Die anderen drei Argumente teilen sich das Intervall $2-4$, da sie

| $i$ | Belastungen $\mathcal{B}_i$ | | | | Rangintervalle | | | |
|---|---|---|---|---|---|---|---|---|
| - | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_1$ | $A_2$ | $A_3$ | $A_4$ |
| 0 | 1 | 1 | 1 | 1 | $1-4$ | $1-4$ | $1-4$ | $1-4$ |
| 1 | 1 | 2 | 2 | 2 | 1 | $2-4$ | $2-4$ | $2-4$ |
| 2 | 1 | 2 | 1,5 | 1,5 | 1 | 4 | $2-3$ | $2-3$ |
| 3 | 1 | 2 | 1,5 | 1,67 | 1 | 4 | 2 | 3 |

Tabelle 7.5.: Evaluierung der Testinstanz 7.1.6

alle jeweils einen Angreifer haben.

In der nächsten Iteration wird der Rang von $A_2$ entschieden. Da das Argument keine Verteidiger besitzt, erhält es mit 4 den letzten Rang.

In der letzten Iteration muss nur noch das Verhältnis zwischen $A_3$ und $A_4$ bestimmt werden. Dieses wird zugunsten von $A_3$ entschieden, da der Verteidiger $A_1$ von $A_3$ nicht angegriffen wird. Der Verteidiger $A_2$ von $A_4$ wird dagegen durch $A_1$ angegriffen.

Das endgültige Ergebnis der Berechnung ist somit $A_1 \succ_{\mathcal{B}} A_3 \succ_{\mathcal{B}} A_4 \succ_{\mathcal{B}} A_2$. Als Abbruchkriterium dient auch hier die Eindeutigkeit der Ränge von allen Argumenten.

Die Belastungen und Ränge jeder Iteration sowie die Terminierung sind auch in diesem Fall korrekt.

**Testinstanz 7.1.7** Die letzte Instanz ist die Vereinigung zweier isomorpher SetAFs, die dem SetAF aus dem vorigen Beispiel 7.1.6 entsprechen. Die Ausführungsergebnisse sind in Tabelle 7.6 zusammengefasst.

| $i$ | Belastungen $\mathcal{B}_i$ | | | | Rangintervalle | | | |
|---|---|---|---|---|---|---|---|---|
| - | $A_1$, $\hat{A}_1$ | $A_2$, $\hat{A}_2$ | $A_3$, $\hat{A}_3$ | $A_4$, $\hat{A}_4$ | $A_1$, $\hat{A}_1$ | $A_2$, $\hat{A}_2$ | $A_3$, $\hat{A}_3$ | $A_4$, $\hat{A}_4$ |
| 0 | 1 | 1 | 1 | 1 | $1-8$ | $1-8$ | $1-8$ | $1-8$ |
| 1 | 1 | 2 | 2 | 2 | $1-2$ | $3-8$ | $3-8$ | $3-8$ |
| 2 | 1 | 2 | 1,5 | 1,5 | $1-2$ | $7-8$ | $3-6$ | $3-6$ |
| 3 | 1 | 2 | 1,5 | 1,67 | $1-2$ | $7-8$ | $3-4$ | $5-6$ |
| 4 | 1 | 2 | 1,5 | 1,67 | $1-2$ | $7-8$ | $3-4$ | $5-6$ |

Tabelle 7.6.: Evaluierung der Testinstanz 7.1.7

Die Belastungen folgen den Werten aus dem Beispiel 7.1.6. In diesem Fall teilen jedoch immer zwei isomorphe Argumente ($A_i$ u. $\hat{A}_i$) die gleiche Belastung. Sie werden daher in das gleiche Intervall einsortiert, das mit dem Rang des entsprechenden Arguments aus der Testinstanz 7.1.6 korreliert.

Ein Effekt bei dieser Instanz ist, dass in der letzten Iteration beide Abbruchkriterien zutreffen, denn die Belastung von *allen* Argumenten ist über zwei Iterationen identisch geblieben.

Die oben beschriebenen Ergebnisse entsprechen auch hier einer korrekten Funktionsweise der Implementierung.

## 7.2. Performance

Die Methodik aus dem obigen Unterkapitel ist für die Messung der Performance nur eingeschränkt geeignet. Für aussagekräftige Messwerte muss eine große Anzahl von SetAFs mit unterschiedlichen Parameterkombinationen erstellt werden. Auf allen so erzeugten Instanzen muss die Implementierung des Algorithmus mehrfach ausgeführt werden. Auch bei der Ausführung sind Parameter variierbar.

Die tatsächlichen Belastungen und Ränge nach der Ausführung lassen sich mit einem vertretbaren Aufwand nicht überprüfen. Diese konkreten Lösungen sind allerdings an dieser Stelle weniger relevant, da der Schwerpunkt dieses Kapitels auf der Performancemessung liegt.

### 7.2.1. Generierung von Testinstanzen

Bei der Erzeugung der SetAFs sind die Variationen von mindestens drei Parametern interessant:

- Die Anzahl von Argumenten im SetAF
- Die Anzahl der Angreifer von jedem Argument
- Die Größe jeder Angreifermenge

Bei der Anzahl und Größe der benötigten SetAFs ist eine manuelle Erzeugung unrealistisch, daher wird auf eine automatisierte Generierung zurückgegriffen.

$Arg$ sei der Parameter für die Größe des SetAF, $Att$ der Parameter für die erwartete Anzahl der Angreifer für jedes Argument des SetAF und $AtGr$ der Parameter für die erwartete Größe jeder Angreifermenge. Die Erzeugung der Instanzen erfolgt dann nach dem folgenden Verfahren:

1. Erstellung der Argumentenmenge $Ar$ mit $Arg$ Argumenten.
2. Für jedes Argument $a \in Ar$ Erstellung der Menge $a^-$, wobei die Größe $|a^-|$ zufällig gewählt wird. Für die Generierung der Zufallszahl wird die Normalverteilung benutzt mit dem Erwartungswert $Att$ und der Standardabweichung $2$.
3. Für jede Angreifermenge $S \in a^-$ wird die Größe $|S|$ zufällig gewählt. Hier wird ebenfalls die Normalverteilung verwendet, wobei $AtGr$ als Erwartungswert dient und die Standardabweichung wieder $2$ beträgt.
4. Befüllung der Mengen $S \in a^-$ mit zufällig gewählten Argumenten aus $Ar$, bis sie ihre gewählte Größe erreichen.

Als konkrete Parameter für das obige Verfahren werden folgende Werte benutzt:

$$Arg \in \{10, 25, 50, 100, 250, 500, 1000\}$$
$$Att \in \{1, 2, 5, 10, 25, 50\}$$
$$AtGr \in \{1, 2, 5, 10, 25, 50\}$$

Die gewählten Werte ergeben $252$ Parameterkombinationen. Die Konfigurationen, bei denen die Größe von Angreifermengen die gesamte Argumentengröße des SetAF übersteigt, sind allerdings ungültig. Von solchen Fällen abgesehen werden zu jeder Parameterkombination hundert SetAFs erstellt. Viele der so erstellten Instanzen sind aus praktischer Sicht nicht sinnvoll. Ein Beispiel stellen die SetAFs dar, bei denen jedes Argument potentiell vielfach von der gesamten Argumentenmenge angegriffen wird (bspw. $Arg = Att = AtGr = 50$). Sie werden dennoch für die Evaluierung verwendet, da sie interessante Erkenntnisse über das tendenzielle Laufzeitverhalten liefern können.

### 7.2.2. Messung

Für die Performancemessung wird ein Windows-Computer mit einem AMD Ryzen™ 5 3600 Prozessor und 16 GB Arbeitsspeicher verwendet.

Um die Abhängigkeit der Laufzeit von $\epsilon$ zu erkennen, werden Messungen mit zehn äquidistanten $\epsilon$-Werten durchgeführt, die zwischen $0,0001$ und $0,001$ liegen. Neben der Zeit wird auch die Anzahl der Iterationen bis zur Terminierung gemessen.

Für jeden $\epsilon$-Wert und jede Testinstanz wird die Implementierung hundert Mal ausgeführt. Mit dieser Ausführungsart wird der störende Effekt von manchen Faktoren verringert. Als Beispiel für einen solchen Effekt kann eine kurzzeitige Systemauslastung durch andere Prozesse genannt werden. Diese fällt bei einer wiederholten Messung weniger ins Gewicht.

Trotzdem verhindern viele Störfaktoren eine exakte Messung. Die Systemauslastung ist niemals konstant und kann durch bestimmte Prozesse auch längerfristig erhöht sein. Ein einfaches Beispiel ist ein im Hintergrund laufendes Softwareupdate, das unter Umständen die Messung unbemerkt verzerren kann. Auch das automatische Heruntertakten der CPU durch thermische Effekte kann die Ergebnisse verändern.

Aus diesen Gründen dienen die Ergebnisse lediglich einer groben Orientierung in Bezug auf die Laufzeittendenzen.

### 7.2.3. Ergebnisse

Die Laufzeiten und ihre Erläuterungen werden nachfolgend in Abhängigkeit zu jedem der vier Parameter gezeigt.

**SetAF-Größe** $Arg$ Die Grafik 7.5 zeigt die Performance des Algorithmus bei unterschiedlich großen SetAFs. Als Zeitwert zu jedem $Arg$-Parameter wurde der Durchschnitt sämtlicher Messungen aller Kombinationen der anderen Parameter genutzt.

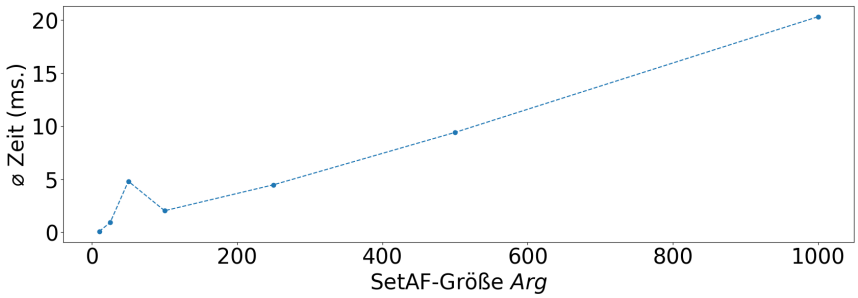



Abbildung 7.5.: SetAF-Größe zu Laufzeit

Es fällt eine überraschende Spitze im Bereich von bis zu $50$ Argumenten auf. Ein Blick auf die Grafik 7.6 erklärt den Grund für dieses Verhalten.

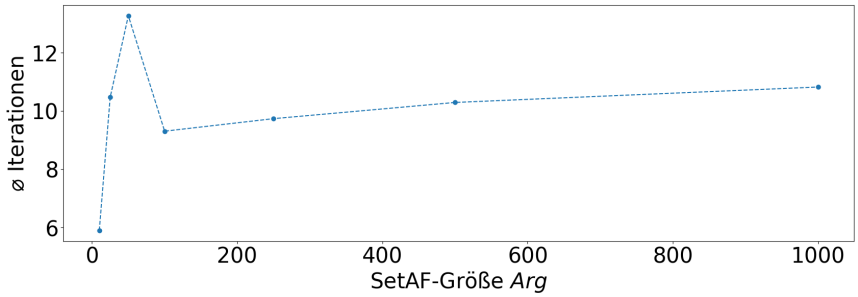



Abbildung 7.6.: SetAF-Größe zu Iterationen

Offensichtlich ist diese Laufzeit durch eine höhere durchschnittliche Anzahl von Iterationen verursacht. Dieser Effekt wiederum entsteht meistens bei den Testinstanzen mit Zyklen, in denen Argumente mit identischen Angreifer- und Verteidigerketten vorkommen. Diese Instanzen müssen durch das $\epsilon$-Abbruchkriterium terminiert werden, was in den meisten Fällen in mehr Iterationen resultiert.

Solche zyklischen Effekte nehmen mit steigendem Verhältnis von Angreifergrößen zu der Gesamtgröße des SetAF an Wahrscheinlichkeit zu. Ein Zyklus ist beispielsweise garantiert, wenn ein SetAF zwei Argumente enthält, die von der gesamten Argumentenmenge angegriffen werden[1]. Der gewählte Parameter $AtGr$ für die erwartete Angreifergröße liegt bei maximal $50$. Die ungünstigsten Verhältnisse von Angreifergrößen zu den SetAF-Größen treten also bei den Testinstanzen mit $Arg \leq 50$ auf.

[1] Für den trivialen Selbstangriff-Zyklus reicht sogar ein solches Argument.

Die „Anomalie“ erklärt sich somit durch die Wahl der gewählten Parameterverhältnisse und nicht durch die Besonderheiten von dem Algorithmus oder der Implementierung.

**Anzahl der Angreifer** $Att$ Die Laufzeitabhängigkeit von der Angreiferanzahl wird in der Abbildung 7.7 dargestellt.

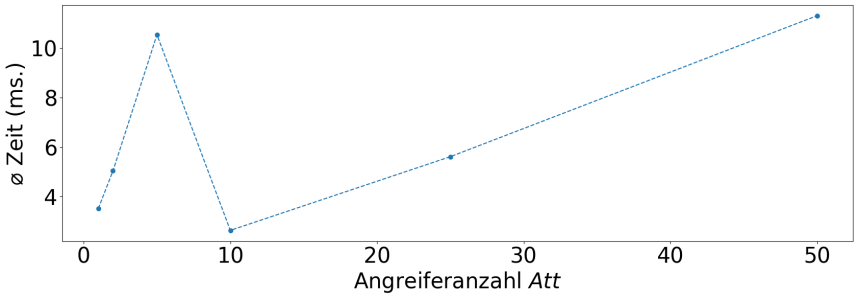



Abbildung 7.7.: Angreiferanzahl zu Laufzeit

Auch hier wird ein unerwarteter Effekt deutlich – die Laufzeiten bei den Angreiferzahlen von $1$, $2$ und $5$ sind länger als bei $10$. Die Messung für die Angreiferzahl $5$ benötigt sogar beinahe so viel Zeit wie die für $50$. Wie schon bei der SetAF-Größe lässt sich das durch ungünstiges zyklisches Verhalten erklären, wie an der Abbildung 7.8 erkennbar.

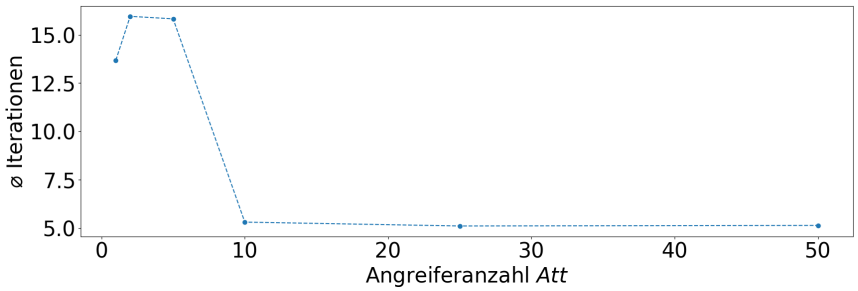



Abbildung 7.8.: Angreiferanzahl zu Iterationen

Von den gewählten Werten benötigt die Implementierung bei $2$ die meisten Iterationen, mit einem leichten Abfall bei $5$. Bei $10$ fällt die durchschnittliche Iterationanszahl unter $5$, wo es sich bei den nachfolgenden Werten nur noch geringfügig ändert.

Im Unterschied zu dem $Arg$-Parameter ist in diesem Fall nicht die Relation der Parameter zueinander für den Effekt verantwortlich, sondern die tatsächliche absolute Anzahl von Angreifern.

Mit steigender Angreiferzahl steigt auch die Wahrscheinlichkeit von Zyklen. Damit wächst auch die Wahrscheinlichkeit, dass die Belastungen von mehreren Ar-

gumenten sich ständig ändern, was tendenziell häufiger zum Abbruch durch das langsamere $\epsilon$-Kriterium führt.

Damit das $\epsilon$-Kriterium aber tatsächlich auftritt, reicht die ständige Veränderung der Belastungen nicht aus. Daneben müssen mindestens zwei Argumente im SetAF existieren, die über alle Iterationen die gleiche Belastung besitzen. Die Wahrscheinlichkeit dessen sinkt allerdings mit steigender erwarteter Angreiferanzahl. Das liegt daran, dass die Argumente seltener identische Ketten von Angreifern und Verteidigern haben.

Ab einer gewissen Angreiferanzahl überwiegt der zweite Effekt. In den generierten SetAFs wird die Ausführung bei den Instanzen mit $Att = 10$ nur noch in seltenen Fällen durch das $\epsilon$-Kriterium abgebrochen. Ab $Att = 25$ führt nur noch das Rangeindeutigkeitskriterium zum Eintritt der Terminierung.

Verantwortlich ist hier somit die randomisierte Generierungsart der Testinstanzen. Bei einer manuellen Erweiterung aller SetAFs um entsprechende Zyklen würde der Effekt der kleinen Angreiferzahlen verschwinden.

**Angreifergrößen** $AtGr$ Das Laufzeitverhalten in Abhängigkeit zu der erwarteten Angreifergröße ist der Abbildung 7.9 zu entnehmen.

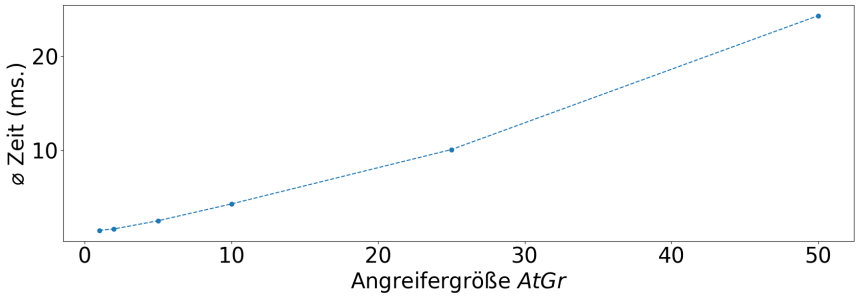



Abbildung 7.9.: Angreifergröße zu Laufzeit

Die durchschnittlich benötigten Iterationen sind in der Abbildung 7.10 ersichtlich.

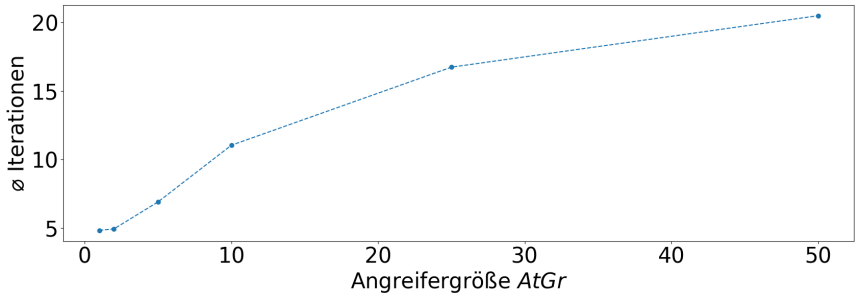



Abbildung 7.10.: Angreifergröße zu Iterationen

An dieser Stelle gibt es keine unerwarteten Effekte. Sowohl die Zeit als auch die

Anzahl der Iterationen steigen mit größeren Angreifermengen.

**Wert von $\epsilon$** Die Grafik 7.11 zeigt die Geschwindigkeit der Terminierung in Abhängigkeit des gewählten Wertes von $\epsilon$ .

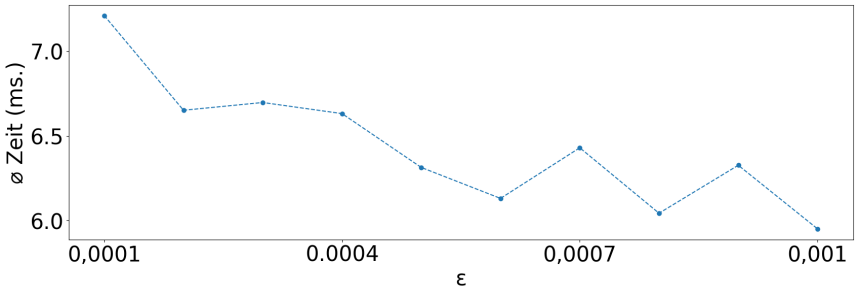



Abbildung 7.11.: $\epsilon$ zu Laufzeit

Das Laufzeitverhalten wird in dem Bereich durch Nebeneffekte verzerrt. Dennoch ist erkennbar, dass mit steigendem $\epsilon$ der Algorithmus tendenziell schneller terminiert. Das ist nicht überraschend, da die Unterschiede in den Belastungen zweier aufeinander folgender Iterationen schneller unter ein größeres $\epsilon$ fallen. Die Abbildung 7.12 bestätigt dies.

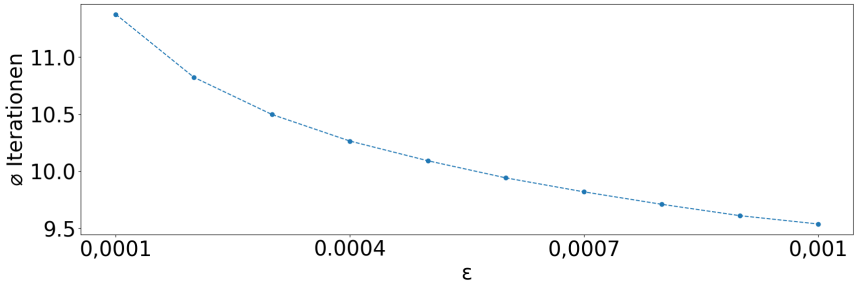



Abbildung 7.12.: $\epsilon$ zu Iterationen

Wie bei der Angreifergröße verhält sich der Algorithmus auch in Abhängigkeit vom $\epsilon$-Wert erwartungsgemäß.

# 8. Fazit

Das Ziel dieser Masterarbeit bestand in der Einführung einer Belastungssemantik $\mathcal{B}$ im Kontext der SetAFs. Neben der Einführung erfolgte auch eine detaillierte Untersuchung dieser Semantik. Untersucht wurden hierbei sowohl ihre mathematischen Eigenschaften als auch das tatsächliche Verhalten nach der Implementierung. Nachfolgend wird eine Zusammenfassung der Ergebnisse wiedergegeben und diskutiert.

## 8.1. Erkenntnisse

Im theoretischen Teil der Arbeit lag der Fokus hauptsächlich auf der Einführung von $\mathcal{B}$ und der Überprüfung davon hinsichtlich der Erfüllung von Rangsemantikeigenschaften. Das Endergebnis hat gezeigt, dass die Semantik sich ähnlich zu ihrem Ursprung verhält – Belastungssemantik für normale Argumentationsframeworks. Lediglich bei den Eigenschaften verlängerte Angriffserweiterung und verlängerte Verteidigungserweiterung fielen Abweichungen auf. Während diese Eigenschaften für die originale Semantik gelten, trifft das für $\mathcal{B}$ nicht zu. Die Meinung ist vertretbar, dass diese Diskrepanz durch den Begriff *Pfad* entsteht. Dieser hat unterschiedliche semantische Bedeutung für Argumentationsframeworks und SetAFs, zumindest im Kontext der Belastungssemantik. Ein Pfad von einem Argument $b$ zu $a$ in einem normalen Argumentationsframework stellt immer einen Angriff oder eine Verteidigung dar. Im Fall der SetAFs gilt diese Aussage nur dann, wenn der Pfad über die schwächsten Elemente der Angriffsmengen verläuft.

Dieser Zusammenhang ergibt sich dadurch, dass die Semantik $\mathcal{B}$ die Angriffskraft einer Menge über die Kraft ihres schwächsten Elements definiert. Eine derartige Berechnung der Angriffskraft ist damit begründbar, dass die Argumente der Angriffsmenge nur im Zusammenschluss einen Angriff hervorbringen. Trotzdem sind andere Berechnungswege denkbar. Drei Beispiele dafür wurden im Kapitel 4 vorgestellt. Der Grundgedanke bei allen drei war es, alle Argumente einer Angriffsmenge in die Berechnung der Angriffskraft miteinzubeziehen. Dadurch wird die problematische Abweichung in der Bedeutung von Pfaden entschärft.

Bei der Analyse der drei alternativen Belastungssemantiken hat sich bestätigt, dass diese Anpassung die erhoffte Wirkung zeigt. Alle erfüllen die verlängerte Angriffserweiterung und Verteidigungserweiterung. Allerdings scheitern sie an unterschiedlichen anderen Rangsemantikmerkmalen.

Die summenbezogene Semantik $\mathcal{B}^+$ zeigt dabei die meisten Unterschiede. Als einzige Semantik in dem Vergleich erfüllt sie den Kardinalität-Vorrang und den Verteidigungsvorrang nicht. Dieser Umstand entsteht durch die sinkende Angriffskraft

von größeren Angriffsmengen bereits in der ersten Iteration. Auch die Durchschnittsemantik $\mathcal{B}^{\varnothing}$ erfüllt nicht alle Eigenschaften, mit denen $\mathcal{B}$ zurechtkommt. Die Sinnhaftigkeit von $\mathcal{B}^{\varnothing}$ lässt sich auch unabhängig von den Eigenschaften hinterfragen. So ist es für die Semantik möglich, dass die Angriffskraft einer Menge mit der Hinzunahme von zusätzlichen Argumenten steigt. Das widerspricht dem Grundgedanken von Angriffsmengen.

Die Produktsemantik hat dagegen vergleichsweise gute Ergebnisse gezeigt. Sie erfüllt alle Eigenschaften, die $\mathcal{B}$ erfüllt, abgesehen von der (strikten) Gegentransitivität. Letztere wird jedoch von keiner der alternativen Semantiken erfüllt. Der Unterschied bei der Gegentransitivität entsteht dadurch, dass die Eigenschaft mit dem gleichen Verständnis der Angriffskraft von Mengen definiert wurde wie $\mathcal{B}$.

Neben einer theoretischen Untersuchung der Belastungssemantik für SetAFs wurde in dieser Arbeit eine praktische Evaluierung durchgeführt. Um den zeitlichen und inhaltlichen Rahmen nicht zu sprengen, konnte hier nur eine Variante der Semantik für die praktische Realisierung gewählt werden. Für die Vergleichbarkeit der Ergebnisse mit der vorhandenen Literatur (vor allem mit [YVC20]) ist die Wahl auf $\mathcal{B}$ gefallen.

Der erste Schritt für die Implementierung dieser Semantik ist ein formeller Algorithmusentwurf. Dieser wird in Kapitel 5 aufgegriffen. Darin werden Verfahren für die Berechnung von Belastungen und Rängen gezeigt sowie ihre Korrektheit begründet. Um die Terminierung des Algorithmus zu garantieren, wurden zudem zwei Abbruchkriterien eingeführt, wobei ein Erreichen von mindestens einem der beiden für beliebige SetAFs gezeigt wurde. Schließlich wird in dem Kapitel die Laufzeit des Verfahrens mit $\mathcal{O}(n \times ( \ |att| \times |Ar| \ + \ |Ar| \times \log(|Ar|) \ ))$ eingegrenzt (für bel. SetAFs $AF = (Ar, att)$).

Die Implementierung von dem Algorithmus dokumentiert das Kapitel 6. Hier werden praktische Entscheidungen genannt und begründet. Dazu gehört die Entscheidung, auf die vorhandene Implementierung der SetAFs in TweetyProject zu verzichten. Stattdessen wurde eine einfache Architektur entworfen, die sich speziell an die Implementierung der Belastungssemantik ausrichtet. Auf dieser Basis erfolgte die Umsetzung des Algorithmus aus dem Pseudocode nach `C++`.

Die Ausführung der Implementierung auf konkreten SetAF-Instanzen hat stets korrekte Ergebnisse geliefert. Im Hinblick auf die $\epsilon$-bezogene Konvergenzgeschwindigkeit wurden jedoch mehrere Verhaltensweisen deutlich, die vor der Evaluation nicht unbedingt offensichtlich waren. Die erste interessante Erkenntnis wurde bei der Berechnung der Lösungen für die zyklischen Instanzen gewonnen. Bei einem zweielementigen Zyklus war die gleiche Anzahl der Iterationen bis zum $\epsilon$-Abbruch wie bei einem dreielementigen Zyklus notwendig. Dadurch wird deutlich, dass die Zykluslänge keine Wirkung auf die Terminierungsgeschwindigkeit besitzt. Im Kontrast dazu hat die Berechnung mehr Iterationen für die Instanz erfordert, bei der sich zwei Zyklen ein gemeinsames Argument teilen.

In der Performancemessung ist weiter aufgefallen, dass die Verhältnisse zwischen der SetAF-Größe, Angreiferzahlen und Angreifergrößen ebenso bedeutend sind wie

die absoluten Größen dieser Parameter. Eine letzte erwähnenswerte Feststellung ist die steigende Anzahl von benötigten Iterationen bei kleineren Angreiferzahlen. Die letzten beiden Effekte, die bei der Messung der Geschwindigkeit aufgefallen sind gelten ausschließlich für eine randomisierte Generierung der SetAFs. Die realistischen Instanzen folgen meistens nicht den Mustern der zufällig erzeugten SetAFs. Somit ist es auch nicht zwangsläufig gegeben, dass das Programm bei richtigen Instanzen solche Verhaltensauffälligkeiten zeigt.

## 8.2. Offene Fragen

Der Bedeutungsunterschied von dem Begriff *Pfad* zwischen Argumentationframeworks und SetAFs wurde bereits angesprochen. Die Abweichung resultiert in einer eingeschränkten Vergleichbarkeit zwischen den Rangsemantiken der beiden Domänen. Aus diesem Grund sind Überlegungen zu alternativen Definitionen von einem Pfad in einem SetAFs interessant. Eine naheliegende Idee besteht darin, einen Pfad nur über die schwächsten Elemente der Angriffsmengen zu definieren. Diese Idee steht allerdings im Konflikt zu Semantiken, die auf andere Weisen die Angriffskraft von Mengen berechnen (bspw. $\mathcal{B}^*$). Ob eine allgemein sinnvolle Definition von einem Pfad möglich ist, kann in künftigen Untersuchungen geprüft werden.

Analog dazu kann auch zu der Gegentransitivität und der strikten Gegentransitivität argumentiert werden. Beide Eigenschaften sind auf die Berechnung der Mengenangriffskraft über das schwächste Element ausgerichtet. Diese Definition ist sinnvoll, allerdings sind abweichende Definitionen denkbar, die ebenfalls ihre Daseinsberechtigung haben. Solche Definitionen und ihre Abhängigkeiten von unterschiedlichen Rangsemantiken können auch Gegenstand weiterer Forschung sein.

Ein weiteres interessantes Thema ist die Übertragbarkeit der Ergebnisse aus [Del17] auf die Domäne der SetAFs. Die Gültigkeitsregeln zwischen den Semantikeigenschaften wurden darin für die normalen Argumentationframeworks beschrieben. In der vorliegenden Masterarbeit war die Eigenschaftenerfüllung zu diesen Regeln kompatibel. Die Untersuchung der Frage, ob das allgemein der Fall ist, war nicht Gegenstand der Masterarbeit. Die Beantwortung davon kann aber für die Analyse von anderen Semantiken hilfreich sein.

## 8.3. Reflexion

Im Anschluss an die Ergebnisbeschreibung folgt an dieser Stelle ein kritischer Rückblick auf die Masterarbeit.

Als Basis für die Untersuchungen diente der Artikel [YVC20]. Entsprechend waren viele Begriffe, Definitionen und Methoden eng an das Werk angelehnt. Das hat interessante Ergebnisse geliefert und einen Vergleich zwischen den beiden Arbeiten ermöglicht. Die Methodik in [YVC20] ist allerdings auf den nh-Categoriser ausgerichtet. An selektiven Stellen war es somit möglich, von der „Vorlage“ abzuweichen

und eigene, auf die Belastungssemantik ausgerichtete Wege zu gehen. Zwei Beispiele dafür wurden bereits im Unterkapitel 8.2 genannt – eigener *Pfad*-Begriff sowie eigene Definition der (strikten) Gegentransitivität.

Für beide Vorgehensweisen lassen sich Pro- und Kontraargumente finden und keine davon lässt sich als objektiv richtig bestimmen. Rückblickend drängt sich dennoch die Frage auf, ob ein Belastungssemantik-spezifisches Vorgehen stellenweise interessantere Ergebnisse geliefert hätte.

Im praktischen Teil der Arbeit wäre ebenfalls eine abweichende Methodik denkbar. Bestimmte Messungen zeigen Effekte, die mehr durch die Generierungsart der Testdaten als durch die Implementierung der Semantik verursacht werden. Diese Effekte lassen sich über angepasste Generierungsmethoden vermeiden. Die Entscheidung, die Messungen trotzdem in der Form darzustellen, wurde bewusst getroffen. Auch wenn die Ergebnisse dadurch verzerrt werden, zeigen die Verzerrungen interessante Effekte. Mit ihnen lassen sich Aussagen über den Algorithmus treffen, die sonst nicht offensichtlich wären. Ein anderer Zeitrahmen hätte jedoch eine Darstellung der Ergebnisse sowohl mit als auch ohne die Nebeneffekte erlaubt.

Die obigen Punkte zeigen potentielle Optimierungsmöglichkeiten für die durchgeführte Analyse. Von ihnen abgesehen sind bei der Verfolgung der Ziele der Masterarbeit keine weiteren Komplikationen aufgetreten. Zusammengefasst lassen sich die Ziele folgenden Ergebnissen gegenüberstellen:

- **Einführung einer Belastungssemantik im SetAF-Kontext:**

  Die Semantik wurde in vier begründeten Variationen vorgestellt.

- **Untersuchung der mathematischen Eigenschaften der neuen Semantik:**

  Alle eingeführten Semantiken wurden auf die Eigenschaften untersucht und miteinander verglichen.

- **Implementierung der neuen Semantik:**

  Die Semantik $\mathcal{B}$ wurde anhand voriger Untersuchungen für die Implementierung gewählt und in `C++` umgesetzt.

- **Evaluation der Implementierung:**

  Die Evaluierung wurde durchgeführt und hat korrekte Ergebnisse gezeigt. Daneben wurde das Laufzeitverhalten in Abhängigkeit von unterschiedlichen Parametern demonstriert.

# Anhang A.

# Kompilierung

Die Implementierung des Programms erfolgte in C++, Version 11. Ein erfolgreicher Kompilierungstest fand auf zwei Systemen statt:

- Desktop PC; Betriebssystem Windows 10; Prozessor AMD Ryzen™ 5 3600.
- Notebook; Betriebssystem Ubuntu 20.04.5 LTS; Prozessor AMD Ryzen™ 7 5700U.

Für die Kompilierung ist ein g++ Compiler für das entsprechende System erforderlich. Eine 64-Bit Kompilierung eignet sich besser zur Auswertung von großen SetAF-Instanzen. In Linux-Systemen ist ein passender Compiler bereits integriert. Unter Windows muss er nachinstalliert werden. Als eine praktische Option bietet sich die Standardinstallation von „MinGW“ in 64-Bit dafür an.

Zur Iteration durch alle Dateien in einem Pfad wurde die Bibliothek „dirent.h“ benutzt. Diese ist auf den meisten Linux-Systemen direkt verfügbar. Unter Windows ist sie ein Teil der Standardinstallation von MinGW. Sollten andere Werkzeuge zur Kompilierung benutzt werden, kann dirent.h bspw. von https://github.com/tronkko/dirent/blob/master/include/dirent.h bezogen werden (Stand 10.01.2023).

Abgesehen von dirent.h wurde auf eine Ausnutzung von plattformspezifischen Funktionalitäten nach Möglichkeit verzichtet. Die Erwartung ist somit, dass das Programm unter den meisten gängigen Systemen kompilierbar ist.

Die einfachste konkrete Möglichkeit dazu besteht darin, einen Python Interpreter und das dazu passende Bauwerkzeug „SCons“ zu installieren[1]. Im Quellcode-Hauptverzeichnis befindet sich eine passende Datei für Kompilierungsanweisungen – „SConstruct“. Ist eine Umgebung zur Kompilierung vorhanden, kann der Vorgang nach dem Wechsel in das Hauptverzeichnis mit dem Kommandozeilenbefehl „scons“ gestartet werden.

Das ausführbare Programm („Burden“ bzw. „Burden.exe“) wird nach einem erfolgreichen Bauvorgang in den Unterordner „Build“ kopiert.

[1] https://scons.org/ oder mit pip install scons

# Anhang B.
# Ausführung

Es gibt folgende Aufrufmöglichkeiten für das Programm:

- **Lösung von einem bestimmten SetAF.**

  Als erster Parameter muss „-saf“ für das saf-Dateiformat angegeben werden. Optional ist „-tgf“ für tgf-Dateien möglich, dieses Format unterstützt allerdings nur reguläre Argumentationframeworks. Diese werden dann als SetAF interpretiert, bei denen jede angreifende Menge aus einem Element besteht.

  Als zweiter Parameter wird der Name der Datei erwartet, die den Graphen beinhaltet.

  Optional kann an dritter Stelle „-v“ (für verbose) angegeben werden. Mit dieser Option wird der Rechenweg und das Endergebnis vollständig ausgegeben.

- **Performancemessung für mehrere SetAFs.**

  Als erster Parameter wird in diesem Fall „-m“ erwartet.

  An zweiter Stelle kommt die Anzahl der Messungen pro SetAF.

  Der dritte Parameter ist das Verzeichnis, in dem sich die Dateien befinden (für Messungen werden nur saf-Dateien unterstützt).

  Im letzten Parameter wird der Name der Ausgabedatei übergeben, in die das Programm die Ergebnisse der Messungen schreibt.

- **Berechnung der Lösungen für mehrere SetAFs.**

  Der erste Parameter ist „-s“.

  Danach wird der Pfad mit saf-Dateien erwartet.

  Zuletzt ist ein Name für die Ausgabedatei zu übergeben.

Das saf-Dateiformat wurde im Verlauf dieser Arbeit speziell zur Speicherung von SetAFs definiert. Eine saf-Datei speichert genau ein SetAF.

Ein SetAF lässt sich über das Format wie folgt beschreiben: Die erste Zeile beinhaltet eine Ganzzahl $n$, die die Anzahl der Argumente im SetAF angibt. Innerhalb der Datei werden die einzelnen Argumente über ihre Nummer referenziert. Die Nummerierung beginnt hierbei mit $0$ (bis $n - 1$). Auf Argumentennamen wurde

verzichtet, da das Format nur für die Belastungssemantik entworfen wurde, die die Abstraktion erfüllt.

Jede nachfolgende Zeile definiert die Angriffe auf jeweils ein Argument. Solche Zeilen beginnen mit der Nummer des angegriffenen Arguments, gefolgt von einem Semikolon. Dahinter stehen die Angreifermengen des Arguments, getrennt durch Semikola. Die Argumente der Angreifermengen werden darin durch Kommas getrennt gespeichert.

Beispiel für das SetAF $AF = (Ar, att)$ mit $Ar = \{A, B, C, D\}$, att = {({D}, A), ({B, C}D), ({D}D)}:

| Beispielhafter Inhalt einer saf-Datei |
|---|
| $4$ |
| $0; 3$ |
| $3; 1, 2; 3$ |

# Literaturverzeichnis


[ABN13] AMGOUD, Leila ; BEN-NAIM, Jonathan: Ranking-Based Semantics for Argumentation Frameworks. In: LIU, Weiru (Hrsg.) ; SUBRAHMANIAN, V. S. (Hrsg.) ; WIJSEN, Jef (Hrsg.): *Scalable Uncertainty Management*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013. – ISBN 978–3–642–40381–1, S. 134–147

[BCG18] BARONI, Pietro ; CAMINADA, Martin ; GIACOMIN, Massimiliano: Abstract Argumentation Frameworks and Their Semantics. In: *Handbook of Formal Argumentation*. London, England: College Publications, 2018, Kapitel 4, S. 159–236

[BDKM16] BONZON, Elise ; DELOBELLE, Jérôme ; KONIECZNY, Sébastien ; MAUDET, Nicolas: A Comparative Study of Ranking-based Semantics for Abstract Argumentation. In: *30th AAAI Conference on Artificial Intelligence (AAAI-2016)*. Phoenix, United States, Februar 2016

[BH01] BESNARD, Philippe ; HUNTER, Anthony: A logic-based theory of deductive arguments. In: *Artificial Intelligence* 128 (2001), Nr. 1, 203-235. http://dx.doi.org/https://doi.org/10.1016/S0004-3702(01)00071-6. – DOI https://doi.org/10.1016/S0004–3702(01)00071–6. – ISSN 0004–3702

[Cam06] CAMINADA, Martin: On the Issue of Reinstatement in Argumentation. In: FISHER, Michael (Hrsg.) ; HOEK, Wiebe van d. (Hrsg.) ; KONEV, Boris (Hrsg.) ; LISITSA, Alexei (Hrsg.): *Logics in Artificial Intelligence*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2006. – ISBN 978–3–540–39627–7, S. 111–123

[CLS05] CAYROL, Claudette ; LAGASQUIE-SCHIEX, Marie-Christine: Graduality in Argumentation. In: *J. Artif. Intell. Res. (JAIR)* 23 (2005), 03, S. 245–297. http://dx.doi.org/10.1613/jair.1411. – DOI 10.1613/jair.1411

[Del17] DELOBELLE, Jérôme: *Ranking-based Semantics for Abstract Argumentation*, Université d'Artois, Diss., 2017

[Dun95] DUNG, Phan M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. In: *Artificial Intelligence* 77 (1995), Nr. 2, 321-357. http://dx.doi.org/https://doi.org/10.1016/

0004-3702(94)00041-X. – DOI https://doi.org/10.1016/0004–3702(94)00041–X. – ISSN 0004–3702

[FB19] FLOURIS, Giorgos ; BIKAKIS, Antonis: A comprehensive study of argumentation frameworks with sets of attacking arguments. In: *International Journal of Approximate Reasoning* 109 (2019), 55-86. http://dx.doi.org/https://doi.org/10.1016/j.ijar.2019.03.006. – DOI https://doi.org/10.1016/j.ijar.2019.03.006. – ISSN 0888–613X

[MT08] MATT, Paul-Amaury ; TONI, Francesca: A Game-Theoretic Measure of Argument Strength for Abstract Argumentation. In: HÖLLDOBLER, Steffen (Hrsg.) ; LUTZ, Carsten (Hrsg.) ; WANSING, Heinrich (Hrsg.): *Logics in Artificial Intelligence*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2008. – ISBN 978–3–540–87803–2, S. 285–297

[NP06] NIELSEN, Søren H. ; PARSONS, S.: A Generalization of Dung's Abstract Framework for Argumentation: Arguing with Sets of Attacking Arguments. In: *ArgMAS*, 2006

[RL08] RAHWAN, Iyad ; LARSON, Kate: Pareto optimality in abstract argumentation. In: *AAAI* AAAI Press, 2008, S. 150–155

[YVC20] YUN, Bruno ; VESIC, Srdjan ; CROITORU, Madalina: Ranking-Based Semantics for Sets of Attacking Arguments. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34 (2020), Apr., Nr. 03, 3033-3040. http://dx.doi.org/10.1609/aaai.v34i03.5697. – DOI 10.1609/aaai.v34i03.5697