FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Complexity Results and Algorithms for Argumentation

Dung's Frameworks and Beyond

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

Johannes Peter Wallner
Registration Number 0327240

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dr. Stefan Woltran
Advisor: Assistant Prof. Georg Weissenbacher, D.Phil.
Assistance: Dr. Wolfgang Dvořák

The dissertation has been reviewed by:

(Associate Prof. Dr. Stefan Woltran)

(Prof. Gerhard Brewka)

Wien, 03.04.2014

(Johannes Peter Wallner)

Erklärung zur Verfassung der Arbeit

Johannes Peter Wallner
Favoritenstraße 9–11, 1040 Wien

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.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

Every work builds upon the support of many people. This is also the case with this thesis. First I would like to thank my family, my parents Ludmilla and Wolfgang who supported me in all situations in life (in addition to giving me one in the first place). Thanks also to my brother Matthias, who told me to give the study of computer science a try. Further I want to thank Matthias' wife Sabine and in particular their son Tobias for being an inspiration.

During my work I had the opportunity to have a lot of splendid colleagues. I enjoyed the fruitful discussions, chats and amicable atmosphere here at the Institute of Information Systems. I cannot give here everyone enough words for the support they gave me, but at least I want to list them in a (probably) incomplete and unsorted set. I want to thank {Günther Charwat, Matti Järvisalo, Friedrich Slivovsky, Srdjan Vesic, Markus Pichlmair, Stefan Ellmauthaler, Florian Lonsing, Andreas Pfandler, Christof Redl, Sylwia Polberg, Reinhard Pichler, Toni Pisjak, Hannes Strass, Ringo Baumann, Georg Weissenbacher, Magdalena Widl, Frank Loebe, Thomas Krennwallner, Lara Spendier, Emanuel Sallinger, Bernhard Bliem, Martin Lackner, Axel Polleres, Martin Baláž, Jozef Frtús, Dragan Doder, Nysret Musliu, Giselle Reis, Daria Stepanova, Thomas Linsbichler, Paolo Baldi, Martin Kronegger}.

There are some people without this work and indeed my whole job at the institute, would not have been fruitful and even more probably would have been actually impossible. As in movies, I give special thanks to Gerhard Brewka, for having me as a guest in Leipzig, I enjoyed the stay and learnt a lot there. Further I thank my two "seniors", Sarah Gaggl and Wolfgang Dvořák, who particularly supported me in our project and gave me a lot of insights into academic life. A very special thanks goes to Stefan Woltran. Not only because he is my main supervisor, but because he gave me invaluable knowledge, support and understanding. I count myself lucky to be your student.

During my studies I have been given the opportunity to participate in the doctoral programme "Mathematical Logic in Computer Science". I am thankful for the beneficial atmosphere and the interdisciplinary workshops within this programme, which was funded by the Vienna University of Technology. Last, but not least, I thank the Vienna Science and Technology Fund (WWTF), Deutsche Forschungsgemeinschaft (DFG) and the Austrian Science Fund (FWF) for funding the following research projects I participated in: ICT08-028, ICT12-015 and I1102.

Abstract

In the last couple of decades argumentation emerged as an important topic in Artificial Intelligence (AI). Having its origin in philosophy, law and formal logic, argumentation in combination with computer science has developed into various formal models, which nowadays are found in diverse applications including legal reasoning, E-Democracy tools, medical applications and many more. An integral part of many formal argumentation theories within AI is a certain notion of abstraction. Hereby the actual contents of arguments are disregarded, but only the relation between them is used for reasoning purposes. One very influential formal model for representing discourses abstractly are Dung's argumentation frameworks (AFs). AFs can simply be represented as directed graphs. The vertices correspond to abstract arguments and directed edges are interpreted as an attack relation for countering arguments. Many variants and generalizations of AFs have been devised, with abstract dialectical frameworks (ADFs) among the most general ones. ADFs are even more abstract than AFs: the relation between arguments is not fixed to the concept attack, but is specified via so-called acceptance conditions, describing the relationship via Boolean functions.

The main computational challenge for AFs and ADFs is to compute jointly acceptable sets of arguments. Several criteria, termed semantics, have been proposed for accepting arguments. Applications of AFs or ADFs unfortunately face the harsh reality that almost all reasoning tasks defined for the frameworks are intractable. Decision problems for AFs can even be hard for the second level of the polynomial hierarchy. ADFs generalize AFs and thus are at least as computationally complex, but exact complexity bounds of many ADF problems are lacking in the literature.

There have been some proposals how to implement reasoning tasks on AFs. Roughly these can be classified into reduction and direct approaches. The former approach solves the problem at hand by translation to another one, for which sophisticated solvers exist. However, at the start of this thesis, reduction approaches for argumentation were purely monolithic. Monolithic reduction approaches result into a single encoding and hardly incorporate domain-specific optimizations for more efficient computation. Direct approaches exploit structural or semantical properties of AFs for efficiency but must typically devise and implement an algorithm from scratch, including the highly consuming task of engineering solutions on a very deep algorithmic level e.g. the development of suitable data structures.

In this thesis we provide three major contributions to the state of the art in abstract argumentation. First, we develop a novel hybrid approach that combines strengths of reduction and direct approaches. Our method reduces the problem at hand to iterative satisfiability (SAT) solving,

i.e. a sequence of calls to a SAT-solver. Due to hardness for the second level of the polynomial hierarchy, we cannot avoid exponentially many SAT calls in the worst case. However, by exploiting inherent parameters of AFs, we provide a bound on the number of calls. Utilizing modern SAT technology to an even greater extent, we also employ more expressive variants of the SAT problem. It turns out that minimal correction sets (MCSes) and backbones of Boolean formulae are very well suited for our tasks. Like the iterative SAT algorithms, our algorithms based upon MCSes and backbones are hybrid approaches as well. Yet they are closer to monolithic reduction approaches and offer the benefit of requiring even less engineering effort and providing more declarativeness.

Our second major contribution is to generalize our algorithms to ADFs. For doing so we first considerably extend ADF theory and provide a thorough complexity analysis for ADFs. Our results show that the reasoning tasks for ADFs are one step up in the polynomial hierarchy compared to their counterparts on AFs. Even though problems on ADFs suffer from hardness up to the third level of the polynomial hierarchy, our analysis shows that bipolar ADFs (BADFs) are not affected by this complexity jump. BADFs restrict the relations between arguments to be either of an attacking or supporting nature, but still offer a variety of interesting relations.

Finally our third contribution is an empirical evaluation of implementations of our algorithms. Our novel algorithms outperform existing state-of-the-art systems for abstract argumentation. These results show that our hybrid approaches are indeed promising and that the provided proof-of-concept implementations can pave the way for applications for handling problems of increasing size and complexity.

Kurzfassung

In den letzten Jahrzehnten hat sich die Argumentationstheorie als wichtiges Teilgebiet der Künstlichen Intelligenz (KI) etabliert. Die vielfältigen Wurzeln dieses jungen Gebietes liegen sowohl in der Philosophie, in den Rechtswissenschaften, als auch in der formalen Logik. Unterstützt durch die Computerwissenschaften ergaben sich aus dieser Theorie ebenso vielfältige Anwendungen, unter anderem für rechtliche Beweisführung, Medizin und E-Democracy. Als zentraler Aspekt, der in vielen Formalisierungen von Argumentation auftaucht, erweist sich eine gewisse Form von Abstraktion. Meist wird in einem solchen Abstraktionsprozess von konkreten Inhalten von Argumenten Abstand genommen und nur deren logische Relation betrachtet. Ein einflussreiches formales Model für die abstrakte Repräsentation von Diskursen sind die so genannten Argumentation Frameworks (AFs), entwickelt von Phan Minh Dung. In diesen AFs werden Argumente einfach als abstrakte Knoten in einem Graph dargestellt. Gerichtete Kanten repräsentieren wiederum die Relation zwischen Argumenten, welche in AFs als Angriff interpretiert wird. Beispielsweise kann ein attackierendes Argument als Gegenargument gesehen werden. In der Literatur wurden diverse Aspekte von AFs verallgemeinert, oder erweitert. Ein besonders genereller Vertreter dieser formalen Modelle sind die Abstract Dialectical Frameworks (ADFs). ADFs sind noch abstrakter als AFs, denn die Relation in ADFs beschränkt sich nicht auf Attacken, sondern kann mittels so genannter Akzeptanzbedingungen frei spezifiziert werden. Diese Bedingungen werden mittels boolescher Formeln modelliert.

Die logischen Semantiken dieser Argumentationsstrukturen bestehen aus verschiedenen Kriterien zur Akzeptanz von Argumenten. Die automatische Berechnung von Mengen von Argumenten die gemeinsam akzeptiert werden können ist eine der wichtigsten Aufgaben auf AFs und ADFs. Allerdings haben praktisch alle solche Problemstellungen eine hohe Berechnungskomplexität. Manche Probleme auf AFs sind sogar hart für die zweite Stufe der polynomiellen Hierarchie. AFs sind Spezialfälle von ADFs, daher sind ADFs auch mindestens so komplex was die Berechnung der Semantiken angeht. Eine genaue Analyse der Komplexitätsschranken war jedoch offen für ADFs.

Um diese Aufgaben dennoch zu bewältigen sind einige Algorithmen für AFs entwickelt worden. Man kann diese in zwei Richtungen klassifizieren. Die erste Richtung beschäftigt sich mit Reduktionen oder auch Übersetzungen. Dabei wird das Ursprungsproblem in ein anderes kodiert, für das performante Systeme existieren. Zu Beginn der Arbeiten an dieser Dissertation waren Reduktionen allerdings von rein monolithischer Natur. Das heißt, dass diese Reduktionen das Ursprungsproblem in eine einzelne Kodierung übersetzen. Zudem wurden in diesen Übersetzungen kaum Optimierungen berücksichtigt. Die zweite Richtung besteht aus direkten Methoden. Diese können leichter strukturelle oder semantische Eigenschaften von AFs nutzen

um effizient Lösungen zu berechnen. Allerdings müssen hierfür Algorithmen von Grund auf neu implementiert werden. Das beinhaltet auch die zeitintensive Aufgabe geeignete Datenstrukturen und andere technische Details auszuarbeiten.

Unser Beitrag zum wissenschaftlichen Stand der Technik in der Argumentationstheorie lässt sich in drei Bereiche gliedern. Erstens entwickeln wir hybride Ansätze, welche die Stärken von Reduktionen und direkten Methoden vereinen. Dabei reduzieren wir die Problemstellungen auf eine Folge von booleschen Erfüllbarkeitsproblemen (SAT). Bei Problemen die hart für die zweite Stufe der polynomiellen Hierarchie sind, lässt es sich, unter komplexitätstheoretischen Annahmen, nicht vermeiden, dass wir im schlimmsten Fall eine exponentielle Anzahl solcher Teilprobleme lösen müssen. Durch Verwendung von inhärenten Parametern von AFs können wir diese Anzahl jedoch beschränken. Zusätzlich zu dem klassischen SAT Problem, zeigen wir, dass sich Probleme in der Argumentation natürlicherweise auf Erweiterungen des SAT Problems reduzieren lassen. Konkret nutzen wir minimal correction sets (MCSes) und backbones von booleschen Formeln hierfür. Reduktionen zu diesen sind ebenfalls hybrid, allerdings näher zu strikt monolithischen Ansätzen und auch deklarativer als die anderen hybriden Methoden.

Als zweites Ergebnis unserer Arbeit stellen wir Verallgemeinerungen unserer hybriden Ansätze für ADFs vor. Hierfür erweitern wir zuerst die theoretische Basis und zeigen wesentliche Komplexitätsresultate von ADFs. Es stellt sich heraus, dass Entscheidungsprobleme von ADFs im Allgemeinen eine um eine Stufe höhere Komplexität aufweisen als die jeweiligen Probleme auf AFs. Trotz der Tatsache, dass damit manche Probleme auf ADFs hart für die dritte Stufe der polynomiellen Hierarchie sind, können wir zeigen, dass eine wichtige Teilklasse von ADFs, die so genannten bipolaren ADFs (BADFs), nicht von der erhöhten Komplexität betroffen sind.

Unser dritter Beitrag besteht aus Implementierungen unserer Methoden für AFs und deren experimentelle Evaluierung. Wir zeigen, dass unsere Implementierungen performanter als andere Systeme im Bereich der Argumentationstheorie sind. Diese Resultate deuten daraufhin, dass unsere hybriden Ansätze gut geeignet sind um die schwierigen Aufgaben, die in dieser Theorie vorkommen, zu bewältigen. Insbesondere können unsere prototypischen Softwareimplementierungen den Weg für Anwendungen, mit noch größeren Strukturen umzugehen, ebnen.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

1.1 Argumentation Theory in AI

Imagine you present your vision to an audience. Some listeners might be curious about your speech and excited about everything you say. Some others might be skeptical about your ideas and remain distanced. An important topic deserves to be heard and fairly judged in any case. What does one do in this circumstance? You try to convince the audience of the importance of your vision and give persuading arguments. Persuasion has many forms. Aristotle distinguished three modes of persuasion: appealing to the presenter's authority or honesty (*ethos*), to the audience's emotions (*pathos*) or to logic (*logos*). We focus here on argumentation by appealing to logic. Arguing is so natural to all of us that we do it all the time. Argumentation is part of our daily lives when we discuss projects with colleagues, talk to partners about deciding where to go for holiday, or try to persuade parents to watch a certain kind of movie. To our mind the following citation succinctly summarizes the most important aspects of argumentation theory.

> *"In its classical treatment within philosophy, the study of argumentation may, informally, 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."* Bench-Capon and Dunne [18]

In the last couple of decades argumentation emerged as a distinct field within Artificial Intelligence (AI) from considerations in philosophy, law and formal logic, constituting nowadays an important subfield of AI [18, 22, 139]. A natural question is how argumentation in AI differs from classical logic? Bench-Capon and Dunne [18] argue as follows. In logic one *proves* statements. If a proof exists then the statement is not refutable. In contrast, the aim of arguments is to *persuade*, not to be formally proven. Moreover, arguments are *defeasible*. Thus, what made an argument convincing, might not be convincing anymore in the light of new information. Argumentation therefore can be considered as non-monotonic, that is, it might be necessary to retract

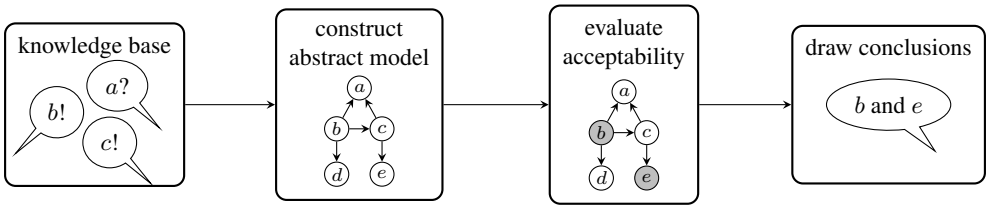

Figure 1.1: Argumentation process

conclusions. Classical logic on the other hand is a monotonic formalism. What is proven correct once, remains correct.

Although a formal mathematical proof of an argument is typically not possible, we still would like to assess whether a certain argumentation is reasonable or persuasive. With the everyday examples one can maybe live with less reasonable choices, but there are many areas where a more strict assessment is necessary. It would be disastrous to consider arbitrary arguments in court to be valid or persuasive. Physicians discussing medical treatment should base their opinions on facts and scientific theory. Indeed in science itself one has to argue why some conclusion should be considered scientific and others not [111].

Automated reasoning is one of the main aspects of AI and argumentation provides a particular challenge due to its non-monotonic derivations. Nevertheless, argumentation has found its way into a variety of applications, such as in legal reasoning [17, 19], decision support systems [1], E-Democracy tools [43, 44], E-Health tools [150], medical applications [95, 138], multi-agent systems [125], and more. An integral concept many formal argumentation theories and systems share is a certain notion of *abstraction*. Embedded in a larger workflow or *argumentation process* [40], formal models of argumentation represent arguments in an abstract way. Briefly put, in the argumentation process we assume a given knowledge base, which holds diverging opinions of some sort. From this we construct an abstract representation of the relevant arguments for our current discourse and model their relations. Typically in this representation one abstracts away from the internal structure of the arguments [137]. Several formal models for the abstract representation have been developed [33]. In the third step we evaluate the arguments. Evaluation in this context often refers to the act of finding sets of jointly acceptable arguments. Criteria for jointly accepting arguments are called *semantics* of the formal models. Finally we draw conclusions from the accepted arguments. For instance an abstract argument may stand for a propositional logic formula [23]. Logically entailed formulae of accepted arguments are then further conclusions. This process is outlined in Figure 1.1.

Dung's argumentation frameworks One of the most prominent formal models for the second and third step of the argumentation process, the construction and evaluation of an abstract model, are Dung's argumentation frameworks (AFs) [59]. The key feature of these frameworks is to provide an abstract notion of arguments and a binary relation between these arguments, interpreted as attack. Consider for example putting forth an argument c, followed by a counterargument b, which is again "attacked" by a counterargument a. This situation could be modeled as shown in Figure 1.2 on the left side. The vertices of this graph are the arguments and the

Figure 1.2: Frameworks for argumentation

directed edges represent the attack relation.

Several semantics have been proposed for AFs [11]. These yield so-called extensions, which are jointly acceptable sets of arguments. A particularly important semantics for AFs is the *preferred* semantics. If we consider the current state of our discourse in Figure 1.2 on the left then, intuitively speaking, we can accept a, since no counterargument was proposed. It is reasonable not to accept b, since we have accepted a counterargument for b. For c we have accepted a so-called defender, namely a which attacks c's attacker. The set $\{a, c\}$ would be *admissible* in the AF. The arguments a and c are not in a direct conflict (no directed edge between those two) and for each attacker on this set a defender is also inside the set. A set which is maximal admissible is called a preferred extension of the AF. An AF might have multiple preferred extensions in general.

Generalizations of Dung's AFs Many generalizations of Dung's AFs have been developed (for an overview see [33]). Recently Brewka and Woltran [34] devised a new kind of framework, which is among the most general abstract frameworks for argumentation. Their formalism is called abstract dialectical frameworks (ADFs) and allows many different relations between arguments. With more modeling capacities, ADFs are situated closer to application domains. For instance in Figure 1.2 on the right side we see a simple ADF. Here the edges are interpreted not necessarily as attacks, but as an abstract relation. The edges are called links in ADFs. A link from argument a to b means that a is a parent of b. The concrete relation between an argument and its parents is specified in the *acceptance condition*. Each argument has one such acceptance condition, written in the figure as a Boolean formula below the argument. Given the status of each of the parents, the acceptance conditions tells us whether we may accept the argument or not. The argument a is always acceptable, denoted by the acceptance condition $\top$, while b can be accepted only if a is also accepted. This can be interpreted as a kind of support relation. On the other hand the relation between b and c is an AF like attack. The acceptance condition of c requires that b is *not* accepted. Semantics of ADFs are defined via three-valued interpretations. These assign each argument either true, false or undecided. Many of the AF semantics were generalized to ADFs [31, 34, 136, 144]. For instance the three-valued interpretation which sets a, b to true and c to false would be admissible in the ADF in Figure 1.2. Although ADFs are a quite recent development, they are very actively researched [29, 32, 89, 145, 147].

State of the art Computing extensions, or interpretations in the ADF case, is one of the main computational challenges in *abstract argumentation*. However, despite the elegancy in design

of the frameworks, practical applications have to face the harsh reality of the high complexity of the reasoning tasks defined on the frameworks, which are mostly intractable. In fact some of the decision problems for AFs are even situated on the second level of the polynomial hierarchy [68] and thus harder than NP complete problems, under standard complexity theoretic assumptions. ADFs generalize AFs and thus the corresponding tasks on ADFs are at least as computationally complex, but exact complexity bounds of many ADF problems are lacking in the literature.

Some approaches to solve hard argumentation problems have been studied (see [47] for an overview), which are typically based on reductions, i.e. translations to other formalisms, or directly provide an algorithm exploiting semantical properties of AFs. However, at the start of this thesis, *reduction approaches* for argumentation were purely monolithic. Monolithic reduction approaches translate the problem at hand into a single encoding and hardly incorporate optimizations for more efficient computation. Moreover, no advanced techniques for reductions to e.g. the satisfiability problem (SAT) [118] have been studied for applicability for abstract argumentation. SAT is the prototypical target for many reduction-based approaches and the canonical NP-complete problem. Nowadays very efficient SAT-solvers are available [84]. Only AF problems situated in the first level of the polynomial hierarchy were reduced to SAT [21]. Non reduction-based approaches, or *direct approaches*, can use domain-specific properties more easily for optimizing efficiency, but must devise and implement an algorithm from scratch, including the highly consuming task of engineering efficient solutions on a very deep algorithmic level e.g. the development of suitable data structures.

1.2 Main Contributions

We extend the state of the art in abstract argumentation with three major contributions.

Advanced Algorithms for AFs We develop a novel *hybrid* approach that combines strengths of reduction and direct approaches. Our algorithms, which we call *search algorithms* use SAT-solvers in an iterative fashion. For computing extensions of a semantics, our search algorithms work internally on a simpler semantics, called base semantics. The idea is to traverse the search space of the base semantics to find extensions of the target semantics. Each step in the search space is delegated to a SAT-solver. All the tasks we solve in this way are "beyond" NP, i.e. hardness was shown for a class above NP for the corresponding decision problem. For the issue of the potential exponential number of SAT calls for second level problems, we use complexity theoretic results for fragments of AFs w.r.t. certain parameters to have a bound on the number of these calls. If we fix inherent parameters of the AF in question, then our search algorithms require only a polynomial number of SAT calls. The main parameter we base our algorithms on is the size of the AF times the number of extensions for a given semantics. Our search algorithms are applicable for preferred, semi-stable [41] and stage [151] semantics and can enumerate all extensions or answer queries for credulous or skeptical acceptance of an argument. An argument is credulously accepted if it is in at least one extension of the semantics. It is skeptically accepted if it is in all extensions of the semantics.

Furthermore we utilize modern SAT technology to an even greater extent and apply variants of SAT to our problems. SAT variants or SAT extensions are often computed via iterative

SAT-solving. We use two extensions of the SAT problem in this thesis. The first are *minimal correction sets* (MCSes) [115]. A correction set is a subset of clauses of an unsatisfiable Boolean formula, which if dropped results in a satisfiable subformula. We apply MCSes to problems for the semi-stable, stage and eager [39] semantics. The main idea of these semantics is based on minimizing the set of arguments, which are neither in an extension or attacked by it (i.e. arguments outside the so-called "range") in addition with other constraints. We utilize MCSes by constructing formulae, such that unsatisfied clauses correspond to arguments not in the range. Adapted MCS solvers can then be used for solving our tasks. The second SAT variant we consider is the so-called *backbone* [110]. A backbone of a satisfiable Boolean formula is the set of literals the formula entails. This concept and backbone solvers can be used to instantiate an algorithm [62] for ideal semantics [60]. Like the search algorithms, our algorithms based upon MCSes and backbones are hybrid approaches as well. Yet they are closer to monolithic reduction approaches and offer the benefit of requiring even less engineering effort, more declarative queries and still reduce to iterative SAT.

Novel Complexity Results and Algorithms for ADFs In this thesis we show how to generalize the search algorithm from AFs to ADFs for preferred semantics. For reduction-based approaches (also non-monolithic ones) it is imperative to understand the exact complexity of the reasoning tasks to solve. Consider a problem which is NP-complete. The upper bound shows the applicability of SAT-solvers. Under standard complexity theoretic assumptions, the lower bound implies that a "simpler" solver, not capable of solving problems for the first level of the polynomial hierarchy, is not sufficient for the problem.

The computational complexity of ADFs was largely unknown. We fill this gap in ADF theory by proving that the computational complexity for tasks on ADFs is exactly "one level higher" in the polynomial hierarchy compared to their counterparts on AFs. This insight indicates that straightforward generalizations of our search algorithms for AFs require more powerful search engines as SAT-solvers, such as solvers for quantified Boolean formulae (QBFs) [117] or disjunctive answer-set programming (ASP) [98]. Even though problems for ADFs suffer from hardness for a class up to the third level of the polynomial hierarchy, we show a positive result for an interesting subclass of ADFs: we prove that for bipolar ADFs (BADFs) [34] with known link types, the complexity *does not increase* compared to the corresponding decision problems on AFs. BADFs offer a variety of relations between arguments such as support and collective attacks of several arguments. As a byproduct to our complexity analysis on ADFs we develop a backbone algorithm for general ADFs and grounded semantics.

Empirical Evaluation Our third major contribution is an empirical evaluation of implementations of our algorithms. We implemented query-based search algorithms for preferred, semi-stable and stage semantics for AFs in the software CEGARTIX. Query-based algorithms answer the question of credulous or skeptical acceptance of an argument w.r.t. a semantics. CEGARTIX outperformed ASPARTIX [85], an existing state-of-the-art system for abstract argumentation based on disjunctive ASP. For SAT extensions, we implemented a set of tools and compared them to ASPARTIX and CEGARTIX. Both were outperformed by the tools based on SAT extensions. However CEGARTIX showed in many cases a comparable performance to the SAT

extension based tools. The results show that our hybrid approaches for abstract argumentation are indeed promising and that the provided proof-of-concept implementations can pave the way for applications for handling problems of increasing size and complexity.

1.3 Structure of the Thesis

The structure of our thesis is outlined in the following.

- In Chapter 2 we provide the necessary background of propositional logic (Section 2.2), AFs and ADFs (Section 2.3) and recall complexity results derived for the frameworks (Section 2.4);
- in Chapter 3 we develop hybrid algorithms for AFs. We distinguish between search algorithms and algorithms based on SAT extensions:
  - in Section 3.2 we review existing complexity results for classes of AFs bounded by inherent parameters;
  - Section 3.3 develops the search algorithms for preferred (Section 3.3.2), semi-stable and stage semantics (Section 3.3.3) and shows how to instantiate them using SAT-solvers. Further we show variants of the search algorithm tailored to query-based reasoning (credulous and skeptical) in Section 3.3.4;
  - SAT extensions are the basis of the algorithms developed in Section 3.4. Here we first give an overview of existing results for MCSes and backbones in Section 3.4.1 and then introduce our algorithms utilizing MCSes in Section 3.4.2 for semi-stable and stage semantics. We extend these algorithms to compute eager and stage-ideal semantics in Section 3.4.3. Backbones are applied in Section 3.4.4 for the computation of ideal semantics.
- Chapter 4 shows how to generalize the algorithms for AFs to ADFs by first establishing a clear picture of the computational complexity of problems on ADFs:
  - we first prove complexity results of ADFs and BADFs in Section 4.1.1 and Section 4.1.2 respectively, for the grounded, admissible and preferred semantics;
  - then we generalize the search algorithms to ADFs for preferred semantics in Section 4.2.1 and show a backbone algorithm for grounded semantics in Section 4.2.2.
- in Chapter 5 we empirically evaluate our approach and experiment with implementations of the algorithms for AFs. We show that the approach is viable and even outperforms existing state-of-the-art systems for abstract argumentation:
  - we first describe our two implementations. CEGARTIX in Section 5.1.1, which implements the query-based search algorithm for preferred, semi-stable and stage semantics. For the algorithms via SAT extensions we describe our implementations in Section 5.1.2, which solve tasks for the semi-stable and eager semantics;

  - after introducing our test setup (Section 5.2.1), we show the promising results of our empirical evaluation of CEGARTIX and the implementations via SAT extensions. CEGARTIX is compared to ASPARTIX in Section 5.2.2. We compare internal parameters of CEGARTIX in Section 5.2.3 and the use of different SAT-solvers in Section 5.2.4. We experiment with the SAT extension algorithms in Section 5.2.5.

- Finally in Chapter 6 we recapitulate our contributions and discuss related work as well as future directions.

1.4 Publications

Parts of the results in this thesis have been published. In the following we list the relevant publications and indicate which sections contain the corresponding contributions.

[31] Gerhard Brewka, Stefan Ellmauthaler, Hannes Strass, Johannes P. Wallner, and Stefan Woltran. *Abstract Dialectical Frameworks Revisited*. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, pages 803–809. AAAI Press / IJCAI, 2013. Section 4.1

[76] Wolfgang Dvořák, Matti Järvisalo, Johannes P. Wallner, and Stefan Woltran. *Complexity-Sensitive Decision Procedures for Abstract Argumentation*. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning, KR 2012*, pages 54–64. AAAI Press, 2012. Sections 3.3 and 5.2

[77] Wolfgang Dvořák, Matti Järvisalo, Johannes P. Wallner, and Stefan Woltran. *Complexity-Sensitive Decision Procedures for Abstract Argumentation*. *Artificial Intelligence*, 206:53–78, 2014. Sections 3.3 and 5.2

[147] Hannes Strass and Johannes P. Wallner. *Analyzing the Computational Complexity of Abstract Dialectical Frameworks via Approximation Fixpoint Theory*. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning, KR 2014*. To appear, 2014. Section 4.1

[153] Johannes P. Wallner, Georg Weissenbacher, and Stefan Woltran. *Advanced SAT Techniques for Abstract Argumentation*. In João Leite, Tran Cao Son, Paolo Torroni, Leon van der Torre, and Stefan Woltran, editors, *Proceedings of the 14th International Workshop on Computational Logic in Multi-Agent Systems, CLIMA 2013*, volume 8143 of *Lecture Notes in Artificial Intelligence*, pages 138–154. Springer, 2013. Section 3.4.1

A longer version of [147] is published as a technical report [146]. A brief system description of CEGARTIX was presented in [75].

CHAPTER 2

Background

In this chapter we introduce the formal background for our work. After some notes on the used notation and general definitions, we start by introducing propositional logic in Section 2.2. Propositional logic underlies or is strongly related to many of the approaches and concepts we use in this thesis. Then in Section 2.3 we introduce formal argumentation theory and in particular the problems we aim to study in this work. Afterwards in Section 2.4, we review the basics of computational complexity and the current state of complexity analysis of problems in argumentation theory.

2.1 General Definitions and Notation

We write variables as $X, x, Y, y, v, I, i, \ldots$ and format constants of special interest differently. For a set of elements X we mean by $\overline{X}$ a new set with $\overline{X} = \{\overline{x} \mid x \in X\}$, i.e. uniformly renaming every element in the set X. Functions, or sometimes called mappings or operators, as usual assign to each element in their domain an element of their codomain. The arity of a function is the number of operands it accepts. For a unary function f we denote by $dom(f)$ the domain of f. We compare unary functions w.r.t. equality, which holds iff both functions have the same domain and assign to each element in the domain the same element, i.e. the functions f and f' are equal, denoted as usual by $f = f'$, iff $dom(f) = dom(f')$ and for all $x \in dom(f)$ it holds that $f(x) = f'(x)$. For a unary function f and a set X we use the restriction of f to X, denoted by $f|_X$, to represent the function f' with $dom(f') = dom(f) \cap X$ and for all $x \in X$ it holds that $f(x) = f'(x)$.

A partially ordered set is a pair $(S, \sqsubseteq)$ with $\sqsubseteq \subseteq S \times S$ a binary relation on S which is reflexive (if $x \in S$ then $x \sqsubseteq x$), antisymmetric (if $x \sqsubseteq y$ and $y \sqsubseteq x$ then $x = y$) and transitive (if $x \sqsubseteq y$ and $y \sqsubseteq z$ then $x \sqsubseteq z$). A function $f : S \to S$ is $\sqsubseteq$-monotone if for each $s, s' \in S$ with $s \sqsubseteq s'$ we also have $f(s) \sqsubseteq f(s')$. Monotone operators on partially ordered sets or special subclasses of partially ordered sets play a major role in argumentation theory and other logical formalisms.

Lastly, every structure we consider in this thesis is assumed to be *finite*. This assumption simplifies some analysis.

2.2 Propositional Logic

Propositional logic can be seen as a formal fundament of many logic-based approaches, formalisms and algorithms. Propositional logic or strongly related concepts are present for instance in the area of satisfiability solving [24], in logic programming [6, 30, 99], other (more expressive) logics or in conditions of many imperative programming languages. Here we introduce the comparably simple and yet expressive propositional logic or also called Boolean logic extended with so-called quantifiers. There is a lot of material available on this topic and the interested reader is pointed to e.g. [48, 104]. We begin with the syntax of this logical language. After that we define the semantics of propositional logic with two and three truth values.

2.2.1 Propositional Formulae

The syntax of propositional logic is based on a set of propositional variables or atoms $\mathcal{P}$. This forms the vocabulary of our language and each element in it is considered a proposition. Sentences, expressions or formulae are built in this language using logical connectives. We typically denote propositional logic formulae as ϕ, ψ, χ or φ. For the connectives we consider the symbols '$\neg$', '$\wedge$', '$\vee$', '$\rightarrow$', '$\leftarrow$' and '$\leftrightarrow$', denoting the logical negation, conjunction, disjunction, (material) implication in two directions and equivalence. Except for negation, these are all binary connectives. We assume that these symbols do not occur in $\mathcal{P}$. Naturally as in any (formal) language there is a grammar to formulate well-formed sentences in propositional logic.

Definition 2.2.1. *Let $\mathcal{P}$ be a set of propositional atoms. We define the set of well-formed propositional formulae over $\mathcal{P}$ ($\mathcal{WFF}_\mathcal{P}$) inductively as follows.*

- $\mathcal{P} \subseteq \mathcal{WFF}_\mathcal{P}$;
- $\{\top, \bot\} \subseteq \mathcal{WFF}_\mathcal{P}$;
- *if $\phi \in \mathcal{WFF}_\mathcal{P}$, then $\neg\phi \in \mathcal{WFF}_\mathcal{P}$; and*
- *if $\phi, \psi \in \mathcal{WFF}_\mathcal{P}$, then $(\phi \circ \psi) \in \mathcal{WFF}_\mathcal{P}$ for $\circ \in \{\wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow\}$.*

We usually omit the subscript and just write $\mathcal{WFF}$ if it is clear from the context to which set of propositional atoms we refer. The two special constants $\top$ and $\bot$ represent the absolute truth and falsity and we assume w.l.o.g. that $\top$ and $\bot$ are not present in our set of propositional variables $\mathcal{P}$. In addition we consider parentheses for easier readability and to uniquely identify the structure of a formula, i.e. if $\phi \in \mathcal{WFF}$, then also $(\phi) \in \mathcal{WFF}$. If we omit the parentheses, then the following order of connectives is applied: $\neg$, $\leftrightarrow$, $\wedge$, $\vee$, $\rightarrow$, $\leftarrow$. This means that e.g. $a \wedge b \vee c$ is considered equal to $(a \wedge b) \vee c$.

Example 2.2.1. *Let $\mathcal{P} = \{p, q\}$. Then the set $\mathcal{WFF}$ built from $\mathcal{P}$ contains among infinitely many other formulae the sentences $p \wedge q$, $\neg p$, $\top \rightarrow \bot$, $(p \vee q) \wedge \neg q$ and $q \vee \neg q$.*

Subsequently we assume that any propositional formula ϕ is well-formed, i.e. $\phi \in \mathcal{WFF}$ and by a formula ϕ we mean that ϕ is a propositional formula. The set of propositional variables $\mathcal{P}$ is extracted from the current context, that is, if we build formulae from a set of variables X, then $X \subseteq \mathcal{P}$. This means we assume that $\mathcal{P}$ contains all the propositional atoms we require. Later on it will be useful to consider the set of propositional atoms occurring in a formula ϕ, which we denote by the unary function $atoms$. This function returns all variables occurring in a formula ϕ and is defined as follows.

Definition 2.2.2. *We define the function* $atoms : \mathcal{WFF} \to 2^{\mathcal{P}}$ *recursively as follows, with* ϕ, ψ, $\chi \in \mathcal{WFF}$*:*

- *if* $\phi \in \mathcal{P}$*, then* $atoms(\phi) = \{\phi\}$*;*
- *if* $\phi \in \{\top, \bot\}$*, then* $atoms(\phi) = \emptyset$*;*
- *if* $\phi = \neg\psi$*, then* $atoms(\phi) = atoms(\psi)$*; and*
- *if* $\phi = (\psi \circ \chi)$*, then* $atoms(\phi) = atoms(\psi) \cup atoms(\chi)$ *for* $\circ \in \{\wedge, \vee, \to, \leftarrow, \leftrightarrow\}$.

Furthermore one can similarly define the concept of a subformula recursively, given a formula ϕ then ψ is a subformula of ϕ if ψ occurs in ϕ.

Definition 2.2.3. *We define the function* $subformulae : \mathcal{WFF} \to 2^{\mathcal{WFF}}$ *recursively as follows, with* ϕ, ψ, $\chi \in \mathcal{WFF}$*:*

- *if* $\phi \in \mathcal{P}$*, then* $subformulae(\phi) = \{\phi\}$*;*
- *if* $\phi \in \{\top, \bot\}$*, then* $subformulae(\phi) = \{\phi\}$*;*
- *if* $\phi = \neg\psi$*, then* $subformulae(\phi) = \{\phi\} \cup subformulae(\psi)$*; and*
- *if* $\phi = (\psi \circ \chi)$*, then* $subformulae(\phi) = \{\phi\} \cup subformulae(\psi) \cup subformulae(\chi)$ *if* $\circ \in \{\wedge, \vee, \to, \leftarrow, \leftrightarrow\}$.

Example 2.2.2. *Let* $\phi = p \wedge q \vee r$ *and* $\psi = \neg\neg p \wedge p \vee \neg p$*, then* $atoms(\phi) = \{p, q, r\}$ *and* $atoms(\psi) = \{p\}$*. Further* $subformulae(\phi) = \{(p \wedge q \vee r), p \wedge q, p, q, r\}$.

Certain structures or patterns of formulae occur often to express certain properties. Therefore it will be convenient to use a uniform renaming of a formula ϕ to ψ according to certain rules.

Definition 2.2.4. *We define the renaming or substitution function in postfix notation* $\cdot[\phi_1/\psi]$*, mapping formulae to formulae, recursively as follows, with* ϕ, ϕ_1, ψ, $\chi \in \mathcal{WFF}$*.*

- $\phi[\phi_1/\psi] = \psi$ *if* $\phi = \phi_1$*;*
- $\phi[\phi_1/\psi] = \phi$ *if* $\phi_1 \neq \phi$ *and* $\phi \in \mathcal{P} \cup \{\top, \bot\}$*;*
- $(\neg\phi)[\phi_1/\psi] = \neg(\phi[\phi_1/\psi])$ *if* $\phi_1 \neq \neg\phi$*; and*
- $(\phi \circ \chi)[\phi_1/\psi] = (\phi[\phi_1/\psi]) \circ (\chi[\phi_1/\psi])$ *if* $\phi_1 \neq \phi \circ \chi$ *for* $\circ \in \{\wedge, \vee, \to, \leftarrow, \leftrightarrow\}$.

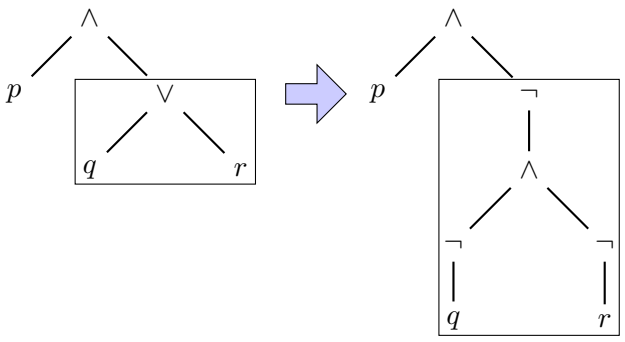

Figure 2.1: Example of the substitution $(p \wedge (q \vee r))[(q \vee r)/\neg(\neg q \wedge \neg r)]$

Intuitively we just replace every occurrence of a subformula ϕ_1 with the formula ψ in a formula, which we exemplify in the following.

Example 2.2.3. *Let $\phi = \neg(p \wedge q)$, then $\phi[p/(r \vee q)] = \neg((r \vee q) \wedge q)$. A more complex example can be seen in Figure 2.1. Here we view the propositional formula as a tree, with each node a connective or an atom occurring in the formula. In this view a subtree corresponds to a subformula. The rectangle represents the subformula to be replaced in the left tree and the replaced subformula in the right tree.*

In certain cases we will rename every atom in a formula ϕ uniformly. For instance if we want to rename every atom p of ϕ to $p^{\mathtt{u}}$, then we denote this by $\phi^{\mathtt{u}}$. More formally we define this as follows.

Definition 2.2.5. *Let ϕ be a formula with $atoms(\phi) = \{x_1, \ldots, x_n\}$ the set of atoms in ϕ. Then we define $\phi^y = \phi[x_1/x_1^y] \cdots [x_n/x_n^y]$.*

This operation gives us a formal handle to easily make *copies* of a formula, e.g. if ϕ is a formula, then $\phi^1 \wedge \phi^2$ is a new formula, which is a conjunction of two subformulae, each of them is just ϕ renamed in two different ways. Note that we assume that any "copied" atom is also present in $\mathcal{P}$.

Example 2.2.4. *Let $\phi = \neg(p \wedge q)$, then $\phi^{\mathtt{u}} = \neg(p^{\mathtt{u}} \wedge q^{\mathtt{u}})$. Further $\phi^1 \wedge \phi^2 = \neg(p^1 \wedge q^1) \wedge \neg(p^2 \wedge q^2)$.*

2.2.2 Quantified Boolean Formulae

A generalization of propositional logic is to extend the set of well-formed formulae to *quantified* formulae, by quantifying propositional variables. We consider the two quantifiers $\exists$ and $\forall$, denoting the existential and universal quantifier respectively. The syntax of the set of well-formed quantified Boolean formulae $\mathcal{QBF}$ is defined as follows, again inductively. For more details on this extension of propositional logic we refer the reader to the corresponding chapter in the Handbook of Satisfiability [36] and also [20, 35, 117].

Definition 2.2.6. *Let $\mathcal{P}$ be a set of propositional atoms and $\mathcal{WFF}_\mathcal{P}$ the set of well-formed formulae built from $\mathcal{P}$. We define $\mathcal{QBF}_\mathcal{P}$ inductively as follows.*

- $\mathcal{WFF}_\mathcal{P} \subseteq \mathcal{QBF}_\mathcal{P}$;
- *if* $\phi \in \mathcal{QBF}_\mathcal{P}$, *then* $\neg\phi \in \mathcal{QBF}_\mathcal{P}$;
- *if* $\phi, \psi \in \mathcal{QBF}_\mathcal{P}$, *then* $\phi \circ \psi \in \mathcal{QBF}_\mathcal{P}$ *for* $\circ \in \{\wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow\}$; *and*
- *if* $\phi \in \mathcal{QBF}_\mathcal{P}$ *and* $p \in \mathcal{P}$, *then* $Qp\phi \in \mathcal{QBF}_\mathcal{P}$ *for* $Q \in \{\exists, \forall\}$.

As in the case of $\mathcal{WFF}$ we usually omit the subscript for $\mathcal{P}$ if this is clear from the context. Further the order of the connectives for omitting parentheses is extended to include the new quantifiers. We order these two before the negation.

Example 2.2.5. *The following formulae are in $\mathcal{QBF}$: $\exists p \forall q(p \vee q)$ and $p \wedge \exists q \leftrightarrow p$.*

Unless noted otherwise we assume that a formula ϕ is in $\mathcal{WFF}$ and specifically mention if a formula contains quantifiers, i.e. is in $\mathcal{QBF} \setminus \mathcal{WFF}$, if it is not clear from the context. The variable x of a quantifier Qx is the quantified variable. For a sequence of the same quantifiers and their quantified variables $Qx_1 \cdots Qx_n\phi$ we use the shorthand $Q\{x_1, \ldots, x_n\}\phi$. In the following we assume that any quantified Boolean formula ϕ is well-formed, i.e. if ϕ is a formula with quantifiers, then $\phi \in \mathcal{QBF}$.

Another important concept is the scope of a quantified atom. A variable occurrence in a QBF may be either bound or free. Intuitively speaking a variable occurrence x is bound in ϕ if it is in the scope of a quantifier, e.g. $\phi = Qx\psi$.

Definition 2.2.7. *We define the function free : $\mathcal{QBF} \rightarrow 2^\mathcal{P}$ recursively as follows, with ϕ, ψ, $\chi \in \mathcal{QBF}$:*

- *if* $\phi \in \mathcal{P}$, *then* $\mathit{free}(\phi) = \{\phi\}$;
- *if* $\phi \in \{\top, \bot\}$, *then* $\mathit{free}(\phi) = \emptyset$;
- *if* $\phi = \neg\psi$, *then* $\mathit{free}(\phi) = \mathit{free}(\psi)$;
- *if* $\phi = \psi \circ \chi$, *then* $\mathit{free}(\psi \circ \chi) = \mathit{free}(\psi) \cup \mathit{free}(\chi)$ *if* $\circ \in \{\wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow\}$; *and*
- *if* $\phi = Qx\psi$, *then* $\mathit{free}(\phi) = \mathit{free}(\psi) \setminus \{x\}$ *for* $Q \in \{\exists, \forall\}$.

A QBF ϕ is said to be *open* if $\mathit{free}(\phi) \neq \emptyset$ and *closed* otherwise. Note that in the special case that a formula $\phi \in \mathcal{WFF}$ does not contain quantifiers and no atoms of $\mathcal{P}$, i.e. it contains only connectives and symbols of $\{\top, \bot\}$, then this QBF is also closed.

Example 2.2.6. *Let $\phi = \exists p(p \wedge q)$, $\psi = \exists p \neg p$ and $\chi = \top \vee \bot$, then all three ϕ, ψ and χ are in $\mathcal{QBF}$ and the first one is open, while the other two are closed, i.e. $\mathit{free}(\phi) = \{q\}$ and $\mathit{free}(\psi) = \emptyset = \mathit{free}(\chi)$.*

We define a slightly different renaming function for QBFs which acts only on *free* occurrences of variables, which will be used for defining the semantics of QBFs.

Definition 2.2.8. *We define the renaming function of free atoms in QBFs in postfix notation* $\cdot[p/\psi]_{free}$*, mapping QBFs to QBFs, recursively as follows, with* $\phi, \psi, \chi \in \mathcal{QBF}$ *and* $p \in \mathcal{P}$*.*

- $p[p/\psi]_{free} = \psi$*;*
- $x[p/\psi]_{free} = x$ *if* $p \neq x$ *and* $x \in \mathcal{P} \cup \{\top, \bot\}$*;*
- $(\neg\phi)[p/\psi]_{free} = \neg(\phi[p/\psi]_{free})$*;*
- $(\phi \circ \chi)[p/\psi]_{free} = (\phi[p/\psi]_{free}) \circ (\chi[p/\psi]_{free})$ *for* $\circ \in \{\wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow\}$*;*
- $Qp\phi[p/\psi]_{free} = Qp\phi$ *for* $Q \in \{\exists, \forall\}$*; and*
- $Qx\phi[p/\psi]_{free} = Qx(\phi[p/\psi]_{free})$ *if* $x \neq p$ *for* $Q \in \{\exists, \forall\}$*.*

2.2.3 Normal Forms

Normal forms of logical formulae play a major role in both theory and practical solving. The reason for this is a uniform structure of a formula in a certain normal form, which one can exploit to solve reasoning tasks on the formula. Moreover algorithms can be streamlined to work with such normal forms and allow for more efficiency. Many normal forms exist on logical formulae. We define the conjunctive normal form (CNF) for propositional formulae in $\mathcal{WFF}$ and the prenex normal form (PNF) of formulae in $\mathcal{QBF}$. As we will see later in Section 2.2.4 any formula in $\mathcal{WFF}$ can be rewritten to a CNF formula and any formula in $\mathcal{QBF}$ can be rewritten to a PNF formula while preserving important properties.

The basic building block for a formula in CNF is the *literal*. A literal is simply an element of $\mathcal{P} \cup \{\top, \bot\}$ or its negation, i.e. $\neg p$ for $p \in \mathcal{P} \cup \{\top, \bot\}$. A disjunction of literals is called a *clause*. A CNF is now a conjunction of clauses.

Definition 2.2.9. *Let* ϕ *be a formula. Then* ϕ *is*

- *a* literal *if* $\phi = p$ *or* $\phi = \neg p$ *for* $p \in \mathcal{P} \cup \{\top, \bot\}$*;*
- *a* clause *if* $\phi = l_1 \vee \cdots \vee l_n$ *with* $l_1, \ldots, l_n$ *literals and* $n \geq 0$*; or*
- in conjunctive normal form (CNF) *if* $\phi = c_1 \wedge \cdots \wedge c_m$ *with* $c_1, \ldots, c_m$ *clauses and* $m \geq 0$*.*

Note that a formula may be literal, clause and in CNF at the same time. In particular a literal is always also a clause and a clause is always in CNF. Further we identify clauses with sets of literals and a formula in CNF by a set of clauses. An empty set of literals represents $\bot$ and an empty set of clauses $\top$. Consider the following example.

Example 2.2.7. *Let* $\phi = p$*,* $\psi = p \vee \neg q \vee r$ *and* $\chi = (p \vee q) \wedge (\neg p \vee r)$*. Then* ϕ *is a literal,* ϕ *and* ψ *are clauses and all three* ϕ*,* ψ *and* χ *are in CNF. Alternatively we may write e.g.* $\psi = \{\{p, \neg q, r\}\}$*. As an example for a formula not in CNF consider* $\chi = (p \wedge r) \rightarrow (q \wedge r)$*. The formula* χ *is neither a literal, a clause nor in CNF.*

For a formula with quantifiers a widely used normal form is the so-called prenex normal form. Intuitively a QBF is in PNF if the quantifiers are all collected left of the formula.

Definition 2.2.10. *Let ϕ be a QBF, $\psi \in \mathcal{WFF}$, $Q_i \in \{\exists, \forall\}$ for $1 \leq i \leq n$ and $\{x_1, \ldots x_n\} \subseteq \mathcal{P}$. Then ϕ is in* prenex normal form (PNF) *if $\phi = Q_1 x_1 \cdots Q_n x_n \psi$.*

Classes of PNFs which are of special interest in our work are QBFs in PNF which begin with a certain quantifier and then have n quantifier "alternations", meaning that we start with e.g. $\exists$, followed possibly by several $\exists$ quantifiers, then have one or more $\forall$ quantifiers and then one or more $\exists$ quantifiers and so on until we have n alternations between quantifiers. If we have only one quantifier then $n = 1$. As we will see later in Section 2.4, these classes can be used for characterization of certain complexity classes.

Definition 2.2.11. *Let ϕ be a QBF in PNF, $\psi \in \mathcal{WFF}$, $n > 0$ an integer and $X_i \subseteq \mathcal{P}$ for $1 \leq i \leq n$. The formula ϕ is*

- *in $\mathcal{QBF}_{\exists,n}$ if $\phi = \exists X_1 \forall X_2 \exists X_3 \cdots Q X_n \psi$ with $Q = \exists$ if n is odd and $Q = \forall$ else, or*
- *in $\mathcal{QBF}_{\forall,n}$ if $\phi = \forall X_1 \exists X_2 \forall X_3 \cdots Q X_n \psi$ with $Q = \forall$ if n is odd and $Q = \exists$ else.*

Example 2.2.8. *Let $\psi \in \mathcal{WFF}$ and $\phi = \exists\{x_1, x_2\} \forall\{x_3, x_4, x_5\} \exists\{x_6\} \psi$. Then $\phi \in \mathcal{QBF}_{\exists,3}$ and ϕ is in PNF.*

2.2.4 Semantics

In this section we introduce the logical semantics of propositional logic with quantifiers. First we deal with the *classical* approach. Here the very basic idea is that every proposition can be true or false. This means we can assign to each proposition a *truth value*. In classical logic we have two truth values: true, which we denote with the constant '$\mathbf{t}$' and false, which we denote as '$\mathbf{f}$'. Further we introduce the semantics of *three-valued* propositional logic which includes the truth value undecided or unknown, which we write as '$\mathbf{u}$'. Three-valued semantics for QBFs can also be defined, but we do not require such a semantics for our work. Hence we only introduce the three-valued semantics for unquantified propositional logic.

The most important concept for the semantics we use here is called the *interpretation* function.

Definition 2.2.12. *Let $V \subseteq \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$. An interpretation I is a function $I : \mathcal{P} \to V$. If $V = \{\mathbf{t}, \mathbf{f}\}$ then I is* two-valued. *If $V = \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ then I is* three-valued.

We only consider two or three-valued interpretations. We usually say that an interpretation is defined directly on a set of atoms or on a formula or omit this information if it is clear from the context, i.e. if I is defined on a propositional formula ϕ, then we mean I is defined on $atoms(\phi)$. For a QBF ϕ we define interpretations on $free(\phi)$. Note that in case I is defined on $\phi \in \mathcal{QBF}$, then $dom(I) = \emptyset$ if ϕ is closed. We use the symbols $I, J, K, \ldots$ for interpretations and $\mathcal{I}, \mathcal{J}, \mathcal{K}$ for sets of interpretations. In the following, unless noted otherwise, we assume that an interpretation I is two-valued. By a partial interpretation I on a set S we mean that $dom(I) \subseteq S$.

Interpretations can equivalently be represented as sets or as a triple of sets, depending if they are two or three-valued. First we define this handy shortcut for the set of atoms mapped to a certain truth value.

Definition 2.2.13. *Let I be an interpretation defined on the set X and $v \in \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$. We define $I^v = \{x \mid I(x) = v\}$.*

Now we can view a two-valued interpretation I also as the set $I^{\mathbf{t}}$. This set defines the assignment as follows, if I is defined on X, then every element in the set $I^{\mathbf{t}}$ is assigned true by I and every element in $X \setminus I^{\mathbf{t}} = I^{\mathbf{f}}$ is assigned false (or in case of unrestricted domain $\mathcal{P} \setminus I^{\mathbf{t}}$). A three-valued interpretation I can be presented equivalently by the triple $(I^{\mathbf{t}}, I^{\mathbf{f}}, I^{\mathbf{u}})$.

An interpretation I on a formula ϕ represents a particular logical view on the propositions in the formula, meaning that certain propositions are assigned true, false or undecided. A formula can be *evaluated* under an interpretation defined on it. Such a formula evaluates to a unique truth value under the interpretation. This unique value represents the truth value of the formula given the logical view on the propositional atoms defined via the interpretation function. We begin with the two-valued scenario. By a slight abuse of notation, which will be convenient, we apply I not only on atoms, but also on formulae, i.e. $I(\phi)$ denotes the evaluation of ϕ under I.

Definition 2.2.14. *Let $\phi \in \mathcal{QBF}$, ψ and χ be subformulae of ϕ, $x \in \mathcal{P}$ and I a two-valued interpretation on ϕ. The* evaluation *of ϕ under I, denoted as $I(\phi)$ is defined recursively as follows.*

- *if $\phi = p$, $p \in \mathcal{P}$, then $I(\phi) = I(p)$,*
- *if $\phi = \top$, then $I(\phi) = \mathbf{t}$,*
- *if $\phi = \bot$, then $I(\phi) = \mathbf{f}$,*
- *if $\phi = \neg\psi$, then $I(\phi) = \mathbf{t}$ iff $I(\psi) = \mathbf{f}$,*
- *if $\phi = \psi \wedge \chi$, then $I(\phi) = \mathbf{t}$ iff $I(\psi) = I(\chi) = \mathbf{t}$,*
- *if $\phi = \psi \vee \chi$, then $I(\phi) = \mathbf{t}$ iff $I(\psi) = \mathbf{t}$ or $I(\chi) = \mathbf{t}$,*
- *if $\phi = \psi \rightarrow \chi$, then $I(\phi) = \mathbf{t}$ iff $I(\psi) = \mathbf{f}$ or $I(\chi) = \mathbf{t}$,*
- *if $\phi = \psi \leftarrow \chi$, then $I(\phi) = \mathbf{t}$ iff $I(\psi) = \mathbf{t}$ or $I(\chi) = \mathbf{f}$,*
- *if $\phi = \psi \leftrightarrow \chi$, then $I(\phi) = \mathbf{t}$ iff $I(\psi) = I(\chi)$,*
- *if $\phi = \exists x\psi$, then $I(\phi) = \mathbf{t}$ iff $I(\psi[x/\top]_{free}) = \mathbf{t}$ or $I(\psi[x/\bot]_{free}) = \mathbf{t}$,*
- *if $\phi = \forall x\psi$, then $I(\phi) = \mathbf{t}$ iff $I(\psi[x/\top]_{free}) = \mathbf{t}$ and $I(\psi[x/\bot]_{free}) = \mathbf{t}$.*

If $I(\phi) \neq \mathbf{t}$, then $I(\phi) = \mathbf{f}$.

$\neg$	
$\mathbf{t}$	$\mathbf{f}$
$\mathbf{f}$	$\mathbf{t}$

$\vee$	$\mathbf{t}$	$\mathbf{f}$
$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}$
$\mathbf{f}$	$\mathbf{t}$	$\mathbf{f}$

$\wedge$	$\mathbf{t}$	$\mathbf{f}$
$\mathbf{t}$	$\mathbf{t}$	$\mathbf{f}$
$\mathbf{f}$	$\mathbf{f}$	$\mathbf{f}$

$\rightarrow$	$\mathbf{t}$	$\mathbf{f}$
$\mathbf{t}$	$\mathbf{t}$	$\mathbf{f}$
$\mathbf{f}$	$\mathbf{t}$	$\mathbf{t}$

$\leftrightarrow$	$\mathbf{t}$	$\mathbf{f}$
$\mathbf{t}$	$\mathbf{t}$	$\mathbf{f}$
$\mathbf{f}$	$\mathbf{f}$	$\mathbf{t}$

Figure 2.2: Truth tables for classical two–valued logic

a	b	c	$\neg c$	$(b \vee \neg c)$	$a \wedge (b \vee \neg c)$
$\mathbf{f}$	$\mathbf{f}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{f}$
$\mathbf{f}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{f}$	$\mathbf{f}$	$\mathbf{f}$
$\mathbf{f}$	$\mathbf{t}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{f}$
$\mathbf{f}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{f}$
$\mathbf{t}$	$\mathbf{f}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}$
$\mathbf{t}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{f}$	$\mathbf{f}$	$\mathbf{f}$
$\mathbf{t}$	$\mathbf{t}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}$
$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{t}$

Figure 2.3: Truth table for the formula $a \wedge (b \vee \neg c)$

The basic connectives of Definition 2.2.14 are also illustrated in the Table 2.2. These tables are usually referred to as "Truth Tables". These truth tables can be straightforwardly generalized to arbitrary formulae, see Figure 2.3, where we see the example formula $a \wedge (b \vee \neg c)$ and all its subformulae. Next we define important concepts of a two-valued interpretation I w.r.t. a formula the interpretation is defined upon and the formula itself.

Definition 2.2.15. *Let ϕ be a formula in $\mathcal{QBF}$ and I a two-valued interpretation on ϕ. Then*

- *I* satisfies *ϕ, denoted as $I \models \phi$, iff $I(\phi) = \mathbf{t}$, I is said to be a* model *of ϕ,*
- *ϕ is* satisfiable *iff there exists an interpretation I on ϕ with $I \models \phi$,*
- *ϕ is* valid *or a tautology, denoted as $\models \phi$ iff all interpretations I on ϕ are models of ϕ.*

Note that in case a formula ϕ is closed, then ϕ is either valid or unsatisfiable, i.e. not satisfiable. We also then do not require any truth assignments of an interpretation to decide this, since a two-valued interpretation I on ϕ has the empty domain $dom(I) = \emptyset$ and evaluates to either true or false. Further one can say that a formula ψ is semantically *entailed* from a formula ϕ, which we denote as $\phi \models \psi$ which holds iff for every interpretation I defined on $atoms(\phi) \cup atoms(\psi)$ if $I \models \phi$ then also $I \models \psi$. In classical propositional logic this means that $\phi \models \psi$ iff $\models \phi \rightarrow \psi$.

Example 2.2.9. *Consider the formulae $\phi = p \wedge q$ and $\psi = p \vee \top$ and let I be an interpretation on ϕ such that $I(p) = \mathbf{t}$ and $I(q) = \mathbf{t}$. Then $I(\phi) = \mathbf{t}$ and thus ϕ is satisfiable. The formula ψ is valid, since both interpretations J and J' on ψ with $J(p) = \mathbf{t}$ and $J'(p) = \mathbf{f}$ evaluate ψ to $\mathbf{t}$. This can be seen because any interpretation evaluates $\top$ to true and hence any interpretation on χ evaluates under $\chi \vee \top$ to true for any subformula χ. Finally ψ is semantically entailed by ϕ, i.e. $\phi \models \psi$.*

Having finally defined the semantics of classical propositional logic, we can state some further important properties, which we will utilize in our work. Two formulae ϕ and χ are semantically equivalent, denoted as $\phi \equiv \chi$ iff $\models \phi \leftrightarrow \chi$, i.e. the formula $\phi \leftrightarrow \chi$ is valid. This means that any model of ϕ is also a model of χ. Next, as briefly mentioned in the Section 2.2.3 about normal forms, we can actually rewrite any propositional formula ϕ to a χ in CNF and still have that $\phi \equiv \chi$.

The translation of an arbitrary propositional formula to a formula in CNF is achievable via several equivalences or also sometimes called laws. First consider a formula ϕ and a subformula ψ_1 of ϕ. If $\psi_1 \equiv \psi_2$, then ϕ is equivalent to χ which is obtained by replacing every occurrence of ψ_1 with ψ_2 in ϕ. Formally we use the substitution function, i.e. by $\phi[\psi_1/\psi_2]$ we mean that every occurrence of ψ_1 is replaced by ψ_2. The following theorem then holds.

Theorem 2.2.1. *Let ϕ, ψ_1, ψ_2 be propositional formulae. If $\psi_1 \equiv \psi_2$ then it holds that $\phi \equiv \phi[\psi_1/\psi_2]$.*

This enables us to replace occurrences of subformulae of a formula with equivalent formulae and still stay semantically equivalent. This is exploited in the transformation of a formula to CNF. The laws for this transformation are based on the following equivalences in classical propositional logic.

Proposition 2.2.2. *Let ϕ, ψ, χ be propositional formulae. Then the following equivalences hold.*

1. $(\phi \circ (\psi \circ \chi)) \equiv ((\phi \circ \psi) \circ \chi)$ *for* $\circ \in \{\wedge, \vee, \leftrightarrow\}$; *(associativity)*
2. $(\phi \leftrightarrow \psi) \equiv ((\phi \rightarrow \psi) \wedge (\phi \leftarrow \psi))$;
3. $(\phi \rightarrow \psi) \equiv (\neg\phi \vee \psi)$;
4. $(\phi \leftarrow \psi) \equiv (\phi \vee \neg\psi)$;
5. $\neg(\phi \wedge \psi) \equiv (\neg\phi \vee \neg\psi)$; *(De Morgan)*
6. $\neg(\phi \vee \psi) \equiv (\neg\phi \wedge \neg\psi)$; *(De Morgan)*
7. $(\phi \circ \psi) \equiv (\psi \circ \phi)$ *for* $\circ \in \{\wedge, \vee, \leftrightarrow\}$; *(commutativity)*
8. $(\phi \wedge \top) \equiv \phi$;
9. $(\phi \wedge \bot) \equiv \bot$;
10. $(\phi \vee \top) \equiv \top$;
11. $(\phi \vee \bot) \equiv \phi$;
12. $(\neg\neg\phi) \equiv \phi$; *(elimination of double negation)*
13. $(\phi \wedge (\psi \vee \chi)) \equiv ((\phi \wedge \psi) \vee (\phi \wedge \chi))$; *(distribution of conjunction over disjunction)*
14. $(\phi \vee (\psi \wedge \chi)) \equiv ((\phi \vee \psi) \wedge (\phi \vee \chi))$. *(distribution of disjunction over conjunction)*

We can use these equivalences for successively rewriting a formula to a formula in CNF. Essentially these equivalences can be read as "rewriting rules", e.g. if we encounter a subformula of the form shown on the left, we rewrite all these subformulae to a form shown on the right. If we apply these exhaustively, except for the associativity and commutativity rules, then the resulting formula is in CNF. This rewriting in general may lead to a drastically larger formula in CNF. For this purpose Tseitin [149] proposed another transformation, which transforms a formula ϕ to χ, such that ϕ is satisfiable iff χ is satisfiable, but we in general do not have that $\phi \equiv \chi$. The benefit of this translation is that the size of χ, i.e. the number of symbols in the formula, does not increase exponentially w.r.t. the size of ϕ in the worst case, as is the case with standard translation, but is only polynomial with the Tseitin translation. Regarding QBFs we can rewrite every QBF ϕ to a QBF χ such that χ is in PNF and $\phi \equiv \chi$.

Example 2.2.10. *Let $\phi = \neg(a \vee b) \vee \neg c$. Then we can transform ϕ to $\chi_1 = (\neg a \wedge \neg b) \vee \neg c$ and in a second step to $\chi_2 = (\neg a \vee \neg c) \wedge (\neg b \vee \neg c)$, which is in CNF and $\phi \equiv \chi_1 \equiv \chi_2$.*

The generalization of two-valued classical logic to three-valued logic is a major step. For this *non-classical* logic we define the concepts relevant for our work. In particular we define the semantics of this three-valued logic on formulae without quantifiers. The semantics differs in such that the now three-valued interpretations must evaluate accordingly to the truth value $\mathbf{u}$. It is worth noting that many other non-classical logics exists, which generalize two-valued classical logic or are developed w.r.t. a very different goal.

Definition 2.2.16. *Let $\phi \in \mathcal{WFF}$, ψ and χ be subformulae of ϕ and I a three-valued interpretation on ϕ. The* evaluation *of ϕ under I, denoted as $I(\phi)$ is defined recursively as follows.*

- *if $\phi \in \mathcal{P}$, then $I(\phi) = I(p)$,*
- *if $\phi = \top$, then $I(\phi) = \mathbf{t}$,*
- *if $\phi = \bot$, then $I(\phi) = \mathbf{f}$,*
- *if $\phi = \neg\psi$, then* $I(\phi) = \begin{cases} \mathbf{t} \text{ if } I(\psi) = \mathbf{f} \\ \mathbf{f} \text{ if } I(\psi) = \mathbf{t} \\ \mathbf{u} \text{ else} \end{cases}$
- *if $\phi = \psi \wedge \chi$, then* $I(\phi) = \begin{cases} \mathbf{t} \text{ if } I(\psi) = \mathbf{t} \text{ and } I(\chi) = \mathbf{t} \\ \mathbf{f} \text{ if } I(\psi) = \mathbf{f} \text{ or } I(\chi) = \mathbf{f} \\ \mathbf{u} \text{ else} \end{cases}$
- *if $\phi = \psi \vee \chi$, then* $I(\phi) = \begin{cases} \mathbf{t} \text{ if } I(\psi) = \mathbf{t} \text{ or } I(\chi) = \mathbf{t} \\ \mathbf{f} \text{ if } I(\psi) = \mathbf{f} \text{ and } I(\chi) = \mathbf{f} \\ \mathbf{u} \text{ else} \end{cases}$
- *if $\phi = \psi \rightarrow \chi$, then* $I(\phi) = \begin{cases} \mathbf{t} \text{ if } I(\psi) = \mathbf{f} \text{ or } I(\chi) = \mathbf{t} \\ \mathbf{f} \text{ if } I(\psi) = \mathbf{t} \text{ and } I(\chi) = \mathbf{f} \\ \mathbf{u} \text{ else} \end{cases}$

$\neg$	
t	**f**
f	**t**
u	**u**

$\vee$	**t**	**u**	**f**
t	**t**	**t**	**t**
u	**t**	**u**	**u**
f	**t**	**u**	**f**

$\wedge$	**t**	**u**	**f**
t	**t**	**u**	**f**
u	**u**	**u**	**f**
f	**f**	**f**	**f**

$\rightarrow$	**t**	**u**	**f**
t	**t**	**u**	**f**
u	**t**	**u**	**u**
f	**t**	**t**	**t**

$\leftrightarrow$	**t**	**u**	**f**
t	**t**	**u**	**f**
u	**u**	**u**	**u**
f	**f**	**u**	**t**

Figure 2.4: Truth tables for strong three–valued logic of Kleene

- *if* $\phi = \psi \leftarrow \chi$, *then* $I(\phi) = \begin{cases} \mathbf{t} \textit{ if } I(\psi) = \mathbf{t} \textit{ or } I(\chi) = \mathbf{f} \\ \mathbf{f} \textit{ if } I(\psi) = \mathbf{f} \textit{ and } I(\chi) = \mathbf{t} \\ \mathbf{u} \textit{ else} \end{cases}$

- *if* $\phi = \psi \leftrightarrow \chi$, *then* $I(\phi) = \begin{cases} \mathbf{t} \textit{ if } I(\psi) = I(\chi) \neq \mathbf{u} \\ \mathbf{f} \textit{ if } \mathbf{u} \neq I(\psi) \neq I(\chi) \neq \mathbf{u} \\ \mathbf{u} \textit{ else} \end{cases}$

Note that we have a distinct symbol for absolute truth $\top$ and for absolute falsity $\bot$, but no distinguished symbol denoting the undecided truth value. Again one can define truth tables, see Table 2.4. Typically a model of a formula ϕ in the context of three-valued semantics can be defined as an interpretation I, s.t. ϕ evaluates to true or undecided under I. Models of three-valued propositional logic are not crucial for our work.

2.3 Argumentation in Artificial Intelligence

In this section we introduce the formal languages in argumentation [18, 22, 139] we study in this work. In line with formal languages (like propositional logic), the frameworks we study here have a structure and semantics. In the following we introduce first the well-known concepts of Argumentation Frameworks (AFs) [59] and then the more general Abstract Dialectical Frameworks (ADFs) [34] and respectively their semantics.

2.3.1 Argumentation Frameworks

Argumentation frameworks as introduced by Dung [59] are structures consisting of a set of abstract arguments and a binary relation on this set. Similarly as in propositional logic we assume a universe of arguments $\mathcal{P}$ from which we take a subset as our currently relevant set of arguments for our discourse.[1]

[1] We re-use the symbol for the universe of arguments the symbol for the universe of atoms in propositional logic, indeed we assume that each argument in an AF can also be used as a propositional atom in a formula and vice versa. This is due to technical reasons for simplifying our translations between the formalisms, but does not imply further connections between them as e.g. in a logical sense we consider arguments as defeasible objects, i.e. argumentation as defined here is a non-monotonic formalism in contrast to propositional logic, which is a classical monotonic formalism.

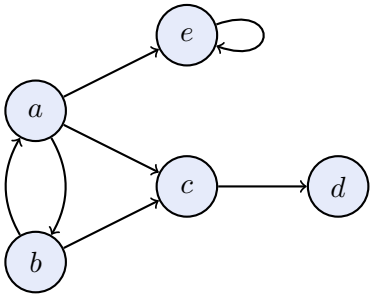

Figure 2.5: Example argumentation framework F

Definition 2.3.1. *An argumentation framework (AF) is a pair $F = (A, R)$ where $A \subseteq \mathcal{P}$ is a set of arguments and $R \subseteq A \times A$ is the attack relation. The pair $(a, b) \in R$ means that a attacks b. We denote by $\mathcal{AF}_{\mathcal{P}}$ the set of all AFs over $\mathcal{P}$.*

An argumentation framework $F = (A, R)$ can be represented as a directed graph, with A being the vertices and R the directed edges, as shown in the following example.

Example 2.3.1. *Let $F = (A, R)$ be an AF with $A = \{a, b, c, d, e\}$ and $R = \{(a, b), (b, a), (a, c), (a, e), (b, c), (c, d), (e, e)\}$. The corresponding graph representation is depicted in Figure 2.5.*

We use some graph-theoretic notions, which can also be applied to AFs. The directed graph of an AF $F = (A, R)$ is simply F itself.[2] The underlying *undirected* graph of F is $F' = (A, R')$, where R' is the undirected version of the binary relation R. For a graph $G = (V, E)$ a *path* in G is a sequence of edges in G, i.e. $P = (e_1, \ldots, e_n)$ with $\{e_1, \ldots, e_n\} \subseteq E$ and for all $1 \leq i < n$ with $e_i = (x_i, y_i)$ and $e_{i+1} = (x_{i+1}, y_{i+1})$ we have $y_i = x_{i+1}$. We say that this is a path from x_1 to y_n for $e_1 = (x_1, y_1)$ and $e_n = (x_n, y_n)$. For any vertex in a (directed) graph there is a path of length 0 to itself. A *connected component* of an undirected graph G is a maximal subset of the vertices $S \subseteq V$ such that for all $x, y \in S$ we have that there exists a path from x to y in G. For a directed graph $G = (V, E)$ a *strongly connected component* in G is a maximal subset of the vertices $S \subseteq V$ such that for all $x, y \in S$ we have that a path from x to y and from y to x exists in G. Note that every (directed) graph has (strongly) connected components. In particular a graph $G = (\{x\}, \emptyset)$ containing no edges and a single vertex has a (strongly) connected component, the singleton set $\{x\}$. A directed graph $G = (V, E)$ is a *directed acyclic graph (DAG)* if there does not exist a cycle in G, i.e. not two $x, y \in V$ such that there is a path from x to y and back to x of length greater than 0.

Given a directed graph $G = (V, E)$ we can look at the set of strongly connected components, i.e. $S = \{S_1, \ldots, S_n\}$ the set of strongly connected components of G with $S_1 \cup \cdots \cup S_n = V$ and each $S', S'' \in S$ are pairwise disjoint. The *condensation* of this graph G is a new directed graph $G' = (S, E')$ with $E' = \{(S_i, S_j) \mid x \in S_i, y \in S_j, S_i \in S, S_j \in S$ and there exists a path from x to y in $G\}$. The condensation of a directed graph is a DAG.

[2] We allow self-loops in (directed) graphs.

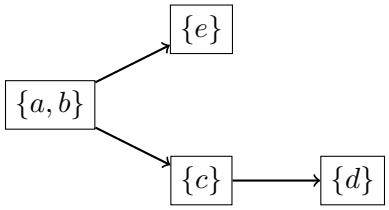

Figure 2.6: Condensation of AF from Example 2.3.1

Furthermore we define $SCCs(G) = S$, i.e. this is the set of strongly connected components of the directed graph G. All these definitions for directed graphs are naturally applicable for AFs.

Example 2.3.2. *Let us look at the AF shown in Figure 2.3.1. There is a path from b to e with $P = ((b, a), (a, e))$. Every vertex itself is a strongly connected component. The set of maximal strongly connected components is given by $SCCs(F) = \{\{a, b\}, \{c\}, \{d\}, \{e\}\}$. The condensation is then shown in Figure 2.6. Here we "collapse" the set of nodes $\{a, b\}$ to a single one. This directed graph is a DAG.*

We introduce the following useful notion of subframeworks of a given AF.

Definition 2.3.2. *Let $F = (A, R)$ be an AF. For a set $S \subseteq A$ we define $F|_S = (S, \{(x, y) \in R \mid \{x, y\} \subseteq S\})$.*

2.3.2 Semantics of Argumentation Frameworks

Semantics of AFs, similarly as for propositional logic, can be seen as assignments of the arguments to a certain status. The terminology in argumentation theory uses *extensions* instead of interpretations, which are sets of arguments that are jointly acceptable w.r.t. some criteria. While in propositional logic we have a set of models for a given formula, on AFs one has a set of extensions. More formally a semantics for argumentation frameworks is given via a function $\sigma : \mathcal{AF}_{\mathcal{P}} \to 2^{2^{\mathcal{P}}}$ which assigns to each AF $F = (A, R)$ a set $\sigma(F) \subseteq 2^A$ of extensions. In contrast to classical propositional logic, several semantics have been defined for AFs. Each of them takes a different point of view what should be accepted or rejected, i.e. what may be in an extension. The symbol σ is used as a *generic* function for the semantics of AFs.

Naturally one can assign two-valued interpretations instead of extensions by setting everything to true which inside an extension and to false otherwise. Indeed such a point of view is also present in argumentation theory with the concept of so-called labelings [42]. A labeling for an AF $F = (A, R)$ is a function $lab : A \to \{in, out, undecided\}$, which assigns to each argument a *status*. This status can be in, out or $undecided$, which situates this concept close to three-valued logic. We will sometimes say that an argument inside an extension is *accepted* w.r.t. that extension.

We now define the most prominent semantics in abstract argumentation on AFs using the extensional point of view. For σ we consider the functions cf, adm, com, grd, prf, stb, sem, stg, $ideal$, $eager$, $stg\text{-}idl$, and $stg2$, which stand for conflict-free, admissible, complete, grounded,

preferred, stable, semi-stable, stage, ideal, eager, stage-ideal, and stage2 extensions, respectively. Conflict-free, admissible, complete, grounded, preferred and stable semantics have been introduced in [59], semi-stable semantics in [37, 41], stage in [151], ideal in [60], eager in [39], stage-ideal in [67] and stage2 in [72]. The semantics for the three-valued labelings can be defined accordingly and we exemplarily show the corresponding definition for the complete semantics.

The basic concept for many of the semantics considered in this work is the admissible extension or just admissible set. Admissibility has two requirements, namely conflict-freeness and defense of all arguments in the set.

Definition 2.3.3. *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is conflict-free in F, if there are no $a, b \in S$, such that $(a, b) \in R$. We say that an argument $a \in A$ is defended by a set $S \subseteq A$ in F if, for each $b \in A$ such that $(b, a) \in R$, there exists a $c \in S$ such that $(c, b) \in R$.*

Formally an argument a is defended by a set of arguments if all attackers of a are counter-attacked by arguments in the set. If for instance we accept the set of arguments then, intuitively speaking, we disagree or reject all attackers of a, since they are attacked. In turn it is reasonable in this framework to view the argument a as acceptable, too. Admissible sets are then conflict-free sets of arguments, where each argument in the set is defended by the set.

Definition 2.3.4. *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is admissible in F, if*

- *S is conflict-free in F; and*
- *each $a \in S$ is defended by S in F.*

Complete extensions are admissible sets which contain every argument they defend, i.e. they are complete in the sense that every defended argument is also accepted, that is, in the set.

Definition 2.3.5. *Let $F = (A, R)$ be an AF. An admissible set $S \subseteq A$ in F is a complete extension in F, if every $s \in A$ which is defended by S in F is in S.*

There is a useful equivalent definition of defense and the concepts of admissibility and complete extensions. It is based on the characteristic function of an AF. Given a set of arguments, this function returns every argument defended by this set.

Definition 2.3.6. *Let $F = (A, R)$ be an AF and $S \subseteq A$. Then $\mathcal{F}_F : 2^A \to 2^A$ is the characteristic function of F and is defined by $\mathcal{F}_F(S) = \{a \mid a \text{ is defended by } S\}$.*

If we consider for an AF $F = (A, R)$ the partially ordered set $(2^A, \subseteq)$, then $\mathcal{F}_F$ is $\subseteq$-monotone.

Lemma 2.3.1 ([59, Lemma 19])**.** *Let $F = (A, R)$ be an AF. $\mathcal{F}_F$ is $\subseteq$-monotone.*

Using this function we can define the admissible sets and complete extensions as follows.

Proposition 2.3.2 ([59, Lemma 18 and Lemma 24])**.** *Let $F = (A, R)$ be an AF. For a conflict-free set $S \in cf(F)$, it holds that*

- *$S \in adm(F)$ iff $S \subseteq \mathcal{F}_F(S)$;*

- $S \in com(F)$ *iff* $S = \mathcal{F}_F(S)$.

The minimal complete extension, w.r.t. subset-inclusion is referred to as the grounded extension.

Definition 2.3.7. *Let $F = (A, R)$ be an AF. A complete extension $S \subseteq A$ in F is the grounded extension in F, if there is no complete extension $S' \subseteq A$ in F such that $S' \subsetneq S$.*

The grounded extension is also the least fixed point of the characteristic function and thus is unique.

Proposition 2.3.3. *Let $F = (A, R)$ be an AF. The grounded extension $S \in grd(F)$ is the least fixed point of $\mathcal{F}_F$.*

Maximal admissible sets, w.r.t. subset-inclusion are called preferred extensions and accept as many arguments as possible, without violating admissibility.

Definition 2.3.8. *Let $F = (A, R)$ be an AF. An admissible set $S \subseteq A$ in F is a preferred extension in F, if there is no admissible set $S' \subseteq A$ in F such that $S \subsetneq S'$.*

Note that we could equivalently define the preferred extension to be maximal complete extensions w.r.t. subset-inclusion. A basic property of the preferred semantics is that admissible sets, complete extensions and hence preferred extensions and the grounded extension always exist for any given framework. This follows in particular due to the fact that the empty set is always admissible in any AF. A popular semantics for which this is not the case is the stable semantics. For the definition of the stable semantics and the closely related semi-stable and stage semantics we make use of the concept of the range of a given set S of arguments, which is simply the set itself and everything it attacks,

Definition 2.3.9. *Let $F = (A, R)$ be an AF and $S \subseteq A$. The range of S (in R), denoted by S^+_R, is given by $S^+_R = S \cup \{a \mid (b, a) \in R, b \in S\}$.*

Then a set is stable if it is conflict-free and attacks everything not contained in it, i.e. its range covers all arguments in the framework. Semi-stable extensions are admissible sets which have a subset maximal range, while stage extensions are conflict-free sets with subset maximal range.

Definition 2.3.10. *Let $F = (A, R)$ be an AF. A conflict-free set $S \subseteq A$ in F is a stable extension in F, if $S^+_R = A$. An admissible set E in F is a semi-stable extension in F if there does not exist a set T admissible in F, with $E^+_R \subset T^+_R$. A conflict-free set E in F is a stage extension in F if there does not exist a set T conflict-free in F, with $E^+_R \subset T^+_R$.*

A basic property of semi-stable and stage semantics is that if an AF has stable extensions, then the semi-stable, stage and stable semantics coincide [41]. The intuition is that semi-stable and stage extensions should be "close" to stable extensions, in case no stable extensions exist.

Regarding the aforementioned labeling approach, we can use a three-valued interpretation for defining complete semantics in the following manner [42].[3]

[3]Our definition diverges from the original in [42] by identifying $in = \mathbf{t}$, $out = \mathbf{f}$ and $undec = \mathbf{u}$.

Definition 2.3.11. *Let $F = (A, R)$ be an AF. A function $I : A \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ is a complete labeling in F iff the following conditions hold for each $a \in A$.*

- $I(a) = \mathbf{t}$ *iff for each b with $(b, a) \in R$ we have $I(b) = \mathbf{f}$; and*
- $I(a) = \mathbf{f}$ *iff there exists a b with $(b, a) \in R$ we have $I(b) = \mathbf{t}$*

Other semantics we defined here can likewise be defined using labelings. We further have the following correspondence.

Proposition 2.3.4 ([42, Theorem 9 and Theorem 10])**.** *Let $F = (A, R)$ be an AF. It holds that $com(F) = \{I^{\mathbf{t}} \mid I$ a complete labeling in $F\}$.*

That is, we can immediately infer a complete labeling from a complete extension and vice versa. Given an AF $F = (A, R)$ and $E \in com(F)$ and $E^+ = E_R^+ \setminus E$, then the following is a complete labeling: $(E, E^+, A \setminus (E_R^+))$. The other direction is is even more simpler, given a three-valued complete labeling I of F, then $I^{\mathbf{t}}$ is a complete extension.

Example 2.3.3. *Consider the AF F from Example 2.3.1. Then we have the following conflict-free sets, admissible sets and extensions of the semantics:*

- $cf(F) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, d\}, \{b, d\}\}$;
- $adm(F) = \{\emptyset, \{a\}, \{b\}, \{a, d\}, \{b, d\}\}$;
- $com(F) = \{\emptyset, \{a, d\}, \{b, d\}\}$;
- $grd(F) = \{\emptyset\}$;
- $stb(F) = \{\{a, d\}\}$;
- $prf(F) = \{\{a, d\}, \{b, d\}\}$;
- $sem(F) = \{\{a, d\}\}$; *and*
- $stg(F) = \{\{a, d\}\}$.

The complete labelings of F are given by $\{(\emptyset, \emptyset, \{a, b, c, d, e\})$, $(\{a, d\}, \{b, c, e\}, \emptyset)$, $(\{b, d\}, \{a, b, e\}, \emptyset)\}$. Here we see that the empty extension is the grounded extension of the framework and corresponds to $\{(\emptyset, \emptyset, \{a, b, c, d, e\})$, i.e. setting all arguments to undecided.

Note that if we would add a single isolated self-attacking argument to F, i.e. $F' = (A', R')$ with $A' = A \cup \{f\}$ and $R' = R \cup \{(f, f)\}$, shown in Figure 2.7, then $stb(F') = \emptyset$, but the set of semi-stable and stage extensions would remain the same, i.e. $sem(F') = sem(F)$ and $stg(F) = stg(F')$.

Although this may appear as a deficiency of stable semantics, we can see on the AF F an interesting concept, namely that the preferred extension $\{b, d\}$ is not *stable, because it does not attack the self-attacking argument e. Such a self-attacking argument ensures that any stable extension necessarily must accept in the extension an attacker of it. This behavior can be used for modeling certain situations, e.g. if we require that in any stable extension one argument out*

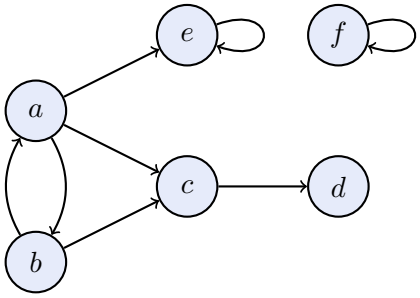

Figure 2.7: Argumentation framework F' containing an isolated self-attacking argument

of a set of arguments is accepted, we can ensure this by simply adding a self-attacking argument, which is attacked by all arguments in this set. On the other hand if it is unwarranted that e.g. by adding a single isolated self-attacking argument to an AF we lose all stable extensions, then the semi-stable or stage semantics can be applied to remedy this behavior.

Notice that all the semantics introduced until now, except the grounded semantics, may have multiple extensions. Semantics specifying a unique extension are called unique-status semantics. If we have multiple extensions for a semantics, then an argument can be once inside an extension and once outside. Reasoning tasks on AFs w.r.t. a semantics σ, apart from simple enumeration of all extensions, include the credulous and skeptical acceptance of arguments. An argument is credulously (skeptically) accepted for a semantics and an AF, if it is present in at least one extension (in all extensions) of the semantics.[4] We call a non-empty extension non-trivial.

Definition 2.3.12. *Given an AF $F = (A, R)$, a semantics σ, $S \subseteq A$ and an argument $a \in A$ then we define the following reasoning tasks.*

- $\mathsf{Enum}_\sigma(F) = \sigma(F)$
- $\mathsf{Cred}_\sigma(a, F) = \begin{cases} \textit{yes} & \textit{if } a \in \bigcup \sigma(F) \\ \textit{no} & \textit{otherwise} \end{cases}$
- $\mathsf{Skept}_\sigma(a, F) = \begin{cases} \textit{yes} & \textit{if } a \in \bigcap \sigma(F) \\ \textit{no} & \textit{otherwise} \end{cases}$
- $\mathsf{AllCred}_\sigma(F) = \bigcup \sigma(F)$;
- $\mathsf{AllSkept}_\sigma(F) = \bigcap \sigma(F)$;
- $\mathsf{Ver}_\sigma(S, F) = \begin{cases} \textit{yes} & \textit{if } S \in \sigma(F) \\ \textit{no} & \textit{otherwise} \end{cases}$

[4]The labeling-based approach (i.e. assigning acceptance, rejection or undecidedness to arguments) can be used for a more fine-grained notion of acceptance of arguments [157, 70]. E.g. one may ask if an argument is accepted in at least one complete labeling, but never rejected in a complete labeling.

Example 2.3.4. *Applying the reasoning tasks to the AF F in Example 2.3.1, we have for the preferred semantics the following credulously and skeptically accepted arguments. Credulously accepted are the arguments a, b and d, i.e.* $\mathsf{AllCred}_{prf}(F) = \{a, b, d\}$ *and* $\mathsf{AllSkept}_{prf}(F) = \{d\}$. *The set $S = \{e\}$ is not a preferred extension in F, hence* $\mathsf{Ver}_{prf}(S, F) = no$.

Three further unique-status semantics we study in this work are the ideal, eager and stage-ideal semantics, which take a particular skeptical stance, similarly as the grounded semantics.

Definition 2.3.13. *Let $F = (A, R)$ be an AF. For an admissible set $S \in adm(F)$, it holds that*

- $S \in ideal(F)$, *if* $S \subseteq \mathsf{AllSkept}_{prf}(F)$ *and there is no* $T \in adm(F)$ *with* $S \subset T \subseteq \mathsf{AllSkept}_{prf}(F)$;
- $S \in eager(F)$, *if* $S \subseteq \mathsf{AllSkept}_{sem}(F)$ *and there is no* $T \in adm(F)$ *with* $S \subset T \subseteq \mathsf{AllSkept}_{sem}(F)$.
- $S \in stg\text{-}idl(F)$, *if* $S \subseteq \mathsf{AllSkept}_{stg}(F)$ *and there is no* $T \in adm(F)$ *with* $S \subset T \subseteq \mathsf{AllSkept}_{stg}(F)$.

That is, the ideal, eager and stage-ideal extensions are the maximal-admissible sets w.r.t. subset-inclusion, composed only of arguments skeptically accepted under preferred, semi-stable and stage semantics, respectively.

Example 2.3.5. *Continuing the Example 2.3.3, based on the AF in Example 2.3.1, then we have that* $ideal(F) = \{\emptyset\}$; *and* $eager(F) = \{\{a, d\}\} = stg\text{-}idl(F)$. *Note that although* $\mathsf{AllSkept}_{prf}(F) = \{d\}$, *the set $\{d\}$ is not admissible in F.*

A recently developed semantics is the stage2 semantics [72, 97], denoted by the function $stg2$. It is one of the SCC-recursive semantics [12]. Towards the definition we need one technical construct, namely the set of *component-defeated* arguments of a set S.

Definition 2.3.14. *Let $F = (A, R)$ be an AF, $x \in A$ and $S \subseteq A$. We define the strongly connected component of x in F with the function $C_F : A \to SCCs(F)$, s.t. $C_F(x) = C_i$ if $C_i \in SCCs(F)$ and $x \in C_i$. A $b \in A$ is component-defeated by S in F if there exists an $a \in S$, s.t. $(a, b) \in R$ and $a \notin C_F(b)$. The set of arguments component-defeated by S in F is denoted by $D_F(S)$.*

Using this we define the $stg2$ semantics as follows.

Definition 2.3.15. *Let $F = (A, R)$ be an AF and $S \subseteq A$. Then $S \in stg2(F)$ if,*

- $|SCCs(F)| = 1$ *and* $S \in stg(F)$; *or*
- $|SCCs(F)| > 1$ *and for each* $C \in SCCs(F)$ *we have* $(S \cap C) \in stg2(F|_{C \setminus D_F(S)})$.

Given the set of skeptically accepted arguments w.r.t. preferred, semi-stable or stage semantics to compute the unique subset-maximal admissible set composed only of the arguments skeptically accepted, we can make use of the following function, which we call restricted characteristic function [62].

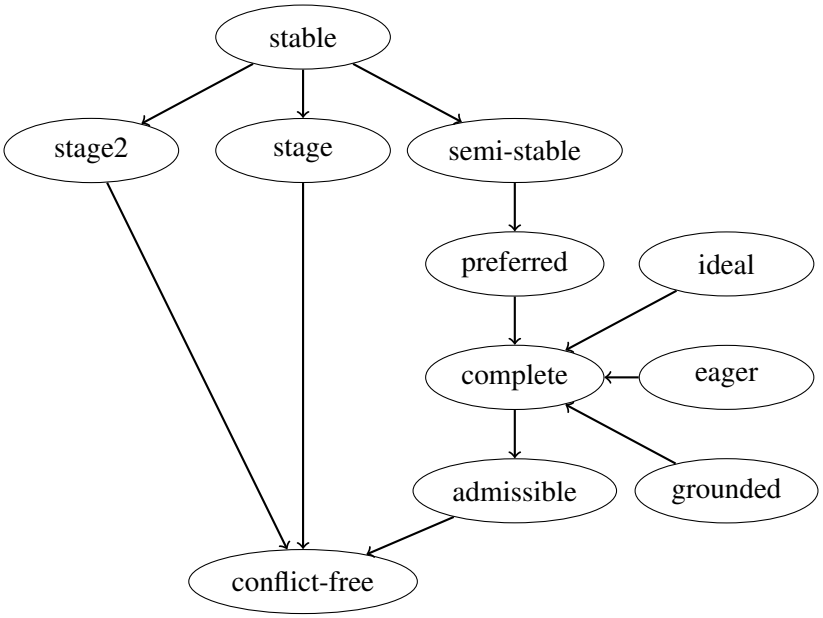

Figure 2.8: Relation between AF semantics

Definition 2.3.16. *Let $F = (A, R)$ be an AF. Then $\hat{\mathcal{F}}_F : 2^A \to 2^A$ is the restricted characteristic function of F and is defined by $\hat{\mathcal{F}}_F(S) = \{a \in S \mid a \text{ is defended by } S\}$.*

The difference of this restricted characteristic function to the characteristic function lies in, as the name suggests, that we restrict the result to be a subset of the set of arguments given, i.e. the restricted characteristic function is equivalent to $\hat{\mathcal{F}}_F(S) = \mathcal{F}_F(S) \cap S$ and we have $S \supseteq \hat{\mathcal{F}}_F(S)$ for each $S \subseteq A$.

In general none of the semantics defined here on AFs coincide. Each of these semantics is defined via different criteria and has a different purpose. In Figure 2.8 we see the relation of the semantics such that an arrow from semantics σ to σ' denotes that for any AF F we have $\sigma(F) \subseteq \sigma'(F)$, which was shown in [7, 8, 10, 41, 39, 59, 60, 72, 151]. If such a subset relation is present we denote this formally by saying that σ preserves σ' with the following definition.

Definition 2.3.17. *Let σ and σ' be semantics on AFs. We say σ is σ'-preserving if for each AF F we have $\sigma(F) \subseteq \sigma'(F)$.*

In [9] the semantics are studied w.r.t. so-called evaluation criteria, that is semantical properties. We here recall those important for our work.

A set of arguments $S \subseteq A$ is *unattacked* in an AF $F = (A, R)$ if for each $a \in A \setminus S$ there is no $b \in S$ such that $(a, b) \in R$.

Definition 2.3.18. *A semantics σ on AFs satisfies*

- *I-maximality if for each AF F and each $S_1, S_2 \in \sigma(F)$, we have if $S_1 \subseteq S_2$ then $S_1 = S_2$;*

	com	grd	prf	stb	sem	stg	$stg2$	$ideal$	$eager$	$stg\text{-}idl$
I-max	no	yes	yes	yes	yes	yes	yes	yes	yes	yes
Reinst.	yes	yes	yes	yes	yes	no	no	yes	yes	no
w-Reinst.	yes	yes	yes	yes	yes	no	yes	yes	yes	no
direct.	yes	yes	yes	no	no	no	yes	yes	no	no

Table 2.1: Evaluation criteria for AF semantics

- *reinstatement if for each AF F and each $S \in \sigma(F)$, we have if a is defended by S, then $a \in S$;*
- *weak reinstatement if for each AF F and each $S \in \sigma(F)$ we have $grd(F) \subseteq S$; and*
- *directionality if for each AF F and each set of unattacked arguments $U \subseteq A$ we have $\sigma(F|_U) = \{S \cap U \mid S \in \sigma(F)\}$.*

In Table 2.1 we summarize which semantics satisfies which of the evaluation criteria. The results were shown in [7, 9, 72], except for the eager and $stg\text{-}idl$ semantics. For the eager semantics the properties were not directly shown, but can be quickly inferred: I-maximality follows from the unique-status result [39, Theorem 3], the reinstatement follows since every eager extension is complete [39, Theorem 4] and weak reinstatement follows from [39, Theorem 5]. To see that directionality does not hold for eager semantics, just consider Example 2.3.3, where we have that $eager(F) = \{\{a, d\}\}$, but for the restricted framework with the set of unattacked arguments $U = \{a, b, c, d\}$ we can construct $F|_U = F''$ and we have $sem(F'') = \{\{a, d\}, \{b, d\}\} = prf(F'') = stb(F'')$ and thus $eager(F'') = \{\emptyset\} \neq eager(F)$. Since $stg\text{-}idl$ is among the unique-status semantics [67, Proposition 1] it follows that it also enjoys I-maximality. Regarding the remaining properties for $stg\text{-}idl$ the answer is negative and witnessed by Example 2.3.6.

Example 2.3.6. *Let $D = (\{a, b, c\}, \{(a, b), (b, c), (c, c)\})$ be an AF. Then we have $stg(D) = \{\{a\}, \{b\}\}$ and thus $stg\text{-}idl(D) = \emptyset$. The grounded extension is given by $grd(D) = \{a\}$ and thus weak-reinstatement does not hold for $stg\text{-}idl$. Neither does reinstatement, since $\emptyset$ defends in D the set $\{a\}$. Consider $D|_{\{a\}} = (\{a\}, \emptyset)$, then we have $stg\text{-}idl(D|_{\{a\}}) = \{\{a\}\}$ and thus directionality does not hold for $stg\text{-}idl$.*

2.3.3 Abstract Dialectical Frameworks

Arguments in AFs are inherently abstract entities, they may represent any object of discourse. The *relation* between arguments is fixed though, namely it represents the attack relation. In the recently developed abstract dialectical frameworks (ADFs) the relation between arguments is also abstract in the sense that various types of such relations are allowed.

In AFs the acceptance of an argument depends on the acceptance of its attackers and defenders, or in other words the *rejection* of its attackers. In ADFs the acceptance/rejection of an

argument s likewise depends on the status of a set of arguments, here called the *parents* and denoted by $par(s)$, but instead of specifying that s is acceptable if all its parents are rejected, in ADFs *any* combination of accepted/rejected parents may lead to acceptance of s. This leads to the notion of an *acceptance condition*, which relates the status of the parents of an argument to either acceptance or rejection. Again as for argumentation frameworks we take our set of arguments from $\mathcal{P}$.

Definition 2.3.19. *An abstract dialectical framework (ADF) is a tuple (S, L, C), where*

- *$S \subseteq \mathcal{P}$ is a set of abstract arguments,*
- *$L \subseteq S \times S$ is a set of links and*
- *$C = \{C_s\}_{s \in S}$ is a set of functions $C_s : 2^{par(s)} \to \{\mathbf{t}, \mathbf{f}\}$. C_s is called acceptance condition of s.*

We denote by $\mathcal{ADF}_{\mathcal{P}}$ the set of all ADFs over $\mathcal{P}$.

The set of parents is defined via the links: $par(s) = \{x \in S \mid (x, s) \in L\}$. The links L are abstract links, denoting *some* relation, but not restricted to a particular one. Originally the acceptance conditions returned in or out, but we chose in this thesis the truth value style, since we often translate problems of argumentation to propositional logic and vice versa. An acceptance condition not only closely resembles a *Boolean function*, i.e. returning $v \in \{\mathbf{t}, \mathbf{f}\}$ for a subset of $par(s)$, we can straightforwardly define each acceptance condition as a Boolean formula. We usually denote by C_s the acceptance condition as the Boolean function and by φ_s the acceptance condition as a Boolean formula, s.t. for $X \subseteq S$ we have $C_s(X \cap par(s)) = \mathbf{t}$ iff $X \models \varphi_s$, i.e. the acceptance condition evaluates to true under $X \cap par(s)$ iff the acceptance formula evaluates to true under X. It is immediate that any such function for an acceptance condition can be written equivalently as a Boolean formula. For a closer study on these two forms the interested reader is referred to [89]. Unless noted otherwise we will assume that only the propositional formulae for each argument are given for an ADF. The difference in use of the two forms may be particularly important for computational purposes, since a naive representation of the Boolean function C_s consists of specifying the result for all subsets of $par(s)$, while the corresponding formula φ_s maybe much more compact, in fact the naive representation may be *exponentially* larger than the formula.

While the acceptance conditions specify the exact condition when we may accept an argument, the links are used to show dependencies between the arguments. For an argument s and the acceptance conditions as Boolean formulae we can see the set of parents of s by looking at $atoms(\varphi_s) = par(s)$. In principle one could define ADFs without links, e.g. just using a set of arguments and one formula for each argument as its acceptance condition, since the links can be extracted via the $atoms$ function. So it would be sufficient to write ADFs as a pair $D = (S, C)$. We however stick to the definition with triples, but will not always define the set of links explicitly, assuming that they are defined implicitly via the acceptance conditions.

Since the occurrence of an argument in $atoms(\varphi_s)$ respectively in $par(s)$ does not imply a particular relation between s and the parent, as was the case in AFs, we distinguish in the context of ADFs four different *classes* of relations [34]. A link can be of an attacking, a supporting or

a dependent nature, or even redundant. These four classes represent many concrete relations between arguments. Formally this is defined via the concepts of attacking and supporting links. A link is dependent if it is neither attacking nor supporting and redundant if it is both. Recall that a two-valued interpretation I can be equivalently represented as a set $I^{\mathbf{t}}$.

Definition 2.3.20. *Let $D = (S, L, C)$ be an ADF and $l \in L$ a link with $l = (a, b)$. The link l is*

- *attacking if there is no $X \subseteq S$ such that $X \not\models \varphi_b$ and $X \cup \{a\} \models \varphi_b$;*
- *supporting if there is no $X \subseteq S$ such that $X \models \varphi_b$ and $X \cup \{a\} \not\models \varphi_b$;*
- *dependent if l is neither attacking nor supporting; and*
- *redundant if l is both attacking and supporting.*

Intuitively a link $l = (a, b)$ is attacking if it is not the case that if for a certain status of arguments b is rejected and by accepting a also b gets accepted. This would imply a sort of support relation between a and b and hence the link is considered not attacking anymore, or in other words a link is attacking if there is no case for support. Similarly for the support relation. Intuitively speaking, if l is redundant, then the status of b is independent of the status of a. This means for all $X \subseteq atoms(\varphi_b)$ we have $X \models \varphi_b$ iff $X \cup \{a\} \models \varphi_b$. We denote by $att_D(s)$ for an ADF $D = (S, L, C)$ the set of attackers of s in D and by $supp_D(s)$ the supporters of s in D. We sometimes write L^+ for the set of supporting links of an ADF and L^- for the attacking links.

The dependent link is more complicated and dependent on the status of other arguments might be attacking or supporting. Consider the following example for ADFs and the link types.

Example 2.3.7. *Let $D = (S, L, C)$ be an ADF and $S = \{a, b, c, d\}$ with*

- $\varphi_a = b$;
- $\varphi_b = \neg a$;
- $\varphi_c = a \leftrightarrow b$; *and*
- $\varphi_d = d \vee \neg d$.

Then $L = \{(a, b), (b, a), (a, c), (b, c), (d, d)\}$. Further $L^+ = \{(b, a), (d, d)\}$ and $L^- = \{(a, b), (d, d)\}$. Let us closer examine that (b, a) is supporting. The interesting acceptance condition is $\varphi_a = b$, and hence it is sufficient to look at the two-valued interpretations $\emptyset$ and $\{b\}$. Under the former φ_a evaluates to false and to true under the latter. This means that $\emptyset \not\models \varphi_a$ and $\emptyset \cup \{b\} \models \varphi_a$, which is a counterexample to (b, a) being an attacking link. There is no counterexample to (b, a) being supporting, thus this link is a supporting link. To see e.g. that (d, d) is redundant, it is sufficient that φ_d is a tautology. We cannot find counterexamples for (d, d) being attacking or supporting, since in all cases φ_d evaluates to true. This would also hold if φ_d is unsatisfiable.

Lastly let us check the link type of (a, c). This link is a dependent link. The counterexample for it being attacking are the interpretations $\{b\}$ and $\{b, a\}$. The formula φ_c evaluates to false

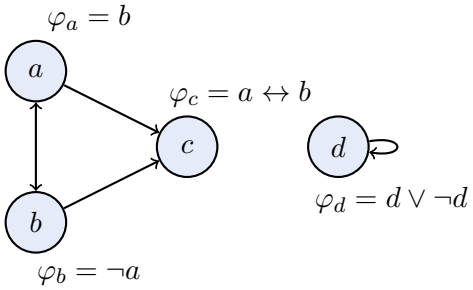

Figure 2.9: ADF with different link types

under former and to true under the latter. Similarly a counterexample for the link being supporting are the sets $\emptyset$ and $\{a\}$. If we would "fix" the truth value of b to e.g. t *by setting its variable in the formula to $\top$, then the resulting formula $a \leftrightarrow \top$ is equivalent to a and the link (a, c) is now supporting, i.e. overall it depends on the value of b.*

An ADF which does not have dependent or redundant links is called a *bipolar* ADF.[5] A special class of bipolar ADFs we are interested here in this thesis is the bipolar ADF with *known* link types, i.e. the sets $att_D(s)$ and $supp_D(s)$ are given for any argument s in D.

Definition 2.3.21. *Let $D = (S, L, C)$ be an ADF. We call D bipolar if L does not contain dependent or redundant links, i.e. $L = L^+ \cup L^-$ and $L^+ \cap L^- = \emptyset$.*

In this thesis we assume that for any given BADF the link types are known.

Example 2.3.8. *The ADF in Example 2.3.7 shown in Figure 2.9 is not a bipolar ADF. This can easily be seen from Example 2.3.7, which explains why e.g. (a, c) is neither attacking nor supporting. An example for a bipolar ADF is shown in Figure 2.10.*

Although at first glance this class of bipolar ADFs or even the class with known link types might appear to be very restrictive, it still includes many types of relations. In particular current applications of ADFs use BADFs [32, 145], where also the link type can be quickly determined.

2.3.4 Semantics of Abstract Dialectical Frameworks

The semantics of ADFs as defined in [31] are based on three-valued interpretations and an intuitive ordering of the three truth values, see Section 2.2.4 for an introduction to three-valued

[5] In some definitions of BADFs [34, 136] also redundant links are allowed. We assume in this work, mainly for convenience reasons, a "cleaned" BADF where each redundant link is simply removed and in the formula representation of the acceptance conditions we replace the variable corresponding to the removed link uniformly by a truth constant (e.g. $\top$). We consider this a harmless assumption, since for current applications of (B)ADFs [32, 145] the link types can be easily deduced.

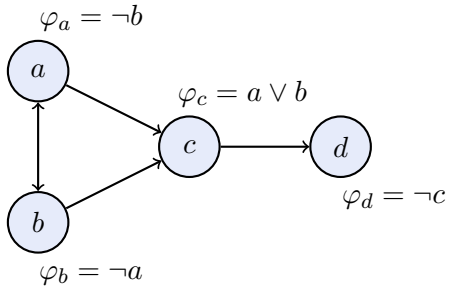

Figure 2.10: Bipolar ADF

logic.[6] We order the truth values $\mathbf{t}$, $\mathbf{f}$ and $\mathbf{u}$ according to their "information content" by the ordering $<_i$. This means we order the two "classical" truth constants $\mathbf{t}$ and $\mathbf{f}$ higher than $\mathbf{u}$, intuitively because $\mathbf{u}$ stands for an undecided or unknown truth value. Thus $\mathbf{u} <_i \mathbf{t}$ and $\mathbf{u} <_i \mathbf{f}$. The two truth values $\mathbf{t}$ and $\mathbf{f}$ are incomparable by $<_i$, which makes this ordering partial. An ordering is partial if it is reflexive, antisymmetric and transitive. This can straightforwardly be extended to compare three-valued interpretations.

Definition 2.3.22. *Let I and J be two three-valued interpretations defined on S. Then $I <_i J$ iff $I(s) <_i J(s)$ for all $s \in S$.*

Likewise we can define $>_i$, $\leq_i$ and $\geq_i$. A useful formal concept in this context of partial orderings is the theory of lattices. A partially ordered set (poset) is simply a pair $(X, \preceq)$ of a set X and a partial order on X, denoted here as $\preceq$. A lattice is a poset where every two elements $a, b \in X$ have a *supremum* and an *infimum*. This means that there exists a least upper bound, or also called join for each pair of elements in X, as well as a greatest lower bound, or also called meet for each pair of elements in X. Formally an element $x \in X$ is an upper bound for a subset $S \subseteq X$ if $y \preceq x$ for all $y \in S$. It is the least upper bound if for all upper bounds $x' \in X$ for S it holds that $x \preceq x'$. Similarly for the infimum. A join or meet-semilattice has only a join or respectively a meet for each pair of elements. Every non-empty finite lattice is *bounded*, i.e. has a greatest and least element, usually denoted by the letters 0 and 1 respectively. These have to satisfy $0 \preceq x \preceq 1$ for all $x \in X$. A non-empty finite meet-semilattice has a least element. We denote the meet of a set of elements $S \subseteq X$ by $\bigsqcap S$ and for two elements with $x \sqcap y$ and the join with '$\bigsqcup$' and '$\sqcup$'.

Given two or more truth values we can define the meet or consensus of these values as the meet of the meet-semilattice given by $(\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}, \leq_i)$. The meet of this simple meet-semilattice is defined by $\mathbf{t} \sqcap \mathbf{t} = \mathbf{t}$, $\mathbf{f} \sqcap \mathbf{f} = \mathbf{f}$ and $\mathbf{u}$ in all other cases. This can be extended to three-valued interpretations on a set S by considering the meet-semilattice defined by $(2^{S \to \{\mathbf{t},\mathbf{f},\mathbf{u}\}}, \leq_i)$.

[6] In [144, 147] an equivalent formalization of three-valued interpretations was used, namely a pair (X, Y) with $X \subseteq Y \subseteq S$ for the set of arguments S. Straightforwardly we can identify a three-valued interpretation I with (X, Y) using $I^{\mathbf{t}} = X$, $I^{\mathbf{u}} = Y \setminus X$ and $I^{\mathbf{f}} = S \setminus Y$, i.e. X represents the arguments set to true and acts as a "lower bound" while Y is the "upper bound". An argument inside these bounds is undecided and an argument not in Y is false.

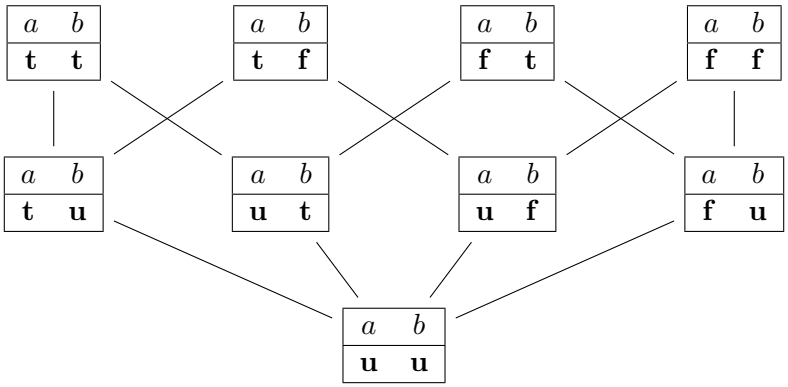

Figure 2.11: Meet-semilattice of three-valued interpretations

Example 2.3.9. *Consider the set $S = \{a, b\}$. Then the three-valued interpretations on this set can be partially ordered by $<_i$, which we show in Figure 2.11. An edge in this context depicts the ordering $<_i$ such that an element is ordered higher if it is drawn higher in the figure. This forms a meet-semilattice with the least element being the interpretation which sets both a and b to $\mathbf{u}$. The maximal elements are the two-valued interpretations on S.*

The semantics of ADFs are now defined via an operator which is similar to the characteristic function of AFs and we therefore call this operator characteristic function of ADFs. Based on a three-valued interpretation a new one is returned by the function, which accepts or rejects arguments based on the given interpretation. We need one formal concept for defining the characteristic function of ADFs, namely the set of all *two-valued* interpretations which are greater or equal w.r.t. their information content to a given *three-valued* interpretation.

Definition 2.3.23. *Let I be a three-valued interpretation defined on S. Then $[I]_2 = \{J \mid J$ a two-valued interpretation on S and $I \leq_i J\}$.*

In other words, given a three-valued interpretation I, then $[I]_2$ contains all concrete instantiations of I if we view the truth value $\mathbf{u}$ as a not yet known truth value. We exemplify this important notion in the following.

Example 2.3.10. *Let us revisit Example 2.3.9. Given an interpretation I on $\{a, b\}$ we see in Figure 2.11 the contents of the set $[I]_2$. Simply consider every interpretation in the figure which has more information than I according to $\leq_i$ and is two-valued. For instance take $I(a) = \mathbf{t}$ and $I(b) = \mathbf{u}$. This is shown in the middle row of the figure. Both J and K with $J(a) = K(a) = \mathbf{t}$ and $J(b) = \mathbf{t}$ and $K(b) = \mathbf{f}$ are in $[I]_2$ and no other interpretation is in $[I]_2$. In other words, we look at the maximal elements of this meet-semilattice w.r.t. $\leq_i$, which are ordered higher than I by $<_i$.*

For the interpretation I setting all arguments to $\mathbf{u}$, we have that all two-valued interpretations defined on $\{a, b\}$ are in $[I]_2$, i.e. all maximal elements of the meet-semilattice.

We can now define the characteristic function of ADFs as follows.[7]

Definition 2.3.24 ([34, Definition 4])**.** *Let $D = (S, L, C)$ be an ADF, I a three–valued interpretation defined over S, $s \in S$ and $\Gamma_D : (S \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}) \to (S \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\})$ a function from three-valued interpretations to three-valued interpretations. Then $\Gamma_D(I) = J$ with*

$$J(s) = \bigsqcap_{K \in [I]_2} K(\varphi_s)$$

That is, given a three-valued interpretation I a new one is returned by Γ_D for an ADF D. The new truth value for each argument s is given by considering all two-valued interpretations that extend I, i.e. all interpretations that assign either $\mathbf{t}$ or $\mathbf{f}$ to an argument, which is assigned $\mathbf{u}$ by I. Now we evaluate the acceptance condition of each argument under all these two–valued interpretations. If all of them agree on the truth value, i.e. all of them evaluate to $\mathbf{t}$ or respectively $\mathbf{f}$, then this is the result or the overall consensus. Otherwise, if there is a disagreement, i.e. we have $\mathbf{t}$ for one evaluation and $\mathbf{f}$ for another, then the result is undecided, i.e. $\mathbf{u}$.

Example 2.3.11. *Consider the three-valued interpretation I on $\{a, b\}$ with $I(a) = \mathbf{t}$ and $I(b) = \mathbf{u}$ as in Example 2.3.10. Then $J, K \in [I]_2$ with $J(a) = J(b) = \mathbf{t}$ and $K(a) = \mathbf{t}$, $K(b) = \mathbf{f}$. Assume that we are given an ADF $D = (\{a, b\}, L, C)$ with $\varphi_a = a$ and $\varphi_b = b$. Then both $J \models \varphi_a$ and $K \models \varphi_a$. This means that $\Gamma_D(I) = I'$ with $I'(a) = \mathbf{t}$. For the argument b we have $J(b) = \mathbf{t} \neq \mathbf{f} = K(b)$ and thus we construct the meet of $\{\mathbf{t}, \mathbf{f}\}$ which is given by $\mathbf{t} \sqcap \mathbf{f} = \mathbf{u}$ and this means that $I'(b) = \mathbf{u}$.*

We can equivalently say that for an ADF $D = (S, L, C)$ with $s \in S$ we have $\Gamma_D(I)(s) = \mathbf{t}$ if φ_s is a tautology if we replace each $I^{\mathbf{t}}$ with $\top$ in φ_s and each $I^{\mathbf{f}}$ with $\bot$ in φ_s. Similarly we can check for unsatisfiability for $\mathbf{f}$. Γ_D is a $\leq_i$-monotone operator, i.e. if $I \leq_i I'$, then $\Gamma_D(I) \leq_i \Gamma_D(I')$.

Proposition 2.3.5 ([34, Definition 4])**.** *Let $D = (S, L, C)$ be an ADF. Γ_D is $\leq_i$-monotonic.*

Proof. Let I, J be two three-valued interpretations on S and $I \leq_i J$. We have to show that $\Gamma_D(I) \leq_i \Gamma_D(J)$ holds. Clearly we have $[I]_2 \supseteq [J]_2$, since we know that $I^{\mathbf{u}} \supseteq J^{\mathbf{u}}$ and if $K \in [J]_2$, then $I^{\mathbf{t}} \subseteq J^{\mathbf{t}} \subseteq K^{\mathbf{t}}$ and $I^{\mathbf{f}} \subseteq J^{\mathbf{f}} \subseteq K^{\mathbf{f}}$ and thus $K \in [I]_2$. Let $\Gamma_D(I) = K_1$ and assume $a \in K_1^{\mathbf{t}}$, then for every $R \in [I]_2$ we have $R \models \varphi_a$ and thus we have for $K_2 = \Gamma_D(J)$ that $a \in K_2^{\mathbf{t}}$. Similarly for the arguments set to false. This implies the claim. □

Based on the notion of the characteristic function of ADFs we define the most important semantics of ADFs for our work, the admissible, complete, grounded and preferred interpretations of an ADF as defined in [31]. Further semantics of ADFs have been defined in [144]. Note that in [144] *two* different forms of semantics, which are generalizations of AFs are introduced. What we study in this thesis are the semantics based on the so-called ultimate operator

[7] Originally in [34] this function was used only for the grounded semantics, formerly called well-founded semantics and was defined on pairs of sets $(I^{\mathbf{t}}, I^{\mathbf{f}})$ from which one can extract a three-valued interpretation I. In [31] this function was adopted for several semantics. In tune with argumentation theory we call it here characteristic function of ADFs.

in [144]. For notation we use the function $\tau : \mathcal{ADF}_{\mathcal{P}} \to 2^{(\mathcal{P} \to \{\mathbf{t},\mathbf{f},\mathbf{u}\})}$, which assigns to each ADF $D = (S, L, C)$ a set of *three-valued interpretations* on S, denoted by $\tau(D)$. By slight abuse of notation we take the names of semantics of AFs, since the ADF semantics are meant to be generalizations. That is we consider for τ the functions adm, com, grd and prf, which denotes the admissible, complete, grounded and preferred interpretations of an ADF.

The definition of admissibility basically relies on the characteristic function of ADFs and is defined as for AFs, with the subtle discrepancy that we compare three-valued interpretations w.r.t. $<_i$ instead of $\subseteq$, which is obviously not adequate for such interpretations.

Definition 2.3.25 ([31])**.** *Let $D = (S, L, C)$ be an ADF, I a three-valued interpretation defined over S is in*

- $adm(D)$ *if* $I \leq_i \Gamma_D(I)$*;*
- $com(D)$ *if* $I = \Gamma_D(I)$*;*
- $grd(D)$ *if* $I \in com(D)$ *and there is no* $J \in com(D)$ *with* $J <_i I$*;*
- $prf(D)$ *if* $I \in adm(D)$ *and there is no* $J \in adm(D)$ *with* $I <_i J$*;*

Note that, as in the AF case, we sometimes abuse our notation slightly by identifying $I \in grd(D)$ with $grd(D)$, since $grd(D)$ is always a singleton set. We say that an interpretation I for a semantics τ and an ADF $D = (S, L, C)$ is non-trivial if there exists an $s \in S$ such that $I(s) \neq \mathbf{u}$. It is interesting to note that we do *not* require a certain conflict-free property, which is already "built-in" into the Γ_D operator. We can however define such a conflict-free interpretation [34].

Definition 2.3.26. *Let $D = (S, L, C)$ be an ADF, I a two-valued interpretation defined over S is conflict-free in D if $I^{\mathbf{t}} \models \varphi_s$ for each $s \in I^{\mathbf{t}}$.*

If we additionally require that I is two-valued and each $s \in I^{\mathbf{f}}$ evaluates to false under $I^{\mathbf{t}}$, then we have a two-valued model of the ADF D.[8]

Definition 2.3.27. *Let $D = (S, L, C)$ be an ADF, I a two-valued interpretation defined over S is a two-valued model of D if $I^{\mathbf{t}} \models \varphi_s$ for each $s \in I^{\mathbf{t}}$ and $I^{\mathbf{t}} \not\models \varphi_{s'}$ for each $s' \in I^{\mathbf{f}}$.*

We use the functions cf and mod for denoting the set of conflict-free interpretations and two-valued models of ADFs.

Example 2.3.12. *Let us go back to the framework D in Example 2.3.7, where we had the arguments $\{a, b, c, d\}$ with $\varphi_a = b$, $\varphi_b = \neg a$, $\varphi_c = a \leftrightarrow b$ and $\varphi_d = d \vee \neg d$. Then we have the models shown in Table 2.2 w.r.t. the semantics of ADFs. The right-most column depicts which semantics include the interpretation in the current row as a τ interpretation. Looking at a bit more complex example, consider ADF $D' = (S, L, C)$ as shown in Figure 2.10 with the arguments $\{a, b, c, d\}$ and $\varphi_a = \neg b$, $\varphi_b = \neg a$, $\varphi_c = a \vee b$ and $\varphi_d = \neg c$. In Table 2.3 we see the corresponding interpretations of this ADF.*

[8] In [147] a two-valued model is also called a two-valued supported model.

a	b	c	d	
$\mathbf{u}$	$\mathbf{u}$	$\mathbf{u}$	$\mathbf{u}$	adm
$\mathbf{u}$	$\mathbf{u}$	$\mathbf{u}$	$\mathbf{t}$	adm, com, grd, prf

Table 2.2: Admissible, complete, grounded and preferred interpretations of the ADF of Example 2.3.7

a	b	c	d	
$\mathbf{u}$	$\mathbf{u}$	$\mathbf{u}$	$\mathbf{u}$	adm, com, grd
$\mathbf{t}$	$\mathbf{f}$	$\mathbf{u}$	$\mathbf{u}$	adm
$\mathbf{t}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{u}$	adm
$\mathbf{f}$	$\mathbf{t}$	$\mathbf{u}$	$\mathbf{u}$	adm
$\mathbf{f}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{u}$	adm
$\mathbf{t}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{f}$	adm, com, prf
$\mathbf{f}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{f}$	adm, com, prf

Table 2.3: Admissible, complete, grounded and preferred interpretations of the ADF of Example 2.3.12

Let us exemplarily investigate why I is admissible in D' with $I^{\mathbf{u}} = S$ from the ADF in Figure 2.10. Since every interpretation is small or equal w.r.t. $\leq_i$ than I it is clearly admissible. It is also complete since $\Gamma_D(I) = I$. This can be seen that there exists for each argument s two interpretations J, K in $[I]_2$ such that $J \models \varphi_s$ and $K \not\models \varphi_s$. I is then also the grounded interpretation of D', since there cannot be a smaller complete one.

Now let us check that I is admissible in D' with $I^{\mathbf{t}} = \{a, c\}$ and $I^{\mathbf{f}} = \{b\}$ and d is set to $\mathbf{u}$. To see that this interpretation is admissible we have two interpretations in $[I]_2 = \{J, K\}$, J sets d to true and K sets d to false, and otherwise equal to I. Then all three arguments a, b and c evaluate under J and K to the same value as I and hence I is admissible. We need not check this for d, since regardless of the value φ_d evaluates under J or K, $I(d) = \mathbf{u}$ is clearly smaller w.r.t. $\leq_i$.

Definition 2.3.28. *Given an ADF $D = (S, L, C)$, a semantics τ, J a three-valued interpretation on S and an argument $s \in S$ then we define the following reasoning tasks.*

- $\mathsf{Enum}_\tau(D) = \tau(D)$
- $\mathsf{Cred}_\tau(s, D) = \begin{cases} \textit{yes} & \textit{if } s \in \bigcup\{I^{\mathbf{t}} \mid I \in \tau(D)\} \\ \textit{no} & \textit{otherwise} \end{cases}$
- $\mathsf{Skept}_\tau(s, D) = \begin{cases} \textit{yes} & \textit{if } s \in \bigcap\{I^{\mathbf{t}} \mid I \in \tau(D)\} \\ \textit{no} & \textit{otherwise} \end{cases}$
- $\mathsf{AllCred}_\tau(D) = \bigcup\{I^{\mathbf{t}} \mid I \in \tau(D)\}$

- $\mathsf{AllSkept}_\tau(D) = \bigcap\{I^{\mathbf{t}} \mid I \in \tau(D)\}$
- $\mathsf{Ver}_\tau(J, D) = \begin{cases} \textit{yes} & \textit{if } J \in \tau(D) \\ \textit{no} & \textit{otherwise} \end{cases}$

Example 2.3.13. *Consider the ADF D' of Example 2.3.12. Then $\mathsf{Cred}_{adm}(a, D') = \textit{yes}$ and $\mathsf{Skept}_{prf}(d, D') = \textit{no}$.*

We now show some basic properties of ADFs. First due to [54, Theorem 8.22] (a variant of the Knaster-Tarski Theorem applicable for our current setting) we know that for any ADF there exists a grounded interpretation and thus also at least one complete pair.[9] Similarly as in AFs one has a strong relation between the semantics defined on ADFs. Every two-valued model is a preferred interpretations of an ADF. Every preferred interpretation is a complete one and in turn is also admissible.

Theorem 2.3.6 ([31, Theorem 2])**.** *Let $D = (S, L, C)$ be an ADF. Then the following inclusions hold: $mod(D) \subseteq prf(D) \subseteq com(D) \subseteq adm(D)$. If $I \in adm(D)$, then $I^{\mathbf{t}} \in cf(D)$.*

Proof. The three inclusions were proven in [31, Theorem 2]. To show that if I is an admissible interpretation then $I^{\mathbf{t}}$ is also conflict-free just observe that $I^{\mathbf{t}} \in [I]$, i.e. the two-valued interpretation $I^{\mathbf{t}}$ is in $[I]_2$. Since for every $J \in [I]_2$ we have that $J \models \varphi_a$ for $a \in I^{\mathbf{t}}$ (since I is admissible), the claim follows. □

The relations between the ADF semantics are shown in Figure 2.12. An arrow from τ_1 to τ_2 denotes that for any ADF D we have $\tau_1(D) \subseteq \tau_2(D)$. The *dotted* line indicates that if $I \in \tau_1(D)$ then $I^{\mathbf{t}} \in \tau_2(D)$.

Definition 2.3.29. *Let $F = (A, R)$ be an AF. Then its associated ADF $D_F = (A, R, C)$ is defined by $\varphi_s = \bigwedge_{(a,s)\in R} \neg a$ for each $s \in A$.*

The associated ADF for an AF is clearly bipolar, it actually only includes attacking links.

Example 2.3.14. *Consider the AF from Example 2.3.1. We show in Figure 2.13 the associated ADF of this AF.*

The correspondence between the semantics of an AF and its associated ADF can now be shown.[10]

Theorem 2.3.7 ([31, Theorem 2 and Theorem 4] and [34, Proposition 1])**.** *Let F be an AF and its associated ADF be D_F. Then*

[9] In [31] the grounded semantics was defined as the least fixed point of Γ_D for an ADF D. Due to [54, Theorem 8.22] we know that there exists such a least fixed point and hence complete interpretations with a particular complete interpretation being the least one w.r.t. $\leq_i$. Thus we can equivalently define the grounded interpretation as the least fixed point of Γ_D or the least complete interpretation w.r.t. $\leq_i$ of an ADF.

[10] Another generalization of the AF stable semantics is the ADF stable semantics [31, 147], however for an ADF associated with an AF two-valued models and two-valued stable models coincide, since, intuitively speaking, AFs do not feature a support relation.

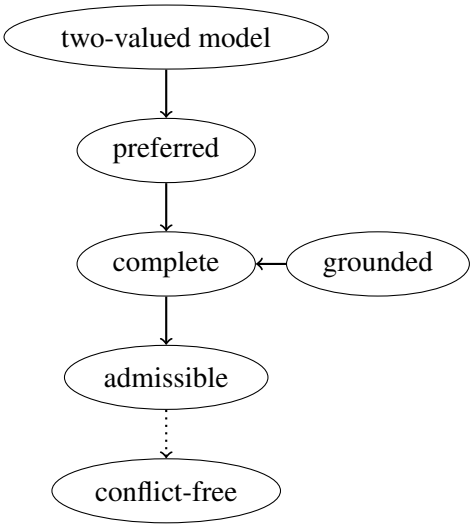

Figure 2.12: Relation between ADF semantics

- $E \in adm(F)$ *iff* $I \in adm(D_F)$ *with* $I^{\mathbf{t}} = E$;
- $E \in com(F)$ *iff* $I \in com(D_F)$ *with* $I^{\mathbf{t}} = E$;
- $E \in grd(F)$ *iff* $I \in grd(D_F)$ *with* $I^{\mathbf{t}} = E$;
- $E \in prf(F)$ *iff* $I \in prf(D_F)$ *with* $I^{\mathbf{t}} = E$;
- $cf(F) = cf(D_F)$; *and*
- $stb(F) = mod(D_F)$.

Similar as in the AF case we can compute the grounded interpretation via iterative applications of Γ_D on the initial interpretation setting all arguments to undecided. This notion will be used for investigating computational properties of the grounded semantics. The proof is very simple for the finite structures we assume in this work.

Definition 2.3.30. *Let $D = (S, L, C)$ be an ADF and $i \geq 0$ an integer. Define the interpretation $grd^i(D)$ as follows. $grd^0(D) = I$ with $\forall s \in S : I(s) = \mathbf{u}$. For $i > 0$ we define $grd^i(D) = \Gamma_D(grd^{i-1}(D))$.*

Proposition 2.3.8. *Let $D = (S, L, C)$ be an ADF $n \geq 0$ an integer and $I \in grd(D)$. It holds that $grd^n(D) \leq_i grd(D)$. For a $j \geq 0$ we have $grd^j(D) = grd(D)$.*

Proof. By definition we have $grd(D) = \Gamma_D(grd(D))$. Clearly we have $grd^0(D) \leq_i grd(D)$, thus $grd^1(D) \leq_i \Gamma_D(grd(D)) = grd(D)$ follows due to monotonicity of Γ_D (see Proposition 2.3.5). By a simple inductive argument we have $grd^n(D) \leq_i grd(D)$ for $n \geq 0$. Since $grd^n(D) \leq_i grd^{n+1}(D)$ holds (due to $grd^0(D) \leq_i grd^1(D)$ and induction) and we have finitely

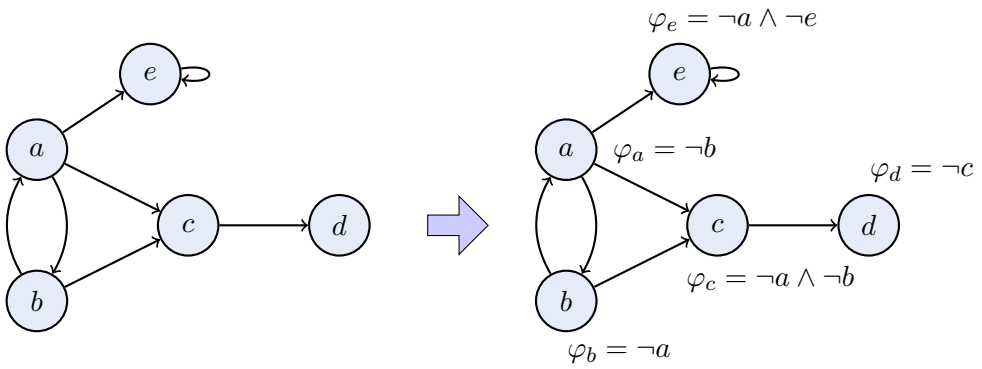

Figure 2.13: Example AF conversion to ADF

many arguments, i.e. S is finite, it follows that after finitely many applications j of Γ_D we reach a fixed point. Since it holds that $grd^{j+1}(D) = grd^j(D) \leq_i grd(D)$ we know that there exists a j s.t. $grd^j(D) = grd(D)$. □

Notice that the j in this proposition is at most $|S|$, since with each application of Γ_D we either arrive at a fixed point or update the status of at least one argument.

2.4 Computational Complexity

A central question in the area of computational complexity [133] is *how difficult is a problem?* Or very informally speaking, *what are the costs for solving a problem in general?* Complexity theory tries to answer these questions with a handy formal "tool set" to analyze the worst-time complexity of problems and classify them accordingly. Complexity classes contain problems which can be computed with a (restricted) formalized machine. That is we rely here on a formal model of computation. Typically the notion of a Turing machine is employed in complexity theory. We briefly review in this section the basics from complexity theory for our complexity analysis.

2.4.1 Basics

A decision problem L (or language) consists of a (possibly infinite) set of *instances* and a question, which can be answered by *yes* or *no*. Sometimes it is useful to define languages equivalently by stating that L consists of all "yes"-instances of the problem. An algorithm now *decides* the problem L if it answers "yes" on an input $l \in L$ iff l is a "yes" instance of L and "no" otherwise. A simple example would be the problem of **SAT**, which has as its instances all propositional logic formulae and the question is now to determine if a given instance is satisfiable. As a formal construct for machines or algorithms we apply the widely used notion of a Turing machine.

Definition 2.4.1. *A Turing machine is a quadruple $M = (K, \Sigma, \delta, s)$ with K a set of states, $s \in K$ the initial state. Σ is a finite set of symbols with $K \cap \Sigma = \emptyset$ and Σ contains two special*

symbols, the blank and first symbols $\{\sqcup, \triangleright\} \subseteq \Sigma$. *We assume that* h *(the halting state), "yes", "no" and the cursor directions '*$\longleftarrow$*', '*$\longrightarrow$*' as well as '*$-$*' are not present in* $K \cup \Sigma$. *Finally,*

- M *is a deterministic Turing machine if* δ *is a transition function*

$$\delta : K \times \Sigma \to (K \cup \{h, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\longleftarrow, \longrightarrow, -\}$$

- M *is a non-deterministic Turing machine if* δ *is a transition relation*

$$\delta \subset K \times \Sigma \times (K \cup \{h, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\longleftarrow, \longrightarrow, -\}$$

Informally speaking, a Turing machine is a description of an algorithm which works in the following way. The machine starts in the initial state s and has a "tape" where it reads and writes. This tape starts with the input to our algorithm/Turing machine. The symbols on the tape may be taken from Σ. The special symbols denote the start symbol $\triangleright$, which just states that this is the beginning of the tape and special empty symbols $\sqcup$. Usually one assumes that the word initially given terminates with a blank symbol (or infinitely many of them at the right side). The transition function or relation now defines the algorithm, i.e. what the machine should do in which situation. A transition function takes as input the current state, the symbol the machine reads and outputs a new state (possibly the same), a new symbol which should be written instead of the current one (possibly the same) and a direction to "move" the reader, i.e. which symbol should be read next, the one left or right from the current position (denoted by $\longleftarrow$ and $\longrightarrow$) or should it stay in the current position (using $-$).

In case of a non-deterministic Turing machine we may have a choice for the next state, i.e. it is non-deterministic in the sense that given a current state and a read symbol we may have several options. Note that the transition relation may be a function, i.e. assigning to each pair of state and symbol exactly one choice. This immediately implies that non-deterministic Turing machines are a generalization of deterministic ones. We now define what it means to "compute" using a Turing machine.

Definition 2.4.2. *Let* M *be a deterministic Turing machine. A configuration is a triple* (q, w, u) *with* $q \in K$ *and* $w, u \in \Sigma^*$. *We say that for this configuration* q *is the current state,* w *is the string left of the cursor including the cursor position and* u *right of the cursor. W.r.t.* M *a configuration* (q, w, u) yields in one step *a configuration* (q', w', u'), *denoted by* $(q, w, u) \xrightarrow{M} (q', w', u')$, *if* i *is the last symbol of* w *and* $\delta(q, i) = (p, \rho, D)$ *with* $q' = p$ *and depending on* D *one of three cases is true.*

- *if* $D = \longrightarrow$ *then* w' *is* w *with the last symbol* i *replaced by* ρ *and the first symbol of* u *appended to it (*$\sqcup$ *if* u *was empty) and* u' *is* u *with the first symbol removed (or* $\sqcup$ *if* u *was empty);*

- *if* $D = \longleftarrow$ *then* w' *is* w *with the last symbol* i *omitted from its end, and* u' *is* u *with* ρ *attached in the beginning;*

- *if* $D = -$ *then* w' *is* w *with the ending* i *replaced by* ρ *and* $u' = u$.

Table 2.4: Transition function of the deterministic Turing machine from Example 2.4.1

$p \in K$	$i \in \Sigma$	$\delta(p,i)$	$p \in K$	$i \in \Sigma$	$\delta(p,i)$
s	$\triangleright$	$(s, \triangleright, \longrightarrow)$	q_5	o	(“no”, o, $-$)
s	$x \in \Sigma \setminus \{d\}$	(“no”, x, $-$)	q_5	n	$(q_6, n, \longleftarrow)$
s	d	$(q_1, d, \longrightarrow)$	q_6	$x \in \Sigma \setminus \{o\}$	(“no”, x, $-$)
q_1	$x \in \Sigma \setminus \{d, \sqcup\}$	$(q_1, x, \longrightarrow)$	q_6	o	$(q_7, o, \longleftarrow)$
q_1	d	(“no”, d, $-$)	q_7	$x \in \Sigma \setminus \{r, o\}$	$(q_7, x, \longrightarrow)$
q_1	$\sqcup$	$(q_2, \sqcup, \longleftarrow)$	q_7	o	(“no”, o, $-$)
q_2	$x \in \Sigma \setminus \{n\}$	(“no”, x, $-$)	q_7	r	$(q_8, r, \longrightarrow)$
q_2	n	$(q_3, n, \longleftarrow)$	q_8	$x \in \Sigma \setminus \{a\}$	(“no”, x, $-$)
q_3	$x \in \Sigma \setminus \{n, d\}$	$(q_3, x, \longrightarrow)$	q_8	a	$(q_9, a, \longrightarrow)$
q_3	n	(“no”, n, $-$)	q_9	$x \in \Sigma \setminus \{a, o\}$	$(q_9, x, \longrightarrow)$
q_3	d	$(q_4, d, \longrightarrow)$	q_9	a	(“no”, a, $-$)
q_4	$x \in \Sigma \setminus \{r\}$	(“no”, x, $-$)	q_9	o	$(q_{10}, o, \longleftarrow)$
q_4	r	$(q_5, r, \longrightarrow)$	q_{10}	$x \in \Sigma \setminus \{g\}$	(“no”, x, $-$)
q_5	$x \in \Sigma \setminus \{n, \sqcup\}$	$(q_5, x, \longrightarrow)$	q_{10}	g	(“yes”, g, $-$)

We can extend “yields in one step” to yields *by considering to the transitive closure of yields in one step. We say that a configuration (q, w, u) yields configuration (q', w', u') in k steps denoted by $(q, w, u) \xrightarrow{M^k} (q', w', u')$ if there are configurations $c_1, \ldots, c_{k+1}$ such that $c_1 = (q, w, u)$, $c_{k+1} = (q', w', u')$ and for each $1 \leq i \leq k$ we have that c_i yields c_{i+1} in one step.*

In case of a non-deterministic Turing machine essentially the same concepts apply, but we may have a choice of the next configuration through the use of the transition relation. That is, given a configuration c_1 the non-deterministic Turing machine yields a configuration in one step to c_2 if it is allowed by the transition relation. For a Turing machine M and $w \in \Sigma^*$ the initial configuration is given by $(s, \triangleright, w)$.

Example 2.4.1. *Let us exemplify the notion of a Turing machine. We define the deterministic Turing machine M with $\Sigma = \{\triangleright, \sqcup, a, d, g, n, o, r\}$, $K = \{s, q_1, \ldots, q_{10}, \text{“yes”}, \text{“no”}\}$ and a transition function as shown in Table 2.4. We leave it as an exercise for the reader to figure out the purpose of this machine.*

Using the formal notion of a Turing machine we can now relate the length of the input, i.e. the number of symbols of the initial word on the tape and the steps for solving a given problem. Before that we need a notion to say that an algorithm actually solves a problem. A deterministic Turing machine M *decides* a problem L if for each “yes” instance $l \in L$ we have that $(s, \triangleright, l)$ yields in M a “yes” state and for each “no” instance $l' \in L$ it answers “no”. We say that M

operates within time $f(n)$ if for any $l \in L$ we have that M yields the result in at most $f(|l|)$ steps. In case of a non-deterministic Turing machine this means that it answers "yes" if there *exists* a computation path from the initial state and input word to a "yes"-state and "no" if there is no such path. A useful notion for time consumption is the so-called O-notation.

Definition 2.4.3. *Let $f, g : \mathbb{N} \to \mathbb{N}$ be functions. We write $f(n) = O(g(n))$ if there are positive integers c and n_0 s.t. for all $n \geq n_0$ we have $f(n) \leq c * g(n)$.*

Given a problem L and a Turing machine M that decides L, then we say that M decides L in polynomial time if there exists a polynomial $p(x)$ s.t. M decides L and operates in $O(p(x))$ steps, respectively time.

Definition 2.4.4. *We define* P *to be the class of decision problems decidable with a deterministic Turing machine in polynomial time.*

Using a non-deterministic Turing machine we arrive straightforwardly at another very important complexity class.

Definition 2.4.5. *We define* NP *to be the class of decision problems decidable with a non-deterministic Turing machine in polynomial time.*

Typically one considers problems in P to be "tractable" and problems not known to be in P to be "intractable". A useful notion is that of the "co-problem" of a problem L. L' is the co-problem of L if for each $l \in L$ that l is a "yes" instance of L iff l is a "no" instance of L'. If L is decidable with a deterministic Turing machine in time $f(n)$ then also the co-problem L' is decidable with a deterministic Turing machine in time $f(n)$, e.g. the co-problem of problems in P are also in P. For NP the class containing the co-problems is called coNP. The somewhat subtle difference being, that a non-deterministic Turing machines decides a problem in NP if there exists a successful computation path leading to a "yes" state in polynomial time, but on the other hand a non-deterministic Turing machines decides a problem in coNP if *all* computation paths lead to a "yes" instance (or "no" if one considers it as a co-problem).

A very useful tool for analyzing the complexity of given problems is the so-called *reduction*.

Definition 2.4.6. *A decision problem A is* P*-reducible to a decision problem B if there is a function f s.t. for each $x \in A$ we have that x is a "yes" instance of A iff $f(x)$ is a "yes" instance of B. And for each $x \in A$, $f(x)$ is computable by a deterministic Turing machine in polynomial time.*

A problem L is *complete* for a complexity class C if $L \in C$ and for all problems $A \in C$ we have that A is P-reducible to L. If the latter condition holds then L is called C-hard.[11] We will use the term reduction for P-reduction. The usual way to show C-hardness of L is to reduce a C-complete problem L' to L, which implies that all problems in C are P-reducible to L. We usually write C-c for completeness in class C. The last formal construct we need from complexity theory is the notion of an oracle machine. An oracle machine for a class C

[11] P-reducibility is sufficient for the complexity classes introduced here. For classes inside P one requires different restrictions on reductions.

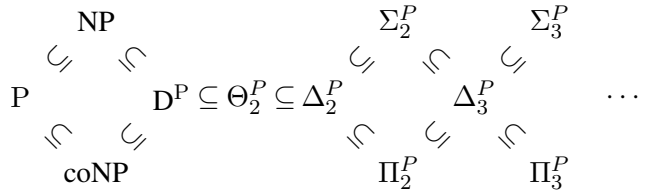

Figure 2.14: Relation between complexity classes

can answer a query for a problem in C in one step. We denote by P^C a deterministic Turing machine with an access to a C-oracle and by NP^C (resp. coNP^C) a non-deterministic Turing machine with access to a C-oracle. We can now define the complexity classes of the so-called polynomial hierarchy as follows.

Definition 2.4.7. *Let* $\Sigma_0^P = \Pi_0^P = \Delta_0^P = \mathrm{P}$. *Define for* $i \geq 1$ *the following classes.*

- $\Delta_i^P = \mathrm{P}^{\Sigma_{i-1}^P}$;
- $\Sigma_i^P = \mathrm{NP}^{\Sigma_{i-1}^P}$;
- $\Pi_i^P = \mathrm{coNP}^{\Sigma_{i-1}^P}$.

Further a problem L *is in* $\mathrm{D^P}$, *i.e.* $L \in \mathrm{D^P}$ *iff* L *can be characterized as two languages* $L_1 \cap L_2$ *(the intersection of "yes" instances) and* $L_1 \in \mathrm{NP}$ *and* $L_2 \in \mathrm{coNP}$.

The last class we consider is Θ_2^P, which consists of problems decidable by a deterministic Turing machine in polynomial time, which is allowed to make $O(\log(n))$ calls to a NP oracle, i.e. a logarithmic number of oracle calls w.r.t. the input size. An alternative name for this class is $\mathrm{P}^{\mathrm{NP}[\log n]}$. The relation between the complexity classes relevant for our work is now depicted in Figure 2.14. Note that it is not known whether the inclusions are proper. Many other useful classes are available [109].

Using the concept of P-reductions, which we usually abbreviate with just reductions, and complete problems for the defined classes from the literature, one has a powerful tool set to analyze the computational complexity of many problems.

2.4.2 Complexity of Abstract Argumentation: State of the Art

Complete problems for a complexity class C are essential for our every-day complexity analysis. They significantly support us to establish complexity results. Hence we review the important results for the problems considered in the previous sections. We abbreviate the problem of deciding satisfiability of a given propositional formula with $\mathbf{SAT}$, the validity by $\mathbf{VALIDITY}$. The problem given two formulae and deciding whether the first is satisfiable and the latter valid we denote by $\mathbf{SAT}$-$\mathbf{VALIDITY}$. The validity of QBFs we abbreviate by $\mathcal{QBF}_{\exists,n}$-$\mathbf{VALIDITY}$ and $\mathcal{QBF}_{\forall,n}$-$\mathbf{VALIDITY}$ for QBFs in $\mathcal{QBF}_{\exists,n}$ and $\mathcal{QBF}_{\forall,n}$ respectively.

Table 2.5: Computational complexity of propositional logic

problem	complexity
SAT	NP-c
VALIDITY	coNP-c
SAT-VALIDITY	$\mathrm{D^P}$-c
$\mathcal{QBF}_{\exists,n}$-**VALIDITY**	Σ_n^P-c
$\mathcal{QBF}_{\forall,n}$-**VALIDITY**	Π_n^P-c

Given a formula ϕ and a two-valued or three-valued interpretation I we can compute $I(\phi)$ in polynomial time. The NP completeness result in the next proposition is due to the seminal result by Cook [52].

Proposition 2.4.1. **SAT** *is* NP*-complete and* **VALIDITY** *is* coNP*-complete.*

Even for the typical normal form of propositional logic, CNF, the hardness still holds.

Proposition 2.4.2. **SAT** *is* NP*-complete, even when restricted to formulae in CNF.*

The NP-hardness even holds for the satisfiability problem of a so-called 3-CNF formula, i.e. a propositional formula $\phi = \{c_1, ..., c_n\}$ in CNF where we have $|c_i| \leq 3$ for $1 \leq i \leq n$. Combining satisfiability and validity leads immediately to a $\mathrm{D^P}$-complete problem.

Corollary 2.4.3. **SAT-VALIDITY** *is* $\mathrm{D^P}$*-complete.*

Although QBFs can be rewritten to propositional formulae, they are more succinct and have a higher complexity, shown in the next result.

Proposition 2.4.4 ([143, 156])**.** *Let* $n \geq 1$ *be an integer.*

- $\mathcal{QBF}_{\exists,n}$-**VALIDITY** *is* Σ_n^P*-complete; and*
- $\mathcal{QBF}_{\forall,n}$-**VALIDITY** *is* Π_n^P*-complete.*

In Table 2.5 we summarize the complexity of the problems on propositional logic we base our results on.

For AFs and ADFs we define decision problems for the reasoning tasks of credulous and skeptical acceptance and verification. We denote by Cred_σ the decision problem of deciding whether a given argument is credulously accepted under σ for a given AF, or respectively an ADF. Similarly Skept_σ denotes the decision problem for skeptical acceptance. By Ver_σ we denote the decision problem to decide for a given AF, respectively an ADF, if a two, respectively three valued interpretation is a σ interpretation of the framework.

We summarize the complexity results of AFs in Table 2.6 (see also [68]). The complexities of the problems on AFs for admissible and preferred semantics are shown by [56], except for the

Table 2.6: Computational complexity of reasoning in AFs

σ	Cred_σ	Skept_σ	Ver_σ
stb	NP-c	coNP-c	in P
adm	NP-c	trivial	in P
com	NP-c	P-c	in P
grd	P-c	P-c	P-c
prf	NP-c	Π_2^P-c	coNP-c
sem	Σ_2^P-c	Π_2^P-c	coNP-c
stg	Σ_2^P-c	Π_2^P-c	coNP-c
$stg2$	Σ_2^P-c	Π_2^P-c	coNP-c
$ideal$	in Θ_2^P	in Θ_2^P	in Θ_2^P
$eager$	Π_2^P-c	Π_2^P-c	D^P-c
$stg\text{-}idl$	Π_2^P-c	Π_2^P-c	D^P-c

Π_2^P-completeness result of skeptical preferred semantics, which is shown by [63]. The complete semantics is studied by [53]. Semi-stable and stage semantics were studied in [64, 82]. The $stg2$ semantics is studied in [71, 72, 97]. Ideal, eager and $stg\text{-}idl$ reasoning was studied by [62, 67, 69]. The P-completeness results are due to [69].[12]

The complexity landscape of ADFs is scarcely populated. We review here the results from the literature which apply to the semantics considered in this thesis. First it is known that the verification problem for the grounded semantics is harder on ADFs than on AFs.[13]

Proposition 2.4.5 ([34, Proposition 13])**.** Ver_{grd} *is* coNP*-hard for ADFs.*

Deciding whether an ADF is a BADF is also intractable. In the following short proof we also see an application of the so-called guess and check paradigm, commonly used to show NP membership. This works as follows, we guess (in polynomial time) a candidate X and then check in polynomial time if this candidate is a "witness" for our solution. In case of propositional logic this would be a guessed (two-valued) interpretation and then checking if this assignment satisfies the formula. This directly corresponds to a non-deterministic Turing machine, where each computation path corresponds to one such guess and check. If the formula is satisfiable then we can have a guess for each possible interpretation and one is guaranteed to succeed, i.e. being a witness to show that the formula is satisfiable. Clearly constructing a guess and checking it, i.e. evaluating the formula under this guessed interpretation, can be done by a non-deterministic

[12] The P upper bounds for stable, admissible and complete extensions were actually improved to membership in the so-called complexity class L in [69], i.e. the class restricting *space* consumption of a Turing machine to be logarithmic w.r.t. the size of the input. More details about such classes can be found in [133].

[13] ADF semantics in [34] are based on two-valued interpretations. However, the proofs are straightforward to adapt for our setting.

Turing machine in polynomial time. Furthermore we also see a typical coNP membership proof by considering the co-problem and showing NP membership.

Proposition 2.4.6 ([34, Proposition 14])**.** *Deciding whether no link in a given ADF is dependent is* coNP*-complete.*

Proof. Hardness was shown in [34, Proposition 14] (note that BADFs there allow redundant links). To see coNP membership consider the co-problem, i,e, deciding whether there exists a dependent link in the ADF. Let $D = (S, L, C)$ be an ADF and $l \in L$. We guess $X, Y \subseteq S$ and $(s, s') = l \in L$. We now check if l is dependent by verifying that $X \models \varphi_{s'}$ and $X \cup \{s\} \not\models \varphi_{s'}$ for a counterexample that l is supporting. For the counterexample that l is attacking we verify that $Y \not\models \varphi_{s'}$ and $Y \cup \{s\} \models \varphi_{s'}$. These checks can clearly be done in polynomial time. □

Deciding the type of a link is in general also intractable.

Proposition 2.4.7. *Deciding whether a given link in a given ADF is*

1. *attacking is* coNP*-complete;*
2. *supporting is* coNP*-complete;*
3. *dependent is* NP*-complete;*
4. *redundant is* coNP*-complete.*

Proof. Item 1 was shown in [89, Proposition 4.4.1 and Proposition 4.4.2]. For showing item 2 one can easily adapt the proof of [89, Proposition 4.4.1 and Proposition 4.4.2] for supporting links.

Let $D = (S, L, C)$ be an ADF and $l \in L$. Showing that it is NP-complete to verify that l is dependent follows as a corollary from Proposition 2.4.6 (hardness follows from [34, Proposition 14]). Lastly deciding if a link is redundant is the same as deciding if a link is attacking and supporting, which can be done by two independent coNP-complete checks. Hence the problem is in coNP. For showing hardness consider the problem of deciding whether ϕ is a valid formula, which is a coNP-complete problem, see Proposition 2.4.1. We now reduce $\mathbf{VALIDITY}$ to this problem. Let ϕ be a formula, now define an ADF $D = (S, L, C)$ with $S = atoms(\phi) \cup \{f\}$ as follows. Let $\varphi_x = x$ and $\varphi_f = f \vee \phi$. We assume w.l.o.g. that $f \notin atoms(\phi)$. The ADF D can be constructed clearly in polynomial w.r.t. the size of ϕ. We now show that ϕ is valid iff $(f, f) \in L$ is a redundant link. If ϕ is valid, then clearly there is no $X \subseteq S$ s.t. $X \not\models \varphi_f$ and (f, f) is attacking and supporting, thus redundant. Assume now that (f, f) is redundant. This means that for any $X \subseteq S$ we have $X \models \varphi_f$ iff $X \cup \{f\} \models \varphi_f$. Since $\{f\} \models \varphi_f$, we know that for any such X it holds that $X \models \varphi_f$. Thus for every $Y \subseteq S$ with $Y \cap \{f\} = \emptyset$ we have $Y \models \varphi_f$ (since $Y \cup \{f\} \models \varphi_f$) and clearly $Y \subseteq atoms(\phi)$ and $S \setminus \{f\} = atoms(\phi)$ we know that any two-valued interpretation on ϕ must satisfy ϕ thus implying that ϕ is valid. □

Finally a positive result is that the verification problem of the grounded semantics becomes significantly easier on BADFs with known link types, i.e. it is tractable.

Proposition 2.4.8 ([34, Proposition 15]). Ver_{grd} *is in* P *for BADFs with known link types.*

We greatly extend the complexity results for ADFs in Chapter 4. In particular we show the computational complexity of the decision problems Cred_σ, Skept_σ and Ver_σ for ADFs and BADFs with known link types for semantics $\sigma \in \{adm, grd, prf\}$.

CHAPTER 3

Advanced Algorithms for Argumentation Frameworks

In this chapter we present novel algorithms for computationally hard problems defined on argumentation frameworks (AFs). In particular we develop algorithms for credulous and skeptical reasoning under preferred, semi-stable, stage, ideal, eager and stage-ideal semantics. Additionally our algorithms are capable of enumerating all extensions. Our algorithms rely on decision procedures for the Boolean satisfiability problem (SAT). The very basic idea is to delegate certain computationally hard subtasks to a search engine capable of solving the SAT problem efficiently. A SAT-solver is a procedure, which receives as input a propositional formula and returns a witness for satisfiability (a model) if the formula is satisfiable and "no" otherwise.

On the one hand delegating hard subtasks allows us to utilize existing efficient search engines of modern SAT-solvers. This means we are able to re-use existing technology and research directly. On the other hand we can, via an algorithm based on SAT-solvers, set up calls to the solver iteratively using domain knowledge for the problem to solve. The overall workflow for our algorithms is shown in Figure 3.1. So first an AF, a semantics and a reasoning mode is given as input. Then in the main procedure we construct Boolean queries for our SAT-solver. After possibly multiple calls to the solver we arrive at our decision. In some cases we need a (simple) postprocessing step.

If one considers fully declarative approaches such as [85], which specify the whole problem in one encoding, and direct instantiations of algorithms for AF problems such as [131], then our approach can be seen as a hybrid variant of these two. In particular we have less declarativeness by applying an imperative algorithm on top of more declarative queries and require some engineering effort to develop our algorithms. But we do not need to engineer search engines for computationally involving subtasks, e.g. NP-hard subtasks.

The algorithms developed for a semantics σ can be grouped into two categories. We call algorithms of the first category *search algorithms*. The other category comprises of algorithms relying on advanced SAT technology, or *SAT extensions*. Intuitively, our search algorithms traverse a search space of a computationally easier semantics σ', called *base semantics*, to compute

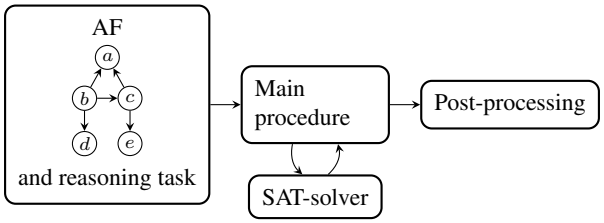

Figure 3.1: Basic workflow for the algorithms based on iterative SAT procedures

extensions of σ. The base semantics is chosen in such a way that its associated reasoning tasks can be performed by a SAT-solver. Overall this leads to an algorithm iteratively calling such a solver. This approach is used for the preferred, semi-stable and stage semantics.

An important feature of our search algorithms is that the number of calls to a SAT-solver is contingent on inherent properties of the AF. Since the problems we tackle are hard for a class in the second level of the polynomial hierarchy (e.g. Σ_2^P hard), we cannot avoid, under standard complexity theoretic assumptions, that we may require an exponential number of oracle or SAT calls. However, we provide a bound on the number of SAT calls by using inherent parameters of the AF. The formal basis for this observation are certain *subclasses* of AFs. We use theoretical investigations of computational properties of AFs in these subclasses [77]. Classes of AFs and their corresponding parameters have been investigated in quite some depth, see e.g. [53, 61, 69]. We use in this thesis mainly the class sol_σ^k, which contains AFs having at most k σ-extensions. More precisely, our search algorithms require a polynomial number of SAT-calls w.r.t. the size of the AF and a *fixed* k to solve the decision problem at hand. Another class we utilize is $\mathsf{stablecons}_\sigma^k$, which requires that each extension has at most k arguments not in the range of the extension.

Our algorithms based on advanced SAT techniques utilize *minimal correction sets* (MCSes) [115] and *backbones* [110]. Given an unsatisfiable formula in CNF, a correction set is a set of clauses which if removed from the formula results in a satisfiable subformula. We show that MCSes naturally correspond to problems for the semi-stable and stage semantics and can be used for eager and stage-ideal semantics. The backbone of a satisfiable formula consists of all literals which are entailed by the formula. This concept supports us to instantiate an existing algorithm for the ideal semantics [67]. In the SAT community several algorithms were studied for the computation of MCSes and backbones, which typically also rely on iterative SAT procedures. The benefit of using these techniques is that they enable us to develop algorithms, which are in-between the search algorithms and fully declarative systems. This means that we are closer to specifying our problem at hand in a declarative manner and require less engineering effort than required to develop a search algorithms. Further we can re-use existing sophisticated solvers for MCSes and backbones.

This chapter is organized as follows

- Section 3.1 briefly recapitulates research on SAT-solvers, which we use for our algorithms;
- in Section 3.2 we recall important sub classes of AFs. These are restricted AFs, which may have at most k extensions for a semantics σ, or at most k arguments in such an extension

which are not in the range of the extension;

- Section 3.3 develops search algorithms for the preferred, semi-stable and stage semantics. We first show a generic algorithm. Subsequently we instantiate this algorithm for the problems at hand;
- lastly in Section 3.4 we apply existing algorithms for computing minimal correction sets and backbones of Boolean formulae to solve problems regarding the semi-stable, stage, eager, ideal and stage-ideal semantics.

Some of our results in this chapter are published. In [76, 77] we developed the search algorithms. The important complexity results underlying many of our approaches were derived by Wolfgang Dvořák in [76]. The approaches using advanced SAT extensions were published in [153].

3.1 SAT Solving

In this section we briefly recall techniques for solving the SAT problem and introduce some useful concepts for applications of SAT. Generally one considers as a satisfiability solver (SAT-solver) a search algorithm for finding a witness (a model) for satisfiability of a given formula ϕ (in CNF), or proving unsatisfiability. Thereby these systems solve the famous NP-complete problem of Boolean satisfiability. Even though this problem is in general intractable, contemporary SAT-solvers such as MiniSAT [84] are capable of solving instances with hundreds of thousands of literals and clauses.

Algorithms underlying such solvers have been under research for many years. The success of SAT-solvers is owed not only to successful theoretical research. In the so-called SAT competition [108] solvers compete against each other and are evaluated w.r.t. their performance. The competitions motivated efforts to come up with novel solving techniques.

Conflict-driven clause learning Modern SAT-solvers incorporate numerous techniques for finding a model of a given formula. One of the most successful techniques is called conflict-driven clause learning (CDCL) [124, 123, 158]. We illustrate here the main ideas of this approach. CDCL is a general concept and certain aspects may vary in concrete CDCL algorithms. First we introduce some basic terminology of SAT solving. Let ϕ be a Boolean formula. Many SAT-solvers construct a witness for ϕ's satisfiability, or prove unsatisfiability, by iteratively assigning variables to truth values in a partial interpretation I. There are two important kinds of assignments. A SAT-solver may assign a value to a variable by a *decision* or by *propagation*. Assignments that are propagated are forced by the current partial assignment, whereas a SAT-solver chooses assignments for decisions usually through heuristics. Every time a SAT-solver decides on a value a corresponding *decision level* l is incremented by one, which is initially 0 (nothing is decided).

Assignment by propagation is computed by a fundamental concept called *Boolean constraint propagation* (BCP). BCP refers to extending a current partial assignment I to I' by propagation on the formula ϕ. I' extends I if $I = I'|_{dom(I)}$. Consider a partial assignment I with $dom(I) =$

$\{x_1, \ldots, x_{n-1}\}$ and a clause $c = \{l_1, \ldots, l_n\} \in \phi$ with $atoms(l_i) = \{x_i\}$ for $1 \leq i \leq n$. That is, I assigns to all literals in c a value, except for l_n. Assume that $I(l_i) = \mathbf{f}$ for all $1 \leq i \leq n-1$. Then the only possibility for an I', which extends I, to satisfy c is to evaluate l_n to true. This can only be achieved by setting the value of x_n, s.t. $I'(l_n) = \mathbf{t}$. This means the value of x_n is *implied* (or forced) by the clause c and the partial interpretation I. A special case of this procedure is if $|c| = 1$, i.e. c is unit. A SAT algorithm typically distinguishes between assignments made by a decision and assignments forced by BCP. Like decisions, propagated assignments have a corresponding decision level, namely the level at which BCP inferred the assignment. A CDCL may *backtrack* to a certain decision level l. Then all decisions and propagations at levels $l' > l$ are *erased* (or reverted).

The idea of CDCL is now to decide upon values for variables, applying BCP on the current partial interpretation and in case of a conflict c, backtracking and augmenting the given formula ϕ with so-called *learnt* clauses. Learnt clauses are crucial for correctness and performance of a CDCL algorithm. Using learnt clauses one avoids to re-visit parts of the search space. A CDCL algorithm starts with decision level $l = 0$ and the initial partial interpretation assigning no variables. At each decision level BCP is applied to propagate values of the current partial interpretation. If no conflict is revealed through BCP, i.e. no clause is unsatisfied by the current partial interpretation, then a CDCL algorithm decides the value for the next unassigned variable. If BCP revealed a conflict via clause $c \in \phi$, i.e. $I \not\models c$, then the algorithm learns a clause c' by conjoining it with ϕ and backtracks to a lower decision level. The clause c' is constructed by inspecting the so-called *implication graph*. From this graph a CDCL algorithm infers the decisions which lead to the conflict c. The implication graph is a directed graph, whose vertices consist of assigned and propagated literals and the clause c. There is a directed edge (l_1, l_2) in this graph if l_2 was propagated by BCP using l_1. Additionally all literals with variables in $atoms(c)$ have a directed edge to c. This means that all assignments by decisions have no incoming edges. The decisions which lead to the conflict c are those vertices corresponding to decisions without incoming edges that have a path to c. There are different strategies for learning a clause from the implication graph. One possibility is to construct the negation of the conjunction of literals which lead to the conflict c, which is equivalent to a clause c'. CDCL algorithms make sure that $\phi \models c'$ holds.

If a clause c' is learnt the algorithm backtracks. Here again CDCL algorithms differ in terms of their backtracking strategy. One choice is to backtrack to the highest decision level of the corresponding decision variables in c', excluding the decision variable of the current decision level. For example assume that the current decision level is $l = 2$ and we have the decisions $p@1$ and $q@2$, denoting that we decided to assign p to true at level one and q to true at level 2. Further assume that we learnt the clause $c' = (\neg p \vee \neg q)$. Then we would backtrack to $l = 1$, since c' contains two decision variables. If we exclude the most recent decision of q, then the highest remaining decision is that of p at level 1. At the now lowered decision level BCP will propagate the value of the most recent decision, before backtracking, to the opposite value as before. In the example, after backtracking we still have the partial assignment of p set to true. Thus BCP infers through c' that q must be false. In this way the CDCL algorithm implicitly "flips" the value of the last decision by applying BCP on the learnt clause.

We show an example computation in Table 3.1. The initial formula is ϕ_0. We assume a

formula	level	decision	Boolean constraint propagation
ϕ_0	$l = 0$	-	-
	$l = 1$	$I(p) = \mathbf{t}$	-
	$l = 2$	$I(q) = \mathbf{t}$	$I(r) = \mathbf{t}$, conflict $(\neg p \vee \neg q \vee \neg r)$
			learn clause $(\neg p \vee \neg q)$ and backtrack to $l = 1$
$\phi_1 = \phi_0 \wedge (\neg p \vee \neg q)$	$l = 1$	$I(p) = \mathbf{t}$	$I(q) = \mathbf{f}$, $I(r) = \mathbf{f}$, $I(s) = \mathbf{t}$, conflict $(r \vee \neg s)$
			learn clause $(\neg p)$ and backtrack to $l = 0$
$\phi_2 = \phi_1 \wedge (\neg p)$	$l = 0$	-	$I(p) = \mathbf{f}$
	$l = 1$	$I(q) = \mathbf{t}$	-
	$l = 2$	$I(r) = \mathbf{t}$	SAT

Table 3.1: CDCL example for $\phi_0 = (\neg p \vee \neg q \vee r) \wedge (\neg p \vee \neg q \vee \neg r) \wedge (r \vee s) \wedge (r \vee \neg s) \wedge (q \vee \neg r)$

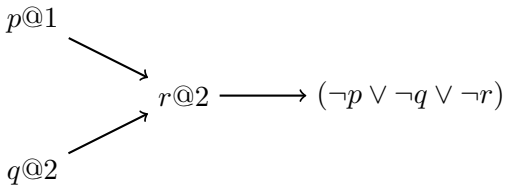

Figure 3.2: Example implication graph

simple ordering on the decisions: (p, q, r, s). We first assign p to true (level 1). BCP cannot infer further implications in this case. On the other hand at decision level 2, after we decided to assign true to q BCP can infer that r must be true as well (via clause $(\neg p \vee \neg q \vee r)$). This leads to an unsatisfied clause $((\neg p \vee \neg q \vee \neg r))$. The corresponding implication graph is shown in Figure 3.2. We conclude to learn $c = (\neg p \vee \neg q)$, since these two decisions lead to the conflict (vertices of decisions without incoming edges with a path to the conflict in the implication graph). We backtrack to the lowest decision level of a variable in c, except for the most recently decided variable q. Back at level 1 BCP infers that q must be false w.r.t. the current partial interpretation. We further conclude that r must be false and s must be true. This again leads to a conflict and we learn $\neg p$. After backtracking to $l = 0$ we decide upon variables on q and r and conclude satisfiability of the formula. The partial interpretation holds then $I(p) = \mathbf{f}$, $I(q) = \mathbf{t}$, $I(r) = \mathbf{t}$. The value for s is not relevant in this case and we may assign it arbitrarily.

In case an empty clause is learnt one concludes unsatisfiability. An empty clause is learnt if a conflict can be derived without using decisions.

Iterative and incremental SAT solving In some applications a single call to SAT solver does not suffice to solve the problem at hand. One can use then a sequence of SAT calls. One starts with an initial formula and checks the satisfiability of this formula. Depending upon the result and the potential witness one can construct another call to the SAT-solver and so on. If

subsequent calls re-use parts of the original satisfiable formula and add or remove new clauses then one can use *incremental* SAT solving (as supported e.g. by [84, 154]). Here SAT-solvers can partially re-use the state of previous calls. All learnt clauses which can still be derived from the subsequent problem can be kept. Learnt clauses derived from dropped constraints have to be dropped as well.

Relaxation literals An interesting concept for applications of SAT are so-called relaxation literals. Consider a formula ϕ with the clause $c = (a \vee b)$. Clearly a SAT-solver has to satisfy this clause for satisfying ϕ. One can now "relax" the constraint c by adding a relaxation literal x not occurring in ϕ to the clause before computation starts. That is the result is $c' = (a \vee b \vee x)$. By assigning x the value $\mathtt{t}$ one can "drop" the clause. This means that c' is satisfied with the assignment of the auxiliary variable x and the original clause is not considered anymore. Simulating that the original clause has to satisfied is likewise achieved by setting x to false. In this way clauses may be "switched on" or "switched off" also during incremental SAT solving.

Other techniques for SAT solving Numerous further techniques for SAT solving have been studied and applied. The decision ordering in CDCL, for instance, is very important and can be improved by suitable variable selection heuristics [120]. Variable ordering heuristics try to "localize" the search procedure. Another important branch of research is devoted to preprocessing techniques [83]. These techniques are applied before the actual SAT solving and may reduce the given instance or already return a witness or decision. An example of preprocessing is subsumption. A clause c_1 subsumes c_2 if $c_1 \subseteq c_2$. During preprocessing the clause c_2 may be discarded completely.

3.2 Classes of Argumentation Frameworks

Subclasses of AFs play an important role in particular in computational analysis. Many problems become in fact easier if the given AF is from a restricted domain. Several classes are identified to this date, see e.g. [53, 61, 69]. For our work we will use two classes, one which restricts the number of extensions and another class which restricts each extension to have a low distance w.r.t. the arguments not in range. Both classes are parametrized by a positive integer $k \geq 0$ and a semantics σ.

Definition 3.2.1. *Let σ be a semantics and $k \geq 0$ an integer. We denote by sol_σ^k the class of all AFs F such that $|\sigma(F)| \leq k$.*

The other subclass can also be defined straightforwardly as follows.

Definition 3.2.2. *For a semantics σ and $k \geq 0$ an integer, we call an AF $F = (A, R)$ k-stable-consistent under σ if for each $E \in \sigma(F)$, $|A \setminus E_R^+| \leq k$ holds. We use $\mathsf{stablecons}_\sigma^k$ to denote the respective classes of AFs for given k and σ.*

When restricted to these classes, some decision problems become easier. For instance, the decision problem Cred_{sem} for general AFs is Σ_2^P-complete, but for AFs in one of these two classes in Δ_2^P.

Theorem 3.2.1 ([77, Theorem 3 and Theorem 4]). *Let $k \geq 0$ be an integer and $\sigma \in \{prf, sem, stg\}$. Then Cred_σ and Skept_σ are in Δ_2^P when restricted to AFs from sol_σ^k.*

The search algorithms in Section 3.3 are based on the proof of this theorem. In particular Algorithm 1 presents the most important aspects for showing this result. For the class $\mathsf{stablecons}_\sigma^k$ and semi-stable and stage semantics one has a similar result, as shown in the next theorem.

Theorem 3.2.2 ([77, Theorem 1]). *Let $k \geq 0$ be an integer and $\sigma \in \{sem, stg\}$. Then Cred_σ and Skept_σ are in Δ_2^P, when restricted to AFs from $\mathsf{stablecons}_\sigma^k$.*

This result is used as a subroutine in one of our search algorithms (Algorithm 8 in Section 3.3.4) and in Algorithm 11. We briefly sketch the proof idea underlying this result from [77, Theorem 1]. Consider we want to decide whether $a \in A$ for an AF $F = (A, R)$ is credulously accepted w.r.t. semi-stable semantics. Let $\mathcal{S} = \{S \mid S \subseteq A\}$. We now iteratively take a maximal element S from $\mathcal{S}$ (w.r.t. $\subseteq$). We check whether there is an admissible set $E \in adm(F)$ s.t. $a \in E$ and $E_R^+ = S$. This subproblem is in NP. It can be decided by guessing a set $X \subseteq A$ and checking if it is admissible (conflict-freeness and defense of all arguments in X can be checked in polynomial time), if $a \in X$ and if $X_R^+ = S$. If the guess succeeds, then we have found a semi-stable extension containing a, thus we have that $\mathsf{Cred}_{sem}(a, F)$ is true. If the guess is not successful, then we check again without requiring that $a \in X$. If this succeeds, then X is a semi-stable extension and we exclude from future iterations all sets from $\mathcal{S}$ which are smaller or equal to S. Otherwise we just exclude S from $\mathcal{S}$. Then we repeat this process by selecting the next maximal element of $\mathcal{S}$. Since we go through $\mathcal{S}$ by choosing always a maximal element we visit all range sets of semi-stable extensions in the worst case (because if we find one, we exclude sets with a smaller range by removing them from $\mathcal{S}$). In other words this procedure goes through the set $\{E_R^+ \mid E \in sem(F)\}$ in order of the cardinality of the ranges. If $\mathsf{Cred}_{sem}(a, F)$ is true, then this procedure finds a witness E which decides our query. If all sets have been visited, we return *no*. If $F \in \mathsf{stablecons}_{sem}^k$, then we can bound the set $\mathcal{S}$ by $\mathcal{S} = \{S \mid S \subseteq A, |A \setminus S| \leq k\}$. This set is polynomial w.r.t. the size of A ($|A|$) and k. This can be seen by the fact that we basically can look at all subsets of A with a cardinality of at most k. The number of such sets is $\Sigma_{j=0}^k \binom{|A|}{j} \leq (|A| + 1)^k$ and thus we have polynomially many such sets w.r.t. $|A|$ and a *fixed* k. For stage semantics one can apply the same procedure, just replacing admissible sets with conflict-free sets.

The following AF acts as our running example in this chapter.

Example 3.2.1. *Let $F = (A = \{a, b, c, d, e, f\}, R = \{(a, b), (b, a), (a, c), (b, c), (c, d), (a, e), (f, f), (f, e)\})$ be an AF. In Figure 3.3 we show the graphical representation of F. The extensions are given by:*

- $cf(F) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{a, d\}, \{b, d\}, \{b, e\}, \{c, e\}, \{d, e\}, \{b, d, e\}\}$;
- $adm(F) = \{\emptyset, \{a\}, \{b\}, \{a, d\}, \{b, d\}\}$;
- $com(F) = \{\emptyset, \{a, d\}, \{b, d\}\}$;
- $grd(F) = \{\emptyset\}$;

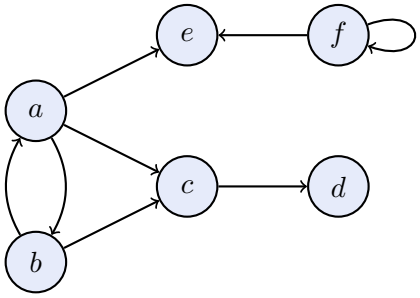

Figure 3.3: Example argumentation framework F

- $stb(F) = \emptyset$;
- $prf(F) = \{\{a, d\}, \{b, d\}\}$;
- $sem(F) = \{\{a, d\}\}$; *and*
- $stg(F) = \{\{a, d\}, \{b, d, e\}\}$.

Further F is in the following classes: sol^2_{prf}, sol^1_{sem}, sol^2_{stg}, *as well as in* $\mathsf{stablecons}^1_{sem}$ *and* $\mathsf{stablecons}^1_{stg}$. *To see that* $F \in \mathsf{stablecons}^1_{sem}$ *consider its only semi-stable extension* $\{a, d\} = S$. *Then* $S^+_R = \{a, b, c, d, e\} = (A \setminus \{f\})$, *i.e.* $|A - S^+_R| = 6 - 5 = 1$.

3.3 Search Algorithms

The basic idea of our search algorithms to solve the computationally hard problems for AFs for the preferred, semi-stable and stage semantics is to consider a simpler semantics or concept, which we call base semantics. We use as base semantics admissible sets or complete extensions for preferred and semi-stable semantics. For the stage semantics we use conflict-free sets as the base semantics.

The idea is to traverse the search space of the base semantics in an iterative manner to find an extension of the computationally more complex semantics. A base semantics is chosen in such a way that the complexity of the corresponding decision problems of this traversal is in NP or in coNP and thus solvable by a SAT-solver. We start with a more abstract and generic version of our search algorithms in Section 3.3.1. We define some auxiliary concepts for the base semantics. We use orderings in our algorithms, in particular *strict preorders* (denoted by $\prec$), which are orderings on a set, satisfying irreflexivity and transitivity. By irreflexivity we mean that for no element x it can be the case that $x \prec x$.

Definition 3.3.1. *Let X be a set and $\prec$ a strict preorder on X. Then we define the maximal elements of X w.r.t. $\prec$ to be $max_\prec(X) = \{x \in X \mid \nexists y \in X \text{ with } x \prec y\}$.*

For instance, given an AF F, we have that $max_\subset(adm(F)) = prf(F)$, i.e. the subset-maximal elements of the set $adm(F)$ are the preferred extensions of F. Clearly we also have

$max_{\subset}(com(F)) = prf(F)$. For the semi-stable and stage semantics one can use the following order which uses the range of arguments in an AF.

Definition 3.3.2. *Let $F = (A, R)$ be an AF and $S, T \subseteq A$. We define the ordering $\prec_R$ on 2^A to be $S \prec_R T$ iff $S_R^+ \subset T_R^+$.*

It is easy to show that $\prec_R$ is indeed a strict preorder.

Proposition 3.3.1. *Let $F = (A, R)$ be an AF. Then $\prec_R$ is a strict preorder.*

Proof. Let $S, T, U \subseteq A$. Since $S_R^+ \not\subset S_R^+$ we have that $\prec_R$ is irreflexive. Assume that $S \prec_R T$ and $T \prec_R U$. Then we have $S_R^+ \subset T_R^+$ and $T_R^+ \subset U_R^+$. Since $\subset$ is transitive we have $S_R^+ \subset U_R^+$ and thus $S \prec_R U$, implying that $\prec_R$ is transitive. □

As expected, we have for any AF F that the following equations hold.

Proposition 3.3.2. *Let $F = (A, R)$ be an AF. It holds that*

- $max_{\subset}(adm(F)) = max_{\subset}(com(F)) = prf(F)$;
- $max_{\prec_R}(adm(F)) = max_{\prec_R}(com(F)) = max_{\prec_R}(prf(F)) = sem(F)$; *and*
- $max_{\prec_R}(cf(F)) = stg(F)$.

Proof. Only the equation $max_{\prec_R}(prf(F)) = sem(F)$ does not follow directly from definition. We know that $prf(F) \subseteq sem(F)$. Assume $E \in max_{\prec_R}(prf(F))$. Suppose there is an $E' \in sem(F)$, s.t. $E \prec_R E'$ (which would imply $E \notin sem(F)$). Then $E' \in prf(F)$ and $E' \in max_{\prec_R}(prf(F))$. This is a contradiction to E being $\prec_R$-maximal.

Assume $E \in sem(F)$. Then $E \in prf(F)$. There is no $E' \in adm(F)$, s.t. $E \prec_R E'$. Therefore such an E' is also not present in $prf(F)$. Therefore $E \in max_{\prec_R}(prf(F))$. □

3.3.1 Generic Algorithm

In this section we present a generic version of our search algorithms. The generic algorithm makes two types of function calls, which can be implemented with a SAT-solver. These two function calls can be represented by checking if a certain set is empty, and if not computing one member of it. This naturally corresponds to a SAT call, which checks whether a formula is satisfiable and if yes, returns a model. The two sets are defined as follows

Definition 3.3.3. *Let $F = (A, R)$ be an AF, σ a semantics on AFs, $\mathcal{S} \subseteq \sigma(F)$, $X \in \sigma(F)$ and $\prec$ a strict preorder on 2^A. Define the following two subsets of σ-extensions*

- $ExtsExcl(\sigma, F, \mathcal{S}, \prec) = \{E \in \sigma(F) \mid \nexists E' \in \mathcal{S}$ *s.t.* $E \prec E'$ *or* $E = E'\}$;
- $GreaterExts(\sigma, F, X, \prec) = \{E \in \sigma(F) \mid X \prec E\}$.

The intended meaning of these sets is that both contain *candidates* to a desired solution or extension. The set $ExtsExcl(\sigma, F, \mathcal{S}, \prec)$ contains a subset of all the extensions of F w.r.t. semantics σ, i.e. $ExtsExcl(\sigma, F, \mathcal{S}, \prec) \subseteq \sigma(F)$, s.t. no extension in this set is smaller to one in $\mathcal{S}$ w.r.t. the order $\prec$ or is identical with one in $\mathcal{S}$. Thereby we can *exclude* previously computed extensions. Since our task is always to compute extensions of semantics satisfying some form of maximality, we can also exclude all candidates smaller to the ones in $\mathcal{S}$. One can view the set $\mathcal{S}$ as the extensions already computed (or visited).

On the other hand $GreaterExts(\sigma, F, X, \prec)$ contains all σ extensions of F which are *greater* than X w.r.t. $\prec$. Simply put, this allows us to iteratively find a greater candidate w.r.t. an ordering to reach a desired extension.

The next rather technical lemma shows a basic property of these two sets, which will be useful for showing termination of the following algorithms.

Lemma 3.3.3. *Let $F = (A, R)$ be an AF, σ a semantics on AFs and $\prec$ a strict preorder on 2^A. Further let (i) $\mathcal{S}, \mathcal{S}' \subseteq \sigma(F)$ with $\mathcal{S} \subset \mathcal{S}'$ s.t. there exists a $D' \in (\mathcal{S}' \setminus \mathcal{S})$ with $D' \not\prec D$ for all $D \in \mathcal{S}$ and (ii) $X, X' \in \sigma(F)$ with $X \prec X'$. It holds that*

- $ExtsExcl(\sigma, F, \mathcal{S}, \prec) \supset ExtsExcl(\sigma, F, \mathcal{S}', \prec)$*; and*
- $GreaterExts(\sigma, F, X, \prec) \supset GreaterExts(\sigma, F, X', \prec)$.

Proof. Let $ExtsExcl(\sigma, F, \mathcal{S}, \prec) = \mathcal{E}$ and $ExtsExcl(\sigma, F, \mathcal{S}', \prec) = \mathcal{E}'$. We now need to show that $\mathcal{E} \supset \mathcal{E}'$. Clearly we have $\mathcal{E}, \mathcal{E}' \subseteq \sigma(F)$. If $E' \in \mathcal{E}'$, then $E' \in \sigma(F)$ and there is no $S' \in \mathcal{S}'$ s.t. $E' \prec S'$ and $E' \notin \mathcal{S}'$. Therefore $E' \in \mathcal{E}$, since $\mathcal{S} \subset \mathcal{S}'$ and thus there is also no $S \in \mathcal{S}$ s.t. $E' \prec S$. Now consider an element $D' \in (\mathcal{S}' \setminus \mathcal{S})$ with $D' \not\prec D$ for all $D \in \mathcal{S}$, which exists due to the assumptions in the lemma. Clearly $D' \notin \mathcal{E}'$. Since there is also no $D \in \mathcal{S}$ s.t. $D' \prec D$ and also $D' \notin \mathcal{S}$, this means that $D' \in \mathcal{E}$. Thus $\mathcal{E} \supset \mathcal{E}'$.

Now let $GreaterExts(\sigma, F, X, \prec) = \mathcal{G}$ and $GreaterExts(\sigma, F, X', \prec) = \mathcal{G}'$. Let $E' \in \mathcal{G}'$. This means that $X \prec X' \prec E'$ (recall that $\prec$ is transitive). Thus clearly $E' \in \mathcal{G}$. Now it is straightforward to see that $X' \notin \mathcal{G}'$, since $X' \not\prec X'$. But since $X \prec X'$ this means that $X' \in \mathcal{G}$. Thus $\mathcal{G} \supset \mathcal{G}'$. □

This leads to our generic Algorithm 1 for computing credulous or skeptical acceptance of an argument a in an AF w.r.t. a semantics σ or enumerating all extensions from $\sigma(F)$.

Algorithm 1 is parametrized by an AF F, an argument a to query under mode M and a semantics σ. The two further parameters σ' and $\prec$ enable us to actually compute this result using the base semantics σ' to search for its maximal elements w.r.t. $\prec$. In Figure 3.4 we see an example computation of this algorithm. The ellipse represents the extensions of the base semantics σ'. In the first step (top in the figure) we find an arbitrary member of it, say E_0^0, for the first outer while loop (lines 2–10) and inner while loop (lines 3–5). Then we iteratively generate new candidates in the inner while loop, which are greater w.r.t. $\prec$. This is represented on the right side of the figure. At some point we reach a maximal element E_n^0. This is one extension in $\sigma(F)$. In the next iteration of the outer while loop, in the middle of the figure, we exclude E_n^0 and all $E \prec E_n^0$ from the search space (gray area) and repeat the process as before. Finally at the bottom of the figure we reach the final state, where we have identified all

Algorithm 1 Generic$(F, a, M, \sigma, \sigma', \prec)$

Require: AF $F = (A, R)$, argument $a \in A$, mode $M \in \{\mathsf{Enum}, \mathsf{Cred}, \mathsf{co\text{-}Skept}\}$, semantics σ and σ', $\prec$ a strict preorder on 2^A and $\sigma(F) = max_{\prec}(\sigma'(F))$

Ensure: returns $\mathcal{S} = \sigma(F)$ if $M = \mathsf{Enum}$, *yes* if $M = \mathsf{Cred}$ ($\mathsf{co\text{-}Skept}$) and $\mathsf{Cred}_\sigma(a, F) = yes$ ($\mathsf{Skept}_\sigma(a, F) = no$), otherwise *no*

1: $\mathcal{S} := \emptyset$
2: **while** $\exists E, E \in ExtsExcl(\sigma', F, \mathcal{S}, \prec)$ **do**
3: **while** $\exists E', E' \in GreaterExts(\sigma', F, E, \prec)$ **do**
4: $E := E'$
5: **end while**
6: **if** $M = \mathsf{Cred}$ and $a \in E$ or $M = \mathsf{co\text{-}Skept}$ and $a \notin E$ **then**
7: return *yes*
8: **end if**
9: $\mathcal{S} := \mathcal{S} \cup \{E\}$
10: **end while**
11: return no (or $\mathcal{S}$ if $M = \mathsf{Enum}$)

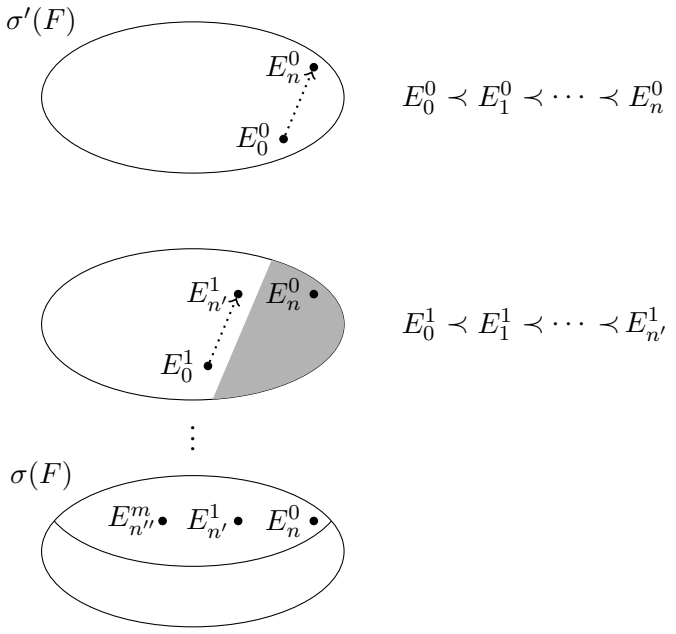

Figure 3.4: Illustration of Algorithm 1

members of $\sigma(F) = \{E_n^0, E_{n'}^1, \ldots, E_{n''}^m\}$. The correctness of this generic algorithm is shown next. Note for showing the termination of the algorithm, i.e. that it finishes computation after a finite number of steps, we apply our general assumption that AFs are finite.

Theorem 3.3.4. *Let $F = (A, R)$ be an AF, $a \in A$, $M \in \{\mathsf{Cred}, \mathsf{co\text{-}Skept}, \mathsf{Enum}\}$, σ and σ' AF semantics, $\prec$ a strict preorder on 2^A and $\sigma(F) = max_{\prec}(\sigma'(F))$. Then*

- *Generic$(F, a, M, \sigma, \sigma', \prec)$ terminates;*
- *$\mathsf{Cred}_\sigma(a, F) = yes$ iff Generic$(F, a, \mathsf{Cred}, \sigma, \sigma', \prec)$ returns yes;*
- *$\mathsf{Skept}_\sigma(a, F) = yes$ iff Generic$(F, a, \mathsf{co\text{-}Skept}, \sigma, \sigma', \prec)$ returns no;*
- *Generic$(F, a, \mathsf{Enum}, \sigma, \sigma', \prec)$ returns $\sigma(F) = \mathcal{S}$.*

Proof. We first show that Generic$(F, a, M, \sigma, \sigma', \prec)$ terminates. We assume that $|A| \geq 0$ and $|R| \geq 0$ are finite integers in our proof. W.l.o.g. we assume that $M = \mathsf{Enum}$, since for the other modes the algorithm just might terminate earlier, i.e. returning "yes". It is straightforward to see that both $ExtsExcl(\sigma', F, \mathcal{S}, \prec)$ and $GreaterExts(\sigma', F, E, \prec)$ are finite, since $\sigma'(F)$ is finite. We first show that for any σ', F, $E \in \sigma(F)$ and $\prec$ as defined in the assumptions that the "inner" while-loop in lines 3–5 terminates. This follows from Lemma 3.3.3, which shows that both sets decrease with each iteration in the corresponding while loop. To see that the assumptions of Lemma 3.3.3 are fulfilled consider for the inner while loop that if $GreaterExts(\sigma', F, E, \prec)$ is non-empty then an E' out of it is selected. We clearly have that $E \prec E'$. For the outer while loop if $ExtsExcl(\sigma', F, \mathcal{S}, \prec)$ is non-empty, then we choose an E out of it s.t. $E \not\prec D$ for any $D \in \mathcal{S}$. Clearly this also holds for the E added in Line 9 to $\mathcal{S}$. Therefore in the next iteration we have that $\mathcal{S}$ is a superset of the set in the previous iteration and contains one member, which is not smaller w.r.t. $\prec$ or equal than all its members from the previous iteration. This implies that both loops terminate since both sets start in each loop with a finite cardinality, which decreases by at least one in each step and the loops terminate if the corresponding sets are empty.

We prove now the last claim, since the decision problems follow straightforwardly. Assume again $M = \mathsf{Enum}$. We now have to show that Generic$(F, a, \mathsf{Enum}, \sigma, \sigma', \prec)$ returns $\sigma(F) = \mathcal{S} = max_{\prec}(\sigma'(F))$. There are three cases to consider.

- The trivial case occurs if $\sigma'(F) = \emptyset$. Then $\sigma(F)$ is also empty and the outer while loop terminates immediately and we return $\mathcal{S} = \emptyset$.
- Assume the algorithm terminated and $X \in \mathcal{S}$ with $\sigma'(F) \neq \emptyset$. We now show that then also $X \in \sigma(F)$. Clearly if $\sigma'(F)$ is non-empty so is $ExtsExcl(\sigma', F, \emptyset, \prec)$. Then we can conclude that $X \in \sigma'(F)$, since it must be either in $GreaterExts(\sigma', F, E, \prec)$ for some E or in $ExtsExcl(\sigma', F, \mathcal{S}', \prec)$ for some $\mathcal{S}'$. Suppose there exists an $X' \in \sigma'(F)$ such that $X \prec X'$. This is a counterexample that $X \in \sigma(F)$. Then the inner while loop did not terminate, since we have that $GreaterExts(\sigma', F, X, \prec)$ is not empty. Thus it follows that $X \in \sigma(F)$.

- Now assume the algorithm terminated and returned $\mathcal{S}$ and $X \in \sigma(F)$. We show that X is also in $\mathcal{S}$. It is easy to see that $X \in \mathit{ExtsExcl}(\sigma', F, \emptyset, \prec)$. Suppose that $X \notin \mathcal{S}$. But then $\mathit{ExtsExcl}(\sigma', F, \mathcal{S}, \prec)$ cannot be empty, since there is no element $X' \in \sigma'(F)$ s.t. $X \prec X'$. Thus there does not exist a $D \in \mathcal{S}$ s.t. $X \prec D$ and we can derive that $X \in \mathit{ExtsExcl}(\sigma', F, \mathcal{S}, \prec)$. Therefore the algorithm cannot have terminated.

Since $\sigma(F) = \mathcal{S}$ if $M = \mathsf{Enum}$, it follows that for any $E \in \sigma(F)$ we enter the outer while loop once. If this E solves the question of credulous acceptance of an argument a positively, then we exit in Line 7 if $M = \mathsf{Cred}$. The case for co-skeptical acceptance is analogous. In all the other cases we return "no" as specified. □

Considering that all computations in Algorithm 1 are easy to compute, except for the two membership checks in Line 2 and Line 3, it is useful to calculate how many of these checks are required in the worst case. We view these two lines as *function calls* and next show the relation of the given AF and the number of required function calls. Note that we assume here a concrete strict preorder for our extensions: either $\subset$ or $\prec_R$. The reason for this choice is that arbitrary preorders may have exponentially many improvement steps.

Proposition 3.3.5. *Let $F = (A, R)$ be an AF, $a \in A$, σ and σ' AF semantics, $\prec \in \{\subset, \prec_R\}$ and $\sigma(F) = max_{\prec}(\sigma'(F))$. The number of function calls (Line 2 and Line 3 of Algorithm 1) of $\mathit{Generic}(F, a, M, \sigma, \sigma', \prec)$ for $M \in \{\mathsf{Cred}, \mathsf{co\text{-}Skept}, \mathsf{Enum}\}$ is $O(|A| \cdot |\sigma(F)|)$.*

Proof. The proposition follows from the proof of Theorem 3.3.4. Simply consider that the outer while loop is entered at most $|\sigma(F)|$ times and the inner while loop must terminate after at most $|A|$ iterations, since if $E \subset E'$, then $|E| < |E'|$ and if $E \prec_R E'$, then $|E_R^+| < |E_R'^+|$ and in both cases at most $|A|$ arguments can be added to the set or the range of a set. □

The upper bound for the running time w.r.t. the function calls also holds for concrete instantiations of Algorithm 1 for preferred semantics in Section 3.3.2 and semi-stable and stage semantics in Section 3.3.3. Thus if $F \in \mathsf{sol}_\sigma^k$, for a fixed integer k, then we have a Δ_2^P procedure for solving credulous or skeptical reasoning under semantics σ, since $|A| \cdot k$ is polynomial w.r.t. the size of the AF and k.

3.3.2 Search Algorithms for Preferred Semantics

We now instantiate the generic Algorithm 1 for preferred semantics. By looking at Proposition 3.3.2 we can infer two base semantics for preferred semantics, admissible sets and complete extensions. For the relation we simply require a subset relation between extensions. This leads to Algorithm 2. We now show how this algorithm fares on our running Example 3.2.1.

Example 3.3.1. *Consider the AF from Example 3.2.1. We show the execution of Algorithm 2 for admissible sets as the base semantics. After initializing $\mathcal{S}$ to the empty set we come to Line 2 (the outer while loop) for the first time. Then $\mathit{ExtsExcl}(adm, F, \emptyset, \subset) = adm(F) = \{\emptyset, \{a\}, \{b\}, \{a, d\}, \{b, d\}\}$. The algorithm itself is defined in a non-deterministic way, i.e. it may return any member of this set. Assume that $E = \emptyset$ was selected. Then $\mathit{GreaterExts}(adm, F, E, \subset) = adm(F) \setminus \{\emptyset\}$. In a subsequent step we might choose $\{a\}$ and afterwards $\{a, d\}$. This set cannot*

Algorithm 2 Preferred(F, a, M, σ')

Require: AF $F = (A, R)$, argument $a \in A$, mode $M \in \{\mathsf{Enum}, \mathsf{Cred}, \mathsf{co\text{-}Skept}\}$, semantics $\sigma' \in \{adm, com\}$

Ensure: returns $\mathcal{S} = prf(F)$ if $M = \mathsf{Enum}$, *yes* if $M = \mathsf{Cred}$ ($\mathsf{co\text{-}Skept}$) and $\mathsf{Cred}_{prf}(a, F) = yes$ ($\mathsf{Skept}_{prf}(a, F) = no$), otherwise *no*

1: $\mathcal{S} := \emptyset$
2: **while** $\exists E, E \in ExtsExcl(\sigma', F, \mathcal{S}, \subset)$ **do**
3: **while** $\exists E', E' \in GreaterExts(\sigma', F, E, \subset)$ **do**
4: $E := E'$
5: **end while**
6: **if** $M = \mathsf{Cred}$ and $a \in E$ or $M = \mathsf{co\text{-}Skept}$ and $a \notin E$ **then**
7: return *yes*
8: **end if**
9: $\mathcal{S} := \mathcal{S} \cup \{E\}$
10: **end while**
11: return no (or $\mathcal{S}$ if $M = \mathsf{Enum}$)

be extended to a superset, that is also admissible, thus we add it to $\mathcal{S}$. In the next iteration we have $ExtsExcl(adm, F, \mathcal{S}, \subset) = \{\{b\}, \{b, d\}\}$. We may now choose $\{b\}$ as our next candidate and extend it to $\{b, d\}$. We end up with $\mathcal{S} = prf(F) = \{\{a, d\}, \{b, d\}\}$.

Using the complete semantics as our base semantics we could have reduced the number of steps, since $\{a\}$ and $\{b\}$ are not complete sets. Thus choosing complete semantics as the base semantics might reduce the overall search space.

As a direct corollary to Theorem 3.3.4 from the generic results and Proposition 3.3.2, we can deduce the correctness of Algorithm 2 for preferred semantics.

Corollary 3.3.6. *Let $F = (A, R)$ be an AF, $a \in A$, $M \in \{\mathsf{Cred}, \mathsf{co\text{-}Skept}, \mathsf{Enum}\}$, $\sigma' \in \{adm, com\}$. Then*

- *Preferred(F, a, M, σ') terminates;*
- *$\mathsf{Cred}_{prf}(a, F) = yes$ iff Preferred$(F, a, \mathsf{Cred}, \sigma')$ returns yes;*
- *$\mathsf{Skept}_{prf}(a, F) = yes$ iff Preferred$(F, a, \mathsf{co\text{-}Skept}, \sigma')$ returns no;*
- *Preferred$(F, a, \mathsf{Enum}, \sigma')$ returns $prf(F) = \mathcal{S}$.*

Having established the algorithm for preferred semantics we now delegate the computationally hard subtasks to a SAT-solver by simply letting such a solver compute the membership checks in Line 2 and Line 3. For this to work we turn our attention first to encodings of these sets in Boolean logic.

We begin with encodings of conflict-free sets and admissible sets, which have been presented in [21]. In general we require that the set of σ-extensions are in a 1-to-1 correspondence to the models of a constructed formula. By 1-to-1 correspondence we mean that if E is a σ-extension

for an AF $F = (A, R)$, then I is a model of the formula s.t. $I \cap A = E$. That is, we might have to "project out" auxiliary variables used in the formula.

$$cf_{A,R} = \bigcup_{(a,b)\in R} \{(\neg a \vee \neg b)\} \tag{3.1}$$

The formula for conflict-free sets simply states that a set (a model) is conflict-free if for each attack in the AF only one of its arguments can be accepted (are true in the model). Very similarly we can define a formula for admissible sets as follows. Here an argument c is defended if each attacker is attacked by an accepted argument.

$$adm_{A,R} = cf_{A,R} \cup \bigcup_{(b,c)\in R} \{(\neg c \vee \bigvee_{(a,b)\in R} a)\} \tag{3.2}$$

For the complete extensions there is also a formula shown in [21], but since we want to directly use SAT-solvers, we utilize a different encoding here, namely one which is close to CNF, i.e. either is in CNF or contains instead of clauses simple implications, which can be directly rewritten to a formula in CNF. Here we have auxiliary variables, denoting the range, i.e. we have that $atoms(range_{A,R}) = A \cup \overline{A}$ for an AF $F = (A, R)$. Intuitively if $\overline{a}$ is true in a model, then the argument a is in the range of the corresponding extension. Recall that, given an AF $F = (A, R)$ the range of a set $S \subseteq A$ is defined as $S^+_R = S \cup \{a \in A \mid (b, a) \in R \text{ and } b \in S\}$.

$$range_{A,R} = \bigcup_{a\in A} \{(\overline{a} \rightarrow (a \vee \bigvee_{(b,a)\in R} b))\} \cup \bigcup_{(b,a)\in R} \{(\overline{a} \leftarrow b), (\overline{a} \leftarrow a)\} \tag{3.3}$$

That is, if $\overline{a}$ is true under an interpretation I, with the intended meaning that a is in the range of $I \cap A$, i.e. $a \in (I \cap A)^+_R$, then also either a must be true in I, or one of its attackers. This is reflected in the first clause. The other clauses form the other direction, i.e. if a or one of its attackers is true, then a must be in the range. We formalize this notion in the following lemma.

Lemma 3.3.7. *Let $F = (A, R)$ be an AF and $S \subseteq A$. Then*

- *there exists exactly one interpretation I with $dom(I) = atoms(range_{A,R})$, s.t. $I \cap A = S$ and $I \models range_{A,R}$; and*
- *$a \in S^+_R$ iff there exists an I, s.t. $I \cap A = S$ with $I(\overline{a}) = \mathbf{t}$ and $I \models range_{A,R}$.*

Proof. We first show the second item. Assume $a \in S^+_R$. We need to show that then there is a two-valued interpretation I s.t. $I \cap A = S$ and $I \models range_{A,R}$ with $I(\overline{a}) = \mathbf{t}$. We construct this interpretation by $I = S \cup \overline{S^+_R}$, i.e. $I = S \cup \{\overline{x} \mid x \in S^+_R\}$. Clearly we have $\overline{a} \in I$. Consider an arbitrary $x \in A$. If $\overline{x} \in I$, then $x \in S^+_R$. Thus either $x \in S$ or an attacker of x is in S. This means that either $x \in I$ or one attacker of it is in I. Thus $I \models \{\overline{x} \rightarrow (x \vee \bigvee_{(b,x)\in R} b)\}$. If $x \in S$, then also $\overline{x} \in I$. Thus $I \models \{\overline{x} \leftarrow x\}$. Also $I \models \bigcup_{(b,x)\in R} \{\overline{x} \leftarrow b\}$, since if an attacker of x is in S (and thus in I), we have that also $x \in S^+_R$ and thus $\overline{x} \in I$.

For the other direction consider a two-valued interpretation I s.t. $\overline{a} \in I$ and $I \models range_{A,R}$. This means that $a \vee \bigvee_{(b,a)\in R} b$ evaluates to true under I. Thus there is an element of $\{a\} \cup \{b \mid$

$(b, a) \in R\}$ which is in I. Since this element is also in $I \cap A = S$ this means that either a or one attacker of it is in S. Thus $a \in S_R^+$.

Finally to show that there exists exactly one I, s.t. $I \cap A = S$ and $I \models range_{A,R}$, consider again $I = S \cup \overline{S_R^+}$. Assume $I \models range_{A,R}$. Now suppose there exists an $I' \neq I$ s.t. $I' \cap A = S$, $dom(I') = atoms(range_{A,R})$ and $I' \models range_{A,R}$. Then there is an $\overline{a}$ for $a \in A$, for which we have $I(\overline{a}) \neq I'(\overline{a})$. We have two cases. First assume that $I(\overline{a}) = \mathbf{t}$. Then since $I \models range_{A,R}$ we have $(\{a\} \cup \bigcup_{(b,a)\in R} b) \cap S \neq \emptyset$, therefore I' does not satisfy $(\overline{a} \leftarrow a)$ or $(\overline{a} \leftarrow b)$ for all attackers b of a. The other case occurs if $I(\overline{a}) = \mathbf{f}$. Then $(\{a\} \cup \bigcup_{(b,a)\in R} b) \cap S = \emptyset$ and since $I'(\overline{a}) = \mathbf{t}$ in this case we have $I' \not\models (\overline{a} \rightarrow (a \vee \bigvee_{(b,a)\in R} b))$. Thus there is only one two-valued interpretation I defined only on $atoms(range_{A,R})$, s.t. $I \cap A = S$ and $I \models range_{A,R}$. □

Using the range we can define a formula for complete extensions as follows.

$$com_{A,R} = \ adm_{A,R} \cup range_{A,R} \cup \bigcup_{a\in A} \{(\overline{a} \leftarrow \bigwedge_{(b,a)\in R} \overline{b})\} \tag{3.4}$$

We summarize the correctness of the encodings in the following proposition.

Proposition 3.3.8. *Let $F = (A, R)$ be an AF. Then it holds that*

- $cf(F) = \{I^{\mathbf{t}} \cap A \mid I \models cf_{A,R}\}$;
- $adm(F) = \{I^{\mathbf{t}} \cap A \mid I \models adm_{A,R}\}$; *and*
- $com(F) = \{I^{\mathbf{t}} \cap A \mid I \models com_{A,R}\}$.[1]

Proof. The first two items follow directly from [21, Proposition 6]. We now show the third item. Assume that $E \in com(F)$. We now have to show that there exists an I s.t. $I \models com_{A,R}$ and $I \cap A = E$. Since $E \in adm(F)$, it follows that $E \models adm_{A,R}$. Due to Lemma 3.3.7 we know that $E \cup \overline{X} \models adm_{A,R} \wedge range_{A,R}$ for $X = E_R^+$. Consider now an arbitrary $x \in A$. If all of its attackers are in E_R^+, then clearly x must be in E_R^+, since either one of its attackers is in E, thus x must be in E_R^+ or otherwise if all attackers of x are in $E_R^+ \setminus E$, then since E is complete also x must be in E and thus in E_R^+. This means $E \cup \overline{X} \models \overline{x} \leftarrow \bigwedge_{(b,x)\in R} \overline{b}$ and thus also $E \cup \overline{X} \models com_{A,R}$.

The other direction is similar. Assume that $I \models com_{A,R}$. This means that $I \cap A = E$ and $E \in adm(F)$ by [21]. Now to see that E is complete in F, we have to show that if an $a \in A$ is defended by E, then also $a \in E$. An argument a is defended by E iff all its attackers are in $E_R^+ \setminus E$, i.e. attacked by E. If this is the case then the set of attackers of a are in E_R^+ and by the formula $com_{A,R}$ we then require that $a \in E_R^+$. Since all attackers are not in E, but in the range of E, this means that a must be in E. □

Now to use a SAT-solver for Algorithm 2, we introduce formulae representing the two sets $ExtsExcl$ and $GreaterExts$. By $\sigma'_{A,R} \in \{adm_{A,R}, com_{A,R}, cf_{A,R}\}$ we denote the formula w.r.t. semantics σ' for an AF $F = (A, R)$.

[1] We assume that the interpretations are defined on A, i.e. $A \subseteq dom(I)$. E.g. if an argument is not attacked in the AF, then it does not occur in $adm_{A,R}$, but we assume that it is in $dom(I)$.

Proposition 3.3.9. *Let $F = (A, R)$ be an AF, $\sigma' \in \{adm, com\}$, $\mathcal{S} \subseteq 2^A$ and $X \subseteq A$. Then it holds that*

- $ExtsExcl(\sigma', F, \mathcal{S}, \subset) = \{I \cap A \mid I \models \sigma'_{A,R} \cup \bigcup_{E \in \mathcal{S}} \{(\bigvee_{a \in (A \setminus E)} a)\}\}$;
- $GreaterExts(\sigma', F, X, \subset) = \{I \cap A \mid I \models \sigma'_{A,R} \cup \bigcup_{a \in (A \cap X)} \{a\} \cup \{(\bigvee_{a \in (A \setminus X)} a)\}\}$

Proof. Due to Proposition 3.3.8 we know that $\sigma'(F) = \{I \cap A \mid I \models \sigma'_{A,R}\}$. Assume $E \in ExtsExcl(\sigma', F, \mathcal{S}, \subset)$. Then $E \in \sigma'(F)$. Therefore there is an interpretation I s.t. $I \cap A = E$ and $I \models \sigma'_{A,R}$. Consider an arbitrary $E' \in \mathcal{S}$. We have that $E \not\subseteq E'$. Therefore there exists an $a \in E$ s.t. $a \notin E'$. This implies that $E \models \bigcup_{E' \in \mathcal{S}} \{\bigvee_{a \in (A \setminus E')} a\}$ and thus I is a model of the formula as stated.

For the other direction assume that $I \models \sigma'_{A,R} \wedge \bigcup_{E' \in \mathcal{S}} \{\bigvee_{a \in (A \setminus E')} a\}$. Again, we have due to Proposition 3.3.8 that $I \cap A = E$ and $E \in \sigma'(F)$. Now suppose that there exists an $E' \in \mathcal{S}$ s.t. $E \subseteq E'$. There is a corresponding clause c in the formula: $\bigvee_{a \in A \setminus E'} a$. Clearly $E \not\models c$. This is a contradiction. Thus there is no such $E' \in \mathcal{S}$ and we have that $E \in ExtsExcl(\sigma', F, \mathcal{S}, \subset)$.

For the second equation the proof is very similar. If $E \in GreaterExts(\sigma', F, X, \subset)$, then $E \in \sigma'(F)$ and $X \subset E$. Then there is an interpretation I s.t. $I \cap A = E$ and $I \models \sigma'_{A,R}$. Further $I \models \bigwedge_{a \in (A \cap X)} a$, since $X \subseteq E$. To see that we have that $I \models \bigvee_{a \in (A \setminus X)} a$, consider an $a \in E \setminus X$, which exists by assumption that $X \subset E$. The other direction proceed analogous as in the previous proof for the first equation. □

This leads us to the SAT based algorithm for preferred semantics in Algorithm 3. Algorithm 3 inherits the properties of the more abstract Algorithm 2 (Corollary 3.3.6 and Proposition 3.3.5).

Algorithm 3 Preferred-SAT(F, a, M, σ')

Require: AF $F = (A, R)$, argument $a \in A$, mode $M \in \{\mathsf{Enum}, \mathsf{Cred}, \mathsf{co\text{-}Skept}\}$, semantics $\sigma' \in \{adm, com\}$

Ensure: returns $\mathcal{I} = prf(F)$ if $M = \mathsf{Enum}$, *yes* if $M = \mathsf{Cred}$ ($\mathsf{co\text{-}Skept}$) and $\mathsf{Cred}_{prf}(a, F) = yes$ ($\mathsf{Skept}_{prf}(a, F) = no$), otherwise *no*

```
excl := ∅
I := ∅
while ∃I, I ⊨ σ'_{A,R} ∪ excl do
    while ∃J, J ⊨ σ'_{A,R} ∪ ⋃_{a∈(A∩I)} {a} ∪ {(⋁_{a∈(A\I)} a)} do
        I := J
    end while
    if M = Cred and a ∈ I or M = co-Skept and a ∉ I then
        return yes
    end if
    excl := excl ∪ {(⋁_{a∈(A\I)} a)}
    I := I ∪ {I ∩ A}
end while
return no (or I if M = Enum)
```

3.3.3 Search Algorithms for Semi-stable and Stage Semantics

The generic algorithm can be easily instantiated for semi-stable and stage semantics. We show here the case for semi-stable semantics in Algorithm 4. The required change for stage semantics is to simply use conflict-free sets as the base semantics instead of admissible sets or complete extension as is the case for semi-stable semantics. For the ordering we use $\prec_R$ for both semi-stable and stage semantics. We show the stage variant in the Appendix in Algorithm 15.

Algorithm 4 Semi-stable(F, a, M, σ')

Require: AF $F = (A, R)$, argument $a \in A$, mode $M \in \{\mathsf{Enum}, \mathsf{Cred}, \mathsf{co\text{-}Skept}\}$, semantics $\sigma' \in \{adm, com\}$
Ensure: returns $\mathcal{S} = sem(F)$ if $M = \mathsf{Enum}$, *yes* if $M = \mathsf{Cred}$ ($\mathsf{co\text{-}Skept}$) and $\mathsf{Cred}_{sem}(a, F) = yes$ ($\mathsf{Skept}_{sem}(a, F) = no$), otherwise *no*

```
1: S := ∅
2: while ∃E, E ∈ ExtsExcl(σ', F, S, ≺_R) do
3:    while ∃E', E' ∈ GreaterExts(σ', F, E, ≺_R) do
4:       E := E'
5:    end while
6:    if M = Cred and a ∈ E or M = co-Skept and a ∉ E then
7:       return yes
8:    end if
9:    S := S ∪ {E}
10: end while
11: return no (or S if M = Enum)
```

We give an example for the algorithm for the semi-stable semantics.

Example 3.3.2. *For the AF F in Example 3.2.1, we have the semi-stable extension $sem(F) = \{\{a, d\}\}$. Assume we choose complete semantics as the base semantics. We begin with $E = \emptyset$ in the first iteration of the loops. Now we extend it to $\{b, d\}$. Clearly $\emptyset_R^+ = \emptyset \prec_R \{b, d\}_R^+ = \{a, b, c, d\}$. In contrast to the algorithm for preferred semantics, we can iterate a third time in the inner while loop and have finally $\{a, d\}$. Regarding the ordering w.r.t. the range we have $\{b, d\}_R^+ \prec_R \{a, d\}_R^+ = \{a, b, c, d, e\}$.*

Again, as for the preferred semantics, we directly obtain the correctness for this algorithm from Theorem 3.3.4 and Proposition 3.3.2, which we summarize in the following corollary.

Corollary 3.3.10. *Let $F = (A, R)$ be an AF, $a \in A$, $M \in \{\mathsf{Cred}, \mathsf{co\text{-}Skept}, \mathsf{Enum}\}$, $\sigma' \in \{adm, com\}$. Then*

- *Semi-stable(F, a, M, σ') terminates;*
- *$\mathsf{Cred}_{sem}(a, F) = yes$ iff Semi-stable$(F, a, \mathsf{Cred}, \sigma')$ returns yes;*
- *$\mathsf{Skept}_{sem}(a, F) = yes$ iff Semi-stable$(F, a, \mathsf{co\text{-}Skept}, \sigma')$ returns no;*
- *Semi-stable$(F, a, \mathsf{Enum}, \sigma')$ returns $sem(F) = \mathcal{S}$.*

Similarly one can directly show the properties for the stage algorithm.

Corollary 3.3.11. *Let $F = (A, R)$ be an AF, $a \in A$, $M \in \{\mathsf{Cred}, \mathsf{co\text{-}Skept}, \mathsf{Enum}\}$. Then*

- *Stage(F, a, M, σ') terminates;*
- $\mathsf{Cred}_{stg}(a, F) = yes$ *iff Stage$(F, a, \mathsf{Cred}, \sigma')$ returns yes;*
- $\mathsf{Skept}_{stg}(a, F) = yes$ *iff Stage$(F, a, \mathsf{co\text{-}Skept}, \sigma')$ returns no;*
- *Stage$(F, a, \mathsf{Enum}, \sigma')$ returns $stg(F) = \mathcal{S}$.*

The SAT based procedure is shown in Algorithm 5 for semi-stable semantics. The formula in Line 4 uses the same idea as for the preferred semantics, we just have to consider the range instead of the arguments themselves. The formula in Line 10 is a bit different. The formula consists of two conjuncts, which are conjoined with $excl$. The left conjunct consists of a single clause and the right of a set of clauses. Let $I = X \cup \overline{X}$. An interpretation $J = S \cup \overline{S}$ satisfies the right conjunct if one of three cases occur. The right conjunct is satisfied by J if (i) $\overline{X} = \overline{S}$, (ii) $\overline{X} \subset \overline{S}$, or (iii) $\overline{X}$ and $\overline{S}$ are incomparable. The left conjunct is satisfied in cases (ii) and (iii). The second and third case can only occur if $X \subset S$, or X and S are incomparable respectively. In the first case the left conjunct ensures that $X \neq S$. In other words, this formula ensures that every interpretation J that satisfies both conjuncts represents either a set of arguments not equal to the one encoded in I but with equal range, or otherwise a set of arguments with a greater or incomparable range to the one encoded in I.

Algorithm 5 Semi-stable-SAT(F, a, M, σ')

Require: AF $F = (A, R)$, argument $a \in A$, mode $M \in \{\mathsf{Enum}, \mathsf{Cred}, \mathsf{co\text{-}Skept}\}$, semantics $\sigma' \in \{adm, com\}$
Ensure: returns $\mathcal{I} = sem(F)$ if $M = \mathsf{Enum}$, *yes* if $M = \mathsf{Cred}$ ($\mathsf{co\text{-}Skept}$) and $\mathsf{Cred}_{sem}(a, F) = yes$ ($\mathsf{Skept}_{sem}(a, F) = no$), otherwise *no*

1: $excl := \top$
2: $\mathcal{I} := \emptyset$
3: **while** $\exists I, I \models \sigma'_{A,R} \wedge range_{A,R} \wedge excl$ **do**
4: **while** $\exists J, J \models \sigma'_{A,R} \wedge range_{A,R} \wedge \bigwedge_{\overline{a} \in (\overline{A} \cap I)} \overline{a} \wedge \bigvee_{\overline{x} \in (\overline{A} \setminus I)} \overline{x}$ **do**
5: $I := J$
6: **end while**
7: **if** $M = \mathsf{Cred}$ and $a \in I$ or $M = \mathsf{co\text{-}Skept}$ and $a \notin I$ **then**
8: return *yes*
9: **end if**
10: $excl := excl \wedge \bigvee_{a \in (A \setminus I)} a \wedge \bigwedge_{\overline{a} \in (\overline{A} \cap I)} ((\bigwedge_{\overline{x} \in (\overline{A} \setminus I)} \neg \overline{x}) \rightarrow \overline{a})$
11: $\mathcal{I} := \mathcal{I} \cup \{I \cap A\}$
12: **end while**
13: return no (or $\mathcal{I}$ if $M = \mathsf{Enum}$)

Again the only change required for the stage semantics is to allow for the base semantics only conflict-free sets. We show the variant for the stage semantics in the Appendix in Algorithm 16.

The correctness of the encodings for semi-stable semantics is shown next. We make use of the fact that in the "solution" set $\mathcal{S}$ only semi-stable extensions are present.

Proposition 3.3.12. *Let $F = (A, R)$ be an AF, $\sigma' \in \{adm, com\}$, $\mathcal{S} \subseteq sem(F)$ and $X \subseteq A$. Further let*

$$\phi = \sigma'_{A,R} \wedge range_{A,R} \wedge \bigwedge_{E \in \mathcal{S}} \Big(\bigvee_{a \in (A \setminus E)} a \wedge \bigwedge_{a \in E_R^+} \big(\big(\bigwedge_{x \in (A \setminus E_R^+)} \neg \overline{x} \big) \rightarrow \overline{a} \big) \Big)$$

$$\psi = \sigma'_{A,R} \wedge range_{A,R} \wedge \bigwedge_{a \in X_R^+} \overline{a} \wedge \bigvee_{x \in (A \setminus X_R^+)} \overline{x}$$

Then it holds that

- $ExtsExcl(\sigma', F, \mathcal{S}, \prec_R) = \{I \cap A \mid I \models \phi\}$;
- $GreaterExts(\sigma', F, X, \prec_R) = \{I \cap A \mid I \models \psi\}$

Proof. Let $D \in ExtsExcl(\sigma', F, \mathcal{S}, \prec_R)$. We now have to show that there is an I with $I \cap A = D$ and $I \models \phi$. Construct $I = D \cup \overline{D_R^+}$. We have that $I \models \sigma'_{A,R}$ due to Proposition 3.3.8. Consider an arbitrary $D' \in \mathcal{S}$. Since it holds that $D' \in sem(F)$, we have that if there is a $D'' \in \sigma'(F)$ with $D'' \subset D'$, then also $D'' \prec_R D'$. Therefore such a D'' is not in $ExtsExcl(\sigma', F, \mathcal{S}, \prec_R)$. This means that it holds that $D \not\subseteq D'$. Therefore $I \models \bigvee_{a \in (A \setminus D')} a$. Because we have $D \not\prec_R D'$ we know that $D_R^+ \not\subset D_R'^+$. This means that either $D_R^+ = D_R'^+$ or D_R^+ and $D_R'^+$ are incomparable w.r.t. $\subseteq$ (since $D' \in sem(F)$ there is no element in $\sigma'(F)$ with a greater range). Consider the first case, i.e. $D_R^+ = D_R'^+$. Then $I \models \bigwedge_{a \in D_R'^+} (\bigwedge_{x \in (A \setminus D_R'^+)} \neg \overline{x}) \rightarrow \overline{a}$, since all antecedents of the implications are satisfied as well as the conclusion. In the second case we have that there is a $y \in D_R^+$ s.t. $y \notin D_R'^+$. Therefore $I \not\models \bigwedge_{x \in (A \setminus D_R'^+)} \neg \overline{x}$. This means the implications are trivially satisfied, i.e. $I \models \bigwedge_{a \in D_R'^+} (\bigwedge_{x \in (A \setminus D_R'^+)} \neg \overline{x}) \rightarrow \overline{a}$. Thus $I \models \phi$.

For the other direction, assume that $I \models \phi$. Due to Proposition 3.3.8, we have that $I \cap A = E$ and $E \in \sigma'(F)$. Consider an $E' \in \mathcal{S}$. We have to show that $E \neq E'$ and that $E_R^+ \not\subset E_R'^+$. Since $E \models \bigcup_{E' \in \mathcal{S}} (\bigvee_{a \in (A \setminus E')} a)$, we know that $E \neq E'$. Furthermore we have that $I \models \bigcup_{E' \in \mathcal{S}} (\bigwedge_{a \in E_R'^+} ((\bigwedge_{x \in (A \setminus E_R'^+)} \neg \overline{x}) \rightarrow \overline{a}))$. Consider again two cases. First assume that $I \models (\bigwedge_{x \in (A \setminus E_R'^+)} \neg \overline{x})$. Then also $\overline{a} \in I$ for all $a \in E_R'^+$. This implies that $E_R^+ = E_R'^+$. In the second case these implications are all trivially satisfied, since the antecedents are false. Then there is an $x \in E_R^+$ s.t. $x \notin E_R'^+$. This means that $E \not\prec_R E'$.

The proof for the second item follows the proof of Proposition 3.3.9, just consider the range atoms instead of the atoms in A. □

3.3.4 Variants for Query Based Reasoning

In Section 3.3.1 we have shown a generic search algorithm applicable for the preferred, semi-stable and stage semantics under credulous and skeptical reasoning modes, as well as showing how to enumerate all extensions. For the query-based reasoning modes, i.e. credulous and skeptical reasoning, we can apply some modifications, which are of heuristic nature, to potentially

improve the overall performance. We show in this section how this can be done for a generic algorithm using SAT, which follows some ideas of the generic Algorithm 1.

Consider we want to solve credulous or skeptical reasoning w.r.t. an AF $F = (A, R)$ and an argument a under a semantics σ. Clearly we just need a witness σ-extension E, with $a \in E$ or $a \notin E$, respectively for the credulous or skeptical query. In the generic algorithm we iteratively construct extensions independently of this mode. Since we can enumerate all extensions we will find a witness if it exists. However we can try to "guide" the algorithm in choosing an extension candidate. Consider the skeptical reasoning problem (for credulous reasoning the idea is analogous). We can force the algorithm, via Boolean constraints, to consider only candidates from the base semantics σ' not containing a and then iteratively construct greater candidates with this property. At some point we reach a candidate E and cannot construct a greater candidate E' for which it holds that $a \notin E'$. This current candidate E is then maximal w.r.t. the ordering *and* having the property that a is not accepted. To decide whether $E \in \sigma(F)$, we check if there exists an $E'' \in \sigma'(F)$, s.t. $E \prec E''$. We allow a to be inside E''. If such an E'' exists, then $E \notin \sigma(F)$ and no greater extension from the base semantics than E is a witness that a is not skeptically accepted. Thus we exclude E and all smaller sets from the search space and search for a fresh candidate. Otherwise if such an E'' does not exist, then $E \in \sigma(F)$. Since $a \notin E$, this means that $\mathsf{Skept}_\sigma(a, F) = no$, which answers our decision problem. The benefit of this heuristic is that we might already exclude candidates which contain a. Furthermore we do not have to "climb" up a chain of candidates to the maximal one. We just need to find a maximal extension of the base semantics not containing a. This idea is shown in Algorithm 6.

This procedure works for preferred, semi-stable and stage semantics and for credulous and skeptical reasoning. Since credulous reasoning for preferred semantics can be decided compactly with one Boolean query (recall that the corresponding decision problem is in NP) we omit it here. In the algorithms we make use of the *accept* or *reject* functions, which simply terminate the algorithm (even if called in a sub-procedure) and return "yes" or "no" respectively.

Overall, the algorithm works as follows. Depending on the reasoning mode, we test whether there is an extension containing a, or whether there is an extension not containing a, hence accepting a credulously or rejecting it for skeptical reasoning. This is encoded via the query-literal q. In line 6, a formula is built to encode extensions of the base semantics, i.e. not taking maximality into account, together with the query q as well as semantics-specific *shortcuts* that can be applied for pruning the search space via learning inferred information; this will be discussed below in more detail. The Shortcuts function allows for refining the base encoding using the inferred information it outputs. The loop in line 7 follows the ideas of the outer loops in the generic Algorithm 1: starting with a model that corresponds to a set of arguments satisfying the base semantics and our query q, each iteration then extends this set to a larger one satisfying q until we have a maximal set satisfying q. In line 11, the condition q is dropped for testing whether the set is maximal among all sets, i.e. whether it is an extension. If this is the case, the algorithm accepts. Otherwise, we learn that none of the smaller sets can be an extension (line 14). Finally, after excluding all sets satisfying the base semantics and q from being a valid extension, the algorithm rejects the query (line 17).

For the preferred semantics we use the following formulae. Recall that for our base seman-

Algorithm 6 Decide$(F, a, M, \sigma, \sigma')$

Require: AF $F = (A, R)$, argument $a \in A$, base-semantics $\sigma' \in \{adm, com\}$ for $\sigma \in \{prf, sem\}$; $\sigma' = cf$ for $\sigma = stg$; and reasoning mode $M = \{\mathsf{co\text{-}Skept}, \mathsf{Cred}\}$;

Ensure: *accept*s the input iff $M = \mathsf{co\text{-}Skept}$ and a is skeptically rejected or $M = \mathsf{Cred}$ and a is credulously accepted in F w.r.t. σ, *reject*s otherwise

1: **if** $M = \mathsf{co\text{-}Skept}$ **then**
2: $q := \neg a$
3: **else if** $M = \mathsf{Cred}$ **then**
4: $q := a$
5: **end if**
6: $\phi := \sigma'_{A,R} \wedge q \wedge \text{Shortcuts}(F, a, M, \sigma, \sigma')$
7: **while** $\exists I, I \models \phi$ **do**
8: **while** $\exists J, J \models \psi^I_\sigma(A, R) \wedge q$ **do**
9: $I \leftarrow J$
10: **end while**
11: **if** $\not\models \psi^I_\sigma(A, R)$ **then**
12: *accept*
13: **else**
14: $\phi \leftarrow \phi \wedge \gamma^I_\sigma$
15: **end if**
16: **end while**
17: *reject*

tics we may choose $\sigma' \in \{adm, com\}$. These are the same formulae as used in the Algorithm 3.

$$\begin{aligned} \psi^I_{prf}(A, R) &= \sigma'_{A,R} \wedge \bigwedge_{a \in I \cap A} a \wedge \Big(\bigvee_{a \in A \setminus I} a \Big) \\ \gamma^I_{prf} &= \bigvee_{a \in A \setminus I} a. \end{aligned}$$

For the semi-stable and stage semantics we make use of the following formulae. Again the base semantics for semi-stable semantics are admissible sets or complete extensions, whereas stage semantics requires conflict-free sets. That is, $\sigma \in \{sem, stg\}$ and the base semantics σ' can be taken from $\{adm, com\}$ for semi-stable semantics and $\sigma' = cf$ for stage semantics.

$$\begin{aligned} \psi^I_\sigma(A, R) &= \sigma'_{A,R} \wedge range_{A,R} \wedge \bigwedge_{\overline{a} \in I \cap \overline{A}} \overline{a} \wedge \Big(\bigvee_{\overline{a} \in \overline{A} \setminus I} \overline{a} \Big) \\ \gamma^I_\sigma &= \bigvee_{\overline{a} \in \overline{A} \setminus I} \overline{a}. \end{aligned}$$

The formulae used in this algorithm differ somewhat from the previous ones for the semi-stable and stage semantics. Intuitively in Algorithm 5 we have to make sure to visit *every* semi-stable extension for the enumeration problem. Here we can restrict ourselves to decision

problems. Therefore we do not have to worry about finding e.g. semi-stable extensions with the same range as the ones previously computed, which was necessary in Algorithm 5. Therefore $\gamma^I_{\sigma'}$ is much simpler here as the formulae used in Line 10 of Algorithm 5. We exclude in future iterations any extension of the base semantics, which has a smaller *or equal* range to the currently found candidate. In contrast to Algorithm 5 where we exclude extensions of the base semantics which are strictly smaller w.r.t. the range or equal to previously found extensions, thereby allowing future iterations to include candidates with the same range as the current candidate. Here actually no excluded base set is a member from $\sigma(F)$. Thus the desired witness from $\sigma(F)$, if it exists, is never excluded and there exists a corresponding model of ϕ.

Shortcuts The shortcuts may either directly return the answer, by *accept*ing or *reject*ing the input, or return a formula, which can be conjoined with our base formula to prune the search space. That is the shortcuts are a sound but not a complete procedure. For preferred semantics one can include a very straightforward shortcut as shown in Algorithm 7. Here we simply check in line 1 whether there is a counter-example for skeptical acceptance of a under the chosen base-semantics witnessed by a set of the base semantics attacking a, and if not, learn in line 4 that this is the case. By learning we mean augmenting the main formulae.

Algorithm 7 Shortcuts$(F, a, \mathsf{co\text{-}Skept}, prf, \sigma')$

Require: AF $F = (A, R)$, argument $a \in A$ and base-semantics $\sigma' \in \{adm, com\}$
Ensure: *accept*s the input only if a is not skeptically accepted in F w.r.t. prf, returns a formula otherwise

1: **if** $\exists I, I \models \sigma'_{A,R} \wedge (\bigvee_{(b,a)\in R} b)$ **then**
2: *accept*
3: **else**
4: *return* $\bigwedge_{(b,a)\in R} \neg b$
5: **end if**

Regarding the Shortcuts function for semi-stable and stage semantics (Algorithm 8), we use a different technique. The basic idea follows the result of Theorem 3.2.2. The idea is to check if there is a set satisfying the base semantics and having a certain range. We start by checking if such a set containing a (for credulous reasoning) with a range of A exists, i.e. answering the credulous acceptance for stable semantics. If not, we ask whether there exists a set X of the base semantics with $a \in X$ s.t. $|X^+_R| = |A| - 1$. If not, then we can increase to $|A| - d$ for $d > 1$ until we arrive at $d = |A|$. A similar approach is also used when we come to the application of minimal correction sets in the next Section 3.4.

The loop in line 8 iterates over sets $S \subseteq A$. In line 9, MAXIMAL returns a maximal-cardinality set S from $\mathcal{S}$, (starting with $S = A$). Line 10 builds the formula f_S encoding that sets of interest have range S. Line 11 tests whether there is a set with range S satisfying q under the base semantics. If so, we have found an extension satisfying q and accept. Line 13 tests whether there is an extension under base semantics which has range S. If so, we learn that we are no longer interested in sets $S' \subseteq S$. Otherwise (line 17) we learn that we are not interested in extensions with range S. Finally, if all sets have been excluded from being the range of an

extension satisfying q, the procedures rejects. In case the bound d is exceeded before this, we return the formula ψ as learnt information. In our experiments in Chapter 5, $\mathcal{S}$ can be very large. Therefore we apply here the bound d for this set.

Algorithm 8 Shortcuts$(F, a, M, \sigma, \sigma')$

Require: AF $F = (A, R)$, argument $a \in A$,
base-semantics $\sigma' \in \{adm, com\}$ for $\sigma = sem$; $\sigma' = cf$ for $\sigma = stg$;
and reasoning mode $M = \{\mathsf{co\text{-}Skept}, \mathsf{Cred}\}$;

Ensure: *accept*s (*reject*s) the input only if $M = \mathsf{co\text{-}Skept}$ and a is not skeptically accepted (is skeptically accepted) or $M = \mathsf{Cred}$ and a is credulously accepted (rejected) in F w.r.t. σ, returns a formula otherwise

1: **if** $M = \mathsf{co\text{-}Skept}$ **then**
2: $q := \neg a$
3: **else if** $M = \mathsf{Cred}$ **then**
4: $q := a$
5: **end if**
6: $\psi := \top$
7: $\mathcal{S} := \{S \subseteq A\}$
8: **while** $(\exists S \in \mathcal{S} : |A \setminus S| \leq d)$ **do**
9: $S := \text{MAXIMAL}(\mathcal{S})$
10: $f_S := \bigwedge_{s \in S} \overline{s} \wedge \bigwedge_{s \in A \setminus S} \neg \overline{s}$
11: **if** $\exists I, I \models \sigma'_{A,R} \wedge q \wedge f_S(X)$ **then**
12: *accept*
13: **else if** $\exists I, I \models \sigma'_{A,R} \wedge f_S(X)$ **then**
14: $\psi := \psi \wedge \left(\bigvee_{s \in A \setminus S} \overline{s} \right)$
15: $\mathcal{S} := \mathcal{S} \setminus \{S' \mid S' \subset S\}$
16: **else**
17: $\psi := \psi \wedge \left(\bigvee_{s \in S} \neg \overline{s} \vee \bigvee_{s \in A \setminus S} \overline{s} \right)$
18: $\mathcal{S} := \mathcal{S} \setminus \{S\}$
19: **end if**
20: **end while**
21: **if** $(\mathcal{S} = \emptyset)$ **then**
22: *reject*
23: **else**
24: *return* ψ
25: **end if**

We proceed with the statement of correctness of this algorithm, which follows from previous descriptions and the proof of Theorem 3.3.4.

Proposition 3.3.13. *Let $F = (A, R)$ be an AF, $a \in A$, $\sigma \in \{prf, sem\}$ and $\sigma' \in \{adm, com\}$. Then*

- *Decide$(F, a, \mathsf{Cred}, \sigma, \sigma')$ accepts iff $\mathsf{Cred}_\sigma(a, F) = yes$;*

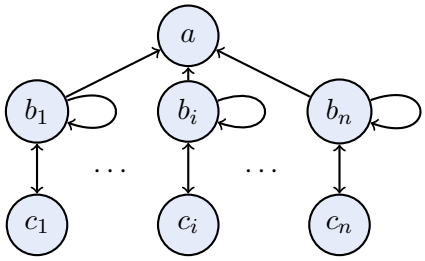

Figure 3.5: Example AF showing difference for running times for admissible and complete base semantics and Algorithm 6

- *Decide*$(F, a, \mathsf{co\text{-}Skept}, \sigma, \sigma')$ *accepts iff* $\mathsf{Skept}_\sigma(a, F) = no$*;*
- *Decide*$(F, a, \mathsf{Cred}, stg, cf)$ *accepts iff* $\mathsf{Cred}_{stg}(a, F) = yes$*; and*
- *Decide*$(F, a, \mathsf{co\text{-}Skept}, stg, cf)$ *accepts iff* $\mathsf{Skept}_{stg}(a, F) = no$*;*

Regarding the number of SAT calls necessary to decide the problem, we can investigate Decide$(F, a, M, \sigma, \sigma')$ without using shortcuts for an AF $F = (A, R)$, $a \in A$ and $\sigma \in \{prf, sem, stg\}$. It turns out that in case of admissible or conflict-free base semantics one can straightforwardly apply the proof idea of Corollary 3.3.5, i.e. the number of SAT calls required by Decide$(F, a, M, \sigma, \sigma')$ for $\sigma' \in \{adm, cf\}$ is $O(|A| \cdot k)$ if $F \in \mathsf{sol}^k_\sigma$.

Consider the skeptical reasoning mode and preferred semantics and that we have chosen admissible sets as the base semantics. (The idea for semi-stable and stage and credulous reasoning is similar.) The "inner" while loop in lines 8–10 follows the same idea as in Algorithm 1. The "outer" while loop is different here, though. Assume that we exclude a set $X \in adm(F)$ from future iterations in Line 14, then there clearly is a larger set $E \in prf(F)$ s.t. $X \subseteq E$ and a (the argument to test) is not in X, but in E. Now suppose there is an $X' \subseteq E$ with $a \notin X'$ and $X' \in adm(F)$. Then we can conclude that $X' \subseteq X$, since otherwise we could have extended X by X' to (at least) $X \cup X'$. This is due to the fact that $X \cup X' \subseteq E$ is conflict-free (since E is) and every argument from $X \cup X'$ is defended by this set (since every argument from X is defended by X and likewise for X') by assumption that these are admissible sets, i.e. $X, X' \in adm(F)$. Therefore if we exclude such a set X, we have excluded all $X' \subseteq E$ with $a \notin X'$ and $X' \in adm(F)$. This means that we will enter the outer while loop never again with an $X' \subseteq E$ and thus have to exclude at most one set X per preferred extension. This implies the number of SAT calls is bounded by $O(|A| \cdot |prf(F)|)$.

The same line of reasoning cannot be applied if we choose complete base semantics. There are AFs s.t. Decide for preferred semantics and complete base may require more than one execution of the outer while loop per preferred extension. Consider the following example.

Example 3.3.3. *Let $F = (A, R)$ be an AF and $n \geq 0$ an integer. Let $B = \{b_1, \ldots, b_n\}$, $C = \{c_1, \ldots, c_n\}$ and $A = B \cup C \cup \{a\}$ and $R = \{(c_i, b_i), (b_i, c_i), (b_i, b_i), (b_i, a) \mid 1 \leq i \leq n\}$. See Figure 3.5 for an illustration. Then all the sets $C' \subset C$ are complete in F. But C is is not*

complete in F, since a is defended by C in F. The set C is however admissible in F. The only preferred extension of F is $C \cup \{a\}$. If we now apply Algorithm 6 to this framework with the admissible base semantics and checking if a is not skeptically accepted, then it computes C in the inner while loop as the first admissible set to exclude from our search space. After that the algorithm immediately terminates. For the complete base semantics we have a different picture, since if the inner while loop terminates we exclude just one set $C \setminus \{c_j\}$ from the search space and may re-enter the loops with a fresh candidate containing c_j. Therefore for only one preferred extension, we enter the outer while loop n times in this example.

We point out that the complete base semantics is nevertheless useful as seen in the empirical evaluation in Section 5, where the complete base outperformed the admissible base on certain AFs.

3.4 Utilizing Minimal Correction Sets and Backbones

In this section we present algorithms to solve reasoning problems associated with several computationally complex semantics on AFs, namely the semi-stable, stage, eager, stage-ideal and ideal semantics using SAT-extensions. We apply sophisticated algorithms for minimal correction sets (MCSes) to solve reasoning tasks under the semi-stable and stage semantics, in particular $\mathsf{AllSkept}_{sem}$ and $\mathsf{AllSkept}_{stg}$, and procedures for computing backbones to solve $\mathsf{AllCred}_{adm}$. The systems we use for computing MCSes or backbones rely on iterative calls to a SAT-solver.

The eager semantics is based on the semi-stable semantics and, given the skeptically accepted arguments under the semi-stable semantics from the MCS algorithm, one can compute the unique eager extension in polynomial time by means of a post-processing step. Similarly one can apply this technique for the unique stage-ideal extension. The algorithm behind the ideal semantics is more complicated and is taken from [67]. The difficult part of this algorithm from a computational point of view is to compute $\mathsf{AllCred}_{adm}$; the remainder can be done by a similar post-processing technique as for eager.

In the following we show how this works in detail. We start with a formal introduction to MCSes and backbones in Section 3.4.1. In Section 3.4.2 we show how to use MCSes to compute the semi-stable and stage extensions, in Section 3.4.3 how to use this result to compute the eager and stage-ideal extension. In Section 3.4.4 we show how to utilize backbones to compute the ideal extensions of a given framework. In all cases we build on existing reductions to SAT for the admissible and stable semantics [21].

3.4.1 SAT Extensions

In some applications encountering an unsatisfiable formula is common. However one might still be interested in *partially* satisfying a formula then. For instance some of the constraints encoded as clauses might not be necessary for a solution. In particular one can think of the concept of hard and soft constraints (or clauses), which denote constraints which have to be satisfied or not respectively. This is captured e.g. in the following definition for MCSes. Recall that if a formula is in CNF, then we can view it as a set of clauses, each containing a set of literals.

Definition 3.4.1. *Given a formula ϕ in CNF and the hard constraint $\chi \subseteq \phi$, a* minimal correction set *is a minimal subset $\psi \subseteq \phi$ such that $\phi \setminus \psi$ is satisfiable and $\psi \cap \chi = \emptyset$. We call the clauses $\phi \setminus \chi$* soft *constraints (clauses).*

Example 3.4.1. *Consider the formula $\phi = (a) \wedge (\neg a \vee \neg c) \wedge (c \vee \neg a)$. This formula in CNF is unsatisfiable, i.e. $\models \neg\phi$. Each clause of this formula is a minimal correction set of ϕ e.g. $\psi = (a)$ is such a MCS. If we require that (a) is a hard clause in ϕ and all the other clauses are soft, then $\psi_2 = (\neg a \vee \neg c)$ and $\psi_3 = (c \vee \neg a)$ are minimal correction sets of ϕ.*

Numerous techniques to compute MCSes were developed (e.g., [93, 115, 132, 140]), and the field is still advancing, see e.g. [119, 121].

Our algorithms in Section 3.4.2 are based upon the concept of MCSes, but do not inherently depend on the implementation details of the MCS algorithm. Algorithm 9 shows a simplified version of the algorithm in [115] that underlies our implementation. Other MCS algorithms can also be used. The requirement we have for our algorithms is that we need to enumerate all models of formulae $\phi \setminus \psi$ given a formula ϕ and each of its minimal correction sets ψ.

Algorithm 9 MCS(ϕ) (simplified version of [115])

Require: $\phi = \bigcup_i \{C_i\}$ is unsatisfiable
Ensure: returns set $\mathcal{M}$ of all minimal correction sets for ϕ

```
1:  ψ := ⋃_i {(a_i ∨ C_i)}, with L := ⋃_i {a_i} a set of fresh atoms
2:  k := 1
3:  M := ∅
4:  while ψ is satisfiable do
5:      ψ_k := ψ ∧ AtMost(k, L)
6:      while ψ_k is satisfiable do
7:          I be such that I ⊨ ψ_k
8:          M := M ∪ {{C_i | a_i ∈ L ∧ I(a_i) := t}}
9:          let D be {¬a_i | a_i ∈ L ∧ I(a_i) := t}
10:         ψ_k := ψ_k ∧ D
11:         ψ := ψ ∧ D
12:     end while
13:     k := k + 1
14: end while
15: return M
```

In Algorithm 9, each (soft) clause C is augmented with a so-called *relaxation literal* a (sometimes also called activation literal), a fresh variable that does not occur anywhere else in ϕ (line 1). (A common optimization is to instrument only clauses contained in an unsatisfiable subset of ϕ.) The effect of dropping C can now be simulated by choosing an interpretation which maps a to $\mathbf{t}$. Given a set $L \subset \mathcal{P}$ of relaxation literals, a cardinality constraint $\mathit{AtMost}(k, L) := |\{a \in L \mid I(a) = \mathbf{t}\}| \leq k$ limits the number of clauses that can be dropped. This constraint can be encoded via a Boolean formula, see e.g. [4, 51]. Algorithm 9 derives *all* MCSes by *systematically* enumerating assignments of relaxation literals for increasingly larger

values of k (cf. the outer loop starting in line 4). The inner loop (line 6) enumerates all MCSes of size k by incrementally *blocking* MCSes represented by a conjunction of relaxation literals D.

The other SAT extension we use in this thesis is called a backbone of a Boolean formula ϕ, which is simply the set of literals the formula entails.

Definition 3.4.2. *Let ϕ be a satisfiable formula. The set $\{l \mid \phi \models l$ and l a literal$\}$ is the backbone of ϕ.*

The restriction to satisfiable formulae in the definition of backbones is due to the fact that for an unsatisfiable formula ϕ we always have $\phi \models l$ for any literal l.

Example 3.4.2. *Let $\phi = (a \vee b) \wedge (\neg a \vee b) \wedge (\neg b \vee \neg c)$ be a formula. Then the backbone of ϕ is $\{b, \neg c\}$.*

To compute the backbone of a formula ϕ (with $I \models \phi$), the currently most efficient algorithms (according to [122, 159]) iteratively "probe" each atom $a \in A$ by subsequently checking the satisfiability of $\phi \wedge \ell$ (with $\ell = \neg a$ if $I(a) = \mathbf{t}$, and $\ell = a$ otherwise). Algorithm 10 illustrates the basic structure of such an implementation. Practical implementations incorporate techniques such as excluding variables with opposing values in subsequent satisfying assignments (line 8 of Algorithm 10), clause reuse, and variable filtering [122, 159].

Algorithm 10 Probing(ϕ) (adapted from [122])

Require: ϕ is satisfiable
Ensure: returns $\{\ell \mid \ell \in \{a, \neg a \mid a \in \mathcal{P}\}$ and $\forall I \,.\, I \models \phi \wedge \ell$ or $I \models \neg\phi\}$

```
1:  S := ∅
2:  let I be s.t. I ⊨ φ
3:  for all ℓ ∈ {a, ¬a | a ∈ P} with I ⊨ ℓ do
4:      if ∄K, K ⊨ φ ∧ ¬ℓ then
5:          S := S ∪ {ℓ}; φ = φ ∪ {ℓ}
6:      else
7:          let K be such that K ⊨ φ ∧ ¬ℓ
8:          let I be a partial interpretation s.t. I = (I^t ∩ K^t, I^f ∩ K^f)
9:      end if
10: end for
11: return S
```

3.4.2 MCS Algorithm for Semi-stable and Stage Semantics

Computing semi-stable extensions inherently requires to compute admissible sets, which are subset-maximal w.r.t. the range. The MCS algorithm computes subset-minimal sets of clauses of a formula in CNF, which if removed result in a satisfiable formula. The idea to exploit the MCS algorithm for the semi-stable semantics is to encode the range as satisfied clauses of

a propositional formula for a given interpretation and additionally requiring that the result is admissible.

For this to work we slightly adapt the formulae from [21] for the stable semantics. Given an AF $F = (A, R)$ and $a \in A$ we define the following formulae.

$$in\_range_{a,R} = (a \vee \bigvee_{(b,a)\in R} b) \tag{3.5}$$

$$all\_in\_range_{A,R} = \bigcup_{a\in A} \{in\_range_{a,R}\} \tag{3.6}$$

The formula $in\_range_{a,R}$ indicates whether the argument a is in the range w.r.t. the atoms set to true in an interpretation. In other words, for an AF $F = (A, R)$ and $a \in A$ we have, $I \models in\_range_{a,R}$ iff $a \in S_R^+$ for $I^{\mathbf{t}} = S$. The formula $all\_in\_range_{A,R}$ is satisfied if all arguments are in the range. Note that this encoding of the range differs from previous formulae, where we encoded the range as variables set to true. We summarize this more formally in the following lemma, which follows directly from definition of $in\_range_{a,R}$.

Lemma 3.4.1. *Let $F = (A, R)$ be an AF, $S \subseteq A$, $a \in A$ and I an interpretation on A. Then $I \models in\_range_{a,R}$ iff $a \in S_R^+$ for $I^{\mathbf{t}} = S$.*

Conjoining the formulae $adm_{A,R}$ and $all\_in\_range_{A,R}$, which we denote by $stb_{A,R}$, results in a formula equivalent to the stable formula in [21].

$$stb_{A,R} = adm_{A,R} \cup all\_in\_range_{A,R} \tag{3.7}$$

An interpretation I which satisfies $adm_{A,R}$ for a given AF $F = (A, R)$ and a *subset-maximal* set of clauses of $all\_in\_range_{A,R}$ corresponds to a semi-stable extension of F. Consequently, we can derive semi-stable extensions from the correction sets computed with the MCS algorithm, as long as no clause from $adm_{A,R}$ is dropped. That is, we consider the clauses of the formula $adm_{A,R}$ as *hard* constraints and the clauses in $all\_in\_range_{A,R}$ as *soft* constraints. Note also, since any AF $F = (A, R)$ has at least one admissible set, we know that $adm_{A,R}$ is always satisfiable. If $stb_{A,R}$ is satisfiable, meaning that F has stable extensions, then immediately this computation yields the stable extensions, which are equal to the semi-stable extensions in this case.

The following proposition shows this result more formally. For a given propositional formula ϕ in CNF and an interpretation I, we define ϕ^I to be the set of clauses in ϕ, which are satisfied by I, i.e. $\phi^I := \{C \in \phi \mid I \models C\}$.

Proposition 3.4.2. *Let $F = (A, R)$ be an AF and $\mathcal{I}_{sem} = \{I \mid I \models adm_{A,R}$ and $\nexists I' : I' \models adm_{A,R}$ s.t. $all\_in\_range^I_{A,R} \subset all\_in\_range^{I'}_{A,R}\}$. Then $sem(F) = \mathcal{I}_{sem}$.*

Proof. Given an $a \in A$, by Lemma 3.4.1 we have that an interpretation I satisfies $in\_range_{a,R}$ iff $a \in S_R^+$ for $I^{\mathbf{t}} = S$. Therefore $all\_in\_range^I_{A,R}$ directly corresponds to S_R^+.

Let $F = (A, R)$ be an AF. Assume $E \in sem(F)$, then define the following interpretation I with $I(a) = \mathbf{t}$ iff $a \in E$. Then $I \models adm_{A,R}$, since E is admissible by definition and due

to [21] we know that I satisfies $adm_{A,R}$. Suppose now there exists an interpretation I' such that $I' \models adm_{A,R}$ and $all\_in\_range^{I}_{A,R} \subset all\_in\_range^{I'}_{A,R}$. But then E would not be maximal w.r.t. the range and hence no semi-stable extension of F.

Assume $E \in \mathcal{I}_{sem}$, which implies $E \in adm(F)$ and as above let I be an interpretation with $I(a) = \mathbf{t}$ iff $a \in E$. Suppose there exists a set $S \in adm(F)$ with $E^+_R \subset S^+_R$. Then $all\_in\_range^{I}_{A,R} \subset all\_in\_range^{I'}_{A,R}$ for an interpretation I' defined as $I'(a) = \mathbf{t}$ iff $a \in S$, which is a contradiction. □

The MCS algorithm can now be straightforwardly applied for the reasoning tasks for the semi-stable semantics, i.e. Cred_{sem}, Skept_{sem} and Enum_{sem}. In contrast to our search algorithms in Section 3.3.1 we do not consider the MCS algorithm purely as a black box, but slightly adapt it for our purpose. Since we need the set of skeptically accepted arguments for computation of the eager extension, we will present this variant in Algorithm 11.

Algorithm 11 MCS-$\mathsf{AllSkept}_{sem}(F)$

Require: AF $F := (A, R)$
Ensure: returns $\mathsf{AllSkept}_{sem}(F)$
1: $\phi := \{a_i \vee C_i \mid C_i \in all\_in\_range_{A,R}\}$ with $L := \bigcup_i \{a_i\}$ a set of fresh atoms
2: $\psi := adm_{A,R} \cup \phi$
3: $k := 0$
4: $X := A$
5: **while** ψ is satisfiable and $k \leq |A|$ **do**
6: $\quad \psi_k := \psi \cup \mathit{AtMost}(k, L)$
7: $\quad X := X \cap \text{Probing}(\psi_k)$
8: $\quad$ **while** ψ_k is satisfiable **do**
9: $\quad\quad$ **let** I be such that $I \models \psi_k$
10: $\quad\quad$ **let** D be $\{\neg a_i \mid a_i \in L \wedge I(a_i) = \mathbf{t}\}$
11: $\quad\quad \psi_k := \psi_k \wedge D$
12: $\quad\quad \psi := \psi \wedge D$
13: $\quad$ **end while**
14: $\quad k := k + 1$
15: **end while**
16: *return* X

Algorithm 11 (adapted from Algorithm 9) computes the set $\mathsf{AllSkept}_{sem}(F)$ for a given AF $F = (A, R)$. The formula ψ consists of the clauses for admissibility and the instrumented clauses of $all\_in\_range_{A,R}$, i.e. these clauses may be dropped during the running time. If $I \models \psi_k$, then $E = I \cap A$ is an admissible set in F and $|E^+_R| = |A| - k$, since we allow to drop k clauses of $all\_in\_range_{A,R}$ and block previously computed MCSes. This means that E is a semi-stable extension of F, since there is no assignment I' which satisfies $adm_{A,R}$ and a superset of $all\_in\_range^{I}_{A,R}$. We utilize the backbone algorithm in line 7 to compute in X the set of skeptically accepted arguments. Since all satisfying interpretations of ψ_k are semi-stable extensions we compute the set of atoms set to true in all such interpretations by applying Algorithm 10. If ψ_0 is satisfiable, then we set $D = \bot$ in Line 10.

Alternatives of Algorithm 9 can also be adapted for our problem as long as all satisfying assignments can be computed w.r.t. the formula reduced by each of its MCSes separately.

For the number of SAT calls Algorithm 11 requires in the worst case consider an AF F from $\mathsf{stablecons}^i_{sem}$. MCS-$\mathsf{AllSkept}_{sem}(F)$ implements the algorithm of the proof of Theorem 3.2.2, which we sketched in Section 3.2. It is easy to see that k in the Algorithm is at most i, since $|A| - i$ is the size of the smallest range of a semi-stable extension in F. Therefore we have at most i SAT calls via Line 5. The inner while loop in lines 8–13, blocks MCSes of size k. We only block clauses from the soft clauses in $all\_in\_range_{A,R}$. This subformula has $|A|$ clauses. As an upper bound for the number of SAT calls from this inner while loop consider that we block each subset of $all\_in\_range_{A,R}$ of size at most i. The number of such sets is $\Sigma_{j=0}^{i}\binom{|A|}{j} \leq (|A|+1)^i$ and thus we have polynomially many such sets w.r.t. $|A|$ and i. The "probing" for applying the backbone requires a number of calls polynomial w.r.t. the number of arguments in the AF. Thus overall we require a polynomial number of SAT calls if the AF in question is from $\mathsf{stablecons}^i_{sem}$ and we consider i to be fixed.

The variant for the stage semantics simply consists of choosing an encoding for conflict-free sets instead of admissible sets. The correctness can directly be deduced in the following proposition from Proposition 3.4.2.

Proposition 3.4.3. *Let $F = (A, R)$ be an AF and $\mathcal{I}_{stg} = \{I \mid I \models cf_{A,R}, \nexists I' : I' \models cf_{A,R}$ s.t. $all\_in\_range^I_{A,R} \subset all\_in\_range^{I'}_{A,R}\}$. Then $stg(F) = \mathcal{I}_{stg}$.*

We show the corresponding algorithm details in the Appendix in Algorithm 17.

3.4.3 MCS Algorithm for Eager and Stage-ideal Semantics

Using the Algorithm 11 for solving the $\mathsf{AllSkept}_{sem}$ problem, we can use its output to calculate the unique eager extension, since we just have to compute the subset-maximal admissible set within $\mathsf{AllSkept}_{sem}(F)$ for an AF F. For this we apply the restricted characteristic function a number of times bounded by the number of arguments in the framework, i.e. $\hat{\mathcal{F}}_F^{|A|}(\mathsf{AllSkept}_{sem}(F))$ results in the eager extension of F. We show this simple idea in Algorithm 12. Clearly the "postprocessing" of the MCS algorithm is polynomial w.r.t. the size of the AF.

Since the stage-ideal extension is defined as the subset maximal admissible set contained in the set of skeptically accepted arguments under stage semantics, one can easily adapt this algorithm for the stage-ideal semantics. We show this variant in the Appendix in Algorithm 18.

Algorithm 12 MCS-Eager(F)

Require: AF $F = (A, R)$
Ensure: returns $eager(F)$
1: $X :=$ MCS-$\mathsf{AllSkept}_{sem}(F)$
2: *return* $\hat{\mathcal{F}}_F^{|A|}(X)$

3.4.4 Backbone Algorithm for Ideal Semantics

For the ideal semantics we make use of a method proposed in [62] (see also [67]), which we recall in Algorithm 13. The important point for our instantiation of this algorithm is that we essentially need to compute $\mathsf{AllCred}_{adm}$ and afterwards, as before for the eager semantics, a post-processing with the function $\hat{\mathcal{F}}_F$. We define for an AF $F = (A, R)$ the auxiliary notion of adjacent arguments of an argument: $adj(a) = \{x \mid (x, a) \in R \text{ or } (a, x) \in R\}$. Additionally we define the restriction of an attack relation for a set S by $R|_S = \{(a, b) \in R \mid a \in S \text{ and } b \in S\}$.

Algorithm 13 Ideal(F) [62]

Require: AF $F = (A, R)$
Ensure: returns $ideal(F)$
1: $Cred := \mathsf{AllCred}_{\mathsf{adm}}(F)$
2: $Out := A \setminus Cred$
3: $X := \{x \in Cred \mid adj(x) \subseteq Out\}$
4: $F' := (X \cup Out, R|_{(X \cup Out)})$
5: *return* $\hat{\mathcal{F}}^{|A|}_{F'}(X)$

Briefly put, Algorithm 13 computes first the credulously accepted arguments w.r.t. admissible sets and then a set X, which consists of all of the credulously accepted arguments, except those, which have an adjacent argument also credulously accepted. This set acts as a kind of approximation of the skeptically accepted arguments w.r.t. the preferred semantics. Constructing then the new framework F' and computing the restricted characteristic function at most $|A|$ times in this new framework for X, suffices for computing the ideal extension.

Now it is straightforward to instantiate this with the help of a backbone algorithm. Given an AF $F = (A, R)$, we first simply compute the backbone of $adm_{A,R}$. Let S be the output of Algorithm 10 on this formula, then $O = \{a \mid \neg a \in S\}$ be the set of variables set to false in every satisfying interpretation of $adm_{A,R}$. Since we know that this formula is satisfiable, this means that $(A \setminus O) = \mathsf{AllCred}_{adm}(F)$. The rest of Algorithm 13 can be achieved with post-processing.

3.5 Summary

In this chapter we developed algorithms for computationally hard semantics on argumentation frameworks. The algorithms solve reasoning tasks of enumerating all extensions as well as credulous and skeptical acceptance w.r.t. preferred, semi-stable, stage, eager, stage-ideal and ideal semantics. The main idea for our algorithms is to delegate subtasks to modern SAT-solvers, which we call in an iterative manner.

The algorithms exploit complexity analytic results to relate the number of SAT-calls with inherent properties of an AF, in particular the number of extensions for a given framework and semantics, as well as a distance measure w.r.t. the range of extensions. We classified the algorithms into two groups, search algorithms, and algorithms based on one of two SAT extensions, namely MCSes and the backbone of a Boolean formula.

name	reference	semantics	reasoning modes	technique
Generic	Algorithm 1	σ	Enum, Cred, Skept	generic search
Preferred-SAT	Algorithm 3	prf	Enum, Cred, Skept	iterative SAT
Semi-stable-SAT	Algorithm 5	sem	Enum, Cred, Skept	iterative SAT
Stage-SAT	Algorithm 16	stg	Enum, Cred, Skept	iterative SAT
Decide	Algorithm 6	prf, sem, stg	Cred, Skept[2]	iterative SAT
MCS-AllSkept$_{sem}$	Algorithm 11	sem	AllSkept	MCS
MCS-AllSkept$_{stg}$	Algorithm 17	stg	AllSkept	MCS
MCS-Eager	Algorithm 12	$eager$	Enum	MCS
MCS-Stage-ideal	Algorithm 18	stg-idl	Enum	MCS
Ideal [67]	Algorithm 13	$ideal$	Enum	backbone

Table 3.2: Algorithms overview

We summarize the presented algorithms in Table 3.2. We see the names, supported semantics and reasoning modes as well as the applied technique in this table. Note that some of the algorithms can be straightforwardly applied to other reasoning modes as presented in this thesis. For instance MCS-AllSkept$_{sem}$ can be adapted to enumerate all semi-stable extensions as well. Further in the presented algorithms based on MCSes we also utilized backbones as an optimization step. The algorithm Ideal (Algorithm 13) is shown in the table for the sake of completeness. It was developed in [67]. We apply the backbone technique to a sub-procedure for this algorithm.

[2]The decision problem Cred$_{prf}(a, F)$ for an AF $F = (A, R)$ and $a \in A$ is in NP and is therefore omitted in the algorithm.

CHAPTER 4

Abstract Dialectical Frameworks: Novel Complexity Results and Algorithms

This chapter studies the computational properties of abstract dialectical frameworks (ADFs) and shows how to apply the core ideas of the search algorithms from Chapter 3 to ADFs. The main goal in this chapter is to adapt the generic search Algorithm 1 to problems for the preferred semantics on ADFs. To this end we require a thorough complexity analysis of the problems on ADFs. In particular the complexity results guide us in two ways. First they show us which solvers are suited for the subtasks we delegate to them in the search algorithm. Second, the hardness results indicate, under the standard assumptions of complexity theory, that the tasks we aim to solve are computationally involved problems.

We show that the computational complexity of the considered decision problems of ADFs is one level higher in the polynomial hierarchy compared to the corresponding problems on AFs. Thus, straightforward generalizations of our search algorithms to ADFs have to cope with problems which are intrinsically harder than it is the case for AFs. In particular this indicates that for preferred semantics one requires solvers capable of solving Σ_2^P problems. It turns out, however, that bipolar ADFs (BADFs) are not affected by this complexity jump; the decision problems on BADFs have the same computational complexity as their corresponding problems on AFs. Thus we can restrict ourselves to SAT-solvers for solving tasks on BADFs and the preferred semantics. As a side result of our complexity analysis we obtain a backbone algorithm for grounded semantics of general ADFs.

We thus present three major contributions in this chapter.

- We prove novel complexity results for decision problems on general ADFs in Section 4.1.1;
- we prove that for bipolar ADFs (BADFs) the complexity is the same as for AFs for the corresponding decision problems in Section 4.1.2; and

- in Section 4.2.1 we show how to generalize the generic search Algorithm 1 for the preferred semantics to ADFs using search engines capable of solving Σ_2^P-complete decision problems for general ADFs and using solvers for NP-complete problems for BADFs. In addition in Section 4.2.2 we show how to compute the grounded interpretation of an ADF via a backbone algorithm.

In particular the results for the class of BADFs is of interest, since they incorporate many relations between arguments, without increasing the corresponding complexity compared to AFs. Preliminary complexity results w.r.t. the grounded semantics, which we partially use, were derived in [34] before work on this thesis has started. Furthermore our results for BADFs can be seen as a generalization of [34, Proposition 15]. The main results in this chapter are published in [31, 147].[1] A longer version of [147] with detailed proofs is available as a technical report [146].

4.1 Complexity Analysis of ADFs

In this section we study the computational complexity of ADFs. We first consider the general case in Section 4.1.1 and afterwards BADFs in Section 4.1.2. For a quick reference of the results, the reader is referred to Table 4.5 on page 108.

4.1.1 Complexity Analysis of General ADFs

The complexity of deciding credulous or skeptical acceptance or verifying a solution on ADFs w.r.t. grounded, admissible and preferred semantics is "one level" higher in the polynomial hierarchy w.r.t. their counterparts on AFs. For grounded semantics the verification problem is $\mathrm{D^P}$-complete for ADFs. We apply proof ideas from [56] to prove hardness of the verification problem of preferred semantics and techniques from [63] for the skeptical reasoning problem of preferred semantics.

Computational Complexity of the Grounded Semantics

In this section we analyze the complexity of credulous and skeptical reasoning as well as the verification problem for grounded semantics. We begin with a rough upper bound and a straightforward computation of the grounded interpretation. This somewhat simplistic view is of use to gain an intuition of the grounded semantics. Afterwards we show tight upper and lower bounds for the decision problems. Here we first consider certain properties of three-valued interpretations. If a three-valued interpretation I satisfies these properties, then it "approximates" the grounded interpretation J of an ADF D by $J \leq_i I$, i.e. everything set to true in the grounded interpretation is also true in I and every argument set to false in the grounded interpretation is also false in I. Since the properties can essentially be checked by a simple "guess & check" approach, we can derive the corresponding complexity bounds for our decision problems. Subsequently

[1] We note that the results in [147] are derived using the general framework of approximation fixed point theory (AFT). However the results can be directly adapted for our setting.

we use these ideas to encode such an approximation into propositional logic. The benefit of the reduction to propositional logic is that we can use this reduction for implementations.

The grounded interpretation for an ADF $D = (S, L, C)$ can be computed by iterative applications of the characteristic function on the initial interpretation, which sets each argument in S to $\mathbf{u}$. For convenience we recall the corresponding Definition 2.3.30 from the background chapter.

Definition 2.3.30. *Let $D = (S, L, C)$ be an ADF and $i \geq 0$ an integer. Define the interpretation $grd^i(D)$ as follows. $grd^0(D) = I$ with $\forall s \in S : I(s) = \mathbf{u}$. For $i > 0$ we define $grd^i(D) = \Gamma_D(grd^{i-1}(D))$.*

Then by Proposition 2.3.8 it follows that $grd^{|S|}(D)$ is the grounded interpretation for an ADF $D = (S, L, C)$, since at each step we either arrive at a fixed point or change the status of at least one argument.

A useful way of looking at the accepted and rejected arguments w.r.t. the grounded interpretation of an ADF D is the following. If an argument s has a tautological acceptance condition, i.e. $\models \varphi_s$, then we have to accept it. Otherwise if φ_s is unsatisfiable, then we have to reject it, i.e. set it to false in the grounded interpretation. This means that any argument set to true in $I_1 = grd^1(D)$ has a tautological acceptance condition, while the arguments set to false in I_1 have an unsatisfiable acceptance condition. Now in the next iteration, we collect all arguments s in $I_2^{\mathbf{t}}$ for $I_2 = grd^2(D)$, for which the formula $\phi = \varphi_s[I_1^{\mathbf{t}}/\top][I_1^{\mathbf{f}}/\bot]$ is valid. If ϕ is unsatisfiable then $s \in I_2^{\mathbf{f}}$. This can be seen from the fact that in $[I_1]_2$ all two-valued interpretations set all arguments in $I_1^{\mathbf{t}}$ to true (which have a valid acceptance condition) and all arguments in $I_1^{\mathbf{f}}$ to false (which have an unsatisfiable acceptance condition). Therefore $\Gamma_D(I_1)(s) = \mathbf{t}$ and thus $s \in I_2$ iff ϕ is valid.

Next follows an easy observation, namely if an argument is undecided in the grounded interpretation, then there is a reason for it. This reason can be expressed by two interpretations. One for showing that we may not set the argument to true, and one for showing that we may not set it to false in the grounded interpretation. This is captured in the next lemma.

Lemma 4.1.1. *Let $D = (S, L, C)$ be an ADF, I its grounded interpretation and $u \in I^{\mathbf{u}}$. There exist two two-valued interpretations $I_u, J_u \in [I]_2$ such that $I_u \models \varphi_u$ and $J_u \not\models \varphi_u$.*

Proof. Assume $u \in I^{\mathbf{u}}$. Suppose now the contrary to the statement in the lemma, i.e. for all two-valued interpretations $J \in [I]_2$ we have either that $J(\varphi_u) = \mathbf{t}$ or $J(\varphi_u) = \mathbf{f}$. We consider the first case, since the second one is symmetric. This means that φ_u evaluates to true for all interpretations in $[I]_2$. Thus $\Gamma_D(I)(u) = \mathbf{t}$. This implies that $I \neq \Gamma_D(I)$, since $I(u) = \mathbf{u} \neq \mathbf{t} = \Gamma_D(I)(u)$. This is a contradiction to I being the grounded interpretation of D. □

We now proceed with a technical lemma, from which we infer the membership part of the computational complexity of the reasoning tasks for the grounded semantics.

Definition 4.1.1. *Let $D = (S, L, C)$ be an ADF and I a three-valued interpretation on S. We define the following three properties for I w.r.t. D.*

1. *for each* $a \in I^{\mathbf{t}}$ *there exists a* $J \in [I]_2$ *s.t.* $J \models \varphi_a$*;*

2. *for each* $r \in I^{\mathbf{f}}$ *there exists a* $J \in [I]_2$ *s.t.* $J \not\models \varphi_r$*;*

3. *for each* $u \in I^{\mathbf{u}}$ *there exist* $J, K \in [I]_2$ *s.t.* $J \models \varphi_u$ *and* $K \not\models \varphi_u$*.*

We denote the set of interpretations which satisfy all three properties w.r.t. D *with* $grd_{\leq_i}(D)$*.*

Now we show that the grounded interpretation satisfies all three properties for any ADF D. On the other hand, if we have a three-valued interpretation J which satisfies the three properties for an ADF D, then $grd(D) \leq_i J$.

Lemma 4.1.2. *Let* $D = (S, L, C)$ *be an ADF and* J *a three-valued interpretation on* S*. It holds that*

- $grd(D) \in grd_{\leq_i}(D)$*; and*
- *if* $J \in grd_{\leq_i}(D)$*, then* $grd(D) \leq_i J$*.*

Proof. Let I be the grounded interpretation of D, then it is straightforward to show that $I \in grd_{\leq_i}(D)$, i.e. it satisfies the three properties of Definition 4.1.1 w.r.t. D. By definition we have $I = \Gamma_D(I)$ (I is complete in D). Therefore if $a \in I^{\mathbf{t}}$, then $I(a) = \mathbf{t} = \Gamma_D(I)(a)$ and thus $K(\varphi_a) = \mathbf{t}$ for any $K \in [I]_2$. For an $r \in I^{\mathbf{f}}$ it is immediate that for any such K we have $K(\varphi_r) = \mathbf{f}$. Therefore the first two properties of Definition 4.1.1 are satisfied. Due to Lemma 4.1.1 also the third property is satisfied.

We now prove the second item. Assume $J \in grd_{\leq_i}(D)$. We show now by induction on $n \geq 1$ that $grd^n(D) \leq_i J$. For $n = 1$ and $grd^1(D) = K_1$ we have that if $s \in K_1^{\mathbf{t}}$ then φ_s is a tautology, implying that $s \in J^{\mathbf{t}}$. Suppose the contrary, i.e. $s \notin J^{\mathbf{t}}$. Because J satisfies the three properties, this implies that there exists a two-valued interpretation which does not satisfy φ_s. This is a contradiction to φ_s being a tautology. The case for $s \in K_1^{\mathbf{f}}$ is symmetric. Now assume the induction hypothesis $K_n = grd^n(D) \leq_i J$ and to show that $grd^{n+1}(D) \leq_i J$ holds consider $grd^{n+1}(D) = K_{n+1}$. By the hypothesis $K_n \leq_i J$ we get that $[J]_2 \subseteq [K_n]_2$. If $s \in (K_{n+1}^{\mathbf{t}} \setminus K_n^{\mathbf{t}})$, then φ_s evaluates to true under any interpretation in $[K_n]_2$. Clearly then there is no interpretation in $[J]_2$, which evaluates φ_s to false, therefore s is in $J^{\mathbf{t}}$, because otherwise one of the three properties would be violated. Similarly for the arguments set to false. This proves the lemma, since $grd^{|S|}(D) = I = grd(D)$. □

Note that checking the three properties of Definition 4.1.1 naturally corresponds to guessing two-valued interpretations and evaluating a formula. To decide whether a certain argument s is *not* true in the grounded interpretation of an ADF D, we can simply guess a three-valued interpretation I, s.t. $I(s) \neq \mathbf{t}$. We further guess two-valued interpretations from $[I]_2$ to witness that I satisfies the three properties. If such a guess succeeds, then due to the preceding lemma it follows that s is not true in the grounded interpretation of D. This line of thought underlies the following proposition. Recall that for any ADF $D = (S, L, C)$ and $s \in S$ we have $\mathsf{Cred}_{grd}(s, D) = \mathsf{Skept}_{grd}(s, D)$, since the grounded interpretation is unique.

Proposition 4.1.3. Cred_{grd} *is* coNP*-complete for ADFs.*

Proof. Let $D = (S, L, C)$ be an ADF and $s \in S$. For membership, consider the co-problem, i.e. deciding whether s is not true in the grounded interpretation. Now we guess a three-valued interpretation I and a set of two-valued interpretations $\mathcal{I} \subseteq [I]_2$ on S with $|\mathcal{I}| = 2 \cdot |S|$ and $s \notin I^{\mathbf{t}}$. Clearly we can construct this guess in polynomial time, since we have at most $|S| \cdot 2 + 1$ interpretations over the vocabulary S. Now we check if $I \in grd_{\leq_i}(D)$, i.e. whether it satisfies the three properties of Definition 4.1.1 w.r.t. D by considering $\mathcal{I}$ as witnesses for each property. If I satisfies the properties w.r.t. D, then $J \leq_i I$ with $J \in grd(D)$ by Lemma 4.1.2. Since $I(s) \neq \mathbf{t}$, we have that $J(s) \neq \mathbf{t}$, since otherwise we have $J(s) = \mathbf{t} \not\leq_i I(s)$ if $I(s) \in \{\mathbf{f}, \mathbf{u}\}$. This implies that s is not true in the grounded interpretation. Therefore $\mathsf{Cred}_{grd}(s, D) = \mathsf{Skept}_{grd}(s, D) = no$. Clearly if we cannot guess such an I and successfully check it, then $\mathsf{Cred}_{grd}(s, D) = yes$.

For hardness, we can adapt the proof of [34, Proposition 13]. We reduce the problem of **VALIDITY** to $\mathsf{Cred}_{grd}(f, D)$. Let ϕ be an arbitrary Boolean formula. Then construct $D = (atoms(\phi) \cup \{f\}, L, C)$ with $\varphi_x = x$ if $x \in atoms(\phi)$ and $\varphi_f = \phi$. The ADF D can be constructed in polynomial time w.r.t. the size of ϕ. Clearly if $I \in grd(D)$, then for any $x \in atoms(\phi)$ we have $I(x) = \mathbf{u}$. Now assume that $\mathsf{Cred}_{grd}(f, D) = yes$. Then $f \in I^{\mathbf{t}}$ and we have that for every X with $X \subseteq atoms(\phi)$ it holds that $X \cup \{f\} \in [I]_2$. Since I is admissible in D, this means that for all such X we have $X \models \varphi_f = \phi$. Thus ϕ is valid and a "yes" instance of **VALIDITY**. Now assume that ϕ is valid. Then clearly J with $J(x) = \mathbf{u}$ for $x \in atoms(\phi)$ and $J(f) = \mathbf{t}$ is the grounded interpretation of D, by the same reasoning as above. □

Using the proof of the preceding proposition, we show the complexity of the verification problem.

Theorem 4.1.4. Ver_{grd} *is* D^{P}*-complete for ADFs.*

Proof. Let $D = (S, L, C)$ be an ADF and I a three-valued interpretation over S. For membership consider the following. For each $x \in I^{\mathbf{t}}$ we can decide whether $\mathsf{Cred}_{grd}(x, D) = yes$ with a coNP check. For a $y \in I^{\mathbf{f}}$ we can adapt the proof of Proposition 4.1.3 to show that it is in coNP to decide whether y is false in the grounded interpretation of D. We guess a three-valued interpretation K with $K(y) \neq \mathbf{f}$. Further we guess a set of two-valued interpretations $\mathcal{K} \subseteq [K]_2$, which verify that $K \in grd_{\leq_i}(D)$. If this guess is successful, then y is not false in the grounded interpretation. Thus the problem of deciding whether y is false in the grounded interpretation is in coNP. Therefore we can verify that $I \leq_i J$ for J the grounded interpretation of D with $|I^{\mathbf{t}} \cup I^{\mathbf{f}}|$ independent problems in coNP, which can be combined into one problem in coNP.

For verifying that $J = I$, we also need to verify the undecided arguments in I. We similarly as before guess a set of two-valued interpretations $\mathcal{I} \subseteq [I]_2$ and check whether $I \in grd_{\leq_i}(D)$. If this check is successful, then by Lemma 4.1.2, we know that $J \leq_i I$. The check whether $J \leq_i I$ holds is in NP. By combining this with the coNP check we have $I = J$, which solves the verification problem. Note that the number and size of the guesses we need to construct for each argument is polynomial w.r.t. the size of the given ADF D.

For hardness consider the **SAT-VALIDITY** problem. Let ϕ and ψ be arbitrary Boolean formulae. W.l.o.g. we assume that $atoms(\phi) \cap atoms(\psi) = \emptyset$. We construct an ADF D as follows. Let $D = (atoms(\phi) \cup atoms(\psi) \cup \{d, s, v\}, L, C)$ with

- $\varphi_x = x$ if $x \in atoms(\phi) \cup atoms(\psi)$;
- $\varphi_d = d$;
- $\varphi_s = \phi \wedge d$; and
- $\varphi_v = \psi$.

The ADF D can be constructed in polynomial time w.r.t. the size of ϕ and ψ. Then we show that ϕ is satisfiable and ψ is valid iff $I = grd(D)$ with $I(x) = \mathbf{u}$ for $x \in atoms(\phi) \cup atoms(\psi) \cup \{d, s\}$ and $I(v) = \mathbf{t}$. We can straightforwardly adapt the hardness proof of Proposition 4.1.3 for seeing that if ψ is valid then $\mathsf{Cred}_{grd}(v, D) = yes$ and vice versa.

Now assume that ϕ is satisfiable and ψ is valid. We show that then I is the grounded interpretation. For any argument except s it is immediate to see that I assigns the same value as the grounded interpretation (if the acceptance condition of an argument a is the formula a, then it is always undecided in the grounded interpretation). What remains to be shown is that s is undecided in the grounded interpretation. Since d is undecided in the grounded interpretation we have two two-valued interpretations in $[grd(D)]_2$, s.t. one evaluates φ_s to true and another to false. Hence I is the grounded interpretation.

Assume otherwise that ϕ is unsatisfiable or ψ is not valid. In the latter case we have $\mathsf{Cred}_{grd}(v, D) = no$ and thus I is not the grounded interpretation of D. If ϕ is unsatisfiable then also φ_s is unsatisfiable and the grounded interpretation sets s to false, unlike the three-valued interpretation I. □

Having established the complexity bounds, we are now ready to define the reduction to propositional logic for computing the grounded interpretation of an ADF. We begin with the definition of the main part of the Boolean formula we construct.

Definition 4.1.2. *Let $D = (\{s_1, ..., s_n\}, L, C)$ be an ADF, where we order the arguments and $1 \leq i \leq n$. Then let*

$$grd\_form_{dec}(s_i) = ((s_i^1 \leftrightarrow \varphi_{s_i}^1) \wedge \cdots \wedge (s_i^{2 \cdot n} \leftrightarrow \varphi_{s_i}^{2 \cdot n}) \wedge \neg s_i^{\mathtt{u}}) \tag{4.1}$$

$$grd\_form_{undec}(s_i) = (\varphi_{s_i}^i \wedge \neg \varphi_{s_i}^{i+n} \wedge s_i^{\mathtt{u}}) \tag{4.2}$$

$$grd\_form(D) = \bigwedge_{1 \leq j \leq n} \big(grd\_form_{dec}(s_j) \vee grd\_form_{undec}(s_j)\big) \tag{4.3}$$

The propositional formula $grd\_form(D)$ is the central part of the reduction for the grounded semantics for ADFs to Boolean logic. Intuitively $grd\_form_{dec}(s_i)$ consists of $2 \cdot n$ copies of an equivalence between the argument s_i and its acceptance condition. Additionally we require that the auxiliary atom $s_i^{\mathtt{u}}$ is false. If this special atom is true in at least one model, then this is interpreted that s_i is undecided in the grounded interpretation. The formula $grd\_form_{undec}(s_i)$ is based on the observation from Lemma 4.1.1. In particular for each undecided argument u in the grounded interpretation of an ADF D we have that there is an $I \in [grd(D)]_2$ and an interpretation $J \in [grd(D)]_2$, s.t. $I(\varphi_u) = \mathbf{t}$ and $J(\varphi_u) = \mathbf{f}$. We now state some formal properties of $grd\_form(D)$. First we define a two-valued interpretation $I_{grd(D)}$.

Table 4.1: Values of $I_{grd(D)}$ for arguments, which are accepted or rejected in the grounded interpretation.

Variable x	s_i^1	$\cdots$	$s_i^{2\cdot n}$	$s_i^{\mathbf{u}}$
$I_{grd(D)}(x)$	$\mathbf{t}$	$\cdots$	$\mathbf{t}$	$\mathbf{f}$

Variable x	s_j^1	$\cdots$	$s_j^{2\cdot n}$	$s_j^{\mathbf{u}}$
$I_{grd(D)}(x)$	$\mathbf{f}$	$\cdots$	$\mathbf{f}$	$\mathbf{f}$

Table 4.2: Values of $I_{grd(D)}$ for arguments, which are undecided in the grounded interpretation.

Variable x	s_k^1	$\cdots$	s_k^k	$\cdots$	s_k^n	s_k^{n+1}	$\cdots$	s_k^{n+k}	$\cdots$	$s_k^{2\cdot n}$	$s_k^{\mathbf{u}}$
$I_{grd(D)}(x)$	$I_{s_1}(s_k)$	$\cdots$	$I_{s_k}(s_k)$	$\cdots$	$I_{s_n}(s_k)$	$J_{s_1}(s_k)$	$\cdots$	$J_{s_k}(s_k)$	$\cdots$	$J_{s_n}(s_k)$	$\mathbf{t}$

Definition 4.1.3. *Let $D = (S = \{s_1, ..., s_n\}, L, C)$ be an ADF with ordered arguments and I its grounded interpretation. Choose for each $s \in S$ two interpretations: $I_s, J_s \in [I]_2$. If $s \in I^{\mathbf{u}}$, then additionally the following two properties must hold:*

- $I_s \models \varphi_s$ *and*
- $J_s \not\models \varphi_s$.

Now we define the two-valued interpretation $I_{grd(D)}$ as follows for $x \in S$:

$$I_{grd(D)}(x) = \begin{cases} \mathbf{t} \text{ if } x = a^i, 1 \leq i \leq 2 \cdot n, a \in I^{\mathbf{t}} \\ \mathbf{f} \text{ if } x = r^i, 1 \leq i \leq 2 \cdot n, r \in I^{\mathbf{f}} \\ \mathbf{f} \text{ if } x = a^{\mathbf{u}}, a \in I^{\mathbf{t}} \text{ or } x = r^{\mathbf{u}}, r \in I^{\mathbf{f}} \\ \mathbf{t} \text{ if } x = u^{\mathbf{u}}, u \in I^{\mathbf{u}} \\ I_{s_i}(u) \text{ if } x = u^i, 1 \leq i \leq n, u \in I^{\mathbf{u}} \\ J_{s_{i-n}}(u) \text{ if } x = u^i, n+1 \leq i \leq 2 \cdot n, u \in I^{\mathbf{u}} \end{cases}$$

To illustrate Definition 4.1.3, consider an ADF $D = (S, L, C)$ with $S = \{s_1, \ldots, s_n\}$. Assume that I is the grounded interpretation of D and $I(s_i) = \mathbf{t}$, $I(s_j) = \mathbf{f}$ and $I(s_k) = \mathbf{u}$ be three arguments in D which are mapped to true, false and undecided respectively. Further assume that I_s and J_s are given for each $s \in S$ as in Definition 4.1.3. Then the values for $I_{grd(D)}$ are given in Table 4.1 for the accepted and rejected arguments and in Table 4.2 for the undecided arguments and their corresponding variables.

One can view the interpretation $I_{grd(D)}$ as "representing" the grounded interpretation of D. We now show that $I_{grd(D)}$ is a model of $grd\_form(D)$, i.e. we show that $I_{grd(D)} \models grd\_form(D)$.

Lemma 4.1.5. *Let $D = (S = \{s_1, \ldots, s_n\}, L, C)$ be an ADF. Then $I_{grd(D)} \models grd\_form(D)$.*

Proof. We have to show for all $s \in S$ that either $I_{grd(D)} \models grd\_form_{dec}(s)$ or $I_{grd(D)} \models grd\_form_{undec}(s)$. We will show that for $s \in I^{\mathbf{t}} \cup I^{\mathbf{f}}$ the former is always the case by induction on $grd^i(D)$ for an integer $i > 0$. For $s \in I^{\mathbf{u}}$ we will then show that the latter is always the case.

- Induction base ($i = 1$): Let $grd^1(D) = J$, then $I_{grd(D)} \models grd\_form_{dec}(s)$ for $s \in J^{\mathbf{t}} \cup J^{\mathbf{f}}$. First consider the case that $s \in J^{\mathbf{t}}$, then φ_s is tautological and $I_{grd(D)}(s^j) = \mathbf{t}$ for $1 \leq j \leq 2 \cdot n$. This means that all equivalences of $grd\_form_{dec}(s)$ are satisfied, and since $I_{grd(D)}(s^{\mathbf{u}}) = \mathbf{f}$ we know that this subformula is satisfied by $I_{grd(D)}$. Similarly for $s \in J^{\mathbf{f}}$.

- Induction hypothesis: For an integer i, let $grd^i(D) = J$, then $I_{grd(D)} \models grd\_form_{dec}(s)$ for $s \in J^{\mathbf{t}} \cup J^{\mathbf{f}}$.

- Induction step: Assume that the induction hypothesis is true and $grd^i(D) = J$. Let $grd^{i+1}(D) = K$. For every $s \in K^{\mathbf{t}}$, the formula φ_s is tautological if the variables from $J^{\mathbf{t}}$ are set to true and $J^{\mathbf{f}}$ are set to false in φ_s, i.e. $\models \varphi_s[J^{\mathbf{t}}/\top][J^{\mathbf{f}}/\bot]$. If this would not be the case, then $s \notin K^{\mathbf{t}}$. Hence all equivalences of $grd\_form_{dec}(s)$ are satisfied, and again since $I_{grd(D)}(s^{\mathbf{u}}) = \mathbf{f}$ we know that $I_{grd(D)} \models grd\_form_{dec}(s)$. Similarly for $s \in K^{\mathbf{f}}$.

For $s_i \in I^{\mathbf{u}}$ we show that $I_{grd(D)} \models grd\_form_{undec}(s_i)$. By construction we know that $I_{grd(D)} \models \varphi^i_{s_i}$ and $I_{grd(D)} \not\models \varphi^{n+i}_{s_i}$. Further we have that $I_{grd(D)}(s_i^{\mathbf{u}}) = \mathbf{t}$, hence $I_{grd(D)} \models grd\_form_{undec}(s_i)$ for all $s_i \in I^{\mathbf{u}}$. □

Next we show that $grd\_form(D)$ entails certain atoms, corresponding to accepted and rejected arguments in the ADF w.r.t. grounded semantics.

Lemma 4.1.6. *Let $D = (S, L, C)$ be an ADF, I its grounded interpretation and $|S| = n$. Then $grd\_form(D) \models \bigwedge_{a \in I^{\mathbf{t}}}(a^1 \wedge \cdots \wedge a^{2n} \wedge \neg a^{\mathbf{u}}) \wedge \bigwedge_{r \in I^{\mathbf{f}}}(\neg r^1 \wedge \cdots \wedge \neg r^{2n} \wedge \neg r^{\mathbf{u}})$.*

Proof. Let $X \subseteq S$ be a set of arguments, then we define the following helper functions $A(X) = \bigwedge_{x \in X}(x^1 \wedge \cdots \wedge x^{2 \cdot n} \wedge \neg x^{\mathbf{u}})$ and $R(X) = \bigwedge_{x \in X}(\neg x^1 \wedge \cdots \wedge \neg x^{2 \cdot n} \wedge \neg x^{\mathbf{u}})$. We now prove by induction on $grd^i(D)$ the lemma. Note that $grd\_form(D)$ is satisfiable by Lemma 4.1.5.

- Induction base ($i = 1$): Let $grd^1(D) = J$, then the entailment $grd\_form(D) \models A(J^{\mathbf{t}}) \wedge R(J^{\mathbf{f}})$ follows, since $\varphi_a \equiv \top$ and $\varphi_r \equiv \bot$ for $a \in J^{\mathbf{t}}$ and $r \in J^{\mathbf{f}}$. Consider the case that $s \in J^{\mathbf{t}}$, then $\neg\varphi_s$ is unsatisfiable, hence $grd\_form_{undec}(s)$ is unsatisfiable. Therefore, since at least one model of grd_form exists, we know that in all models of it we have to satisfy $A(J^{\mathbf{t}})$. The proof for $R(J^{\mathbf{f}})$ is analogous.

- Induction hypothesis: For an integer i, let $grd^i(D) = J$, then $grd\_form(D) \models A(J^{\mathbf{t}}) \wedge R(J^{\mathbf{f}})$.

- Induction step: Assume that the induction hypothesis is true. Let $grd^i(D) = J$ and $grd^{i+1}(D) = K$. If $grd\_form(D) \models A(J^{\mathbf{t}}) \wedge R(J^{\mathbf{f}})$, then in all models of $grd\_form(D)$ (which exist, due to Lemma 4.1.5) we have that the renamed atoms of $a \in J^{\mathbf{t}}$ are set to true and $r \in J^{\mathbf{f}}$ are set to false, i.e. $J(a^j) = \mathbf{t}$ and $J(r^j) = \mathbf{f}$ for $1 \leq j \leq 2 \cdot n$. Hence $\varphi_b[J^{\mathbf{t}}/\top][J^{\mathbf{f}}/\bot]$ is tautological for $b \in K^{\mathbf{t}}$, i.e. cannot be falsified anymore and $\varphi_c[J^{\mathbf{t}}/\top][J^{\mathbf{f}}/\bot]$ for $c \in K^{\mathbf{f}}$ cannot evaluate to true. This means that both subformulae $grd\_form_{undec}(b)$ and $grd\_form_{undec}(c)$ are unsatisfiable. In all models satisfying $grd\_form_{dec}(b)$ we have that $b^1 \wedge \cdots \wedge b^{2 \cdot n} \wedge \neg b^{\mathbf{u}}$ is entailed. Similarly for c.

This proves the claim. □

The next result follows from Lemma 4.1.5.

Lemma 4.1.7. *Let $D = (S, L, C)$ be an ADF and I its grounded interpretation, then the formula $grd\_form(D) \wedge \bigwedge_{u \in I^{\mathbf{u}}} u^{\mathbf{u}}$ is satisfiable.*

The previous lemmata show that the grounded interpretation can be computed by considering the backbone of $grd\_form(D)$ (see Section 4.2.2). Here we show that the encoding matches the complexity of the verification problem of the grounded semantics of ADFs.

Proposition 4.1.8. *Let $D = (S, L, C)$ be an ADF and I a three-valued interpretation on S. Further let*

$$\phi = grd\_form(D) \wedge \bigwedge_{u \in I^{\mathbf{u}}} u^{\mathbf{u}}$$

$$\psi = grd\_form(D) \rightarrow \Big(\bigwedge_{a \in I^{\mathbf{t}}} (a^1 \wedge \cdots \wedge a^{2n} \wedge \neg a^{\mathbf{u}}) \wedge \bigwedge_{r \in I^{\mathbf{f}}} (\neg r^1 \wedge \cdots \wedge \neg r^{2n} \wedge \neg r^{\mathbf{u}}) \Big)$$

Then I is the grounded interpretation of D iff ϕ is satisfiable and ψ is valid.

Proof. We now show that the formula ϕ is satisfiable and the formula ψ is valid iff I is the grounded interpretation of D. Assume I is the grounded interpretation of D, then ϕ is satisfiable due to Lemma 4.1.7 and ψ is valid due to Lemma 4.1.6. Assume now that I is not the grounded interpretation of D, but the three-valued interpretation J is, with $J \neq I$. Then there exists an $s \in S$, such that $I(s) \neq J(s)$. The following cases may occur:

- $I(s) = v \neq \mathbf{u} = J(s)$ for $v \in \{\mathbf{t}, \mathbf{f}\}$: Due to Lemma 4.1.7 we have that $\neg s^{\mathbf{u}}$ cannot be entailed from $grd\_form(D)$, hence ψ is not valid.
- $I(s) = \mathbf{u} \neq v = J(s)$ for $v \in \{\mathbf{t}, \mathbf{f}\}$: Due to Lemma 4.1.6 we know that $grd\_form(D) \wedge s^{\mathbf{u}}$ is not satisfiable.
- $I(s) = v \neq v' = J(s)$ for $v, v' \in \{\mathbf{t}, \mathbf{f}\}$: Then ψ is not valid, due to Lemma 4.1.6.

This proves the proposition. □

A natural question regarding the encoding is why we have many duplicates in this reduction to propositional logic (although still only polynomially many w.r.t. the size of the ADF). Assume that we have determined which arguments are true and which are false in the grounded interpretation I of an ADF. For showing that a certain argument x is undecided in I, one needs two witnesses from $[I]_2$, one interpretation evaluating φ_x to false and another evaluating φ_x to true. Consider an ADF $D = (\{a, b, c\}, L, \{\varphi_a = a, \varphi_b = b, \varphi_c = (\neg(a \leftrightarrow b))\})$. The grounded interpretation of D sets all three arguments to undecided. For witnesses we thus require two two-valued interpretations for each argument. The example shows that in general one cannot use just two such interpretations for showing the undecidedness for *all* undecided arguments. In Table 4.3 we see why this is the case. The table shows four interpretations from $[I]_2$ projected on

Table 4.3: Example for two two-valued interpretations, which are not sufficient to show undecidedness w.r.t. the grounded semantics. For showing that a and b are undecided, we can use I_1 with I_4 or I_2 with I_3, but with both choices, c evaluates to the same truth value under both interpretations.

I	a	b	$\varphi_a = a$	$\varphi_b = b$	$\varphi_c = \neg(a \leftrightarrow b)$
I_1	$\mathbf{f}$	$\mathbf{f}$	$\mathbf{f}$	$\mathbf{f}$	$\mathbf{t}$
I_2	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{f}$
I_3	$\mathbf{t}$	$\mathbf{f}$	$\mathbf{t}$	$\mathbf{f}$	$\mathbf{f}$
I_4	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{t}$

the two arguments a and b (the value for c is not relevant) and the evaluation on the acceptance conditions. We need more than two interpretations from $\{I_1, I_2, I_3, I_4\}$ to witness that all three arguments are undecided in D w.r.t. the grounded interpretation. Therefore we need in general more than two copies of each argument in the encoding, since we may need for each argument two interpretations as witnesses. For clarity we have chosen to include $2 \cdot |S|$ many copies of each argument in the encoding.

Computational Complexity of the Admissible Semantics

The computational complexity of the admissible semantics of ADFs is crucial for an understanding of the preferred semantics and other admissibility-based semantics. We begin with the most basic decision problem, the verification problem.

Proposition 4.1.9. Ver_{adm} *is* coNP*-complete for ADFs.*

Proof. Let $D = (S, L, C)$ be an ADF and I a three-valued interpretation on S. For membership consider the co-problem, i.e. verify that I is not admissible in D. Guess an argument $s \in S$ such that $I(s) \in \{\mathbf{t}, \mathbf{f}\}$ and a two-valued interpretation I' that extends I, i.e. $I' \in [I]_2$. Check that I is not admissible, i.e. $I'(\varphi_s) \neq I(s)$.

For hardness we provide a reduction from the problem if a given propositional formula ϕ over vocabulary P is valid. Construct an ADF D with arguments $P \cup \{a\}$, where $a \notin P$ and $\varphi_s = s$ if $s \in P$ and $\varphi_a = \phi$. Further construct a three-valued interpretation I with $I(s) = \mathbf{u}$ for $s \in P$ and $I(a) = \mathbf{t}$. The ADF D and I can be constructed in polynomial time w.r.t. the size of ϕ. We show that I is admissible iff ϕ is valid. The set of two-valued extensions $I' \in [I]_2$ of I, if restricted to P, equals the set of possible two-valued interpretations of ϕ. Hence if ϕ is valid, I will be admissible, since then all two-valued interpretations are models of ϕ and likewise for all extensions I' of I we have $I'(\varphi_a) = \mathbf{t}$. Similarly if ϕ is not valid then $\Gamma_D(I)(a) \neq \mathbf{t}$, since there is an interpretation that falsifies ϕ and hence an extension $I' \in [I]_2$ with $I'(\varphi_a) = \mathbf{f}$. Thus I is not admissible. □

Next follows a rather basic observation, which proves useful, namely that self-attacking arguments are always undecided in any admissible interpretation for any ADF. This is not sur-

prising, since the same occurs with labelings and AFs. By a self-attack in ADFs we mean an acceptance condition like $\varphi_s = \neg s$ for an argument s.

Observation 4.1.10. *Let $D = (S, L, C)$ be an ADF and $s \in S$ with $\varphi_s = \neg s$ and $I \in adm(D)$. Then $I(s) = \mathbf{u}$.*

Proof. Suppose there exists a $I \in adm(D)$ with $I(s) \neq \mathbf{u}$. If $I(s) = \mathbf{t}$, then $\Gamma_D(I) = J$ with $J(s) = \mathbf{f}$, since $I(\varphi_s) = \mathbf{f}$. The case with $I(s) = \mathbf{f}$ is handled similarly. □

We now come to the main result in this section, the complexity of credulous reasoning under admissible semantics. Recall that the interpretation setting all arguments to $\mathbf{u}$ is always admissible and thus no argument is skeptically accepted w.r.t. admissibility. We now use a reduction from QBFs. There is a useful reading of QBFs we apply in our proofs. Consider a QBF $\psi \in \mathcal{QBF}_{\forall,3}$ with $\psi = \forall X \exists Y \forall Z \phi$. Then ψ is valid iff we have for every $X' \subseteq X$ that there is a $Y' \subseteq Y$ s.t. for any $Z' \subseteq Z$ we have that $X' \cup Y' \cup Z' \models \phi$. This can be inferred by looking at Definition 2.2.14 for semantics of QBFs. For the next proof we use QBFs in $\mathcal{QBF}_{\exists,2}$. One can straightforwardly adapt this line of reasoning of QBFs accordingly for this and other classes.

Proposition 4.1.11. Cred_{adm} *is Σ_2^P-complete for ADFs.*

Proof. Let $D = (S, L, C)$ be an ADF and $s \in S$. Σ_2^P membership can be proven by guessing a three-valued interpretation I, for which it holds that $I(s) = \mathbf{t}$. Then we check if $I \in adm(D)$, which itself is a coNP-complete problem, see Proposition 4.1.9.

For hardness we provide a reduction from the problem $\mathcal{QBF}_{\exists,2}$-**VALIDITY**. Let $\phi \in \mathcal{QBF}_{\exists,2}$ be a closed QBF of the form $\phi = \exists X \forall Y \psi$. Construct an ADF $D = (S, L, C)$ as follows. $S = X \cup Y \cup \{f\}$. $L = \{(z, z), (z, f) \mid z \in X \cup Y\}$. Let the acceptance conditions be $\varphi_x = x$ if $x \in X$, $\varphi_y = \neg y$ for $y \in Y$ and finally $\varphi_f = \psi$. See Figure 4.1 for an illustration of the constructed ADF. The ADF D can be constructed in polynomial time w.r.t. the size of ϕ. Due to Observation 4.1.10 we know that in any admissible model I of D the value of an argument $y \in Y$ is set to $\mathbf{u}$, i.e. $I(y) = \mathbf{u}$. We now prove that $\mathsf{Cred}_{adm}(f, D) = yes$ iff ϕ is valid.

Assume now that $\mathsf{Cred}_{adm}(f, D) = yes$. Then there exists a three-valued interpretation $I \in adm(D)$, s.t. $I(f) = \mathbf{t}$. We show that in this case ϕ is valid. By definition we know that $I \leq_i \Gamma_D(I)$. Since $I|_Y$ sets all arguments to undecided, we have that for $X' = I^{\mathbf{t}} \cap X$ and any $Y' \subseteq Y$ that $X' \cup Y' \cup \{f\} \in [I]_2$. Since $I(f) = \mathbf{t} = \Gamma_D(I)(f) = \mathbf{t}$ we have that $X' \cup Y' \cup \{f\} \models \varphi_f = \psi$. Thus $X' \cup Y' \models \psi$. Since this holds for any $Y' \subseteq Y$ this implies that ϕ is valid.

Now assume that ϕ is valid. This means that there exists a set $X' \subseteq X$, s.t. for any $Y' \subseteq Y$ we have $X' \cup Y' \models \psi$. We now show that $I = (X' \cup \{f\}, X \setminus X', Y)$ is admissible in the constructed ADF D. We directly have $I(x) \leq_i \Gamma_D(I)(x)$ for $x \in X$. Further $I(y) = \mathbf{u}$ for $y \in Y$, thus $I(y) \leq_i \Gamma_D(I)(y)$. What remains to be shown is that $I(f) = \mathbf{t} = \Gamma_D(I)(f)$. Consider a $J \in [I]_2$. Clearly $J \cap X = X'$ by construction. Therefore for any such J it follows that $J \models \varphi_f = \psi$, since ψ is satisfied by an interpretation K with $K \cap X = X'$ and any $Y'' \subseteq Y$ with $K \cap Y = Y''$. Therefore I is admissible in D. □

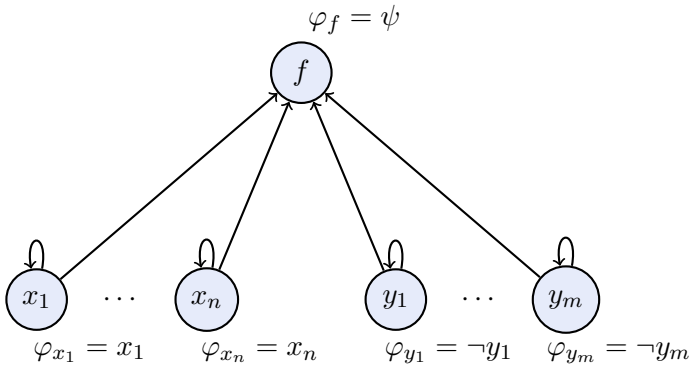

Figure 4.1: Constructed ADF for hardness proof of Proposition 4.1.11.

Computational Complexity of the Preferred Semantics

Regarding the preferred semantics and its computational complexity, we first prove an auxiliary result.

Lemma 4.1.12. *Let $D = (S, L, C)$ be an ADF, $s \in S$ and ϕ a propositional formula. If $\varphi_s = \neg s \vee \phi$ and $I \in adm(D)$. Then $I(s) \neq \mathbf{f}$.*

Proof. Suppose $I(s) = \mathbf{f}$. Then immediately $I(\varphi_s) = \mathbf{t}$. But then $\Gamma_D(I)(s) = \mathbf{t} \neq \mathbf{f}$, which implies that I is not admissible in D. □

Now we consider the existence of a non-trivial admissible interpretation for showing the complexity of the verification problem of preferred interpretations. This proof idea is adapted from [56], where this technique is used to show the corresponding complexity for AFs. Recall that an interpretation is non-trivial if there exists an argument *not* set to undecided.

Lemma 4.1.13. *Deciding whether for a given ADF there exists a* non-trivial *admissible interpretation of this ADF is Σ_2^P hard.*

Proof. We show hardness by a reduction from a closed QBF $\phi = \exists X \forall Y \psi \in \mathcal{QBF}_{\exists,2}$. Construct $D = (S, L, C)$ as follows. $S = X \cup \overline{X} \cup Y \cup \{f\}$. The acceptance conditions are defined by $\varphi_x = f \wedge \neg\overline{x}$ for $x \in X$, $\varphi_{\overline{x}} = f \wedge \neg x$ for $\overline{x} \in \overline{X}$, $\varphi_y = \neg y$ for $y \in Y$ and $\varphi_f = \neg f \vee \psi$. For an illustration of the constructed ADF see Figure 4.2. The ADF D can be constructed in polynomial time w.r.t. the size of ϕ. Now we show that there exists an $I \in adm(D)$ with I non-trivial iff ϕ is valid. By Observation 4.1.10 we know that each $y \in Y$ is never assigned a value $\mathbf{t}$ or $\mathbf{f}$ in any admissible interpretation. By Lemma 4.1.12 we know that f is never assigned $\mathbf{f}$. Now suppose $I(f) = \mathbf{u}$ and there exists a $x \in X$ or $\overline{x} \in \overline{X}$ such that $I(x) \neq \mathbf{u}$ or $I(\overline{x}) \neq \mathbf{u}$. Then $I(x)$ cannot be $\mathbf{t}$, since for this to hold we require $I(f) = \mathbf{t}$. Similarly for $\overline{x}$. The last case is to consider $I(x) = \mathbf{f}$. This can only be if $I(\overline{x}) = \mathbf{t}$, which is a contradiction to the assumption that $I(f) = \mathbf{u}$. Hence in any $I \in adm(D)$ if an argument is assigned a value different from $\mathbf{u}$, then it also holds that $I(f) = \mathbf{t}$.

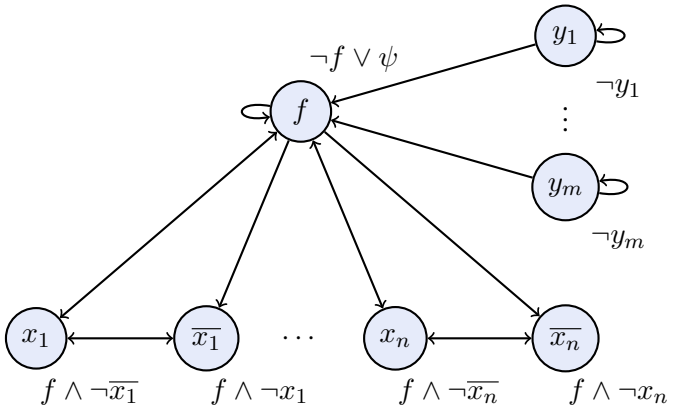

Figure 4.2: Constructed ADF for hardness proof of Lemma 4.1.13.

Assume that $I \in adm(D)$ and I is non-trivial. Then $I(f) = \mathbf{t}$. This means $\Gamma_D(I)(f) = \mathbf{t}$ and for all $J \in [I]_2$ it holds that $J(\varphi_f) = \mathbf{t} = J(\psi)$. Clearly for $I^{\mathbf{t}} \cap X = X'$ and any $Y' \subseteq Y$ we have $X' \cup Y' \cup \{f\} \in [I]_2$. Thus $X' \cup Y' \models \psi$. This implies that ϕ is valid.

Assume now that ϕ is valid. Then there exists an $X' \subseteq X$, s.t. for any $Y' \subseteq Y$ we have that $X' \cup Y \models \psi$. Then we show that $I = (X' \cup \{f\} \cup \overline{X \setminus X'}, X \setminus X' \cup \overline{X'}, Y)$ is admissible in D. Clearly for any $s \in S \setminus \{f\}$ we have $I(s) \leq_i \Gamma_D(I)(s)$. It remains to be shown for f. Since $\{X' \cup Y'' \cup \{f\} \mid Y'' \subseteq Y\} = [I]_2$ it follows that any interpretation in $[I]_2$ evaluates φ_f to true. Thus I is admissible in D. □

Now we can prove that Ver_{prf} is Π_2^P-complete.

Proposition 4.1.14. Ver_{prf} *is* Π_2^P*-complete for ADFs.*

Proof. Let $D = (S, L, C)$ be an ADF and I a three valued interpretation on S. To show membership, consider the co-problem, i.e. is I *not* preferred in D, or equivalently, does there exist an $I' \in adm(D)$ with $I <_i I'$. Membership can be proven by guessing a three-valued interpretation I' such that $I <_i I'$ and checking if I' is admissible in D. Checking if I' is admissible in D is a coNP-complete problem, see Proposition 4.1.9.

To show hardness, consider the special case for verifying that the interpretation I' is preferred with $I'(s) = \mathbf{u}, \forall s \in S$. The interpretation I' is not preferred iff there exists a non-trivial admissible interpretation in D, i.e. an admissible interpretation setting at least one argument to either $\mathbf{t}$ or $\mathbf{f}$. Due to to Lemma 4.1.13 this problem is Σ_2^P hard. □

It is easy to see that Cred_{adm} and Cred_{prf} coincide for ADFs. If a three-valued interpretation I is admissible in an ADF D, then there exists an $I' \in prf(D)$ with $I \leq_i I'$, since if I is not preferred in D, then there must exist a greater three-valued interpretation w.r.t. $\leq_i$. Finally we turn our attention to the skeptical acceptance of an argument under preferred semantics. Recall that this is one of the most challenging problems for AFs (it is Π_2^P-complete for AFs). We apply proof techniques from [63] to prove our results. We begin by defining our reduction.

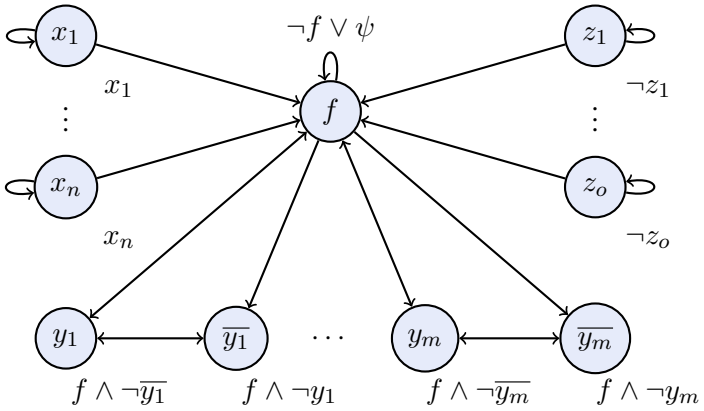

Figure 4.3: Constructed ADF for hardness proof of Theorem 4.1.17.

Definition 4.1.4. *Let $\phi \in \mathcal{QBF}_{\forall,3}$ be a closed QBF, with the form $\phi = \forall X \exists Y \forall Z \psi$. We define the ADF $D_{prf(\phi)} = (S, L, C)$ with $S = X \cup Y \cup \overline{Y} \cup Z \cup \{f\}$ and the acceptance conditions as follows.*

- $\varphi_x = x$ *if* $x \in X$,
- $\varphi_y = \neg\overline{y} \wedge f$ *if* $y \in Y$,
- $\varphi_{\overline{y}} = \neg y \wedge f$ *if* $\overline{y} \in \overline{Y}$,
- $\varphi_z = \neg z$ *if* $z \in Z$,
- $\varphi_f = \neg f \vee \psi$.

As usual the links L are defined implicitly via C.

See Figure 4.3 for an illustration of the constructed ADF from Definition 4.1.4. The ADF $D_{prf(\phi)}$ can be constructed in polynomial time w.r.t. the size of ϕ. We now prove some properties of this ADF. First we show that for any two-valued truth assignment on the arguments in X there is at least one preferred interpretation setting the arguments in X to these values.

Lemma 4.1.15. *Let $\phi = \forall X \exists Y \forall Z \psi$ be a closed QBF, $D_{prf(\phi)} = (S, L, C)$, the set of all two-valued interpretations on X be $\mathcal{J} = \{J \mid x \in X, J(x) = \mathbf{t}$ or $J(x) = \mathbf{f}\}$ and the set of preferred models of $D_{prf(\phi)}$ restricted to X be $\mathcal{I} = \{I|_X \mid I \in prf(D_{prf(\phi)})\}$. It then holds that $\mathcal{J} = \mathcal{I}$.*

Proof. Assume $J \in \mathcal{J}$, we need to show that then there exists an $I \in prf(D_{prf(\phi)})$ with $J = I|_X$. Consider J' with $J'|_X = J$ and $J'(s) = \mathbf{u}$ for $s \in S \setminus X$. Clearly J' is admissible in $D_{prf(\phi)}$. Now we have that there exists an $I \in prf(D_{prf(\phi)})$ s.t. $J' \leq_i I$.

Now assume that $I \in prf(D_{prf(\phi)})$. We need to show that then there exists a $J \in \mathcal{J}$ such that $I|_X = J$. Suppose there exists an $x \in X$ such that $I(x) = \mathbf{u}$. This is a contradiction to I

being preferred in $prf(D_{prf(\phi)})$, since we can consider an I' for which it holds that $I'(s) = I(s)$ for $s \in S \setminus \{x\}$, but $I'(x) \neq \mathbf{u}$. Then I' is also admissible, which contradicts that I is preferred. This means that $I(x) \neq \mathbf{u}$ for any $x \in X$. Since $\mathcal{J}$ contains any possible two-valued truth value combination of the variables in X, we can find a matching $J \in \mathcal{J}$ such that the claim holds. □

Next we show that an admissible interpretation setting all arguments except the ones in Z not to undecided is a preferred interpretation of $D_{prf(\phi)}$ and in this case there is no preferred interpretation which sets to X the same values, but leaves the others (those in $S \setminus X$) undecided.

Lemma 4.1.16. *Let $\phi = \forall X \exists Y \forall Z \psi$ be a closed QBF, $D_{prf(\phi)} = (S, L, C)$, $I \in adm(D_{prf(\phi)})$. If I assigns a truth value different from $\mathbf{u}$ to variables in $X \cup Y \cup \overline{Y}$ and $I(f) = \mathbf{t}$, then the following two statements hold.*

- *$I \in prf(D_{prf(\phi)})$; and*
- *$\nexists J \in prf(D_{prf(\phi)})$ such that $J|_X = I|_X$ and $I(s) = \mathbf{u}$ for $s \in (S \setminus X)$*

Proof. Assume $I \in adm(D_{prf(\phi)})$ and $I(s) \neq \mathbf{u}$ for $s \in S \setminus Z$ and $I(f) = \mathbf{t}$. Due to Observation 4.1.10 it holds that any admissible model sets the variables in Z to undecided, and I is admissible, that is there cannot be an I' with $I <_i I'$ such that I' is admissible in $D_{prf(\phi)}$. Hence I is preferred in $D_{prf(\phi)}$.

Now suppose that $J \in prf(D_{prf(\phi)})$ and $J(s) = \mathbf{u}$ for $s \in S \setminus X$ and $J|_X = I|_X$. Clearly $J <_i I$, hence J cannot be preferred. □

Finally we are able to prove that Skept_{prf} is Π_3^P-complete for ADFs.

Theorem 4.1.17. *Skept_{prf} is Π_3^P-complete for ADFs.*

Proof. Let $D = (S, L, C)$ be an ADF and $s \in S$. To show membership consider the co-problem, i.e. deciding whether there exists a preferred interpretation I' of D, which does not set s to $\mathbf{t}$. Checking if I' is preferred in D is a Π_2^P-complete problem, see Proposition 4.1.14.

For hardness consider a reduction from a closed QBF $\phi = \forall X \exists Y \forall Z \psi$, then construct $D_{prf(\phi)}$ as in Definition 4.1.4. First we use previous lemmata to show some properties of the constructed ADF. By Lemma 4.1.15 we can infer that for any $X' \subseteq X$ there is a set of preferred interpretations, setting X' to true and $X \setminus X'$ to false. Also due to Lemma 4.1.15, we know that any preferred interpretation is captured this way, i.e. there is no preferred interpretation setting one $x \in X$ to undecided. Further in any admissible interpretation we have that any $y \in Y$ is set to undecided, due to Observation 4.1.10. Further as shown in Lemma 4.1.12 we know that $I(f) \neq \mathbf{f}$ for any $I \in adm(D_{prf(\phi)})$. We now prove that $\mathsf{Skept}_{prf}(f, D) = yes$ iff ϕ is valid.

Assume that ϕ is valid. Now consider an arbitrary $X' \subseteq X$. Since ϕ is valid, we know that there is a $Y' \subseteq Y$ s.t. for any $Z' \subseteq Z$ we have $X' \cup Y' \cup Z' \models \psi$. Let the three-valued interpretation I be defined as

- $I^{\mathbf{t}} = X' \cup Y' \cup (\overline{Y \setminus Y'}) \cup \{f\}$;
- $I^{\mathbf{f}} = (X \setminus X') \cup (Y \setminus Y') \cup \overline{Y'}$; and

- $I^{\mathbf{u}} = Z$.

We now show that I is preferred in $D_{prf(\phi)}$. We can directly infer that $I(s) \leq_i \Gamma_{D_{prf(\phi)}}(I)(s)$ for $s \in X \cup Y \cup Z$. It remains to show that $I(f) \leq_i \Gamma_{D_{prf(\phi)}}(I)(f)$ to see that I is admissible. Since $[I]_2 = \{X' \cup Y' \cup Z' \cup (\overline{Y \setminus Y'}) \cup \{f\} \mid Z' \subseteq Z\}$ we know that any two-valued interpretation in $[I]_2$ is a model of ψ and thus also a model of φ_f. Therefore I is admissible in $D_{prf(\phi)}$. Clearly I is also preferred since only arguments in Z are undecided.

Now we show that all admissible interpretations which assign to the arguments in X the same value as I either also set f to true or are not preferred. Suppose that there is a $J \in adm(D)$ with $J|_X = I$ and $J(f) \neq \mathbf{t}$. Similarly as in the proof of Lemma 4.1.13 we know that f is never assigned the value false in an admissible interpretation and the arguments in Y and $\overline{Y}$ are assigned a value different from undecided only if f is assigned true. This means that J sets all arguments except the ones from X to undecided. Clearly then $J \leq_i I$ and J cannot be preferred. Thus for an arbitrary $X' \subseteq X$ all preferred interpretations $K \in prf(D_{prf(\phi)})$ with $K^{\mathbf{t}} \cap X = X'$ also have the property that $K(f) = \mathbf{t}$. Since ϕ is valid by assumption, we infer that for any such X' we can find corresponding values of Y' and have a preferred interpretation setting f to true and no preferred interpretation setting it otherwise. Therefore f is skeptically accepted.

Now we prove the other direction of the correctness of the reduction, i.e. assume that $\mathsf{Skept}_{prf}(f, D) = yes$. We have to show that then ϕ is valid. Due to Lemma 4.1.15, we have all combinations for setting the truth values for variables in X in the preferred interpretations. Furthermore, since for each of them the value of f is set to true, the values for $Y \cup \overline{Y}$ must be set to true or false and cannot be undecided. Therefore, again consider an arbitrary $X' \subseteq X$. There is at least one $I \in prf(D_{prf(\phi)})$ s.t. $I^{\mathbf{t}} \cap X = X'$. Further let $Y' = I^{\mathbf{t}} \cap Y$. Now we have $X' \cup Y' \cup (\overline{Y \setminus Y'}) \cup Z' \cup \{f\} \in [I]_2$ for any $Z' \subseteq Z$, since $I^{\mathbf{u}} = Z$. Since $I(f) \leq_i \Gamma_{D_{prf(\phi)}}(I)(f)$ we know that $X' \cup Y' \cup Z' \cup \{f\} \models \varphi_f$ and thus $X' \cup Y' \cup Z' \models \psi$. Since this holds for any $X' \subseteq X$ this implies that ϕ is valid. □

4.1.2 Complexity Analysis of Bipolar ADFs

Brewka and Woltran [34, Proposition 15] already showed that the verification problem of the grounded semantics is tractable if the ADF is a bipolar ADF with known links. We can significantly extend their observation to admissible and preferred semantics and show that BADFs with known link types have the same computational complexity as AFs. We note that some of the following ideas are already present in the proof of [34, Proposition 15]. Consider the characteristic function Γ_D for an ADF D. If D is a general ADF, then we basically have to consider for the computation of $\Gamma_D(I)$ for a three-valued interpretation I all two-valued interpretations from $[I]_2$ and evaluate them on the acceptance conditions. For BADFs it suffices to consider per argument s in the BADF just *one* interpretation in $[I]_2$ to decide whether $I(s) \leq_i \Gamma_D(I)(s)$. We call this interpretation *canonical* and provide a function $canon_D : 2^{S \to \{\mathbf{t},\mathbf{f},\mathbf{u}\}} \times S \times \{\mathbf{t}, \mathbf{f}\} \to 2^{S \to \{\mathbf{t},\mathbf{f}\}}$. This function takes a three-valued interpretation, an argument and a truth value as input. It results in a two-valued interpretation. The idea is that for an BADF $D = (S, L = L^+ \cup L^-, C)$ and a three-valued interpretation I on S, an argument $s \in S$ we have $canon_D(I, s, \mathbf{t}) = J$ and $J(\varphi_s) = \mathbf{t}$ iff $\Gamma_D(I)(s) = \mathbf{t}$. For computing the result $\Gamma_D(I)(s)$ we then need two such canonical interpretations. One for each of the two truth values $\mathbf{t}$ and $\mathbf{f}$.

Table 4.4: Result of $canon_D(I, s, v) = J$ for a BADF D

$v = \mathbf{t}$	$I^{\mathbf{t}}$	$I^{\mathbf{u}}$	$I^{\mathbf{f}}$
$supp_D(s)$	$J^{\mathbf{t}}$	$J^{\mathbf{f}}$	$J^{\mathbf{f}}$
$att_D(s)$	$J^{\mathbf{t}}$	$J^{\mathbf{t}}$	$J^{\mathbf{f}}$

$v = \mathbf{f}$	$I^{\mathbf{t}}$	$I^{\mathbf{u}}$	$I^{\mathbf{f}}$
$supp_D(s)$	$J^{\mathbf{t}}$	$J^{\mathbf{t}}$	$J^{\mathbf{f}}$
$att_D(s)$	$J^{\mathbf{t}}$	$J^{\mathbf{f}}$	$J^{\mathbf{f}}$

Definition 4.1.5. *Let $D = (S, (L^+ \cup L^-), C)$ be a bipolar ADF, I a two-valued interpretation on S, $s \in S$ and $v \in \{\mathbf{t}, \mathbf{f}\}$. We define a function $canon_D : 2^{S \to \{\mathbf{t},\mathbf{f},\mathbf{u}\}} \times S \times \{\mathbf{t}, \mathbf{f}\} \to 2^{S \to \{\mathbf{t},\mathbf{f}\}}$ as follows.*

- *if $v = \mathbf{t}$, then $canon_D(I, s, v) = (I^{\mathbf{t}} \cup (I^{\mathbf{u}} \cap att_D(s)), I^{\mathbf{f}} \cup (I^{\mathbf{u}} \cap supp_D(s)))$;*
- *if $v = \mathbf{f}$, then $canon_D(I, s, v) = (I^{\mathbf{t}} \cup (I^{\mathbf{u}} \cap supp_D(s)), I^{\mathbf{f}} \cup (I^{\mathbf{u}} \cap att_D(s)))$.*

That is, if $v = \mathbf{t}$, then the canonical interpretation for I and s (w.r.t. D) is given by the two-valued interpretation J as follows. Now if an argument is set to true by I, then so does J, similarly if an argument is assigned false by I, then also J assigns it false. The key difference is in the undecided arguments of I. Here J sets all *attackers of s* to true if they are undecided in I. Further J sets all *supporters of s* to false if they are undecided in I. Clearly J is two-valued and $I \leq_i J$ and thus $J \in [I]_2$. See Table 4.4 for an illustration of how to create the canonical interpretation J from I. Recall that we require for any BADF that $L^+ \cap L^- = \emptyset$.

We now prove that $\Gamma_D(I)$ can be computed solely using canonical interpretations.

Lemma 4.1.18. *Let $D = (S, (L^+ \cup L^-), C)$ be a bipolar ADF, I a three-valued interpretation on S and $s \in S$. Let $J_{\mathbf{t}} = canon_D(I, s, \mathbf{t})$ and $J_{\mathbf{f}} = canon_D(I, s, \mathbf{f})$. Then it holds that*

- *$\Gamma_D(I)(s) = \mathbf{t}$ iff $J_{\mathbf{t}}(\varphi_s) = \mathbf{t}$;*
- *$\Gamma_D(I)(s) = \mathbf{f}$ iff $J_{\mathbf{f}}(\varphi_s) = \mathbf{f}$;*
- *$\Gamma_D(I)(s) = \mathbf{u}$ iff $J_{\mathbf{t}}(\varphi_s) = \mathbf{f}$ and $J_{\mathbf{f}}(\varphi_s) = \mathbf{t}$.*

Proof. Consider the first item. By definition we have $\Gamma_D(I)(s) = \mathbf{t}$ iff for all $K \in [I]_2$ we have $K(\varphi_s) = \mathbf{t}$. What we now prove is that for all $K \in [I]_2$ we have $K(\varphi_s) = \mathbf{t}$ iff $J_{\mathbf{t}}(\varphi_s) = \mathbf{t}$. The "only-if" direction is trivially satisfied, since $J_{\mathbf{t}} \in [I]_2$.

For the "if" direction we proceed with a proof by induction on $n \geq 0$. We define some auxiliary notation. Let K_1, K_2 be two two-valued interpretations defined on S. Then $K_1 \Delta K_2 = (K_1^{\mathbf{t}} \cap K_2^{\mathbf{f}}) \cup (K_1^{\mathbf{f}} \cap K_2^{\mathbf{t}})$, i.e. results in a set of arguments assigned differently in the two interpretations. Let K be an arbitrary two-valued interpretation on S. Assume that $K(\varphi_s) = \mathbf{f}$. We prove that in this case then $K_n(\varphi_s) = \mathbf{f}$ for $(K_n^{\mathbf{t}} \setminus K^{\mathbf{t}}) \subseteq att_D(s)$ and $(K_n^{\mathbf{f}} \setminus K^{\mathbf{f}}) \subseteq supp_D(s)$ with $|K \Delta K_n| = n$ holds for K_n a two-valued interpretation. That is, if we are allowed to change the assigned value of at most n arguments in K to result in K_n with setting attackers of s from false to true and supporters from true to false, then the evaluation still yields the same result.

- Induction base: For $n = 0$ we have $K_0 = K$ and clearly $K_0(\varphi_s) = \mathbf{f}$. For $n = 1$ there are two cases, since $|K_1 \Delta K| = 1$ implies that K_1 and K assign to all arguments the same value except for one. Assume the value changes for argument x.
  - If $x \in att_D(s)$, then $K(x) = \mathbf{f}$ and $K_1(x) = \mathbf{t}$. Suppose that $K_1(\varphi_s) = \mathbf{t}$, then $K \not\models \varphi_s$ and $K \cup \{x\} = K_1 \models \varphi_s$. This is a contradiction to x being an attacker of s;
  - If $x \in supp_D(s)$, then $K(x) = \mathbf{t}$ and $K_1(x) = \mathbf{f}$. Suppose that $K_1(\varphi_s) = \mathbf{t}$, then $K \setminus \{x\} = K_1 \models \varphi_s$ and $K \not\models \varphi_s$. This is a contradiction to x being a supporter of s;
- Induction hypothesis: If K_n is a two-valued interpretation on S with $(K_n^{\mathbf{t}} \setminus K^{\mathbf{t}}) \subseteq att_D(s)$ and $(K_n^{\mathbf{f}} \setminus K^{\mathbf{f}}) \subseteq supp_D(s)$ with $|K \Delta K_n| = n$, then $K_n(\varphi_s) = \mathbf{f}$ holds.
- Induction step: Assume that the induction hypothesis is true. We now have to show that for a two-valued interpretation K_{n+1} with $(K_{n+1}^{\mathbf{t}} \setminus K^{\mathbf{t}}) \subseteq att_D(s)$ and $(K_{n+1}^{\mathbf{f}} \setminus K^{\mathbf{f}}) \subseteq supp_D(s)$ with $|K \Delta K_{n+1}| = n + 1$ it holds that $K_{n+1} \not\models \varphi_s$. Suppose the contrary, i.e. $K_{n+1} \models \varphi_s$. We now "undo" one change, i.e. consider an $x \in K \Delta K_{n+1}$. Let K_n be a two-valued interpretation on S s.t. $K_n|_{S\setminus\{x\}} = K_{n+1}|_{S\setminus\{x\}}$ and $K_n(x) = K(x)$. This implies that $|K_{n+1} \Delta K_n| = 1$. Then we can infer $|K \Delta K_n| = n$ and we know by the hypothesis that $K_n \not\models \varphi_s$. By the same reasoning as in the induction base we can infer a contradiction by supposing $K_{n+1} \models \varphi_s$, namely case one from the induction base if $x \in att_D(s)$ or case two from the induction base if $x \in supp_D(s)$. Therefore also $K_{n+1} \not\models \varphi_s$ holds.

Now to finally show the "if" direction suppose the contrary, i.e. $J_{\mathbf{t}}(\varphi_s) = \mathbf{t}$ and there is a $K \in [I]_2$ s.t. $K(\varphi_s) = \mathbf{f}$. Then clearly there is an integer n s.t. $|J_{\mathbf{t}} \Delta K| = n$. Furthermore by construction also $(J_{\mathbf{t}}^{\mathbf{t}} \setminus K^{\mathbf{t}}) \subseteq att_D(s)$ and $(J_{\mathbf{t}}^{\mathbf{f}} \setminus K^{\mathbf{f}}) \subseteq supp_D(s)$. Thus $J_{\mathbf{t}}(\varphi_s) = \mathbf{f}$, by the induction proof above. This is a contradiction. This proves the first item. The second item follows suit by an analogous induction proof (we set attackers to false and supporters to true in the canonical interpretation in this case).

For the third item we need to prove $\Gamma_D(I)(s) = \mathbf{u}$ iff $J_{\mathbf{t}}(\varphi_s) = \mathbf{f}$ and $J_{\mathbf{f}}(\varphi_s) = \mathbf{t}$. By definition $\Gamma_D(I)(s) = \mathbf{u}$ iff there are two two-valued interpretations $K_1, K_2 \in [I]_2$, s.t. $K_1 \models \varphi_s$ and $K_2 \not\models \varphi_s$. By the proof above we have that if $J_{\mathbf{t}}(\varphi_s) = \mathbf{f}$ and $J_{\mathbf{f}}(\varphi_s) = \mathbf{t}$ holds, then two such interpretation exist. What remains to be proven is the "only if" direction. We show that $\Gamma_D(I)(s) = \mathbf{u}$ implies $J_{\mathbf{t}}(\varphi_s) = \mathbf{f}$, the other statement follows directly.

Assume $\Gamma_D(I)(s) = \mathbf{u}$ and suppose $J_{\mathbf{t}}(\varphi_s) = \mathbf{t}$. Then by the proof above we have $\Gamma_D(I)(s) = \mathbf{t}$, which is a contradiction. □

Using $canon_D$ we can check if a three-valued interpretation is admissible in a BADF D. We simply have to compute the two canonical interpretations for each argument and truth value ($\mathbf{t}$ and $\mathbf{f}$) in the ADF and evaluate a formula. All these steps are inherently polynomial.

Theorem 4.1.19. Ver_{adm} *is in* P *for BADFs with known link types.*

Proof. Let $D = (S, (L^+ \cup L^-), C)$ be a BADF and I a three-valued interpretation on S. We have to show that we can check $I(s) \leq_i \Gamma_D(I)(s)$ for each $s \in S$ in polynomial time w.r.t. the size of D. By Lemma 4.1.18 we require for this to compute $canon_D(I, s, v)$. We set $v = I(s)$. By Definition 4.1.5 we can compute $canon_D(I, s, I(s))$ in polynomial time. Evaluating a Boolean formula under an interpretation is also possible in polynomial time. Doing this for all arguments is clearly also polynomial. □

Using this result we can infer the remaining upper bounds for BADFs easily. Furthermore hardness carries over from AFs.

Corollary 4.1.20. *For BADFs with known link types the following claims hold.*

- Cred_{adm} *is* NP*-complete;*
- Cred_{prf} *is* NP*-complete;*
- Ver_{prf} *is* coNP*-complete; and*
- Skept_{prf} *is* Π_2^P*-complete.*

Proof. Let $D = (S, (L^+ \cup L^-), C)$ be a bipolar ADF and $s \in S$. Regarding the first item, due to Proposition 4.1.19 we can just guess a three-valued interpretation I and check if it is admissible in D. For item two, $\mathsf{Cred}_{adm}(s, D) = yes$ iff $\mathsf{Cred}_{prf}(s, D) = yes$.

For the membership of the verification problem, consider its co-problem, i.e. checking whether I is not preferred in D. This is a problem in NP, since we can guess an I' with $I \leq_i I'$ and check if $I' \in adm(D)$, which is in NP.

Membership of the skeptical acceptance of an argument for preferred semantics in D can be seen again by considering the co-problem, i.e. whether this does not hold. Here we guess an I with $I(s) \neq \mathbf{t}$ and check if I is preferred in D.

Hardness follows in all cases from AFs, see Proposition 2.3.7 and Table 2.6. □

Finally, the grounded interpretation can be computed in polynomial time.

Proposition 4.1.21. *The grounded interpretation for a given BADF with known link types can be computed in polynomial time.*

Proof. Let $D = (S, (L^+ \cup L^-), C)$ be a bipolar ADF. Consider an algorithm as shown in Proposition 2.3.8, i.e. starting with the initial three-valued interpretation setting all arguments to undecided and then applying the characteristic function iteratively until we reach a fixed point. Now using Lemma 4.1.18 we can easily do this by using the canonical interpretations. This process requires at most $|S|$ many computations of the characteristic function. Using the canonical interpretations we can do all of the computation steps in polynomial time and thus overall also in polynomial time. Note that we require for one computation step and argument two canonical interpretations (once for checking if this argument should be set to true, and once for checking if we have to set it to false). □

This directly yields the result that all our considered reasoning tasks are in P for the grounded semantics and BADFs with known link types.

4.2 Algorithms for ADFs

In this section we generalize our search algorithms from Section 3 to (B)ADFs and the preferred semantics and present a backbone algorithm for the grounded semantics.

4.2.1 Search Algorithm for Preferred Semantics

We instantiate our generic Algorithm 1 with ADFs, by slightly adapting it. For preferred semantics we use admissible interpretations as our base semantics. The reason for this is simply that by definition we have $max_{<_i}(adm(D)) = prf(D)$ for any ADF D. We begin by defining the two relevant sets of interpretations.

Definition 4.2.1. *Let $D = (S, L, C)$ be an ADF, $\mathcal{S} \subseteq prf(D)$, $X \in prf(D)$. Define the following two subsets of preferred interpretations.*

- $InterpExcl(D, \mathcal{S}) = \{I \in adm(D) \mid \nexists I' \in \mathcal{S} \text{ s.t. } I \leq_i I'\}$;
- $GreaterInterp(D, X) = \{I \in adm(D) \mid X <_i I\}$.

These two sets of three-valued interpretations clearly mirror the sets of the generic algorithm from Section 3. That is the two sets contain admissible interpretations and $InterpExcl(D, \mathcal{S})$ restricts to those, which are not smaller or equal to the ones in $\mathcal{S}$, w.r.t. $\leq_i$. Whereas the set $GreaterInterp(D, X)$ contains admissible interpretations, which are greater to X w.r.t. $<_i$.

It is now straightforward to instantiate Algorithm 1 with preferred semantics for ADFs. We show this in Algorithm 14.

We can directly infer all desired properties for our algorithm shown earlier for AFs from the proof of Theorem 3.3.4.

Corollary 4.2.1. *Let $D = (S, L, C)$ be an ADF, $a \in S$, $M \in \{\mathsf{Cred}, \mathsf{co\text{-}Skept}, \mathsf{Enum}\}$. Then*

- *Preferred-ADF(D, a, M) terminates;*
- $\mathsf{Cred}_{prf}(a, D) = yes$ *iff Preferred-ADF(D, a, Cred) returns yes;*
- $\mathsf{Skept}_{prf}(a, D) = yes$ *iff Preferred-ADF$(D, a, \mathsf{co\text{-}Skept})$ returns no;*
- *Preferred-ADF(D, a, Enum) returns $prf(F) = \mathcal{S}$.*

Considering the running time of Algorithm 14 we state the number of membership checks in Line 2 and Line 3 the algorithm executes in the worst case. We view these two lines as *function calls*. The following proposition follows directly from Proposition 3.3.5.

Proposition 4.2.2. *Let $D = (S, L, C)$ be an ADF, $a \in S$. The number of function calls (Line 2 and Line 3 of Algorithm 14) of Preferred-ADF(D, a, M) for $M \in \{\mathsf{Cred}, \mathsf{co\text{-}Skept}, \mathsf{Enum}\}$ is $O(|S| \cdot |prf(D)|)$.*

Algorithm 14 Preferred-ADF(D, a, M)

Require: ADF $D = (S, L, C)$, argument $a \in S$, mode $M \in \{\mathsf{Enum}, \mathsf{Cred}, \mathsf{co\text{-}Skept}\}$.
Ensure: returns $\mathcal{S} = prf(D)$ if $M = \mathsf{Enum}$, *yes* if $M = \mathsf{Cred}$ ($\mathsf{co\text{-}Skept}$) and $\mathsf{Cred}_{prf}(a, D) = yes$ ($\mathsf{Skept}_{prf}(a, D) = no$), otherwise *no*

```
S := ∅
while ∃I, I ∈ InterpExcl(D, S) do
    while ∃I', I' ∈ GreaterInterp(D, I) do
        I := I'
    end while
    if M = Cred and a ∈ I^t or M = co-Skept and a ∉ I^t then
        return yes
    end if
    S := S ∪ {I}
end while
return no (or S if M = Enum)
```

For concrete instantiations of Preferred-ADF, we have to consider the ADF in question. If it is a general ADF, then the two membership checks have to be delegated to a solver capable of Σ_2^P problems. The reason being naturally the high computational complexity of the admissible semantics of ADFs. That is, to generate an admissible interpretation, which is e.g. greater than a given one, we can guess a greater three-valued interpretation and check if it is admissible. This is a problem computable by a non-deterministic Turing machine with access to an NP oracle. If the ADF in question is a BADF with known link types, we can restrict ourselves to an NP solver (like a SAT-solver), since the verification problem of admissible interpretations is in P if we have a BADF with known link types.

4.2.2 Backbone Algorithm for Grounded Semantics

In Section 4.1.1 we have shown the computational complexity of general ADFs w.r.t. grounded semantics. It turns out the verification complexity is $\mathrm{D^P}$-complete. To actually compute the grounded interpretation of a given ADF D, we have yet only seen the easy iterative approach of Proposition 2.3.8. Interestingly, we can delegate the computation also to a backbone solver. Consider the main formula $grd\_form(D)$. By Lemma 4.1.6 we know that certain atoms, directly corresponding to arguments set to true or false in the grounded interpretation of the ADF are entailed by this formula. Likewise through Lemma 4.1.7 we know that the arguments which are undecided in the grounded interpretation are *not* entailed. Clearly this suggests using a backbone solver for computing all entailed literals.

Given an ADF $D = (S = \{s_1, \ldots, s_n\}, L, C)$, if for an argument $s_i \in S$ the literal $s_i^{\mathtt{u}}$ is entailed, we can infer that s_i is not undecided in $I \in grd(D)$. If additionally s_i^i is entailed then $s_i \in I^{\mathbf{t}}$ and otherwise if $\neg s_i^i$ is entailed, then $s_i \in I^{\mathbf{f}}$. This leads to the next proposition. The Probing algorithm is presented in Section 3.4.1.

Proposition 4.2.3. *Let $D = (S = \{s_1, \ldots, s_n\}, L, C)$ be an ADF. Let the result of the backbone*

Table 4.5: Computational complexity of reasoning in ADFs and BADFs with known link types

	σ	Cred_σ	Skept_σ	Ver_σ
ADF	*grd*	coNP-c	coNP-c	$\mathrm{D^P}$-c
	adm	Σ_2^P-c	trivial	coNP-c
	prf	Σ_2^P-c	Π_3^P-c	Π_2^P-c
BADF	*grd*	in P	in P	in P
	adm	NP-c	trivial	in P
	prf	NP-c	Π_2^P-c	coNP-c

algorithm be $X = \mathit{Probing}(grd\_form(D))$. *Construct the three-valued interpretation* I *on* S *as follows for* $1 \leq i \leq n$.

$$I(s_i) = \begin{cases} \mathbf{t} \text{ if } s_i^i, \neg s_i^{\mathtt{u}} \in X \\ \mathbf{f} \text{ if } \neg s_i^i, \neg s_i^{\mathtt{u}} \in X \\ \mathbf{u} \text{ else} \end{cases}$$

Then it holds that $I \in grd(D)$.

Proof. This follows directly from Lemma 4.1.6 and Lemma 4.1.7 and by the correctness of the Probing algorithm, i.e. X contains all entailed literals of $grd\_form(D)$. □

Note that a backbone of a formula ϕ can be computed by $2 \cdot |atoms(\phi)|$ *parallel* calls to a SAT-solver. This implies that we can compute the grounded interpretation of D by polynomially many non-adaptive calls to a SAT-solver w.r.t. the number of arguments in the ADF.

4.3 Summary

In this chapter we provided a thorough complexity analysis for the most important problems on ADFs and BADFs with known link types. In Table 4.5 we summarize the results from the complexity analysis of (B)ADFs. We can recapitulate that for general ADFs the corresponding reasoning tasks are one level higher in the polynomial hierarchy, while for BADFs with known link types we stay at the same worst time complexity as for AFs.

Furthermore we showed how to generalize the generic search algorithm from Chapter 3 to preferred semantics for (B)ADFs. Here the distinction between ADFs and BADFs is crucial, since for the former we require much more powerful search engines (capable of dealing with Σ_2^P problems), whereas for BADFs we may use SAT-solvers. Search engines powerful enough for our purposes for general ADFs include QBF-solvers [117] and disjunctive answer-set programming solvers [98, 99]. Providing concrete encodings for these tasks for ADFs as well as BADFs is ongoing work. Finally we presented a backbone algorithm for the grounded semantics. For BADFs the corresponding problems of the grounded semantics are all in P.

CHAPTER 5

Implementation and Empirical Evaluation

This chapter provides the third major contribution of this thesis, the practical instantiation of our proposed algorithms in form of executable software programs and their experimental evaluation. Although our algorithms in Chapter 3 satisfy certain appealing theoretic properties, it is not clear from theory alone if they are suited for AF problems in concrete implementations.

We implemented the search Algorithm 6 in the software CEGARTIX and the SAT extension based algorithms from Section 3.4. Both were evaluated on randomly generated AFs with two parametrized creation methods. All our tools are available online (see Table 5.3 on page 123).

Therefore we give two contributions in this chapter.

- We provide implementations of our search algorithms in the form of CEGARTIX in Section 5.1.1 and algorithms based on SAT extensions for AFs in Section 5.1.2;
- and compare these programs with state-of-the-art systems in abstract argumentation in Section 5.2.

The results of the performance analysis for the search algorithms of this chapter have been published in [75, 76, 77]. The results for the algorithms based on SAT extensions have been published in [153].

5.1 System Description

We implemented two prototypical systems for experimenting with our algorithms from Chapter 3. The first one, called CEGARTIX, implements the search algorithm Decide (Algorithm 6), including its shortcuts from Section 3.3.4. This algorithm is designed to be well-suited for deciding whether an argument is credulously accepted w.r.t. semi-stable or stage semantics, or skeptically accepted w.r.t. preferred, semi-stable or stage semantics. The second system implements the algorithms based on SAT extensions from Section 3.4. We call this set of tools simply

SAT extension based algorithms. In particular we implemented an MCS-based algorithm for computing credulous and skeptical reasoning w.r.t. semi-stable semantics and enumerating the semi-stable extensions. Further we implemented also a variant for returning all skeptically accepted arguments of an AF w.r.t. semi-stable semantics and used this implementation with a simple post processor to return the eager extension. Further we implemented the backbone-based algorithm for ideal semantics.

The input language for all our systems follows the format of the well-known system ASPARTIX [85]. ASPARTIX is an answer-set programming (ASP) [30, 87, 99] approach to solve reasoning tasks for many AF semantics. For specifying an AF the following format is used, which corresponds to so-called *facts* of an ASP encoding. Given an AF $F = (A, R)$ the input for ASPARTIX is

$$\{\texttt{arg}(a). \mid a \in A\} \cup \{\texttt{att}(a, b). \mid (a, b) \in R\}$$

Example 5.1.1. *Let $F = (\{a, b, c\}, \{(a, b), (b, a), (b, c)\})$ be an AF. Then the ASPARTIX format would be*

```
arg(a).
arg(b).
arg(c).
att(a,b).
att(b,a).
att(b,c).
```

Our systems support the same input language, with the requirement that each line has at most one statement (`arg` or `att`).

5.1.1 CEGARTIX

CEGARTIX, borrowing terminology from CEGAR-style approaches [49, 50] (counter-example guided abstraction refinement) is a C++ implementation for UNIX systems of Algorithm 6 (Decide). Briefly put, this algorithm searches for a witness or a counter-example for deciding the problem at hand. It searches for such a witness or counterexample by computing an initial candidate, which is complete or admissible for the given AF. The algorithm then computes candidates which are greater w.r.t. a certain ordering in an iterative fashion. Thus it borrows some mechanics from the CEGAR approach. It is capable of solving the following reasoning tasks for an AF.

- Credulous reasoning with semi-stable and stage semantics; and
- skeptical reasoning with preferred, semi-stable and stage semantics.

These are all problems, in our context, which are complete for the second level of the polynomial hierarchy. CEGARTIX employs SAT-solvers to compute the computationally intensive tasks and handles itself the formula generation and SAT-solver calls. Initially, in v0.1, CEGARTIX was built on top of MiniSAT [84] v2.2.0. MiniSAT is used in an *incremental* mode. This

means that, if possible, during the computation we retain the solver state after a formula was found satisfiable and add further clauses directly to this state without a fresh restart. In v0.2 of CEGARTIX we additionally support clasp [102] v2.0.5 in a non-incremental mode (as well as removing incremental solving for MiniSAT). In v0.3 of CEGARTIX we now support arbitrary SAT-solvers as long as they adhere to the input/output of the SAT competitions [108]. In v0.3 we provide two variants of CEGARTIX, one, which uses again MiniSAT incrementally and one without this feature. CEGARTIX is invoked as follows.

```
./CEGARTIX file options
```

The `file` should contain the AF in the ASPARTIX input format. For the options we support the following (v0.3, for other versions see the help option).

- `-argument=string` specifies which argument should be checked;
- `-semantics=string` specifies the semantics. Allowed values are `pref` (default), `semi` and `stage`;
- `-mode=string` is used to define `cred`ulous or `skept`ical reasoning;
- `-depthArg=int` specifies the bound d for the shortcuts; $d \in \{-1, 0, 1, 2\}$ where $d = -1$ means no shortcut (Algorithms 7 and 8);
- `-oracle=string` defines the used SAT-solver. Can be one of `miniSAT`, `clasp`, or `external`. The last one requires also the next option;
- `-external=string` the path for the external SAT-solver; and
- `--help` or `--help-verb` prints further help information.

After invocation CEGARTIX prints to standard out the answer to the query with some statistics.

5.1.2 SAT Extension based Algorithms

The second implementation within the scope of our thesis is actually a set of tools. They are all based on SAT extensions as discussed in Section 3.4. The following prototypical programs are implemented for UNIX systems.

- MCS, solves $\mathsf{Enum}_{sem}(F)$, $\mathsf{AllSkept}_{sem}(F)$, $\mathsf{Cred}_{sem}(a, F)$ and $\mathsf{Skept}_{sem}(a, F)$ for an AF $F = (A, R)$ and $a \in A$;
- MCS eager, returns the eager extension;
- Backbone ideal, returns the ideal extension.

The tool MCS computes the result via an adapted version of CAMUS [115] v1.0.5, a solver for systematically computing all minimal correction sets of a formula. In the thesis we showed the variant for $\mathsf{AllSkept}_{sem}$ (Algorithm 11), but the other variants can be directly inferred (just replace the intersection of the models with a check for credulous or skeptical acceptance or with an enumeration). We note that the current implementation does not feature the optimization with using a backbone solver for computing all skeptically accepted arguments, but simply enumerates the models and computes the intersection. The program is invoked by a simple shell script `semi.sh`, which invokes a parser for the given AF and the adapted CAMUS.

```
./semi.sh file options [argument]
```

Again we provide the options.

- `cred` cred(ulous) reasoning, requires argument
- `skept` skept(ical) reasoning, requires argument
- `enum` enum(eration) of all semi-stable extensions
- `allSkept` returns all skeptically accepted arguments
- `argument` the argument for the query

The MCS eager tool calls MCS for computing all skeptically accepted arguments w.r.t. semi-stable semantics and then invokes a post-processor to find the eager extension. This post processor computes the restricted characteristic function (see Definition 2.3.16). We implemented this computation with a simple ASP encoding. The MCS eager tool is again wrapped in a small shell script `eager.sh` and invoked by

```
./eager.sh file
```

which returns the eager extension of the AF encoded as ASP facts in `file`. The postprocessing step is achieved via an ASP call to clingo [98] v3.0.4. Finally Backbone ideal computes the ideal extension of a given AF. The backbone solver JediSAT [159] v0.2 beta is used to compute all credulously accepted arguments w.r.t. admissibility, i.e. $\mathsf{AllCred}_{adm}(F)$ for an AF F. The tool then computes the ideal extension via the Algorithm 13 from [62]. This is again achieved with an ASP encoding. It is very similar to the one used for the eager extension, just the AF has to be slightly adapted. The invocation is again simple, and handled by the shell script `ideal.sh`.

```
./ideal.sh file
```

The translation from an AF to a formula is handled in all tools of this section by a C++ program, which was adapted from CEGARTIX.

5.2 Experiments

We evaluate our set of tools w.r.t. their performance compared to state-of-the art systems in argumentation. We begin with evaluating CEGARTIX. First we compare it to ASPARTIX [85], which is based on ASP encodings to compute extensions of a given AF and a semantics. On many instances we indeed experienced a better performance with CEGARTIX compared to ASPARTIX. Subsequently we inspect in more detail the choice of the base semantics of CEGARTIX. Recall that we have here multiple choices. We consider admissible and complete base semantics in our experiments. Although for many tasks this does not influence the overall performance, for some AFs the complete base semantics was superior to the admissible one w.r.t. the number of SAT calls and running time. Then we study the effect of the choice of the SAT-solver within CEGARTIX, by inspecting the performance difference between using MiniSAT, clasp and March. Finally we report on the performance of our algorithms based on SAT extensions and compare them to CEGARTIX and ASPARTIX, depending on the reasoning task.

5.2.1 Test Setup

As benchmarks we generated AFs ranging from 60 to 1000 arguments using two parameterized methods for randomly generating the attack relation, following the lines of [74] for benchmarking.[1] For each choice of parameters we created ten random AFs. We note that, as identified in [73], there is still need for diverse benchmark libraries for AFs incorporating e.g. AFs from applications.

Random AFs generated by inserting for any pair of arguments (a, b) with $a \neq b$ the attack from a to b with a given probability p.

Grid AFs that are sub-graphs of an $n \times m$ grid-structure. We consider two different neighborhoods, one connecting arguments vertically and horizontally and one that additionally connects the arguments diagonally. Such a connection is a mutual attack with a given probability p and an attack in only one direction otherwise. In the last case the direction is chosen with 0.5 probability. See Figure 5.1 for examples.

Note that the number of attacks scales linearly with the number of arguments for grid AFs, while it scales quadratically with the number of arguments for random AFs. Further we would like to stress that the generated AFs are by no means tailored to the parameters our approach is based on.

For both methods, we used the values $p \in \{0.1, 0.2, 0.3, 0.4\}$ and in addition for the grid AFs $n \in \{5, 10, 15\}$. The parameter m can then be calculated from the total number of arguments in the framework. For the number of arguments in the tests for CEGARTIX in Sections 5.2.2, 5.2.3 and 5.2.4, we distinguished between medium-sized AFs with 60, 70, 80, 90, 100, 110, 130, 160 and 200 arguments as well as larger AFs with 300, 400, 500, 600, 700, 800, 900 and 1000 arguments. For the former we generated random and grid attack relations, for the latter only grid

[1] The generator is available at the CEGARTIX web page, see Table 5.3.

Figure 5.1: Examples of grid-structured AFs

AFs. Overall this resulted in 360 random AFs, 1080 medium-sized grid AFs as well as 960 large grid AFs for each neighborhood. The total number is 4440 generated AFs.

For the tests evaluating the tools based on SAT extensions in Section 5.2.5 we created AFs for the following number of arguments with the same parameters as above. We considered random AFs (A, R) of size $|A| \in \{100, 150, 200, 225, 250, 275, 300, 325, 350\}$, which totaled in 360 AFs. For evaluating the tools based on SAT extensions, we have chosen to use larger random AFs than in the evaluation for CEGARTIX, since the SAT-based procedures appear to be able to handle small-sized AFs very well. We left grid AFs out in the comparison between CEGARTIX and the tools based on SAT extensions, since these were mostly trivial for CEGARTIX.

All tests were executed under OpenSUSE with Intel Xeon processors (2.33 GHz) and 49 GB memory. Further we set a timeout of five minutes for each individual run and measure the whole time for the workflow from Figure 3.1, i.e. combining parsing, solving and if applicable post-processing time. When two runs of two different systems were successfully computed in time we compared the respective results and found in all cases that the systems agree, i.e. the performance tests were also used as test instances for correctness of the implementation.

5.2.2 Evaluation of CEGARTIX

We compare the performance of CEGARTIX to that of a recently proposed, state-of-the-art argumentation reasoning system [74] that exploits advances in answer-set programming (ASP) via the so-called `metasp` approach[2]. This system is a further improvement of the ASPARTIX system [85]. For comparing CEGARTIX with the ASP-based approach, we used the state-of-the-art no-good learning disjunctive ASP solver claspD [58] (v1.1.2) combined with the grounder gringo [100] (v3.0.4).

For this comparison, we employed the base semantics com for $\sigma \in \{prf, sem\}$, and cf for stg within CEGARTIX. The parameter for the shortcuts function (Algorithm 8) for $\sigma \in \{sem, stg\}$ was set to $d = 1$. The reason for setting the shortcut parameter d to 1 was to allow for semi-stable and stage semantics a fast computation if a stable extension exists ($d = 0$ is sufficient for this) or if a semi-stable or stage extension with a range covering almost all arguments exists. On the other hand by setting d to 1 we keep the number of checks in the shortcut low.

[2]Skeptical and credulous reasoning in `metasp` is done by introducing constraints in the so-called meta answer-set programs; for details, see [101].

Table 5.1: Timeouts encountered with ASPARTIX on medium-sized random/grid AFs

Arguments	Skept_{prf}	Cred_{sem}	Skept_{sem}	Cred_{stg}	Skept_{stg}
100	0/1	0/0	0/0	0/0	0/3
110	0/2	0/1	0/6	0/7	0/11
130	0/2	0/1	0/4	0/77	0/120
160	0/4	0/25	0/36	0/349	1/411
200	1/18	0/85	29/89	22/497	44/562

We let the solvers compute queries for all the considered semantics, that is for the semantics prf, sem and stg, using skeptical reasoning with all three, and additionally credulous reasoning with the latter two. For the random instances three different arguments were queried, while for the grid instances we used five arguments. This resulted in a total of 107400 benchmark instances.

We considered two metrics for comparison, namely, (cumulative) running time in seconds for all queries of a particular reasoning mode without timed out instances, and separately the number of timeouts. This has the effect that the figures depicting cumulative running times exclude timed out runs, and therefore show a faster performance if the number of timed out queries was high. For performance comparison of CEGARTIX with ASPARTIX, using the above mentioned `metasp` approach, we considered only medium-sized AFs.

Figures 5.2 and 5.3 present results on comparing ASPARTIX and CEGARTIX. On the left, the line plots comparing the cumulative running times (using logarithmic scale) are shown. On the right, the scatter plots present running time differences of individual queries, *including* timed out instances. The dotted lines denote ASPARTIX and the solid lines show the performance for CEGARTIX. For the plots on the left side, the running times for queries made on AFs with a particular number of arguments are grouped together. The number of timeouts are shown in Table 5.1 for ASPARTIX. Using CEGARTIX with the parameters noted above, no timeouts were encountered.

First consider Figure 5.2 for skeptical reasoning with random AFs (upper left plot). CEGARTIX behaves very similarly for all three semantics with respect to the considered metric, and outperforms ASPARTIX. The difference of ASPARTIX and CEGARTIX is more distinct with semi-stable and stage semantics and less so for preferred semantics. Note that for AFs with 200 arguments, 29 and 44 timeouts were encountered for ASPARTIX under skeptical semi-stable and stage reasoning, respectively.

The grid structured AFs typically performed quite differently than random AFs, which is why we investigate them separately in Figure 5.2 (lower left plot). In many cases the running times were in fact very close to 0. One can see that grid AFs are mostly trivial for CEGARTIX. CEGARTIX solved skeptical reasoning tasks with preferred and semi-stable semantics combined within 10 seconds, while ASPARTIX took significantly more time. The stage semantics is the only one here with higher running times for CEGARTIX. We will consider the larger grid AFs below for a more detailed study of CEGARTIX.

Credulous reasoning shows in general a similar picture to skeptical, with the exception that

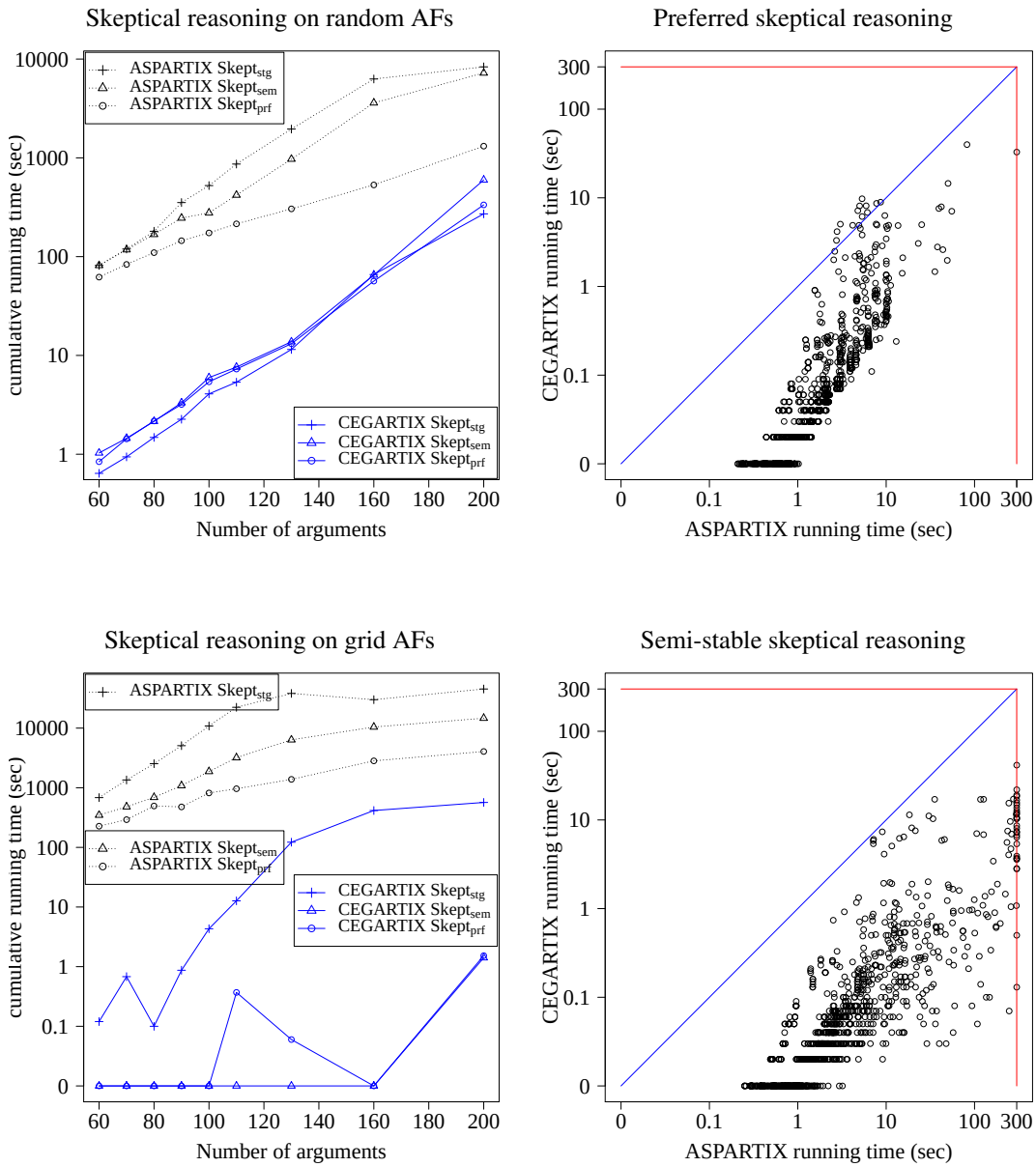

Figure 5.2: Comparison of ASPARTIX and CEGARTIX: Cumulative running times using logarithmic scale (left), scatter plots (right).

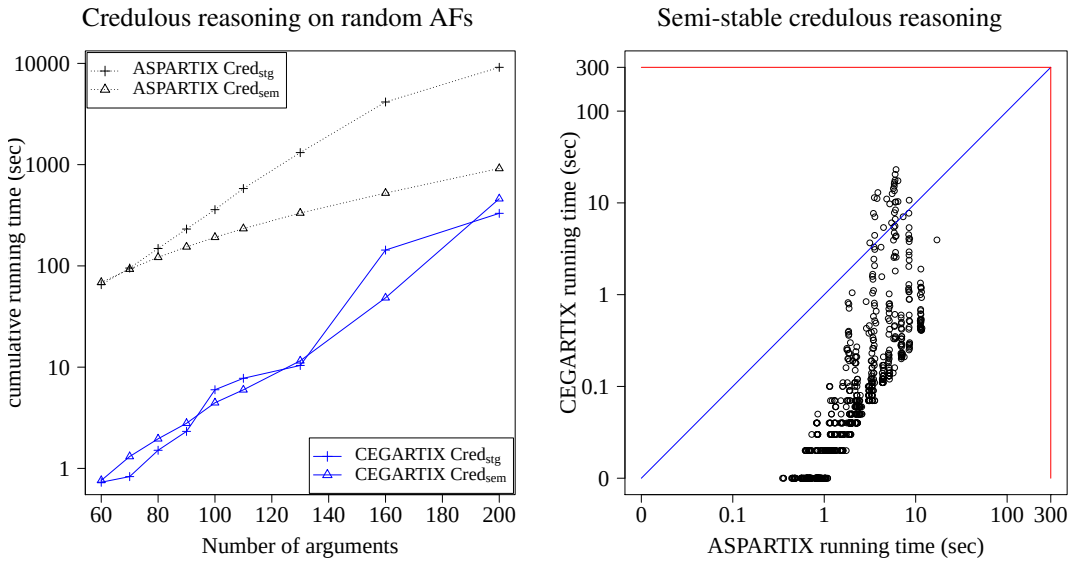

Figure 5.3: Comparison of ASPARTIX and CEGARTIX: Cumulative running times using logarithmic scale (left), a scatter plot (right).

for semi-stable reasoning on random AFs, ASPARTIX and CEGARTIX appear to be closer with respect to performance, which can be seen in Figure 5.3 (on the left). However, a closer look at the encountered timed out instances reveals that CEGARTIX significantly outperforms ASPARTIX also in this case.

For deeper understanding, we also looked into scatter plots comparing ASPARTIX and CEGARTIX. These depict running time comparisons for individual instances for both solvers, including timed out instances. The scatter plots in Figures 5.2 and 5.3 are for skeptical preferred, semi-stable reasoning, and credulous semi-stable acceptance. The x-axis shows the running time of individual queries with ASPARTIX and the y-axis with CEGARTIX on the same instance. Due to the low running times of CEGARTIX on grid AFs, we only considered random AFs for scatter plots.

For preferred semantics, many queries are in favor of CEGARTIX. Except for a few instances, which are drastically faster for CEGARTIX, the difference is usually within a few seconds, however. Semi-stable semantics yields a different picture, depending on the reasoning mode. For skeptical queries CEGARTIX clearly outperforms ASPARTIX on basically all instances. Credulous reasoning overall also is solved faster by CEGARTIX. A number of queries, however, were computed more efficiently by ASPARTIX. We omit the scatter plots for stage reasoning as they show a behavior similar to skeptical semi-stable semantics.

In general, stage and semi-stable semantics show a similar behavior, which reflects their similar nature as well as the similar procedures used for the evaluation of the semantics. It seems that, when considering range-maximality, the choice of the base-semantics (cf or com) has only minor effects on the qualitative behavior of the semantics. We will see a similar behavior in the next section, when comparing adm and com as base-semantics for semi-stable.

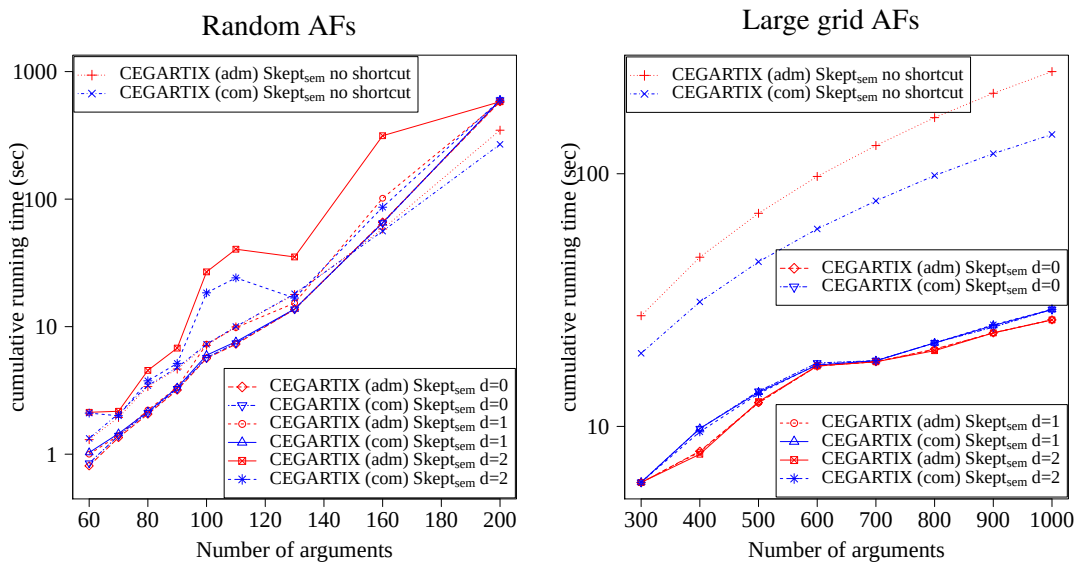

Figure 5.4: The effect of different parameter settings for the shortcut function and the choice of base semantics for skeptical semi-stable reasoning.

Finally we made some experiments regarding the memory usage of the algorithms. We let both CEGARTIX and ASPARTIX compute the skeptical acceptance under stage semantics on AFs with 200 arguments (these are the hardest instances used in the above comparison) and additionally enforced a hard limit of 4 GB of memory, which was never reached. This indicates that memory consumption is not an issue for our algorithms.

5.2.3 Impact of Base Semantics and Shortcuts within CEGARTIX

In this section we investigate the choice of the base semantics for solving the reasoning problems, as well as the effect of the shortcuts for semi-stable semantics, within CEGARTIX. We will again distinguish between random AFs and grid AFs, since CEGARTIX behaves quite differently on these two classes of AFs. We again queried three arguments for random AFs and five for grid AFs.

First we consider results regarding the impact of the base semantics. On random AFs there appears to be only a minimal effect for most CEGARTIX parameters and reasoning tasks. The cumulative running times were comparable between admissible and complete base semantics. Figure 5.4 shows the performance resulting from applying different combinations of (i) parameter settings for the shortcut $d = 0, 1, 2$ and (ii) the alternative base semantics for semi-stable semantics and skeptical reasoning. CEGARTIX with the complete base semantics is slightly faster than CEGARTIX using the admissible base for $d = 2$, for example. For credulous reasoning the difference between the choice of the base semantics was even smaller. The results for preferred semantics were similar.

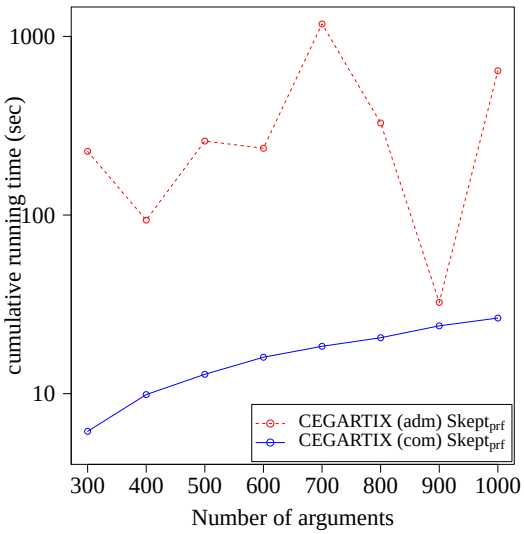

Figure 5.5: Performance comparison for the alternative choices of the base semantics for preferred skeptical reasoning on large grid AFs.

The choice of the base semantics, however, has a stronger influence for reasoning under the preferred semantics on grid AFs. For certain queries the number of oracle calls rises tremendously using the admissible base semantics. This in turn can be seen in the cumulative running times in Figure 5.5 for large grid AFs. In contrast, only a few SAT-solver calls are made when using the complete base semantics. We encountered a higher number of SAT-calls in 130 out of 9600 queries for preferred semantics on large grid AFs with admissibility base semantics. Algorithm 6 entered in these queries the outer while loop (line 7) more than 40 times, i.e. had to consider a new admissible set many times, while on the remaining 9470 queries this number was lower. In fact the average number the loop was entered in these 130 queries was 1179.01. This is also reflected in the running time. From these 130 queries we have that 75 had a running time of five up to 277.08 seconds. These make up the largest proportion of the cumulative running time in Figure 5.5. Additionally we encountered 305 timeouts using admissibility base semantics for preferred reasoning and large grid AFs. For complete base semantics either using the shortcut was enough, or otherwise the query was decided without entering the outer loop in all large grid AFs, i.e. indicating that there did not exist a complete extension without the queried argument.

Regarding the number of SAT-calls for semi-stable semantics and skeptical reasoning, out of the 1080 queries on random AFs, using the shortcut with $d = 0$ the algorithm could solve 996 within the shortcut and the remaining ones with a single application of the outer while loop (line 7) of Algorithm 6, for both base semantics. Since the number of SAT calls within the outer while loop is linearly bounded by the number of arguments in the AF, this means that the overall number of SAT-calls was low in these cases. Without the shortcuts also the outer while loop was entered at most once for semi-stable skeptical reasoning.

For the performance of the shortcuts, we experienced a slight decrease of performance for random AFs using the shortcuts for semi-stable skeptical reasoning, see Figure 5.4 on the left. For the grid AFs we again have a different picture. Here, as can be seen in Figure 5.4 on the right, the shortcuts decrease the overall running time, due to the fact that for grid AFs with a neighborhood of at most four arguments we have no odd-cycles in the AFs, which in turn means that computing the stable extensions is sufficient. This is reflected in the running time using the shortcuts for semi-stable semantics. Here it is even sufficient to consider depth $d = 0$.

5.2.4 Effect of the Choice of SAT Solver within CEGARTIX

In CEGARTIX, we can use Minisat either in an incremental fashion or non-incrementally. Furthermore, there is a command line option for employing an external SAT solver binary as the core solver.

We investigated how the choice of the core SAT solver effects the performance of CEGARTIX. In addition to Minisat and Clasp, we used March_nh (as submitted to the SAT Challenge 2012) as an external solver. The results are shown in Figure 5.6 for both the random and grid instances. For the random instances two arguments were queried and for the grid instances three arguments were queried in this test. First, one can observe that March_nh is not a competitive choice as the core SAT solver. Furthermore, we excluded the following number of timeouts for March_nh on the grid instances:1 instance with 600 nodes, 2 with 700, 4 with 800, 1 with 900, 11 with 1000 nodes. On the random instances, we observe quite similar performance when employing Minisat (non-)incrementally; in other words, it appears that for these instances incrementality does not improve performance. Employing Clasp appears to yield slightly better scaling than employing Minisat. However, the situation is different on the grid instances. First, we observe that non-incremental Minisat clearly yields better performance than Clasp on these more structured instances. Furthermore, employing the incremental interface of Minisat gives an additional improvement of a similar order over the non-incremental employment of Minisat.

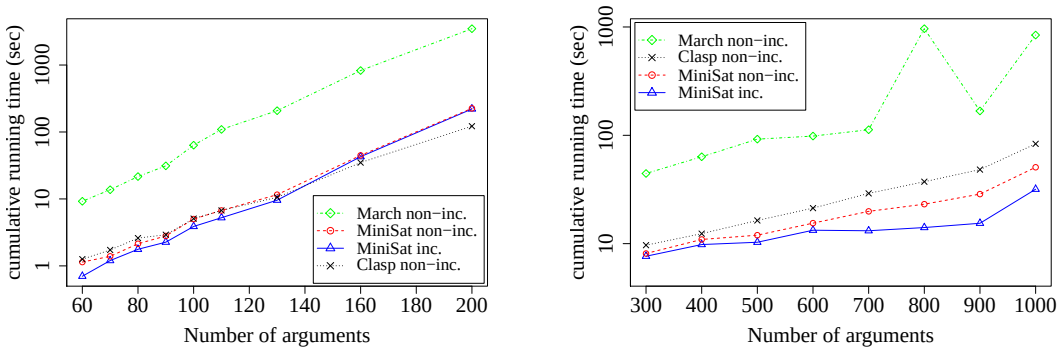

Figure 5.6: Comparison of different variants of CEGARTIX (non-incremental and incremental applications of Minisat, non-incremental application of Clasp): cumulative running times over the random instances (left) and grid instances (right).

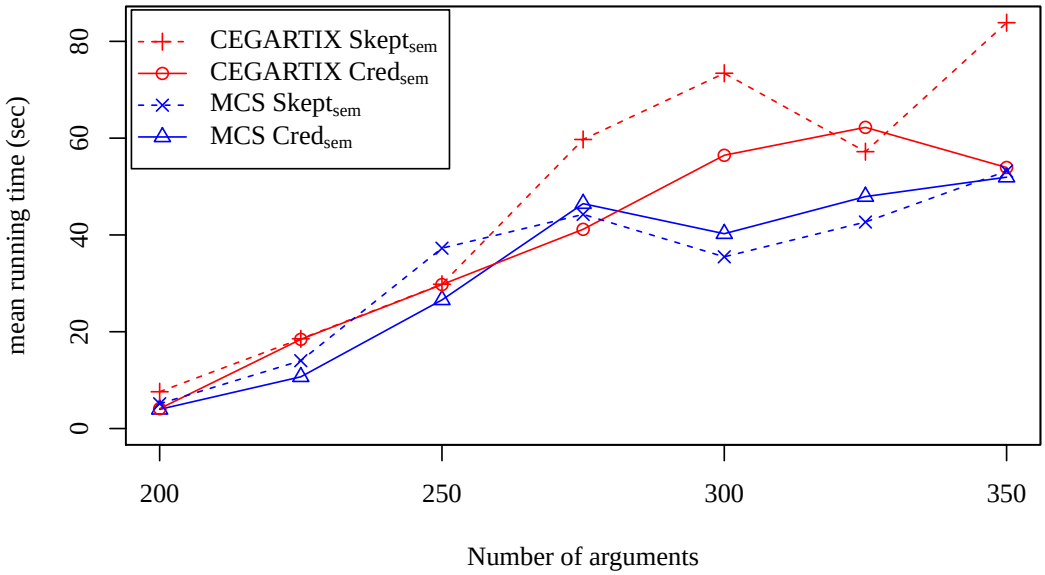

Figure 5.7: Mean running time for CEGARTIX and the MCS-based algorithm.

5.2.5 Evaluation of SAT Extensions based Algorithms

For evaluating our tools MCS, MCS eager and Backbone ideal we conducted tests for the following reasoning tasks.

- Credulous and skeptical reasoning for semi-stable semantics
- Enumeration of all semi-stable extensions
- Computing the ideal extension
- Computing the eager extension

We compare credulous and skeptical reasoning for semi-stable semantics with CEGARTIX. We chose version 0.1a of CEGARTIX for our tests, since in this version CEGARTIX is able to

Table 5.2: Number of solved instances for CEGARTIX and the MCS-based algorithm.

reasoning task \ $\|A\|$	200	225	250	275	300	325	350	% solved overall
CEGARTIX Cred$_{sem}$	120	120	112	91	71	64	50	74.8%
CEGARTIX Skept$_{sem}$	120	120	104	84	69	60	48	72%
MCS Cred$_{sem}$	117	120	117	111	85	77	76	83.7%
MCS Skept$_{sem}$	117	120	116	102	79	73	73	81%

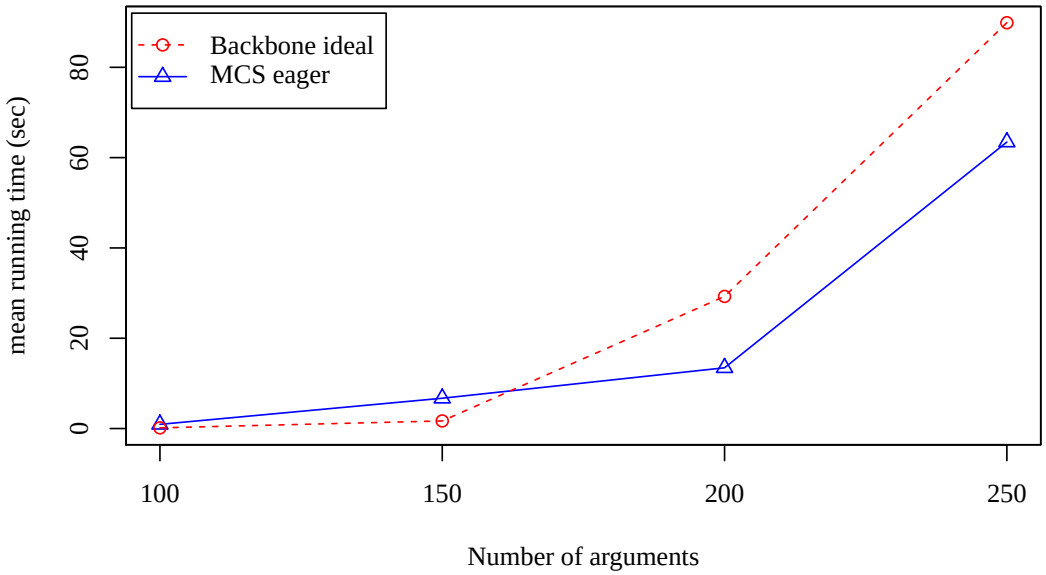

Figure 5.8: Mean running time for computing the ideal respectively eager extension.

utilize incremental SAT-solving techniques and further versions of CEGARTIX mainly feature capabilities to use different SAT solvers. We let both CEGARTIX and the MCS-based approach compute the queries for three pre-specified arguments for random AFs with at least 200 arguments, i.e. credulous and skeptical acceptance with three different arguments. This gives us 120 queries per AF size and in total 840 queries. The results are summarized in Figure 5.7, where we show the mean running time in seconds for both approaches, *excluding* timed out runs. We grouped together queries on AFs with the same number of arguments. We see that the MCS-based approach is competitive and outperforming CEGARTIX. Note that by excluding the timeouts, which are shown in Table 5.2, the figures slightly favor CEGARTIX for large AFs.

It is interesting to note that the expected edge density, which we set between 0.1 and 0.4 appears to play an important role for the performance of the SAT-based approaches. Out of the total 212 timeouts encountered for credulous reasoning under semi-stable semantics for the solver CEGARTIX for all considered queries, 113 were on AFs with 0.1, 75 on AFs with 0.2 and 24 on AFs with 0.3 expected edge density. Showing a similar picture, the MCS-approach had 137 total timeouts and 105 of them with 0.1 and 32 with 0.2 expected edge density. For skeptical reasoning the results are similar.

For comparing our MCS-approach w.r.t. the enumeration of all semi-stable extensions we use an ASP approach [74] utilizing `metasp` for our performance test. For this ASP approach we used gringo 3.0.5 and claspD 1.1.4 [98]. We tested both approaches on the same AFs as for the credulous and skeptical reasoning under semi-stable semantics and out of the 280 AFs we tested, the MCS-approach solved (i.e. enumerated all semi-stable extensions) 172 instances

Table 5.3: Overview of implementations

System name	URL	algorithm
CEGARTIX	`http://www.dbai.tuwien.ac.at/proj/argumentation/cegartix/`	Algorithm 6
SAT Extensions based tools	`http://www.dbai.tuwien.ac.at/proj/argumentation/sat-based/`	Algorithms 11,12,13

while ASP with `metasp` solved only seven instances within the time limit of five minutes.

For ideal and eager semantics, we report the mean computation time for AFs of size $|A| \in \{100, 150, 200, 250\}$ in Figure 5.8 to compute the unique extension. Hence we compute the ideal respectively eager extension for each AF separately, which gives us 40 computations per number of arguments and 160 such calls in total per semantics. We encountered one timeout for eager reasoning on AFs with size 200 and ten with AFs of size 250. For ideal reasoning we encountered 17 timeouts with AFs of size 250. Other systems capable of solving these tasks are e.g. ASPARTIX, but which could only solve, within the time frame of five minutes, instances with a low number of arguments, i.e. AFs with less than 30 arguments. For this the reason we excluded this system in a comparison with our implementations. For ideal reasoning ASPARTIX uses a complex ASP encoding technique [92] for the DLV solver [112] (we used build BEN/Dec 16 2012 of DLV). The system ConArg [26], which is based on constraint satisfaction solvers, appears to be more competitive. ConArg is a visual tool, so more detailed performance comparisons are subject of future work (very recently a command-line tool was presented for ConArg [25]). We tested some randomly generated AFs with 100 and 150 arguments and let ConArg compute the ideal extension, which it solved within ten seconds for the AFs with 100 arguments and took more than a minute for AFs with 150 arguments, but one has to factor in that a graphical representation of large graphs may consume a part of the resources needed for solving the problem.

5.3 Summary

In this chapter we evaluated our algorithms empirically by implementing them and subsequently comparing their performance to state-of-the-art systems in abstract argumentation. We implemented the solver CEGARTIX for credulous reasoning for semi-stable and stage semantics and skeptical reasoning tasks for preferred, semi-stable and stage semantics. Further we implemented the algorithms based on SAT extensions. We studied the performance of these systems on generated instances of two kinds, random and grid-structured AFs. CEGARTIX showed good performance compared to ASPARTIX, a system based on ASP encodings for argumentation. Further we investigated parameter choices within CEGARTIX and found that on certain instances it can make a significant difference which base semantics was chosen. In particular for preferred skeptical reasoning the complete base semantics was outperforming the admissible base semantics. The shortcuts for CEGARTIX influenced the running time. The shortcuts

Table 5.4: Supported reasoning tasks and solver

System name	supported reasoning modes	internal solvers
CEGARTIX	Skept_{prf}, Skept_{sem}, Skept_{stg}, Cred_{sem}, Cred_{stg}	MiniSAT [84] v2.2.0, clasp [102], v2.0.5, other solvers
MCS	Enum_{sem}, Cred_{sem}, Skept_{sem}, $\mathsf{AllSkept}_{sem}$	CAMUS [115], v1.0.5
MCS eager	Enum_{eager}	CAMUS [115], v1.0.5
Backbone ideal	Enum_{ideal}	JediSAT [159], v0.2 beta

decreased the overall running time for random and grid AFs, but in our experiments the performance difference between $d = 0$, $d = 1$ or $d = 2$ was negligible. The used SAT-solver within CEGARTIX can be configured and both clasp and MiniSAT showed comparable performance. Using incremental SAT-solving increased the overall performance.

We compared the performance of the algorithms based on SAT extensions with CEGARTIX and ASPARTIX. CEGARTIX performed similar as the SAT extension tools, but the latter showed less timeouts for larger random AFs. For enumeration solution we experienced a better performance for the SAT extension based tools compared to ASPARTIX with the `metasp` approach.

All our tools are publicly available on the web, see Table 5.3. The used AF instances are available on these web pages as well. In Table 5.4 we summarize the supported reasoning modes and solvers of the implementations.

CHAPTER 6

Discussion

In this chapter we recapitulate our contributions, point to important related work and give directions for future research.

6.1 Summary

In this thesis we developed novel algorithms for two formalisms in abstract argumentation, Dung's argumentation frameworks (AFs) and the more recent abstract dialectical frameworks (ADFs). We provided algorithms for solving credulous and skeptical acceptance of an argument in an AF, as well as for enumeration of all extensions. The algorithms are capable of solving these tasks for the preferred, semi-stable, stage, ideal, eager and stage-ideal semantics. Further we showed how to generalize the algorithms to work with ADFs for preferred and grounded semantics.

Many existing approaches for algorithms in abstract argumentation can be classified into either direct approaches or reduction based approaches. The former aims at building a dedicated procedure from scratch, typically exploiting domain specific knowledge for increasing the overall performance. The idea of the latter approach is to encode the problem at hand into another usually declarative language, for which sophisticated solvers exist. Many existing reduction based approaches are monolithic in the sense that a single encoding is constructed for the problem at hand. One of the main benefits of reduction approaches is that they may re-use software and can often be instantiated faster than direct approaches. We choose the best of the two worlds of direct and reduction approaches and developed a hybrid variant. We delegate certain complex subtasks to search engines, but hereby do not translate the whole problem in one monolithic encoding.

The problems we solve in this thesis are all "beyond" NP, i.e. hardness was shown for almost all considered tasks for a class above NP. A purely monolithic efficient reduction approach requires search engines matching the complexity of the problem to solve. In contrast, our novel algorithms iteratively call engines to compute subtasks, which are less complex than the overall

task. We showed that the number of required calls in the worst case is related to two inherent parameters of the given AF/ADF. This means if these parameters are low, then so is the number of required calls.

We classified our algorithms into two categories:

- search algorithms; and
- algorithms utilizing SAT extensions.

We showed that both approaches can solve numerous reasoning tasks for AFs and ADFs. The basic idea of the search algorithms to compute a reasoning task for a semantics σ is to fall back to a simpler semantics σ', called the base semantics. From this base semantics we require that its maximal elements w.r.t. a preorder are exactly those in σ. Now we let the solvers compute candidates from the base semantics and then iteratively maximize them to find a maximal candidate in σ. Using this approach we can solve tasks for enumerating all solutions or decide credulous or skeptical acceptance of σ for an argument a. For AFs we applied search algorithms for preferred, semi-stable and stage semantics and for ADFs we applied them to preferred semantics.

We considered two SAT extensions in our work, the minimal correction sets (MCSes) and backbones. We derived algorithms based on MCSes in a very natural way for the semi-stable and stage semantics. Using a simple post processor we showed how to find the eager extension and the stage-ideal extension. For the ideal semantics we instantiated an existing algorithm using backbones.

Existing algorithms for computing MCSes and backbones from the SAT community rely on iterative SAT solving. We slightly adapted an algorithm for MCSes and directly used an algorithm for backbones for our purposes. For the computation of subtasks for our search algorithms, we can, for AFs, in all cases use a SAT-solver, since the corresponding decision problems are in NP or in coNP. For ADFs the picture is different. Here we first had to establish a clear theoretical understanding of the computational complexity which was missing in the literature for almost all considered problems. The result of this analysis is that the reasoning tasks for ADFs are "one step up" in the polynomial hierarchy compared to their counterparts on AFs. This directly implies that our search algorithms, under standard complexity theoretic assumptions, cannot be instantiated with a SAT-solver, unless we take an exponential blow-up into account. However we can use solvers capable of solving tasks of the second level of the polynomial hierarchy (e.g. solving tasks which are in Σ_2^P). Examples of such solvers are quantified Boolean formulae (QBF) solvers or answer-set programming (ASP) solvers capable of dealing with disjunctive ASP programs. As a side result from our complexity analysis we developed a backbone algorithm for computing the grounded interpretation of an ADF.

A second major contribution of our complexity analysis is the investigation of BADFs with known link types. Here we proved that the reasoning tasks on these BADFs have the same complexity as their counterparts on AFs. This clearly suggests to use again less powerful solvers for BADFs such as SAT-solvers or ASP solvers (and normal logic programs).

To show the feasibility of our approach we conducted experiments on prototypical implementations of our algorithms for AFs. The experiments revealed a significant performance boost

compared to ASPARTIX, a state-of-the-art system for computing many semantics for AFs. CEGARTIX is our prototype for instantiating the search algorithms. For the algorithms based on SAT extension, we provide a suite of tools. All programs are available online (see Table 5.3).

6.2 Related Work

Naturally no academic work exists purely in a vacuum. In this section we describe the most important related works to our thesis. Algorithms applicable for problems in abstract argumentation are discussed in a recent survey [47]. As mentioned earlier one can distinguish between *reduction-based* and *direct* approaches. The former aim at translating a given problem to another, usually well-studied problem, for which efficient search engines are available. The goal of the latter approach is to directly develop algorithms tailored to the given problem domain. We review the most important related work in this regard and give further pointers to the literature. Subsequently we discuss algorithms based on decompositions of the given AF and conclude this section with further related work.

Reduction based Approaches The main directions for reduction based approaches for abstract argumentation feature SAT, QBFs, answer-set programming (ASP), constraint satisfaction problems (CSP) and equational systems. As a common feature all these formalisms specify some form of constraints to be satisfied. By carefully choosing the right constraints one can then implement the problems from abstract argumentation in these formalisms. We highlight particularly important related work.

The most well-known system is probably ASPARTIX [74, 85, 92], a reduction-based approach with ASP as the target language. This system features an easy-to-use interface. One simply has to provide a text file with the AF as input in a very simple language. Our systems support this language as well (see Section 5.1). This format is sometimes also called ASPARTIX format. Many systems in abstract argumentation are compared with ASPARTIX [25, 45, 74, 129, 131]. Other ASP based approaches have been collected in a recent overview paper [148].

Another approach using CSP [26] is ConArg. This system is capable of computing many semantics and features a GUI. Reductions to equational systems have been studied in [96].

Related work in the SAT area is, for obvious reasons, very close to our work. The most fundamental work is by Besnard and Doutre in [21], who encoded several AF semantics in Boolean formulae, such that the extensions and models correspond. Many of their encodings, or in cases key ideas of their encodings, are visible in more recent approaches. We also partially use their encodings. Although [21] provided also encodings for preferred extensions, it is clear from the computational complexity point of view that we require for this to work more powerful engines than SAT-solvers. In [2, 86] several problems for AFs were reduced to encodings for QBFs.

The most relevant SAT based approach is PrefSat by Cerutti et al. [45], which enumerates all preferred labelings. In contrast to other SAT based systems, this is not a *monolithic* reduction approach. This means that not a single encoding is constructed, which has as its solutions exactly the answer to the original problem, but multiple calls to a SAT-solver are applied. Indeed the overall scheme is similar to our search algorithms. The key difference lies actually in the use

of labelings instead of extensions in the Boolean encodings. Unlike CEGARTIX, the resulting system of their work enumerates all preferred labelings.

PrefSat encodes labelings of an AF $F = (A = (x_1, \ldots, x_n), R)$ by generating three variables per argument, i.e. the set of variables in the constructed formula are $\{I_i, O_i, U_i \mid 1 \leq i \leq |A|\}$. These then correspond in the final result naturally to a labeling. This means a three-valued labeling J corresponds to a model K if $J = (I, O, U)$ with $I = \{I_i \mid K(I_i) = \mathbf{t}\}$, $O = \{O_i \mid K(O_i) = \mathbf{t}\}$ and $U = \{U_i \mid K(U_i) = \mathbf{t}\}$. Encoding the basic constraint that for every argument exactly one labeling is assigned can be done as follows.

$$\bigwedge_{1 \leq i \leq |A|} \big((I_i \vee O_i \vee U_i) \wedge (\neg I_i \vee O_i) \wedge (\neg I_i \vee \neg U_i) \wedge (\neg O_i \vee U_i)\big)$$

Then one can encode the conditions for a labeling to be complete (see Definition 2.3.11) by conjoining certain subformulae. For instance the formula

$$\bigwedge_{1 \leq i \leq |A|, \exists (y, x_i) \in R} \Big(\bigwedge_{(x_j, x_i) \in R} (\neg I_i \vee O_j)\Big)$$

encodes that if there is an attacker for the argument x_i, then one creates a clause for each such attacker x_j such that if x_i is assigned the labeling in (I_i is true), then x_j must necessarily be out (O_j is true). One can encode the other conditions as well in Boolean logic. Several equivalent formulae for doing this have been investigated by Cerutti et al. [45]. In our encoding for complete extensions (see Lemma 3.3.7 and Proposition 3.3.8) we can infer a labeling of an argument x via the value it is assigned in a model and its "range" variable $\overline{x}$. If $\overline{x}$ is false in the model, then x is undecided. Otherwise if $\overline{x}$ is true and x is false, then x is out. If x is true in the model, then x is in. The labelings/extensions can be directly computed with both our and their encoding. Comparing these encodings w.r.t. their performance is subject to future work.

The idea of the iterative SAT algorithm PrefSat is very close to our Algorithm 3. The basic methodology is to iteratively find complete labelings and extend them to preferred labelings by iterative calls to a SAT-solver. If such a preferred labeling is found, one excludes it from future iterations and searches for a fresh candidate. At the end all preferred labelings are enumerated.

The system PrefSat from [45] is capable of using two state-of-the-art SAT-solvers, PrecoSAT (from the 2009 SAT competition) and Glucose [5]. PrefSat was compared with ASPARTIX [85] (also using `metasp`, see [74]) and the system presented in [131]. In the experiments the iterative SAT approach outperformed the other systems. We view the results of Cerutti et al. as an independent development with many similarities to our approach. Their performance analysis draws a similar conclusion than ours, namely that the iterative SAT scheme performs very well compared to monolithic systems. Thus our results and theirs complement each other and show that the overall idea behind our approaches is beneficial for implementing hard problems arising in abstract argumentation.

Direct Algorithms for Abstract Argumentation Direct approaches include labeling-based approaches [38] and game-theoretic approaches [126]. For labeling based approaches several procedures have been published [57, 126, 129, 130, 131, 152]. Central for these algorithms is that if one fixes a label of some argument, then for complete labelings one can propagate the

status to other arguments. Game-theoretic approaches base the computation on a game between proponent and opponent. Both "players" act consecutively and play arguments, accordingly to attacks and game rules. If the proponent has a winning strategy, then an argument is acceptable in an extension. Two systems for game-theoretic approaches have been implemented, Dungine [142] and Dung-O-Matic[1].

We recall here the recently developed labeling based Algorithm 9 from [129]. This algorithm shows important features of labeling based algorithms. The algorithm uses five labelings, in, out, $undec$, $must\text{-}out$ and $ignored$. The intuition behind these labelings is the following. The first three labels again are interpreted as accept the argument, reject the argument (attacked by an argument which is in) and no decision was yet applied to this argument w.r.t. the labeling. The last two labels are new and are auxiliary and technical labelings. The labeling $must\text{-}out$ means that the current assigned labels are not a "legal" complete labeling in the sense that an argument is in, but one of its attackers is not out, but $must\text{-}out$. This means this argument should be assigned at the end of the algorithm out, otherwise this will not constitute to a complete (and thus also preferred) labeling. The $ignored$ label is simply used to state that we try to find a complete labeling where we do not accept this argument. Therefore the two new labelings are tailored specifically to Algorithm 9 of [129] and are no argumentative labelings in the sense that they describe the acceptance or rejection of arguments.

Labeling based algorithms often work with so-called transitions, which re-label arguments. In the case of Algorithm 9 from [129] there are two types of transitions, $IN\text{-}TRANS$ and $IGNORE\text{-}TRANS$. Both are functions, which take an argument x as input. The former re-labels x to in, all attacked arguments, i.e. all y with $(x, y) \in R$ to out. All attackers of x are set to $must\text{-}out$, if they are not already out. Clearly these should be out, but we still require to accept an attacker of it, or in other words a defender of x. The $IGNORE\text{-}TRANS$ simply sets the argument to $ignored$.

Algorithm 9 of [129] starts with an initial labeling, which sets all argument to $undec$ and has a data structure $E_{\text{preferred}}$. This algorithm is a recursive function, which takes as input a labeling. If the current labeling L does not have any undecided arguments, then it is checked (i) if there is no argument set to $must\text{-}out$ and (ii) if the arguments set to in by L are a subset of any previously found solution in $E_{\text{preferred}}$. If this is not the case, then a preferred labeling is obtained and added to $E_{\text{preferred}}$. Otherwise there exists an argument set to undecided. We select such an argument which is labeled $undec$ and branch by applying the two types of transitions to this argument. At each branch we simply call the recursive function again.

The recursive calls of this algorithm for the AF $F = (\{a, b, c\}, \{(a, b), (b, a), (b, c)\})$ are shown in Figure 6.1.[2] We shortened the labels i, o, u, ig, mo for the in, out, undecided, ignored and must-out labels. The leaves all have no undecided arguments (here the recursion terminates). The preferred labelings can be identified with solid borders. More details and refinements of this algorithm are given in [129]. The refinements include several optimizations. First the chosen argument in each branch for applying the transitions is selected not arbitrarily but depending on the already computed labelings. Secondly the search space is pruned via cutting off paths where further execution will not yield a preferred labeling. For instance if a $must\text{-}out$ argument is

[1] `http://www.arg.dundee.ac.uk/?page_id=279`

[2] This example is slightly modified from Figure 3.1 from [129].

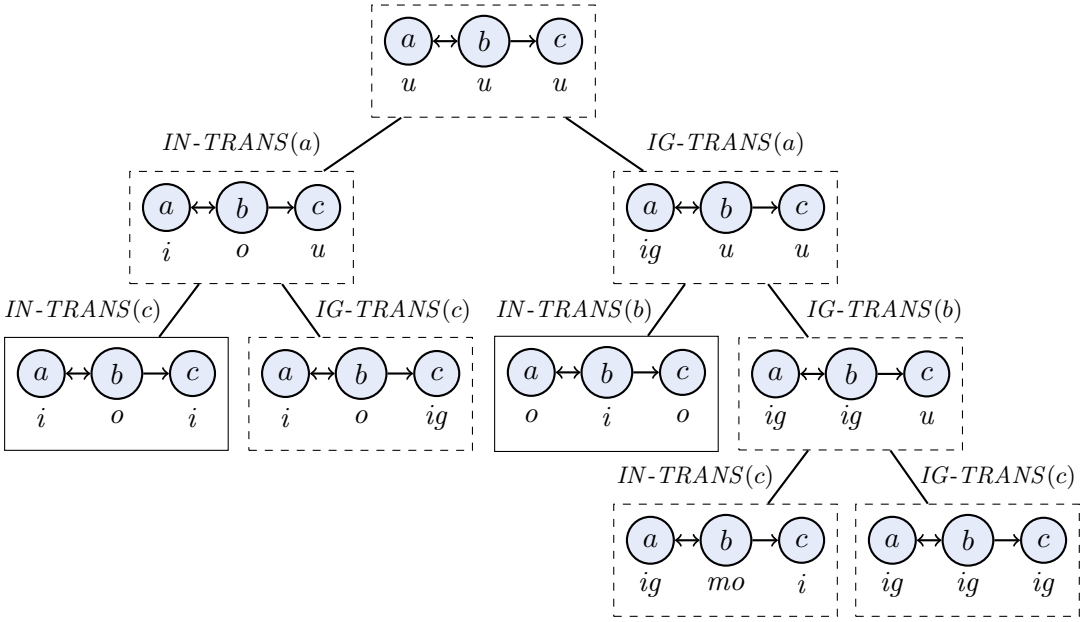

Figure 6.1: Recursive calls of Algorithm 9 from [129]

not attacked by an argument which is currently undecided we cannot accept an attacker of this argument anymore and thus a required defender is missing. Further pruning mechanisms are applied, which make sure that some unnecessary overhead is reduced.

In [129] also labeling based algorithms for credulous and skeptical acceptance w.r.t. preferred semantics are studied and the algorithm ideas are extended to further semantics e.g. semi-stable and ideal semantics.

The algorithms of [129] are also compared to existing strategies for labeling based algorithm such as [57, 126] theoretically and empirically using implementations. The empirical evaluation of the work of [129] showed good performance w.r.t. implementations of earlier labeling based algorithms [57, 126], ASPARTIX and dynPARTIX [46, 78].

Other labeling based algorithms also found their way into implementations. For instance CompArg [152] and PyAAL [135] are systems to compute labelings of various semantics for AFs.

Decomposition and Splitting Approaches When viewing reduction approaches and direct approaches for abstract argumentation, then our algorithms and the iterative algorithms from [45] can be seen as a sort of hybrid between these two. There is another hybrid approach, which emerged from a direct algorithm based on dynamic programming [46, 78], which first decomposes the given AF. The given AF is viewed as an undirected graph and one generates a so-called tree decomposition out of this graph. A tree-decomposition can be seen as a syntactic translation of the original graph, such that the result is a tree, where each node contains a so-called bag, which is a subset of the original arguments of the AF. A tree is a connected graph without cycles.

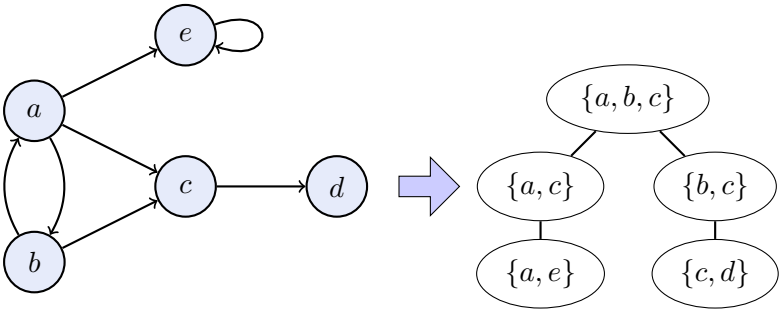

Figure 6.2: Tree decomposition of AF from Example 2.5 with width 2

This tree decomposition in particular has three features which are exploited in the follow-up algorithms. We first sketch these feature in a more intuitive way. First we have a tree structure which we can traverse bottom-up (or top-down). Secondly the bags contain only subsets of the original AF (a subset of the arguments and thus a restricted AF). The tree decomposition now allows to compute a kind of "partial" result only on the current bag which we are visiting in e.g. a bottom-up manner. The bag of a node may be considerably smaller then the original AF. Thirdly tree decompositions satisfy a property called "connectedness" condition. This says that for any two bags, if there is an argument x inside both of them, then x is also present in *all* bags in a path from the first to the second bag. This ensures in particular that when during the computation one finds a bag *not* containing a certain argument anymore, which was present before, then this argument will never appear again further up in the tree, i.e. we can by clever design of the overall algorithm "finish" the subtasks regarding this argument.

More formally given a graph $G = (V, E)$ a tree decomposition is a pair (T, B), where $T = (N, F)$ is a rooted tree, such that $B : N \to 2^V$ is a function which assigns to each node of the decomposition a subset of V (also called a the node's bag), s.t. the following three conditions are satisfied. The third condition is the connectedness condition.

1. For every vertex $v \in V$, there is a node $t \in N$, s.t. $v \in B(t)$;

2. for every edge $(x, y) \in E$, there is a node $t \in N$, s.t. $\{x, y\} \subseteq B(t)$;

3. for every vertex $v \in V$, the set $\{t \in N \mid v \in B(t)\}$ induces a connected subtree of T.

Given a tree $T = (N, F)$ and a subset of the vertices $X \subseteq N$, then the induced subtree of T w.r.t. X is $T' = (X, \{(x, y) \in F \mid \{x, y\} \subseteq X\}$. See Figure 6.2 for an example. Now there are many tree decompositions for a given graph. In particular the metric width of a tree decomposition can vary. The width of such a tree decomposition is the size of the largest bag in this tree decomposition minus 1. This width also measures in way how "close" a graph is to being a tree. For instance for trees one can always construct a tree decomposition of width 1 (every bag has two nodes). On the other hand the trivial tree decomposition containing exactly one node with a bag of all original arguments is a valid tree decomposition. The problem of

finding an optimal tree decomposition w.r.t. width is intractable [3], but heuristics and software implementations exists [128].

The approach from dynPARTIX [46, 78] now first computes a tree decomposition with a potentially low width. Then it traverses this tree decomposition in a bottom-up manner for the computation of preferred semantics. The concrete dynamic programming algorithm for enumeration of preferred extensions is somewhat technically involved. In essence so-called colorings of the arguments are computed in each bag, or sets of such colorings. Colorings are a data structure to hold information of the status of arguments computed so far. Then these colorings are propagated upwards and the new arguments from the next bag are considered. We refer the reader to [80] for details of the algorithm. A particularly important aspect of dynPARTIX is that the runtime depends on the treewidth of the decomposition.

In later works, the computation in each bag is delegated to an ASP solver in the framework of D-FLAT [27, 28]. First the given AF is decomposed to a tree decomposition. Then the bottom-up algorithm is applied using for each computation an ASP call. Comparing the dynamic programming approach using D-FLAT with our algorithms (e.g. Algorithm 1), we have in our search algorithms a sequence of calls to a SAT-solver for semantical (sub)problems. On the other hand using dynPARTIX and D-FLAT one syntactically decomposes the given AF and then applies a dynamic programming algorithm using ASP calls for restricted AFs. Furthermore, when we compare the inherent parameters used in the two approaches, then dynPARTIX and D-FLAT utilize treewidth as the parameter, while we utilize semantical parameters. First experiments for dynPARTIX showed that if the structure of the given AF emits a tree decomposition of low width, then the computation can be efficiently done.

Another related approach to the decomposition approach via tree decomposition is via splitting [14, 15, 16]. The basic idea is to split an AF into two parts. We then have a left and right part. For all attackers from the right to the left side we attach to the restricted AF from the left side auxiliary arguments, which attack those attacked arguments and back (symmetric attacks). Then the extensions are computed on this adapted framework from the left part. The auxiliary arguments act as "gadgets" to simulate that we either may put them inside an extension or not. For each extension of the left part we adapt the right restricted framework by basically propagating the extension. For instance if an argument a is inside an extension E of the left part and in the original (whole) framework there is an attack (a, b) to an argument b of the right part, then we insert auxiliary arguments and attacks on the right framework for ensuring that b is not in the extension. The concrete gadgets depend on the chosen semantics. We can then combine the extensions of the right part with the extension of the left part. In this way one can enumerate all extensions.

This idea was implemented and empirically evaluated in [16]. The results show that if the AF in question has certain characteristics, e.g. having a suitable SCC structure for splitting, then this approach increased the overall performance.

Other Related Systems and Algorithm Concepts An interesting approach for optimizing queries for preferred semantics is studied in [114]. Here the idea is to first compute the grounded extension of an AF and then propagating the result. This may dramatically influence the running time. The preprocessing step can be computed in polynomial time and may cover the status of

many arguments. Further it was observed that if we want to check for credulous and skeptical acceptance of arguments w.r.t. the preferred semantics we can restrict the given AF only to relevant arguments for answering the query [113]. In particular one can look at the condensation of the given AF (its SCC graph, see Section 2.3.1). The condensation is a forest of rooted trees. Now it is sufficient for deciding the query to consider only those SCCs which have a path to the one containing the argument under scrutiny. This in particular is based on the directionality property of preferred semantics.

Regarding related work on algorithms for ADFs, there are two systems for ADFs. The first is adfsys [91] and the other its successor DIAMOND [90]. Both are based on (disjunctive) ASP encodings. For more complex semantics more than one ASP call is applied or the input ADF is first preprocessed. This preprocessor maps a given ADF with Boolean formulae as acceptance conditions into an ADF with functional representations. This representation essentially gives the truth table of the formula, thus may be exponential in space w.r.t. the size of the ADF with formulae.

Another interesting approach for ADFs are translations of ADFs to AFs by Brewka, Dunne and Woltran [29] and Ellmauthaler [89]. Here translations for a fixed semantics of ADFs are given to a fixed semantics of AFs. They presented translations for computing ADF models via stable extensions of AFs. They also show that the stable semantics of ADFs, as defined in [34] can be translated to AFs. Such translations can also be used for analysing the complexity and for providing algorithms.

Leaving the area of abstract argumentation there are two related works which we highlight due to their similarities to our algorithms. The first one is an approach for the field of abduction [134]. Briefly put in propositional abduction we are working with propositional formulae and are given a knowledge base which contains our knowledge of the world. Then we have certain observations or manifestations which we would like to have explained. Usually we are given a set of hypotheses, from which we may draw these explanations. The task is then to find certain explanations of the manifestations which are consistent with our background knowledge. Typically one additionally applies certain constraints for these explanations, e.g. they should be minimal w.r.t. the number of hypotheses used. Propositional abduction features problems complete for the second level of the polynomial hierarchy.

Pfandler, Rümmele and Szeider [134] proposed to use a so-called backdoor approach for abduction, which is based on parameterized complexity theory. Their approach encodes the problem in a Boolean formula. In general this formula may be of exponential size w.r.t. the input instance. However they show that this formula can be constructed in quadratic time with a constant factor that is exponential in a certain "distance" measure. This distance measure describes the distance of the given propositional theory in the abduction problem to the classes of Horn or Krom formulae. A clause is Horn if it has at most one positive literal and a clause is Krom if it has at most two literals. The satisfiability problem for formulae from these classes can be decided in polynomial time. The distance is now measured in terms of the size of the smallest so-called strong backdoor set of the given theory to one of these classes. We define an auxiliary concept of simplifying a formula ϕ in CNF with a (partial) truth assignment γ, denoted as $\phi[\gamma]$ as follows. A clause $c \in \phi$ is in $\phi[\gamma]$ if $atoms(c) \cap dom(\gamma) = \emptyset$. That is if the partial truth assignment does not set a variable of c to a value, then this clause c is present in $\phi[\gamma]$ unchanged.

If $l \in c$ and $\gamma(l) = \mathbf{t}$, then this clause is not in $\phi[\gamma]$. If $l \in c$ and $\gamma(\neg l) = \mathbf{t}$, then $c' = c \setminus \{l\}$ is in $\phi[\gamma]$. No other clauses are present in $\phi[\gamma]$. Briefly put, we remove all clauses already satisfied by γ from ϕ. Additionally we remove all literals, which are evaluated to false from the clauses. Let C be a class of formulae in CNF. A strong C-backdoor set of a formula ϕ in CNF is a set of variables B, s.t. $\phi[\gamma] \in C$ for all possible truth assignments γ on the variables B. For instance a strong Krom-backdoor for the formula $\phi = (a \vee b \vee c)$ would be $B = \{a\}$. If a is set to true in a partial interpretation γ, then $\phi[\gamma] = \top$. Otherwise if $\gamma(a) = \mathbf{f}$, then $\phi[\gamma] = (b \vee c)$.

If we compare the approach of [134] to our algorithms, then the main difference is that in the backdoor approach to abduction inherent parameters of the given instance determine the worst case *size* of the resulting formula and in our algorithms parameters govern the worst case number of calls to a SAT-solver, but all of the formulae in these calls can be constructed in polynomial time.

Finally a related approach to our algorithms is the CEGAR (*counter-example guided abstraction refinement*) approach [49, 50]. In our algorithms, we use the NP decision procedures as NP oracles in an iterative fashion. Such approaches fall under the general CEGAR approach originating from the field of model checking. The CEGAR approach has been harnessed for solving various other intrinsically hard reasoning problems [13, 55, 94, 105, 106, 107, 127, 141, 155]. However, we are not aware of earlier work on developing CEGAR-based procedures, which are complexity-sensitive. In our algorithms if the AF in question is in a parameterized class of AFs which have milder complexity, then this implies a direct relation between the size of the AF, the parameter and the number of oracle calls.

6.3 Future Work

Optimizations and Extensions of Algorithms First of all our general scheme of search algorithms can be further extended to work with other semantics. In particular the novel SCC-based stage2 [72] semantics appears intriguing. Since stage2 is based on the stage semantics an adaption of our search algorithm for stage semantics in an recursive SCC scheme seems to be an interesting approach for the computationally complex stage2 semantics. Regarding SCCs in general, as observed by [113] we can restrict ourselves to only relevant arguments for semantics satisfying SCC-recursiveness [12] or directionality if we want to answer credulous or skeptical acceptance. It would be interesting to incorporate this optimization in our algorithms for preferred semantics.

Further optimizations for AF algorithms include the realizability studies by Dunne et al. [65, 66, 116]. They show which sets of extensions for a semantics σ are realizable, i.e. that one can construct an AF which has exactly the given set of sets as its σ-extensions. These kind of results may be used to reduce the search space during running time of our algorithms. For instance if we currently have a certain set of preferred extensions already computed, then using realizability results we may be able to conclude that certain kinds of extensions are not possible anymore and can include corresponding constraints for pruning the search space. Furthermore the problems observed with complete base semantics for the Algorithm 6 may be explained more thoroughly using results from realizability.

Regarding algorithms for ADFs, we currently used only the admissible base semantics for preferred semantics. Utilizing complete semantics for ADFs in the search algorithm may again lead to more efficient computation as we witnessed for AFs. Additionally we showed only how to generalize the search algorithms to preferred semantics. It would be interesting to see if other semantics generalized from AFs, e.g. semi-stable semantics, can be captured within our general search algorithm scheme.

Computational Complexity The complexity analysis of ADFs is still not finished. Recently an extension-based semantics for ADFs [136] was proposed. A detailed comparison of the novel extension-based semantics with the ADF semantics as defined in [31, 144] w.r.t. complexity is currently missing. Furthermore parametrized complexity, investigation of tractable fragments and backdoors [61, 79, 80, 81] are completely untouched topics for ADFs.

Implementations and Experimental Evaluation For concrete implementations we provided programs using SAT-solvers. It would be interesting to see the difference if one uses other oracles such as CSP or ASP solvers instead of SAT-solvers. For ADFs our algorithms are currently not implemented and we may use QBF or disjunctive ASP solvers for general ADFs. For bipolar ADFs we can use SAT-solvers or ASP solvers dealing with normal logic programs. For iterative schemes it might also be interesting to try nested HEX programs [88], which offer nesting of ASP programs.

Currently there is a lack of benchmark suites for performance tests in argumentation [73]. In particular AFs from applications are missing. Clearly this is highly desirable for future experimentation.

General Future Work Abstract argumentation is embedded in a larger workflow. In the introduction we sketched this argumentation process (see Figure 1.1). Briefly put, we instantiate a framework out of a given knowledge base, compute sets of jointly acceptable arguments and draw conclusions from them. For creating arguments several formal approaches have been studied [103, 137]. For instance one can create arguments out of a knowledge bases consisting of Boolean formulae or logical rules. In the argumentation process computing the status of abstract arguments is one intermediate step. Using knowledge from other steps of the workflow may dramatically increase the overall performance. In particular some framework structures might not be possible using certain instantiation schemes. Further it might be possible to augment e.g. an AF with further information from the original knowledge base and include this information for computing acceptability of arguments. Currently the steps are mostly viewed as independent challenges and the process of combining knowledge from different steps is just in its infancy, but might provide tremendous performance gains in solving problems for real world applications.

Bibliography

[1] Leila Amgoud and Henri Prade. Using Arguments for Making and Explaining Decisions. *Artificial Intelligence*, 173(3-4):413–436, 2009.

[2] Ofer Arieli and Martin W. A. Caminada. A QBF-Based Formalization of Abstract Argumentation Semantics. *Journal of Applied Logic*, 11(2):229–252, 2013.

[3] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of Finding Embeddings in a k-tree. *SIAM Journal of Algebraic Discrete Methods*, 8:277–284, 1987.

[4] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality Networks: A Theoretical and Empirical Study. *Constraints*, 16(2):195–221, 2011.

[5] Gilles Audemard and Laurent Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. In Craig Boutilier, editor, *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, pages 399–404, 2009.

[6] Chitta Baral and Michael Gelfond. Logic Programming and Knowledge Representation. *The Journal of Logic Programming*, 19 & 20:73–148, 1994.

[7] Pietro Baroni, Martin W. A. Caminada, and Massimiliano Giacomin. An Introduction to Argumentation Semantics. *The Knowledge Engineering Review*, 26(4):365–410, 2011.

[8] Pietro Baroni and Massimiliano Giacomin. Evaluating Argumentation Semantics with Respect to Skepticism Adequacy. In Lluis Godo, editor, *Proceedings of the Eight European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2005*, volume 3571 of *Lecture Notes in Computer Science*, pages 329–340. Springer, 2005.

[9] Pietro Baroni and Massimiliano Giacomin. On Principle-Based Evaluation of Extension-Based Argumentation Semantics. *Artificial Intelligence*, 171(10-15):675–700, 2007.

[10] Pietro Baroni and Massimiliano Giacomin. A Systematic Classification of Argumentation Frameworks where Semantics Agree. In Philippe Besnard, Sylvie Doutre, and Anthony Hunter, editors, *Proceedings of the Second Conference on Computational Models of Argument, COMMA 2008*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, pages 37–48. IOS Press, 2008.

[11] Pietro Baroni and Massimiliano Giacomin. Semantics of Abstract Argument Systems. In Guillermo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 25–44. Springer, 2009.

[12] Pietro Baroni, Massimiliano Giacomin, and Giovanni Guida. SCC-Recursiveness: A General Schema for Argumentation Semantics. *Artificial Intelligence*, 168(1-2):162–210, 2005.

[13] Clark W. Barrett, David L. Dill, and Aaron Stump. Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification, CAV 2002*, volume 2404 of *Lecture Notes in Computer Science*, pages 236–249. Springer, 2002.

[14] Ringo Baumann. Splitting an Argumentation Framework. In James P. Delgrande and Wolfgang Faber, editors, *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2011*, volume 6645 of *Lecture Notes in Computer Science*, pages 40–53. Springer, 2011.

[15] Ringo Baumann, Gerhard Brewka, Wolfgang Dvořák, and Stefan Woltran. Parameterized Splitting: A Simple Modification-Based Approach. In Esra Erdem, Joohyung Lee, Yuliya Lierler, and David Pearce, editors, *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*, pages 57–71. Springer, 2012.

[16] Ringo Baumann, Gerhard Brewka, and Renata Wong. Splitting Argumentation Frameworks: An Empirical Evaluation. In Sanjay Modgil, Nir Oren, and Francesca Toni, editors, *Revised Selected Papers of the First International Workshop on Theories and Applications of Formal Argumentation, TAFA 2011*, volume 7132 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2012.

[17] Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in AI and Law: Editors' Introduction. *Artificial Intelligence Law*, 13(1):1–8, 2005.

[18] Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in Artificial Intelligence. *Artificial Intelligence*, 171(10-15):619–641, 2007.

[19] Trevor J. M. Bench-Capon, Henry Prakken, and Giovanni Sartor. Argumentation in Legal Reasoning. In Guillermo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 363–382. Springer US, 2009.

[20] Marco Benedetti and Hratch Mangassarian. QBF-Based Formal Verification: Experience and Perspectives. *Journal on Satisfiability, Boolean Modeling and Computation*, 5(1-4):133–191, 2008.

[21] Philippe Besnard and Sylvie Doutre. Checking the Acceptability of a Set of Arguments. In James P. Delgrande and Torsten Schaub, editors, *Proceedings of the Tenth International Workshop on Non-Monotonic Reasoning, NMR 2004*, pages 59–64, 2004.

[22] Philippe Besnard and Anthony Hunter. *Elements of Argumentation*. MIT Press, 2008.

[23] Philippe Besnard and Anthony Hunter. Argumentation Based on Classical Logic. In Guillermo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 133–152. Springer US, 2009.

[24] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[25] Stefano Bistarelli, Fabio Rossi, and Francesco Santini. A First Comparison of Abstract Argumentation Systems: A Computational Perspective. In Domenico Cantone and Marianna Nicolosi Asmundo, editors, *Proceedings of the 28th Italian Conference on Computational Logic, CILC 2013*, volume 1068 of *CEUR Workshop Proceedings*, pages 241–245. CEUR-WS.org, 2013.

[26] Stefano Bistarelli and Francesco Santini. ConArg: A Constraint-Based Computational Framework for Argumentation Systems. In *Proceedings of the 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011*, pages 605–612. IEEE, 2011.

[27] Bernhard Bliem. Decompose, Guess & Check: Declarative Problem Solving on Tree Decompositions. Master's thesis, Vienna University of Technology, 2012.

[28] Bernhard Bliem, Michael Morak, and Stefan Woltran. D-FLAT: Declarative Problem Solving Using Tree Decompositions and Answer-Set Programming. *Theory and Practice of Logic Programming*, 12(4-5):445–464, 2012.

[29] Gerhard Brewka, Paul E. Dunne, and Stefan Woltran. Relating the Semantics of Abstract Dialectical Frameworks and Standard AFs. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011*, pages 780–785. IJCAI/AAAI, 2011.

[30] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczyński. Answer Set Programming at a Glance. *Communications of the ACM*, 54(12):92–103, 2011.

[31] Gerhard Brewka, Stefan Ellmauthaler, Hannes Strass, Johannes P. Wallner, and Stefan Woltran. Abstract Dialectical Frameworks Revisited. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, pages 803–809. AAAI Press / IJCAI, 2013.

[32] Gerhard Brewka and Thomas F. Gordon. Carneades and Abstract Dialectical Frameworks: A Reconstruction. In Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Guillermo R. Simari, editors, *Proceedings of the Third International Conference on Computational Models of Argument, COMMA 2010*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 3–12. IOS Press, 2010.

[33] Gerhard Brewka, Sylwia Polberg, and Stefan Woltran. Generalizations of Dung Frameworks and Their Role in Formal Argumentation. *Intelligent Systems, IEEE*, 2014. To appear.

[34] Gerhard Brewka and Stefan Woltran. Abstract Dialectical Frameworks. In Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczyński, editors, *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning, KR 2010*, pages 102–111. AAAI Press, 2010.

[35] Uwe Bubeck. *Model-based Transformations for Quantified Boolean Formulas*. PhD thesis, Faculty of Electrical Engineering, Computer Science and Mathematics. University of Paderborn, 2010.

[36] Hans Kleine Büning and Uwe Bubeck. Theory of Quantified Boolean Formulas. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 735–760. IOS Press, 2009.

[37] Martin W. A. Caminada. Semi-Stable Semantics. In Paul E. Dunne and Trevor J. M. Bench-Capon, editors, *Proceedings of the First Conference on Computational Models of Argument, COMMA 2006*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 121–130. IOS Press, 2006.

[38] Martin W. A. Caminada. An Algorithm for Computing Semi-Stable Semantics. In Khaled Mellouli, editor, *Proceedings of the Ninth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2007*, volume 4724 of *Lecture Notes in Computer Science*, pages 222–234. Springer, 2007.

[39] Martin W. A. Caminada. Comparing Two Unique Extension Semantics for Formal Argumentation: Ideal and Eager. In *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence, BNAIC 2007*, pages 81–87, 2007.

[40] Martin W. A. Caminada and Leila Amgoud. On the Evaluation of Argumentation Formalisms. *Artificial Intelligence*, 171(5-6):286–310, 2007.

[41] Martin W. A. Caminada, Walter A. Carnielli, and Paul E. Dunne. Semi-Stable Semantics. *Journal of Logic and Computation*, 22(5):1207–1254, 2012.

[42] Martin W. A. Caminada and Dov M. Gabbay. A Logical Account of Formal Argumentation. *Studia Logica*, 93(2):109–145, 2009.

[43] Dan Cartwright and Katie Atkinson. Political Engagement Through Tools for Argumentation. In Philippe Besnard, Sylvie Doutre, and Anthony Hunter, editors, *Proceedings of the Second Conference on Computational Models of Argument, COMMA 2008*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, pages 116–127. IOS Press, 2008.

[44] Dan Cartwright and Katie Atkinson. Using Computational Argumentation to Support E-participation. *IEEE Intelligent Systems*, 24(5):42–52, 2009.

[45] Federico Cerutti, Paul E. Dunne, Massimiliano Giacomin, and Mauro Vallati. Computing Preferred Extensions in Abstract Argumentation: A SAT-Based Approach. In Elizabeth Black, Sanjay Modgil, and Nir Oren, editors, *Proceedings of the Second International Workshop on Theory and Applications of Formal Argumentation, Revised Selected papers, TAFA 2013*, volume 8306 of *Lecture Notes in Computer Science*, pages 176–193. Springer, 2014.

[46] Günther Charwat and Wolfgang Dvořák. dynPARTIX 2.0 - Dynamic Programming Argumentation Reasoning Tool. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the Fourth International Conference on Computational Models of Argument, COMMA 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 507–508. IOS Press, 2012.

[47] Günther Charwat, Wolfgang Dvořák, Sarah A. Gaggl, Johannes P. Wallner, and Stefan Woltran. Implementing Abstract Argumentation - A Survey. Technical Report DBAI-TR-2013-82, Technische Universität Wien, 2013.

[48] Alonzo Church. *Introduction to Mathematical Logic*. Princeton Landmarks in Mathematics and Physics. Princeton University Press, 1996.

[49] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-Guided Abstraction Refinement for Symbolic Model Checking. *Journal of the ACM*, 50(5):752–794, 2003.

[50] Edmund M. Clarke, Anubhav Gupta, and Ofer Strichman. SAT-Based Counterexample-Guided Abstraction Refinement. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 23(7):1113–1123, 2004.

[51] Michael Codish and Moshe Zazon-Ivry. Pairwise Cardinality Networks. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *International Conference on Logic for Programming Artificial Intelligence and Reasoning, LPAR 2010*, volume 6355 of *Lecture notes of Computer Science*, pages 154–172. Springer, 2010.

[52] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *Conference Record of Third Annual ACM Symposium on Theory of Computing (STOC-71)*, pages 151–158, 1971.

[53] Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Symmetric Argumentation Frameworks. In Lluis Godo, editor, *Proceedings of the Eighth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2005*, volume 3571 of *Lecture Notes in Computer Science*, pages 317–328. Springer, 2005.

[54] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, second edition, 2002.

[55] Leonardo de Moura, Harald Ruess, and Maria Sorea. Lazy Theorem Proving for Bounded Model Checking over Infinite Domains. In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction, CADE-18*, volume 2392 of *Lecture Notes in Computer Science*, pages 438–455. Springer, 2002.

[56] Yannis Dimopoulos and Alberto Torres. Graph Theoretical Structures in Logic Programs and Default Theories. *Theoretical Computer Science*, 170(1-2):209–244, 1996.

[57] Sylvie Doutre and Jérôme Mengin. Preferred Extensions of Argumentation Frameworks: Query Answering and Computation. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning, IJCAR 2001*, volume 2083 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2001.

[58] Christian Drescher, Martin Gebser, Torsten Grote, Benjamin Kaufmann, Arne König, Max Ostrowski, and Torsten Schaub. Conflict-Driven Disjunctive Answer Set Solving. In Gerhard Brewka and Jérôme Lang, editors, *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning, KR 2008*, pages 422–432. AAAI Press, 2008.

[59] Phan Minh Dung. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence*, 77(2):321–358, 1995.

[60] Phan Minh Dung, Paolo Mancarella, and Francesca Toni. Computing Ideal Sceptical Argumentation. *Artificial Intelligence*, 171(10-15):642–674, 2007.

[61] Paul E. Dunne. Computational Properties of Argument Systems Satisfying Graph-theoretic Constraints. *Artificial Intelligence*, 171(10-15):701–729, 2007.

[62] Paul E. Dunne. The Computational Complexity of Ideal Semantics. *Artificial Intelligence*, 173(18):1559–1591, 2009.

[63] Paul E. Dunne and Trevor J. M. Bench-Capon. Coherence in Finite Argument Systems. *Artificial Intelligence*, 141(1/2):187–203, 2002.

[64] Paul E. Dunne and Martin W. A. Caminada. Computational Complexity of Semi-Stable Semantics in Abstract Argumentation Frameworks. In Steffen Hölldobler, Carsten Lutz, and Heinrich Wansing, editors, *Proceedings of the Eleventh European Conference on Logics in Artificial Intelligence, JELIA 2008*, volume 5293 of *Lecture Notes in Computer Science*, pages 153–165. Springer, 2008.

[65] Paul E. Dunne, Wolfgang Dvořák, Thomas Linsbichler, and Stefan Woltran. Characteristics of Multiple Viewpoints in Abstract Argumentation. In Christoph Beierle and Gabriele Kern-Isberner, editors, *Proceedings of the Fourth Workshop on Dynamics of Knowledge and Belief, DKB 2013*, pages 16–30, 2013.

[66] Paul E. Dunne, Wolfgang Dvořák, Thomas Linsbichler, and Stefan Woltran. Characteristics of Multiple Viewpoints in Abstract Argumentation. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning, KR 2014*, 2014. To appear.

[67] Paul E. Dunne, Wolfgang Dvořák, and Stefan Woltran. Parametric Properties of Ideal Semantics. *Artificial Intelligence*, 202:1–28, 2013.

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

[69] Wolfgang Dvořák. *Computational Aspects of Abstract Argumentation*. PhD thesis, Vienna University of Technology, 2012.

[70] Wolfgang Dvořák. On the Complexity of Computing the Justification Status of an Argument. In Sanjay Modgil, Nir Oren, and Francesca Toni, editors, *Proceedings of the First International Workshop on Theory and Applications of Formal Argumentation, TAFA 2011, Revised Selected Papers*, volume 7132 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2012.

[71] Wolfgang Dvořák and Sarah A. Gaggl. Computational Aspects of cf2 and stage2 Argumentation Semantics. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the Fourth International Conference on Computational Models of Argument, COMMA 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 273–284. IOS Press, 2012.

[72] Wolfgang Dvořák and Sarah A. Gaggl. Stage Semantics and the SCC-Recursive Schema for Argumentation Semantics. *Journal of Logic and Computation*, 2014. To appear.

[73] Wolfgang Dvořák, Sarah A. Gaggl, Stefan Szeider, and Stefan Woltran. Benchmark libraries for argumentation. In Sascha Ossowski, editor, *Agreement Technologies*, volume 8 of *Law, Governance and Technology Series*, chapter The Added Value of Argumentation, pages 389–393. Springer, 2013.

[74] Wolfgang Dvořák, Sarah A. Gaggl, Johannes P. Wallner, and Stefan Woltran. Making Use of Advances in Answer-Set Programming for Abstract Argumentation Systems. In Hans Tompits, Salvador Abreu, Johannes Oetsch, Jörg Pührer, Dietmar Seipel, Masanobu Umeda, and Armin Wolf, editors, *Proceedings of the 19th International Conference on Applications of Declarative Programming and Knowledge Management, INAP 2011, Revised Selected Papers*, volume 7773 of *Lecture Notes in Artificial Intelligence*, pages 114–133. Springer, 2013.

[75] Wolfgang Dvořák, Matti Järvisalo, Johannes P. Wallner, and Stefan Woltran. CEGARTIX: A SAT-Based Argumentation System. Presented at the Pragmatics of SAT Workshop (PoS 2012) `http://www.dbai.tuwien.ac.at/research/project/argumentation/papers/DvorakJWW2012PoS.pdf`, 2012.

[76] Wolfgang Dvořák, Matti Järvisalo, Johannes P. Wallner, and Stefan Woltran. Complexity-Sensitive Decision Procedures for Abstract Argumentation. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning, KR 2012*, pages 54–64. AAAI Press, 2012.

[77] Wolfgang Dvořák, Matti Järvisalo, Johannes P. Wallner, and Stefan Woltran. Complexity-Sensitive Decision Procedures for Abstract Argumentation. *Artificial Intelligence*, 206:53–78, 2014.

[78] Wolfgang Dvořák, Michael Morak, Clemens Nopp, and Stefan Woltran. dynPARTIX - A Dynamic Programming Reasoner for Abstract Argumentation. In Hans Tompits, Salvador Abreu, Johannes Oetsch, Jörg Pührer, Dietmar Seipel, Masanobu Umeda, and Armin Wolf, editors, *Proceedings of the 19th International Conference on Applications of Declarative Programming and Knowledge Management, INAP 2011, Revised Selected Papers*, volume 7773 of *Lecture Notes in Artificial Intelligence*, pages 259–268. Springer, 2013.

[79] Wolfgang Dvořák, Sebastian Ordyniak, and Stefan Szeider. Augmenting Tractable Fragments of Abstract Argumentation. *Artificial Intelligence*, 186:157–173, 2012.

[80] Wolfgang Dvořák, Reinhard Pichler, and Stefan Woltran. Towards Fixed-parameter Tractable Algorithms for Abstract Argumentation. *Artificial Intelligence*, 186:1–37, 2012.

[81] Wolfgang Dvořák, Stefan Szeider, and Stefan Woltran. Reasoning in Argumentation Frameworks of Bounded Clique-Width. In Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Guillermo R. Simari, editors, *Proceedings of the Third International Conference on Computational Models of Argument, COMMA 2010*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 219–230. IOS Press, 2010.

[82] Wolfgang Dvořák and Stefan Woltran. Complexity of Semi-stable and Stage Semantics in Argumentation Frameworks. *Information Processing Letters*, 110(11):425–430, 2010.

[83] Niklas Eén and Armin Biere. Effective Preprocessing in SAT Through Variable and Clause Elimination. In Fahiem Bacchus and Toby Walsh, editors, *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing, SAT 2005*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

[84] Niklas Eén and Niklas Sörensson. An Extensible SAT-Solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing, SAT 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[85] Uwe Egly, Sarah A. Gaggl, and Stefan Woltran. Answer-Set Programming Encodings for Argumentation Frameworks. *Argument and Computation*, 1(2):147–177, 2010.

[86] Uwe Egly and Stefan Woltran. Reasoning in Argumentation Frameworks Using Quantified Boolean Formulas. In Paul E. Dunne and Trevor J. M. Bench-Capon, editors, *Proceedings of the First Conference on Computational Models of Argument, COMMA 2006*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 133–144. IOS Press, 2006.

[87] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer Set Programming: A Primer. In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutierrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors, *Fifth International Reasoning Web Summer School, RW 2009*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer, 2009.

[88] Thomas Eiter, Thomas Krennwallner, and Christoph Redl. HEX-Programs with Nested Program Calls. In Hans Tompits, Salvador Abreu, Johannes Oetsch, Jörg Pührer, Dietmar Seipel, Masanobu Umeda, and Armin Wolf, editors, *Proceedings of the 19th International Conference on Applications of Declarative Programming and Knowledge Management, INAP 2011, Revised Selected Papers*, volume 7773 of *Lecture Notes in Computer Science*, pages 269–278. Springer, 2013.

[89] Stefan Ellmauthaler. Abstract Dialectical Frameworks: Properties, Complexity, and Implementation. Master's thesis, Technische Universität Wien, Institut für Informationssysteme, 2012.

[90] Stefan Ellmauthaler and Hannes Strass. The DIAMOND System for Argumentation: Preliminary Report. In Michael Fink and Yuliya Lierler, editors, *Proceedings of the Sixth International Workshop on Answer Set Programming and Other Computing Paradigms, ASPOCP 2013*, pages 97–107, 2013.

[91] Stefan Ellmauthaler and Johannes P. Wallner. Evaluating Abstract Dialectical Frameworks with ASP. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the Fourth International Conference on Computational Models of Argument, COMMA 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 505–506. IOS Press, 2012.

[92] Wolfgang Faber and Stefan Woltran. Manifold Answer-Set Programs and Their Applications. In Marcello Balduccini and Tran Cao Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, volume 6565 of *Lecture Notes in Artificial Intelligence*, pages 44–63. Springer, 2011.

[93] Alexander Felfernig, Monika Schubert, and Christoph Zehentner. An Efficient Diagnosis Algorithm for Inconsistent Constraint Sets. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 26(1):53–62, 2012.

[94] Cormac Flanagan, Rajeev Joshi, Xinming Ou, and James B. Saxe. Theorem Proving Using Lazy Proof Explication. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Pro-*

ceedings of the 15th International Conference on Computer Aided Verification, CAV 2003, volume 2725 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2003.

[95] John Fox, David Glasspool, Vivek Patkar, Mark Austin, Liz Black, Matthew South, Dave Robertson, and Charles Vincent. Delivering Clinical Decision Support Services: There is Nothing as Practical as a Good Theory. *Journal of Biomedical Informatics*, 43(5):831–843, 2010.

[96] Dov M. Gabbay. An Equational Approach to Argumentation Networks. *Argument & Computation*, 3(2-3):87–142, 2012.

[97] Sarah A. Gaggl. *A Comprehensive Analysis of the cf2 Argumentation Semantics: From Characterization to Implementation*. PhD thesis, Vienna University of Technology, 2013.

[98] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The Potsdam Answer Set Solving Collection. *AI Communications*, 24(2):105–124, 2011.

[99] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[100] Martin Gebser, Roland Kaminski, Arne König, and Torsten Schaub. Advances in *gringo* Series 3. In James P. Delgrande and Wolfgang Faber, editors, *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2011*, volume 6645 of *Lecture Notes in Computer Science*, pages 345–351. Springer, 2011.

[101] Martin Gebser, Roland Kaminski, and Torsten Schaub. Complex Optimization in Answer Set Programming. *Theory and Practice of Logic Programming*, 11(4-5):821–839, 2011.

[102] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. Conflict-Driven Answer Set Solving. In M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007*, pages 386–392. AAAI Press/The MIT Press, 2007.

[103] Nikos Gorogiannis and Anthony Hunter. Instantiating Abstract Argumentation with Classical Logic Arguments: Postulates and Properties. *Artificial Intelligence*, 175(9-10):1479–1497, 2011.

[104] Michael Huth and Mark Ryan. *Logic in Computer Science Modelling and Reasoning about Systems*. Cambridge University Press, 2nd edition edition, 2004.

[105] Mikolás Janota, Radu Grigore, and João Marques-Silva. Counterexample Guided Abstraction Refinement Algorithm for Propositional Circumscription. In Tomi Janhunen and Ilkka Niemelä, editors, *Proceedings of the 12th European Conference on Logics in Artificial Intelligence, JELIA 2010*, volume 6341 of *Lecture Notes in Computer Science*, pages 195–207. Springer, 2010.

[106] Mikolás Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. Solving QBF with Counterexample Guided Refinement. In Alessandro Cimatti and Roberto Sebastiani, editors, *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing, SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 114–128. Springer, 2012.

[107] Mikolás Janota and João Marques-Silva. Abstraction-Based Algorithm for 2QBF. In Karem A. Sakallah and Laurent Simon, editors, *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing, SAT 2011*, volume 6695 of *Lecture Notes in Computer Science*, pages 230–244. Springer, 2011.

[108] Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The International SAT Solver Competitions. *AI Magazine*, 33(1):89–94, 2012.

[109] David S. Johnson. A Catalog of Complexity Classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume A, Algorithms and Complexity*, chapter 9, pages 67–161. MIT Press, 1990.

[110] Philip Kilby, John K. Slaney, Sylvie Thiébaux, and Toby Walsh. Backbones and Backdoors in Satisfiability. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the 20th National Conference on Artificial Intelligence, AAAI 2005*, pages 1368–1373. AAAI Press / The MIT Press, 2005.

[111] James Ladyman. *Understanding Philosophy of Science*. Routledge, 2002.

[112] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[113] Beishui Liao and Huaxin Huang. Partial Semantics of Argumentation: Basic Properties and Empirical Results. *Journal of Logic and Computation*, 23(3):541–562, 2012.

[114] Beishui Liao, Liyun Lei, and Jianhua Dai. Computing Preferred Labellings by Exploiting SCCs and Most Sceptically Rejected Arguments. In Elizabeth Black, Sanjay Modgil, and Nir Oren, editors, *Proceedings of the Second International Workshop on Theory and Applications of Formal Argumentation, TAFA 2013, Revised Selected papers*, volume 8306 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2013.

[115] Mark H. Liffiton and Karem A. Sakallah. Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints. *Journal of Automated Reasoning*, 40(1):1–33, 2008.

[116] Thomas Linsbichler. On the Limits of Expressiveness in Abstract Argumentation Semantics: Realizability and Signatures. Master's thesis, Vienna University of Technology, 2013.

[117] Florian Lonsing. *Dependency Schemes and Search-Based QBF Solving: Theory and Practice*. PhD thesis, Johannes Kepler University, Linz, Austria, 2012.

[118] Sharad Malik and Georg Weissenbacher. Boolean Satisfiability Solvers: Techniques and Extensions. In *Software Safety and Security - Tools for Analysis and Verification*, NATO Science for Peace and Security Series. IOS Press, 2012.

[119] Yuri Malitsky, Barry O'Sullivan, Alessandro Previti, and João Marques-Silva. A Portfolio Approach to Enumerating Minimal Correction Subsets for Satisfiability Problems. In *Proceedings of the Eleventh International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming, CPAIOR 2014*, 2014. To appear.

[120] João Marques-Silva. The Impact of Branching Heuristics in Propositional Satisfiability Algorithms. In Pedro Barahona and José Júlio Alferes, editors, *Proceedings of the Ninth Portuguese Conference on Artificial Intelligence, EPIA 1999*, volume 1695 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1999.

[121] João Marques-Silva, Federico Heras, Mikolás Janota, Alessandro Previti, and Anton Belov. On Computing Minimal Correction Subsets. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, pages 615–622. IJCAI/AAAI, 2013.

[122] João Marques-Silva, Mikolás Janota, and Inês Lynce. On Computing Backbones of Propositional Theories. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence, ECAI 2010*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 15–20. IOS Press, 2010.

[123] João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-Driven Clause Learning SAT Solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 131–153. IOS Press, 2009.

[124] João Marques-Silva and Karem A. Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[125] Peter McBurney, Simon Parsons, and Iyad Rahwan, editors. *Argumentation in Multi-Agent Systems - Eighth International Workshop, ArgMAS 2011, Taipei, Taiwan, May 3, 2011, Revised Selected Papers*, volume 7543, 2012.

[126] Sanjay Modgil and Martin W. A. Caminada. Proof Theories and Algorithms for Abstract Argumentation Frameworks. In Iyad Rahwan and Guillermo R. Simari, editors, *Argumentation in Artificial Intelligence*, pages 105–132. 2009.

[127] David Monniaux. Quantifier Elimination by Lazy Model Enumeration. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Proceedings of the 22nd International Conference on Computer Aided Verification, CAV 2010*, volume 6174 of *Lecture Notes in Computer Science*, pages 585–599. Springer, 2010.

[128] Michael Morak. A Dynamic Programming-Based Answer Set Programming Solver. Master's thesis, Vienna University of Technology, 2011.

[129] Samer Nofal. *Algorithms for Argument Systems*. PhD thesis, University of Liverpool, 2013.

[130] Samer Nofal, Katie Atkinson, and Paul E. Dunne. Algorithms for Decision Problems in Argument Systems under Preferred Semantics. *Artifical Intelligence*, 207:23–51, 2014.

[131] Samer Nofal, Paul E. Dunne, and Katie Atkinson. On Preferred Extension Enumeration in Abstract Argumentation. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the Fourth International Conference on Computational Models of Argument, COMMA 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 205–216. IOS Press, 2012.

[132] Alexander Nöhrer, Armin Biere, and Alexander Egyed. Managing SAT Inconsistencies with HUMUS. In *Proceedings of the Workshop on Variability Modelling of Software-Intensive Systems*, pages 83–91. ACM, 2012.

[133] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[134] Andreas Pfandler, Stefan Rümmele, and Stefan Szeider. Backdoors to Abduction. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, pages 1046–1052. IJCAI/AAAI, 2013.

[135] Mikolaj Podlaszewski, Martin W. A. Caminada, and Gabriella Pigozzi. An Implementation of Basic Argumentation Components. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pages 1307–1308. IFAAMAS, 2011.

[136] Sylwia Polberg, Johannes P. Wallner, and Stefan Woltran. Admissibility in the Abstract Dialectical Framework. In João Leite, Tran Cao Son, Paolo Torroni, Leon van der Torre, and Stefan Woltran, editors, *Proceedings of the 14th International Workshop on Computational Logic in Multi-Agent Systems, CLIMA 2013*, volume 8143 of *Lecture Notes in Artificial Intelligence*, pages 102–118. Springer, 2013.

[137] Henry Prakken. An Abstract Framework for Argumentation with Structured Arguments. *Argument and Computation*, 1(2):93–124, 2010.

[138] Philip R. Quinlan, Alastair Thompson, and Chris Reed. An Analysis and Hypothesis Generation Platform for Heterogeneous Cancer Databases. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the Fourth International Conference on Computational Models of Argument, COMMA 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 59–70. IOS Press, 2012.

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

[140] Emanuele Di Rosa, Enrico Giunchiglia, and Marco Maratea. Solving Satisfiability Problems with Preferences. *Constraints*, 15(4):485–515, 2010.

[141] Roberto Sebastiani. Lazy Satisfibility Modulo Theories. *Journal of Satisfiability, Boolean Modeling and Computation*, 3(3-4):141–224, 2007.

[142] Matthew South, Gerard Vreeswijk, and John Fox. Dungine: A Java Dung Reasoner. In Philippe Besnard, Sylvie Doutre, and Anthony Hunter, editors, *Proceedings of the Second Conference on Computational Models of Argument, COMMA 2008*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, pages 360–368. IOS Press, 2008.

[143] Larry J. Stockmeyer and Albert R. Meyer. Word Problems Requiring Exponential Time. In *ACM Symposium on Theory of Computing (STOC-73)*, pages 1–9. ACM Press, 1973.

[144] Hannes Strass. Approximating Operators and Semantics for Abstract Dialectical Frameworks. *Artificial Intelligence*, 205:39–70, 2013.

[145] Hannes Strass. Instantiating Knowledge Bases in Abstract Dialectical Frameworks. In João Leite, Tran Cao Son, Paolo Torroni, Leon van der Torre, and Stefan Woltran, editors, *Proceedings of the Fourteenth International Workshop on Computational Logic in Multi-Agent Systems, CLIMA 2013*, volume 8143 of *Lecture Notes in Artificial Intelligence*, pages 86–101. Springer, 2013.

[146] Hannes Strass and Johannes P. Wallner. Analyzing the Computational Complexity of Abstract Dialectical Frameworks via Approximation Fixpoint Theory. Technical Report 2, Computer Science Institute, Leipzig University, 2013.

[147] Hannes Strass and Johannes P. Wallner. Analyzing the Computational Complexity of Abstract Dialectical Frameworks via Approximation Fixpoint Theory. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning, KR 2014*. To appear, 2014.

[148] Francesca Toni and Marek Sergot. Argumentation and Answer Set Programming. In Marcello Balduccini and Tran Cao Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, volume 6565 of *Lecture Notes in Computer Science*, pages 164–180. Springer, 2011.

[149] Grigori S. Tseitin. On the Complexity of Derivation in Propositional Calculus. In A. O. Silenko, editor, *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125. 1968.

[150] Eugenio Di Tullio and Floriana Grasso. A Model for a Motivational System Grounded on Value Based Abstract Argumentation Frameworks. In Patty Kostkova, Martin Szomszor, and David Fowler, editors, *Proceedings of the Fourth International Conference on Electronic Healthcare, eHealth 2011, Revised Selected Papers*, volume 91 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 43–50. Springer, 2012.

[151] Bart Verheij. Two Approaches to Dialectical Argumentation: Admissible Sets and Argumentation Stages. In John-Jules Ch. Meyer and Linda van der Gaag, editors, *Proceedings of the Eighth Dutch Conference on Artificial Intelligence (NAIC'96)*, pages 357–368, 1996.

[152] Bart Verheij. A Labeling Approach to the Computation of Credulous Acceptance in Argumentation. In Manuela M Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007*, pages 623–628, 2007.

[153] Johannes P. Wallner, Georg Weissenbacher, and Stefan Woltran. Advanced SAT Techniques for Abstract Argumentation. In João Leite, Tran Cao Son, Paolo Torroni, Leon van der Torre, and Stefan Woltran, editors, *Proceedings of the 14th International Workshop on Computational Logic in Multi-Agent Systems, CLIMA 2013*, volume 8143 of *Lecture Notes in Artificial Intelligence*, pages 138–154. Springer, 2013.

[154] Jesse Whittemore, Joonyoung Kim, and Karem A. Sakallah. SATIRE: A New Incremental Satisfiability Engine. In *Proceedings of the 38th Design Automation Conference, DAC 2001*, pages 542–545. ACM, 2001.

[155] Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo de Moura. Efficiently Solving Quantified Bit-Vector Formulas. In Roderick Bloem and Natasha Sharygina, editors, *Proceedings of the 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010*, pages 239–246. IEEE, 2010.

[156] Celia Wrathall. Complete Sets and the Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.

[157] Yining Wu and Martin W. A. Caminada. A Labelling-Based Justification Status of Arguments. *Studies in Logic*, 3(4):12–29, 2010.

[158] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient Conflict Driven Learning in Boolean Satisfiability Solver. In *Proceedings of the International Conference on Computer-Aided Design, ICCAD 2001*, pages 279–285, 2001.

[159] Charlie Shucheng Zhu, Georg Weissenbacher, Divjyot Sethi, and Sharad Malik. SAT-based Techniques for Determining Backbones for Post-silicon Fault Localisation. In Zeljko Zilic and Sandeep K. Shukla, editors, *Proceedings of the IEEE International High Level Design Validation and Test Workshop, HLDVT 2011*, pages 84–91. IEEE, 2011.

APPENDIX A

Algorithms

In this appendix we list further algorithms of Chapter 3. In particular we list the algorithms for the stage semantics and the stage-ideal semantics. The algorithms for the former are very similar to the semi-stable variants, while the algorithm for the latter can be inferred from the eager algorithm.

Algorithm 15 shows the general procedure for the stage semantics and Algorithm 16 is its variant using a SAT-solver.

Algorithm 15 Stage(F, a, M)

Require: AF $F = (A, R)$, argument $a \in A$, mode $M \in \{\mathsf{Enum}, \mathsf{Cred}, \mathsf{co\text{-}Skept}\}$
Ensure: returns $\mathcal{S} = stg(F)$ if $M = \mathsf{Enum}$, *yes* if $M = \mathsf{Cred}$ ($\mathsf{co\text{-}Skept}$) and $\mathsf{Cred}_{stg}(a, F) = yes$ ($\mathsf{Skept}_{stg}(a, F) = no$), otherwise *no*

1: $\mathcal{S} = \emptyset$
2: **while** $\exists E, E \in ExtsExcl(cf, F, \mathcal{S}, \preceq_R)$ **do**
3: **while** $\exists E', E' \in GreaterExts(cf, F, E, \preceq_R)$ **do**
4: $E := E'$
5: **end while**
6: **if** $M = \mathsf{Cred}$ and $a \in E$ or $M = \mathsf{co\text{-}Skept}$ and $a \notin E$ **then**
7: return *yes*
8: **end if**
9: $\mathcal{S} := \mathcal{S} \cup \{E\}$
10: **end while**
11: return no (or $\mathcal{S}$ if $M = \mathsf{Enum}$)

For the SAT extension based algorithms we describe here the variant for computing all skeptically accepted argument of an AF w.r.t. stage semantics in Algorithm 17.

Finally Algorithm 18 shows how to compute the stage-ideal extension of an AF.

Algorithm 16 Stage-SAT(F, a, M)

Require: AF $F = (A, R)$, argument $a \in A$ and mode $M \in \{\mathsf{Enum}, \mathsf{Cred}, \mathsf{co\text{-}Skept}\}$
Ensure: returns $\mathcal{I} = stg(F)$ if $M = \mathsf{Enum}$, *yes* if $M = \mathsf{Cred}$ ($\mathsf{co\text{-}Skept}$) and $\mathsf{Cred}_{stg}(a, F) = yes$ ($\mathsf{Skept}_{stg}(a, F) = no$), otherwise *no*

$excl = \top$
$\mathcal{I} = \emptyset$
while $\exists I, I \models cf_{A,R} \wedge range_{A,R} \wedge excl$ **do**
  while $\exists J, J \models cf_{A,R} \wedge range_{A,R} \wedge \bigvee_{a \in (A \cap I)} \overline{a} \wedge \bigvee_{a \in (A \setminus I)} \overline{a}$ **do**
    $I := J$
  end while
  if $M = \mathsf{Cred}$ and $a \in I$ or $M = \mathsf{co\text{-}Skept}$ and $a \notin I$ **then**
    return *yes*
  end if
  $excl := excl \wedge \bigvee_{a \in (A \setminus I)} a \wedge \bigwedge_{\overline{a} \in (\overline{A} \cap I)} ((\bigwedge_{\overline{x} \in (\overline{A} \setminus I)} \neg \overline{x}) \rightarrow \overline{a})$
  $\mathcal{I} := \mathcal{I} \cup \{I \cap A\}$
end while
return no (or $\mathcal{I}$ if $M = \mathsf{Enum}$)

Algorithm 17 MCS-$\mathsf{AllSkept}_{stg}(F)$

Require: AF $F := (A, R)$
Ensure: returns $\mathsf{AllSkept}_{stg}(F)$

$\phi = \{a_i \vee C_i \mid C_i \in all\_in\_range_{A,R}\}$ with $L := \bigcup_i \{a_i\}$ a set of fresh atoms
$\psi = cf_{A,R} \cup \phi$
$k = 0$
$X = A$
while ψ is satisfiable and $k \leq |A|$ **do**
  $\psi_k = \psi \cup \mathit{AtMost}(k, L)$
  $X = X \cap \mathit{Probing}(\psi_k)$
  while ψ_k is satisfiable **do**
    let I be such that $I \models \psi_k$
    let D be $\{\neg a_i \mid a_i \in L \wedge I(a_i) = \mathbf{t}\}$
    $\psi_k = \psi_k \wedge D$
    $\psi = \psi \wedge D$
  end while
  $k = k + 1$
end while
return X

Algorithm 18 MCS-Stage-ideal(F)

Require: AF $F = (A, R)$
Ensure: returns $stg\text{-}idl(F)$
1: $X = \mathsf{MCS\text{-}AllSkept}_{stg}(F)$
2: *return* $\hat{\mathcal{F}}_F^{|A|}(X)$

APPENDIX B

Curriculum Vitae

Johannes Peter Wallner
Curriculum Vitae

Address:	Vienna University of Technology Institute of Information Systems Database and Artificial Intelligence Group Favoritenstrasse 9 A-1040 Wien, Austria
Phone:	+43-1-58801-18409
Fax:	+43-1-58801-9-18409
E-Mail:	wallner@dbai.tuwien.ac.at
Website:	http://www.dbai.tuwien.ac.at/user/wallner/
Citizenship:	Austria
Birthday:	13.5.1984

Education

University

2011 - present
- Ph.D. study, Vienna University of Technology.
  Supervisor: Stefan Woltran

2008 - 2010
- Master study, Vienna University of Technology: "Computational Intelligence", passed with distinction.
  Thesis: *A Hybrid Approach for Model-Based Random Testing*
  Supervisor: Univ. Prof. Dr. Franz Wotawa
  Co-supervisor: Univ. Ass. Dr. Bernhard Peischl

2003 - 2008
- Bachelor study, Vienna University of Technology: "Software & Information Engineering"

Participation in Doctoral Programme

2011 - present
- Doctoral programme *Mathematical Logic in Computer Science*, Vienna University of Technology.

Participation in Doctoral Consortium/Summer School/Winter School

Feb. 2014
- Advanced Winter School on Reasoning Engines for Rigorous System Engineering, ReRiSE 2014.

Jul. 2013
- Advanced Course on AI (ACAI) 2013 summer school; student session topic: *Advanced Procedures for Hard Problems in Abstract Argumentation*.

Jun. 2012
- Doctoral Consortium at the thirteenth conference on Principles of Knowledge Representation and Reasoning (KR) 2012. Topic: *Computational Properties of Abstract Dialectical Frameworks*.

Academic Working Experience

Project Employment

Jun. 2013 - present
- Project assistant in the Database and AI Group, Vienna University of Technology.
  International project (Germany/Austria): *Abstract Dialectical Frameworks: Advanced Tools for Formal Argumentation*.
  Funded by Deutsche Forschungsgemeinschaft (DFG) and Austrian Science Fund (FWF) through project I1102
  Project leaders: Gerd Brewka and Stefan Woltran

Sept. 2012 - May 2013
- Project assistant in the Database and AI Group, Vienna University of Technology.
  Project: *SEE: SPARQL Evaluation and Extensions*.
  Funded by WWTF – Wiener Wissenschafts-, Forschungs- und Technologiefonds (ICT 12-015)
  Project leader: Reinhard Pichler

May 2011 - Aug. 2012
- Project assistant in the Database and AI Group, Vienna University of Technology.
  Project: *New Methods for Analyzing, Comparing, and Solving Argumentation Problems*.
  Funded by WWTF – Wiener Wissenschafts-, Forschungs- und Technologiefonds (ICT 08-028)
  Project leader: Stefan Woltran

Other Project Activities

2012 - 2013
- Project member in bilateral project Austria/Slovakia
  Project: *New Directions in Abstract Argumentation*
  Funded by Slovenská akademická informaná agentúra (SAIA) and Österreichischer Austauschdienst (ÖAD); project number 2012-03-15-0001
  Project coordinators: Jozef Siška, Comenius Univ. Bratislava and Stefan Woltran

Reviewing

- ECAI 2012, ESSLLI 2013, CILC 2013 and LPNMR 2013

Professional Activities

- Fourth ASP Competition 2013, member of organizing committee
- Assistance in writing project proposal (co-author): *Abstract Dialectical Frameworks: Advanced Tools for Formal Argumentation*, funded by DFG/FWF.

Research Visits

Apr. - Jun. 2012 • Gerhard Brewka, Leipzig University

Scientific Talks

16. Sep. 2013 • *Advanced SAT Techniques for Abstract Argumentation*. CLIMA'13

5. July. 2013 • *SAT-based Argumentation Systems*. Advanced Course on AI (ACAI) 2013, student session.

2. Apr. 2012 • *Knowledge Base Change and Abstract Dialectical Frameworks*. Dynamics of Argumentation, Rules and Conditionals (DARC) workshop.

28. Sept. 2011 • *Making Use of Advances in Answer-Set Programming for Abstract Argumentation Systems*. INAP'11

23. Aug. 2010 • *A hybrid approach for model-based random testing*. VALID'10

Awards and Grants

- ECCAI Travel Award for attending ACAI 2013
- Grant for attending doctoral consortium at KR 2012
- Distinguished Student Paper Prize for "Complexity-Sensitive Decision Procedures for Abstract Argumentation" at KR 2012

Teaching

- Course "Abstract Argumentation" at the Vienna University of Technology (winter 2012)
- Co-advisor for master thesis: Stefan Ellmauther, *Abstract Dialectical Frameworks: Properties, Complexity, and Implementation*, 2012

Language skills

- German (native)
- English (fluent)

Publications

Journals

[1] Wolfgang Dvořák, Matti Järvisalo, Johannes P. Wallner, and Stefan Woltran. Complexity-Sensitive Decision Procedures for Abstract Argumentation. *Artificial Intelligence*, 206:53–78, 2014.

[2] Axel Polleres and Johannes P. Wallner. On the relation between SPARQL1.1 and Answer Set Programming. *Journal of Applied Non-Classical Logics*, 23(1–2):159–212, 2013.

Conferences and Workshops

[3] Hannes Strass and Johannes P. Wallner. Analyzing the Computational Complexity of Abstract Dialectical Frameworks via Approximation Fixpoint Theory. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning, KR 2014*, Vienna, Austria, July 2014. To appear.

[4] Mario Alviano, Francesco Calimeri, Günther Charwat, Minh Dao-Tran, Carmine Dodaro, Giovambattista Ianni, Thomas Krennwallner, Martin Kronegger, Johannes Oetsch, Andreas Pfandler, Jörg Pührer, Christoph Redl, Francesco Ricca, Patrik Schneider, Martin Schwengerer, Lara K. Spendier, Johannes P. Wallner, and Guohui Xiao. The Fourth Answer Set Programming Competition: Preliminary Report. In Pedro Cabalar and Tran Cao Son, editors, *Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2013*, volume 8148 of *Lecture Notes in Artificial Intelligence*, pages 42–53, Corunna, Spain, September 2013. Springer.

[5] Thomas Ambroz, Günther Charwat, Andreas Jusits, Johannes P. Wallner, and Stefan Woltran. ARVis: Visualizing Relations between Answer Sets. In Pedro Cabalar and Tran Cao Son, editors, *Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2013*, volume 8148 of *Lecture Notes in Artificial Intelligence*, pages 73–78, Corunna, Spain, September 2013. Springer.

[6] Günther Charwat, Giovambattista Ianni, Thomas Krennwallner, Martin Kronegger, Andreas Pfandler, Christoph Redl, Martin Schwengerer, Lara K. Spendier, Johannes P. Wallner, and Guohui Xiao. VCWC: A Versioning Competition Workflow Compiler. In Pedro Cabalar and Tran Cao Son, editors, *Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2013*, volume 8148 of *Lecture Notes in Artificial Intelligence*, pages 233–238, Corunna, Spain, September 2013. Springer.

[7] Sylwia Polberg, Johannes P. Wallner, and Stefan Woltran. Admissibility in the Abstract Dialectical Framework. In João Leite, Tran Cao Son, Paolo Torroni, Leon van der Torre,

and Stefan Woltran, editors, *Proceedings of the 14th International Workshop on Computational Logic in Multi-Agent Systems, CLIMA 2013*, volume 8143 of *Lecture Notes in Artificial Intelligence*, pages 102–118, Corunna, Spain, September 2013. Springer.

[8] Johannes P. Wallner, Georg Weissenbacher, and Stefan Woltran. Advanced SAT Techniques for Abstract Argumentation. In João Leite, Tran Cao Son, Paolo Torroni, Leon van der Torre, and Stefan Woltran, editors, *Proceedings of the 14th International Workshop on Computational Logic in Multi-Agent Systems, CLIMA 2013*, volume 8143 of *Lecture Notes in Artificial Intelligence*, pages 138–154, Corunna, Spain, September 2013. Springer.

[9] Gerhard Brewka, Stefan Ellmauthaler, Hannes Strass, Johannes P. Wallner, and Stefan Woltran. Abstract Dialectical Frameworks Revisited. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, pages 803–809, Beijing, China, August 2013. AAAI Press / IJCAI.

[10] Wolfgang Dvořák, Sarah A. Gaggl, Johannes P. Wallner, and Stefan Woltran. Making Use of Advances in Answer-Set Programming for Abstract Argumentation Systems. In Hans Tompits, Salvador Abreu, Johannes Oetsch, Jörg Pührer, Dietmar Seipel, Masanobu Umeda, and Armin Wolf, editors, *Proceedings of the 19th International Conference on Applications of Declarative Programming and Knowledge Management, INAP 2011, Revised Selected Papers*, volume 7773 of *Lecture Notes in Artificial Intelligence*, pages 114–133. Springer, 2013.

[11] Günther Charwat, Johannes P. Wallner, and Stefan Woltran. Utilizing ASP for Generating and Visualizing Argumentation Frameworks. In Michael Fink and Yuliya Lierler, editors, *Proceedings of the Fifth Workshop on Answer Set Programming and Other Computing Paradigms, ASPOCP 2012*, pages 51–65, Budapest, Hungary, September 2012.

[12] Stefan Ellmauthaler and Johannes P. Wallner. Evaluating Abstract Dialectical Frameworks with ASP. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the Fourth International Conference on Computational Models of Argument, COMMA 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 505–506, Vienna, Austria, September 2012. IOS Press.

[13] Wolfgang Dvořák, Matti Järvisalo, Johannes P. Wallner, and Stefan Woltran. Complexity-Sensitive Decision Procedures for Abstract Argumentation. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning, KR 2012*, pages 54–64, Rome, Italy, June 2012. AAAI Press.

[14] Wolfgang Dvořák, Sarah A. Gaggl, Johannes P. Wallner, and Stefan Woltran. Making Use of Advances in Answer-Set Programming for Abstract Argumentation Systems. In Salvador Abreu, Johannes Oetsch, Jörg Pührer, Dietmar Seipel, Hans Tompits, Masanobu Umeda, and Armin Wolf, editors, *Proceedings of the 19th International Conference on*

Applications of Declarative Programming and Knowledge Management, INAP 2011, pages 117–130, Vienna, Austria, September 2011.

[15] Stefan Mohacsi and Johannes P. Wallner. A hybrid approach for model-based random testing. In Lydie du Bousquet, Juho Perälä, and Pascal Lorenz, editors, *Proceedings of the Second International Conference on Advances in System Testing and Validation Lifecycle, VALID 2010*, pages 10–15, Nice, France, August 2010. IEEE.

Miscellaneous

[16] Günther Charwat, Wolfgang Dvořák, Sarah A. Gaggl, Johannes P. Wallner, and Stefan Woltran. Implementing Abstract Argumentation - A Survey. Technical Report DBAI-TR-2013-82, Technische Universität Wien, 2013.

[17] Hannes Strass and Johannes P. Wallner. Analyzing the Computational Complexity of Abstract Dialectical Frameworks via Approximation Fixpoint Theory. Technical Report 2, Computer Science Institute, Leipzig University, 2013.

[18] Wolfgang Dvořák, Matti Järvisalo, Johannes P. Wallner, and Stefan Woltran. CEGARTIX: A SAT-Based Argumentation System. Presented at the Pragmatics of SAT Workshop (PoS 2012), 2012.

[19] Wolfgang Dvořák, Sarah A. Gaggl, Johannes P. Wallner, and Stefan Woltran. Making Use of Advances in Answer-Set Programming for Abstract Argumentation Systems. Technical Report DBAI-TR-2011-70, Technische Universität Wien, 2011.

[20] Johannes P. Wallner. Hybrid approach for model-based random testing. Master's thesis, Technische Universität Wien, Institut für Informationssysteme, 2010.

Last updated: April 2, 2014

References

Associate Prof. Dr. Stefan Woltran
Vienna University of Technology
Institute of Information Systems
Database and Artificial Intelligence Group
Favoritenstraße 9-11
A-1040 Vienna, Austria

Prof. Dr. Reinhard Pichler
Vienna University of Technology
Institute of Information Systems
Database and Artificial Intelligence Group
Favoritenstraße 9-11
A-1040 Vienna, Austria

Prof. Dr. Gerhard Brewka
Leipzig University
Computer Science Institute
Intelligent Systems Department
Augustusplatz 10
04109 Leipzig, Germany