FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Argumentation Frameworks with Claims and Collective Attacks

Complexity Results and Answer-Set Programming Encodings

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Alexander Greßler, BSc
Matrikelnummer 01225159

an der Fakultät für Informatik

der Technischen Universität Wien

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

Wien, 23. Juli 2019

Alexander Greßler	Stefan Woltran

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Argumentation Frameworks with Claims and Collective Attacks

Complexity Results and Answer-Set Programming Encodings

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Alexander Greßler, BSc
Registration Number 01225159

to the Faculty of Informatics

at the TU Wien

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

Vienna, 23rd July, 2019

Alexander Greßler | Stefan Woltran

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Alexander Greßler, BSc

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

Wien, 23. Juli 2019

Alexander Greßler

Acknowledgements

I want to thank all the people who supported me through my studies. As most of them are native German speakers, I would like to compose the more comprehensive acknowledgements in German.

Danksagung

Ich möchte diese Gelegenheit nutzen, um mich bei all jenen zu bedanken, die mich während meines Studiums begleitet haben.

Zuerst gebührt mein Dank meinen Betreuern Stefan Woltran und Wolfgang Dvořák. Vielen Dank dafür, dass ihr euch bereit erklärt habt, meine Diplomarbeit zu betreuen. Ich habe viel von eurer Expertise während der Arbeit an diesem Projekt gelernt und eure Ratschläge waren eine große Hilfe.

Mein besonderer Dank gilt meinen Eltern Hildegard und Norbert, ohne deren fortwährende Unterstützung mein Studium in dieser Art sicherlich nicht möglich gewesen wäre.

Außerdem möchte ich mich bei allen Kommilitoninnen und Kommilitonen bedanken, die ich während des Studiums kennenlernen durfte. Im Speziellen möchte ich Florian, Sanja und Sören für die gemeinsam verbrachten Lerngruppen danken.

Abschließend möchte ich mich bei meinem guten Freund Erich bedanken, ohne den meine bisherige Studienzeit vermutlich anders verlaufen wäre.

Kurzfassung

Abstract argumentation frameworks, eingeführt von Dung in 1995, stellen einen der am weitesten verbreiten Formalismen zur Modellierung von Argumentationsprozessen im Bereich der künstlichen Intelligenz dar. Ein solches Framework modelliert den Argumentationsprozess als gerichteten Graphen, wobei die Knoten die Argumente repräsentieren und dessen gerichteten Kanten Attacken zwischen diesen Argumenten darstellen, während der innere Aufbau der Argumente abstrahiert wird. Als *Extensionen* bezüglich einer Semantik werden jene kohärenten Mengen an Argumenten bezeichnet, die bestimmte Eigenschaften erfüllen und zusammen akzeptiert werden. Nachdem daher das Framework aus einer Wissensbasis konstruiert wurde, hängt es nur noch von dieser abstrakten Struktur ab, welche Extensionen das Framework, bezüglich einer Semantik, besitzt. Wir betrachten im Folgenden zwei Generalisierungen solcher abstract argumentation frameworks.

Die erste Generalisierung die wir betrachten, *abstract argumentation frameworks with collective attacks*, erlaubt Attacken von Mengen von Argumenten auf Argumente. Dies ermöglicht es, Situationen auf eine sehr natürliche Art und Weise zu modellieren, in der eine Menge von Argumenten ein anderes Argument widerlegt, während jedes dieser Argumente einzeln dazu nicht im Stande wäre. Im Weiteren ermöglicht die zweite betrachtete Generalisierung, *claim augmented abstract argumentation frameworks*, Argumente mit *claims* zu augmentieren. Diese Augmentation ermöglicht es, die Extensionen anstelle von Argumenten in claims auszudrücken, welche von mehreren Argumenten geteilt werden können, wodurch auch die Abstraktion gelockert wird.

Im Fokus des ersten Teils der Arbeit steht die Komplexitätsanalyse dieser zwei Generalisierungen. Wir betrachten fünf Entscheidungsprobleme für diverse Semantiken für beide Typen von Frameworks und verorten sie in der Polynomialzeithierarchie.

Der zweite Teil dieser Arbeit ist der effizienten Berechnung der Extensionen solcher Frameworks gewidmet. Wir nutzen den weitverbreiteten Formalismus der *Answer-Set Programmierung* und präsentieren Kodierungen für diverse Semantiken, wobei wir für manche Semantiken unterschiedliche Möglichkeiten der Kodierung betrachten.

Abschließend befasst sich der dritte und letzte Teil mit den durchgeführten Experimenten. Wir präsentieren die gesammelten Daten und diskutieren die Resultate sowie die daraus gezogenen Schlüsse.

Abstract

Abstract argumentation frameworks, as proposed by Dung in 1995, constitute one of the most widely used formalisms to model argumentation processes in the field of artificial intelligence. An abstract argumentation framework models an argumentation process as a directed graph, representing arguments as vertices and attacks between those arguments as directed arcs between the respective vertices, abstracting away the inner structure of the arguments. Furthermore, the coherent sets of arguments that are jointly acceptable under a given semantics, satisfying certain properties, are commonly called *extensions*. Thus, after the argumentation frameworks has been constructed from a knowledge base, the set of extensions under a given semantics will be solely determined by this abstract structure. In the following, we consider two generalizations of such abstract argumentation frameworks.

The first generalization that we consider, *abstract argumentation frameworks with collective attacks*, allows for attacks to be between sets of arguments and arguments, enabling a natural way of modeling that a set of arguments might defeat another argument if considered together which, in general, is not the case if viewed individually. Furthermore, the second considered generalization, *claim augmented abstract argumentation frameworks*, introduces the augmentation of arguments using *claims*. Such augmented frameworks allow for an intuitive way of expressing the extension in terms of claims, that might be shared across multiple arguments, relaxing the abstraction to some extent.

In the first part of the thesis, we focus on the computational complexity of these generalizations. To this end, we consider five common decision problems as well as various semantics for both types of argumentation frameworks and locate their position on the polynomial hierarchy.

The second part of the thesis is devoted to the efficient computation of the extension of such argumentation frameworks. To achieve this, we make use of the well-established formalism of *answer-set programming* and give encodings for the various semantics, considering multiple approaches where applicable.

Finally, the third part is dedicated to the conducted experiments. We present our results and the acquired data and discuss our findings and the drawn conclusions.

Contents

CHAPTER 1

Introduction

The process of argumentation is one the most frequent types of communication between humans. Therefore, it seems natural to utilize the advances in artificial intelligence to assist us in this process.

1.1 Argumentation in artificial intelligence

Informally, "the study of argumentation may [...] be considered as concerned with how assertions are proposed, discussed, and resolved in the context of issues upon which several diverging opinions may be held" [BD07]. Argumentation has been enjoying increasing interests as core study within the field of artificial intelligence since the 1990s, including areas as defeasible reasoning, multi-agent systems and legal argumentation [vEGK+14]. As a non-monotonic formalism, its use-cases range from single-agent systems, evaluating their own knowledge base and trying to draw conclusions, to situations in which multiple agents argue over some common field of interest.

The argumentation process typically consists of multiple steps, usually starting with either some knowledge base or some document in natural language. If a knowledge base serves as starting point, one might construct the arguments and conflicts using classical logical [GH11]. In case of a document, the argument and conflicts might be constructed using argumentation mining. Such a mining process can, for example, leverage natural language processing to identify the arguments within a document as well as their internal structure and interactions between them [PM11]. Next, especially in abstract argumentation frameworks, one might abstract away from the internal structure of the arguments and use the obtained framework to identify coherent sets of arguments that are jointly accepted under some semantics and draw conclusions. This step is the main focus of this thesis, especially how to compute such sets of arguments and the computational costs associated in doing so. Finally, the extensions have to be reinterpreted in terms of the original knowledge base or document.

Abstract argumentation frameworks

A Dung abstract argumentation framework, as introduced by Dung in 1995 [Dun95], can be represented by a directed graph in which the vertices represent arguments, while the directed edges model the conflicts between them. Such a framework is referred to as "abstract", as it makes no assertion regarding the internal structure of the arguments. Furthermore, we consider two generalizations of such Dung abstract argumentation frameworks, which extend the definition of such frameworks.

The first generalization that we consider, abstract argumentation frameworks with collective attacks [NP06b], allows for attacks to be between sets of arguments and arguments, enabling a natural way of modeling that a set of arguments might defeat another argument if considered together which, in general, is not the case if viewed individually. Thus, such abstract argumentation frameworks with collective attacks differ from Dung abstract argumentation frameworks in terms of their attack relation. Furthermore, abstract argumentation frameworks with collective attacks have recently regained interest as topic of research [Pol17, DFW18, YVC18, FB19].

Moreover, claim augmented abstract argumentation frameworks [DW19], the second considered generalization, introduce the augmentation of arguments using *claims*. Thus, in addition to Dung abstract argumentation frameworks, each argument will be associated a claim, reflecting its inner structure. Such augmented frameworks allow for an intuitive way of expressing the extension in terms of claims, that might be shared across multiple arguments, relaxing the abstraction to some extent. Therefore, this process is closer to the pipeline mentioned previously and the resulting claims have to be reinterpreted in term of the original context from which they have been constructed. Recently, claim augmented abstract argumentation frameworks have attracted attention in use cases for which the less abstract handle is beneficial [BGR16, BGLR16, DW19].

One of the most fundamental problems for such argumentation systems is to determine which sets of arguments are considered to be the coherent sets of arguments that are jointly acceptable under some semantic for a given argumentation framework, often referred to as "extensions". Various semantics have been proposed in the literature [Dun95, BCG11, BDG11, Cam07, DDW13]. One common approach to solve such problems is to reduce the solving process to already well studied problems and use existing solving techniques to efficiently solve the novel problem. Thus, different formalisms have been studied to evaluate their suitability for the computation of extensions of Dung abstract argumentation frameworks. Most prominently, these include SAT-solving [BD04], QBF-solving [EW06], CSP-solving [AD13] as well as ASP-solving [EGW08]. Highly efficient solvers exist for the latter and the paradigm is well suited to encode argumentation frameworks, thus answer-set programming is the approach that we focus on. Furthermore, there exists a surprising relation between a class of claim augmented abstract argumentation frameworks and abstract argumentation frameworks with collective attacks. For some semantics, these coincide in term of their extensions and thus, the ASP encodings for one of them can be used to compute the extensions of the other.

1.2 Main contributions

Answer-set programming encodings already exist for Dung abstract argumentation frameworks [EGW08]. However, such encodings are still missing for abstract argumentation frameworks with collective attacks and claim augmented abstract argumentation frameworks. To ensure that answer-set programming is also a suitable for those generalizations, additional complexity analysis is required. Thus, we provide complexity results for abstract argumentation frameworks with collective attacks and complement the known complexity results for claim augmented abstract argumentation frameworks to cover all the semantics that we consider. Furthermore, we then provide answer-set programming encodings to enumerate the extensions of abstract argumentation frameworks with collective attacks and claim augmented abstract argumentation frameworks, considering multiple approaches where applicable.

1.3 Structure of the thesis

This thesis is divided into seven chapters, starting this with introduction as Chapter 1, followed by

- Chapter 2, formal background. In this section, we will introduce the notations and concepts required for the subsequent chapters. We will start with the basics of complexity theory in Section 2.1, followed by Section 2.2 introducing the syntax and semantics of answer-set programming and some extensions required. Finally, in Section 2.3, we will formally define argumentation frameworks as well as the considered semantics and decision problems. Moreover, this section will generalize the definitions for Dung abstract argumentation frameworks to also cover abstract argumentation frameworks with collective attacks and claim augmented abstract argumentation frameworks.

- Novel complexity results for the five decision problems introduced in Chapter 2 will be given in Chapter 3. In Section 3.1, we will present our results for abstract argumentation frameworks with collective attacks, while Section 3.2 will be dedicated to the results for claim augmented abstract argumentation frameworks.

- In Chapter 4, we will present our answer-set programming encodings to compute the extensions of abstract argumentation frameworks with collective attacks in Section 4.1 and claim augmented abstract argumentation frameworks in Section 4.2. Furthermore, for some encodings for claim augmented abstract argumentation frameworks, we will provide more than one encoding, utilizing different approaches.

- The conducted experiments and the gathered data will be presented in Chapter 5. Furthermore, in Section 5.1, we will outline how and which test case were selected for our experiments and present our algorithms used to modify the selected instances to obtain (well-formed) claim augmented abstract argumentation framework

instances. The experiments in Section 5.2 will be dedicated to non well-formed claim augmented abstract argumentation frameworks, while the Sections 5.3 and 5.4 will present our findings for well-formed claim augmented abstract argumentation frameworks. Moreover, in Section 5.5, we will present our results for the conversion from well-formed claim augmented abstract argumentation frameworks to abstract argumentation frameworks with collective attacks.

- Finally, in Chapter 6, we will summarize our results and conclude this thesis.

1.4 Published results and systems

The complexity results and encodings for abstract argumentation frameworks with collective attacks have already been published [DGW18] with slightly different notation and have been presented at the SAFA 2018[1] workshop. Furthermore, the encodings have been included in the ASPARTIX system and are available at `https://www.dbai.tuwien.ac.at/research/argumentation/aspartix/setaf.html` and `https://www.dbai.tuwien.ac.at/research/argumentation/aspartix/caf.html`.

[1] https://safa2018.argumentationcompetition.org

CHAPTER 2

Formal Background

Within this chapter, we will introduce the basic concepts and formalisms used within the subsequent chapters. More specifically, in Section 2.1, we will give the basic notations of complexity theory and introduce some of the required concepts and complexity classes. Section 2.2 will illustrate the syntax and semantics of answer-set programs as well as the syntactic extensions and employed programming techniques. Finally, in Section 2.3, we will introduce the basic definitions of argumentation frameworks, starting with Dung's abstract argumentation frameworks, followed by two generalizations, namely abstract argumentation frameworks with collective attacks and claim augmented abstract argumentation frameworks.

2.1 Complexity theory

In this section, we will introduce the basic notation required for the complexity results presented in subsequent parts of this thesis. One of the most central concepts of complexity theory will be a computational model commonly referred to as *Turing machine*, proposed by Alan Turing in 1937 [Tur37]. We will omit a formal definition of the Turing machine and refer to [AB09] for a more in-depth discussion of the concepts introduced in this section. Basically, a Turing Machine consists of a read/write head and an infinitely long tape divided into cells, each containing either a symbol or being blank. Thus, a program for such a Turing machine modifies the content of those cells and moves the read/write head based on the read input and the current internal state of the machine.

The complexity classes we will introduce in this section are classes of decision problems, i.e. problems that can be answered either by "yes" or "no". An algorithm for such a decision problem thus categorizes instances of that problem into "yes"-instances, possessing some set of properties and "no"-instances, lacking those properties.

Typically, one is most interested in the time- and space complexity of an algorithm. To this end, the big-$\mathcal{O}$ notation is used to categorize the asymptotic, worst case complexity of an algorithm such that for a function f, every instance of the considered problem can be answered in space- or time $\mathcal{O}(f(n))$, where n denotes the size of the instance. Formally, $f(n) = \mathcal{O}(g(n))$ if there are positive integers c and n_0, such that for all $n \geq n_o$, $f(n) \leq c \cdot g(n)$. Using this notation, one can define the three complexity classes that will be most relevant in the scope of thesis, namely P, NP, and Σ_2^P.

The complexity class P consists of all decision problems $\mathcal{P}$ such that there is a deterministic Turing machine TM that, for all instances I of $\mathcal{P}$, gives the correct result and runs in time $\mathcal{O}(|I|^k)$ for some constant k, where $|I|$ is the size of I. Furthermore, the class NP is defined analogously for non-deterministic Turing machines.

Before introducing the last of the three before mentioned classes, we introduce two more concepts, namely *oracle augmented Turing machines* and *reductions*. An oracle augmented Turing machine has an additional oracle band and additional transition operations that allow for constant-time testing of membership in the language of some decision problem. Once again, we refer to [AB09] for a more detailed definition. Furthermore, we will focus on polynomial-time many-one reductions, sometimes referred to as "Karp reductions". A decision problem P is polynomial-time many-one reducible, usually denoted by $P \leq_p Q$, if there is a polynomial time computable function f such that x is a "yes"-instance of P if and only if $f(x)$ is a "yes"-instance of Q. For the rest of this thesis, we will refer to "polynomial-time many-one reduction" simply as reduction.

Furthermore, we will call a decision problem P to be *hard* for a complexity class C if for any problem Q in C it holds that $Q \leq_p P$ and we will call P *complete* for C, denoted by C-c, if P is in C and P is C-hard. Furthermore, to show that a new problem is C-hard, due to transitivity of reductions, it is sufficient to reduce a problem known to be C-hard to the new problem.

Using this definitions, we can introduce the polynomial hierarchy, which consists of three families of complexity classes Δ_ℓ^P, Σ_ℓ^P and Π_ℓ^P, where $0 \leq \ell$, defined as follows:

- $\mathsf{P} = \Sigma_0^\mathsf{P} = \Pi_0^\mathsf{P} = \Delta_0^\mathsf{P}$,
- $\Sigma_\ell^\mathsf{P} = \mathsf{NP}^\Sigma_{\ell-1}$,
- $\Pi_\ell^\mathsf{P} = \mathsf{co}\Sigma_\ell^\mathsf{P}$ and
- $\Delta_\ell^\mathsf{P} = \mathsf{P}^{\Sigma_{\ell-1}^\mathsf{P}}$

for $\ell \geq 1$, where the notation C^D refers to the set of decision problems solvable by a Turing machine of class C, augmented with an oracle for some D-complete problem. This concept refers to the idea of *relative computability*. The relation between these classes is illustrated in Figure 2.1.

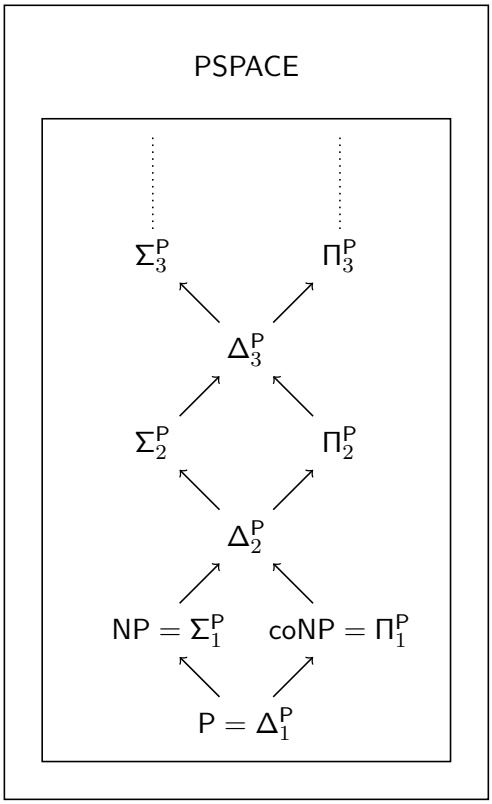

Figure 2.1: The polynomial hierarchy

2.2 Answer-set programming

Answer-set programming (ASP) [BET11] is a declarative problem-solving approach based on the *stable model semantics* proposed by Gelfond and Lifschitz in 1988 [GL88]. The term ”answer-set programming” was coined by Lifschitz in 1999 [Lif99] and proposed by others at around the same time [MT99]. Due to its relation to the logic programming paradigm, answer-set programming makes use of concepts of formal logic, such as *non-monotonic reasoning*, and focuses on describing a problem and its solution instead of the flow control, as in imperative programming languages.

Despite its similarities to *Boolean satisfiability (SAT)*, some answer-set solvers, such as ASSAT[1], even make use of SAT solvers to solve answer-set programs, answer-set programming offers some features that distinguish itself from SAT. Most notably, these include the use of *grounding* to allow the use of variables, as well as model minimality due to foundedness check, the negation-as-failure operator and the use of transitive closure. Moreover, answer-set programming has been designed to be ”purely” declarative, in the sense that the order of rules within a program and the order of sub-goals within a rule

[1] http://assat.cs.ust.hk/

do not matter. Moreover, termination of an answer-set program is less of a key-point as it is in other logic programming formalisms such as *prolog*, enabling the user to focus on specifying the problem and its solutions without the need to know the precise inner workings of the employed solver.

Due to its expressiveness and the existence of efficient solvers such as *DLV*[2] and *clasp*[3], answer-set programming has experienced increasing popularity as problem-solving formalism. Therefore, answer-set programming has been employed for various use-cases, including planning [TB01], diagnosis [BG03], argumentation [EGW08], the semantic web [Pol05] and natural language processing [EGL16] and was used as part of the decision support system for the space shuttle [NBG+01]. Moreover, competitions are held regularly to evaluate the performance of competing answer-set solvers [GMR19].

2.2.1 Syntax

In order to define the syntax of answer-set programs, we assume some *first-order vocabulary* $\Phi = (\mathcal{C}, \mathcal{V}, \mathcal{F}, \mathcal{P})$ with non-empty finite sets $\mathcal{C}$ of *constant symbols*, $\mathcal{V}$ of *variables*, $\mathcal{F}$ *function symbols* and $\mathcal{P}$ *predicate symbols*. Function symbols as well as predicate symbols have an associated arity $n \geq 0$. By convention, we will denote constant symbols by integers or identifiers starting with lowercase letters, while variables will start with uppercase letters.

A term is either a variable, constant symbol or inductively built from terms using function symbols. An *atom* is an expression of the form $p(t_1, \ldots, t_n)$ where

- $p \in \mathcal{P}$ with arity n and
- $t_1, \ldots, t_n$ are terms.

An atom or a term is referred to as *ground* if it contains no variables. Furthermore, an answer-set program Π_Φ, with regard to some vocabulary Φ, is a set of *rules* of the form

$$r = a_1 \mid \ldots \mid a_n \text{ :- } b_1, \ldots b_l, \text{ not } b_{l+1}, \ldots, \text{ not } b_m.$$

with $\{a_1, \ldots, a_n, b_1, \ldots, b_m\}$ literals from Φ. The expression "not" is called the negation-as-failure operator and refers to default negation. Moreover, the set $\{a_1, \ldots, a_n\}$ is referred to as the *head* $H(r)$ of a rule r, while $\{b_1, \ldots, b_l\}$ is called the *positive body* $B^+(r)$ and $\{b_{l+1}, \ldots, b_m\}$ is called the *negative body* $B^-(r)$, with the union $B(r) = B^+(r) \cup B^-(r)$ of both constituting the *body* of the rule r. Finally, a rule is called a *fact*, if $m = 0$.

[2] http://www.dlvsystem.com/
[3] https://potassco.org/

2.2.2 Semantic

In order to define the semantics of an answer-set program Π_Φ, we first have to define some auxiliary concepts. The *Herbrand universe* $HU(\Pi_\Phi)$ of a program Π_Φ is the set of all ground terms of Π_Φ constructible using the function and constant symbols from $\mathcal{F}$ in Φ. Furthermore, the *Herbrand base* $HB(\Pi_\Phi)$ of a program Π_Φ is the set of all ground atoms constructible using the predicate symbols from $\mathcal{P}$ in Φ and terms from $HU(\Pi_\Phi)$.

A first order interpretation $I_\Phi = (D, \cdot')$, with regard to some first-order vocabulary Φ, is a pair where

- D is a non-empty domain and
- $\cdot'$ is an interpretation function assigning
  - some $c' \in D$ to each constant symbol $c \in \mathcal{C}$ of Φ,
  - some $X' \in D$ to each free variable X and
  - some subset $p' \subseteq D^n$ for each n-ary predicate symbol $p \in \mathcal{P}$ in Φ.

If not stated otherwise, we will implicitly use a *Herbrand interpretation* of a program Π_Φ where $D = HU(\Pi_\Phi)$ and $t' = t$ for each term $t \in HU(\Phi_\Psi)$. Moreover, an interpretation $I_\Phi = (D, \cdot')$ satisfies

- an n-ary ground atom $p(t_1, \ldots, t_n)$, denoted by $I_\Phi \models p(t_1, \ldots, t_n)$, if $(t_1, \ldots, t_n) \in p'$,
- an n-ary ground default negated atom not $p(t_1, \ldots, t_n)$, denoted by $I_\Phi \models$ not $p(t_1, \ldots, t_n)$, if $(t_1, \ldots, t_n) \notin p'$,
- a ground rule r, denoted by $I_\Phi \models r$, if
  - $I_\Phi \models a$ for some $a \in H(r)$ or
  - $I_\Phi \not\models a$ for some $a \in B^+(r)$ or
  - $I_\Phi \models a$ for some $a \in B^-(r)$.
- a program Π_Φ, denoted by $I_\Phi \models \Pi_\Phi$, if $I_\Phi \models r$ for all rules $r \in \Pi_\Phi$. Such an interpretation is called a *model* of Π_Φ.

The central concept for the semantics of answer-set programs is the *Gelfond-Lifschitz (GL-) reduct*. The GL-reduct ${\Pi_\Phi}^{I_\Phi}$ of a ground program Π_Φ, with regard to an interpretation I_Φ, is the program

$$\{H(r) \text{ :- } B^+(r). \mid r \in \Pi_\Phi \text{ such that } I_\Phi \not\models a \text{ for all literals } a \in B^-(r)\}$$

To deal with possibly non-ground programs, the concept of *grounding* is used to define which interpretations of such a programs are considered to be answer-sets.. Thus, by $ground_{HU(\Pi_\Phi)}(x)$, we denote the set of all possible instances of x, where all variables are instantiated with all constants from $HU(\Pi_\Phi)$ for each atom, rule or program x. Using this notation, we define an interpretation I_Φ of a program Π_Φ to be an *answer-set* of Π_Φ if and only if it is a subset-minimal model of $ground_{HU(\Pi_\Phi)}(\Pi_\Phi)^{I_\Phi}$.

Moreover, in order to guarantee some bounds of the solving in practice, we will introduce the concept of *rule safety*. A rule r is called *safe*, if all of the variables that occur within the rule also occur in the positive body $B^+(r)$ of the rule. Furthermore, an answer-set program is called safe, if all its rules are safe. A safe program only has finitely many models that are all finite. Thus, program safety guarantees termination for model computation. For the rest of this thesis, we will only consider safe programs.

Furthermore, rules with an empty head are referred to as *constraints*. Such a rule will eliminate all answer-sets candidates that satisfy the body of the rule. A constraint rule

$$:\text{-}\ B.$$

is the short-hand notation for a rule

$$p :\text{-}\ B,\ \text{not}\ p.$$

where B is the body of the rule without p and p does not occur elsewhere in the program.

2.2.3 Syntactic extensions and programming techniques

The "basic" syntax and semantics of answer-set programs defined in the previous subsections cover all constructs required for the answer-set encodings provided within this thesis, with three exceptions that we will define here. Modern answer-set solvers and their respective grounders support various extensions of the basic syntax defined so far, most of them focusing on easing the use of the solver without increasing the expressiveness, while some do [EIST05]. We will employ four such features, namely *anonymous variables*, *suppression of irrelevant atoms*, *model projection* and *conditional literals*.

Anonymous variables can be used to increase readability, if a variable within a rule is just used as a placeholder without actually requiring its value. Thus, a rule

$$p(A) :\text{-}\ q(A, B).$$

is equivalent to a rule

$$p(A) :\text{-}\ q(A, \_).$$

Suppression of irrelevant atoms can be used to output only the interesting atoms, suppressing atoms that belong to the input or that are only required to compute those atoms that we are interested in. The syntax we will use is the one the solver "clasp" uses, thus we will add rules of the form

$$\#\text{show } p/n.$$

to state that the atoms over the n-ary predicate p should occur within the output. If one such rule is given, all atoms that do not match such are rule will be suppressed. However, if there are multiple answer-sets that only differ in atoms that are suppressed, the output will still contain sets of literals for each such answer-set, which will then be identical. To prevent this, we can use the model projection feature to ensure that we will receive unique sets of literals as output. This feature can be used to project the answer-set to sets of literals that are unique over predicates stated in the show-directives. Model projection can also be taken into account within the solving algorithm, instead of ensuring the uniqueness as a post-processing step after solving, which can be more efficient [GKS09]. Using both features, it is possible to guarantee unique sets of literals over the predicate p as output.

Furthermore, conditional literals can be used as the last literal in the body of a rule or within the head of a rule. A rule with a conditional literal in the body has the form

$$H \text{ :- } B, p : q_1, \ldots, q_n.$$

where H is the head of the rule, B is the body of the rule without the conditional literal and $p : q_1, \ldots, q_n$ being the conditional literal with $\{p, q_1, \ldots, q_n\}$ being literals. For a given interpretation I, such a rule can be rewritten into a rule

$$H \text{ :- } B, p_1, \ldots, p_k.$$

where all p_l are instantiation of p where all variables in $p_l, q_1, \ldots, q_n$ are replaced by ground terms such that $I \models q_1, \ldots, q_n$.

Conditional body literals can be very useful for expressing a conjunction over arbitrarily many ground atoms in the body of a rule, especially if the number of such literals is not known at the time of modeling. In contrast to these, conditional head literals express a disjunction within the head of a rule. A rule with a conditional literal in the head has the form:

$$H \mid p : q_1, \ldots, q_n \text{ :- } B.$$

where H is the head of the rule without the conditional literal, B is the body of the rule and $p : q_1, \ldots, q_n$ being the conditional literal with $\{p, q_1, \ldots, q_n\}$ being literals. For a given interpretation I, such a rule can be rewritten into a rule

$$H \mid p_1 \mid \ldots \mid p_n \text{ :- } B.$$

where all p_l are instantiations of p where all variables in $p_l, q_1, \ldots, q_n$ are replaced by ground terms such that $I \models q_1, \ldots, q_n$. Moreover, conditional head literals can be used in conjunction with *guards*, to infer more than one such literal. The rules we will use will be of the form

$$\{p : q\} > 0 \text{ :- } B.$$

stating that an arbitrary, non-zero number of such instantiation of p might be inferred.

Besides of those features, there are two common modeling techniques that we will use and thus want to introduce, starting with the *guess-and-check* paradigm [Lif02, calling it generate/define/test]. The idea is to split the program into two parts: one that spans the entire search space and one that eliminates all illegal solutions. Often times, an additional part is added to hold the facts that specify a specific instance. Consider the following example encoding for the vertex coloring problem:

$$\Pi_{input} = \begin{cases} vertex(a).\ vertex(b).\ vertex(c). \\ edge(a,b).\ edge(b,c).\ edge(c,a). \\ color(green).\ color(red).\ color(blue). \end{cases}$$
$$\Pi_{guess} = \{coloring(V,C) : color(C) \text{ :- } vertex(V).$$
$$\Pi_{check} = \{\text{ :- } coloring(V1,C), coloring(V2,C), edge(V1,V2).$$

In this example, Π_{input} holds the input facts, while Π_{guess} spans the search space using a conditional head literal and Π_{check} eliminates all answer-set candidates in which two adjacent vertices share some color. Thus, the program $\Pi = \Pi_{input} \cup \Pi_{guess} \cup \Pi_{check}$ in conjunction with the rule $\#showcoloring/2.$ will yield the expected output:

$$AS(\Pi) = \begin{cases} \{\text{coloring(a,blue)},\ \text{coloring(b,red)},\ \text{coloring(c,green)}\} \\ \{\text{coloring(a,blue)},\ \text{coloring(b,green)},\ \text{coloring(c,red)}\} \\ \{\text{coloring(a,red)},\ \text{coloring(b,blue)},\ \text{coloring(c,green)}\} \\ \{\text{coloring(a,red)},\ \text{coloring(b,green)},\ \text{coloring(c,blue)}\} \\ \{\text{coloring(a,green)},\ \text{coloring(b,red)},\ \text{coloring(c,blue)}\} \\ \{\text{coloring(a,green)},\ \text{coloring(b,blue)},\ \text{coloring(c,red)}\} \end{cases}$$

where $AS(\Pi)$ denotes the set of all answer-sets of the program Π.

Finally, we will refer to second important programming paradigm as *saturation technique* [EG95]. It will be used to ensure that guesses meet some maximality condition. The

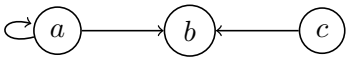

Figure 2.2: A Dung abstract argumentation framework

paradigm will be explained in subsequent chapters of the thesis, when it first becomes relevant, having a proper example to illustrate the idea and propose in Subsection 4.1.2.

2.2.4 Complexity

The computational complexity of answer-set programming is well understood. Deciding whether a normal program, i.e. a program without disjunction, has at least one answer set is NP-complete in data size [DEGV97]. Moreover, for answer-set programs with disjunction in general, this rises up to Σ_2^P-complete [EG93], one level above the complexity for normal answer-set programs, in the polynomial hierarchy. Furthermore, diverse extensions of answer-set programs exist, lifting the complexity to various higher levels. HEX-programs [EKR+17], for example, can make use of *external predicates* querying various data sources, such as imperative programs or web services, causing the query answering to become potentially undecidable.

2.3 Argumentation frameworks

One approach of modeling an argumentation process has been proposed by Dung in 1995 [Dun95] and has since been refined by several authors. In the following subsections, we will introduce and define Dung abstract argumentation frameworks, as well as two generalizations thereof.

2.3.1 Dung abstract argumentation frameworks

An *abstract argumentation framework (AF)* is a pair (A, R), where

- A is a finite set of *arguments* and
- $R \subseteq A \times A$ is the attack relation representing conflicts between arguments.

Dung abstract argumentation frameworks can be represented by a directed graph $G = (V, E)$, where V is the set of arguments and the edges E correspond to the attacks. Thus, an $AF = (\{a, b, c\}, \{(a, a), (a, b), (c, b)\})$ can be represented by the graph in Figure 2.2.

Multiple generalizations of Dung argumentation frameworks have been proposed, two of which we will introduce the next two subsections.

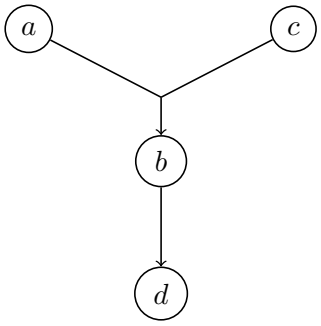

Figure 2.3: An abstract argumentation framework with collective attacks

2.3.2 Abstract argumentation frameworks with collective attacks

The first generalization we will introduce are *abstract argumentation frameworks with collective attacks (SETAF)* [NP06b]. An abstract argumentation framework with collective attacks (SETAF) is a pair (A, R), where

- A is a finite set of arguments and
- $R \subseteq (2^A \setminus \emptyset) \times A$ is the attack relation representing conflicts between non-empty sets of arguments and single arguments.

Analogously to Dung AFs, also abstract argumentation frameworks with collective attacks can be represented using directed hypergraphs. A $SETAF = (\{a, b, c, d\}, \{(\{a, c\}, b), (\{b\}, d)\})$ can be represented by the graph in Figure 2.3.

2.3.3 Claim augmented abstract argumentation frameworks

The second generalization we will consider are *claim augmented abstract argumentation frameworks (CAF)* [DW19]. A claim augmented abstract argumentation framework (CAF) is a triple (A, R, γ), where

- (A, R) is an abstract argumentation framework and
- $\gamma : A \mapsto C$ is a mapping from arguments to some set of claims C.

Such a claim augmented abstract argumentation framework is considered to be *well-formed* if, for any two arguments $\{a, b\} \subseteq A$ such that $\gamma(a) = \gamma(b)$, $\{x \mid (a, x) \in R\} = \{x \mid (b, x) \in R\}$, i.e. arguments with the same claim attack the same arguments. Moreover, claim augmented abstract argumentation frameworks can also by represented using directed graphs. A $CAF = (\{a, b, c\}, \{(a, b), (b, c)\}, (a \mapsto \alpha, b \mapsto \beta, c \mapsto \alpha))$ can be represented by the graph in Figure 2.4.

Note that this claim augmented abstract argumentation framework is not well-formed, as $\gamma(a) = \gamma(c)$, but $\{x \mid (a, x) \in R\} = \{b\} \neq \emptyset = \{x \mid (c, x) \in R\}$.

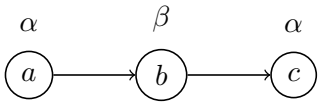

Figure 2.4: A non well-formed claim augmented abstract argumentation framework

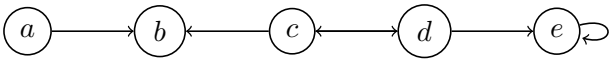

Figure 2.5: A Dung abstract argumentation framework

2.3.4 Semantics

Various semantics have been proposed in the literature [Dun95, BCG11, BDG11, Cam07, DDW13], defining which sets of arguments $E \subseteq A$ for a given argumentation framework, satisfying certain properties, are considered to be the coherent sets of arguments that are jointly acceptable under this semantics. Such sets of arguments are often referred to as extensions. We will consider the following seven semantics for a Dung abstract argumentation framework $AF = (A, R)$. A set $S \subseteq A$ is called

- *conflict-free* in AF, if there are no two arguments $\{a, b\} \subseteq S$ such that $(a, b) \in R$.
- *admissible* in AF, if S in conflict-free in AF and each $a \in S$ is defended by S in AF, i.e. for each argument $b \in A$ such that $(b, a) \in R$ there is some argument $c \in S$ such that $(c, b) \in R$.
- *complete* in AF, if S is admissible in AF and it holds that for each argument $a \in A$ defended by S in AF, $a \in S$.
- *preferred* in AF, if S is admissible in AF and for each admissible extension $T \subseteq A$ of AF, it holds that $S \not\subset T$.
- *stable* in AF, if S is conflict-free in AF and for each argument $a \in A \setminus S$ there is some argument $b \in S$ such that $(b, a) \in R$.
- *semi-stable* in AF, if S is admissible in AF and for each admissible extension $T \subseteq A$ of AF, $S_R^+ \not\subset T_R^+$, where $E_R^+ = E \cup \{a \mid \exists b \in E \text{ such that } (b, a) \in R\}$ is called the *range* of E for some set of arguments $E \subseteq A$.
- *stage* in AF, if S is conflict-free in AF and for each conflict-free extension $T \subseteq A$ of AF, $S_R^+ \not\subset T_R^+$.

To illustrate these semantics, we will give the extensions for the Dung abstract argumentation framework depicted in Figure 2.5. For this framework, the

- conflict-free extensions are: $\{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}, \{b, d\}\}$

- admissible extensions are: $\{\emptyset, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}\}$
- complete extensions are: $\{\{a\}, \{a, c\}, \{a, d\}\}$
- preferred extensions are: $\{\{a, c\}\, \{a, d\}\}$
- stable extensions are: $\{\{a, d\}\}$
- semi-stable extensions are: $\{\{a, d\}\}$
- stage extensions are: $\{\{a, d\}\}$

For an abstract argumentation framework with collective attacks (SETAF) $AF = (A, R)$, a set $S \subseteq A$ is called

- *conflict-free* in AF, if there is no argument $a \in S$ such that there is a set $S' \subseteq S$ with $(S', a) \in R$.
- *admissible* in AF, if S is conflict-free in AF and each $a \in S$ is defended by S in AF, i.e. for each set of argument $S' \subseteq A$ such that $(S', a) \in R$ there is some set of argument $S'' \subseteq S$ such that $(S'', b) \in R$ for some argument $b \in S'$.
- *stable* in AF, if S is conflict-free in AF and for each argument $a \in A \setminus S$ there is some set of arguments $S' \subseteq S$ such that $(S', a) \in R$.

The remaining semantics are defined analogously as for Dung abstract argumentation frameworks, using a slightly adapted definition of the range: $E_R^+ = E \cup \{a \mid \exists S \subseteq E \text{ such that } (S, a) \in R\}$.

Furthermore, the extensions of claim augmented abstract argumentation frameworks are defined in terms of claims instead of arguments: for a claim augmented abstract argumentation framework (A, R, γ) and a semantics σ for Dung abstract argumentation framework, the set of extensions under σ is defined as $\{claim(E) \mid E \in \sigma((A, R))\}$ where $claim(E) = \{\gamma(a) \mid a \in E\}$. This definition of the semi-stable and stage semantics for claim augmented abstract argumentation frameworks requires maximality of their range, which is defined in terms of arguments, while the extension itself is expressed in terms of claims. Therefore, we will consider two additional variants for the semi-stable and stage semantics for claim augmented abstract argumentation frameworks, focusing on maximizing claims instead of arguments, related to a similar concept for logic programs introduced in [CSAD15]. Thus, for a claim augmented abstract argumentation framework $CAF = (AF, \gamma)$, where $AF = (A, R)$ is a Dung abstract argumentation framework, $claim(S)$ of a set $S \subseteq A$ is called

- semi-stable$_c$ in CAF, if S is admissible in AF and for each admissible extension $T \subseteq A$ of AF, $S_R^{+_{claim}} \not\subset T_R^{+_{claim}}$.

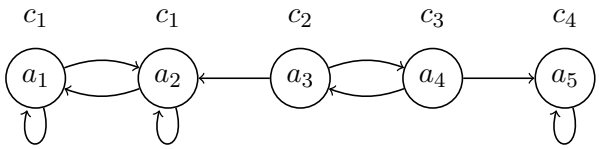

Figure 2.6: A well-formed claim augmented abstract argumentation framework

- stage$_c$ in CAF, if S is conflict-free in AF and for each conflict-free extension $T \subseteq A$ of AF, $S_R^{+_{claim}} \not\subset T_R^{+_{claim}}$.

where $E_R^{+_{claim}} = claim(E) \cup \{c \in claim(A) \mid \forall a \in A(\gamma(a) = c \rightarrow \exists b \in E \text{ such that } (b, a) \in R)\}$ for some set of arguments $E \subseteq A$.

To illustrate that these semantics in fact differ from the traditional semantics, consider the well-formed claim augmented abstract argumentation framework depicted in Figure 2.6, for which the

- the traditional semi-stable and stage extensions are $\{\{c_2\}, \{c_3\}\}$ and
- the semi-stable and stage extensions with claim-centric range are $\{\{c_3\}\}$.

2.3.5 Complexity

For a given semantics σ, we will consider five decision problems:

- *Credulous acceptance*, $Cred_\sigma(AF, a)$: Is there some extension S of AF under σ such that $a \in S$?
- *Skeptical acceptance*, $Skept_\sigma(AF, a)$: Does $a \in S$ hold for all extension S of AF under σ?
- *Verification*, $Ver_\sigma(AF, S)$: Is S an extension of AF under σ?
- *Existence*, $Exists_\sigma(AF)$: Does AF have some extension under σ?
- *Non-empty existence*, $Exists_\sigma^{\neg\emptyset}(AF)$: Does AF have some non-empty extension under σ?

For these decision problems and the seven considered semantics, the complexity results [DT96, DB02, DW09, DW10, Dvo12, DD18] for Dung abstract argumentation frameworks as well as abstract argumentation framework with collective attacks, where $\mathcal{C}$-c denoted completeness for a complexity class $\mathcal{C}$, are depicted in Table 2.1.

Furthermore, the complexity results [DW19] for claim augmented abstract argumentation frameworks, are depicted in Table 2.2. Moreover, tighter upper bounds can be given

σ	$Cred_\sigma$	$Skept_\sigma$	Ver_σ	$Exists_\sigma$	$Exists_\sigma^{\neg\emptyset}$
conflict-free	in P	trivial	in P	trivial	in P
admissible	NP-c	trivial	in P	trivial	NP-c
complete	NP-c	P-c	in P	trivial	NP-c
preferred	NP-c	Π_2^P-c	coNP-c	trivial	NP-c
stable	NP-c	coNP-c	in P	NP-c	NP-c
semi-stable	Σ_2^P-c	Π_2^P-c	coNP-c	trivial	NP-c
stage	Σ_2^P-c	Π_2^P-c	coNP-c	trivial	in P

Table 2.1: Complexity landscape for Dung abstract argumentation frameworks

σ	$Cred_\sigma$	$Skept_\sigma$	Ver_σ	$Exists_\sigma$	$Exists_\sigma^{\neg\emptyset}$
conflict-free	in P	trivial	NP-c	trivial	in P
admissible	NP-c	trivial	NP-c	trivial	NP-c
complete	NP-c	P-c	NP-c	trivial	NP-c
preferred	NP-c	Π_2^P-c	Σ_2^P-c	trivial	NP-c
stable	NP-c	coNP-c	NP-c	NP-c	NP-c

Table 2.2: Complexity landscape for non well-formed claim augmented abstract argumentation frameworks

σ	$Cred_\sigma$	$Skept_\sigma$	Ver_σ	$Exists_\sigma$	$Exists_\sigma^{\neg\emptyset}$
conflict-free	in P	trivial	in P	trivial	in P
admissible	NP-c	trivial	in P	trivial	NP-c
complete	NP-c	P-c	in P	trivial	NP-c
preferred	NP-c	Π_2^P-c	coNP-c	trivial	NP-c
stable	NP-c	coNP-c	in P	NP-c	NP-c

Table 2.3: Complexity landscape for well-formed claim augmented abstract argumentation frameworks

for the verification of well-formed claim augmented abstract argumentation frameworks, depicted in Table 2.3. The complexity results for the existence problem follow from the results for Dung abstract argumentation frameworks.

In this work, we will complement these results by giving complexity results for abstract argumentation frameworks with collective attacks as well as for the semi-stable and stage semantics and their variants with claim-centric ranges in Chapter 3.

2.3.6 Conversion of well-formed instances of claim augmented abstract argumentation frameworks into instances of abstract argumentation frameworks with collective attacks

As mentioned in the introduction, there is a relation between well-formed claim augmented abstract argumentation frameworks and abstract argumentation frameworks with collective attacks. Therefore, it is possible to convert instances of well-formed claim augmented abstract argumentation frameworks to instances of abstract argumentation frameworks with collective attacks such that the extensions are identical for the conflict-free, admissible, complete, preferred and stable semantics [DRW19]. With such a conversion at hand, one can compute the extensions of claim augmented abstract argumentation frameworks using the encodings for abstract argumentation frameworks with collective attacks. Thus, for the experiments in Subsection 5.5, the Algorithm 2.1 will be used to convert instance of well-formed claim augmented abstract argumentation frameworks to instances of abstract argumentation frameworks with collective attacks.

Algorithm 2.1: CAFtoSETAFconversion

Input: A: set of arguments, R: set of attacks between claims and arguments, γ: function mapping arguments to claims

Output: A': new set of arguments, R' new set of attack between sets of arguments and arguments

```
1  claims ← {γ(a) | a ∈ A} ;
2  A' ← claims ;
3  foreach claim c in claims do
4  |  argumentsWithClaim ← {a | γ(a) = c};
5  |  attackers ← ordered set {{γ(b) | (b, a) ∈ R} | a ∈ argumentsWithClaim} ;
6  |  if ∅ ∈ attackers then
   |  |  /* There is an argument of the current claim that is
   |  |     not attacked, thus the claim will not be attacked
   |  |     in the abstract argumentation framework with
   |  |     collective attacks encoding                        */
7  |  |  continue ;
8  |  end
9  |  foreach pick one claim from each set in attackers and collect into attack do
10 |  |  add (attack, c) to R' ;
11 |  end
12 end
```

CHAPTER 3

Complexity Results

In this chapter, we will give the complexity results for the decision problems introduced in Subsection 2.3.5. We will consider the semantics listed in Subsection 2.3.4 for abstract argumentation frameworks with collective attacks and will give the results for the stage and semi-stable semantics for non well-formed and well-formed claim augmented abstract argumentation frameworks for the traditional and the claim-centric versions.

3.1 Abstract argumentation framework with collective attacks

The complexity results for abstract argumentation frameworks with collective attacks are depicted in Table 3.1. We will give proofs of these results in the following subsections. Furthermore, we get hardness from the complexity results for Dung abstract argumentation frameworks, see Table 2.1, as each Dung abstract argumentation framework can be interpreted as an abstract argumentation framework with collective attacks. Thus, we will only give results for the respective upper bounds.

Definition 1. The characteristic function of an abstract argumentation framework with collective attacks $SETAF = (A, R)$ is the function $\mathcal{F} : 2^A \to 2^A$ with $\mathcal{F}(S) = \{a \in A : a \text{ is defended by } S\}$.

Furthermore, as for Dung abstract argumentation frameworks, deciding whether an argument is defended by a set of arguments is still in P.

Proposition 1. *Given an abstract argumentation framework with collective attacks $SETAF$, a set of arguments S and an argument a. Deciding whether $a \in \mathcal{F}(S)$ is in* P.

Proof. We simple iterate over all attacks $(B, x) \in R$ and whenever $x = a$, we again iterate over all attacks $(B', x') \in R$ and test whether $x' \in B$ and $B' \subseteq S$. If there is an

σ	$Cred_\sigma$	$Skept_\sigma$	Ver_σ	$Exists_\sigma$	$Exists_\sigma^{\neg\emptyset}$
conflict-free	in P	trivial	in P	trivial	in P
admissible	NP-c	trivial	in P	trivial	NP-c
complete	NP-c	P-c	in P	trivial	NP-c
preferred	NP-c	Π_2^P-c	coNP-c	trivial	NP-c
stable	NP-c	coNP-c	in P	NP-c	NP-c
semi-stable	Σ_2^P-c	Π_2^P-c	coNP-c	trivial	NP-c
stage	Σ_2^P-c	Π_2^P-c	coNP-c	trivial	in P

Table 3.1: Complexity results for abstract argumentation frameworks with collective attacks

attack (B, a) such that there is no defending attack (B', x'), we have $a \notin \mathcal{F}(S)$ otherwise $a \in \mathcal{F}(S)$. Clearly, the above can be implemented in polynomial time and thus deciding whether $a \in \mathcal{F}(S)$ is in P. □

3.1.1 Conflict-free semantics

In this subsection, we will show that the results for the conflict-free semantics depicted in Table 3.1 hold. We first observe, that the empty set is always a conflict-free extension. Thus, $Skept_{\text{conflict-free}}(SETAF, a)$ is trivially false and $Exists_{\text{conflict-free}}(SETAF)$ is trivially true for any abstract argumentation framework with collective attacks $SETAF = (A, R)$ and argument $a \in A$.

For the non-trivial results, we will start with the credulous acceptance problem.

Lemma 1. *Deciding $Cred_{\text{conflict-free}}(SETAF, a)$ for an arbitrary abstract argumentation framework with collective attacks $SETAF = (A, R)$ and an argument $a \in A$ is in P.*

Proof. In order to decide credulous acceptance for an argument a, we simply check whether $(\{a\}, a) \notin R$. If this is the case, then $\{a\} \in$ conflict-free(F) and thus a is credulously accepted, otherwise a cannot be in any conflict-free set and thus is not credulously accepted. Therefore, $Cred_{\text{conflict-free}}(SETAF, a)$ is in P. □

Next, we will prove the result for the verification problem.

Lemma 2. *Deciding $Ver_{\text{conflict-free}}(SETAF, S)$ for an arbitrary abstract argumentation framework with collective attacks $SETAF = (A, R)$ and a set of arguments $S \subseteq A$ is in P.*

Proof. To verify that a set S is a conflict-free extension, we iterate over all $(B, x) \in R$ and test whether $B \cup \{x\} \subseteq S$. If this is the case, then there is a conflict in S and we

terminate. Otherwise, if none of the attacks is contained in S, the set is conflict-free. This, is clearly in P. □

Finally, we will present a proof for the non-empty existence problem.

Lemma 3. *Deciding* $Exists^{\neg\emptyset}_{conflict\text{-}free}(SETAF)$ *for an arbitrary abstract argumentation framework with collective attacks* $SETAF = (A, R)$ *is in* P.

Proof. To decide whether there is a non-empty conflict-free set, we test each argument for being credulously accepted. If one of them is, then there is some non-empty extension and vice versa. Again, this is clearly in P. □

3.1.2 Stable semantics

Analogously to the previous subsection, we will show in this subsection that the results for the stable semantics, depicted in Table 3.1, hold.

From the remaining non-trivial results, we will first consider the verification problem.

Lemma 4. *Deciding* $Ver_{stable}(SETAF, S)$ *for an arbitrary abstract argumentation framework with collective attacks* $SETAF = (A, R)$ *and a set of arguments* $S \subseteq A$ *is in* P.

Proof. One can verify that a given set S is a stable extension of $SETAF$, by first checking in P, cf. Lemma 2, that it is conflict-free and then that for each $a \in A \setminus S$ there is an attack $(B, a) \in R$ with $B \subseteq S$. As both can be done in P, we obtain P membership of $Ver_{\text{stable}}(SETAF, S)$. □

Secondly, we will prove our results for the credulous acceptance problem.

Lemma 5. *Deciding* $Cred_{stable}(SETAF, a)$ *for an arbitrary abstract argumentation framework with collective attacks* $SETAF = (A, R)$ *and an argument* $a \in A$ *is* NP*-c.*

Proof. Recall that we get hardness from the complexity results for Dung abstract argumentation frameworks, see Table 2.1, as each Dung abstract argumentation framework can be interpreted as an abstract argumentation framework with collective attacks. This also applies for the remaining proofs for abstract argumentation frameworks with collective attacks.

We first use the non-determinism to guess a set S and then check whether the argument a under question is in the set. Finally, we check that S is stable which can be done in P, cf. Lemma 4, resulting in a NP procedure to decide $Cred_{\text{stable}}(SETAF, a)$. □

Next, we will present our proof for the skeptical acceptance problem.

Lemma 6. *Deciding $Skept_{stable}(SETAF, a)$ for an arbitrary abstract argumentation framework with collective attacks $SETAF = (A, R)$ and an argument $a \in A$ is* coNP*-c.*

Proof. As before, we first use the non-determinism to guess a set S and then, in order to find a counter example, we check in P, cf. Lemma 4, that S is stable and verify that the argument a under question is not in the set. Thus, $Skept_{\text{stable}}(SETAF, a)$ is in NP. □

Finally, we will give NP procedures for the $Exists_{\text{stable}}(SETAF)$ and $Exists^{\neg\emptyset}_{\text{stable}}(SETAF)$ problems.

Lemma 7. *Deciding $Exists_{stable}(SETAF)$ and $Exists^{\neg\emptyset}_{stable}(SETAF)$ for an arbitrary abstract argumentation framework with collective attacks $SETAF = (A, R)$ is* NP*-c.*

Proof. Again, we first use the non-determinism to guess a set S and verify that it is conflict-free, which can be done in P, cf. Lemma 2. Next, we iterate over all arguments $a \in A \setminus S$ and for each such argument we iterate over all attacks and check whether there is an attack $(B, a) \in R$ such that $B \subseteq S$. Moreover, for $Exists^{\neg\emptyset}_{\text{stable}}(SETAF)$, we additional verify that S is not empty, giving NP-c procedures for $Exists_{\text{stable}}(SETAF)$ and $Exists^{\neg\emptyset}_{\text{stable}}(SETAF)$. □

3.1.3 Admissible semantics

In this subsection, we will prove our results for the admissible semantics, depicted in Table 3.1. We first observe, that the empty set is always admissible. Thus, for any abstract argumentation framework with collective attacks $SETAF = (A, R)$ and argument $a \in A$, $Skept_{\text{admissible}}(SETAF, a)$ is trivially false and $Exists_{\text{admissible}}(SETAF)$ is trivially true,.

Regarding the remaining decision problems, we firstly consider the verification problem.

Lemma 8. *Deciding $Ver_{admissible}(SETAF, S)$ for an arbitrary abstract argumentation framework with collective attacks $SETAF = (A, R)$ and a set of arguments $S \subseteq A$ is in* P*.*

Proof. One can verify that a given set S is admissible, by first checking that it is conflict-free and then, for each $a \in S$, that $a \in \mathcal{F}(S)$. Both can be done in P, cf. Lemma 2 and Proposition 1, giving a P procedure for deciding $Ver_{\text{admissible}}(SETAF, S)$. □

Next, we prove our results for the credulous acceptance problem.

Lemma 9. *Deciding $Cred_{admissible}(SETAF, a)$ for an arbitrary abstract argumentation framework with collective attacks $SETAF = (A, R)$ and an argument $a \in A$ is* NP*-c.*

Proof. As for the stable semantics, we first use the non-determinism to guess a set S and then check whether the argument a under question is in the set. Finally, we check that S is admissible, which can be done in P, cf. Lemma 8, resulting in a NP procedure to decide $Cred_{\text{admissible}}(SETAF, a)$. □

Finally, we will give a NP procedure for the $Exists^{\neg\emptyset}_{\text{admissible}}(SETAF)$ problem.

Lemma 10. *Deciding $Exists_{admissible}(SETAF)$ for an arbitrary abstract argumentation framework with collective attacks $SETAF = (A, R)$ is* NP*-c.*

Proof. Again, we first use the non-determinism to guess a non-empty set S. Next, we check that S is admissible, which can be done in P, cf. Lemma 8, resulting in a NP procedure to decide $Exists^{\neg\emptyset}_{\text{admissible}}(SETAF)$. □

3.1.4 Complete semantics

As in the previous subsections, we will prove our results for the complete semantics, depicted in Table 3.1, in this subsection. Furthermore, the extension obtained by iteratively applying the characteristic function $\mathcal{F}$, starting with the empty set, until the least fixed-point is reached, is called the ground extension of an abstract argumentation framework with collective attacks. This is the unique minimal complete extension of this abstract argumentation framework with collective attacks and thus can be computed in polynomial time. Furthermore, such an extension always exists and therefore, $Exists_{\text{complete}}(SETAF)$ is trivially true for any abstract argumentation framework with collective attacks $SETAF = (A, R)$. Moreover, $Skept_{\text{complete}}(SETAF, a)$ is decidable in P as it is true if and only if a is contained in the ground extension.

Next, we consider the verification problem.

Lemma 11. *Deciding $Ver_{complete}(SETAF, S)$ for an arbitrary abstract argumentation framework with collective attacks $SETAF = (A, R)$ and a set of arguments $S \subseteq A$ is in* P*.*

Proof. We can verify that a given set S is a complete extension, by

- checking that it is admissible and
- for each $a \in A \setminus S$, check that $a \notin \mathcal{F}(S)$.

Both can be done in P, cf. Lemma 8 and Proposition 1, resulting in a P procedure to decide $Ver_{\text{complete}}(SETAF, S)$. □

Finally, we prove the results for the credulous acceptance and the non-empty existence problem.

Lemma 12. *Deciding $Cred_{complete}(SETAF, a)$ and $Exists^{\neg\emptyset}_{complete}(SETAF)$ for an arbitrary abstract argumentation framework with collective attacks $SETAF = (A, R)$ and an argument $a \in A$ is* NP*-c.*

Proof. As for the admissible semantics, we first use the non-determinism to guess a set S. For the $Cred_{\text{complete}}(SETAF, a)$ problem we additional check, that the argument a under question is in the set. Next, we check that S is admissible and iterate over all $b \in A \setminus S$ and verify that $b \notin \mathcal{F}(S)$, which can be done in P, cf. Lemma 8 and Proposition 1, resulting in a NP procedure to decide $Cred_{\text{complete}}(SETAF, a)$ and $Exists^{\neg\emptyset}_{\text{complete}}(SETAF)$. □

3.1.5 Preferred semantics

In this subsection, we will provide proofs for our complexity results for the preferred semantics, depicted in Table 3.1. Firstly, as each abstract argumentation framework with collective attacks $SETAF = (A, R)$ has a preferred extensions, as there always is some admissible extension, $Exists_{\text{preferred}}(SETAF)$ is trivially true.

Secondly, we will consider the verification problem.

Lemma 13. *Deciding $Ver_{preferred}(SETAF, S)$ for an arbitrary abstract argumentation framework with collective attacks $SETAF = (A, R)$ and a set of arguments $S \subseteq A$ is* coNP*-c.*

Proof. One can verify that a given set S is a preferred extension, by

- checking that it is admissible and
- verifying that each $S' \supset S$ is not admissible.

The former is in P, cf. Lemma 8. For the latter, in order to find a counter example, we utilize the non-determinism to guess a set $S' \subseteq A$ and verify in P that it is admissible and a proper superset of S, resulting in a coNP algorithm to decide $Ver_{\text{preferred}}(SETAF, S)$. □

Next, we will prove our result for the credulous acceptance and the non-empty existence problem.

Lemma 14. *Deciding $Cred_{preferred}(SETAF, a)$ and $Exists^{\neg\emptyset}_{preferred}(SETAF)$ for an arbitrary abstract argumentation framework with collective attacks $SETAF = (A, R)$ and an argument $a \in A$ is* NP*-c.*

Proof. We can exploit the fact that an argument is contained in a preferred extension if and only if it is contained in some admissible extension. Thus, $Cred_{\text{preferred}}(SETAF, a) = Cred_{\text{admissible}}(SETAF, a)$ and $Exists^{\neg\emptyset}_{\text{preferred}}(SETAF) = Exists^{\neg\emptyset}_{\text{admissible}}(SETAF)$ and thus, $Cred_{\text{preferred}}(SETAF, a)$ and $Exists^{\neg\emptyset}_{\text{preferred}}(SETAF)$ are decidable in NP, as both problems $Cred_{\text{admissible}}(SETAF, a)$ and $Exists^{\neg\emptyset}_{\text{admissible}}(SETAF)$ are in NP, cf. Table 3.1. □

Finally, we will provide a Π^{P}_2 algorithm for the skeptical acceptance problem.

Lemma 15. *Deciding* $Skept_{preferred}(SETAF, a)$ *for an arbitrary abstract argumentation framework with collective attacks* $SETAF = (A, R)$ *and an argument* $a \in A$ *is* Π_2^{P}*-c.*

Proof. In order to find a counter example, we utilize the non-determinism to guess a set $S \subseteq A \setminus \{a\}$ and verify in P that it is admissible, cf. Lemma 8. Next, we utilize the non-determinism once more to verify that S is maximal admissible, by guessing a set $S' \subseteq A$ and verifying that it is admissible and that $S \subset S'$, resulting in a Π_2^{P} procedure. □

3.1.6 Semi-stable semantics

We will present proofs for our complexity results for semi-stable semantics, depicted in Table 3.1, in this subsection. Firstly, as there always is some admissible extension, each abstract argumentation framework with collective attacks $SETAF = (A, R)$ has a semi-stable extension and thus, $Exists_{\text{semi-stable}}(SETAF)$ is trivially true.

Secondly, we consider the verification problem.

Lemma 16. *Deciding* $Ver_{semi\text{-}stable}(SETAF, S)$ *for an arbitrary abstract argumentation framework with collective attacks* $SETAF = (A, R)$ *and a set of arguments* $S \subseteq A$ *is* coNP*-c.*

Proof. We can verify that a given set S is semi-stable by

- checking that it is admissible and
- verifying that each $S' \subseteq A$ with $S'^{+}_{R} \supset S^{+}_{R}$ is not admissible.

The former is in P, cf. Lemma 8. For the latter, we utilize the non-determinism to guess a set $S' \subseteq A$ and verify in P that it is admissible and that its range is a proper superset of the range of S, resulting in a coNP algorithm to decide $Ver_{\text{semi-stable}}(SETAF, S)$. □

Thirdly, we will prove our result for the non-empty existence problem.

Lemma 17. *Deciding* $Exists^{\neg\emptyset}_{semi\text{-}stable}(SETAF)$ *for an arbitrary abstract argumentation framework with collective attacks* $SETAF = (A, R)$ *is* NP*-c.*

Proof. We have that there is a non-empty semi-stable extension if and only if there is a non-empty admissible set. Therefore, $Exists^{\neg\emptyset}_{\text{semi-stable}}(SETAF, a) = Exists^{\neg\emptyset}_{\text{admissible}}(SETAF, a)$ and thus, $Exists^{\neg\emptyset}_{\text{semi-stable}}(SETAF, a)$ is decidable in NP as $Exists^{\neg\emptyset}_{\text{admissible}}(SETAF, a)$ is. □

Next, we will give a Σ_2^{P} procedure for the credulous acceptance problem.

Lemma 18. *Deciding* $Cred_{semi\text{-}stable}(SETAF, a)$ *for an arbitrary abstract argumentation framework with collective attacks* $SETAF = (A, R)$ *and an argument* $a \in A$ *is* Σ_2^{P}*-c.*

Proof. We utilize the non-determinism to guess a set $S \subseteq A$ such that $a \in S$ and verify in P, cf. Lemma 8, that it is admissible. Next, we make another non-deterministic guess for a set $S' \subseteq A$ and verify that it is admissible and that $S_R^+ \subset S_R'^+$, resulting in a Σ_2^{P} procedure. □

Finally, we give an algorithm that decides the skeptical acceptance problem.

Lemma 19. *Deciding* $Skept_{semi\text{-}stable}(SETAF, a)$ *for an arbitrary abstract argumentation framework with collective attacks* $SETAF = (A, R)$ *and an argument* $a \in A$ *is* Π_2^{P}*-c.*

Proof. To find a counter example, we utilize the non-determinism to guess a set $S \subseteq A$ such that $a \notin S$ and verify in P, cf. Lemma 8, that it is admissible. Next, we make another non-deterministic guess for a set $S' \subseteq A$ and verify that it is admissible and that $S_R^+ \subset S_R'^+$, resulting in a Π_2^{P} procedure. □

3.1.7 Stage semantics

In this subsection, we will present proofs for our complexity results for stage semantics, depicted in Table 3.1. Firstly, as there always is some conflict-free extension, each abstract argumentation framework with collective attacks $SETAF = (A, R)$ has a stage extension and thus, $Exists_{\text{stage}}(SETAF)$ is trivially true.

Secondly, we consider the verification problem.

Lemma 20. *Deciding* $Ver_{stage}(SETAF, S)$ *for an arbitrary abstract argumentation framework with collective attacks* $SETAF = (A, R)$ *and a set of arguments* $S \subseteq A$ *is* coNP*-c.*

Proof. We can verify that a given set S is a stage extension by

- checking that it is conflict-free and
- verifying that each $S' \subseteq A$ with $S_R'^+ \supset S_R^+$ is not conflict-free.

The former is in P, cf. Lemma 2. For the latter, we utilize the non-determinism to guess a set $S' \subseteq A$ and verify in P that it is conflict-free and that its range is a proper superset of the range of S, resulting in a coNP algorithm to decide $Ver_{\text{stage}}(SETAF, S)$. □

Thirdly, we will prove our result for the non-empty existence problem.

Lemma 21. *Deciding* $Exists_{stage}^{\neg\emptyset}(SETAF)$ *for an arbitrary abstract argumentation framework with collective attacks* $SETAF = (A, R)$ *is in* P*.*

Proof. We have that there is a non-empty stage extension if and only if there is a non-empty conflict-free set. Therefore, $\mathit{Exists}^{\neg\emptyset}_{\text{stage}}(\mathit{SETAF}, a) = \mathit{Exists}^{\neg\emptyset}_{\text{conflict-free}}(\mathit{SETAF}, a)$ and thus, $\mathit{Exists}^{\neg\emptyset}_{\text{stage}}(\mathit{SETAF}, a)$ is decidable in P, as $\mathit{Exists}^{\neg\emptyset}_{\text{conflict-free}}(\mathit{SETAF}, a)$ is, cf. Table 3.1. □

Next, we will give a Σ^{P}_2 procedure for the credulous acceptance problem.

Lemma 22. *Deciding $\mathit{Cred}_{stage}(\mathit{SETAF}, a)$ for an arbitrary abstract argumentation framework with collective attacks $\mathit{SETAF} = (A, R)$ and an argument $a \in A$ is Σ^{P}_2-c.*

Proof. We utilize the non-determinism to guess a set $S \subseteq A$ such that $a \in S$ and verify in P, cf. Lemma 2, that it is conflict-free. Next, we make another non-deterministic guess for a set $S' \subseteq A$ and verify that it is conflict-free and that $S^+_R \subset S'^+_R$, resulting in a Σ^{P}_2 procedure. □

Finally, we give an algorithm that decides the skeptical acceptance problem.

Lemma 23. *Deciding $\mathit{Skept}_{stage}(\mathit{SETAF}, a)$ for an arbitrary abstract argumentation framework with collective attacks $\mathit{SETAF} = (A, R)$ and an argument $a \in A$ is Π^{P}_2-c.*

Proof. To find a counter example, we utilize the non-determinism to guess a set $S \subseteq A$ such that $a \notin S$ and verify in P, cf. Lemma 2, that it is conflict-free. Next, we make another non-deterministic guess for a set $S' \subseteq A$ and verify that it is conflict-free and that $S^+_R \subset S'^+_R$, resulting in a Π^{P}_2 procedure. □

3.2 Claim augmented abstract argumentation frameworks

The complexity results of the semi-stable, stage, semi-stable$_c$ and stage$_c$ semantics for non well-formed claim augmented abstract argumentation frameworks are depicted in Table 3.2, while the results for well-formed claim augmented abstract argumentation frameworks are depicted in Table 3.3.

For both, non well-formed and well-formed claim augmented abstract argumentation frameworks, we get hardness from the results for Dung abstract argumentation frameworks, cf. Table 2.1, for the Cred_σ, Skept_σ and $\mathit{Exists}^{\neg\emptyset}_\sigma$ problems for $\sigma \in \{\text{semi-stable}, \text{stage}, \text{semi-stable}_c, \text{stage}_c\}$. For a given Dung abstract argumentation framework $AF = (A, R)$, we can construct a well-formed claim augmented abstract argumentation framework $CAF = (A, R, \gamma : a \mapsto a)$ such that $E = \{a_1, \ldots, a_n\}$ is an extension of AF under the semi-stable and stage semantics if and only if E is an extension of CAF under the same semantics or the respective claim-centric variant, as there is a one-to-one mapping between arguments and claims. Furthermore, the *Exists* problem is trivial for semi-stable, stage, semi-stable$_c$, stage$_c$, as there is always a conflict-free and an admissible extension. Thus, in the following subsections, we will give upper bounds for the Cred_σ, Skept_σ, Ver_σ and $\mathit{Exists}^{\neg\emptyset}_\sigma$ as well as additional lower bounds for the Ver_σ problem for non well-formed

σ	$Cred_\sigma$	$Skept_\sigma$	Ver_σ	$Exists_\sigma$	$Exists_\sigma^{\neg\emptyset}$
semi-stable	Σ_2^P-c	Π_2^P-c	Σ_2^P-c	trivial	NP-c
stage	Σ_2^P-c	Π_2^P-c	Σ_2^P-c	trivial	in P
semi-stable$_c$	Σ_2^P-c	Π_2^P-c	Σ_2^P-c	trivial	NP-c
stage$_c$	Σ_2^P-c	Π_2^P-c	Σ_2^P-c	trivial	in P

Table 3.2: Complexity landscape for non well-formed claim augmented abstract argumentation frameworks

σ	$Cred_\sigma$	$Skept_\sigma$	Ver_σ	$Exists_\sigma$	$Exists_\sigma^{\neg\emptyset}$
semi-stable	Σ_2^P-c	Π_2^P-c	coNP-c	trivial	NP-c
stage	Σ_2^P-c	Π_2^P-c	coNP-c	trivial	in P
semi-stable$_c$	Σ_2^P-c	Π_2^P-c	coNP-c	trivial	NP-c
stage$_c$	Σ_2^P-c	Π_2^P-c	coNP-c	trivial	in P

Table 3.3: Complexity landscape for well-formed claim augmented abstract argumentation frameworks

claim augmented abstract argumentation frameworks. While the lower bounds for the Ver_σ problem of the semi-stable, stage, semi-stable$_c$ and stage$_c$ semantics for well-formed claim augmented abstract argumentation frameworks carry over from Dung abstract argumentation frameworks, by the method stated above.

3.2.1 Semi-stable semantics

In this subsection, we will prove our results for the semi-stable semantics for non well-formed and well-formed claim augmented abstract argumentation frameworks, depicted in the Tables 3.2 and 3.3.

Firstly, we will consider the verification problem for non well-formed claim augmented abstract argumentation frameworks.

Lemma 24. *Deciding $Ver_{\text{semi-stable}}(CAF, S)$ for an arbitrary non well-formed claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$ and a set of claims $S \subseteq claim(A)$ is Σ_2^P-c.*

Proof. Recall that we get hardness from the results for Dung abstract argumentation frameworks, cf. Table 2.1. This also applies for the remaining proofs for claim augmented abstract argumentation frameworks.

We can verify that a given set S is a semi-stable extension, by

- guessing a set $E \subseteq A$ with $claim(E) = S$,

- checking that E is a semi-stable extension of (A, R).

The first is in NP, while the second is known to be in coNP from Dung abstract argumentation frameworks, cf. Table 2.1, yielding a Σ_2^P algorithm.

Furthermore, we can show hardness by a reduction from $Cred_{\text{semi-stable}}$ for Dung abstract argumentation frameworks. Let $AF = (A', R')$ be a Dung abstract argumentation framework and b an argument for the $Cred_{\text{semi-stable}}(AF, b)$ problem. We construct a claim augmented abstract argumentation framework $CAF' = (A'' = A' \cup \{x\}, R', \gamma')$ with $x \notin A'$ and $\gamma'(b) = c_1$, $\gamma'(b') = c_2$ for all $b' \in A'' \setminus \{b\}$. Then, as the argument x is not involved in any attack, it is contained in every semi-stable set of arguments of CAF' and thus, as $\gamma'(x) = c_2$, c_2 is contained in every semi-stable extension of CAF'. Therefore, as CAF' only contains two claims, the only candidate sets for semi-stable extensions are $\{c_1, c_2\}$ and $\{c_2\}$ Moreover, as b is the only argument with claim c_1, $\{c_1, c_2\}$ is a semi-stable extension if and only if the argument b is contained in some semi-stable set of arguments of CAF'. Thus, $Cred_{\text{semi-stable}}(AF, b)$ is true if and only if $Ver_{\text{semi-stable}}(CAF', \{c_1, c_2\})$ is true. As the $Cred_{\text{semi-stable}}$ problem for Dung abstract argument frameworks is known to be Σ_2^P-c, cf. Table 2.1, the $Ver_{\text{semi-stable}}$ problem is Σ_2^P-hard for non well-formed claim augmented abstract argumentation frameworks. □

Next, we will consider the verification problem for well-formed claim augmented abstract argumentation frameworks.

Lemma 25. *Deciding $Ver_{\text{semi-stable}}(CAF, S)$ for an arbitrary well-formed claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$ and a set of claims $S \subseteq claim(A)$ is* coNP*-c.*

Proof. We can verify that a given set S is a semi-stable extension, by

- calculating the maximal admissible set $E \subseteq A$ with $claim(E) = S$ and
- verifying that each $E' \subseteq A$ with $E_R^+ \subset E_R'^+$ is not admissible.

The first is in P as shown in [DW19] because CAF is well-formed, while towards a counter example, we can utilize the non-determinism to guess the set $E' \subseteq A$, verify in P that it is admissible, cf. Table 2.1, and that its range is a proper superset of the range of E, yielding a coNP algorithm for deciding $Ver_{\text{semi-stable}}(CAF, S)$. □

Note that the complexity results for the remaining decision problems coincide for non well-formed and well-formed claim augmented abstract argumentation frameworks, thus we will not have to distinguish between those for the remaining proofs, starting with the proof for the credulous acceptance problem.

Lemma 26. *Deciding $Cred_{\text{semi-stable}}(CAF, c)$ for an arbitrary claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$ and a claim $c \in claim(A)$ is Σ_2^P-c.*

Proof. We utilize the non-determinism to guess a set of arguments $S \subseteq A$ such that there is some argument $a \in S$ with claim $\gamma(a) = c$. Next, we verify that S is a semi-stable extension of (A, R), which is coNP-c, cf. Table 2.1, yielding a Σ_2^P-c algorithm. □

Analogously, we provide a procedure for the skeptical acceptance problem.

Lemma 27. *Deciding* $Skept_{semi\text{-}stable}(CAF, c)$ *for an arbitrary claim augmented abstract argumentation framework* $CAF = (A, R, \gamma)$ *and a claim* $c \in claim(A)$ *is* Π_2^P*-c.*

Proof. To construct a counter example, we utilize the non-determinism to guess a set of arguments $S \subseteq A$ such that there no argument $a \in S$ with claim $\gamma(a) = c$. Next, we verify that S is a semi-stable extension, which is coNP-c, cf. Table 2.1, yielding a Σ_2^P-c algorithm. □

Finally, we present our algorithm for deciding the non-empty existence problem.

Lemma 28. *Deciding* $Exists^{\neg\emptyset}_{semi\text{-}stable}(CAF)$ *for an arbitrary claim augmented abstract argumentation framework* $CAF = (A, R, \gamma)$ *is* NP*-c.*

Proof. We have that there is a non-empty semi-stable extension if and only if there is a non-empty admissible set, thus $Exists^{\neg\emptyset}_{\text{semi-stable}}(CAF) = Exists^{\neg\emptyset}_{\text{admissible}}((A, R))$. As the latter is known to be NP-c, cf. Table 2.1, so is $Exists^{\neg\emptyset}_{\text{semi-stable}}(CAF)$. □

3.2.2 Stage semantics

We will prove our results for the stage semantics for non well-formed and well-formed claim augmented abstract argumentation frameworks, depicted in the Tables 3.2 and 3.3, in this subsection.

Firstly, we will consider the verification problem for non well-formed claim augmented abstract argumentation frameworks.

Lemma 29. *Deciding* $Ver_{stage}(CAF, S)$ *for an arbitrary non well-formed claim augmented abstract argumentation framework* $CAF = (A, R, \gamma)$ *and a set of claims* $S \subseteq claim(A)$ *is* Σ_2^P*-c.*

Proof. We can verify that a given set S is a stage extension, by

- guessing a set $E \subseteq A$ with $claim(E) = S$,
- checking that E is a stage extension of (A, R).

The first is in NP, while the second is known to be in coNP from Dung abstract argumentation frameworks, cf. Table 2.1, yielding a Σ_2^P algorithm.

Furthermore, we can show hardness by a reduction from $Cred_{\text{stage}}$ for Dung abstract argumentation frameworks. Let $AF = (A', R')$ be a Dung abstract argumentation framework and b an argument for the $Cred_{\text{stage}}(AF, b)$ problem. We construct a claim augmented abstract argumentation framework $CAF' = (A'' = A' \cup \{x\}, R', \gamma')$ with $x \notin A'$ and $\gamma'(b) = c_1$, $\gamma'(b') = c_2$ for all $b' \in A'' \setminus \{b\}$. Then, as the argument x is not involved in any attack, it is contained in every stage set of arguments of CAF' and thus, as $\gamma'(x) = c_2$, c_2 is contained in every stage extension of CAF'. Therefore, as CAF' only contains two claims, the only candidate sets for stage extensions are $\{c_1, c_2\}$ and $\{c_2\}$ Moreover, as b is the only argument with claim c_1, $\{c_1, c_2\}$ is a stage extension if and only if the argument b is contained in some stage set of arguments of CAF'. Thus, $Cred_{\text{stage}}(AF, b)$ is true if and only if $Ver_{\text{stage}}(CAF', \{c_1, c_2\})$ is true. As the $Cred_{\text{stage}}$ problem for Dung abstract argument frameworks is known to be Σ_2^P-c, cf. Table 2.1, the Ver_{stage} problem is Σ_2^P-hard for non well-formed claim augmented abstract argumentation frameworks. □

Next, we will consider the verification problem for well-formed claim augmented abstract argumentation frameworks.

Lemma 30. *Deciding $Ver_{stage}(CAF, S)$ for an arbitrary well-formed claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$ and a set of claims $S \subseteq claim(A)$ is* coNP*-c.*

Proof. We can verify that a given set S is a stage extension, by

- calculating the maximal conflict-free set $E \subseteq A$ with $claim(E) = S$ and
- verifying that each $E' \subseteq A$ with $E_R^+ \subset E_R'^+$ is not conflict-free.

The first is in P as shown in [DW19] because CAF is well-formed, while towards a counter example, we can utilize the non-determinism to guess the set $E' \subseteq A$, verify in P that it is conflict-free, cf. Table 2.1, and that its range is a proper superset of the range of E, yielding a coNP algorithm for deciding $Ver_{\text{stage}}(CAF, S)$. □

Note that the complexity results for the remaining decision problems coincide for non well-formed and well-formed claim augmented abstract argumentation frameworks, thus we will not have to distinguish between those for the remaining proofs, starting with the proof for the credulous acceptance problem.

Lemma 31. *Deciding $Cred_{stage}(CAF, c)$ for an arbitrary claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$ and a claim $c \in claim(A)$ is Σ_2^P-c.*

Proof. We utilize the non-determinism to guess a set of arguments $S \subseteq A$ such that there is some argument $a \in S$ with claim $\gamma(a) = c$. Next, we verify that S is a stage extension, which is coNP-c, cf. Table 2.1, yielding a Σ_2^P-c algorithm. □

Analogously, we provide a procedure for the skeptical acceptance problem.

Lemma 32. *Deciding* $Skept_{stage}(CAF, c)$ *for an arbitrary claim augmented abstract argumentation framework* $CAF = (A, R, \gamma)$ *and a claim* $c \in claim(A)$ *is* Π_2^P*-c.*

Proof. To construct a counter example, we utilize the non-determinism to guess a set of arguments $S \subseteq A$ such that there no argument $a \in S$ with claim $\gamma(a) = c$. Next, we verify that S is a stage extension, which is coNP-c, cf. Table 2.1, yielding a Σ_2^P-c algorithm. □

Finally, we present our algorithm for deciding the non-empty existence problem.

Lemma 33. *Deciding* $Exists^{\neg\emptyset}_{stage}(CAF)$ *for an arbitrary claim augmented abstract argumentation framework* $CAF = (A, R, \gamma)$ *is in* P*.*

Proof. We have that there is a non-empty stage extension if and only if there is a non-empty conflict-free set, thus $Exists^{\neg\emptyset}_{\text{stage}}(CAF) = Exists^{\neg\emptyset}_{\text{conflict-free}}((A, R))$. As the latter is known to be in P, cf. Table 2.1, so is $Exists^{\neg\emptyset}_{\text{stage}}(CAF)$. □

3.2.3 Semi-stable semantics with claim-centric range

Before we give the complexity results for the semi-stable$_c$ semantics, we first show some auxiliary results, adapting some of the translations defined in [DW11].

Definition 2. Given a claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$. We define translation $Tr_1(CAF) = (A', R', \gamma')$ where

$$\begin{aligned} A' &= A \cup \{a' \mid a \in A\} \\ R' &= R \cup \{(a, a'), (a', a), (a', a') \mid a \in A\} \\ \gamma'(a') &= c_a \text{ and } \gamma'(a) = \gamma(a) \text{ for all } a \in A \text{ such that } c_a \notin claim(A) \end{aligned}$$

Lemma 34. *For every claim augmented abstract argumentation framework* $CAF = (A, R, \gamma)$*, the following three propositions are equivalent:*

1. $C \in preferred(CAF)$
2. $C \in preferred(Tr_1(CAF))$
3. $C \in semi\text{-}stable_c(Tr_1(CAF))$

Proof. We first show that $C \in$ conflict-free(CAF) if and only if $C \in$ conflict-free$(Tr_1(CAF))$:
$\Rightarrow$: Let $E \subseteq A$ be conflict-free in CAF such that $claim(E) = C$. As $E \subseteq A$, it cannot contain any a'. Thus, E is conflict-free in $Tr_1(CAF)$, as all additional attacks contain at least one argument a', which are not contained in E and therefore $C \in$ conflict-free$(Tr_1(CAF))$.
$\Leftarrow$: Let $E \subseteq A'$ conflict-free in $Tr_1(CAF)$ such that $claim(E) = C$. As all arguments a' are self-attacking, they cannot be contained in any conflict-free set of $Tr_1(CAF)$, thus E cannot contain any such arguments. Furthermore, as $R \subseteq R'$, E is a conflict-free set in CAF and thus $C \in$ conflict-free(CAF).
Moreover, clearly $E \in$ admissible(A, R) if and only if $E \in$ admissible$(Tr_1(A, R))$, as all attacks from any argument a' are symmetric, i.e. every argument a attacked by some argument a' defends itself against a'. Furthermore, as preferred extensions are subset maximal admissible sets, we obtain that $E \in$ preferred(A, R) if and only if $E \in$ preferred$(Tr_1(A, R))$. Thus, (1) $\Leftrightarrow$ (2).
Next, for (2) $\Rightarrow$ (3), let C be a preferred extension of $Tr_1(CAF)$ and $E \subseteq A'$ be a preferred set witnessing C. Furthermore, towards a contradiction, let $F \subseteq A'$ be an admissible set and $E^{+claim}_{R'} \subset F^{+claim}_{R'}$. As E is a preferred set of CAF, there must be some $a \in E \setminus F$. Furthermore, as all arguments $b' \in A' \setminus A$ are not conflict-free, it must hold that $a \in A$ and thus, by the construction of Tr_1, there must be some argument a' such that a is the only argument attacking a' and a' is the only argument with claim $\gamma(a')$. Therefore, $\gamma(a') \in E^{+claim}_{R'}$ but $\gamma(a') \notin F^{+claim}_{R'}$, contradicting that $E^{+claim}_{R'} \subset F^{+claim}_{R'}$. Thus, such a set F cannot exist and therefore, (2) $\Rightarrow$ (3).
Finally, for (3) $\Rightarrow$ (2), let C be a semi-stable$_c$ extension of $Tr_1(CAF)$ and $E \subseteq A'$ be a admissible set witnessing C. Towards a contradiction, let $F \subseteq A'$ be a preferred set such that $E \subset F$. Then, $E^{+claim}_{R'} \subseteq F^{+claim}_{R'}$. Furthermore, as $E \subset F$, there must be some $a \in F \setminus E$ and thus some $a' \in A'$ attacked by a. As, by the construction of Tr_1, a' is the only argument with claim $\gamma(a')$ and is only attacked by a (except for itself), $\gamma(a') \in F^{+claim}_{R'}$ and $\gamma(a') \notin E^{+claim}_{R'}$ and thus $E^{+claim}_{R'} \subset F^{+claim}_{R'}$, contradicting that C is a semi-stable$_c$ extension of $Tr_1(CAF)$. Thus, such a set F cannot exist and therefore, (3) $\Rightarrow$ (2). □

Using this lemma, we will prove our complexity results the semi-stable$_c$ semantics for non well-formed and well-formed claim augmented abstract argumentation frameworks, depicted in the Tables 3.2 and 3.3.

Firstly, we will consider the verification problem for non well-formed claim augmented abstract argumentation frameworks.

Lemma 35. *Deciding $Ver_{\text{semi-stable}_c}(CAF, S)$ for an arbitrary non well-formed claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$ and a set of claims $S \subseteq claim(A)$ is Σ^{P}_2-c.*

Proof. We can verify that a given set S is a semi-stable$_c$ extension, by

- guessing a set $E \subseteq A$ with $claim(E) = S$,

- checking that E is admissible and
- verifying that each $E' \subseteq A$ with $E_R'^{+_{claim}} \supset E_R^{+_{claim}}$ is not admissible.

The first is in NP, while the second is known to be in P from Dung abstract argumentation frameworks, cf. Table 2.1. Finally, checking that every proper superset is not admissible can be solved in coNP by a standard guess & check algorithm, i.e. guess a set and verify that it is admissible, compute the claims of its range and verify its a superset of the range of the original set, yielding a Σ_2^P algorithm.

Furthermore, we can show hardness by a reduction from $Ver_{\text{preferred}}$ of claim augmented abstract argumentation frameworks. As shown in Lemma 34, $C \in \text{preferred}(CAF)$ if and only if $C \in \text{semi-stable}_c(Tr_1(CAF))$. Thus, we construct $Tr_1(CAF)$ in polynomial time and then $Ver_{\text{preferred}}(CAF, C) = Ver_{\text{semi-stable}_c}(Tr_1(CAF), C)$. Thus, as $Ver_{\text{preferred}}$ for non well-formed claim augmented abstract argumentation frameworks is Σ_2^P-c, cf. Table 2.2, so is $Ver_{\text{semi-stable}_c}$ for non well-formed claim augmented abstract argumentation frameworks. □

Next, we will consider the verification problem for well-formed claim augmented abstract argumentation frameworks.

Lemma 36. *Deciding $Ver_{\text{semi-stable}_c}(CAF, S)$ for an arbitrary well-formed claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$ and a set of claims $S \subseteq claim(A)$ is* coNP*-c.*

Proof. We can verify that a given set S is a semi-stable$_c$ extension, by

- calculating the maximal admissible set $E \subseteq A$ with $claim(E) = S$,
- verifying that each $E' \subseteq A$ with $E_R'^{+_{claim}} \supset E_R^{+_{claim}}$ is not admissible.

The first is in P as shown in [DW19], while the second can be solved in coNP by a standard guess & check algorithm, i.e. guess a set and verify that it is admissible and its range is a superset of the range of the original set, thus yielding a coNP algorithm for deciding $Ver_{\text{semi-stable}_c}(CAF, S)$ for well-formed claim augmented abstract argumentation frameworks. □

Note that the complexity results for the remaining decision problems coincide for non well-formed and well-formed claim augmented abstract argumentation frameworks, thus we will not have to distinguish between those for the remaining proofs, starting with the proof for the credulous acceptance problem.

Lemma 37. *Deciding $Cred_{\text{semi-stable}_c}(CAF, c)$ for an arbitrary claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$ and a claim $c \in claim(A)$ is Σ_2^P-c.*

Proof. We utilize the non-determinism to guess a set of arguments $S \subseteq A$ such that there is some argument $a \in S$ with claim $\gamma(a) = c$. Next, we verify that S is admissible and that its range is maximal by guessing another set $S' \subseteq A$, verifying that it is admissible and that its range is a superset of the range of S, yielding a Σ_2^{P}-c algorithm for deciding $\mathit{Cred}_{\text{semi-stable}_c}(CAF, c)$. □

Analogously, we provide a procedure for the skeptical acceptance problem.

Lemma 38. *Deciding $\mathit{Skept}_{\text{semi-stable}_c}(CAF, c)$ for an arbitrary claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$ and a claim $c \in \mathit{claim}(A)$ is Π_2^{P}-c.*

Proof. To construct a counter example, we utilize the non-determinism to guess a set of arguments $S \subseteq A$ such that there no argument $a \in S$ with claim $\gamma(a) = c$. Next, we verify that S is admissible and that its range is maximal by guessing another set $S' \subseteq A$, verifying that it is admissible and that its range is a superset of the range of S, yielding a Π_2^{P}-c algorithm for deciding $\mathit{Skept}_{\text{semi-stable}_c}(CAF, c)$. □

Finally, we present our algorithm for deciding the non-empty existence problem.

Lemma 39. *Deciding $\mathit{Exists}^{\neg\emptyset}_{\text{semi-stable}_c}(CAF)$ for an arbitrary claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$ is NP-c.*

Proof. We have that there is a non-empty semi-stable$_c$ extension if and only if there is a non-empty admissible set, thus $\mathit{Exists}^{\neg\emptyset}_{\text{semi-stable}_c}(CAF) = \mathit{Exists}^{\neg\emptyset}_{\text{admissible}}((A, R))$. As the latter is known to be NP-c, cf. Table 2.1, so is $\mathit{Exists}^{\neg\emptyset}_{\text{semi-stable}_c}(CAF)$. □

3.2.4 Stage semantics with claim-centric range

Similarly as for the semi-stable$_c$ semantics, we first show some auxiliary results, adapting some of the translations defined in [DW11].

Definition 3. Given a claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$. We define translation $Tr_2(CAF) = (A', R', \gamma')$ where

$$\begin{aligned} A' &= A \cup \{a' \mid a \in A\} \\ R' &= R \cup \{(b, a), (a, b'), \mid (a, b) \in R\} \\ &\quad \cup \{(a, b) \mid a \in A, (b, b) \in R\} \\ &\quad \cup \{(a, a'), (a', a') \mid a \in A\} \\ \gamma'(a') &= c_a \text{ and } \gamma'(a) = \gamma(a) \text{ for all } a \in A \text{ such that } c_a \notin \mathit{claim}(A) \end{aligned}$$

Lemma 40. *For every claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$, the following three propositions are equivalent:*

1. $C \in \mathit{stage}(CAF)$

2. $C \in stage(Tr_2(CAF))$

3. $C \in stage_c(Tr_2(CAF))$

Proof. First, observe that every set of arguments is conflict-free in CAF if and only if it is conflict-free in $Tr_2(CAF)$, as all added arguments are self-attacking and thus cannot occur in any conflict-free set and we only add attacks between arguments $\{a, b\} \subseteq A$, if there was already one in at least one direction or the attacked argument was self-attacking. Moreover, $\{\emptyset\}$ is a, and then also the only, stage extension of CAF if and only if all arguments are self-attacking which is the case if and only if $\{\emptyset\}$ is also the only stage$_c$ extension of CAF.
Regarding (1) $\Leftrightarrow$ (3): For a set of arguments E, let $(E_R^+)' := \{a' \in A \mid a \in E_R^+\}$. Clearly, $(E_R^+)' \subseteq E_{R'}^+$, as for any $(a, b) \in R$ we have that $(a, b') \in R'$ and $(a, a') \in R'$ for all $a \in A$. Furthermore, for every maximal (with regard to $\subseteq$) conflict-free set of $Tr_2(CAF)$, $A \subseteq E_{R'}^+$, as all arguments in A are either contained or, due to the fact that E is maximal, are attacked by E. Thus, for every maximal conflict-free set $E \subseteq A$ which, due to the fact that all arguments a' are self-attacking, are the only witnessing candidates for the stage and stage$_c$ extensions, $E_R^{+claim} = claim(A) \cup claim((E_R^+)')$ which, as each argument in $(E_R^+)'$ has a unique claim, will be maximal if and only if $(E_R^+)'$ is maximal.
Analogously, (1) $\Leftrightarrow$ (2) follows by observing that $E_R^+ = A \cup (E_R^+)'$ which will be maximal if and only if $(E_R^+)'$ is maximal. □

In the following, using this lemma, we will prove our complexity results the stage$_c$ semantics for non well-formed and well-formed claim augmented abstract argumentation frameworks, depicted in the Tables 3.2 and 3.3.

Firstly, we will consider the verification problem for non well-formed claim augmented abstract argumentation frameworks.

Lemma 41. *Deciding $Ver_{stage_c}(CAF, S)$ for an arbitrary non well-formed claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$ and a set of claims $S \subseteq claim(A)$ is Σ_2^P-c.*

Proof. We can verify that a given set S is a stage$_c$ extension, by

- guessing a set $E \subseteq A$ with $claim(E) = S$,
- checking that E is conflict-free and
- verifying that each $E' \subseteq A$ with $E_R'^{+claim} \supset E_R^{+claim}$ is not conflict-free.

The first is in NP, while the second is known to be in P from Dung abstract argumentation frameworks, cf. Table 2.1. Finally, checking that every proper superset is not conflict-free can be solved in coNP by a standard guess & check algorithm, i.e. guess a set and verify

that it is conflict-free, compute the claims of its range and verify its a superset of the range of the original set, yielding a Σ_2^{P} algorithm.

Furthermore, we can show hardness by a reduction from Ver_{stage}. As shown in Lemma 40, $Ver_{\text{stage}}(CAF, C)$ is true if and only if $Ver_{\text{stage}_c}(Tr_2(CAF), C)$ is true. Thus, we construct $Tr_2(CAF)$ in polynomial time and then $Ver_{\text{stage}}(CAF, C) = Ver_{\text{stage}_c}(Tr_2(CAF), C)$. Thus, as Ver_{stage} for non well-formed claim augmented abstract argumentation frameworks is Σ_2^{P}-c, cf. Table 3.2, so is Ver_{stage_c} for non well-formed claim augmented abstract argumentation frameworks. □

Next, we will consider the verification problem for well-formed claim augmented abstract argumentation frameworks.

Lemma 42. *Deciding $Ver_{stage_c}(CAF, S)$ for an arbitrary well-formed claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$ and a set of claims $S \subseteq claim(A)$ is* coNP*-c.*

Proof. We can verify that a given set S is a stage$_c$ extension, by

- calculating the maximal conflict-free set $E \subseteq A$ with $claim(E) = S$,
- verifying that each $E' \subseteq A$ with $E'^{+_{claim}}_R \supset E^{+_{claim}}_R$ is not conflict-free.

The first is in P as shown in [DW19], while the second can be solved in coNP by a standard guess & check algorithm, i.e. guess a set and verify that it is conflict-free and its range is a superset of the range of the original set, thus yielding a coNP algorithm for deciding $Ver_{\text{stage}_c}(CAF, S)$ for well-formed claim augmented abstract argumentation frameworks. □

Note that the complexity results for the remaining decision problems coincide for non well-formed and well-formed claim augmented abstract argumentation frameworks, thus we will not have to distinguish between those for the remaining proofs, starting with the proof for the credulous acceptance problem.

Lemma 43. *Deciding $Cred_{stage_c}(CAF, c)$ for an arbitrary claim augmented abstract argumentation framework $CAF = (A, R, \gamma)$ and a claim $c \in claim(A)$ is Σ_2^{P}-c.*

Proof. We utilize the non-determinism to guess a set of arguments $S \subseteq A$ such that there is some argument $a \in S$ with claim $\gamma(a) = c$. Next, we verify that S is conflict-free and that its range is maximal by guessing another set $S' \subseteq A$, verifying that it is conflict-free and that its range is a superset of the range of S, yielding a Σ_2^{P}-c algorithm for deciding $Cred_{\text{stage}_c}(CAF, c)$. □

Analogously, we provide a procedure for the skeptical acceptance problem.

Lemma 44. *Deciding* $Skept_{stage_c}(CAF, c)$ *for an arbitrary claim augmented abstract argumentation framework* $CAF = (A, R, \gamma)$ *and a claim* $c \in claim(A)$ *is* Π_2^{P}*-c.*

Proof. To construct a counter example, we utilize the non-determinism to guess a set of arguments $S \subseteq A$ such that there no argument $a \in S$ with claim $\gamma(a) = c$. Next, we verify that S is conflict-free and that its range is maximal by guessing another set $S' \subseteq A$, verifying that it is conflict-free and that its range is a superset of the range of S, yielding a Π_2^{P}-c algorithm for deciding $Skept_{\text{stage}_c}(CAF, c)$. □

Finally, we present our algorithm for deciding the non-empty existence problem.

Lemma 45. *Deciding* $Exists^{\neg\emptyset}_{stage_c}(CAF)$ *for an arbitrary claim augmented abstract argumentation framework* $CAF = (A, R, \gamma)$ *is in* P*.*

Proof. We have that there is a non-empty stage_c extension if and only if there is a non-empty conflict-free set, thus $Exists^{\neg\emptyset}_{\text{stage}_c}(CAF) = Exists^{\neg\emptyset}_{\text{conflict-free}}((A, R))$. As the latter is known to be in P, cf. Table 2.1, so is $Exists^{\neg\emptyset}_{\text{stage}_c}(CAF)$. □

CHAPTER 4

Answer-Set Programming Encodings

Within this chapter, we will present answer-set programming encodings for the conflict-free, admissible, complete, stable, preferred, semi-stable and stage semantics for abstract argumentation frameworks with collective attacks and claim augmented abstract argumentation frameworks. In Section 4.1 we will discuss encodings for abstract argumentation frameworks with collective attacks, while Section 4.2 will focus on claim augmented abstract argumentation frameworks. Moreover, Subsection 4.2.2 will be dedicated to encodings for non well-formed claim augmented abstract argumentation frameworks, while the Subsection 4.2.3 and 4.2.4 will focus on argument-driven and claim-driven encodings for well-formed claim augmented abstract argumentation frameworks. Finally, in Subsections 4.2.5 and 4.2.6, we will propose encodings for the version of stage and semi-stable semantics with claim-centric ranges.

4.1 Encodings for abstract argumentation frameworks with collective attacks

4.1.1 Input and output format

All encodings for abstract argumentation frameworks with collective attacks will share a common input and output format. The input will consist of facts

$$arg(X).$$

stating that X is an argument. Moreover, facts of the form

$$att(X, Y).$$

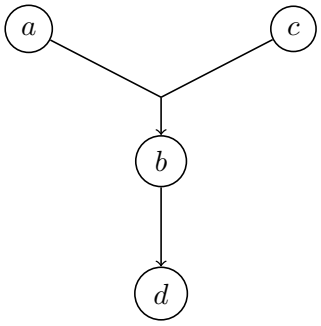

Figure 4.1: An abstract argumentation framework with collective attacks

state that the attack named X is direct at argument Y, while

$$mem(X, Y).$$

encodes that argument Y is a member of attack X.

Furthermore, the output consists of facts

$$in(Y)$$

stating that the argument Y is in the respective extension of the abstract argumentation framework with collective attacks specified in the input.

Thus, the abstract argumentation framework with collective attacks depicted in Figure 4.1 could be encoded as:

$$\Pi_{setaf\_ex} = \begin{cases} \text{arg}(a).\ \text{arg}(b).\ \text{arg}(c).\ \text{arg}(d). \\ \text{att}(a_1,\ b).\ \text{att}(a_2,\ d). \\ \text{mem}(a_1,\ a).\ \text{mem}(a_1,\ c).\ \text{mem}(a_2,\ b). \end{cases}$$

4.1.2 Encodings

All encodings given in this Subsection will share a common core fragment:

$$\Pi_{setaf\_core} = \begin{cases} \text{in(Y) :- arg(Y), not out(Y).} \\ \text{out(Y) :- arg(Y), not in(C).} \\ \text{\#show in/1.} \end{cases}$$

The first two rules of the core fragment model the first guess within the guess and check paradigm, while the third rule projects the output to atoms over the unary predicate ”in”. Using this core fragment, we can define the encoding for conflict-free sets as

$$\Pi_{setaf\_cf} = \Pi_{setaf\_core} \cup \{\text{:- in(Y), att(X, Y), in(Z) : mem(X, Z).}$$

The additional constraint will eliminate all guesses of the core fragment Π_{setaf_core} for which there is an argument Y in the extension of the unary predicate "in" such that there is an attack X direct at Y for which all member arguments $Z_1, \ldots, Z_n$ are also in the extension of the unary predicate "in" and thus are not conflict-free. Moreover, this encoding can be extended to encode two further semantics. First, for stable sets as:

$$\Pi_{setaf\_st} = \Pi_{setaf\_cf} \cup \begin{cases} \text{blocked(X) :- att(X, \_), mem(X, Y), out(Y).} \\ \text{:- out(Y), blocked(X) : att(X, Y).} \end{cases}$$

where the first rule will introduce an auxiliary predicate "blocked(X)", encoding that the attack X is blocked, i.e. cannot fire due to one of its members Y being "out". Thus, the constraint in the second rule will eliminate all guesses that set an argument Y out for which all attacks are blocked.

Secondly, for admissible sets,

$$\Pi_{setaf\_adm} = \Pi_{setaf\_cf} \cup$$
$$\begin{cases} \text{defeated(X) :- att(X, \_), mem(X, Y), att(Z, Y), in(Y2) : mem(Z, Y2).} \\ \text{:- in(Y), att(X, Y), not defeated(X).} \end{cases}$$

encodes the set of all admissible extensions. The first additional rule defines the auxiliary predicate "defeated(X)" with the intended meaning that the attack X is defeated i.e. does not fire. This is the case if the attack includes some argument Y such that there is an attack Z for which all its members are in the extension of the unary predicate "in". Using this auxiliary predicate, the constraint in the second rule eliminates all guesses that include some argument Y in the extension of "in" for which there is an attack that is not defeated. Furthermore, this encoding can be extended to the encoding for complete sets by adding one more constraint:

$$\Pi_{setaf\_co} = \Pi_{setaf\_adm} \cup \{\text{:- out(Y), defeated(X) : att(X, Y).}$$

This additional constraint will eliminate all guesses which set an argument Y "out", even though all attacks on this argument are defeated, i.e. the argument Y is defended by the guess encoded in the extension of "in".

Furthermore, to state the encoding for the preferred semantics, we will make use of the saturation technique mentioned in Subsection 2.2.3. Therefore, we will first give the encoding for the preferred semantics and then explain when and how the saturation technique is used. Thus, the encoding for preferred sets can be implemented as:

$$\Pi_{setaf\_pr} = \Pi_{setaf\_adm} \cup$$

```
non_trivial :- out(_).                                                    (1)
sIn(X) : out(X) :- non_trivial.                                           (2)
sIn(X) :- in(X), non_trivial.                                             (3)
fail :- sIn(Y), att(X, Y), non_trivial, sIn(Z) : mem(X, Z).               (4)
fail | needAttack(X2) : att(X2, Z), mem(X, Z) :- sIn(Y), att(X, Y), non_trivial.  (5)
sIn(Y) :- att(X, _), needAttack(X), mem(X, Y), non_trivial.               (6)
sIn(X) :- fail, arg(X), non_trivial.                                      (7)
needAttack(X) :- fail, att(X, _), non_trivial.                            (8)
:- not fail, non_trivial.                                                 (9)
```

For the preferred semantics, we have to make sure that our guessed set is a maximal admissible set. Thus, for each guess over "in" made by Π_{setaf_adm}, we have to ensure that this guess is maximal admissible and eliminate it otherwise. Therefore, for each guess, additional guesses will be made trying to extend the original guess. Using the saturation technique, we will then ensure that the original guess is either maximal or eliminate it. Unfortunately, we will only be allowed to use stratified default negation in the part where the saturation technique is used (2 - 9), which will make the encoding of admissibility more cumbersome. The basic idea will be to make guesses via "sIn", extending the original guess encoded in "in" and "out". If all additional guesses derive the predicate "fail", the original guess will be maximal admissible. If there is an extended guess that does not derive "fail", the original guess cannot be admissible and thus has to be excluded.

Therefore, we will have to define some auxiliary predicates to model the extended guess. The first one will be "non_trivial", stating that the original guess does not already include all argument. Such a original guess will be trivially maximal and thus there cannot be a extended guess for it, which would cause this guess to be excluded, as there could be no extended guess that could derive "fail". Therefore, to account for this technicality, all rules of the saturation technique will contain "non_trivial" in their positive bodies. For all non-maximal original guesses, we will guess an extended set of arguments via the rules 2 and 3. Furthermore, rule 4 will derive "fail" for all extended guess that are not conflict-free. Finally, the rules 5 and 6 will ensure that the extended guess is admissible or try to extend it if possible or derive "fail". Unfortunately, as mentioned before, we can only use stratified negation, causing the encoding to become more involved as the encoding for Π_{setaf_adm}. Thus, the auxiliary predicate "needAttack(X)" encodes that we need to include all members of the attack X in order to defend some other argument which is already in. To complete the encoding, rules 7 and 8 will saturate all extended guesses deriving "fail", while rule 9 will eliminate all extended guesses not deriving "fail".

As the saturation technique is not quite intuitive, we will try to illustrate the idea behind it. Let $G = \{in(a_1), \ldots, in(a_n)\}$ be some initial guess of Π_{setaf_pr} and assume it encodes a maximal admissible set. Then, if we ignore the trivial case that G is maximal, every

extended guess of G will derive "fail". Therefore, the model M containing G and all the saturated atoms via rule 7 and 8 and "fail" will be a minimal model of the reduct $\Pi^M_{setaf\_pr}$ and thus an answer set. Furthermore, as we have started with a maximal admissible original guess, this is the expected result, as it models a preferred extension. Furthermore, assume G were a non-maximal admissible guess. Then, there would be some admissible extended guess of G and thus an interpretation I modeling this extended guess. However, as I does not derive "fail", the constraint in rule 9 will prevent it from becoming an answer set. Moreover, any extended guess J of the initial guess G that does contain "fail" will, due to the saturation in rule 7 and 8, be an proper superset of I and, as the only occurrence of default negation in the saturation part is in rule 9, which is unsatisfied by I as it does not contain "fail", I will be a model of the reduct $\Pi^J_{setaf\_pr}$ and as $I \subset J$, J cannot be an answer set. Again, this is the expected behavior, as the initial original guess did not model a preferred extension.

Furthermore, leveraging the saturation technique once again, we can define the encoding for stage sets as:

$$\Pi_{setaf\_stg} = \Pi_{setaf\_cf} \cup$$

```
range(Y) :- in(Y).                                                          (1)
range(Y) :- att(X, Y), in(Z) : mem(X, Z).                                   (2)
notrange(Y) :- arg(Y), not range(Y).                                        (3)
non_trivial :- notrange(_).                                                 (4)
extendedRange(Y) :- range(Y), non_trivial.                                  (5)
extendedRange(Y) : notrange(Y) :- non_trivial.                              (6)
baseGuess(Y) | needAttack(X) : att(X, Y) :- extendedRange(Y), non_trivial.  (7)
baseGuess(Y) :- att(X, _), needAttack(X), mem(X, Y), non_trivial.           (8)
fail :- baseGuess(Y), att(X, Y), non_trivial, baseGuess(Z) : mem(X, Z).     (9)
baseGuess(Y) :- fail, arg(Y), non_trivial.                                  (10)
extendedRange(Y) :- fail, arg(Y), non_trivial.                              (11)
needAttack(X) :- fail, att(X, _), non_trivial.                              (12)
:- not fail, non_trivial.                                                   (13)
```

As the stage encoding aims to maximize the range of a set of argument, we first encode the range in rules 1-3 using "range(X)" and "notrange(X)" to encode that an argument X is in the range or not in the range of the original guess respectively. Furthermore, similarly as for the preferred encoding, "non_trivial" in rule 4 will be used to ensure that the range of the original guess is not already maximal and rules 5 and 6 will guess an extended range based on the original guess of Π_{setaf_cf}, although this time trying to extend the range and not the original guess itself. The remaining part of the program, except for the parts required for the saturation, will try to justify the guessed extended range by finding a conflict-free base guess that will witness it or infer "fail". Thus, by

rule 7, for every argument Y postulated to be in the extended range, we either require Y to be in the base guess or require some attack X direct at Y to be supported by the base guess, i.e. require all members of X to be in the base guess, which will be enforced by rule 8. Finally, rule 9 will derive "fail" for all base guesses that are not conflict-free, while rules 10 to 13 saturate such base guesses as for the preferred encoding.

Finally, given the encoding for the stage semantics, the encoding for semi-stable sets can be stated by adding the encoding for admissibility and one more rule:

$$\Pi_{setaf\_sst} = \Pi_{setaf\_stg} \cup \Pi_{setaf\_adm} \cup$$
$$\{\text{fail} \mid \text{needAttack(Z)} :\text{- att(Z, Y1)}, \text{mem(X, Y1)} :\text{- baseGuess(Y)}, \text{att(X, Y)}, \text{non\_trivial}.$$

The additional rule will ensure that the base guess is admissible. Thus, for every argument Y attacked by some attack X, we need the base guess to support some attack Z directed at a member Y1 of X or infer "fail".

4.2 Encodings for claim augmented abstract argumentation frameworks

In this section, we will give answer-set encodings to compute conflict-free, admissible, complete, preferred, stable, semi-stable and stage extension of a claim augmented abstract argumentation frameworks. Furthermore, we will give encodings for non well-formed claim augmented abstract argumentation frameworks as well optimized encodings for well-formed claim augmented abstract argumentation frameworks, exploiting the uniqueness property introduced in [DW19]. Moreover, for well-formed claim augmented abstract argumentation frameworks, we will give argument-driven and claim-driven encodings, one focusing on guessing on arguments, while the other will make guesses on claims.

4.2.1 Input and output format

Analogously as in the encodings for abstract argumentation framework with collective attacks, all encodings for claim augmented abstract argumentation frameworks will share a common input and output format. The input consists of facts

$$arg(X, C).$$

stating that C is the claim of argument X and

$$att(X, Y).$$

stating that argument/claim X attacks argument Y for non well-formed/well-formed claim augmented abstract argumentation frameworks.

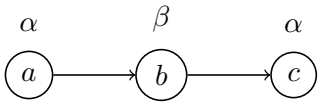

Figure 4.2: A non well-formed claim augmented abstract argumentation framework

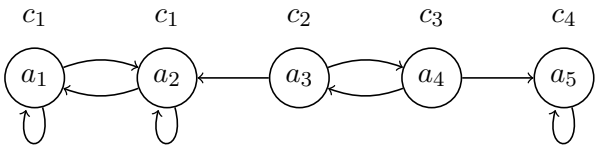

Figure 4.3: A well-formed claim augmented abstract argumentation framework

Moreover, the output consists of facts

$$in(C)$$

stating that the claim C is in the respective extension of the claim augmented abstract argumentation framework specified in the input.

Therefore, the non well-formed claim augmented abstract argumentation framework depicted in Figure 4.2 can be encoded as:

$$\Pi_{nwf\_ex} = \begin{cases} \text{arg}(a,\, \alpha).\ \text{arg}(b,\, \beta).\ \text{arg}(c,\, \alpha). \\ \text{att}(a,\, b).\ \text{att}(b,\, c). \end{cases}$$

Furthermore, the well-formed claim augmented abstract argumentation framework depicted in Figure 4.3 can be encoded as:

$$\Pi_{wf\_ex} =$$
$$\begin{cases} \text{arg}(a_1,\, c_1).\ \text{arg}(a_2,\, c_1).\ \text{arg}(a_3,\, c_2).\ \text{arg}(a_4,\, c_3).\ \text{arg}(a_5,\, c_4). \\ \text{att}(c_1,\, a_1).\ \text{att}(c_1,\, a_2).\ \text{att}(c_2,\, a_2).\ \text{att}(c_2,\, a_4).\ \text{att}(c_3,\, a_3).\ \text{att}(c_3,\, a_5).\ \text{att}(c_4,\, a_5). \end{cases}$$

4.2.2 Encodings for non well-formed claim augmented abstract argumentation framework

All encodings in the subsection will share a common core component, encoding the initial guess on the claims in via the predicates ”in” and ”out” and ensuring that all the guessed claims are witnessed by at least one argument in the witnessing set of arguments encoded by ”argIn”, as well as projecting on the predicate ”in”:

$$\Pi_{nwf\_core} = \begin{cases} \text{in(C) :- arg(\_, C), not out(C).} \\ \text{out(C) :- arg(\_, C), not in(C).} \\ \text{\{ argIn(X) : arg(X, C) \} > 0 :- in(C).} \\ \text{\#show in/1.} \end{cases}$$

Using the core component Π_{nwf_core}, we can define the encoding for conflict-free extensions as

$$\Pi_{nwf\_cf} = \Pi_{nwf\_core} \cup \{\text{:- argIn(X), argIn(Y), att(X, Y).}$$

eliminating all original guesses containing two arguments in "argIn" that are conflicting. As for abstract argumentation frameworks with collective attacks, we can extend the encoding for the conflict-free semantics to compute stable sets as

$$\Pi_{nwf\_st} = \Pi_{nwf\_cf} \cup \begin{cases} \text{defeated(X) :- att(Y, X), argIn(Y).} \\ \text{:- arg(X, \_), not argIn(X), not defeated(X).} \end{cases}$$

where the additional auxiliary predicate "defeated" encodes that the argument X is attacked by some member of the witnessing set. Furthermore, the constraint in the second rule will eliminate all candidates that spare at least one argument that is not defeated, ensuring that the witnessing set is stable.

Moreover, using the encoding for conflict-free extensions, we can also give the encoding for admissible extensions as

$$\Pi_{nwf\_adm} = \Pi_{nwf\_cf} \cup \begin{cases} \text{notDefended(X) :- att(Y, X), not argIn(Z) : att(Z, Y).} \\ \text{:- argIn(X), notDefended(X).} \end{cases}$$

The first rule introduces an auxiliary predicate "notDefended(X)", encoding that the argument X is not defended i.e. there is some argument Y attacking X such that the witnessing set encoded via "argIn" does not contain any argument attacking Y. Finally, the second rule enforces admissibility by eliminating all interpretations which witnessing sets contain an argument X that is not defended.

Furthermore, using the encoding for admissible extensions, we can give the encoding for complete extensions as

$$\Pi_{nwf\_co} = \Pi_{nwf\_adm} \cup \{\text{:- arg(X, \_), not argIn(X), not notDefended(X).}$$

eliminating all candidates that spare at least one argument that is defended, ensure that the witnessing set is complete.

For the remaining three encodings, we will employ the saturation technique once again. Thus, the encoding for preferred sets can be implemented as:

$$\Pi_{nwf\_pr} = \Pi_{nwf\_adm} \cup$$

$$\begin{cases} \text{argOut(X) :- arg(X, \_), not argIn(X).} & (1) \\ \text{non\_trivial :- argOut(X).} & (2) \\ \text{eArgIn(X) : argOut(X) :- non\_trivial.} & (3) \\ \text{eArgIn(X) :- argIn(X), non\_trivial.} & (4) \\ \text{fail :- eArgIn(X), eArgIn(Y), att(X, Y), non\_trivial.} & (5) \\ \text{fail | eArgIn(Z) : att(Z, Y) :- eArgIn(X), att(Y, X), non\_trivial.} & (6) \\ \text{eArgIn(X) :- fail, arg(X, \_), non\_trivial.} & (7) \\ \text{:- not fail, non\_trivial.} & (8) \end{cases}$$

Rules 2 and 3 will encode an extended guess to the original guess of Π_{nwf_adm} using "eArgIn". Rule 5 will ensure that the extended guess is conflict-free, while rule 6 will ensure admissibility.

Moreover, the encoding for the stage semantics can be implemented as:

$$\Pi_{nwf\_stg} = \Pi_{nwf\_cf} \cup$$

$$\begin{cases} \text{range(X) :- argIn(X).} & (1) \\ \text{range(Y) :- argIn(X), att(X, Y).} & (2) \\ \text{notInRange(X) :- arg(X, \_), not range(X).} & (3) \\ \text{non\_trivial :- notInRange(X).} & (4) \\ \text{extendedRange(X) :- range(X), non\_trivial.} & (5) \\ \text{extendedRange(X) : notInRange(X) :- non\_trivial.} & (6) \\ \text{baseSet(X) | baseSet(Y) : att(Y, X) :- extendedRange(X), non\_trivial.} & (7) \\ \text{fail :- baseSet(X), baseSet(Y), att(X, Y), non\_trivial.} & (8) \\ \text{extendedRange(X) :- fail, arg(X, \_), non\_trivial.} & (9) \\ \text{baseSet(X) :- fail, arg(X, \_), non\_trivial.} & (10) \\ \text{:- not fail, non\_trivial.} & (11) \end{cases}$$

Similarly to the encoding for abstract argumentation frameworks with collective attacks, we will guess an extended range and try to find a conflict-free base set witnessing the extended range. Thus, rules 5 and 6 will encode the extended range using the predicate "extendedRange", while rule 7 will try to find a witnessing base set for each argument X in the extended range, either by adding X to the base set or by adding some argument Y to the base set that attacks X. Finally, rule 8 will eliminate all base sets that are not conflict-free.

Similarly as before, the encoding for the stage semantics can be extended to compute semi-stable sets:

$$\Pi_{nwf\_sst} = \Pi_{nwf\_stg} \cup \Pi_{nwf\_adm} \cup \{\text{fail} \mid \text{baseSet(Z)} : \text{att(Z, Y)} \text{ :- baseSet(X), att(Y, X), non\_trivial.}$$

The additional rule will try to defend each argument X in the base set or infer "fail". Therefore, for each argument Y attacking X, some argument Z attacking Y is added to the base set or "fail" is inferred.

4.2.3 Argument-driven encodings for well-formed claim augmented abstract argumentation framework

In this subsection, we will be giving the first set of encodings for well-formed claim augmented abstract argumentation frameworks. These encodings will be focusing on guessing on arguments, while the encodings in the next subsection will make guesses on claims. Once again, all encodings of this subsection will share a core fragment, carrying out the initial guess on arguments encoded using the predicates "inArg" and "outArg", inferring and projecting on the "in" claims:

$$\Pi_{wfa\_core} = \begin{cases} \text{inArg(X) :- not outArg(X), arg(X, \_).} \\ \text{outArg(X) :- not inArg(X), arg(X, \_).} \\ \text{in(C) :- inArg(X), arg(X, C).} \\ \text{\#show in/1.} \end{cases}$$

Furthermore, using the core fragment, we can give the encoding for conflict-free sets as:

$$\Pi_{wfa\_cf} = \Pi_{wfa\_core} \cup \{\text{:- inArg(X), in(C), att(C, X).}$$

eliminating all guesses that include an argument X that is attacked by some claim C which is in the extension of "in", i.e. there is some argument Y in the extension of "inArg" such that $\gamma(Y) = C$. Thus, the encoding for stable sets can be constructed by adding one more constraint:

$$\Pi_{wfa\_st} = \Pi_{wfa\_cf} \cup \{\text{:- outArg(X), not in(C) : att(C, X).}$$

eliminating all guesses that exclude an argument for which all attacking claims are not witnessed by the guess itself.

Moreover, the encoding for the admissible semantics can be implemented as:

$$\Pi_{wfa\_adm} = \Pi_{wfa\_cf} \cup \{\text{:- inArg(Y), att(C, Y), arg(X, C), not in(C1) : att(C1, X).}$$

The additional rule will eliminate all guesses that are not admissible. Thus, all guesses will be eliminated for which there is an argument Y that is in and for which there is an argument X with claim C attacking Y such that all claims C_1 attacking X are not witnessed by the guess.

Contrary to the previous two sets of encodings, the encoding for complete sets in this subsection will not build upon the encoding for admissible sets but will be given as:

$$\Pi_{wfa\_co} = \Pi_{wfa\_cf} \cup$$
$$\begin{cases} \text{not\_defended(Y) :- arg(X, C), att(C, Y), not in(C1) : att(C1, X).} \\ \text{:- inArg(X), not\_defended(X).} \\ \text{:- outArg(X), not not\_defended(X).} \end{cases}$$

We introduce an auxiliary predicate "not_defended(Y)" stating that an argument Y is attacked by some argument X, which itself is not being attacked by the current guess. This predicate will be used to ensure completeness and can be used to give a more concise constraint for admissibility.

For the encoding of preferred sets, we once again make use of the saturation technique:

$$\Pi_{wfa\_pr} = \Pi_{wfa\_adm} \cup$$
$$\begin{cases} \text{non\_trivial :- outArg(\_).} & (1) \\ \text{eArgIn(X) :- inArg(X), non\_trivial.} & (2) \\ \text{eArgIn(X) : outArg(X) :- non\_trivial.} & (3) \\ \text{eIn(C) :- eArgIn(X), arg(X, C), non\_trivial.} & (4) \\ \text{fail :- eIn(C), eArgIn(X), att(C, X), non\_trivial.} & (5) \\ \text{fail | eArgIn(Z) : att(C1, Y), arg(Z, C1) :- eArgIn(X), att(C, X), arg(Y, C), non\_trivial.} & (6) \\ \text{eArgIn(X) :- fail, arg(X, \_), non\_trivial.} & (7) \\ \text{eIn(C) :- fail, arg(\_, C), non\_trivial.} & (8) \\ \text{:- not fail, non\_trivial.} & (9) \end{cases}$$

Rules 2-4 make a guess encoded in "eArgIn" to extend the original guess. Rule 5 ensures conflict-freeness, while rules 6 and 7 ensure admissibility. Thus, for every argument in the extension of "eArgIn(X)" and for every argument Y attacking X, we need some claim C1 in that attacks Y, thus protecting X against Y or infer "fail".

Moreover, the encoding for stage sets can be implemented as:

$$\Pi_{wfa\_stg} = \Pi_{wfa\_cf} \cup$$

```
inRange(X) :- inArg(X).                                                          (1)
inRange(X) :- in(C), att(C, X).                                                  (2)
non_trivial :- arg(X, _), not inRange(X).                                        (3)
eInRange(X) :- inRange(X), non_trivial.                                          (4)
eInRange(X) : arg(X, _), not inRange(X) :- non_trivial.                          (5)
eIn(C) :- arg(X, C), baseSet(X), non_trivial.                                    (6)
baseSet(X) | baseSet(Y) : att(C, X), arg(Y, C) :- eInRange(X), non_trivial.      (7)
fail :- baseSet(X), eIn(C), att(C, X), non_trivial.                              (8)
eInRange(X) :- fail, arg(X, _), non_trivial.                                     (9)
baseSet(X) :- fail, arg(X, _), non_trivial.                                      (10)
:- not fail, non_trivial.                                                        (11)
```

We first compute the range of the original guess via the predicate ”inRange” and try to guess an extended rage via the predicate ”eInRange” in rules 4-6 and try to find a base set witnessing the extended range. Thus, in rule 7, for every argument X in the extended range, we either need the argument to be in the base set or some argument Y attacking X to be in the base set. Moreover, rule 8 derives ”fail” for all base sets that are not conflict-free.

Furthermore, the encoding for semi-stable sets can be implemented as:

$$\Pi_{wfa\_sst} = \Pi_{wfa\_adm} \cup \Pi_{wfa\_stg} \cup$$

```
{fail | baseSet(Z) : att(C1, Y), arg(Z, C1) :- baseSet(X), att(C, X), arg(Y, C), non_trivial.
```

The additional rule will check for admissibility of the base guess, by ensuring that for each argument X in the base set and every argument Y attacking X, either an argument Z attacking Y is contained in the base set or derive ”fail”.

4.2.4 Claim-driven encodings for well-formed claim augmented abstract argumentation framework

Like the previous sets of encodings, also the encodings of this subsection will share a common core component, encoding the initial guess on the claims in via the predicates ”in” and ”out” and project on the predicate ”in”:

$$\Pi_{wfc\_core} = \begin{cases} \text{in(C) :- arg(\_, C), not out(C).} \\ \text{out(C) :- arg(\_, C), not in(C).} \\ \text{\#show in/1.} \end{cases}$$

The encodings of this subsection will make use of the way of computing the unique maximal conflict-free and admissible extension introduced in [DW19]. Thus, the conflict-free encoding can be implemented as:

$$\Pi_{wfc\_cf} = \Pi_{wfc\_core} \cup \begin{cases} \text{inArg(X) :- in(C), arg(X, C), not att(C1, X) : in(C1).} \\ \text{:- in(C), not inArg(X) : arg(X, C).} \end{cases}$$

The first rule will encode the, if it exists, unique maximal conflict-free set of arguments supporting the claims guess via ”in” into ”inArg”. Thus, the second rule will ensure that the selected set of arguments supports the expected claims and eliminates all candidates that do not.

Furthermore, the given encoding for conflict-free sets can be extended to encode the stable semantics by adding one more constraint:

$$\Pi_{wfc\_st} = \Pi_{wfc\_cf} \cup \{\text{:- arg(X, \_), not inArg(X), not in(C) : att(C, X).}$$

The constraint will eliminate all candidate extensions of ”inArg” that do not include some argument X for which all attacking claims are not guessed ”in” the core component.

Moreover, the encoding for admissible sets does not build upon the encoding for conflict-free sets this time, but instead builds upon the core component directly:

$$\Pi_{wfc\_adm} = \Pi_{wfc\_core} \cup \begin{cases} \text{notDefended(X) :- att(C, X), arg(Y, C), out(C1) : att(C1, Y).} \\ \text{inArg(X) :- in(C), arg(X, C), not notDefended(X), out(C1) : att(C1, X).} \\ \text{:- in(C), not inArg(X) : arg(X, C).} \end{cases}$$

The auxiliary predicate ”notDefended(X)” has the intended meaning that the argument X is not defended, i.e. there is some argument Y that attacks X which is not attacked by the guessed claims. Moreover, the second rule, in addition to its analogue in the encoding for conflict-free sets, also takes admissibility into account by excluding all not defended arguments.

Building upon the encoding for admissible sets, one can implement the complete encoding by adding one more constraints, eliminating all candidates that do not include some defended argument:

$$\Pi_{wfc\_co} = \Pi_{wfc\_adm} \cup \{\text{:- arg(X, \_), not notDefended(X), not inArg(X).}$$

Likewise, the encoding for preferred sets can be stated by extending the encoding for admissible sets using the saturation technique:

$$\Pi_{wfc\_pr} = \Pi_{wfc\_adm} \cup$$

```
non_trivial :- arg(X, _), not inArg(X).                                    (1)
claim(C) :- arg(_, C), non_trivial.                                        (2)
eIn(C) | eOut(C) :- claim(C), non_trivial.                                 (3)
eNotInArg(X) :- arg(X, C), eOut(C), non_trivial.                           (4)
eNotInArg(X) :- eIn(C), att(C, X), non_trivial.                            (5)
eNotInArg(X) :- att(C, X), arg(Y, C), non_trivial, eOut(C2) : att(C2, Y).  (6)
fail :- eIn(C), non_trivial, eNotInArg(X) : arg(X, C).                     (7)
fail :- inArg(X), eNotInArg(X), non_trivial.                               (8)
fail :- non_trivial, eNotInArg(X) : arg(X, _), not inArg(X).               (9)
eIn(C) :- fail, arg(_, C), non_trivial.                                    (10)
eOut(C) :- fail, arg(_, C), non_trivial.                                   (11)
eNotInArg(X) :- fail, arg(X, _), non_trivial.                              (12)
:- not fail, non_trivial.                                                  (13)
```

Rules 2 and 3 will guess another set of claims, while rules 4-6 will collect all argument into the extension of ”eNotInArg” that are not part of the maximal admissible set belonging to the guessed set of claims. Finally, rule 7 will derive ”fail” if one ore more of the guessed claims is not witnessed by the remaining, not excluded arguments while rules 8 and 9 will derive ”fail” if the guessed set of claims does not yield a proper superset of the respective set of the original guess of Π_{wfc_adm}.

Moreover, the encoding for the stage semantics, building upon the encoding for conflict-free sets, can be implemented as:

$$\Pi_{wfc\_stg} = \Pi_{wfc\_cf} \cup \Pi_{wfc\_stg_{core}}$$

where

$$\Pi_{wfc\_stg_{core}} =$$

```
range(X) :- inArg(X).                                                    (1)
range(X) :- in(C), att(C, X).                                            (2)
non_trivial :- arg(X, _), not range(X).                                  (3)
partialBaseSet(X) | eIn(C) : att(C, X) :- range(X), non_trivial.         (4)
partialBaseSet(X) : arg(X, _), not range(X) |
      eIn(C) : arg(X, _), not range(X), att(C, X) :- non_trivial.        (5)
eIn(C) :- partialBaseSet(X), arg(X, C), non_trivial.                     (6)
fail :- eNotInArg(X), partialBaseSet(X), non_trivial.                    (7)
eNotInArg(X) :- eIn(C), att(C, X), non_trivial.                          (8)
fail :- eIn(C), non_trivial, eNotInArg(X) : arg(X, C).                   (9)
eIn(C) :- fail, arg(_, C), non_trivial.                                  (10)
eNotInArg(X) :- fail, arg(X, _), non_trivial.                            (11)
partialBaseSet(X) :- fail, arg(X, _), non_trivial.                       (12)
:- not fail, non_trivial.                                                (13)
```

For each argument X in the range of the original guess, rule 4 will either postulate X to be included in the base set of the extended guess via the predicate ”partialBaseSet(X)” or by adding some claim C to the set of claims ”eIn(C)” of the extended guess. Moreover, rule 5 will do the same for one argument not in the range of the original guess, thus producing a proper superset. Note that the encoding will not compute the base set, thus the use of the predicate ”partialBaseSet”, and instead compute the arguments excluded from it, as due to the fact that the underlying claim augmented abstract argumentation framework is well-formed, having on argument not excluded per required claim is sufficient. Nonetheless, for each argument forced into the base set via the predicate ”partialBaseSet”, rule 6 will include its claim into the set of claims of the extended range. Moreover, rule 7 will derive ”fail” if the current candidate excludes some argument from the extended range that has to be included via the argument ”partialBaseSet”, while rule 8 will exclude all arguments are attacked by the guessed claims and thus are not conflict-free. Finally, rule 9 will derive ”fail” if all arguments have been excluded that could support some claim in the extension of ”eIn”.

Finally, the encoding for semi-stable sets can be constructed by replacing Π_{wfc_cf} by Π_{wfc_adm} and adding one more rule:

$$\Pi_{wfc\_sst} = \Pi_{wfc\_adm} \cup \Pi_{wfc\_stg_{core}} \cup$$
$$\{\text{eNotInArg(X) | eIn(C1) : att(C1, Y) :- att(C, X), arg(Y, C), non\_trivial.}$$

The additional rule will ensure, that each argument X attacked by some argument Y is either excluded from the base set or that some claim attacking Y is required for it.

4.2.5 Argument-driven encodings for well-formed claim augmented abstract argumentation framework with claim-centric range

The argument-driven encoding for the stage$_c$ semantics can be implemented as:

$$\Pi_{wfa\_stg_c} = \Pi_{wfa\_cf} \cup$$

```
attacked(X) :- in(C), att(C, X).                                                    (1)
inRange(C) :- in(C).                                                                (2)
inRange(C) :- arg(_, C), attacked(X) : arg(X, C).                                   (3)
non_trivial :- arg(_, C), not inRange(C).                                           (4)
eInRange(C) :- inRange(C), non_trivial.                                             (5)
eInRange(C) : arg(_, C), not inRange(C) :- non_trivial.                             (6)
baseSet(X) : arg(X, C) | eNeedAttacked(C) :- eInRange(C), non_trivial.              (7)
eIn(C) :- arg(X, C), baseSet(X), non_trivial.                                       (8)
fail | eNeedClaim(C1) : att(C1, X) :- eNeedAttacked(C), arg(X, C), non_trivial.     (9)
fail | baseSet(X) : arg(X, C) :- eNeedClaim(C), non_trivial.                        (10)
fail :- baseSet(X), eIn(C), att(C, X), non_trivial.                                 (11)
eInRange(C) :- fail, arg(_, C), non_trivial.                                        (12)
baseSet(X) :- fail, arg(X, _), non_trivial.                                         (13)
eNeedAttacked(C) :- fail, arg(_, C), non_trivial.                                   (14)
eNeedClaim(C) :- fail, arg(_, C), non_trivial.                                      (15)
:- not fail, non_trivial.                                                           (16)
```

In contrast to Π_{wfa_stg}, this encoding defines the range in term of claims instead of arguments. Thus, in rule 5 and 6, we guess an extended range and justify it via rule 7 by, for each claim, either adding some argument of that claim into the base set or by attacking all arguments with that claim via the auxiliary predicate "eNeedAttacked" and further forcing some argument with that claim into the base set using another auxiliary predicate "eNeedClaim".

Moreover, the encoding for semi-stable$_c$ sets can be constructed by starting with the encoding for admissible sets and adding one more rule:

$$\Pi_{wfa\_sst_c} = \Pi_{wfa\_adm} \cup \Pi_{wfa\_stg_c} \cup$$

```
fail | eNeedClaim(C1) : att(C1, Y) :- baseSet(X), att(C, X), arg(Y, C), non_trivial.
```

The additional rule will ensure admissibility by defending each argument X of the base set against all attackers Y by requiring some claim attacking Y to be witnessed by the base set.

4.2.6 Claim-driven encodings for well-formed claim augmented abstract argumentation framework with claim-centric range

The claim-driven encoding for the stage$_c$ semantics can be implemented as:

$$\Pi_{wfc\_stg_c} = \Pi_{wfc\_cf} \cup \Pi_{wfc\_stg_{c_{core}}}$$

where

$$\Pi_{wfc\_stg_{c_{core}}} =$$

```
attacked(X) :- in(C), att(C, X).                                                    (1)
range(C) :- in(C).                                                                  (2)
range(C) :- out(C), attacked(X) : arg(X, C).                                        (3)
non_trivial :- arg(_, C), not range(C).                                             (4)
eIn(C) | eNeedAttacked(C) :- range(C), non_trivial.                                 (5)
eIn(C) : arg(_, C), not range(C) |
    eNeedAttacked(C) : arg(_, C), not range(C) :- non_trivial.                      (6)
eNotInArg(X) :- eIn(C), att(C, X), non_trivial.                                     (7)
fail | eIn(C1) : att(C1, X) :- eNeedAttacked(C), arg(X, C), non_trivial.           (8)
fail :- eIn(C), non_trivial, eNotInArg(X) : arg(X, C).                              (9)
eIn(C) :- fail, arg(_, C), non_trivial.                                             (10)
eNeedAttacked(C) :- fail, arg(_, C), non_trivial.                                   (11)
eNotInArg(X) :- fail, arg(X, _), non_trivial.                                       (12)
:- not fail, non_trivial.                                                           (13)
```

Once again, we guess and extended range in rules 5 and 6, without explicitly building the base set but instead collecting all excluded arguments via ”eNotInArg” and ensuring that not all arguments are excluded for each required claim in rule 9.

Finally, the encoding for semi-stable$_c$ sets can be implemented as:

$$\Pi_{wfc\_sst_c} = \Pi_{wfc\_core} \cup \Pi_{wfc\_stg_{c_{core}}} \cup$$

```
attacked(X) :- in(C), att(C, X).                                                    (1)
notDefended(X) :- att(C, X), arg(Y, C), not attacked(Y).                            (2)
inArg(X) :- in(C), arg(X, C), not notDefended(X), out(C1):att(C1, X).              (3)
:- in(C), not inArg(X) : arg(X, C).                                                 (4)
eNotInArg(X) | eIn(C1) : att(C1, Y) :- att(C, X), arg(Y, C), non_trivial.          (5)
```

Similarly as for the regular claim-driven semi-stable encoding, rules 1-4 differ slightly from the stage semantics, due to the reuse of the predicate ”attacked”. The addition rule 5 finally ensures admissibility by excluding all not defended arguments.

CHAPTER 5

Experiments

In this chapter, we will present the algorithms that were used to modify the base instances to obtain the test cases that were used for our experiments. Moreover, we will present the results of the experiments, enumerating extensions of argumentation frameworks, that have been conducted. The experiments will focus on claim augmented abstract argumentation frameworks, except for the Section 5.5, which will also utilize the encodings for abstract argumentation frameworks with collective attacks.

All experiments have been carried out on a machine equipped with an AMD Opteron™ 6128 and 32GB RAM using clingo[1] 4.5.4. Furthermore, each run has been performed with a 30 minutes time limit and exceeding the time limit is punished with an assumed run time of 60 minutes. Moreover, other errors, such as exceeding memory capabilities, were ignored and do not appear in the figures and tables provided in the following sections. Finally, the parameters for all clingo calls, in addition to the paths of the encoding and instance files, were ”–models=0”, ”–project”, ”–time-limit=1800” and ”–q=2”.

5.1 Instance selection and modification

To conduct the experiments discussed in this chapter, we required some, as close as possible to, real-world test cases to properly test competing implementation of the various semantics. Thus, as a starting point, we used some of the benchmarks instances of the International Competition on Computational Models of Argumentation (ICCMA) 2017, available at: `https://argumentationcompetition.org/2017/results.html`. We have selected 20 test cases for each semantics randomly, namely:

- For the complete semantics:
  admbuster_100000.tgf, afinput_exp_acyclic_indvary2_step2_batch_yyy07.tgf,

[1] https://potassco.org/

BA_120_30_3.tgf, BA_120_40_4.tgf, BA_180_10_2.tgf, BA_180_30_5.tgf, BA_60_70_3.tgf, bw2.pfile-4-02.pddl.2.cnf.tgf, ferry2.pfile-L2-C3-01.pddl.1.cnf.tgf, ferry2.pfile-L2-C4-05.pddl.1.cnf.tgf, irvine-shuttle_20091229_1547.gml.80.tgf, los_angeles_2016-01.gml.80.tgf, scc_1476_70_15_24.tgf, sembuster_60.tgf, stb_340_455.tgf, thecomet_20131025_1906.gml.20.tgf, WS_200_28_90_50.tgf, WS_300_32_30_70.tgf, WS_400_32_50_10.tgf, WS_500_16_30_70.tgf

- For the stable semantics:
  admbuster_20000.tgf, admbuster_500000.tgf, afinput_exp_acyclic_indvary2_step2_batch_yyy10.tgf, BA_120_20_2.tgf, BA_120_40_3.tgf, bw2.pfile-3-06.pddl.5.cnf.tgf, commuteorg-shuttle_20150308_1938.gml.20.tgf, ER_300_100_7.tgf, ER_300_30_9.tgf, ER_400_50_4.tgf, ER_500_100_8.tgf, scc_1213_40_10_3.tgf, scc_840_40_20_24.tgf, sembuster_1500.tgf, sembuster_7500.tgf, stb_522_11.tgf, stb_531_83.tgf, stb_792_333.tgf, WS_400_16_30_50.tgf, WS_400_24_70_10.tgf

- For the preferred semantics:
  admbuster_1000000.apx, admbuster_4000.apx, afinput_exp_acyclic_indvary2_step2_batch_yyy10.apx, afinput_exp_cycles_depvary_step4_batch_yyy03.apx, BA_140_30_4.apx, BA_160_20_3.apx, BA_200_30_2.apx, BA_80_70_5.apx, bw3.pfile-3-01.pddl.1.cnf.apx, ER_200_40_8.apx, ER_300_100_4.apx, ER_400_70_1.apx, ER_500_60_10.apx, scc_1109_50_10_5.apx, sembuster_300.apx, sembuster_3600.apx, translink-archiver_20151219_0124.gml.80.apx, tursib_20110626_1306.gml.20.apx, view2gt_20150927_1744.gml.50.apx, WS_100_12_10_70.apx

- For the semi-stable semantics:
  afinput_exp_acyclic_depvary_step6_batch_yyy04.apx, afinput_exp_acyclic_indvary2_step2_batch_yyy07.apx, afinput_exp_cycles_indvary2_step1_batch_yyy04.apx, BA_120_40_4.apx, BA_180_30_5.apx, bw2.pfile-3-08.pddl.5.cnf.apx, ER_200_30_8.apx, ER_200_70_5.apx, ER_400_60_4.apx, ER_500_60_10.apx, ER_500_80_10.apx, ferry2.pfile-L2-C1-04.pddl.4.cnf.apx, ferry2.pfile-L2-C2-04.pddl.1.cnf.apx, ferry2.pfile-L2-C4-05.pddl.1.cnf.apx, grd_10745_1_4.apx, grd_13651_2_3.apx, grd_1790_4_8.apx, sembuster_6000.apx, WS_500_16_30_70.apx, WS_500_24_30_70.apx

- For the stage semantics:
  admbuster_1000000.apx, BA_120_40_4.apx, BA_160_90_1.apx, BA_60_60_3.apx, el-dorado-transit_20151217_1024.gml.20.apx, ER_100_50_6.apx, ER_400_30_4.apx, ER_500_80_10.apx, ferry2.pfile-L3-C3-010.pddl.1.cnf.apx, grd_12259_4_5.apx, grd_1790_4_8.apx, grd_3018_3_7.apx, scc_7123_70_5_5.apx, scc_8752_70_20_1.apx, stb_593_45.apx, tursib_20110626_1306.gml.20.apx,

view2gt_20150927_1744.gml.50.apx, WS_300_16_70_30.apx,
WS_300_24_50_50.apx, WS_400_32_70_30.apx

As our empirical testing required benchmark instances for claim augmented abstract argumentation frameworks, we needed benchmark instances that represent such frameworks. Unfortunately, we were unable to find real-world benchmarks for this use case and thus decided to use the aforelisted benchmarks as a starting point and modify them in such a way, that they can be used for our testing purposes. Therefore, we implemented two algorithms in Java 11 to modify the selected ICCMA 2017 Dung abstract argumentation frameworks into non well-formed and well-formed instances. We will introduce those two algorithms in the next two subsections.

5.1.1 Generation of non well-formed instances

We generated nine test instances $t_1, \ldots, t_9$ for each benchmark instance from the set of selected ICCMA 2017 instances and use those to test two competing encodings against each other. The Algorithm 5.1 randomly distributes $\frac{i}{10} \times k$ claims over the arguments of instance t_i, where k is the number of arguments of the original benchmark instance. Therefore, the Algorithm 5.1 initializes the list *workingClaims* with the claims $c_1, \ldots, c_{\lfloor |A| \times f \rfloor}$, where $|A|$ is the number of arguments and f is the factor of claims to distribute in relation to the number of argument. Next, the Algorithm 5.1 randomly select a claim c from this list and removes it. Finally, c is assigned to some argument and the list is re-initialized whenever it is empty. Thus, the Algorithm 5.1 expects a set of arguments A as input over which the claims are supposed to be distributed, as well as a factor f between 0.1 and 0.9 representing the number of claims to distribute.

Algorithm 5.1: GeneratedNonWellformedInstance

Input: A: set of arguments,
f: float factor between 0.1 and 0.9 representing the number of claims to distribute
Result: Augments the set of arguments A with claims

```
1  C ← c_1, ..., c_⌊|A|×f⌋ ;
2  workingClaims ← copy(C) ;
3  foreach argument a in A do
4      if workingClaims is empty then
5          workingClaims ← copy(C);
6      end
7      i ← randomBetweenExclusiveUpper(0, |workingClaims|) ;
8      c ← workingClaims.getAndRemove(i) ;
9      assign claim c to argument a ;
10 end
```

5.1.2 Generation of well-formed instances

Similarly, as the Algorithm 5.1 for non well-formed test instances, the Algorithm 5.2 for well-formed instances generates nine test instances per base benchmark, varying in the amount of claims distributed, depending on the number of arguments of the base benchmark. However, as those instances have to be well-formed, we modify the attacks of the base benchmark to ensure well-formedness. Furthermore, in an effort to preserve the original structure of the instance as much as possible, we try to keep the number of attacks of the generated instance as close as possible to the amount the original one. Therefore, attacks are either removed or added, depending on the current delta of attacks between the original benchmark and the generated test instance.

Thus, the Algorithm 5.2 requires the same arguments as the Algorithm 5.1 with an additional set R, representing the attacks of the original framework. It starts by invoking the Algorithm 5.1, randomly distributing claims over the arguments. Thus, as the resulting framework will not be well-formed in general, the remaining parts of the Algorithm 5.2 will construct a new set of attacks to ensure well-formedness. As we will try to preserve the original structure of the instance as much as possible, the algorithm uses the variable *counter* to keep track of the number of attacks added or removed in comparison to the original framework. Therefore, for each claim c, the Algorithm 5.2 collects all arguments a attacked by any of the arguments with that claim and decides, based on the *counter* variable, whether or not the claim c will attack the argument a or not. If the attack is added, the *counter* is increased by number of arguments with claim c that were not attacking the argument a and if not, then the *counter* is decreased by the number of arguments with claim c that did attack the argument a. Finally, this process is repeated for all claims and the new set of attacks from claims on arguments R' is returned.

5.2 Non well-formed claim augmented abstract argumentation framework

For the first set of tests, we have used the encodings presented in Subsection 4.2.2, labeled "adapted", and the ones available at [2] for Dung abstract argumentation frameworks with an additional projection on claims, labeled "naive", depicting the results in Table 5.1.

For the "naive" encodings, we renamed the already existing predicate "in" of the encodings into "inArg" and then added the following three rules to ensure compatibility with the instances for claim augmented abstract argumentation frameworks and return claims instead of arguments:

```
arg(X)  :- arg(X, _).
in(C)   :- inArg(X), arg(X, C).
#show in/1.
```

[2] https://www.dbai.tuwien.ac.at/proj/argumentation/systempage/dung.html

Algorithm 5.2: GeneratedWellformedInstance

Input: A: set of arguments,
R: set of attacks between arguments and arguments,
f: float factor between 0.1 and 0.9 representing the number of claims to distribute
Result: Augments the set of arguments A with claims and returns R', a new set of attacks between claims and arguments

```
1  GeneratedNonWellformedInstance(A, f) ;        /* Distribute claims */
2  C ← claims(A) ;
3  counter ← 0 ;
4  doAdd ← randomBoolean() ;
5  foreach claim c in C in random order do
6      arguments ← arguments of A with claim c ;
7      attacked ← {b | ∃a ∈ arguments such that (a, b) ∈ R} ;
8      foreach argument a in attacked do
9          if doAdd then
10             add (c, a) to R' ;
11             counter ← counter + |{b | b ∈ arguments such that (b, a) ∉ R}| ;
12         else
13             counter ← counter − |{b | b ∈ arguments such that (b, a) ∈ R}| ;
14         end
15         if counter > 0 then
16             doAdd ← false ;
17         else if counter < 0 then
18             doAdd ← true ;
19         else
20             doAdd ← randomBoolean() ;
21         end
22     end
23 end
```

		STG	SST	PR	CO	ST
# Instances		180	180	180	180	180
Solved	adapted	101	135	107	148	118
	naive	45	78	93	159	128
Exclusively solved	adapted	56	57	23	0	0
	naive	0	0	9	11	10
Count timeout	adapted	52	18	46	32	62
	naive	63	29	51	21	52
Avg time solved (s)	adapted	1355,27	625,49	1143,48	666,28	1253,76
	naive	2207,12	1325,22	1506,56	431,18	1085,21
Median time solved (s)	adapted	93,52	45,39	67,24	1,75	6,27
	naive	3600,00	542,89	506,95	1,41	6,29
Avg ratio solve time		0,28	0,11	0,71	11,46	1,42
Median ratio solve time		0,03	0,02	0,28	1,26	1,00

Table 5.1: Experiment results for non well-formed claim augmented abstract argumentation frameworks with a timeout of 1800s and additional 1800s penalty

The line "# Instances" give the number of instances tested for each semantics. Thus, as we have used 20 base instances and created 9 variants for each of those, as described in Section 5.1, this gives 180 for all semantics. The line labeled "Solved" gives the number of instances that were solved within the time-limit for both encodings and for each semantics. Analogously, the line "Exclusively solved" gives the number of instances that were solved by the respective encoding and not by the other one. Furthermore, the line "Avg time solved (s)" and "Median time solved (s)" give the average and median solving times in seconds for both encodings and each semantics. Finally, the lines "Avg ratio solve time" and "Median ratio solve time" give the average and median ratio between the solving time of the "adapted" encoding and the "naive" encoding.

Overall, the adapted approach outperformed the naive approach for the stage, semi-stable and preferred semantics, while the naive approach performed slight better for the complete and stable semantics. In the following five subsections, we will give more detailed results for the stable, complete, preferred, stage and semi-stable semantics. The tables in the subsections of this Section will start with a column "% c", indicating the percentage of claims distributed in relation to the amount of arguments. Furthermore, the column "#" indicates the number of instances for the given percentage of claims. Moreover, the columns "solved", "ex." and "t.o." give the number of solved and exclusively solved instances and the number of instances that hit the time-out limit for a given percentage of claims and a given approach. Next, the columns "avg t (s)" and "med t (s)" give the average and median solving time in seconds for a given percentage and approach. Finally, the column "ratio" gives the average and median ratios between the solving time of the two approaches for a given percentage of claims.

5.2.1 Stable semantics

The results for the stable semantics are depicted in Table 5.2 and Figure 5.1:

% c	#	solved		ex.		t.o.		avg t (s)		med t (s)		ratio	
		a	n	a	n	a	n	a	n	a	n	avg	med
10%	20	14	15	0	1	6	5	1093,08	935,02	4,61	4,74	1,36	1,00
20%	20	13	15	0	2	7	5	1272,84	1022,38	11,59	11,60	1,38	1,00
30%	20	13	14	0	1	7	6	1272,94	1114,98	11,60	11,60	1,39	1,00
40%	20	13	14	0	1	7	6	1274,70	1116,63	10,66	10,70	1,41	1,00
50%	20	13	14	0	1	7	6	1273,95	1115,67	11,62	11,65	1,37	1,00
60%	20	13	14	0	1	7	6	1273,96	1114,86	11,59	11,67	1,46	1,00
70%	20	13	14	0	1	7	6	1273,96	1115,81	11,55	11,65	1,42	1,00
80%	20	13	14	0	1	7	6	1274,17	1116,21	12,56	12,64	1,48	1,00
90%	20	13	14	0	1	7	6	1274,24	1115,33	12,48	12,53	1,47	1,00

Table 5.2: Experiment results for non well-formed stable extensions with a timeout of 1800s and additional 1800s penalty

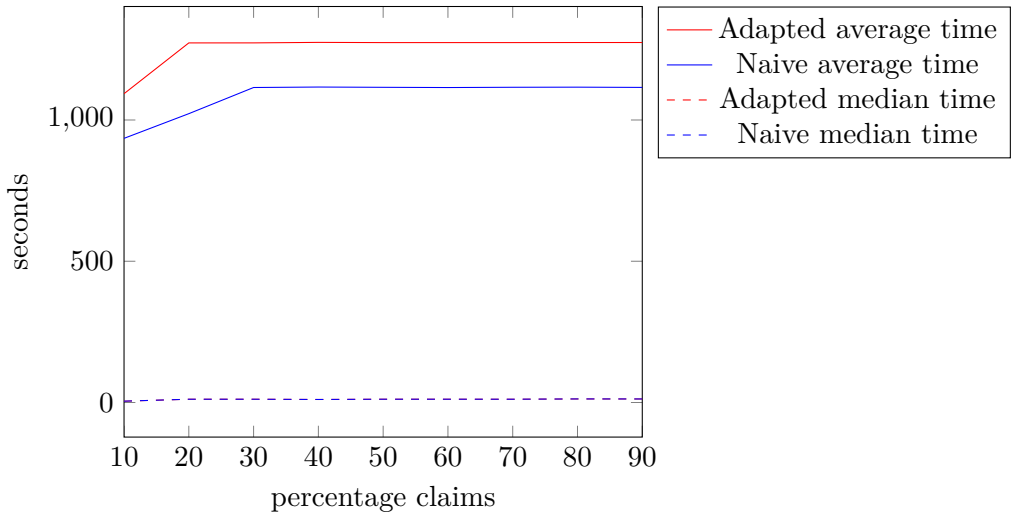

Figure 5.1: Experiment results for non well-formed stable extensions with a timeout of 1800s and additional 1800s penalty

The naive approach was superior for the stable semantics, while the was no clear scaling observable in regard to the percentage of claims, except for a slightly better performance for the instances with 10%.

5.2.2 Complete semantics

The results for the complete semantics are depicted in Table 5.3 and Figure 5.2:

% c	#	solved		ex.		t.o.		avg t (s)		med t (s)		ratio	
		a	n	a	n	a	n	a	n	a	n	avg	med
10%	20	17	18	0	1	3	2	572,95	373,00	0,14	0,12	4,08	1,07
20%	20	17	17	0	0	3	3	573,88	545,05	0,15	0,14	3,00	1,00
30%	20	17	18	0	1	3	2	590,58	392,62	0,29	0,28	12,43	1,21
40%	20	16	18	0	2	4	2	733,24	368,09	0,77	0,64	15,43	1,22
50%	20	16	18	0	2	4	2	734,93	370,12	1,14	1,10	14,38	1,30
60%	20	16	17	0	1	4	3	729,07	545,74	1,95	1,88	11,36	1,27
70%	20	16	17	0	1	4	3	733,31	545,73	5,34	4,10	12,82	1,29
80%	20	17	18	0	1	3	2	593,42	369,37	5,78	4,35	13,76	1,33
90%	20	16	18	0	2	4	2	735,15	370,93	10,12	7,64	15,87	1,35

Table 5.3: Experiment results for non well-formed complete extensions with a timeout of 1800s and additional 1800s penalty

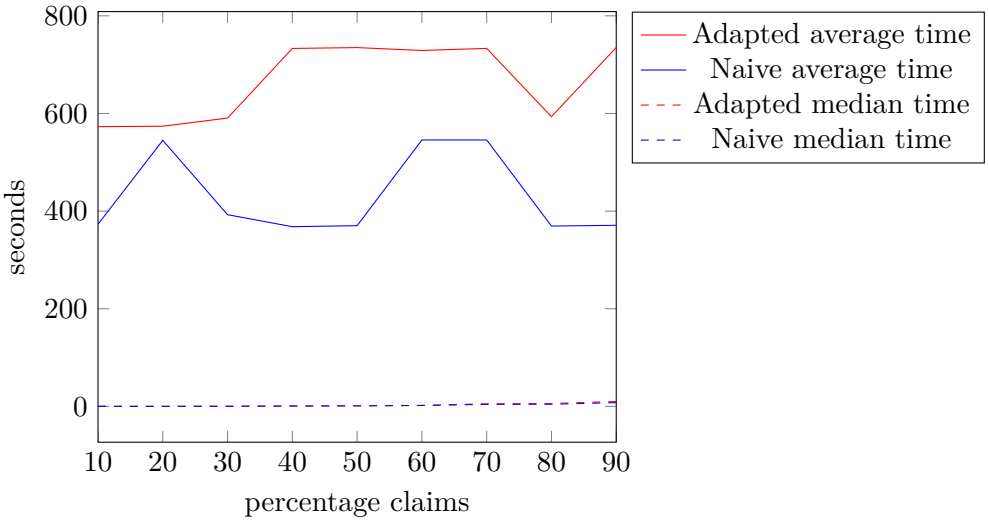

Figure 5.2: Experiment results for non well-formed complete extensions with a timeout of 1800s and additional 1800s penalty

As before, the naive approach was superior for the complete semantics. Moreover, both approaches did not scale with the percentage of claims and even showed different behavior.

5.2.3 Preferred semantics

The results for the preferred semantics are depicted in Table 5.4 and Figure 5.3:

% c	#	solved		ex.		t.o.		avg t (s)		med t (s)		ratio	
		a	n	a	n	a	n	a	n	a	n	avg	med
10%	20	12	12	1	1	5	4	1092,87	1046,80	55,13	246,78	0,70	0,28
20%	20	12	10	3	1	5	6	1201,23	1491,18	67,24	374,78	0,73	0,34
30%	20	11	10	2	1	6	6	1343,24	1537,00	67,85	472,75	0,77	0,79
40%	20	12	10	3	1	5	6	1104,03	1556,21	67,38	593,22	0,69	0,27
50%	20	12	10	3	1	5	6	1107,59	1576,26	66,71	689,92	0,69	0,28
60%	20	12	11	2	1	5	5	1110,02	1475,07	67,03	825,44	0,70	0,30
70%	20	12	11	2	1	5	5	1110,37	1491,82	67,43	794,35	0,70	0,29
80%	20	12	10	3	1	5	6	1110,75	1626,64	67,82	903,08	0,70	0,28
90%	20	12	9	4	1	5	7	1111,26	1758,08	66,79	962,72	0,69	0,27

Table 5.4: Experiment results for non well-formed preferred extensions with a timeout of 1800s and additional 1800s penalty

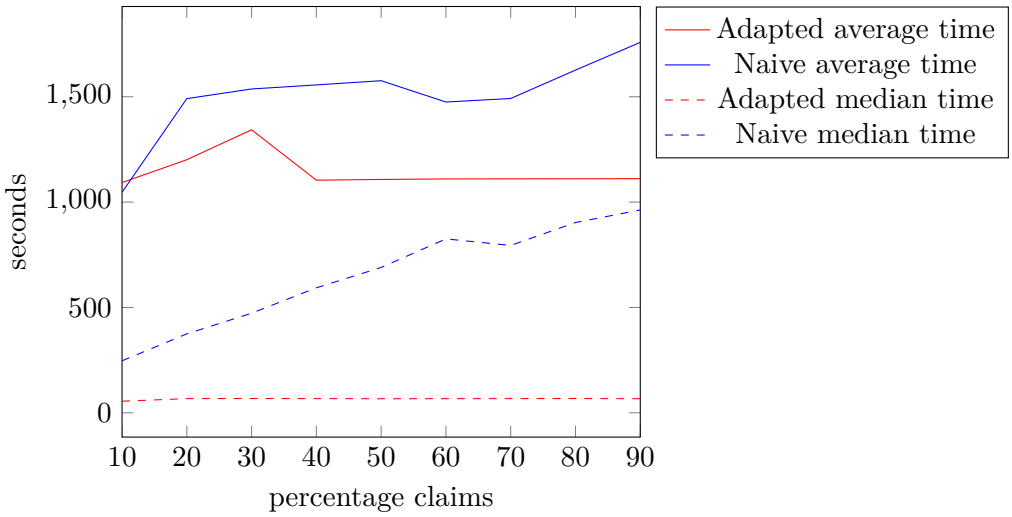

Figure 5.3: Experiment results for non well-formed preferred extensions with a timeout of 1800s and additional 1800s penalty

For the preferred semantics, the adapted approach outperformed the naive approach and the naive approach seemed to scale with the percentage of claims, while the adapted approach did not.

5.2.4 Stage semantics

The results for the stage semantics are depicted in Table 5.5 and Figure 5.4:

% c	#	solved		ex.		t.o.		avg t (s)		med t (s)		ratio	
		a	n	a	n	a	n	a	n	a	n	avg	med
10%	20	13	8	5	0	4	4	949,79	1261,23	20,75	29,59	0,17	0,02
20%	20	12	6	6	0	5	6	1174,02	1805,69	73,99	1822,56	0,14	0,03
30%	20	11	6	5	0	6	6	1315,22	1902,00	38,16	2135,84	0,27	0,02
40%	20	11	5	6	0	6	7	1399,14	2178,32	66,94	3600,00	0,30	0,02
50%	20	11	5	6	0	6	7	1453,80	2238,54	71,15	3600,00	0,30	0,02
60%	20	11	5	6	0	6	7	1471,83	2348,23	236,62	3600,00	0,30	0,03
70%	20	10	3	7	0	7	9	1524,97	2741,17	129,29	3600,00	0,35	0,04
80%	20	11	3	8	0	6	9	1477,95	2739,27	251,59	3600,00	0,30	0,03
90%	20	11	4	7	0	6	8	1430,75	2649,65	211,98	3600,00	0,34	0,03

Table 5.5: Experiment results for non well-formed stage with a timeout of 1800s and additional 1800s penalty

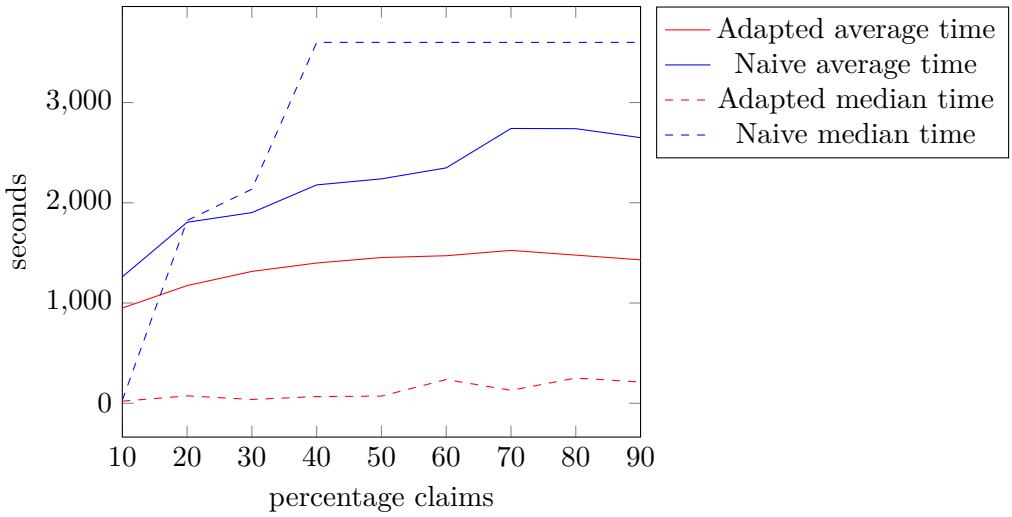

Figure 5.4: Experiment results for non well-formed stage extensions with a timeout of 1800s and additional 1800s penalty

As for the preferred semantics, the adapted approach clearly outperformed the naive approach for the stage semantics. Especially the naive approach seemed to hit the time-limit more often when increasing the percentage of claims.

5.2.5 Semi-stable semantics

The results for the semi-stable semantics are depicted in Table 5.6 and Figure 5.5:

% c	#	solved		ex.		t.o.		avg t (s)		med t (s)		ratio	
		a	n	a	n	a	n	a	n	a	n	avg	med
10%	20	15	10	5	0	2	2	616,86	878,48	17,51	154,87	0,11	0,02
20%	20	15	10	5	0	2	2	625,96	904,73	17,69	179,02	0,12	0,02
30%	20	15	9	6	0	2	3	623,15	1263,30	19,35	400,80	0,12	0,02
40%	20	15	9	6	0	2	3	626,08	1295,41	33,50	775,83	0,12	0,01
50%	20	15	8	7	0	2	3	621,11	1330,97	50,93	542,89	0,04	0,02
60%	20	15	8	7	0	2	4	616,21	1539,73	51,73	883,25	0,12	0,02
70%	20	15	8	7	0	2	4	647,28	1516,16	51,37	787,32	0,12	0,02
80%	20	15	8	7	0	2	4	625,81	1610,01	45,39	1097,95	0,11	0,01
90%	20	15	8	7	0	2	4	626,92	1588,72	48,96	1078,29	0,11	0,01

Table 5.6: Experiment results for non well-formed semi-stable extensions with a timeout of 1800s and additional 1800s penalty

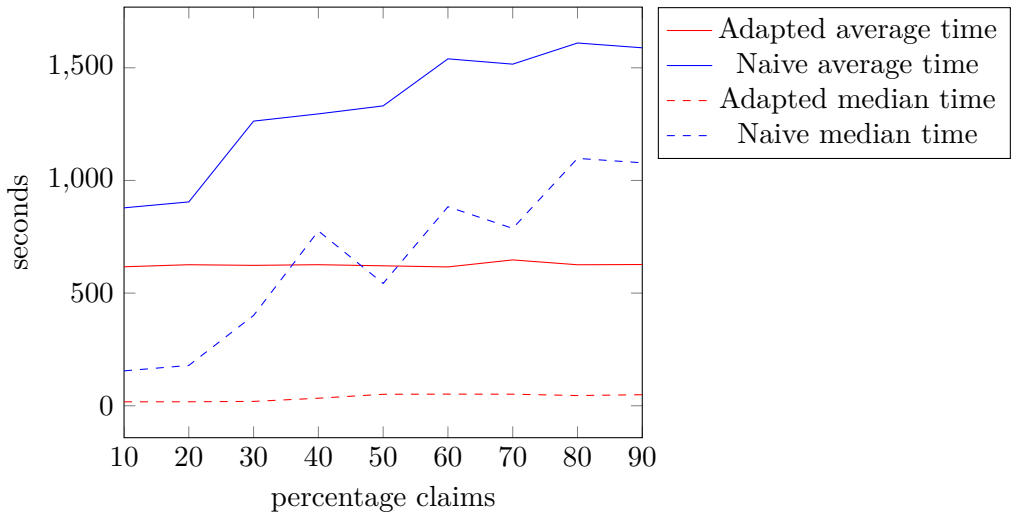

Figure 5.5: Experiment results for non well-formed semi-stable extensions with a timeout of 1800s and additional 1800s penalty

For the semi-stable semantics, the adapted approach outperformed the naive approach. Moreover, while the adapted approach produced very stable results over all test-instances, the performance of the naive approach seemed to degrade when increasing the percentage of claims.

5.2.6 Discussion

The naive encodings performed better for the stable and complete semantics, while the adapted encodings where superior for the preferred, semi-stable and stage semantics. Interestingly, those are the ones that require an additional maximization. Moreover, especially the curves for the complete semantics are quite erratic and do not indicate any correlation with the percentage of claims distributed over the arguments.

5.3 Well-formed claim augmented abstract argumentation framework

In this section, we will present the results of the tests for well-formed claim augmented abstract argumentation frameworks. For these tests, the competing encodings used were the claim-driven encodings presented in Subsection 4.2.4 and the argument-driven encodings presented in Subsection 4.2.3.

As before, we will give an overview over all semantics in Table 5.7 and present the detailed results for the individual semantics in the following subsection, finishing with a discussion of the results. Furthermore, the tables in the subsections of this Section will share the same structure as those in the subsections of the Section 5.2.

		STG	**SST**	**PR**	**CO**	**ST**
# Instances		180	180	180	180	180
Solved	claim-driven	80	101	160	162	151
	argument-driven	101	141	141	164	170
Exclusively solved	claim-driven	0	6	19	0	1
	argument-driven	21	46	0	2	20
Count timeout	claim-driven	88	33	11	18	10
	argument-driven	61	19	16	16	10
Avg time solved (s)	claim-driven	1910,69	941,43	277,60	365,26	288,03
	argument-driven	1408,26	670,53	485,44	333,30	248,02
Median time solved (s)	claim-driven	3600,00	8,78	1,89	0,04	2,37
	argument-driven	6,59	15,67	6,36	0,03	1,27
Avg ratio solve time		89,18	4,33	0,43	1,13	9,19
Median ratio solve time		2,50	2,40	0,30	1,00	3,00

Table 5.7: Experiment results for well-formed claim augmented abstract argumentation frameworks with a timeout of 1800s and additional 1800s penalty

The structure of the Table 5.7 is the same as for the Table 5.1. While the argument-driven approach was superior for the stage semantics, the claim-driven approach performed better for the preferred semantics. Furthermore, the

5.3.1 Stable semantics

The results for the stable semantics are depicted in Table 5.8 and Figure 5.6:

% c	#	solved		ex.		t.o.		avg t (s)		med t (s)		ratio	
		c	a	c	a	c	a	c	a	c	a	avg	med
10%	20	19	20	0	1	0	0	43,65	9,03	0,31	0,10	34,67	3,00
20%	20	18	19	1	2	0	1	47,39	184,53	0,61	0,32	4,78	2,51
30%	20	17	19	0	2	1	1	218,65	186,91	1,24	0,94	6,09	3,00
40%	20	17	19	0	2	1	1	228,84	190,78	1,71	1,04	6,91	3,00
50%	20	17	19	0	2	1	1	247,93	199,69	4,32	2,65	6,59	3,58
60%	20	17	19	0	2	1	1	263,08	273,13	4,60	3,11	6,89	3,67
70%	20	17	19	0	2	1	1	297,49	223,61	4,80	2,47	7,22	4,20
80%	20	15	19	0	4	2	1	469,78	299,84	8,11	7,06	4,17	2,12
90%	20	14	17	0	3	3	3	829,15	664,69	25,90	15,72	3,42	1,42

Table 5.8: Experiment results for well-formed stable extensions with a timeout of 1800s and additional 1800s penalty

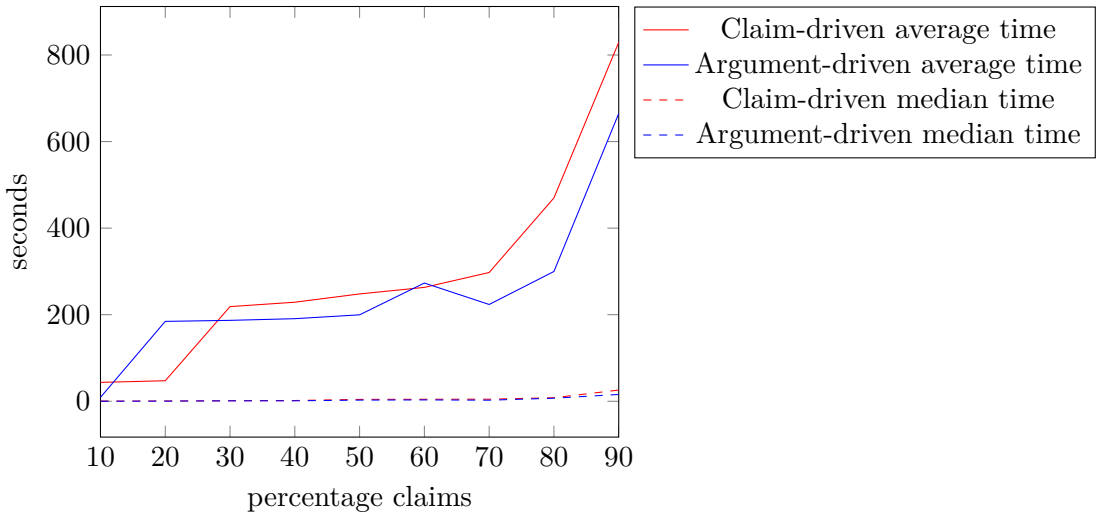

Figure 5.6: Experiment results for well-formed stable extensions with a timeout of 1800s and additional 1800s penalty

While the claim-driven approach was superior for instances with a low number of claims, the argument-driven approach performed better for instances with a larger number of claims. Both encodings seemed to scale with the percentage of claims.

5.3.2 Complete semantics

The results for the complete semantics are depicted in Table 5.9 and Figure 5.7:

% c	#	solved		ex.		t.o.		avg t (s)		med t (s)		ratio	
		c	a	c	a	c	a	c	a	c	a	avg	med
10%	20	19	19	0	0	1	1	180,58	180,60	0,02	0,02	1,06	1,00
20%	20	19	19	0	0	1	1	180,82	180,86	0,03	0,03	1,05	1,00
30%	20	19	19	0	0	1	1	189,39	191,85	0,03	0,03	1,15	1,00
40%	20	18	18	0	0	2	2	362,10	361,94	0,05	0,04	1,07	1,00
50%	20	17	17	0	0	3	3	542,15	542,19	0,08	0,06	1,02	1,00
60%	20	17	18	0	1	3	2	544,67	406,20	0,10	0,10	1,28	1,00
70%	20	17	18	0	1	3	2	543,12	392,76	0,11	0,10	1,30	1,00
80%	20	18	18	0	0	2	2	375,87	376,32	0,13	0,12	1,07	1,00
90%	20	18	18	0	0	2	2	368,65	366,98	0,25	0,26	1,14	1,00

Table 5.9: Experiment results for well-formed complete extensions with a timeout of 1800s and additional 1800s penalty

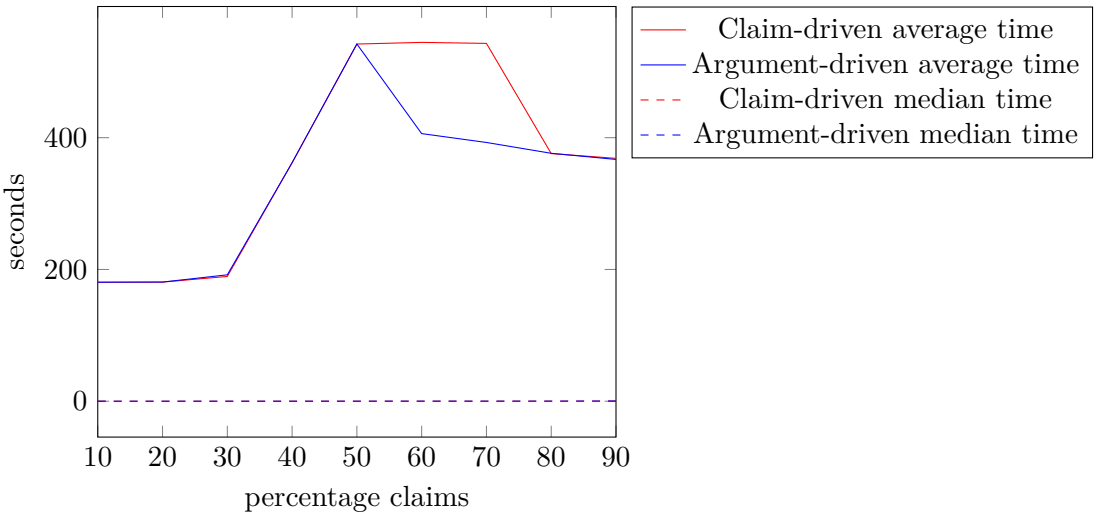

Figure 5.7: Experiment results for well-formed complete extensions with a timeout of 1800s and additional 1800s penalty

Both approaches performed very similar, except for the 60% and 70% instances, for which the argument-driven approach performed better. While both approaches scaled with the number of claims for the lower percentage instances, they both peaked at 50% and decreased afterwards.

5.3.3 Preferred semantics

The results for the preferred semantics are depicted in Table 5.10 and Figure 5.8:

% c	#	solved		ex.		t.o.		avg t (s)		med t (s)		ratio	
		c	a	c	a	c	a	c	a	c	a	avg	med
10%	20	19	19	0	0	0	0	1,81	127,65	0,32	2,65	0,25	0,12
20%	20	19	17	2	0	0	0	4,38	103,30	0,58	0,69	0,30	0,19
30%	20	18	17	1	0	1	1	199,87	403,74	0,78	3,90	0,33	0,21
40%	20	18	16	2	0	1	2	215,30	468,16	0,93	5,13	0,37	0,25
50%	20	18	16	2	0	1	1	221,65	305,73	1,89	6,31	0,44	0,31
60%	20	18	15	3	0	1	2	256,68	485,42	2,92	10,35	0,41	0,30
70%	20	17	14	3	0	2	3	457,94	727,35	5,74	16,87	0,43	0,34
80%	20	17	14	3	0	2	3	459,81	802,95	6,77	62,91	0,43	0,36
90%	20	16	13	3	0	3	4	680,95	992,57	7,30	72,07	0,91	0,45

Table 5.10: Experiment results for well-formed preferred extensions with a timeout of 1800s and additional 1800s penalty

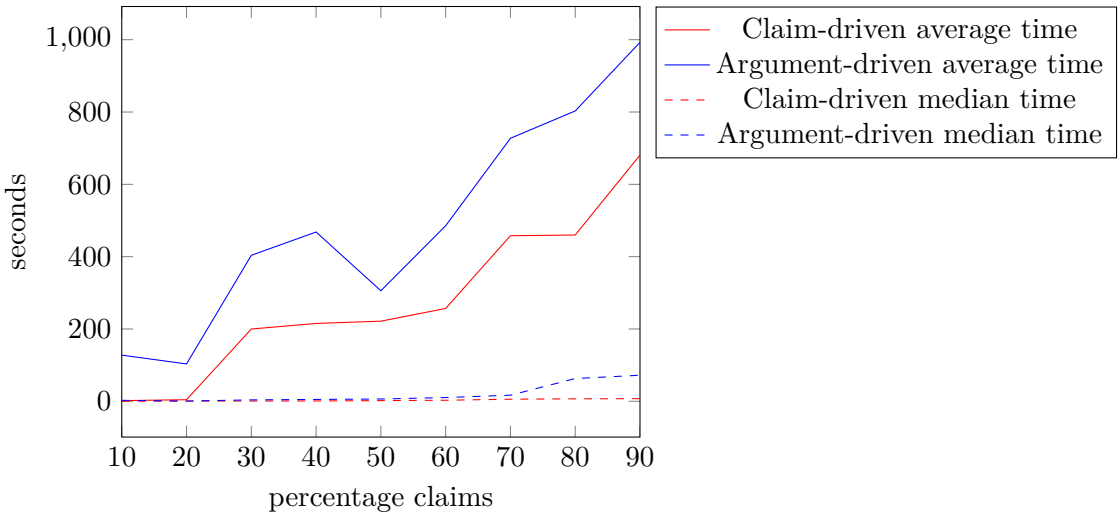

Figure 5.8: Experiment results for well-formed preferred extensions with a timeout of 1800s and additional 1800s penalty

For the preferred semantics, the claim-driven approach outperformed the argument-driven approach. Both encodings scaled with the percentage of claims, with the argument-driven approach experiencing a dip down for the 50% instances.

5.3.4 Stage semantics

The results for the stage semantics are depicted in Table 5.11 and Figure 5.9:

% c	#	solved		ex.		t.o.		avg t (s)		med t (s)		ratio	
		c	a	c	a	c	a	c	a	c	a	avg	med
10%	20	13	14	0	1	6	4	1159,96	801,03	1,36	0,16	25,61	2,24
20%	20	12	14	0	2	7	4	1331,46	822,83	7,58	0,17	59,01	3,88
30%	20	9	11	0	2	10	7	1911,28	1452,88	3600,00	3,85	49,63	2,36
40%	20	9	11	0	2	10	7	1898,41	1402,56	3600,00	10,43	58,98	3,25
50%	20	8	9	0	1	11	9	2084,65	1800,38	3600,00	1803,13	38,17	2,25
60%	20	8	9	0	1	11	9	2099,89	1801,24	3600,00	1808,35	38,81	1,75
70%	20	8	13	0	5	11	5	2113,30	1101,17	3600,00	12,61	426,33	5,75
80%	20	7	10	0	3	11	8	2272,94	1740,60	3600,00	1250,52	43,74	1,67
90%	20	6	10	0	4	11	8	2394,30	1751,63	3600,00	1296,35	60,75	1,22

Table 5.11: Experiment results for well-formed stage extensions with a timeout of 1800s and additional 1800s penalty

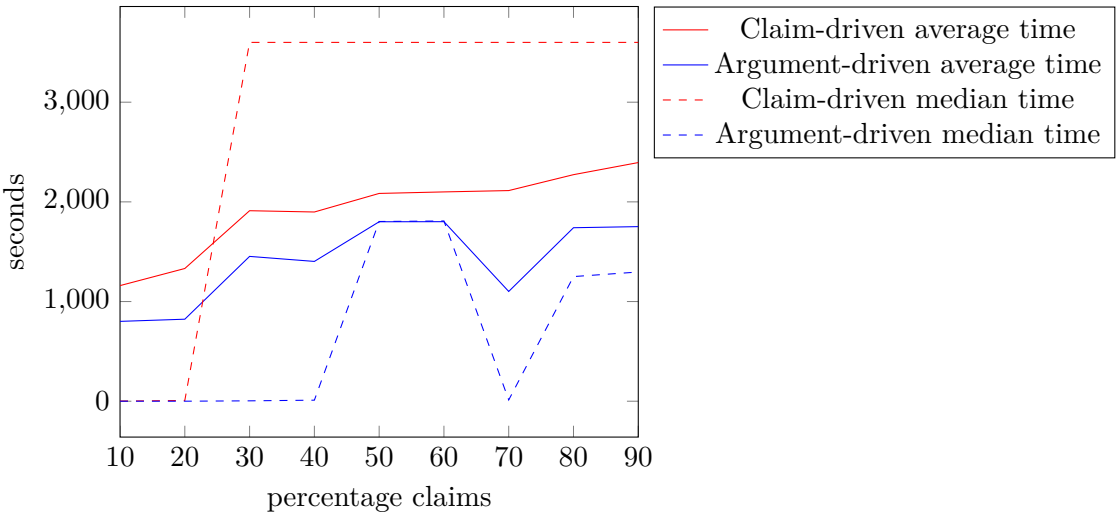

Figure 5.9: Experiment results for well-formed stage extensions with a timeout of 1800s and additional 1800s penalty

The argument-driven approach was superior for the stage semantics, with the claim-driven encoding predominantly hitting the time-limit except for the instance with a low percentage of claims.

5.3.5 Semi-stable semantics

The results for the semi-stable semantics are depicted in Table 5.12 and Figure 5.10:

% c	#	solved		ex.		t.o.		avg t (s)		med t (s)		ratio	
		c	a	c	a	c	a	c	a	c	a	avg	med
10%	20	16	19	0	3	2	0	526,49	245,18	2,45	7,20	3,13	1,35
20%	20	13	17	1	5	3	1	788,54	418,05	1,60	5,25	3,75	1,74
30%	20	11	16	0	5	4	1	983,52	446,16	1,69	2,42	2,86	1,88
40%	20	11	16	0	5	4	2	976,93	565,94	11,48	11,77	4,19	2,50
50%	20	10	16	0	6	5	1	1221,91	517,43	63,79	10,78	4,48	3,67
60%	20	10	14	0	4	4	3	1048,05	825,25	18,44	11,15	4,56	2,92
70%	20	10	15	1	6	4	3	1083,87	931,22	162,84	138,93	5,74	2,94
80%	20	10	14	1	5	4	4	1065,04	986,50	70,61	124,45	5,18	3,47
90%	20	10	14	3	7	3	4	889,57	1110,27	15,69	290,08	5,65	3,74

Table 5.12: Experiment results for non well-formed semi-stable extensions with a timeout of 1800s and additional 1800s penalty

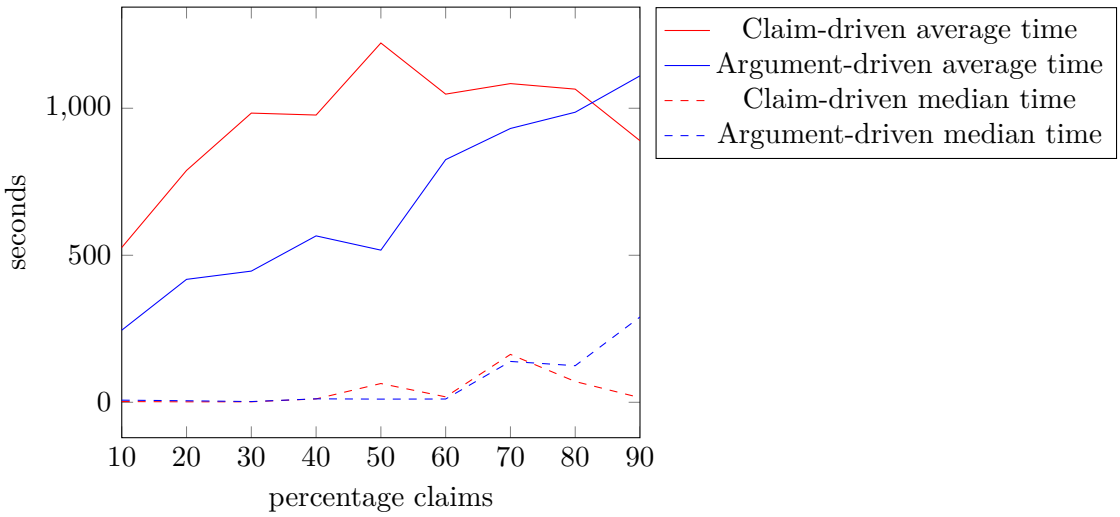

Figure 5.10: Experiment results for well-formed semi-stable extensions with a timeout of 1800s and additional 1800s penalty

The argument-driven approach performed better up until the 80% instances, for which on the claim-driven encoding outperformed the argument-driven approach. While the claim-driven approach did not scale with the number of claims, the argument-driven approach did.

5.3.6 Discussion

Interestingly, the claim- and argument-driven encodings performed very similarly for the stable and complete semantics. Furthermore, while the claim-driven encoding performed slightly better for the preferred semantics, the argument-driven encodings outperformed the claim-driven encodings for the range-based semantics, i.e. the stage and semi-stable semantics.

5.4 Well-formed claim augmented abstract argumentation framework with claim-centric ranges

In this section, we will present the results of the tests for the semi-stable$_c$ and stage$_c$ semantics for well-formed claim augmented abstract argumentation frameworks. For these tests, the competing encodings used were the claim-driven encodings presented in Subsection 4.2.6 and the argument-driven encodings presented in Subsection 4.2.5. As before, we will give an overview over all semantics in Table 5.13 and present the detailed results for the individual semantics in the following subsection, finishing with a discussion of the results. Furthermore, the tables in the subsections of this Section will share the same structure as those in the subsections of the Section 5.2.

		STG$_c$	**SST$_c$**
# Instances		180	180
Solved	claim-driven	89	109
	argument-driven	106	166
Exclusively solved	claim-driven	0	0
	argument-driven	17	57
Count timeout	claim-driven	79	26
	argument-driven	61	6
Avg time solved (s)	claim-driven	1741,81	794,52
	argument-driven	1377,91	287,54
Median time solved (s)	claim-driven	623,49	12,23
	argument-driven	9,31	6,98
Avg ratio solve time		26,99	11,58
Median ratio solve time		1,20	3,67

Table 5.13: Experiment results for stage$_c$ and semi-stable$_c$ for well-formed claim augmented abstract argumentation frameworks with a timeout of 1800s and additional 1800s penalty

The structure of the Table 5.13 is the same as for the Table 5.1. Overall, the argument-driven approach outperformed the claim-driven approach for both semantics.

5.4.1 Stage$_c$ semantics

The results for the stage$_c$ semantics are depicted in Table 5.14 and Figure 5.11:

% c	#	solved		ex.		t.o.		avg t (s)		med t (s)		ratio	
		c	a	c	a	c	a	c	a	c	a	avg	med
10%	20	13	14	0	1	6	5	1136,92	949,75	0,08	0,05	5,29	1,00
20%	20	10	13	0	3	9	6	1705,31	1187,74	0,39	0,16	6,27	1,20
30%	20	9	11	0	2	10	8	1900,40	1579,11	3600,00	285,85	3,13	1,00
40%	20	10	11	0	1	9	8	1733,61	1542,46	484,72	6,73	9,15	1,50
50%	20	11	13	0	2	8	6	1602,83	1182,47	439,36	9,25	43,18	4,04
60%	20	9	11	0	2	10	8	1949,16	1576,85	3600,00	40,29	106,95	1,41
70%	20	10	12	0	2	9	7	1799,70	1466,86	1014,60	65,78	12,40	2,25
80%	20	10	11	0	1	8	6	1702,35	1296,92	806,62	11,84	28,09	4,00
90%	20	7	10	0	3	10	7	2191,21	1637,87	3600,00	1284,14	28,80	1,68

Table 5.14: Experiment results for well-formed stage$_c$ extensions with a timeout of 1800s and additional 1800s penalty

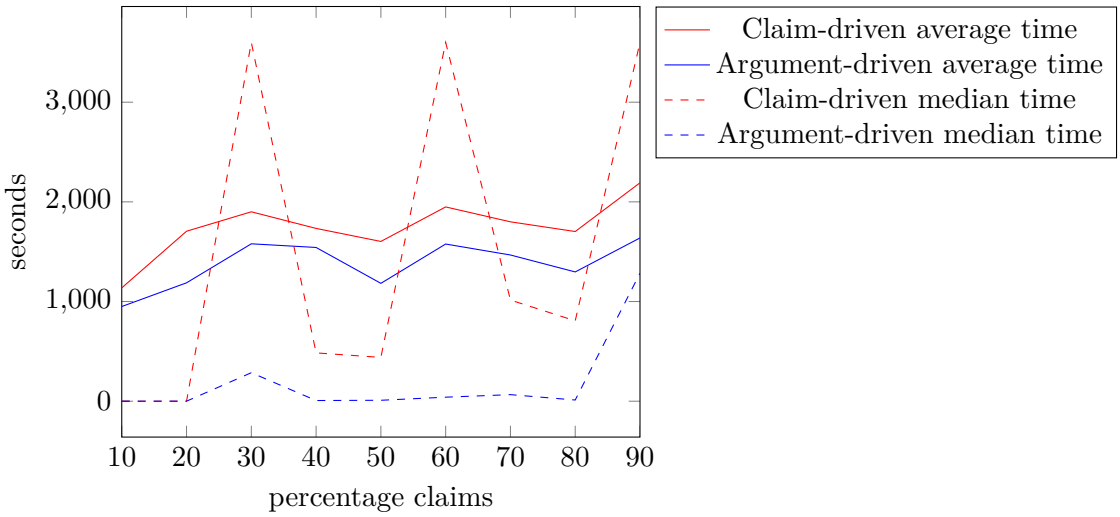

Figure 5.11: Experiment results for well-formed stage$_c$ extensions with a timeout of 1800s and additional 1800s penalty

For the semi-stable$_c$ semantics, the argument-driven approach outperformed the claim-driven approach. Moreover, the claim-driven approach timed out for nearly half the instances for all percentages, causing the median time to spike frequently.

5.4.2 Semi-stable$_c$ semantics

The results for the semi-stable$_c$ semantics are depicted in Table 5.15 and Figure 5.12:

% c	#	solved		ex.		t.o.		avg t (s)		med t (s)		ratio	
		c	a	c	a	c	a	c	a	c	a	avg	med
10%	20	18	20	0	2	0	0	151,66	27,56	2,19	0,80	13,98	1,98
20%	20	14	20	0	6	2	0	566,66	62,13	0,96	1,44	12,90	1,92
30%	20	12	19	0	7	4	0	974,60	108,39	3,63	1,26	14,41	4,07
40%	20	12	19	0	7	3	0	787,22	166,69	11,31	4,16	9,53	3,66
50%	20	12	18	0	6	3	1	891,78	349,19	45,35	5,26	8,33	3,50
60%	20	10	17	0	7	4	2	1055,86	539,69	37,74	11,35	9,59	4,81
70%	20	10	18	0	8	4	1	1072,74	420,96	70,51	12,20	11,53	5,05
80%	20	10	17	0	7	4	1	1080,18	496,59	46,08	14,90	9,69	3,75
90%	20	11	18	0	7	2	1	750,97	453,21	16,79	28,57	13,52	4,98

Table 5.15: Experiment results for non well-formed semi-stable$_c$ extensions with a timeout of 1800s and additional 1800s penalty

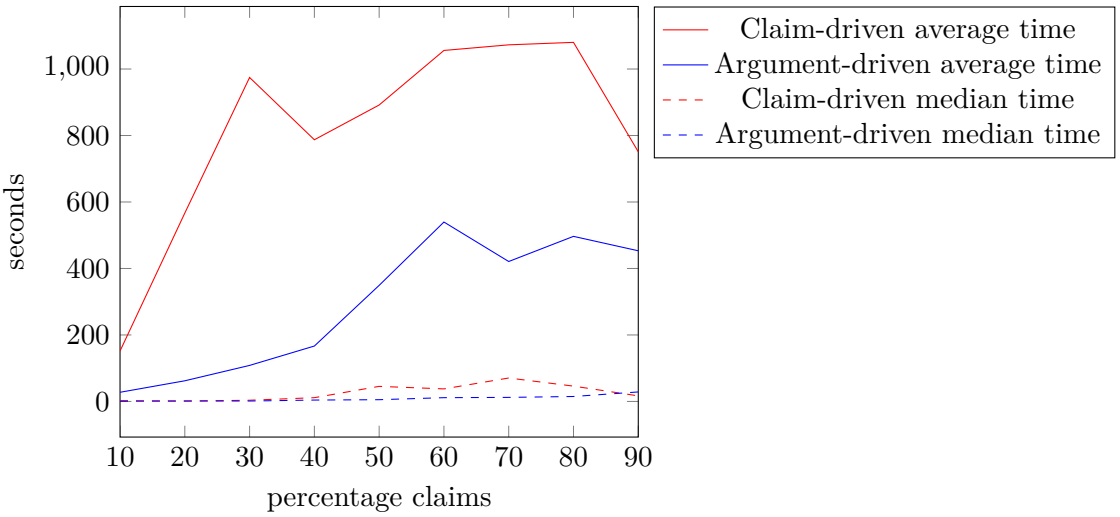

Figure 5.12: Experiment results for well-formed semi-stable$_c$ extensions with a timeout of 1800s and additional 1800s penalty

For the semi-stable$_c$ semantics, the argument-driven approach clearly outperformed the claim-driven approach. Both encodings seem to scale loosely with the percentage of claims, although the claim-driven approach dipped significantly for the 90% instances.

5.4.3 Discussion

Interestingly, the claim-driven encodings performed worse than the argument-driven encodings for both semantics, even though their ranges are defined in terms of claims and the well-formedness allows to compute the conflict-free and admissible sets in polynomial time. Maybe an extension of answer-set programming that allows for them to be computed more directly would be beneficial to the performance of the claim-driven encodings.

5.5 Conversion to abstract argumentation frameworks with collective attacks

In this section, we will present the results of the tests of the conversion approach from well-formed claim augmented abstract argumentation framework to abstract argumentation framework with collective attacks introduced in Subsection 2.3.6. For these test, we selected the argument-driven encodings for the complete and stable semantics presented in Subsection 4.2.3 and the claim-driven preferred encoding presented in Subsection 4.2.4, referred to as "standard". Furthermore, the encodings given in Section 4.1 were used in conjunction with our conversion algorithm presented in Subsection 2.3.6, referred to as "conversion". The conversion was aborted when the output reached a size of 1 gigabyte and such cases were ignored when compiling the runtime results. As before, we will give an overview over all semantics in Table 5.16 and present the detailed results for the individual semantics in the following subsection, finishing with a discussion of the results. However, the standard approach outperformed the conversion approach for all semantics, with the conversion itself either hitting the 1 gigabyte limit or finishing fast. Furthermore, the tables in the subsections of this Section will share the same structure as those in the subsections of the Section 5.2 with an additional column "c total t (s)", giving the average and median conversion times in seconds for a given percentage of claims.

		ST	**CO**	**PR**
# Instances		180	180	180
Solved	standard	170	165	158
	conversion	118	148	80
Exclusively solved	standard	52	17	80
	conversion	0	0	2
Count timeout	standard	10	15	13
	conversion	17	14	11
Avg time solved (s)	standard	239,19	325,13	318,17
	conversion	570,40	333,54	451,06
Median time solved (s)	standard	1,17	0,03	2,13
	conversion	1,62	0,02	0,16
Avg ratio solve time		1,22	1,22	6,27
Median ratio solve time		0,35	1,00	1,78
Count conversion aborted		45	18	37
Average total time (s)		451,42	334,13	578,54
Median total time (s)		0,50	0,33	2,71

Table 5.16: Experiment results for the conversion of well-formed claim augmented abstract argumentation frameworks to abstract argumentation frameworks with collective attacks with a timeout of 1800s and additional 1800s penalty

The structure of the Table 5.16 is the same as for the Table 5.1, with 3 additional lines.

Thus, the line ”Count conversion aborted” gives the number of conversions that were aborted due to the resulting encoding exceeding the file-size limit of 1 gigabyte. Moreover, the lines ”Average total time (s)” and ”Median total time (s)” give the total time, i.e. the solving time plus the conversion time in seconds for the conversion approach and each semantics.

5.5.1 Stable semantics

The results for the stable semantics are depicted in Table 5.17 and Figure 5.13:

% c	solved		ex.		t.o.		avg sv t (s)		med sv t (s)		ratio		c total t (s)	
	s	c	s	c	s	c	s	c	s	c	avg	med	avg	med
10%	20	11	9	0	0	0	10,92	0,16	0,10	0,00	5,75	3,00	0,75	0,30
20%	19	11	8	0	1	1	184,62	300,53	0,37	0,10	2,48	1,00	301,23	0,52
30%	19	11	8	0	1	2	186,60	554,92	0,52	0,31	1,41	0,42	555,78	0,99
40%	19	12	7	0	1	2	191,48	593,51	0,74	0,49	0,80	0,24	596,26	1,40
50%	19	15	4	0	1	2	198,29	590,83	1,18	4,46	0,49	0,17	602,66	5,65
60%	19	16	3	0	1	1	241,56	496,25	1,69	15,88	0,45	0,23	510,60	17,32
70%	19	14	5	0	1	3	228,83	764,21	1,65	6,52	0,40	0,16	779,30	7,81
80%	18	14	4	0	2	3	408,99	743,03	8,70	70,28	0,55	0,29	754,94	78,24
90%	18	14	4	0	2	3	501,46	809,96	15,64	79,51	0,61	0,42	817,6	96,17

Table 5.17: Experiment results for the conversion approach for stable extensions with a timeout of 1800s and additional 1800s penalty

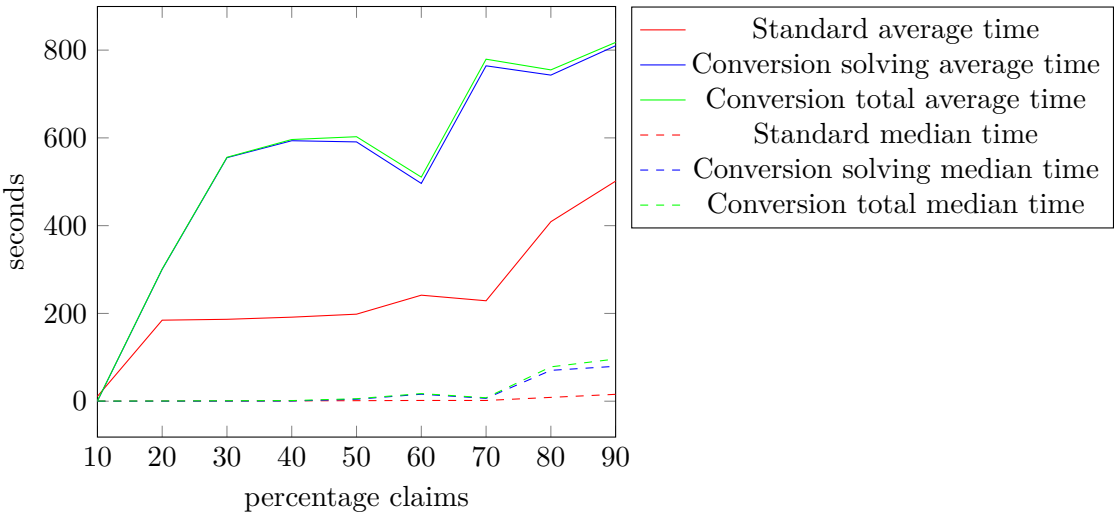

Figure 5.13: Experiment for the conversion approach for stable extensions with a timeout of 1800s and additional 1800s penalty

The standard approach outperformed the conversion approach for the stable semantics. Both approaches seemed to scale with the percentage of claims, with the conversion approach experiencing a dip for the 60% instances.

5.5.2 Complete semantics

The results for the complete semantics are depicted in Table 5.18 and Figure 5.14:

% c	solved		ex.		t.o.		avg sv t (s)		med sv t (s)		ratio		c total t (s)	
	s	c	s	c	s	c	s	c	s	c	avg	med	avg	med
10%	19	18	1	0	1	0	180,59	0,46	0,02	0,00	3,43	1,00	0,87	0,27
20%	19	18	1	0	1	0	180,88	19,90	0,03	0,01	1,98	1,00	20,53	0,32
30%	19	17	2	0	1	1	181,19	224,35	0,03	0,02	1,22	1,00	224,93	0,30
40%	18	16	2	0	2	2	362,00	442,25	0,05	0,05	0,87	1,00	442,82	0,37
50%	18	15	3	0	2	3	426,95	601,50	0,06	0,04	0,84	1,00	601,99	0,35
60%	18	16	2	0	2	2	450,28	440,22	0,10	0,04	0,71	1,00	440,83	0,37
70%	18	16	2	0	2	2	399,66	420,50	0,10	0,07	0,62	0,63	421,16	0,41
80%	18	16	2	0	2	2	377,33	428,48	0,12	0,08	0,65	0,91	429,16	0,44
90%	18	16	2	0	2	2	367,33	424,21	0,15	0,15	0,65	0,67	424,92	0,52

Table 5.18: Experiment results for the conversion approach for complete extensions with a timeout of 1800s and additional 1800s penalty

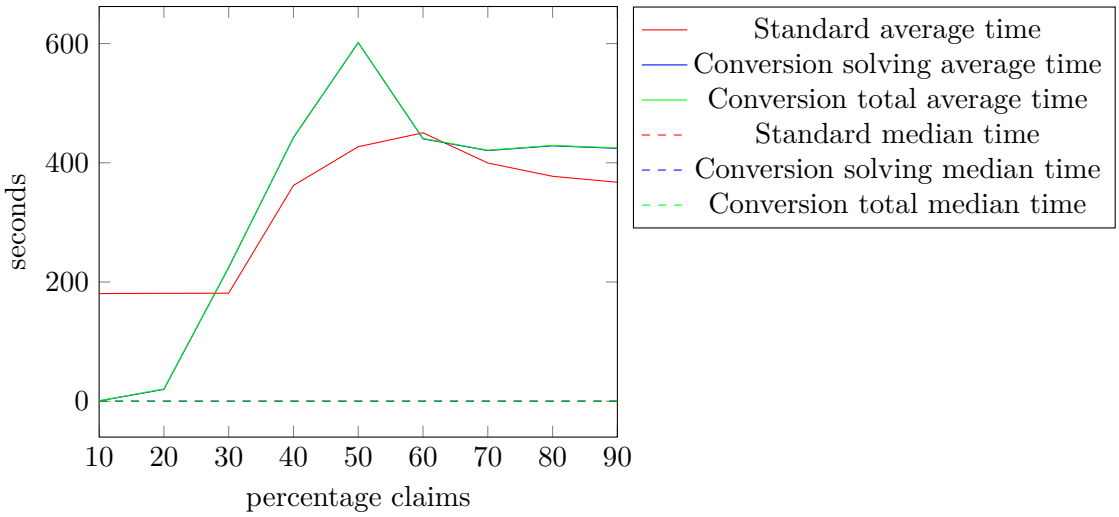

Figure 5.14: Experiment for the conversion approach for complete extensions with a timeout of 1800s and additional 1800s penalty

As for the stable approach, the standard approach outperformed the conversion approach for the complete semantics. However, the conversion approach performed better for instances with a small number of claims. Both approaches scaled with the number of claims until the 60% instances, when both approaches tapered off.

5.5.3 Preferred semantics

The results for the preferred semantics are depicted in Table 5.19 and Figure 5.15:

% c	solved		ex.		t.o.		avg sv t (s)		med sv t (s)		ratio		c total t (s)	
	s	c	s	c	s	c	s	c	s	c	avg	med	avg	med
10%	19	11	9	1	0	0	1,68	0,58	0,32	0,00	38,51	3,00	1,24	0,27
20%	19	10	9	0	0	0	7,49	0,44	0,58	0,01	7,19	3,00	0,75	0,27
30%	18	10	9	1	1	0	199,56	106,24	0,97	0,02	2,60	3,00	106,59	0,29
40%	18	9	9	0	1	1	216,50	360,58	1,15	0,04	2,21	2,00	360,88	0,35
50%	18	9	9	0	1	1	243,85	362,02	1,97	0,16	1,48	1,41	362,35	0,53
60%	18	9	9	0	1	1	250,83	366,00	2,66	0,27	1,21	1,13	366,35	0,63
70%	17	8	9	0	2	2	457,92	729,12	4,86	1,08	1,17	1,07	729,44	1,44
80%	15	7	8	0	4	3	822,01	1084,76	5,53	2,05	1,09	1,00	1085,04	2,04
90%	16	7	9	0	3	3	663,68	1094,85	8,16	17,69	0,97	1,00	1095,15	18,03

Table 5.19: Experiment results for the conversion approach for preferred extensions with a timeout of 1800s and additional 1800s penalty

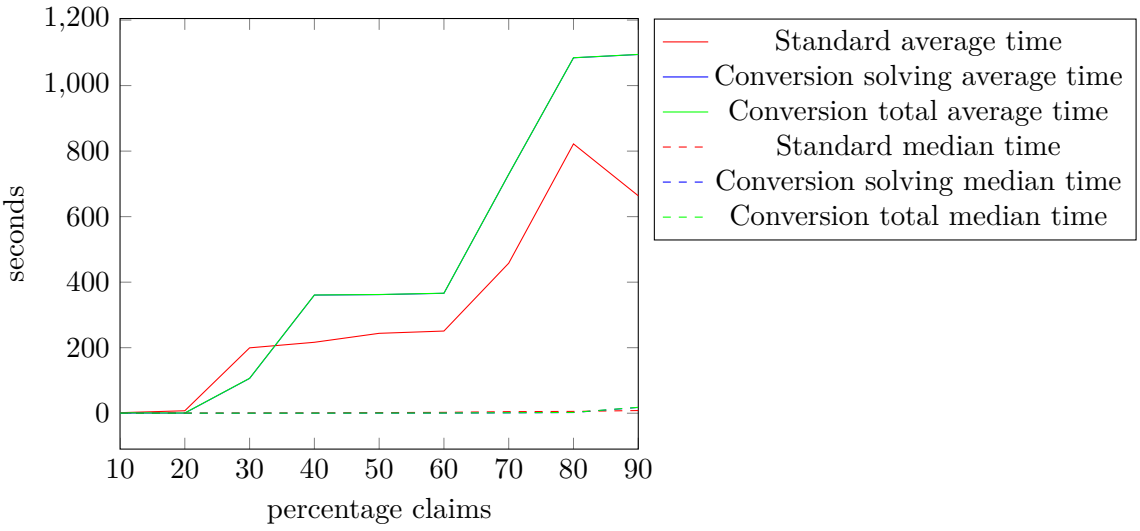

Figure 5.15: Experiment for the conversion approach for preferred extensions with a timeout of 1800s and additional 1800s penalty

As for the previous encodings, the standard approach outperformed the conversion approach for the preferred semantics, except for instances with a small number of claims. Both approaches seem to scale with the percentage of claims, with the standard approach experiencing a dip for the 90% instances.

5.5.4 Discussion

The standard approach clearly outperformed the conversion approach, with many instances hitting the 1 gigabyte limit and thus not being taken into account. Nevertheless, an improved conversion algorithm might be able to produce an improved conversion result faster, which could render the conversion approach more viable.

CHAPTER 6

Conclusion

In this final chapter, we give a summary of the results of this thesis and an outlook for future work.

In Chapter 3, we gave complexity results for abstract argumentation frameworks with collective attacks, summarized in Table 6.1, and for the semi-stable, stage, semi-stable$_c$ and stage$_c$ semantics for claim augmented abstract argumentation frameworks, summarized in Table 6.2 and Table 6.3. Most notably, we showed that the complexity for abstract argumentation frameworks with collective attacks is the same as for Dung abstract argumentation frameworks. Furthermore, the improved upper bounds for the Ver_σ problem for well-formed claim augmented abstract argumentation frameworks in comparison to non well-formed, coNP-c vs Σ_2^P-c, also hold for the semi-stable, stage, semi-stable$_c$ and stage$_c$ semantics.

Furthermore, in Chapter 4, we presented encodings for the conflict-free, admissible, stable, complete, preferred, semi-stable and stage semantics for abstract argumentation frameworks with collective attacks and claim augmented abstract argumentation frameworks as well as encodings for the stage$_c$ and semi-stable$_c$ semantics for claim augmented abstract argumentation frameworks. These encodings have been integrated into the ASPARTIX[1] system to also handle abstract argumentation frameworks with collective attacks and claim augmented abstract argumentation frameworks.

Moreover, recall that we have considered multiple approaches, cf. Chapter 4, to enumerate the extensions of claim augmented abstract argumentation frameworks. However, our experiments in Chapter 5 suggest, that no approach seems to be superior for every semantics. Choosing the right approach for a given semantics can result in a significant performance increase.

[1] https://www.dbai.tuwien.ac.at/research/argumentation/aspartix/

σ	$Cred_\sigma$	$Skept_\sigma$	Ver_σ	$Exists_\sigma$	$Exists_\sigma^{\neg\emptyset}$
conflict-free	in P	trivial	in P	trivial	in P
admissible	NP-c	trivial	in P	trivial	NP-c
complete	NP-c	P-c	in P	trivial	NP-c
preferred	NP-c	Π_2^P-c	coNP-c	trivial	NP-c
stable	NP-c	coNP-c	in P	NP-c	NP-c
semi-stable	Σ_2^P-c	Π_2^P-c	coNP-c	trivial	NP-c
stage	Σ_2^P-c	Π_2^P-c	coNP-c	trivial	in P

Table 6.1: Complexity landscape for abstract argumentation frameworks with collective attacks

σ	$Cred_\sigma$	$Skept_\sigma$	Ver_σ	$Exists_\sigma$	$Exists_\sigma^{\neg\emptyset}$
conflict-free	in P	trivial	NP-c	trivial	in P
admissible	NP-c	trivial	NP-c	trivial	NP-c
complete	NP-c	P-c	NP-c	trivial	NP-c
preferred	NP-c	Π_2^P-c	Σ_2^P-c	trivial	NP-c
stable	NP-c	coNP-c	NP-c	NP-c	NP-c
semi-stable	**Σ_2^P-c**	**Π_2^P-c**	**Σ_2^P-c**	**trivial**	**NP-c**
stage	**Σ_2^P-c**	**Π_2^P-c**	**Σ_2^P-c**	**trivial**	**in P**
semi-stable$_c$	**Σ_2^P-c**	**Π_2^P-c**	**Σ_2^P-c**	**trivial**	**NP-c**
stage$_c$	**Σ_2^P-c**	**Π_2^P-c**	**Σ_2^P-c**	**trivial**	**in P**

Table 6.2: Complexity landscape for non well-formed claim augmented abstract argumentation frameworks. Novel results are highlighted in boldface

Future work includes possible optimizations of our encodings, for example utilizing domain-specific heuristics [GKR+13]. Furthermore, for well-formed claim augmented abstract argumentation frameworks, computation of the unique maximal conflict-free and admissible sets of arguments for a given set of claims might be done more efficiently using external, imperative functions [EFKR12]. Moreover, the computation of a DNF for the conversion approach from well-formed claim augmented abstract argumentation frameworks to abstract argumentation frameworks with collective attacks, cf. Subsection 2.3.6, could probably be improved, resulting in a more concise encoding and/or improved conversion and/or solving time.

σ	$Cred_\sigma$	$Skept_\sigma$	Ver_σ	$Exists_\sigma$	$Exists_\sigma^{\neg\emptyset}$
conflict-free	in P	trivial	in P	trivial	in P
admissible	NP-c	trivial	in P	trivial	NP-c
complete	NP-c	P-c	in P	trivial	NP-c
preferred	NP-c	Π_2^P-c	coNP-c	trivial	NP-c
stable	NP-c	coNP-c	in P	NP-c	NP-c
semi-stable	**Σ_2^P-c**	**Π_2^P-c**	**coNP-c**	**trivial**	**NP-c**
stage	**Σ_2^P-c**	**Π_2^P-c**	**coNP-c**	**trivial**	**in P**
semi-stable$_c$	**Σ_2^P-c**	**Π_2^P-c**	**coNP-c**	**trivial**	**NP-c**
stage$_c$	**Σ_2^P-c**	**Π_2^P-c**	**coNP-c**	**trivial**	**in P**

Table 6.3: Complexity landscape for well-formed claim augmented abstract argumentation frameworks. Novel results are highlighted in boldface

Related work

The work probably most closely related to ours is [EGW08], also utilizing answer-set programming compute extension of argumentation frameworks. However, the most important difference to our work is, that that their work focused on Dung abstract argumentation frameworks, while we focused on abstract argumentation frameworks with collective attacks and claim augmented abstract argumentation frameworks. Moreover, other encodings and algorithms for abstract argumentation frameworks with collective attacks have been proposed [NP06a, SR17].

List of Figures

List of Tables

List of Algorithms

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

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

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

[BD04] Philippe Besnard and Sylvie Doutre. Checking the acceptability of a set of arguments. In James P. Delgrande and Torsten Schaub, editors, *10th International Workshop on Non-Monotonic Reasoning (NMR 2004), Whistler, Canada, June 6-8, 2004, Proceedings*, pages 59–64, 2004.

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

[BDG11] Pietro Baroni, Paul E. Dunne, and Massimiliano Giacomin. On the resolution-based family of abstract argumentation semantics and its grounded instance. *Artif. Intell.*, 175(3-4):791–813, 2011.

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

[BG03] Marcello Balduccini and Michael Gelfond. Diagnostic reasoning with a-prolog. *TPLP*, 3(4-5):425–461, 2003.

[BGLR16] Pietro Baroni, Guido Governatori, Ho-Pun Lam, and Régis Riveret. On the justification of statements in argumentation-based reasoning. In Chitta Baral, James P. Delgrande, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pages 521–524. AAAI Press, 2016.

[BGR16] Pietro Baroni, Guido Governatori, and Régis Riveret. On labelling statements in multi-labelling argumentation. In Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 489–497. IOS Press, 2016.

[Cam07] Martin Caminada. Comparing two unique extension semantics for formal argumentation: Ideal and eager. *Proc. of the 19th Belgian-Dutch Conference on Artificial Intelligence*, pages 81–87, 2007.

[CSAD15] Martin Caminada, Samy Sá, João Alcântara, and Wolfgang Dvořák. On the equivalence between logic programming semantics and argumentation semantics. *Int. J. Approx. Reasoning*, 58:87–111, 2015.

[DB02] Paul E. Dunne and Trevor J. M. Bench-Capon. Coherence in finite argument systems. *Artif. Intell.*, 141(1/2):187–203, 2002.

[DD18] Wolfgang Dvořák and Paul E. Dunne. Computational problems in formal argumentation and their complexity. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 14, pages 631–687. College Publications, 2018. also appears in IfCoLog Journal of Logics and their Applications 4(8):2557–2622.

[DDW13] Paul E. Dunne, Wolfgang Dvořák, and Stefan Woltran. Parametric properties of ideal semantics. *Artif. Intell.*, 202:1–28, 2013.

[DEGV97] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity, Ulm, Germany, June 24-27, 1997*, pages 82–101. IEEE Computer Society, 1997.

[DFW18] Wolfgang Dvořák, Jorge Fandinno, and Stefan Woltran. On the expressive power of collective attacks. In Sanjay Modgil, Katarzyna Budzynska, and John Lawrence, editors, *Computational Models of Argument - Proceedings of COMMA 2018, Warsaw, Poland, 12-14 September 2018*, volume 305 of *Frontiers in Artificial Intelligence and Applications*, pages 49–60. IOS Press, 2018.

[DGW18] Wolfgang Dvořák, Alexander Greßler, and Stefan Woltran. Evaluating SETAFs via answer-set programming. In Matthias Thimm, Federico Cerutti, and Mauro Vallati, editors, *Proceedings of the Second International Workshop on Systems and Algorithms for Formal Argumentation (SAFA 2018) co-located with the 7th International Conference on Computational Models of*

Argument (COMMA 2018), Warsaw, Poland, September 11, 2018., volume 2171 of *CEUR Workshop Proceedings*, pages 10–21. CEUR-WS.org, 2018.

[DRW19] Wolfgang Dvořák, Anna Rapberger, and Stefan Woltran. Shaping abstract argumentation for the argumentation pipeline – how to avoid multiple arguments with the same claim. Technical Report DBAI-TR-2019-115, Technische Universität Wien, Database and Artificial Intelligence Group, 2019. `https://www.dbai.tuwien.ac.at/research/report/dbai-tr-2019-115.pdf`.

[DT96] Yannis Dimopoulos and Alberto Torres. Graph theoretical structures in logic programs and default theories. *Theor. Comput. Sci.*, 170(1-2):209–244, 1996.

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

[Dvo12] Wolfgang Dvořák. *Computational Aspects of Abstract Argumentation*. PhD thesis, Technische Universität Wien, 2012.

[DW09] Paul E. Dunne and Michael J. Wooldridge. Complexity of abstract argumentation. In Guillermo Ricardo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 85–104. Springer, 2009.

[DW10] Wolfgang Dvořák and Stefan Woltran. Complexity of semi-stable and stage semantics in argumentation frameworks. *Inf. Process. Lett.*, 110(11):425–430, 2010.

[DW11] Wolfgang Dvořák and Stefan Woltran. On the intertranslatability of argumentation semantics. *J. Artif. Intell. Res.*, 41:445–475, 2011.

[DW19] Wolfgang Dvořák and Stefan Woltran. Complexity of abstract argumentation under a claim-centric view. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.*, pages 2801–2808. AAAI Press, 2019.

[EFKR12] Thomas Eiter, Michael Fink, Thomas Krennwallner, and Christoph Redl. Conflict-driven ASP solving with external sources. *TPLP*, 12(4-5):659–679, 2012.

[EG93] Thomas Eiter and Georg Gottlob. Complexity results for disjunctive logic programming and application to nonmonotonic logics. In Dale Miller, editor, *Logic Programming, Proceedings of the 1993 International Symposium,*

Vancouver, British Columbia, Canada, October 26-29, 1993, pages 266–278. MIT Press, 1993.

[EG95] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.*, 15(3-4):289–323, 1995.

[EGL16] Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. *AI Magazine*, 37(3):53–68, 2016.

[EGW08] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. ASPARTIX: implementing argumentation frameworks using answer-set programming. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, volume 5366 of *Lecture Notes in Computer Science*, pages 734–738. Springer, 2008.

[EIST05] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 90–96. Professional Book Center, 2005.

[EKR+17] Thomas Eiter, Tobias Kaminski, Christoph Redl, Peter Schüller, and Antonius Weinzierl. Answer set programming with external source access. In Giovambattista Ianni, Domenico Lembo, Leopoldo E. Bertossi, Wolfgang Faber, Birte Glimm, Georg Gottlob, and Steffen Staab, editors, *Reasoning Web. Semantic Interoperability on the Web - 13th International Summer School 2017, London, UK, July 7-11, 2017, Tutorial Lectures*, volume 10370 of *Lecture Notes in Computer Science*, pages 204–275. Springer, 2017.

[EW06] Uwe Egly and Stefan Woltran. Reasoning in argumentation frameworks using quantified boolean formulas. In Paul E. Dunne and Trevor J. M. Bench-Capon, editors, *Computational Models of Argument: Proceedings of COMMA 2006, September 11-12, 2006, Liverpool, UK*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 133–144. IOS Press, 2006.

[FB19] Giorgos Flouris and Antonis Bikakis. A comprehensive study of argumentation frameworks with sets of attacking arguments. *Int. J. Approx. Reasoning*, 109:55–86, 2019.

[GH11] Nikos Gorogiannis and Anthony Hunter. Instantiating abstract argumentation with classical logic arguments: Postulates and properties. *Artif. Intell.*, 175(9-10):1479–1497, 2011.

[GKR+13] Martin Gebser, Benjamin Kaufmann, Javier Romero, Ramón Otero, Torsten Schaub, and Philipp Wanko. Domain-specific heuristics in answer set programming. In *AAAI*. AAAI Press, 2013.

[GKS09] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Solution enumeration for projected boolean search problems. In Willem Jan van Hoeve and John N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings*, volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2009.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press, 1988.

[GMR19] Martin Gebser, Marco Maratea, and Francesco Ricca. The seventh answer set programming competition: Design and results. *CoRR*, abs/1904.09134, 2019.

[Lif99] Vladimir Lifschitz. Action languages, answer sets, and planning. In Krzysztof R. Apt, Victor W. Marek, Mirek Truszczynski, and David S. Warren, editors, *The Logic Programming Paradigm: A 25-Year Perspective*, pages 357–373. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[Lif02] Vladimir Lifschitz. Answer set programming and plan generation. *Artif. Intell.*, 138(1-2):39–54, 2002.

[MT99] Victor W. Marek and Miroslaw Truszczynski. Stable logic programming - an alternative logic programming paradigm. In Krzysztof R. Apt, Victor W. Marek, Miroslaw Truszczynski, and David S. Warren, editors, *The Logic Programming Paradigm: A 25-Year Perspective*. Springer, 1999.

[NBG+01] Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An a-prolog decision support system for the space shuttle. In I. V. Ramakrishnan, editor, *Practical Aspects of Declarative Languages, Third International Symposium, PADL 2001, Las Vegas, Nevada, USA, March 11-12, 2001, Proceedings*, volume 1990 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2001.

[NP06a] Søren Holbech Nielsen and Simon Parsons. Computing preferred extensions for argumentation systems with sets of attacking arguments. In Paul E. Dunne and Trevor J. M. Bench-Capon, editors, *Computational Models of Argument: Proceedings of COMMA 2006, September 11-12, 2006, Liverpool,*

UK, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 97–108. IOS Press, 2006.

[NP06b] Søren Holbech Nielsen and Simon Parsons. A generalization of Dung's abstract framework for argumentation: Arguing with sets of attacking arguments. In Nicolas Maudet, Simon Parsons, and Iyad Rahwan, editors, *Argumentation in Multi-Agent Systems, Third International Workshop, ArgMAS 2006, Hakodate, Japan, May 8, 2006, Revised Selected and Invited Papers*, volume 4766 of *Lecture Notes in Computer Science*, pages 54–73. Springer, 2006.

[PM11] Raquel Mochales Palau and Marie-Francine Moens. Argumentation mining. *Artif. Intell. Law*, 19(1):1–22, 2011.

[Pol05] Axel Polleres. Semantic web languages and semantic web services as application areas for answer set programming. In Gerhard Brewka, Ilkka Niemelä, Torsten Schaub, and Miroslaw Truszczynski, editors, *Nonmonotonic Reasoning, Answer Set Programming and Constraints, 24.-29. April 2005*, volume 05171 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[Pol17] Sylwia Polberg. *Developing the abstract dialectical framework*. PhD thesis, TU Wien, Institute of Information Systems, 2017.

[SR17] Chiaki Sakama and Tjitze Rienstra. Representing argumentation frameworks in answer set programming. *Fundam. Inform.*, 155(3):261–292, 2017.

[TB01] Le-Chi Tuan and Chitta Baral. Effect of knowledge representation on model based planning: experiments using logic programming encodings. In Alessandro Provetti and Tran Cao Son, editors, *Answer Set Programming, Towards Efficient and Scalable Knowledge Representation and Reasoning, Proceedings of the 1st Intl. ASP'01 Workshop, Stanford, CA, USA, March 26-28, 2001*, 2001.

[Tur37] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

[vEGK+14] Frans H. van Eemeren, Bart Garssen, Erik C. W. Krabbe, A. Francisca Snoeck Henkemans, Bart Verheij, and Jean H. M. Wagemans, editors. *Handbook of Argumentation Theory*. Springer, 2014.

[YVC18] Bruno Yun, Srdjan Vesic, and Madalina Croitoru. Toward a more efficient generation of structured argumentation graphs. In Sanjay Modgil, Katarzyna Budzynska, and John Lawrence, editors, *Computational Models of Argument - Proceedings of COMMA 2018, Warsaw, Poland, 12-14 September 2018*,

volume 305 of *Frontiers in Artificial Intelligence and Applications*, pages 205–212. IOS Press, 2018.