Computational Approaches to Reasoning in Structured Argumentation

Tuomo Lehtonen

Doctoral dissertation, to be presented for public examination with the permission of the Faculty of Science of the University of Helsinki in Auditorium E204, Physicum, Gustaf Hällströmin katu 2 on the 16th of June, 2023, at 12 o’clock.
Supervisors
Matti Järvisalo, University of Helsinki, Finland
Johannes P. Wallner, Graz University of Technology, Austria

Pre-examiners
Mario Alviano, University of Calabria, Italy
Francesca Toni, Imperial College London, United Kingdom

Opponent
Henry Prakken, Utrecht University and University of Groningen, The Netherlands

Custos
Matti Järvisalo, University of Helsinki, Finland

Contact information
Department of Computer Science
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/
Telephone: +358 2941 911

Copyright © 2023 Tuomo Lehtonen
ISSN 1238-8645 (print)
ISSN 2814-4031 (online)
ISBN 978-951-51-9324-7 (paperback)
ISBN 978-951-51-9325-4 (PDF)
Helsinki 2023
Unigrafia
Computational Approaches to Reasoning in Structured Argumentation

Tuomo Lehtonen

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
tuomo.lehtonen@helsinki.fi

Helsinki, June 2023, 94 + 117 pages
ISSN 1238-8645 (print)
ISSN 2814-4031 (online)
ISBN 978-951-51-9324-7 (paperback)
ISBN 978-951-51-9325-4 (PDF)

Abstract

Formal argumentation is a prominent research area within artificial intelligence, aiming to capture rational reasoning when faced with conflicting claims. Argumentation has been applied in fields such as law, medicine, debate analysis and explainable artificial intelligence. We focus on structured argumentation formalisms, which are aimed at capturing argumentative reasoning starting with a base of (uncertain) knowledge and ending with identifying acceptable claims. In particular, we investigate efficient approaches to deciding the acceptability of claims as concrete computational problems in the context of two central structured formalisms, assumption-based argumentation (ABA) and abstract rule-based argumentation (ASPIC†). In both ABA and ASPIC†, arguments for claims are supported by premises (named assumptions in ABA) via a specified inference system, and conflicts are identified based on the elements used to support a claim. Notably, the explicit construction of all arguments that a collection of premises and rules gives rise to is not polynomially bounded.

As a main contribution of this thesis, we develop new algorithmic approaches for efficiently computing acceptable claims that leave the construction of arguments implicit. We focus on the logic programming and default logic instantiations of ABA, the logic programming instantiation of ABA† (ABA extended with preferences over assumptions), and an instantiation of ASPIC† with a language consisting of atoms. To avoid the
explicit construction of arguments for acceptance in ASPIC\(^+\), we establish new characterizations of central semantics which conventionally rely on the construction of arguments. We develop algorithms based on answer set programming (ASP) and propositional satisfiability (SAT) for deciding the acceptance of claims in ABA and ASPIC\(^+\), motivated by the success of applying ASP and SAT solvers to various computationally hard problems. We show empirically that our algorithmic approach has significantly better runtime performance and scalability than existing algorithmic approaches. Our results suggest that it is advantageous to avoid the construction of arguments in practical algorithms and to use efficient declarative methods directly on structured argumentation. Implementations of the algorithms developed in the thesis are made available in open source.

Further, we establish the complexity of reasoning problems in ASPIC\(^+\) and ABA\(^+\) under several semantics. We show that the complexity of deciding the acceptance of claims in ASPIC\(^+\) with no preferences under admissible, complete, stable and preferred semantics is the same as the complexity of the corresponding problems in ABA. We prove that when preferences are included under the well-behaved weakest-link principle and elitist ordering in ASPIC\(^+\), the complexity of deciding acceptance under stable semantics is higher than without preferences. For ABA\(^+\), we show that the complexity of acceptance under stable semantics is the same with and without preferences, but is higher under admissible semantics with preferences included.

Computing Reviews (2012) Categories and Subject Descriptors:
Computing methodologies → Nonmonotonic, default reasoning and belief revision
Theory of computation → Constraint and logic programming
Computing methodologies → Logic programming and answer set programming
Theory of computation → Problems, reductions and completeness

General Terms:
argumentation, algorithms, computational complexity

Additional Key Words and Phrases:
formal argumentation, structured argumentation, assumption-based argumentation, abstract rule-based argumentation, ASPIC\(^+\), answer set programming, propositional satisfiability
Acknowledgements

During the writing of this thesis I was a part of the Constraint Reasoning and Optimization group at the Department of Computer Science at the University of Helsinki, Finland. I am lucky to have been a part of this highly stimulating scientific environment. I would firstly like to thank my supervisors, Matti Järvisalo and Johannes P. Wallner, for their invaluable guidance in my academic journey and for their part in our joint research.

I am very grateful for the salaried doctoral researcher position provided by the University of Helsinki through an AI-specific funding call via the Doctoral Programme in Computer Science. I am also thankful for the financial support for conference attendance provided by Academy of Finland through projects 322869 and 328718. Furthermore, I wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

I have learned that research, like most things, is best enjoyed with people. I have enjoyed the interesting discussions and encounters I have had the pleasure of taking part in, both here in Helsinki and abroad. Thank you especially to my other coauthors, Andreas Niskanen, Anna Rapberger, Markus Ulbricht, Daphne Odekerken and AnneMarie Borg.

I am very grateful to Professor Mario Alviano and Professor Francesca Toni for pre-examining this thesis and to Professor Henry Prakken for agreeing to act as the opponent in my public examination. I want to thank Andreas Niskanen for his helpful feedback and Pirjo Moen for her aid in finalizing this thesis. Additionally, thank you to the reviewers of the papers in this thesis for their comments.

I am happy to have had lovely people in my life before, during, and hopefully after writing the current work. Thanks to my friends, family and other pleasant people. Finally, Tuisku, thank you for being such a delightful, lovable and loving companion.

Helsinki, May 2023
Tuomo Lehtonen
vi
# Contents

1 Introduction ................................. 1
   1.1 Original Publications ......................... 7
   1.2 Contributions of the Author ..................... 9
   1.3 Organization of the Thesis ..................... 10

2 Preliminaries ............................... 11
   2.1 Assumption-based Argumentation ................ 11
      2.1.1 Logic Programming Instantiation ............... 17
      2.1.2 Default Logic Instantiation .................... 17
      2.1.3 LP-ABA$^+$ ............................... 18
   2.2 Abstract Rule-based Argumentation ............... 20
   2.3 Computational Complexity ....................... 26
   2.4 Propositional Satisfiability .................... 27
   2.5 Answer Set Programming ....................... 28

3 Reasoning in Assumption-based Argumentation .......... 31
   3.1 Logic Programming Instantiation ................. 31
      3.1.1 Encoding ABA in ASP ......................... 33
      3.1.2 ASP Encodings for Admissible, Complete, Stable and
            Grounded Semantics ............................ 34
      3.1.3 Algorithms for Preferred and Ideal Semantics .... 36
      3.1.4 Empirical Evaluation ......................... 38
   3.2 Logic Programming Instantiation with Preferences ................................. 39
      3.2.1 Complexity of Reasoning in LP-ABA$^+$ .......... 41
      3.2.2 Algorithms for Deciding Acceptance in LP-ABA$^+$ .... 41
      3.2.3 Empirical Evaluation ......................... 44
   3.3 Default Logic Instantiation ..................... 46
      3.3.1 SAT-based Algorithms for Reasoning in DL-ABA .... 46
      3.3.2 Empirical Evaluation ......................... 48
4 Reasoning in Abstract Rule-based Argumentation 51
4.1 ASPIC$^+$ without Preferences .......................... 52
   4.1.1 Rephrasing Semantics .............................. 52
   4.1.2 ASP Encodings for Stable, Admissible, Complete and
         Preferred Semantics ............................. 55
   4.1.3 Empirical Evaluation .............................. 57
4.2 ASPIC$^+$ with Preferences ................................ 58
   4.2.1 Redefining Stable Semantics ....................... 58
   4.2.2 Algorithms for Acceptance under Stable Semantics . 62
   4.2.3 Empirical Evaluation .............................. 63

5 Conclusions 65

References 69

Appendix A Details on ASP Encodings 91
Chapter 1

Introduction

Argumentation is a major part of reasoning. When faced with uncertainty, inconsistencies, or conflicting beliefs, it is rational to put forward arguments for and against different stances and evaluate their plausibility. Such conflicts arise everywhere in life, from politics and science to disagreements over which restaurant to choose for dinner. Intuitively, reaching rational conclusions from conflicting information is more difficult than applying deductive reasoning. Nothing can be learned from applying deduction to inconsistent beliefs, since anything can be deduced from a contradiction. Due to the importance of argumentation, it is natural that argumentation has been studied from many viewpoints, especially philosophy, artificial intelligence and informal logic [137]. The particular viewpoint of this thesis is that of argumentation in artificial intelligence, or formal argumentation [12, 16, 21, 47].

A central goal of formal argumentation is the formalization of argumentation processes so as to allow the construction of autonomous agents capable of this important facet of intelligent behaviour.

The applicability of argumentation formalisms as a basis for capturing rational reasoning as well as practical applications has been widely studied. Reductions from other paradigms of handling conflicting and uncertain information to argumentation formalisms corroborate the fact that argumentation captures broad facets of reasoning [138]. For example, many forms of non-monotonic reasoning, including logic programming, default logic, and autoepistemic logic have been given argumentative interpretations [30, 70]. In terms of applications, argumentation has been shown to be relevant in various different settings [12], including medical applications [44, 53, 61, 85, 87, 110, 147, 179], explainable artificial intelligence and decision making [59, 60, 62, 142, 143, 166, 180], legal reasoning [19, 20, 22, 134, 136, 139, 140, 141, 172], and multi-agent systems [41, 65, 86, 132].
There are multiple aspects to argumentation, with equally varied formal approaches [12]. In this thesis we focus on argumentation as a reasoning process, omitting phenomena such as dialogue and rhetoric. Argumentation as a reasoning process can be seen as consisting of four steps: the construction of arguments for claims from a knowledge base, the determination of conflicts between arguments, the evaluation of the acceptability of arguments, and finally the identification of justified conclusions [38]. There are various approaches to formalizing the process. The approaches are traditionally divided into structured [23, 26, 95, 122] and abstract argumentation formalisms [6, 15, 35, 70]. This thesis focuses on structured argumentation formalisms, which capture all of the steps, from the application of inference rules to support claims to the identification of justified conclusions. Abstract argumentation, on the other hand, focuses on the third step, the evaluation of the acceptability of arguments. In abstract argumentation, the arguments and conflicts between arguments are assumed as given, and acceptable sets of arguments are identified based on an attack relation between arguments. An underlying theme across this thesis is that when considering the whole argumentative reasoning process, the intermediate steps of explicitly constructing arguments and attacks may be needlessly inefficient. We investigate the argumentative reasoning process in the context of central structured argumentation formalisms and propose efficient approaches to the identification of justified conclusions that leave the construction of arguments and attacks implicit.

Structured argumentation formalisms include components for inferring claims, namely a form of premises and inference rules. The inference system specifies when a set of premises supports an argument for a claim. Attacks between arguments depend on the structure of the arguments and are determined based on elements such as the notion of what contradicts a specific premise. Determining the acceptability of arguments in structured argumentation formalisms is done by either deferring to conflict-resolution in abstract argumentation or by a mechanism specific to the structured formalism at hand. The criteria applied in evaluating the joint acceptability of sets of arguments are called argumentation semantics. As a basic requirement, an acceptable set of arguments must not attack itself and must defend itself against opposing viewpoints via counterattacks. Finally, deciding which claims can be rationally accepted is done based on the information of acceptability of sets of arguments. Two central computational problems for this are credulous and skeptical justification (also called acceptance) of a claim [78]. A claim is accepted credulously under a semantics if a set of arguments that supports the claim is accept-
able under the semantics. Skeptical acceptance of a claim requires that all acceptable sets of arguments support the claim. Different semantics and acceptance problems are suitable in different contexts. Furthermore, many different structured argumentation formalisms have been proposed with different characteristics and properties, thereby also being suited for different application scenarios [16, 23]. Prominent structured formalisms include assumption-based argumentation (ABA) [30, 57, 72, 162], abstract rule-based argumentation (ASPIC$^+$) [38, 121, 122, 135], defeasible logic programming (DeLP) [25, 93, 94, 95], deductive argumentation [24, 26], and Carneades [104].

We focus specifically on acceptance problems in different variants of ABA and ASPIC$^+$, which are arguably two of the most studied structured argumentation formalisms. By focusing on acceptance problems in structured argumentation formalisms, we capture all of the four steps of the argumentative reasoning process. ABA and ASPIC$^+$ do not restrict the kind of inference system that can be used for deriving arguments. This allows for using an inference system with simple rules with atomic components or more expressive consequence relations. Thus ABA and ASPIC$^+$ maintain flexibility in the manner of inferring arguments to accommodate different applications. Among other formalisms, Carneades and versions of DeLP and ABA have been reconstructed as fragments of ASPIC$^+$ [92, 135, 165]. Fragments of ASPIC$^+$ have also been reconstructed as ABA [74, 107]. As an example of an inference system, arguably the most-studied instantiation of ABA is the logic programming instantiation, where the rules are similar to ground normal logic programming rules: one atom is the head of the rule, the body of the rule contains finitely many atoms and the head is derived if all of the body elements are derived. Both ABA and ASPIC$^+$ can be used to instantiate abstract argumentation frameworks. ASPIC$^+$ was explicitly conceived as a principled way of constructing abstract argumentation frameworks, allowing the investigation of guaranteed rationality properties for argumentative reasoning [5, 37, 38, 39, 66, 74, 108, 109, 121, 135]. ABA semantics were originally defined on sets of assumptions [30], but were later found to directly correspond to abstract argumentation semantics defined on sets of arguments [73]. This direct connection to abstract argumentation can be seen as beneficial, as abstract argumentation is one of the most prominent approaches to formal argumentation [15, 137].

We consider ABA and ASPIC$^+$ both with and without preferences, which can be used to include additional comparative information, such as the plausibility of a fact or the wishes of an agent. Indeed, preferences are considered an important part of argumentative reasoning [6, 7, 8, 9, 18, 25,
Many approaches to structured argumentation incorporate preferences, either directly as a built-in feature, as in the case of ASPIC\textsuperscript{+}, or through extensions of the formalisms, as in the case of ABA\textsuperscript{+} [14, 56, 63, 64] and p-ABA [168, 169], two distinct approaches to incorporate preferential information in ABA. Typically attacks fail (as in ASPIC\textsuperscript{+}) or are reversed (as in ABA\textsuperscript{+}) if the attacker is less preferred than the attacked argument. There are also approaches to model preferences implicitly, via the primary components of, for instance, ABA [86, 153, 160]. Different approaches for reasoning with preferences have different foreseeable application scenarios. For instance, ABA\textsuperscript{+} has been applied in medical reasoning [61].

The complexity of deciding the acceptance of claims varies depending on the formalism, semantics, and the manner of deriving claims. Choosing a more expressive system for deriving claims makes the complexity of acceptance problems higher. The complexity of deciding acceptance of claims in ABA is largely well understood [58, 69, 77, 78, 113]. For example, in the logic programming instantiation of ABA, deciding if a given set of assumptions supports a claim is a polynomial-time problem. It follows that in this instantiation the complexity of deciding the acceptance of a claim is the same as the complexity of deciding acceptance of arguments in abstract argumentation, namely NP-hard (and in many cases NP-complete) under most of the prominent semantics [78]. Fewer complexity results for deciding acceptance in ASPIC\textsuperscript{+} have been established, and only for relatively restricted fragments [130].

In this thesis, we make both algorithmic and complexity-theoretic contributions for deciding acceptance of claims in ABA and ASPIC\textsuperscript{+}. We develop approaches for reasoning directly on the primary components of the formalisms. An essential component for achieving this is an alternative characterization of ASPIC\textsuperscript{+} semantics, defined directly in terms of premises and defeasible rules. Originally, the acceptance of a claim in ASPIC\textsuperscript{+} was defined via the construction of an abstract argumentation framework and deciding the satisfaction of the given semantics in the abstract argumentation framework. Computationally this is problematic because the number of abstract arguments that an ASPIC\textsuperscript{+} instance gives rise to is not bounded polynomially when using the specified argument generation procedure, even when disregarding arguments redundant in terms of their support [152]. The ASPIC\textsuperscript{+} definitions for acceptance of a claim in a sense suggest a step that exceeds the complexity bounds of the problem. In the worst case, deciding the acceptance of a claim by first generating an abstract argumentation framework results in constructing an exponential
input for a subprocedure that has the same complexity as the whole acceptance problem. Previous algorithms for reasoning in ASPIC⁺ indeed have an explicit argument construction step [14, 115, 150, 154]. Furthermore, even though the semantics of ABA were originally defined without reference to the explicit construction of arguments [30], many algorithms based on constructing an abstract argumentation framework have been proposed. The construction of arguments is similarly problematic for ABA as for ASPIC⁺, and has been observed to be a performance bottleneck [115]. Dedicated algorithms have also been developed for deciding acceptance of claims in ABA, such as dispute derivations [52, 161]. However, the existing dedicated algorithms have not been significantly more performant in practice [115]. This motivates developing efficient approaches to reasoning in ASPIC⁺ and ABA that avoid the explicit construction of an abstract argumentation framework.

In terms of our algorithmic approach, we develop algorithms for deciding the acceptance of claims declaratively. The declarative approach to solving a computational problem separates the problem description and the specifics of how to solve the problem. First, the problem of interest is encoded in a declarative language of choice such that solutions to the original problem and the encoded problem correspond. Secondly, a solution to the encoding (of a particular problem instance) is obtained with a solver for the declarative language. The declarative approach is an attractive choice for computationally hard problems as there are very efficient and optimized solvers [4, 13, 88, 97, 100, 101, 102, 116, 119, 151] for many declarative languages, such as propositional satisfiability (SAT) [28] and answer set programming (ASP) [103, 125]. Additionally, any improvements to the solvers benefit existing algorithms based on these methods. Declarative techniques have been successfully employed for many argumentation problems and a significant portion of efficient implementations for reasoning in abstract argumentation have been based on a declarative approach [1, 2, 17, 29, 42, 45, 79, 80, 82, 83, 84, 90, 91, 114, 126, 127, 128, 129, 156, 157, 159, 163, 170, 171]. For deciding acceptance in ABA and ASPIC⁺, approaches that first construct an abstract argumentation framework and then declaratively solve the abstract argumentation instance have been proposed [14, 115, 150, 154]. We use SAT and ASP as the declarative languages for the algorithms we develop in this thesis. Deciding SAT and deciding if a ground normal logic program exists are both NP-complete problems [51, 118]. A benefit of SAT and ASP is that modern solvers can be used incrementally, preserving information learned on previous solver calls to obtain solutions to successive calls faster [81, 99]. This may signif-
significantly speed up solving problems that are (presumably) beyond NP and thus need multiple calls to a solver.

The contributions of this thesis consist of practical algorithms and theoretical results for reasoning problems in ABA and ASPIC+. Our approach conforms to the complexity bounds for these problems by avoiding the explicit construction of arguments. For ASPIC+, we give new definitions for central semantics to achieve this. We also show complexity results for acceptance problems in ASPIC+ and ABA+ under central semantics.

In more detail, we consider two instantiations of ABA, namely the logic programming and propositional default logic instantiations. We refer to them as LP-ABA and DL-ABA, respectively. We also consider the logic programming instantiation of ABA+ (LP-ABA+), a generalization of ABA where preferences over assumptions are included and the attack relation modified based on the preferences. Our main contributions for ABA are twofold. First, we develop efficient declarative algorithms based on ASP and SAT for central reasoning problems for these three versions of ABA. Secondly, we prove complexity results for acceptance problems under many central semantics in LP-ABA+. For our computational approach, it is crucial to consider the acceptability of sets of assumptions instead of sets of arguments. The computational complexity of problems in LP-ABA+ has not, to the best of our knowledge, been previously studied. We show that the complexity of reasoning in LP-ABA+ is higher than in LP-ABA for many semantics, stable semantics being an exception. For these variants of ABA, we develop ASP encodings for deciding the acceptance of claims under many central semantics for which deciding acceptance is in (co)NP. Moreover, for semantics under which acceptance problems are (presumably) beyond NP, we introduce counterexample-guided abstraction refinement (CEGAR) [50] algorithms that make iterative ASP or SAT calls incrementally. We provide an empirical evaluation of the performance of our approach on a large benchmark set. Our implementations outperform previous approaches by orders of magnitude, in terms of both runtime, and the size of instances a given approach is able to solve within reasonable time.

Furthermore, we develop approaches to deciding the acceptability of claims in ASPIC+. In addition to ASPIC+ being more general, the acceptance of claims in ASPIC+ was originally exclusively defined via the potentially exponential translation to an abstract argumentation framework. To overcome this, we introduce a new perspective for deciding the acceptance of claims that leaves the argument construction implicit. We focus on the instantiation of ASPIC+ with the logical language consisting of atoms. As a fundamental element enabling our novel algorithmic ap-
proach, we define argumentation semantics for sets of defeasible elements, namely premises and defeasible rules, in contrast to the conventional definition of ASPIC\(^+\) semantics which sanctions the acceptability of sets of arguments. The new definitions can conceptually be seen as extending the assumption set definition of ABA semantics to ASPIC\(^+\). We give redefinitions for central semantics of ASPIC\(^+\) both without preferences and with a well-behaved manner of lifting preferences from premises and rules to arguments, namely the weakest link principle with elitist ordering [121]. Based on our definitions, we prove complexity results and develop efficient algorithms for these problems. We show that with no preferences included, deciding the acceptance of claims under admissible, complete, stable and preferred semantics has the same complexity as the corresponding problems in ABA and abstract argumentation. Furthermore, we show that the inclusion of preferences increases the complexity of deciding acceptance of claims under stable semantics. In particular, we show that deciding the acceptance of a claim credulously and skeptically is \(\Sigma_2^P\)-complete and \(\Pi_2^P\)-complete, respectively. Finally, we utilize our new definitions for deciding acceptance of claims in ASP-based algorithms. For deciding acceptance under stable semantics with preferences included, we develop CEGAR algorithms harnessing recent advances in incremental answer set solving. We evaluate our approach empirically and show that our approach significantly outperforms the other currently available implementations of deciding acceptance of claims in ASPIC\(^+\).

1.1 Original Publications

The core of this thesis consists of the following publications. We refer to these as Articles I-V.


We focus on LP-ABA and LP-ABA+ in Articles I and II and on DL-ABA in Article III. Articles IV and V consider ASPIC+, the former without preferences and the latter with preferences.

In more detail, in Article I we formulate ASP-based algorithms for credulous and skeptical acceptance under several semantics of LP-ABA and LP-ABA+. We show empirically that our algorithms outperform available algorithms for these problems. We also provide complexity results for deciding acceptance under stable, admissible and grounded semantics in LP-ABA+.

We focus on problems that are (presumably) beyond NP in Articles II and III. In Article II, we give an improved CEGAR algorithm for preferred semantics in ABA and show empirically that this is an improvement over our answer set optimization approach from Article I. We also give CEGAR algorithms for admissible and complete semantics in LP-ABA+ and show they outperform existing algorithms. Furthermore, we provide complexity results for complete semantics in LP-ABA+. In Article III we develop CEGAR algorithms for deciding acceptance of claims in DL-ABA under admissible, complete, stable and grounded semantics and empirically evaluate the scalability of the algorithms.

In Article IV we focus on ASPIC+ without preferences, and give an equivalent reformulation of admissible, complete, stable and preferred semantics on sets of defeasible elements, instead of sets of arguments. Enabled by this theoretical advancement, we develop ASP-based algorithms for these problems and empirically show their scalability. Complexity membership results follow from our algorithms, showing that the complexity of deciding acceptance of claims under these semantics is the same as the corresponding ones for ABA and abstract argumentation.
Finally, we investigate ASPIC$^+$ with preferences in Article V, focusing on the weakest-link principle with elitist ordering. We formulate a definition of stable semantics using defeasible elements that preserves acceptable claims. Based on our reformulations, we develop CEGAR algorithms with novel ASP encodings and show better empirical performance than existing algorithms. Furthermore, using our reformulations, we prove that the complexity of deciding both acceptance problems is complete for the second level of the polynomial hierarchy.

1.2 Contributions of the Author

Article I The present author formulated the ASP encodings and the proofs together with the other authors. The present author implemented the software and conducted the experiments, and, together with the other authors, analyzed the results.

Article II The present author formulated the ASP encodings and algorithms, implemented the software and conducted the experiments and, together with the other authors, analyzed the results.

Article III The present author formulated the SAT encodings and algorithms, implemented the software and conducted the experiments. The present author analyzed the empirical results together with the other authors.

Article IV The present author established the new definitions for the semantics and their correspondence to the classical formulation and formulated the ASP encodings together with the other authors. The present author implemented the software and conducted the experiments, and analyzed the results together with the other authors.

Article V The present author established the new definitions for stable semantics in ASPIC$^+$ with preferences and the correspondence between the new definitions and the conventional definitions and formulated the ASP encodings together with the other authors. The present author implemented the software and ran the experiments, and together with the other authors, analyzed the results.
1.3 Organization of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, we give the necessary background on the considered argumentation formalisms and computational techniques. We cover the main contributions of Articles I, II and III in Chapter 3, namely algorithmic approaches to reasoning in variants of assumption-based argumentation, along with underlying theoretical results. In Chapter 4, we cover the main contributions of Articles IV and V, namely an alternative characterization of many problems in ASPIC$^+$ along with algorithmic approaches and complexity results that this characterization enables. We conclude the thesis and outline directions for further research in Chapter 5.
Chapter 2
Preliminaries

We present the necessary preliminaries on the argumentation formalisms, computational problems and algorithmic methods relevant for this thesis. We recall the two formalisms we focus on: assumption-based argumentation (ABA) [30, 57, 162] and abstract rule-based argumentation (ASPIC+) [121, 122, 135]. Specifically, we focus on the following versions of ABA: the logic programming and default logic instantiations [30] as well as LP-ABA+ [14, 56, 64], a generalization of the logic programming fragment that equips ABA with preferences over assumptions. We focus on ASPIC+ instantiated with atoms as the logical language. Lastly, we briefly overview background on computational complexity, propositional satisfiability (SAT) and answer set programming (ASP) to the extent relevant to our discussion.

2.1 Assumption-based Argumentation

Assumption-based argumentation (ABA) [30, 57, 162] is a central approach to structured argumentation. There are numerous application avenues for ABA, including medical decision making [53, 57, 61] and explainable AI [62]. The basic concepts of ABA are assumptions and rules with which further sentences are derived. In general, ABA allows for choosing any logical language and rules for deriving claims. The specification of the form of the language and inference system (or deductive system) is called an instantiation of ABA. Perhaps the most studied instantiation of ABA is the logic programming instantiation (LP-ABA). We consider LP-ABA and the propositional default logic instantiation of ABA (DL-ABA). Default logic is one of the most prominent approaches to non-monotonic reasoning [10, 105, 146]. Default logic is more expressive than normal logic programming and, accordingly, deciding the acceptance of claims (under
central semantics) in the DL instantiation of ABA has a higher complexity than the corresponding problems in the LP instantiation.

Formally an ABA framework is defined as follows. We assume a deductive system \((\mathcal{L}, \mathcal{R})\), where \(\mathcal{L}\) is a formal language, i.e., a set of sentences, and \(\mathcal{R}\) a set of inference rules over \(\mathcal{L}\). A rule \(r \in \mathcal{R}\) has the form \(a_0 \leftarrow a_1, \ldots, a_n\) with \(a_i \in \mathcal{L}\). We denote the head of rule \(r\) by \(\text{head}(r) = a_0\) and the (possibly empty) body of \(r\) with \(\text{body}(r) = \{a_1, \ldots, a_n\}\).

**Definition 1.** An ABA framework is a tuple \(F = (\mathcal{L}, \mathcal{R}, A, -)\), where \((\mathcal{L}, \mathcal{R})\) is a deductive system, \(A \subseteq \mathcal{L}\) a non-empty set of assumptions, and \(-\) a function assigning a contrary sentence (from \(\mathcal{L}\)) to each assumption.

We assume further that the sets \(\mathcal{L}, \mathcal{R}\) and \(A\) are finite. We consider only ABA frameworks where assumptions do not occur as heads of rules. This implies a property called flatness (namely that a set of assumptions only derives assumptions contained in the set), which ensures desirable properties [57]. We sometimes do not explicitly mention all contraries with the intended meaning that an unmentioned contrary is not derivable in the framework.

**Example 1.** An ABA framework (of the logic programming instantiation) is illustrated in Figure 2.1(a) as an example. There are eight sentences in \(\mathcal{L}\), four of which are assumptions \(A\), and four rules in \(\mathcal{R}\). In this framework, the assumption \(a\) has no derivable contrary, and thus the contrary of \(a\) is left unmentioned.

A central concept in ABA is how a sentence can be derived from a given set of assumptions and a set of rules. Several notions of derivability have been studied [71, 76]. The original definition is forward-derivability [30] (denoted here by “\(\vdash\)”), which is defined via chaining rules to conclude a sentence. Tree-derivability (denoted here by “\(\models\)”), where a derivation takes the form of a proof tree, has since become popular [57]. In ABA (without preferences) the notions are equivalent with respect to the commonly considered reasoning problems [71, 76]. Tree-derivability is central for defining LP-ABA\(^+\), and the equivalence with forward-derivability does not carry over in general to LP-ABA\(^+\). On the other hand, forward-derivability forms the basis of our computational approach to deciding acceptance in ABA (Sections 3.1 and 3.3). We give here the definitions for both.

A tree-derivation is a proof tree where the leaves in the tree are assumptions and each internal node corresponds to a rule. A sentence \(s \in \mathcal{L}\) is tree-derivable from a set of assumptions \(X \subseteq A\) and rules \(R \subseteq \mathcal{R}\), denoted by \(X \models_R s\), if either \(X = \{s\}\) and \(R = \emptyset\), or there is a finite tree \(T\) with
sentences $\mathcal{L} = \{a, b, c, d, w, x, y, z\}$

assumptions $\mathcal{A} = \{a, b, c, d\}$

contraries $\overline{b} = x$, $\overline{c} = y$, $\overline{d} = z$

rules $\mathcal{R} = \{(w \leftarrow a), (y \leftarrow b, w), (x \leftarrow c), (z \leftarrow a, b)\}$

Figure 2.1: Example: (a) ABA framework, (b) tree-derivation of $y$, (c) and forward-derivation of $y$.

the root labeled with $s$, and the set of labels for the leaves is $X$ (with the possible addition of $\top$) and for each node that is not a leaf, labelled with $s' \in \mathcal{L}$, there is a rule $r \in \mathcal{R}$ with $s'$ as the head and the children of the node labeled with exactly the body elements of $r$. The symbol "$\top$" (which is not contained in $\mathcal{L}$) represents an empty rule body, signifying that the head of this rule is tree-derivable from the empty set. Derivation trees can be seen as argument structures, so that there is an argument for a sentence $s \in \mathcal{L}$ if and only if there is a tree-derivation for $s$.

**Example 2.** For the deductive system in Figure 2.1(a), a tree-derivation $\{a, b\} \models_{\mathcal{R}} y$ for the sentence $y$ is shown in Figure 2.1(b). Here $\mathcal{R} = \{(w \leftarrow a), (y \leftarrow b, w)\}$, i.e., the tree-derivation uses two rules.

Forward-derivability is defined by chaining rules from a set of assumptions, using the given assumptions or sentences derived earlier in the chain. Formally, a sentence $a \in \mathcal{L}$ is forward-derivable from a set $X \subseteq \mathcal{A}$ via rules $\mathcal{R}$, denoted by $X \vdash_{\mathcal{R}} a$, if $a \in X$ or there is a sequence of rules $(r_1, \ldots, r_n)$ such that $\text{head}(r_n) = a$ and for each rule $r_i$ we have $r_i \in \mathcal{R}$ and each sentence in the body of $r_i$ is derived from rules earlier in the sequence or is in $X$, i.e., $\text{body}(r_i) \subseteq X \cup \bigcup_{j<i} \text{head}(r_j)$. Forward-derivability is less strict than tree-derivability in the sense that an assumption set that contains assumptions that are not needed for a tree-derivation for a sentence forward-derives, but does not tree-derive, that sentence. In addition, the
exact rules that are used are not specified in forward-derivation in contrast to tree-derivation.

**Example 3.** For the deductive system in Figure 2.1(a), a sequence of two rules corresponding to a forward-derivation for \( y \) from \( X = \{a, b\} \) is shown in Figure 2.1(c).

There is a natural correspondence between the two notions of derivability in ABA without preferences [71, 76]. Namely, if there is a tree-derivation \( X \models_R s \) for some \( X \subseteq A \) and \( R \subseteq \mathcal{R} \), one can directly use the leaves and rules for showing \( X \vdash_R s \). On the other hand, if \( X \vdash_R s \) holds, then there must exist a subset \( R \subseteq \mathcal{R} \) and \( X' \subseteq X \) which forward-derive \( s \) and do not contain elements unnecessary for the derivation. These form a tree-derivation.

**Example 4.** Consider the deductive system in Figure 2.1. For this system it holds that \( \{a, b, c\} \vdash_R y \). However, \( \{a, b, c\} \models_R y \) for any \( R \subseteq \mathcal{R} \). To see this, note that one cannot form a tree-derivation with leaves \( a \), \( b \), and \( c \) and root \( y \) (there are no rules that connect these sentences). In other words, \( c \) is redundant in deriving \( y \).

If the type of derivation is not relevant, we generally simply refer to derivations. Similarly we may omit the subscript for the rules used if the context does not require specifying the rules. The deductive closure for an assumption set \( X \) with respect to rules \( \mathcal{R} \) is given by \( Th_{\mathcal{R}}(X) = \{a \in \mathcal{L} \mid X \models_R a \text{ for some } R \subseteq \mathcal{R}\} \). In ABA (without preferences), this can be equivalently defined with \( \vdash \).

Towards identifying acceptable sets of assumptions, a set of assumptions \( A \) attacks a set of assumptions \( B \) if one can derive the contrary of an assumption in \( B \) via the given deductive system. The semantics of ABA are based on the concepts of conflict-freeness and defense. An assumption set \( A \subseteq A \) is conflict-free if \( A \) does not attack itself and defends the assumption set \( B \subseteq A \) if \( A \) attacks all sets of assumptions that attack \( B \). We consider the following semantics.

**Definition 2.** Let \( F = (\mathcal{L}, \mathcal{R}, A, -) \) be an ABA framework. Further, let \( A \subseteq A \) be a conflict-free set of assumptions in \( F \). The set \( A \) is

- admissible if \( A \) defends itself,
- complete if \( A \) is admissible and contains every set of assumptions defended by \( A \),
- grounded if \( A \) is the intersection of all complete assumption sets,
Table 2.1: Admissible assumption sets of Example 5 compared to other semantics $\sigma$ and $ABA^+$ semantics $\prec-sigma$ with $a < d$.

<table>
<thead>
<tr>
<th>Set</th>
<th>$Th_R(\cdot)$</th>
<th>ABA semantics $\sigma$</th>
<th>$ABA^+$ semantics $\prec-sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>adm</td>
<td>adm</td>
</tr>
<tr>
<td>${a}$</td>
<td>${a, w}$</td>
<td>adm, com, grd, ideal</td>
<td>adm</td>
</tr>
<tr>
<td>${c}$</td>
<td>${c, x}$</td>
<td>adm</td>
<td>adm</td>
</tr>
<tr>
<td>${c,d}$</td>
<td>${c,d,x}$</td>
<td>adm</td>
<td>adm</td>
</tr>
<tr>
<td>${a,c}$</td>
<td>${a,c,w,x}$</td>
<td>adm</td>
<td>adm</td>
</tr>
<tr>
<td>${a,b}$</td>
<td>${a,b,w,y,z}$</td>
<td>adm, com, prf, stb</td>
<td>-</td>
</tr>
<tr>
<td>${a,c,d}$</td>
<td>${a,c,d,w,x}$</td>
<td>adm, com, prf, stb</td>
<td>adm, com, grd, prf, stb</td>
</tr>
</tbody>
</table>

- preferred if $A$ is admissible and there is no admissible set of assumptions $B$ with $A \subseteq B$;
- stable if each $\{x\} \subseteq A \setminus A$ is attacked by $A$, and
- ideal if $A$ is the $\subseteq$-maximal admissible assumption set within the intersection of all preferred assumption sets.

We use the term $\sigma$-assumption set for an assumption set under a semantics $\sigma \in \{adm, com, grd, stb, prf, ideal\}$, i.e., admissible, complete, grounded, stable, preferred, and ideal semantics, respectively.

**Example 5.** For the example ABA framework in Figure 2.1(a), Table 2.1 lists all admissible assumption sets (first column), the deductive closure (second column), and all semantics whose criteria the corresponding assumption set satisfies (third column). Consider the assumption set $\{a,b\}$: the deductive closure also contains $w$, $y$, and $z$. The last two sentences are contrary to assumptions $c$ and $d$ respectively. This means that $\{a,b\}$ attacks assumption sets $\{c\}$ and $\{d\}$, and every assumption set containing either $c$ or $d$. The set itself is attacked only by assumption sets containing the assumption $c$, since $Th_R(\{c\}) = \{x\}$ and $\overline{b} = x$. Since every assumption set containing $c$ is attacked by $\{a,b\}$, it holds that $\{a,b\}$ is admissible. This set is complete, since any assumption set $A \subseteq \mathcal{A}$ not contained in $\{a,b\}$ is not defended by $\{a,b\}$. To see that $\{a,b\}$ is also preferred, consider any proper superset $A$ of $\{a,b\}$: then either $c \in A$ or $d \in A$, which violates conflict-freeness. Thus, no proper superset is admissible. Finally, $\{a,b\}$ is stable since it attacks all other assumptions.

Further, the framework has three complete assumption sets: $\{a\}$, $\{a,b\}$, and $\{a,c,d\}$. The assumption set $\{a\}$ is the intersection of the complete
assumption sets and thus is the grounded assumption set. The other two
complete assumption sets are also preferred and stable. As \{a\} is the in-
tersection of the preferred assumptions sets and is also admissible, it is the
ideal assumption set.

We recall an alternative characterization of grounded semantics [30].
For an ABA framework \(F\) (that is flat), the grounded semantics can be
equivalently defined via the function \(\text{def}_F(A) = \{a \in A \mid A \text{ defends } \{a\}\}\).
The grounded assumption set is the least fixed point of \(\text{def}_F(\emptyset)\).

A central reasoning task is to find out whether a given sentence is
accepted under a given semantics. For this, two prominent computational
problem in ABA are credulous and skeptical acceptance of a sentence under
a given semantics.

**Definition 3.** Let \(F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \neg)\) be an ABA framework and \(\sigma\) be a
semantics. A sentence \(s \in \mathcal{L}\) is

- credulously accepted in \(F\) under semantics \(\sigma\) if there is a \(\sigma\)-assumption
  set \(A\) such that \(s \in \text{Th}_\mathcal{R}(A)\), and

- skeptically accepted in \(F\) under semantics \(\sigma\) if \(s \in \text{Th}_\mathcal{R}(A)\) for all
  \(\sigma\)-assumption sets \(A\).

There is both a unique grounded and a unique ideal assumption set.
This implies that credulous and skeptical reasoning coincide for both of
these semantics. Acceptance under grounded semantics also coincides with
skeptical reasoning under complete semantics. Since each preferred assump-
tion set is complete, each complete assumption set is admissible, and each
admissible assumption set is a subset of some preferred assumption set, it
follows that credulous reasoning under admissible, complete, and preferred
semantics coincide [30, 57].

**Example 6.** Continuing from Example 5 (see also Table 2.1), every sen-
tence is credulously accepted under admissible (and therefore also complete
and preferred) and stable semantics. No sentence is skeptically accepted
under admissible semantics (only sentences derivable from the empty set
are skeptically accepted under admissible semantics). The sentences \(a\) and
\(w\) are skeptically accepted under complete, preferred, and stable semantics,
and both credulously and skeptically accepted under grounded and ideal sem-
antics.

Next we provide the definitions of the variants of ABA we consider
in this work, namely the logic programming (LP-ABA) and default logic
instantiations (DL-ABA) of ABA [30], and the logic programming instan-
tiation of ABA\(^+\) (LP-ABA\(^+\)) [56, 64].
2.1 Assumption-based Argumentation

2.1.1 Logic Programming Instantiation

In LP-ABA, the language $L$ consists of atomic entities (no complex or compound structures), all components are finite, and no rule has an assumption as its head. Further, we remark that we assume that rules are given explicitly, and thus derivability is decidable in polynomial time.\(^1\)

**Definition 4 (LP-ABA).** An ABA framework $F = (L, R, A, \neg)$ belongs to the logic programming instantiation of ABA (LP-ABA) if

- $L$ is a set of atoms,
- sets $L$, $R$, and $A$ are finite, and
- for each rule $r \in R$:
  - $\text{body}(r)$ is finite,
  - the head of $r$ is not an assumption, and
  - $r$ is stated explicitly as a set of body elements and head.

2.1.2 Default Logic Instantiation

Towards defining the DL-ABA instantiation, we briefly recall concepts of propositional default logic [105, 146]. Let $(L_p, R_p)$ be a deductive system for propositional logic, i.e., $L_p$ is the set of all propositional formulas and there is a derivation for $a \in L_p$ from $X \subseteq L_p$ via $R_p$ if and only if $X$ classically entails $a$. One may select any sound and complete inference system for classical propositional logic as $R_p$. A propositional default theory is a pair $T = (W, D)$, where $W \subseteq L_p$ and $D$ is a set of default rules of the form $r = c \leftarrow a, Mb_1, \ldots, Mb_n$ with $c, a, b_1, \ldots, b_n \in L_p$ and $Mb_i \notin L_p$ ($Mb_i$ are not propositional formulas). We refer to $c$ as the conclusion, to $a$ as the prerequisite, and to $\{Mb_1, \ldots, Mb_m\}$ as the justifications of the default rule $r$. We use the shorthands $M(r) = \{Mb_1, \ldots, Mb_n\}$, $\text{prereq}(r) = a$ and $\text{conc}(r) = c$. Intuitively, $Mb$ is interpreted as $\neg b$ can not be proven and thus it is consistent to assume $b$. ABA instantiated with propositional default logic is defined as follows [30].

**Definition 5 (DL-ABA).** Let $(W, D)$ be a propositional default theory. The assumption-based argumentation framework corresponding to $(W, D)$ is $F = (L, R, W, A, \neg)$ with

---

\(^1\)Strictly speaking, the logic programming instantiation as originally defined [30] is slightly more restrictive than what we define. In particular, in the original definition $L$ contains only assumptions and the contraries of assumptions. Nevertheless, the instantiations exhibit the same complexity.
• $L = L_p \cup \{ M\alpha \mid \alpha \in L_p \}$,
• $R = R_p \cup D$,
• $A = \{ Mb \mid Mb \text{ occurs in some default rule in } D, b \in L_p \}$, and
• $\overline{Mb} = \neg b$ for all $Mb \in A$.

For brevity, as $(W, D)$ uniquely determines the corresponding ABA framework, we identify $(W, D)$ with the corresponding ABA framework $F$ and write $F = (W, D)$ referring to the ABA framework corresponding to $(W, D)$. Given a DL-ABA framework, derivability from a set of assumptions $A$ is defined by $W \cup A \vdash Ra$. In other words, $a$ is derivable from $A$ if there is a forward-derivation from $W \cup A$ concluding $a$ (note that $R$ includes default rules from the default theory). We illustrate DL-ABA with the following example.

Example 7. Consider $W = (\{\neg d \lor \neg c \})$ and the default rules $D$: \{(r_1 = c \leftarrow Mc), (r_2 = \neg a \leftarrow Mb), (r_3 = \neg b \leftarrow d, Ma)\}$. The corresponding ABA framework has by definition three assumptions, $A = \{Ma, Mb, Mc\}$ with contraries $\overline{Ma} = \neg a$, $\overline{Mb} = \neg b$, $\overline{Mc} = \neg c$. By itself, $\{Mc\}$ derives $c$, and moreover $d$ via $W$, because $c \land (\neg d \lor \neg c)$ entails $d$. $\{Mc, Ma\}$ also derives $\neg b$ since the rule $r_3$ is applicable once $d$ is derived. Thus $\{Mc, Ma\}$ attacks $\{Mb\}$ and is therefore admissible, complete and stable.

The grounded set is $\{Mc\}$. This set is not attacked and it does not defend any assumptions outside itself: $\{Ma\}$ is attacked by $\{Mb\}$ and $\{Mb\}$ is attacked by $\{Mc, Ma\}$. Furthermore, $\{Mb\}$ is admissible since it attacks $\{Ma\}$, which is needed to derive $\neg b$. $\{Ma\}$ is not admissible because to derive $\neg b$ to counter the attack from $Mb$, $Mc$ is needed to derive $d$.

2.1.3 LP-ABA$^+$

We move on to ABA$^+$, a generalization of ABA that includes preferences over assumptions. We focus on the logic programming fragment of ABA$^+$ (assuming the restrictions of Definition 4), which we refer to as LP-ABA$^+$.

Definition 6. An LP-ABA$^+$ framework is a tuple $F = (L, \mathcal{R}, A, \neg, \leq)$, where $(L, \mathcal{R}, A, \neg)$ is an LP-ABA framework and $\leq$ a preorder on $A$.

---

2 Example 1 in Article III is erroneous. We used the following DL-ABA framework as an example: $W = (\{\neg b \lor \neg a\} \land (\neg b \lor \neg c))$ and $D = \{(r_1 = a \leftarrow Ma), (r_2 = b \leftarrow Mb), (r_3 = c \leftarrow Mc), (r_4 = d \leftarrow a \land b, Md)\}$. Here $\{Ma, Mb\}$ is inconsistent with $W$ and thus attacks every assumption, because an inconsistent propositional formula entails every sentence. We claim in Article III, for instance, that $\{Md\}$ is the grounded assumption set. The grounded assumption set is in fact empty because $\{Md\}$ does not counterattack $\{Ma, Mb\}$, and other sets are similarly attacked.
A preorder is a reflexive and transitive binary relation. The strict counterpart \(<\) of \(\leq\) is defined as usual by \(a < b\) iff \(a \leq b\) and \(b \not\leq a\), for \(a, b \in A\).

The definition of attacks from LP-ABA frameworks is generalized as follows to \(<\)-attacks in LP-ABA\(^+\) frameworks, with “\(<\)” highlighting the fact that preferences, induced by \(<\), are used for deciding successful attacks. Although successful attacks are often called defeats in the argumentation literature, here we use the original notation of LP-ABA\(^+\) and use the term \(<\)-attacks.

**Definition 7.** Let \((L, R, A, -, \leq)\) be an LP-ABA\(^+\) framework, and \(A, B \subseteq A\) be two sets of assumptions. Assumption set \(A\) \(<\)-attacks \(B\) in \(F\) iff

- \(A' \models_R \bar{b}\), for some \(A' \subseteq A\), \(b \in B\), and \(\not\exists a' \in A'\) with \(a' < b\), or
- \(B' \models_R \bar{a}\) for some \(a \in A\) and \(B' \subseteq B\) such that \(\exists b' \in B'\) with \(b' < a\).

In words, set \(A\) \(<\)-attacks \(B\) iff (i) from \(A\), via subset \(A'\), one can tree-derive a contrary of an assumption \(b \in B\) and no member in \(A'\) is strictly less preferred than \(b\), or (ii) from \(B\), via subset \(B'\) one can tree-derive a contrary of an assumption \(a \in A\) and some member of \(B'\) is strictly less preferred than \(a\). Attacks of type (i) are normal \(<\)-attacks and those of type (ii) reverse \(<\)-attacks, with the intuition that the (non-preference based) conflict in (i) succeeds and in case of (ii) is reversed by the preference relation.

Comparing \(<\)-attacks to LP-ABA attacks, in an LP-ABA\(^+\) framework \((L, R, A, -, \leq)\) with \(\leq = \emptyset\) (i.e., no preferences are imposed), the normal \(<\)-attacks coincide with attacks and there are no reverse \(<\)-attacks. That is, without preferences one can use the standard LP-ABA definitions for attacks, and they coincide with normal \(<\)-attacks.

The notions of conflict-freeness and defense, as well as the semantics are straightforwardly generalized from LP-ABA to LP-ABA\(^+\) by replacing attacks with \(<\)-attacks. For conflict-freeness, there is no need for a new definition, since if \(A\) attacks \(B\) in an LP-ABA\(^+\) framework, then either \(A\) \(<\)-attacks \(B\) or \(B\) \(<\)-attacks \(A\). This means that each non-preference-based attack is either present as-is as a \(<\)-attack, or reversed, but never “lost”.

The definitions for the semantics for LP-ABA\(^+\) are generalized from LP-ABA by replacing the concepts of attack and defence with \(<\>-attack and \(<\>-defence. When referring to assumption sets for LP-ABA\(^+\), we use “\(<\)-assumption set” (for example, “\(<\>-admissible assumption set”) to distinguish from LP-ABA semantics. The reasoning tasks for LP-ABA\(^+\) are the same as for LP-ABA, replacing semantics \(\sigma\) with \(<\>-\sigma\).
Example 8. Recall the LP-ABA framework from Example 5, shown in Table 2.1. If we extend the example framework by the preference $a < d$, several semantics change, as shown in the rightmost column in Table 2.1. Notably, $\{a, b\}$ is not $<$-admissible (and also not $<$-complete, $<$-preferred, or $<$-stable). The reason for this is that from $\{a, b\}$ one can tree-derive $z = \bar{d}$, which means that $\{a, b\}$ attacks (when not taking preferences into account) the set $\{d\}$. However, since $d$ is strictly more preferred than $a$, it holds that this attack is reversed when using $<$-attacks. That is, $\{d\}$ reversely $<$-attacks $\{a, b\}$. In fact, $\{d\}$ is not $<$-attacked at all, implying that no set $<$-defends $\{a, b\}$. Under the preference, both $\{a\}$ and $\{d\}$ are $<$-unattacked. Further, $\{d\} <<$-defends $\{c\}$: $\{c\}$ is $<$-attacked only by $\{a, b\}$ (normally), which is countered by $\{d\}$ (reversely), and, thus, also by $\{a, d\}$. This means that $\{a, c, d\}$ is $<$-admissible. In fact, this set is also $<$-complete, $<$-grounded, $<$-preferred, and $<$-stable.

In contrast to ABA, the $<$-grounded semantics in LP-ABA$^+$ is in general not the least fixed point of the defence operator. However, in LP-ABA$^+$ frameworks satisfying the property described by the so-called Fundamental Lemma, the least fixed point of $\text{def}_F(\emptyset)$ is the $<$-grounded assumption set $[56, 63]$. The property states intuitively that adding a defended element to an admissible set preserves admissibility.

Property 1 (FL-property). An LP-ABA$^+$ framework $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-, \leq)$ satisfies the FL-property if $A \cup \{x\}$ is $<$-adm in $F$ for any $<$-admissible $A \subseteq \mathcal{A}$ in $F$ and $x \in \text{def}_F(A)$.

2.2 Abstract Rule-based Argumentation

We move on to abstract rule-based argumentation (ASPIC$^+$) $[121, 122, 135]$. ASPIC$^+$ is a central structured argumentation formalism, developed as a general framework for instantiating abstract argumentation frameworks (AFs) $[38, 135]$. ASPIC$^+$ has been shown to capture a wide variety of approaches to argumentation, including ABA $[135]$, Carneades $[92]$, a version of DeLP $[165]$, as well as other forms of reasoning under inconsistency, such as reasoning with maximal consistency $[11, 31]$, ontological knowledge bases $[55, 176]$ and prioritized default logic $[32, 175]$. ASPIC$^+$ subsumes flat ABA (without preferences) $[135]$. In this work we focus on an instantiation of ASPIC$^+$ where all sets constituting an ASPIC$^+$ framework are finite, the sentences are atoms and the rules are explicitly given as input.
We assume a set (language) $\mathcal{L}$ composed of atoms $x$. We start with contrariness: contraries represent an asymmetric conflict relation, while contradictories are symmetric.

**Definition 8.** Let a contrary function be $\neg : \mathcal{L} \rightarrow 2^\mathcal{L}$. We say that $a \in \mathcal{L}$ is a contrary of $b \in \mathcal{L}$ if $a \in \overline{b}$ and $b \notin \overline{a}$. We say that $a$ is a contradictory of $b$ if $a \in \overline{b}$ and $b \in \overline{a}$.

A central part of an ASPIC$^+$ framework is a knowledge base $\mathcal{K} \subseteq \mathcal{L}$ comprised of a defeasible part (ordinary premises $\mathcal{K}_p$) and a non-defeasible part (axioms $\mathcal{K}_n$). Claims are derived via a set of rules over $\mathcal{L}$, denoted by $\mathcal{R}$. This set is composed of defeasible rules $a_1, \ldots, a_n \Rightarrow b$ and strict rules $a_1, \ldots, a_n \rightarrow b$. We denote the set of defeasible rules by $\mathcal{R}_d$ and the set of strict rules by $\mathcal{R}_s$. When we do not distinguish between strict or defeasible rules, we write $a_1, \ldots, a_n \sim b$. A partial function $n : \mathcal{R}_d \rightarrow \mathcal{L}$ gives names to defeasible rules. For a rule $r = a_1, \ldots, a_n \sim b$, we denote its head by $\text{head}(r) = b$ and its body by $\text{body}(r) = \{a_1, \ldots, a_n\}$.

For preferences, we consider preorders (i.e., reflexive and transitive binary relations) $\leq = \leq_p \cup \leq_d$, composed of a preorder on ordinary premises $\leq_p$ and a preorder on defeasible rules $\leq_d$.

**Definition 9.** An argumentation theory (AT) is a tuple $(\mathcal{L}, \mathcal{R}, n, \neg, \mathcal{K}, \leq)$, with a knowledge base $\mathcal{K} \subseteq \mathcal{L}$, rules $\mathcal{R} = \mathcal{R}_d \cup \mathcal{R}_s$ over $\mathcal{L}$, a contrary function $\neg : \mathcal{L} \rightarrow 2^\mathcal{L}$, a partial function $n : \mathcal{R}_d \rightarrow \mathcal{L}$, and a preorder $\leq = \leq_p \cup \leq_d$.

We assume each part of an AT to be finite. Arguments are constructed from parts of an AT. An argument represents a “derivation tree” starting from elements in the knowledge base and uses rules to derive a conclusion.

**Definition 10.** Given an AT $T = (\mathcal{L}, \mathcal{R}, n, \neg, \mathcal{K}, \leq)$, the set of arguments in $T$ is inductively defined as follows:

- if $x \in \mathcal{K}$, then $A = x$ is an argument with $\text{Conc}(A) = x$, and
- if $A_1, \ldots, A_n$ are arguments, $x_i = \text{Conc}(A_i)$ for $1 \leq i \leq n$, and $(x_1, \ldots, x_n \sim x) \in \mathcal{R}$, then $A = A_1, \ldots, A_n \sim x$ is an argument with $\text{Conc}(A) = x$.

We make use of the following shorthands. Let $T = (\mathcal{L}, \mathcal{R}, n, \neg, \mathcal{K}, \leq)$ be an AT, and $A$ an argument in $T$. If $A = x \in \mathcal{K}$, then $\text{Sub}(A) = \{A\}$ and $\text{Rules}(A) = \emptyset$. Further, if $A = A_1, \ldots, A_n \sim x$, then $\text{Sub}(A) = \{A\} \cup \bigcup_{i=1}^n \text{Sub}(A_i)$, $\text{TopRule}(A) = (\text{Conc}(A_1), \ldots, \text{Conc}(A_n) \sim x)$, and $\text{Rules}(A) = \{\text{TopRule}(A)\} \cup \bigcup_{i=1}^n \text{Rules}(A_i)$. Finally, $\text{Prem}_d(A) = \text{Sub}(A) \cap \mathcal{K}_p$, and $\text{DefRules}(A) = \text{Rules}(A) \cap \mathcal{R}_d$. 
That is, we define shorthands for the subarguments (Sub) of an argument, the rules and defeasible rules in the argument (Rules and DefRules), the topmost rule (TopRule), and the ordinary premises (Prem). Further, 
\[\text{defPart}(A) = \text{Prem}(A) \cup \text{DefRules}(A).\]
If \(A \in \mathcal{K}\), then TopRule(A) is undefined. We extend the shorthands \(f \in \{\text{Sub}, \text{DefRules}, \text{Prem}\}\) to sets of arguments as \(f(A) = \bigcup_{A \in A} f(A)\). We allow only finite structures as arguments (i.e., arguments which are “trees” of finite size), and consider as arguments those arguments \(A\) for which Sub\((A)\) is finite (disallowing infinite chaining of rules, e.g., via \(x \Rightarrow x\)).

Conflicts among opposing viewpoints are defined via attacks between arguments. Three different types of attacks are distinguished in ASPIC\(^+\).

**Definition 11.** Given an AT \(T = (\mathcal{L}, \mathcal{R}, n, \neg, \mathcal{K}, \leq)\) and two arguments \(A\) and \(B\) in \(T\), argument \(A\) attacks argument \(B\) iff \(A\) undercuts, rebuts, or undermines \(B\), where

- \(A\) undercuts \(B\) (on \(B'\)) iff \(\text{Conc}(A) \in \overline{n(r)}\) for some \(B' \in \text{Sub}(B)\) such that TopRule\((B') = r\) is defeasible;
- \(A\) rebuts \(B\) (on \(B'\)) iff \(\text{Conc}(A) \in \overline{x}\) for some \(B' = B_1, \ldots, B_n \Rightarrow x \in \text{Sub}(B)\); and
- \(A\) undermines \(B\) (on \(x\)) iff \(\text{Conc}(A) \in \overline{x}\) and \(x \in \text{Prem}(B)\).

That is, an argument attacks another argument on the defeasible parts of the latter. Ordinary premises can be attacked by undermining, and defeasible rules can be attacked by rebutting the conclusion or undercutting the rule itself. For rebuts and undermining one distinguishes further if \(\text{Conc}(A)\) and \(x\) are contraries or contradictories: in the former case we say that \(A\) contrary undermines (rebuts) \(B\) and in the latter that \(A\) contradictory undermines (rebuts) \(B\).

The preorders \(\leq_p\) and \(\leq_d\) on the defeasible parts are lifted to strict partial orders (i.e., to irreflexive, asymmetric, and transitive binary relations) \(<_p\) and \(<_d\), via lifting operators. We focus on the elitist lifting [122]. As usual, the strict part of \(\leq_p\) (\(\leq_d\)) is defined by \(x <_p y\) iff \(x \leq_p y\) and \(y \not\leq_p x\) (same for \(\leq_d\)).

**Definition 12 (Elitist ordering).** Let \(\leq\) be a preorder on a set \(X\). Define \(<\) for two non-empty \(Y, Z \subseteq X\) by \(Y < Z\) iff \(\exists a \in Y\) s.t. \(\forall b \in Z\) we have \(a < b\). Moreover, \(\emptyset \not< Y\) and \(Z < \emptyset\) for each non-empty \(Z\).

That is, \(Z\) is preferred to \(Y\) if there is at least one element in \(Y\) that is strictly less preferred to each element in \(Z\). The empty set is a special case,
being strictly preferred to each non-empty set, and cannot be less preferred than any set.

For two preorders $\leq_p$ and $\leq_d$ and their liftings $\prec = \prec_p \cup \prec_d$, one defines a strict partial order $\prec$ on arguments, denoting the preference (ranking) on arguments using certain principles. We focus here on the weakest-link principle, by which one considers all defeasible elements of arguments in the comparison. An argument $B$ is strictly preferred to $A (A \prec B)$ whenever

- if $\text{DefRules}(A) = \text{DefRules}(B) = \emptyset$, then $\text{Prem}_d(A) \prec \text{Prem}_d(B)$;
- if $\text{Prem}_d(A) = \text{Prem}_d(B) = \emptyset$, then $\text{DefRules}(A) \prec \text{DefRules}(B)$;
- else $\text{Prem}_d(A) \prec \text{Prem}_d(B)$ and $\text{DefRules}(A) \prec \text{DefRules}(B)$.

Preferences are used to determine which attack succeed (and become defeats) and which do not. Defeats between arguments are defined as follows.

**Definition 13.** Given an AT $T = (\mathcal{L}, \mathcal{R}, n, -, K, \leq)$ and two arguments $A$ and $B$ in $T$, argument $A$ defeats argument $B$ iff $A$ successfully undercut, rebuts, or undermines $B$, where

- $A$ successfully undercut $B$ if $A$ undercut $B$;
- $A$ successfully rebuts $B$ (on $B'$) iff $A$ contrary rebuts $B$, or $A$ rebuts $B$ on $B'$ and $A \neq B'$; and
- $A$ successfully undermines $B$ (on $x$) iff $A$ contrary undermines $B$, or $A$ undermines $B$ on $x$ and $A \neq x$.

In other words, an undercut always succeeds, as do contrary rebuts and undermining attacks. For the contradictory variants of rebut and undermining (i.e., when conclusion of $A$ and the ordinary premise or conclusion of a defeasible rule in $B$ are contradictories of each other), the preference order $\prec$ on arguments decides whether the attack succeeds: if $A$ is strictly less preferred to the attacked subargument, the attack fails, otherwise it succeeds as a defeat.

**Example 9.** Let $T = (\mathcal{L}, \mathcal{R}, n, -, K, \leq)$ be an AT with

- $\mathcal{L} = \{a, b, c, w, x, y, z\}$,
- $\mathcal{K}_p = \{a, b, c\}, \mathcal{K}_n = \emptyset$,
- $\mathcal{R}_d = \{(r_1 : a \Rightarrow y), (r_2 : b \Rightarrow x)\}$,
- $\mathcal{R}_s = \{(y \rightarrow z), (c \rightarrow w)\}$,
- $\bar{x} = \{z\}, \bar{z} = \{x\}, \bar{c} = \{y\}, n(r_1) = \{w\}$
We give the names \( r_i \) for the defeasible rules before the rule. Moreover, let \( a \leq_p b \) and \( r_1 \leq_d r_2 \). The AT is shown in Figure 2.2 (left) with the arguments it gives rise to. Defeasible premises and rules are marked with dashed lines. The arguments and their defeat relation are shown on the right side of this figure. It holds that \( A_4 \) undercut both \( A_5 \) and \( A_7 \) (on \( A_5 \)), \( A_5 \) contrary undermines both \( A_1 \) and \( A_4 \) (on \( c \)), and \( A_7 \) contradictory rebuts \( A_6 \). Since \( x \) and \( z \) are contradictory to each other, but \( A_7 \) concludes \( z \) via a strict rule, it is the case that \( A_6 \) does not contradictory rebut \( A_7 \). All attacks except the last one are defeats. The last one fails because \( A_7 \prec A_6 \) (\( \text{Prem}_d(A_7) = \{a\} \prec_p \{b\} = \text{Prem}_d(A_6) \) due to \( a \prec_p b \) and \( \text{DefRules}(A_7) = \{r_1\} \prec_d \{r_2\} = \text{DefRules}(A_6) \) due to \( r_1 \prec_d r_2 \)). The unsuccessful contradictory rebut is denoted as a dotted arrow in the AF.

Conditions which ensure the satisfaction of rationality properties in ASPIC\(^+\) have been investigated [37, 122]. It turns out that these conditions are useful for our computational approach, as well. In particular, we assume some of these conditions to hold for our approach when preferences are included (if there are no preferences, our computational results hold regardless of the rationality conditions). These conditions constrain the set of strict rules in an AT, and we make use of three such conditions here, borrowing from Modgil and Prakken [122].

A set of strict rules \( \mathcal{R}_s \) is said to be closed under transposition if for each \( a_1, \ldots, a_n \rightarrow b \in \mathcal{R}_s \) it holds that for each \( i, 1 \leq i \leq n \), we have \( a_1, \ldots, a_{i-1}, b', a_{i+1}, \ldots, a_n \rightarrow a'_i \in \mathcal{R}_s \) for all contradictories \(-b' = b\) and \(-a'_i = a_i\), and at least one contradictory of each exists. An AT \( T \) is strict-consistent if there are no arguments \( A, B \) with \( \text{defPart}(A) = \text{defPart}(B) = \emptyset \) s.t. \( \text{Conc}(A) \) is a contrary of \( \text{Conc}(B) \), or \( \text{Conc}(A) \) and \( \text{Conc}(B) \) are contradictory to each other.

**Definition 14.** We say an AT \( T \) is well-formed if

- \( \mathcal{R}_s \) is closed under transposition,
- \( T \) is strict-consistent, and
• if $x$ is a contrary of $y$ then $y \notin \mathcal{K}_n$ and $y \neq \text{head}(r)$ for all $r \in \mathcal{R}_s$.

For intuition, these conditions aim to avoid certain inconsistencies that may arise. For example, strict-consistency is quite immediate: if an AT is not strict-consistent then there are arguments composed only of axioms and strict rules concluding contrary or contradictory atoms. For details we refer the reader to the work of Modgil and Prakken [122]. We remark that we are using a subset of the conditions of “well-defined” ATs as defined by Modgil and Prakken [122].

The semantics of ATs are defined via a translation to abstract argumentation frameworks (AFs) [70]. An AF is a pair $F = (A, D)$ of a set of (abstract) arguments $A$ and defeats $D \subseteq A \times A$ between arguments. If $(A, B) \in D$ we say that $A$ defeats $B$. Similarly, $S \subseteq A$ defeats $B \in A$ if there is an $A \in S$ with $(A, B) \in D$. We say that $S$ defends an argument $A$ if for each $B \in A$ such that $(B, A) \in D$, there is a $C \in S$ such that $(C, B) \in D$. We consider admissible, complete, stable and preferred extensions, with the corresponding functions $\sigma \in \{\text{adm}, \text{com}, \text{stb}, \text{prf}\}$. A semantics $\sigma(F) \subseteq 2^A$ returns a set of extensions. An extension under a semantics $\sigma$ is a $\sigma$-extension for short.

**Definition 15.** Given an AF $F = (A, D)$, a set $E \subseteq A$ is conflict-free (in $F$) if there are no $A, B$ in $E$ such that $(A, B) \in D$. For a conflict-free $E$, it holds that

- $E$ is admissible if each $A \in E$ is defended by $E$;
- $E$ is complete if $E \in \text{adm}(F)$ and each $A$ defended by $E$ is in $E$;
- $E$ is preferred if $E \in \text{adm}(F)$ and there is no $T \in \text{adm}(F)$ with $E \subset T$; and
- $E$ is stable if $E$ defeats each argument in $A \setminus E$.

ATs can be translated to AFs as follows. Let $T = (\mathcal{L}, \mathcal{R}, n, \mathcal{K}, \leq)$ be an AT. An AF $F = (A, D)$ corresponds to $T$ if $A$ is the set of all arguments in $T$ and $D$ the defeat relation based on $T$. Credulous and skeptical acceptance (or justification) are central computational problems in ASPIC$^+$. They are defined with reference to conclusions of the AF corresponding to a given argumentation theory. Given an argumentation theory $T$ and the corresponding AF $F$, an atom $x \in \mathcal{L}$ is

- skeptically justified under semantics $\sigma$ in $T$ if there is an argument concluding $x$ in every $\sigma$-extension in $F$, and
• credulously justified under semantics $\sigma$ in $T$ if there is a $\sigma$-extension containing an argument concluding $x$ in $F$.

Example 10. Continuing Example 9, the AF corresponding to the AT is shown in Figure 2.2 (right). There are two stable extensions: $E_1 = \{A_1, A_2, A_3, A_4, A_6\}$ and $E_2 = \{A_2, A_3, A_5, A_6, A_7\}$. Since $\text{conc}(A_4) = w$, $A_4 \in E_1$, and there is no argument in $E_2$ with conclusion $w$, it holds that $w$ is credulously but not skeptically justified under stable semantics.

2.3 Computational Complexity

We briefly recall the computational complexity classes relevant for this thesis. We assume familiarity with basic concepts of complexity theory, such as reductions and Turing machines [133]. The class P contains decision problems that can be solved with a deterministic Turing machine whose running time is polynomially bounded with respect to the size of the input. The class NP contains problems that can be decided in polynomial time with a non-deterministic Turing machine. The class coNP contains problems whose complement is in NP. For a given complexity class $C$, a $C$-hard problem is at least as hard as any problem in $C$. More precisely, if every problem in $C$ can be reduced in polynomial time to a problem $P$, then $P$ is $C$-hard. A problem is said to be $C$-complete if it is $C$-hard and also contained in $C$. A $C$-oracle is a procedure that decides a $C$-complete problem in constant time. Complexity classes higher in the polynomial hierarchy can be recursively defined based on oracles. The classes NP and coNP are on the first level of the hierarchy. Problems that belong to the first level when assuming access to an NP-oracle are on the second level of the hierarchy. In particular, a decision problem is in $\Sigma^P_2$ if the problem can be decided via a non-deterministic polynomial time algorithm that has access to an NP-oracle. The class $\Pi^P_2$ contains problems whose complement is in $\Sigma^P_2$. The classes $\Delta^P_2$ and $\Theta^P_2$ contain problems that can be decided in polynomial time by a deterministic algorithm that has access to an NP-oracle, with the latter class allowing at most logarithmic number of oracle calls.

The complexity of deciding credulous and skeptical acceptance of claims in LP- and DL-ABA is well understood [58, 69, 77, 78, 105]. Table 2.2 shows the complexity of acceptance problems in LP-ABA and Table 2.3 shows the results for DL-ABA, to the extent relevant for this thesis.
2.4 Propositional Satisfiability

The propositional satisfiability problem (SAT) [28] is the problem of deciding whether a given formula in propositional logic is satisfiable. A formula is satisfiable if there is a truth assignment to the variables of the formula so that the formula evaluates to true. SAT solvers restrict instances to be in conjunctive normal form (CNF), but any formula in propositional logic can be transformed into a CNF with linear overhead in a standard way [164]. A CNF formula is a conjunction of clauses, which in turn comprise of a disjunction of literals. A literal $l$ is either a variable $x$ or the negation of a variable $\neg x$. A truth assignment sets each variable to either true or false. A truth assignment satisfies a clause if it satisfies at least one literal in the clause. The whole formula is satisfied by a truth assignment if each clause in the formula is satisfied.

Propositional formulas are referred to as SAT instances and a computer program that decides if a given instance is satisfiable is referred to as a SAT solver. Solvers also typically report a model if an instance is satisfiable. Modern SAT solvers are typically based on the conflict-driven clause learning (CDCL) algorithm [13, 27, 119, 149, 151]. Notably, when an assignment is detected to lead to a conflict, CDCL adds information to the instance.
to rule out the assignment, possibly along with other related conflicting assignments. The solvers use various additional techniques to guide the search towards a satisfying assignment [28]. Advances to solver technology has increased the practical efficiency of SAT in recent years [88].

Since SAT is NP-complete, more than one call to a SAT solver is needed to solve problems that are harder than NP (assuming that $P \neq NP$ and using a polynomial-size encoding of the problem). To increase the practical efficiency of such algorithms, SAT solvers are able to use information learned in previous calls to speed up solving in successive calls on related problem instances. This is referred to as incrementality [81].

2.5 Answer Set Programming

We recall the language of normal logic programs under answer set semantics [34, 96, 98]. A (normal) answer set program $\pi$ consists of rules $r$ of the form

$$h \leftarrow b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m.$$  

where $h$ (the head of the rule) and each $b_i$ (constituting the body of the rule) are atoms. An atom $b_i$ is of the form $p(t_1, \ldots, t_n)$ with $p$ a predicate of arity $n$, and each $t_j$ either a constant or a variable. A literal is an atom ($a$) or the default negation of an atom ($\text{not } a$). An answer set program, a rule, and an atom, respectively, is ground if it is free of variables. We will use the convention that the first letter of a variable is uppercase and that of constants is lowercase.

A rule is positive if $k = m$ (i.e., the body contains no negated atoms), a fact if $m = 0$ (i.e., the body is empty). A fact is shortened to “$h$” by omitting $\leftarrow$. A constraint $\leftarrow b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m$ is a shorthand for $x \leftarrow b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m, \text{ not } x$, where $x$ is a fresh ground atom not occurring anywhere else in the program. We do not make use of functions within ASP programs. We use an additional syntactic construct called conditional literals [96] for succinctly representing rules with variable lengths of bodies in our ASP programs. The conditional $p(X) : q(r, X)$ yields $p(x)$ for each $x$ for which $q(r, x)$ hold.

For a program containing variables, let $GP$ be the set of rules obtained by applying all possible substitutions from the variables to the set of constants appearing in the program. An interpretation $I$, i.e., a subset of all the ground atoms, satisfies a positive rule $r = h \leftarrow b_1, \ldots, b_k$ if the presence of all positive body elements $b_1, \ldots, b_k$ in $I$ implies that the head atom is in $I$, i.e., $\{b_1, \ldots, b_k\} \subseteq I$ implies $h \in I$. For a program $\pi$ consisting only of positive rules, let $Cl(\pi)$ be the uniquely determined interpretation
I that satisfies all rules in \( \pi \) and no subset of \( I \) satisfies all rules in \( \pi \).

Interpretation \( I \) is an answer set of a ground program \( \pi \) if \( I = Cl(\pi^I) \) where \( \pi^I = \{ (h ← b_1, \ldots, b_k) \mid (h ← b_1, \ldots, b_k, not b_{k+1}, \ldots, not b_m) ∈ \pi, \{b_{k+1}, \ldots, b_m\} ∩ I = ∅ \} \) is the reduct. An interpretation \( I \) is an answer set of a non-ground program \( \pi \) if \( I \) is an answer set of \( GP \) of \( \pi \). A program is satisfiable if the program has at least one answer set and unsatisfiable otherwise.

Similarly to SAT, modern ASP solvers are typically based on CDCL [3, 101]. Recently, ASP solvers have introduced incremental solving as well [99], a feature we utilize in our algorithms for (presumably) beyond NP problems. Another method for solving optimization problems (such as finding preferred sets in argumentation) is answer set optimization [33]. In particular, we use ASPRIN [33], an optimization system built on top of an ASP solver that reports answer sets that are for example \( \subseteq \)-maximal with respect to a specified predicate. That is, when maximizing for the inclusion of arity one predicate \( p \), \( I \) is an optimal answer set if there is no answer set \( J \) such that \( \{ p(t) \mid p(t) ∈ I \} \subset \{ p(t) \mid p(t) ∈ J \} \).
Chapter 3

Reasoning in Assumption-based Argumentation

We begin the overview of the main contributions of this thesis with our work on variants of assumption-based argumentation (ABA), published in Articles I, II and III. We focus on three variants of ABA: the logic programming and default logic instantiations (LP-ABA and DL-ABA) and LP-ABA\(^+\). Our contributions consist of efficient algorithms for these versions of ABA, as well as central complexity results for LP-ABA\(^+\). A key idea in our approaches to all of the instantiations is deciding acceptance of claims directly based on acceptable sets of assumptions and not generating arguments explicitly. We develop algorithms for deciding credulous and skeptical acceptance based on ASP for LP-ABA and LP-ABA\(^+\) and conduct empirical evaluations of our approach, suggesting significantly better performance than existing implementations. We also prove complexity results for reasoning in LP-ABA\(^+\) under \(<\)-admissible, \(<\)-complete, \(<\)-grounded and \(<\)-stable semantics. For deciding acceptance of claims in DL-ABA, we develop algorithms based on incremental SAT solving.

3.1 Logic Programming Instantiation

We start by giving an overview of existing algorithmic approaches to deciding credulous and skeptical acceptance in LP-ABA. A notable approach is dispute derivations, a dialectic process for determining the credulous acceptability of a given sentence under certain semantics [52, 54, 71, 161]. The most recent dispute derivation implementation is ABAGRAPh [52], written in Prolog, improving on the earlier implementations GRAPHARG [54], PROXDD [161] and CASAPI [89]. ABAGRAPh can reason credulously un-
der admissible and grounded semantics. A recent development is flexible dispute derivations, an extension of dispute derivations that uses forward derivations, generalizing the dispute derivation approach to stable semantics and enumeration of assumption sets under a given semantics [67, 68]. Some algorithms translate ABA frameworks to abstract argumentation frameworks, allowing the use of implementations for deciding acceptance in abstract argumentation. For LP-ABA, ABA2AF (by the present author) translates ABA frameworks to AFs and uses an AF solver on the obtained AF [115]. ABA2AF supports credulous and sceptical reasoning under admissible, stable and preferred semantics. TWEETYPROJECT, a Java library of various approaches to logical aspects of artificial intelligence and knowledge representation, also implements reasoning in LP-ABA [154, 155]. TWEETYPROJECT implements credulous and skeptical reasoning and \( \sigma \)-assumption set enumeration under admissible, complete, stable, preferred, ideal and grounded semantics. TWEETYPROJECT offers two approaches. One is a translation from an ABA framework to an AF and using an AF solver for the problem. The other is to generate subsets of the assumptions and to check the satisfaction of the given semantics as given in the ABA definitions. Lastly, ABAPLUS [14], an implementation for reasoning in LP-ABA, can also be used to reason in LP-ABA. ABAPLUS translates an ABA framework to an AF and employs an AF solver, and supports enumeration of assumption sets under complete, preferred, stable grounded, and ideal semantics.

We identify key insights for LP-ABA that allow us to encode many central LP-ABA reasoning tasks in ASP. Namely, we show how to encode both the derivation of atoms from a set of assumptions and the conditions of admissible, complete and stable semantics. This lends itself to normal ASP programs that non-deterministically guess a set of assumptions and check whether this set is an admissible (or complete or stable) assumption set. To decide credulous or sceptical acceptance, the programs can additionally check whether or not the obtained assumption set derives a given query. Our encodings can also be used to enumerate \( \sigma \)-assumption sets under all of the considered semantics. Further, we encode finding the grounded assumption set in ASP, and two approaches for answering sceptical acceptance under preferred semantics (using ASP optimization and a counterexample-guided abstraction refinement algorithm). We also implement an algorithm for finding the ideal assumption set using our ASP encoding for admissible semantics.

\[1\] The work on flexible dispute derivations was published after the writing of Articles I and II.
ASP is particularly well-suited for problems in LP-ABA, which include checking derivations from sets of assumptions with the logic programming-like rules of LP-ABA. ASP implements the concept of negation as failure, meaning that a fact is assumed to not hold if it can not be shown to hold. Derivability can be encoded in ASP by simply asserting that if the precedents hold, the consequent holds as well. This is in contrast to classical implication: if derivation rules \( p \leftarrow q \) and \( q \leftarrow p \) are viewed as implications, one would infer the derivability of \( p \) and \( q \) even if neither \( p \) nor \( q \) are assumptions or derivable in other ways.

### 3.1.1 Encoding ABA in ASP

We begin the discussion of the ASP encodings we develop by explaining how to represent an ABA framework in ASP and encoding the credulous and skeptical acceptance, uniformly for all semantics. Given an ABA framework \( F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \neg) \) with \( \mathcal{R} = \{r_1, \ldots, r_n\} \), we represent the framework \( F \) in ASP as the facts

\[
\text{ABA}(F) = \{\text{assumption}(a). \mid a \in \mathcal{A}\} \cup \\
\{\text{head}(i, b). \mid r_i \in \mathcal{R}, b = \text{head}(r_i)\} \cup \\
\{\text{body}(i, b). \mid r_i \in \mathcal{R}, b \in \text{body}(r_i)\} \cup \\
\{\text{contrary}(a, b). \mid b = \overline{a}, a \in \mathcal{A}\}.
\]

It is indicated with \( \text{assumption}(a) \) that \( a \) is an assumption, whereas \( \text{contrary}(a, b) \) indicates that \( b \) is the contrary of \( a \). The rules are expressed separately for heads and bodies of rules, with a unique rule index linking them. In particular, \( \text{head}(i, b) \) is interpreted as \( b \) being the head of the rule with index \( i \). Similarly, \( \text{body}(i, b) \) indicates that \( b \) is contained in the body of the rule with index \( i \).

**Example 11.** Consider the ABA framework \( F \) with

- sentences \( \mathcal{L} = \{a, b, x, y\} \),
- assumptions \( \mathcal{A} = \{a, b\} \),
- contraries \( \overline{a} = y, \overline{b} = x \),
- rules \( \mathcal{R} = \{(x \leftarrow a, y), (y \leftarrow b)\} \).
F is represented in ASP as the set of facts

\[ \text{ABA}(F) = \{ \text{assumption}(a). \text{assumption}(b). \]
\[ \text{head}(1, x). \text{body}(1, a). \text{body}(1, y). \]
\[ \text{head}(2, y). \text{body}(2, b). \]
\[ \text{contrary}(a, y). \text{contrary}(b, x). \} \]

As we focus on acceptance problems, we append a fact for a query to a program to represent the sentence whose acceptance status is queried. A queried sentence \( s \in \mathcal{L} \) is specified by adding to the program the fact

\[ \text{query}(s). \]

Then credulous or skeptical queries are computed by adding one of the following constraints to the program. The constraint

\[ \leftarrow \text{not supported}(X), \text{query}(X). \]

rules out answers where the query is not derived from an assumption set, corresponding to credulous acceptance. As explained later, \text{supported} indicates atoms that are supported by the assumptions identified by an answer set. To capture skeptical reasoning, the constraint

\[ \leftarrow \text{supported}(X), \text{query}(X). \]

enforces a counterexample check for skeptical acceptance. In particular, the resulting answer set program does not have any answer sets if and only if the queried sentence is skeptically accepted.

### 3.1.2 ASP Encodings for Admissible, Complete, Stable and Grounded Semantics

We detail here our ASP encodings for stable, admissible and complete semantics which we introduced in Article I. The encoding \( \pi_{\text{stb}} \), given in Listing 3.1, gives the full encoding for finding stable assumption sets. In Lines 1 and 2, an assumption set is guessed (identified by \text{in}). Lines 3–5 encode all atoms derivable from this set. Line 6 determines which assumptions are attacked (or defeated) by the guessed assumption set and Line 7 enforces that the guessed assumptions set does not attack an assumption contained in itself, corresponding to conflict-freeness. Finally, Line 8 enforces stability: all assumptions outside the guess must be attacked.

Admissible and complete assumption sets can be answered with similar programs, with changes replacing Line 8 of \( \pi_{\text{stb}} \). In particular, the rules
3.1 Logic Programming Instantiation

Listing 3.1: Module $\pi_{stb}$.

\begin{verbatim}
1 in(X) ← assumption(X), not out(X).
2 out(X) ← assumption(X), not in(X).
3 supported(X) ← assumption(X), in(X).
4 applicable_by_in(R) ← head(R,X), applicable_by_in(R).
5 applicable_by_in(R) ← supported(X) : body(R,X).
6 defeated(X) ← supported(Y), contrary(X,Y).
7 ← in(X), defeated(X).
8 ← out(X), not defeated(X).
\end{verbatim}

given in $\Delta_{adm}$ (Listing 3.2) replace Line 8 in the encoding for admissible semantics. The constraint on Line 5 enforces that the obtained set of assumptions defends itself. The encoding for complete semantics can be obtained by adding the following constraint to the encoding of admissible semantics.

\begin{verbatim}
9 ← out(X), not attacked_by_undefeated(X).
\end{verbatim}

This rule enforces that every assumption outside the set is attacked by assumptions that are not attacked by the set. If an assumption violates this condition, the assumption is defended by the guessed assumption set.

We provide in Article I an encoding to compute the grounded assumption set as the least fixed point of the defence operator [30], which yields the assumptions that are defended by a given set of assumptions. The encoding uses techniques similar to the ones we presented in this section to compute the defence operator iteratively. Appropriately for the complexity, the grounded program does not make a non-deterministic guess of the assumptions.
3.1.3 Algorithms for Preferred and Ideal Semantics

We also develop an algorithm for skeptical acceptance under preferred semantics and implement an algorithm for acceptance under ideal semantics in Article I. In particular, skeptical acceptance under preferred semantics is \( \Pi^P_2 \)-complete and acceptance under ideal semantics is in \( \Theta^P_2 \). The algorithm for preferred semantics uses ASPRIN [33], an optimization system for ASP. A preferred assumption set can be found by maximizing an admissible (or complete) assumption set with respect to set inclusion. This can be done via ASPRIN by adding the following lines to the encoding for admissible (or complete) semantics:

\[
\#\text{preference}(p1, \text{superset})\{\text{in}(X) : \text{assumption}(X)\}.
\]

\[
\#\text{optimize}(p1).
\]

Skeptical reasoning can be answered by using the query mode of ASPRIN.

In Article II, we investigate speeding up the computation of acceptance under preferred semantics using recent advances in incremental answer set solving [99]. We propose a counterexample-guided abstraction refinement (CEGAR) algorithm [49, 50], shown successful for many problems on the second level of the polynomial hierarchy, including problems in argumentation [43, 80, 127, 156, 157]. To speed up successive calls, we use an ASP solver incrementally [99].

In a CEGAR algorithm, an abstraction of the problem is encoded in the chosen language (ASP in our case). An abstraction is an overapproximation: all solutions to the original problem are also solutions of the abstraction, but not (necessarily) vice versa. Given this abstraction, the following steps are iterated. First, a solver call obtains a solution to the abstraction, also called a candidate. Then, another solver call checks whether the candidate is a solution to the original problem by checking for the existence of a counterexample. If a counterexample does not exist, the candidate is a solution to the original problem. Otherwise, the counterexample is ruled out from future consideration by adding a constraint to the abstraction (also called refining the abstraction) and the first step is repeated. For many problems, it is possible to refine by more than just the specific counterexample, such as all subsets of the counterexample. If no candidate turns out to be a solution and no more candidates satisfy the abstraction after the refinements, it can be concluded that there is no solution to the original problem.

We define some notation to simplify the presentation of our CEGAR algorithm for skeptical acceptance under preferred semantics. Let \( I \) be an answer set of a given program, \( p \) an ASP predicate of arity one, and \( M = \)
\{l_1, \ldots, l_n\} a set of ASP literals. Then we define \[ p(I) = \{ p(x) \mid p(x) \in I \}, \]
and \[ \text{constr}(M) = \leftarrow l_1, \ldots, l_n. \]
Intuitively, \( p(I) \) corresponds to the atoms that satisfy the predicate \( p \) in a given interpretation \( I \), and \( \text{constr}(M) \) is a constraint stating that at least one of the ASP literals of \( M \) must be absent in an answer set.

Our incremental algorithm for skeptical acceptance under preferred semantics is shown in Algorithm 1. The algorithm searches for a preferred assumption set that does not contain the query, which is a counterexample to the query being skeptically accepted, as follows:

1. The algorithm uses the complete semantics as an abstraction, because all preferred assumption sets are complete. Additionally, as the algorithm is looking for a counterexample to the query being skeptically accepted, a candidate shall not derive the query. The candidate generation is performed on Line 2 of Algorithm 1.

2. This set is further maximized (with respect to set inclusion) while maintaining completeness and not deriving the query (Lines 5–7). For convenience, let us refer to the obtained maximal set of assumptions as \( M \).

3. A counterexample to the query being skeptically accepted is a proper superset of \( M \) that is complete and derives the query (Line 8; note that the constraint restricting the query from being derived is not present). If there is no such superset, then \( M \) is indeed a preferred assumption set. Further, by construction, \( M \) does not derive the query and thus constitutes a counterexample to the query being skeptically accepted.

4. If, on the other hand, a superset of \( M \) is complete, the search is continued from step 1. All subsets of \( M \) are ruled out of the search to limit the search space, by leaving the subset-excluding constraints in the program (Line 7). This can be done because all of the subsets have now been shown to not be preferred.

Finally, we implement the ideal algorithm introduced by Dung et al. [73] using our encoding for admissible semantics. The algorithm starts with an overapproximation of an ideal set consisting of all assumptions that are credulously accepted under admissible semantics. The cautious mode of CLINGO [99] gives the intersection of all answer sets of a program. Thus we can compute this overapproximation of an ideal set at the start of the algorithm via collecting the cautiously out assumptions with our encoding for admissibility. The rest of the algorithm runs in polynomial time.
Algorithm 1 Skeptical acceptance under preferred semantics

Require: ABA framework $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \neg)$, $s \in \mathcal{L}$
Ensure: return YES if $s$ is skeptically accepted under preferred semantics in $F$, NO otherwise

1: $\pi := \text{ABA}(F) \cup \pi_{com}$
2: while $\pi \cup \{\text{constr}(\text{supported}(s))\}$ is satisfiable do
3: Let $I$ be the found answer set
4: $\pi := \pi \cup \{\text{constr}(\text{out}(I))\}$
5: while $\pi \cup \{\text{constr}(\text{supported}(s))\} \cup \text{in}(I)$ is satisfiable do
6: Let $I$ be the found answer set
7: $\pi := \pi \cup \{\text{constr}(\text{out}(I))\}$
8: if $\pi \cup \text{in}(I)$ is unsatisfiable then return NO
9: return YES

3.1.4 Empirical Evaluation

The incremental algorithms we proposed are implemented in Python. They, along with the proposed encodings, are available in open source at https://bitbucket.org/coreo-group/aspforaba.

We compare our approach against previous state-of-the-art algorithms. These are ABAGRAPH and ABA2AF for admissible semantics, and the former for grounded semantics and the latter for stable semantics, and ABAPLUS for complete and ideal semantics. We use a benchmark set from earlier related experimental work [52, 115] for admissible, stable, grounded and preferred semantics. Due to the benchmarks not satisfying input restrictions of ABAPLUS, we also generated new benchmarks with similar parameters for comparison under complete and ideal semantics. We use a timeout limit of 10 minutes. Table 3.1 shows the results of the comparison. Our novel ASP approach outperforms the other approaches for all of the reasoning problems by orders of magnitude.

Since the ASP approach has no timeouts on these instances (which have less than 100 atoms per instance), we generated larger benchmarks to test the scalability of our approach. The results are summarized in Table 3.2. The ASP approach scales much higher: instances with up to 2000 atoms (over 20 times more than in the benchmark set used for comparison against other approach) are solved relatively quickly under most semantics.

The results for preferred semantics in Table 3.1 are for our algorithm using ASPRIN. Figure 3.1 shows the difference between ASPRIN and the incremental algorithm we present in Article II for skeptical acceptance under preferred semantics. The incremental approach outperforms the approach...
Table 3.1: Runtime comparison. Mean (mean), median (med.) and cumulative running times (cum.) over solved instances. #timeout is the number of timeouts. Number of instances: 1728 (adm, grd), 4613 (stb), 680 (prf), 120 (com, ideal).

<table>
<thead>
<tr>
<th>Problem</th>
<th>Approach</th>
<th>#timeout</th>
<th>Running times (s)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>mean</td>
<td>med.</td>
<td>cum.</td>
<td></td>
</tr>
<tr>
<td>ABA adm</td>
<td>ASP</td>
<td>0</td>
<td>0.018</td>
<td>0.012</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABAGRAPHS</td>
<td>200</td>
<td>8.464</td>
<td>1.056</td>
<td>12932</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABA2AF</td>
<td>364</td>
<td>13.990</td>
<td>0.572</td>
<td>19078</td>
<td></td>
</tr>
<tr>
<td>ABA stb</td>
<td>ASP</td>
<td>0</td>
<td>0.008</td>
<td>0.004</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABA2AF</td>
<td>648</td>
<td>10.942</td>
<td>1.040</td>
<td>43386</td>
<td></td>
</tr>
<tr>
<td>ABA grd</td>
<td>ASP</td>
<td>0</td>
<td>0.127</td>
<td>0.056</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABAGRAPHS</td>
<td>210</td>
<td>9.979</td>
<td>0.984</td>
<td>15148</td>
<td></td>
</tr>
<tr>
<td>ABA prf</td>
<td>ASP</td>
<td>0</td>
<td>0.333</td>
<td>0.328</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>enum. wo/query</td>
<td>ABA2AF</td>
<td>255</td>
<td>6.082</td>
<td>0.464</td>
<td>2585</td>
<td></td>
</tr>
<tr>
<td>ABA com</td>
<td>ASP</td>
<td>0</td>
<td>0.005</td>
<td>0.004</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>enum. wo/query</td>
<td>ABAPLUS</td>
<td>9</td>
<td>15.287</td>
<td>0.268</td>
<td>1697</td>
<td></td>
</tr>
<tr>
<td>ABA ideal</td>
<td>ASP</td>
<td>0</td>
<td>0.025</td>
<td>0.024</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>find the ideal set</td>
<td>ABAPLUS</td>
<td>18</td>
<td>22.490</td>
<td>0.322</td>
<td>2293</td>
<td></td>
</tr>
</tbody>
</table>

Based on ASPRIN. Note that enumeration mode is used for ASPRIN, as it was faster than the direct query mode on these benchmarks.

Another translation of ABA semantics into logic programming has been introduced, such that reasoning under stable semantics in LP-ABA corresponds to the ASP semantics [40]. We implement this translation of stable semantics in LP-ABA as ASP and show very similar performance to our encoding of stable semantics in Article I.

### 3.2 Logic Programming Instantiation with Preferences

We continue to LP-ABA+. The only previous algorithmic approach to LP-ABA+ that we are aware of is ABAPLUS [14]. ABAPLUS is a translation-based approach, translating LP-ABA+ frameworks to AFs, accounting for preferences, and solving the AFs with an AF solver. ABAPLUS supports σ-assumption set enumeration instead of credulous or sceptical reasoning. This means that to answer credulous or sceptical acceptance, ABAPLUS needs to enumerate all σ-assumption sets. ABAPLUS can be used to reason over regular ABA frameworks as well, because LP-ABA+ is a generalization
Table 3.2: Scalability of our ASP algorithms on larger frameworks. The considered problems are credulous acceptance for admissible, complete and stable semantics, preferred assumption set enumeration, and finding the ideal assumption set. Number of instances for each value of $|\mathcal{L}|$: 100 ($adm$, $com$, $stb$), 10 ($prf$, $ideal$).

| $|\mathcal{L}|$ | $adm$ | $com$ | $stb$ | $prf$ | $ideal$ |
|---|---|---|---|---|---|
| 50 | 0 (0) | 0 (0) | 0 (0) | 0 (0) | 0 (0) |
| 250 | 0 (0) | 0 (0) | 0 (0) | 0 (1) | 0 (1) |
| 500 | 0 (1) | 0 (1) | 0 (0) | 0 (3) | 0 (1) |
| 1000 | 0 (3) | 0 (3) | 0 (1) | 0 (9) | 0 (4) |
| 1500 | 0 (14) | 0 (12) | 0 (4) | 0 (32) | 0 (18) |
| 2000 | 0 (99) | 0 (75) | 0 (19) | 5 (145) | 0 (156) |
| 2500 | 22 (126) | 10 (201) | 0 (74) | 7 (269) | 4 (293) |
| 3000 | 70 (174) | 58 (223) | 18 (174) | 10 (0) | 9 (464) |
| 3500 | 85 (211) | 79 (254) | 48 (232) | 10 (0) | 10 (0) |
| 4000 | 89 (108) | 87 (135) | 81 (158) | 10 (0) | 10 (0) |

Figure 3.1: Runtime comparison of ASPIN (enumeration) and incremental ASP on skeptical acceptance under preferred semantics.
of ABA. \textsc{ABAplus} supports $\sigma$-assumption set enumeration under $<\text{-stable}$, $<\text{-grounded}$, $<\text{-complete}$, $<\text{-preferred}$ and $<\text{-ideal}$ semantics. \textsc{ABAplus} requires that a property called axiom of weak contraposition is satisfied by the LP-ABA$^+$ framework, which implies the satisfaction of the FL-property (Property 1) [56].

3.2.1 Complexity of Reasoning in LP-ABA$^+$

The computational complexity of acceptance problems in LP-ABA$^+$ has not been previously studied, to the best of our knowledge. We prove complexity results and develop algorithms for acceptance under $<\text{-stable}$, $<\text{-grounded}$, $<\text{-admissible}$ and $<\text{-complete}$ semantics.

In Article I we identify cases where computing the necessary $<\text{-attack}$ relations under different semantics is decidable in polynomial time and cases where it is not. Firstly, given assumption sets $A$ and $B$, it can be decided in polynomial time whether $A$ normally $<\text{-attacks}$ $B$, $A$ reversely $<\text{-attacks}$ $B$, or $A$ normally $<\text{-attacks}$ each subset $B' \subseteq B$ that $<\text{-attacks}$ $A$.

Only the first two $<\text{-attack}$ types are needed for verifying that a set of assumptions is $<\text{-stable}$, because a conflict-free set of assumptions $A$ is $<\text{-stable}$ if and only if it $<\text{-attacks}$ each singleton assumption outside of $A$. This implies that deciding credulous and skeptical acceptance under $<\text{-stable}$ semantics is NP-complete and coNP-complete, respectively (hardness follows from the hardness of the corresponding problems in ABA).

On the other hand, we show that checking whether $A$ reversely $<\text{-attacks}$ each subset $B' \subseteq B$ that $<\text{-attacks}$ $A$ is coNP-complete. Checking reverse counterattacks is needed to establish whether an assumption set defends itself. This suggests a higher complexity for deciding acceptance under $<\text{-admissible}$, $<\text{-complete}$ and $<\text{-grounded}$ semantics compared to the corresponding problems in ABA. We show that credulous acceptance under $<\text{-admissible}$ semantics is $\Sigma^P_2$-complete. Credulous acceptance under complete semantics is also included in $\Sigma^P_2$. For frameworks satisfying the FL-property (recall Property 1) finding the unique grounded assumption set is in $\Delta^P_2$ (that is, it can be computed with a polynomial number of calls to an NP-oracle). The complexity results for LP-ABA$^+$ that we proved in Articles I and II are summarized in Table 3.3. The membership results are proven based on our algorithmic approach.

3.2.2 Algorithms for Deciding Acceptance in LP-ABA$^+$

Based on our observations, we develop ASP encodings for credulous and skeptical acceptance under stable semantics, and incremental ASP-based al-
Table 3.3: Complexity of deciding acceptance of an atom in the logic programming fragment of LP-ABA$^+$. The results for $<$-grounded semantics hold for frameworks satisfying the FL-property (recall Property 1). We establish these results in Articles I and II.

<table>
<thead>
<tr>
<th>Semantics</th>
<th>Credulous acceptance</th>
<th>Skeptical acceptance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;$-stable</td>
<td>NP-complete</td>
<td>coNP-complete</td>
</tr>
<tr>
<td>$&lt;$-grounded</td>
<td>in $\Delta^P_2$ (FL)</td>
<td>in $\Delta^P_2$ (FL)</td>
</tr>
<tr>
<td>$&lt;$-admissible</td>
<td>$\Sigma^P_2$-complete</td>
<td>$\Sigma^P_2$</td>
</tr>
<tr>
<td>$&lt;$-complete</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Algorithms for credulous acceptance under admissible complete, and grounded semantics. First, we represent an LP-ABA$^+$ framework as follows in ASP, building on the ASP representation of an ABA framework. For a given LP-ABA$^+$ framework $F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, -, \leq)$, let

$$\text{ABA}^+(F) = \text{ABA}(F) \cup \{\text{preferred}(x, y) \mid y \leq x\}$$

be the facts representing the LP-ABA$^+$ framework.

**$<$-Stable semantics** Checking the satisfiability of a single normal ASP program suffices for acceptance under $<$-stable semantics in LP-ABA$^+$. The program computes stability via the observation that a set of assumptions $A$ is $<$-stable if and only if each assumption that is not normally $<$-attacked by $A$ is either in $A$ or reversely $<$-attacked by $A$. Thus, we introduce an encoding with the following components. First, a set of assumptions $A$ is non-deterministically guessed. Then the set of assumptions that are not normally $<$-attacked by $A$ are identified by checking if an $<$-attack towards a particular assumption $b$ can be constructed from those assumptions in $A$ that are not less preferred than $b$. Finally, it is enforced that each assumption that is not in $A$ and is not normally $<$-attacked by $A$ is reversely $<$-attacked by $A$. Reverse $<$-attack from $A$ to assumption $b$ is done by checking if from $b$ one can derive the contrary of some $a \in A$ such that $b < a$. If so, then $a$, and therefore $A$, reversely $<$-attacks $b$.

**$<$-Grounded semantics** We develop an algorithm for deciding the acceptance of a claim under $<$-grounded semantics in LP-ABA$^+$ frameworks that satisfy the FL-property (shown as Algorithm 2). In these frameworks the $<$-grounded assumption set equals the least fixed point of the defence operator. The algorithm iteratively adds to a set grounded the assumptions that are defended by grounded, starting with the empty set. Checking if an
Algorithm 2 Computing the $\langle -$grounded assumption set

Require: \( ABA \) framework \( F = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \neg) \) satisfying the FL-property

Ensure: return the $\langle -$grd-assumption set of \( F \)

1: \( \text{grounded} := \emptyset \)
2: \( \pi := ABA^+(F) \cup \pi_{\text{grd+ subroutine}} \)
3: \( \text{while} \ \text{change} = \text{true} \ \text{do} \)
4: \( \text{change} := \text{false} \)
5: \( S := \mathcal{A} \setminus \text{grounded} \)
6: \( \text{while} \ S \neq \emptyset \ \text{do} \)
7: \( \text{pick any } a \in S \)
8: \( \text{if } \pi \cup \{ \text{def}(x) \mid x \in \text{grounded} \} \cup \{ \text{target}(a) \} \text{ is unsat then} \)
9: \( \text{grounded} := \text{grounded} \cup \{ a \} \)
10: \( \text{change} := \text{true} \)
11: \( S := S \setminus \{ a \} \)
12: \( \text{return } \text{grounded} \)

assumption is $\langle -$defended by a set of assumptions requires the fourth type of $\langle -$attack mentioned before, and is thus NP-complete. Our algorithm encodes this check as an ASP program (Line 2) and uses it as a subroutine to compute the $\langle -$grounded assumption set (Line 8), using a polynomial number of calls to an ASP solver. If the check on Line 8 is unsatisfiable, then each $\langle -$attack to the “target” assumption is countered by \( \text{grounded} \) and thus the target is defended by \( \text{grounded} \) and can be added to \( \text{grounded} \) (Line 9).

$\langle -$Admissible and $\langle -$complete semantics

We develop CEGAR algorithms for credulous acceptance under admissible and complete semantics, making use of incremental answer set solving.

First, we identify properties that suggest an abstraction for admissible semantics, and a tighter abstraction for complete semantics. Refining the abstraction by as much as possible is desirable, as this reduces the search space and thereby, likely, the runtime of the algorithm. Given an assumption set \( A \), we define the set \( U \) to contain each assumption \( a \in \mathcal{A} \) that \( A \) does not $\langle -$attack.

**Proposition 3.1.** Given an LP-ABA\(^+\) framework \( F \), a conflict-free set of assumptions \( A \) in \( F \), let \( U \) contain each assumption that \( A \) does not $\langle -$attack. It holds that

- \( A \) is $\langle -$admissible iff there is no set \( B \subseteq U \) such that \( A \) does not $\langle -$attack \( B \) and \( B \) $\langle -$attacks \( A \), and
• \( A \) is \(<\text{-complete}\) iff \( A \) is \(<\text{-admissible}\) and for all \( \alpha \in A \setminus A \) it holds that \( \alpha \) is \(<\text{-attacked}\) by a \( B \subseteq U \) such that \( A \) does not \(<\text{-attack}\) \( B \).

Our abstraction for \(<\text{-admissible}\) semantics is that a candidate assumption set is conflict-free and derives the queried atom. With the candidate generation, each assumption that is individually not \(<\text{-attacked}\) by the candidate are identified (the set \( U \), for “undefeated”). Then, it is checked with another solver call if there is a counterexample, following the first item of Proposition 3.1. Namely, if there is a \( B \subseteq U \) that the candidate does not \(<\text{-attack}\) but \( B \) \(<\text{-attacks}\) the candidate, then the candidate is not a solution. If such a \( B \) does not exist, the candidate is an \(<\text{-admissible}\) assumption set. Otherwise, the candidate is ruled out from further consideration and another candidate is searched for.

For \(<\text{-complete}\) semantics, we are able to restrict the search space for the candidate generation more, based on the second item of Proposition 3.1. In particular, we only consider an assumption set as a candidate if the following holds, in addition to conflict-freeness and deriving the queried atom. Let \( U \) be the set of assumptions that the candidate assumption set \( A \) does not individually \(<\text{-attack}\). Then for each \( a \in U \), either \( a \in A \) or \( U \) \(<\text{-attacks}\) \( a \). This restriction only rules out assumption sets that are not \(<\text{-complete}\), by the second item of Proposition 3.1. Note that constructing \( U \) and checking the individual attacks from \( U \) can be done in polynomial time and can thus be added as restrictions to the candidate generation. Finally, given a candidate, in addition to checking that it is \(<\text{-admissible}\), it is checked that the candidate does not defend any assumptions outside the candidate.

### 3.2.3 Empirical Evaluation

We compare our algorithms against the state-of-the-art implementation for reasoning in LP-ABA\(^+\), ABAPLUS [14]. As ABAPLUS requires that input frameworks satisfy the WCP property [14, 56], we let ABAPLUS modify input frameworks to satisfy WCP and use the resulting frameworks for our experiments. Due to computational limitations, the frameworks we are able to obtain in this manner are quite small, having up to 30 sentences per framework. We use a time limit of 10 minutes.

Our implementation is available in open source at https://bitbucket.org/coreo-group/aspforaba. The results are summarized in Table 3.4 and show that our ASP-based algorithms clearly outperform ABAPLUS. The ASP algorithms have no timeouts for these instances, while ABAPLUS times out on 9 instances for each problem (namely, on all but one of the
Table 3.4: Runtime comparison for LP-ABA$^+$ problems between the ASP-based algorithms we introduce and the previous state-of-the-art solver, ABAPLUS. Runtimes are over solved instances.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Approach</th>
<th>#timeouts</th>
<th>Running times (s)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;-stb</td>
<td>ASP</td>
<td>0</td>
<td>0.018 0.008 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>enum. wo/query</td>
<td>ABAPLUS</td>
<td>9</td>
<td>15.583 0.268 1729</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;-grd</td>
<td>ASP</td>
<td>0</td>
<td>0.380 0.168 46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABAPLUS</td>
<td>9</td>
<td>15.611 0.268 1732</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;-adm</td>
<td>ASP</td>
<td>0</td>
<td>0.114 0.040 14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>enumeration</td>
<td>ABAPLUS</td>
<td>9</td>
<td>15.442 0.560 1714</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;-com</td>
<td>ASP</td>
<td>0</td>
<td>0.096 0.040 12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>enumeration</td>
<td>ABAPLUS</td>
<td>9</td>
<td>14.240 0.550 1581</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.2: Runtime comparison of the incremental ASP approach using the weaker and stronger abstraction for finding a <-complete assumption set under.

largest instances). Figure 3.2 shows the effect of the more restrictive abstraction for complete semantics. Runtimes are vastly and almost ubiquitously improved with the stronger abstraction, fitting the intuition that reducing the search space early, namely in the candidate generation, can reduce the overall runtime significantly.
3.3 Default Logic Instantiation

In Article III, we propose algorithms for a more expressive instantiation of ABA, namely the propositional default logic instantiation (DL-ABA). We show that obtaining derivable sentences from a set of assumptions in DL-ABA amounts to propositional entailment checks. Thus SAT is a natural choice for algorithms for DL-ABA. As the complexity of reasoning under many central semantics in DL-ABA is (presumably) beyond NP [58, 69, 105], we develop CEGAR algorithms making use of incremental SAT solving. To the best of our knowledge, no algorithms for deciding acceptance problems in DL-ABA have been developed before. The stable semantics of DL-ABA correspond to the standard semantics of DL and there are implementations of reasoning in DL [46, 48, 124, 148].

3.3.1 SAT-based Algorithms for Reasoning in DL-ABA

We show in Article III how to capture derivability from a set of assumptions in DL-ABA using propositional entailment checks. In particular, given a set of assumptions $A \subseteq A$, one can check derivability of sentences from $A$ by first iteratively collecting the heads of each default rule that is applicable from $A$. A default rule is applicable if the assumptions in the rule body are contained in $A$ and the rest of the rule body is entailed by $W$ joined with the previously collected heads of applicable default rules. A sentence $x$ is derived from $A$ if and only if the background formula $W$ together with the heads of applicable default rules entail $x$.

This correspondence forms the theoretical underpinning of our algorithms. We propose algorithms for computing assumption sets under stable, admissible, complete and grounded semantics. The first three are CEGAR algorithms while the algorithm for grounded semantics computes the least fixed point of the defence operator. We employ incremental SAT solver calls to speed up solving. We discuss the CEGAR algorithms for credulous acceptance (Algorithm 3) with stable semantics as an example.

Candidate generation  A candidate solution is a set of assumptions. At first, there are no restrictions on the set, reflected by the empty propositional formula $\phi$ over the assumptions (Line 1 of Algorithm 3). A candidate $A$ is obtained with a SAT solver call (Line 2). Then the conclusions of default rules that are usable from the candidate are collected to $C$ with a fixed point algorithm following our observation above (Line 3). In particular, we start with $C = \emptyset$. For each rule that is not contained in $C$ but

\[ \text{We were unable to obtain a working version.} \]
whose assumptions are a subset of $A$ it is checked if the prerequisite of the rule is entailed by $W \cup C$. If so, the conclusion of the rule is added to $C$. The process continues until a fixed point is reached.

**Counterexamples** There are multiple cases invalidating the candidate: the query is not derived, the candidate is self-conflicting, or the semantic criteria are otherwise violated (Line 4). Checking if these conditions for validity hold for a candidate can be done as follows:

- Given $C$ and a query $q$, checking if a $q$ is entailed by $A$ amounts to checking if $W \cup C$ entail the query.

- Given $A$ and $C$, conflict-freeness can be verified by checking that the contrary of any $a \in A$ is not entailed by $W \cup C$. This can be done by an iterative procedure that removes elements from a set $X$, initialized as $X := A$. On each step it is checked with a SAT call if there is a model to $W \cup C$ that sets any $x$ with $Mx \in X$ to true. If so, $\neg x$ is not entailed by $W \cup C$ and thus $Mx$ is not attacked by $A$. Each such $Mx$ can be removed from $X$. If this procedure removes all assumptions from $X$, $A$ is conflict-free. Otherwise, if no more assumptions can be removed while $X$ is non-empty, $A$ is self-conflicting.

- Lastly, given (conflict-free) $A$ and $C$, checking the stability of $A$ amounts to checking that $W \cup C$ entails the negation of all assumptions $Mb /\notin A$. This holds if $W \cup C$ is inconsistent with $\bigvee_{Mb /\notin A}^b$.

**Refinements** The refinement of the abstraction upon finding a counterexample depends on the type of counterexample that is found. If $A$ is self-conflicting, superset of $A$ are ruled out and the algorithm proceeds to

---

**Algorithm 3** Credulous reasoning for stable, admissible and complete

Require: ABA framework $(W, D)$ and a query $q \in L_p$
Ensure: return YES if $q$ is credulously acceptable, NO otherwise
1: Let $\phi$ be an empty propositional formula over $A$
2: while $A \leftarrow \text{SAT}(\phi)$ do
3: \hspace{1em} $C \leftarrow \text{default_conclusions}(A)$
4: \hspace{1em} if derived($C, q$) and CF($A, C$) and semantics($A, C$) then
5: \hspace{2em} return YES
6: \hspace{1em} $\phi \leftarrow \text{Refine}(A)$
7: return NO
the next candidate, because the same conflict would be present in a super-
set of $A$. If $A$ does not derive the queried sentence or if $A$ is conflict-free but
not stable, subsets of $A$ are ruled out, because no subset of $A$ can derive the
query or the contrary of the assumption that is not attacked by $A$, either.
The search is continued to another candidate after adding the refinements.

Variations of Algorithm 3 The algorithms for admissible and com-
plete semantics have similar structure as the one for stable semantics. The
differences are in the conditions specific for the semantics in counterex-
ample checking and refinements when a counterexample is not found. To
decide skeptical acceptance, a search for a counterexample is performed.
In particular, the query is enforced to not be derivable in a candidate and
if a candidate is found to be a solution, then the query is not skeptically
accepted.

Grounded semantics Acceptance under grounded semantics has lower
complexity than the other considered semantics, being decidable with an
algorithm making a polynomial number of calls to an NP-solver. This is
based on all DL-ABA frameworks being flat, implying that the grounded
assumption set is the least fixed point of the defence operator [30]. Given a
set of assumptions $S$, computing what $S$ defends can be done by obtaining
the following sets: 1) $C$, the conclusions of default rules applicable from $S$,
2) $U$, the set of assumptions not attacked by $S$ based on $C$, and 3) $C'$, the
set of conclusions of default rules applicable from $U$. The set $S$ defends each
assumption whose negation is not entailed by $U \cup C'$. These assumptions
are added to $S$ and the process continued until nothing further is defended.

3.3.2 Empirical Evaluation
We implemented the algorithms using the PySAT Python interface [111]
and Glucose 3 [13] as the SAT solver. The implementation is available
in open source at https://bitbucket.org/coreo-group/satfordl-aba.
We test the implementation using 400 CNF formulas from a real-world ap-
lication, originally used for benchmarking iterative SAT-based algorithms
for the beyond-NP problem of backbone computation [112]. We generate
default rules using the variables occurring in the formulas and selected a
query literal at random. The number of default rules ranges from 100 to
400 and the total number of assumptions from 20 to 200 per instance. We
used a time limit of 10 minutes.

Table 3.5 shows the results. Notably, even instances with a larger num-
ber of default rules or assumptions can be solved within the time limit. The
### Table 3.5: Detailed runtime results for DL-ABA. There are a total of 400 instances per each combination of $|A|$ and $|D|$.

| $|A|$ | $|D|$ | #timeouts (mean runtime over solved instances (s)) |
|------|------|--------------------------------------------------|
|      |      | **stb** cred | **adm** cred | **com** cred | **grd** cred | **stb** skept |
| 100  | 20   | 20 (56) | 29 (51) | 28 (53) | 9 (11) | 22 (55) |
| 20   | 200  | 65 (68) | 66 (55) | 65 (59) | 12 (16) | 66 (62) |
| 400  | 97   | 88 (66) | 92 (60) | 10 (16) | 96 (71) |
| 100  | 115  | 108 (84) | 108 (84) | 13 (15) | 118 (117) |
| 50   | 200  | 180 (57) | 180 (59) | 8 (17) | 197 (111) |
| 400  | 264  | 223 (27) | 225 (22) | 13 (20) | 263 (57) |
| 100  | 285  | 201 (52) | 201 (55) | 12 (16) | 238 (109) |
| 100  | 299  | 235 (15) | 235 (17) | 12 (24) | 301 (32) |
| 100  | 20   | 293 (45) | 229 (25) | 231 (24) | 13 (22) | 297 (38) |
| 200  | 312  | 239 (7) | 239 (8) | 16 (21) | 310 (25) |
| 400  | 312  | 239 (11) | 240 (13) | 16 (21) | 316 (17) |

Hardness clearly increases as the number of rules or assumptions increases. In accordance with the complexity, deciding acceptance under grounded semantics is empirically easier than the other problems.
Chapter 4
Reasoning in Abstract Rule-based Argumentation

In this chapter, we turn our focus to the computation of acceptable claims in ASPIC\(^+\). This work is published in Articles IV and V. We focus on an instantiation of ASPIC\(^+\) with atomic sentences. We consider both the case with no preferences, and the case with preferences between premises and rules, using the weakest-link principle and elitist ordering for lifting preferences to comparisons between arguments.

TweetyProject [154, 155, 158] supports credulous and skeptical reasoning in ASPIC\(^+\) via a translation to AFs. TOAST [150] is another approach for ASPIC\(^+\), also translating an argumentation theory to an AF and employing an AF solver. Further implementations for reasoning in contexts drawing inspiration from ASPIC\(^+\) (as considered here) include EPR [167] and Arg2P [36]. Out of these, only TweetyProject is available and directly comparable to our approach.

Our contributions with respect to ASPIC\(^+\) are threefold. First, we re-define several central semantics and thereby the corresponding acceptance problems in ASPIC\(^+\) without relying on the construction of AFs. Via the new definitions, we propose declarative algorithms and prove complexity results for these problems. We develop ASP-based algorithms for deciding acceptance under admissible, complete, stable and preferred semantics in ASPIC\(^+\) without preferences and conduct an empirical evaluation in Article IV. The complexity of deciding acceptance under these semantics follows from our algorithms. In particular, we show that the complexity of deciding the acceptance of claims is the same in ASPIC\(^+\) without preferences as in LP-ABA and abstract argumentation under the semantics we consider. Specifically, credulous reasoning is NP-complete under admissible, complete, stable and preferred semantics, and skeptical reasoning is
coNP-complete under stable semantics and $\Sigma^P_2$-complete under preferred semantics. In Article V we consider ASPIC$^+$ with preferences. We provide a new definition for stable semantics defined directly on defeasible elements, under the weakest-link principle and elitist ordering, preserving accepted conclusions. We develop CEGAR algorithms based on the new definition, using incremental answer set solving for deciding the acceptance of atoms in this instantiation. Via the new definition for stable semantics, we show that the complexity of credulous acceptance is $\Sigma^P_2$-complete and skeptical acceptance is $\Pi^P_2$-complete. We show empirically that our approach outperforms the available comparison, TweetyProject.

4.1 ASPIC$^+$ without Preferences

In Article IV we consider the admissible, complete, stable and preferred semantics, which are recognized as central argumentation semantics. We give new definitions for these semantics in ASPIC$^+$ without preferences. We characterize acceptable sets of defeasible elements (premises and defeasible rules) and show equivalence with respect to acceptance problems between our definitions and the conventional definition based on constructing an AF. These new definitions underpin both the ASP encodings we propose and our complexity results.

4.1.1 Rephrasing Semantics

In ASPIC$^+$, the attacks between arguments are completely determined by their defeasible elements, i.e., the premises and defeasible rules the arguments contain. This motivates us to consider defeasible elements $(P,D)$ with $P \subseteq \mathcal{K}_p$ and $D \subseteq \mathcal{R}_d$. We refer to such a pair as an assumption, since intuitively the pairs have a similar role here as assumption sets have in ABA. In this section we explain how the credulous and skeptical acceptance in ASPIC$^+$ can be restated with reference only to assumptions instead of arguments.

First, we say that an assumption derives an atom (or, an atom is derivable from an assumption), i.e., $(P,D) \vdash x$, when one can chain rules from $D$ and $\mathcal{R}_s$ such that all body elements are in $P$, $\mathcal{K}_n$ or derived earlier, and $x$ is the conclusion of the last rule. We say that an argument $A$ is based on $(P,D)$ if all of the defeasible elements of $A$ are contained in $P$ and $D$, and that a defeasible rule $r$ is applicable by $(P,D)$ if the body elements of $r$ are derivable from $(P,D)$. An atom $x$ is derivable from an assumption if and only if an argument based on this assumption concludes $x$. 
We now consider attacks from assumptions to singular defeasible elements. With no preferences included, all attacks succeed as defeats, so we use the terms interchangeably in this section. Similarly to the conventional definitions of attack in ASPIC\(^+\), we have attacks corresponding to undermining, rebutting and undercutting. We say that an assumption \((P, D)\) attacks

- a premise \(p\) if \((P, D)\) derives a contrary or contradictory of the premise (undermine), and

- a defeasible rule \(r\) if either
  - \((P, D)\) derives the contrary of \(r\) (undercut), or
  - \((P, D)\) derives the contrary or contradictory of head\((r)\) (rebut).

Note that, rebutting the head of a rule is an attack on this head only in the context of this particular rule, but the same head derived via a strict rule is unattacked. Therefore, for the purposes of our computational approach, we consider undercuts (in addition to rebuts) to target rules instead of atoms. We define \(\text{att}(P, D)\) as the set of defeasible elements attacked by \((P, D)\).

There is a direct correspondence between attacks between arguments and attacks between assumptions. In particular, \((P, D)\) attacks a defeasible element if and only if an argument based on \((P, D)\) attacks all arguments containing the defeasible element (on some subargument).

Towards defining semantics on assumptions, we define that the assumption \((P, D)\) defends a defeasible element \(x\) if there are no attacks to \(x\) from the set of premises and defeasible rules that are not attacked by \((P, D)\). In other words, if one excludes the premises and defeasible rules that are attacked by \((P, D)\) from the set of all premises and defeasible rules and the remainder does not attack \(x\), then \((P, D)\) defends \(x\). In such a case, all attackers of \(x\) are attacked by \((P, D)\). We refer to the tuple of premises and defeasible rules defended by \((P, D)\) as \(\text{def}(P, D)\). That is, \(= \text{def}(P, D) = \{p \in K_p \mid (P, D) \text{ defends } p\}, \{r \in R_d \mid (P, D) \text{ defends } r\}\).

**Example 12.** Consider the argumentation theory with

\[
\begin{align*}
\mathcal{L} & = \{a, b, c, w, x, y, z, r_1, r_2, r_3\}, \\
K_p & = \{b\}, K_n = \{a, c\}, \\
\overline{r_2} & = \{z\}, \overline{b} = \{w\}, \text{ and } \overline{w} = \{x\} \\
R_d & = \{(r_1 : a \Rightarrow w), (r_2 : c \Rightarrow y), (r_3 : x \Rightarrow z)\}, \\
R_s & = \{(b \Rightarrow x)\}.
\end{align*}
\]
Consider the assumptions $M_1 = (\emptyset, \emptyset)$, $M_2 = (\{b\}, \{r_3\})$, and $M_3 = (\{b\}, \{r_1\})$. It holds that $\text{att}(M_1) = \emptyset$ (no contrary is derivable from only strict elements), $\text{att}(M_2) = \{r_1, r_2\}$ (since $\overline{w}$ and $\overline{r_2}$ are both derivable from $M_2$ and $r_1$ concludes $w$), and $\text{att}(M_3) = \{b, r_1\}$. Further, $\text{def}(M_1) = (\emptyset, \{r_3\})$ since $r_3$ is never attacked by an assumption. Lastly, $\text{def}(M_2) = (\{b\}, \{r_3\})$ (while $(\emptyset, \{r_1\})$ attacks $b$, $r_1$ is attacked by $M_2$) and $\text{def}(M_3) = (\{b\}, \{r_1, r_2, r_3\})$.

We are now ready to define argumentation semantics on assumptions. First, an assumption is conflict-free if it attacks no premises or defeasible rules contained in itself. For defining when an assumption $(P, D)$ satisfies the admissible, complete, stable or preferred semantics, we first require that $(P, D)$ is conflict-free and that all rules in $D$ are applicable by $(P, D)$.

Such an assumption $(P, D)$ is

- **admissible** if all elements in $P$ and $D$ are defended by $(P, D)$,
- **complete** if $(P, D)$ is admissible, each premise defended by $(P, D)$ is included in $P$, and each defeasible rule defended by $(P, D)$ is either included in $D'$ or not applicable by $\text{def}(P, D)$,
- **stable** if each premise is either in $P$ or attacked by $(P, D)$ and each defeasible rule is either in $D$, attacked by $(P, D)$, or not applicable by the defeasible elements not attacked by $(P, D)$, and
- **preferred** if there is no admissible assumption $(P', D')$ with $P \subseteq P'$ and $D \subseteq D'$ with at least one of the relations strict.

For $\sigma \in \{\text{adm, com, stb, prf}\}$, standing for admissible, complete, stable and preferred semantics, respectively, we refer to an assumption satisfying a semantics $\sigma$ as a $\sigma$-assumption. To see the need for the applicability conditions for complete and stable semantics, consider a rule that contains body elements that are not derivable in the argumentation theory at all. This rule can not give rise to arguments, so no argument using it is contained in a complete (AF) extension, and thus it does not need to be included in a complete assumption even if defended by the assumption. Similarly it does not need to be attacked by a stable assumption.

**Example 13.** The complete assumptions of the AT of Example 12 are $(\emptyset, \emptyset)$, $(\{b\}, \{r_3\})$, and $(\emptyset, \{r_1, r_2\})$. Note that each assumption defends $r_3$. However, the complete assumption $(\emptyset, \emptyset)$, for instance, does not include $r_3$. 

---

1 The applicability condition allows for a clearer correspondence between assumptions and the arguments based on them.
The assumption does not defend \( b \), the body of \( r_3 \), and thus \((\emptyset, \emptyset)\) is complete without including \( r_3 \). The assumptions \((\{b\}, \{r_3\})\), and \((\emptyset, \{r_1, r_2\})\) are also stable and preferred. For example, \((\{b\}, \{r_3\})\) contains all premises and derives \( x, z \), attacking \( r_1 \) and \( r_2 \) (the first by concluding the contrary of \( r_1 \)) but not \( b \) or \( r_3 \). We see the correspondence with the conventional AF-based definition of semantics here. The set of arguments based on \((\emptyset, \{r_1, r_2\})\) consist of the arguments \([a], [c], [[c] \Rightarrow y]\) and \([\lceil a \rceil \Rightarrow w]\) (each argument is enclosed in square brackets here). This is a stable and preferred extension: \([\lceil a \rceil \Rightarrow w]\) attacks \([b]\), which is the basis of all other arguments of the AF corresponding to this argumentation theory (and the set of arguments is conflict-free).

We show in Article IV that, under the four semantics considered, there is a natural correspondence between \( \sigma \)-assumptions and \( \sigma \)-extensions. In particular, the set of arguments that can be constructed from a \( \sigma \)-assumption is a \( \sigma \)-extension, and conversely the defeasible elements used in a \( \sigma \)-extension constitute a \( \sigma \)-assumption. The correspondence of credulous and skeptical acceptance between the definitions follows.

### 4.1.2 ASP Encodings for Stable, Admissible, Complete and Preferred Semantics

Our new definitions enable us to provide algorithms for deciding credulous and skeptical acceptance that do not incorporate the potentially exponential argument construction suggested by the conventional semantics definitions. On a high level, the algorithms make a non-deterministic guess over the defeasible elements of an argumentation theory and check if the assumption satisfies the conditions of the semantics. Checking the conditions can be done in polynomial time for all but preferred semantics, which needs another non-deterministic guess for an admissible superset of the original guess. We prove complexity of credulous and skeptical acceptance under the four semantics by these algorithms, summarized in Table 4.1. We show

Table 4.1: Complexity of deciding acceptance of an atom in ASPIC\(^+\) with no preferences. We establish these results in Article IV.

<table>
<thead>
<tr>
<th>Semantics</th>
<th>Credulous acceptance</th>
<th>Skeptical acceptance</th>
</tr>
</thead>
<tbody>
<tr>
<td>admissible</td>
<td>NP-complete</td>
<td>in ( P )</td>
</tr>
<tr>
<td>complete</td>
<td>NP-complete</td>
<td>in ( P )</td>
</tr>
<tr>
<td>preferred</td>
<td>NP-complete</td>
<td>( \Pi^P_2 )-complete</td>
</tr>
<tr>
<td>stable</td>
<td>NP-complete</td>
<td>co( \text{NP} )-complete</td>
</tr>
</tbody>
</table>
the membership results and hardness follows from previously introduced simple reductions from abstract argumentation.

We implement the algorithms for deciding acceptance under stable, admissible, complete and preferred semantics, based on finding $\sigma$-assumption sets, as ASP encodings. First, to represent an argumentation theory $AT$ in ASP, we give each $r \in R$ in the AT a name with the function $n'$ (extended from the potentially partial naming function $n$ specified by the AT). We represent an $AT = (L, R, n, \neg, K)$ in ASP as the facts

\[
AT(T) = \{\text{axiom}(a) \mid a \in K_n\} \cup \\
\{\text{premise}(a) \mid a \in K_p\} \cup \\
\{\text{head}(n'(r), b) \mid r \in R_d, b = \text{head}(r)\} \cup \\
\{\text{body}(n'(r), b) \mid r \in R_d, b \in \text{body}(r)\} \cup \\
\{\text{strict\_head}(n'(r), b) \mid r \in R_s, b = \text{head}(r)\} \cup \\
\{\text{strict\_body}(n'(r), b) \mid r \in R_s, b \in \text{body}(r)\} \cup \\
\{\text{contrary}(a, b) \mid b \in p, a \in L\}.
\]

We provide encodings for the four semantics, supporting both credulous and skeptical acceptance as well as the enumeration of $\sigma$-assumption. The encodings for the problems are similar to those of ABA (see Section 3.1.2). We summarize the differences here. Representing a queried atom and enforcing credulous or skeptical acceptance are done in the same manner as for ABA. The main differences of the encodings are that (i) here we guess both a set of premises and a set of defeasible rules and use only those, along with axioms and strict rules to decide what is derived from a guess, (ii) here we have rules for checking each attack type, and (iii) here we check the applicability of default rules, in accordance with our semantics definitions for assumptions (e.g. for stable semantics, an undefeated rule is allowed to be out of the stable pair if it is not applicable by the undefeated elements).

Concretely, we have the following rules for the guess:

\[
\text{in}(X) \leftarrow \text{axiom}(X).
\]

\[
\text{in}(X) \leftarrow \text{premise}(X), \neg \text{out}(X).
\]

\[
\text{out}(X) \leftarrow \text{premise}(X), \neg \text{in}(X).
\]

\[
\text{in}(X) \leftarrow \text{strict\_head}(X, \_).
\]

\[
\text{in}(X) \leftarrow \text{head}(X, \_), \neg \text{out}(X).
\]

\[
\text{out}(X) \leftarrow \text{head}(X, \_), \neg \text{in}(X).
\]
The set of atoms derivable from the guess is encoded by allowing the use of all strict rules and only the defeasible rules that are in the assumption, and the derivation to start from axioms and premises in the assumption. Thus we obtain which atoms are supported.

We add rules for each attack type from assumptions to defeasible elements:

\[
\begin{align*}
\text{defeated}(X) & \leftarrow \text{supported}(Y), \text{contrary}(X, Y), \text{premise}(X). \\
\text{defeated}(X) & \leftarrow \text{supported}(Y), \text{contrary}(X, Y), \text{head}(X, \_).
\end{align*}
\]

\[
\begin{align*}
\text{defeated}(X) & \leftarrow \text{head}(X, S), \text{supported}(Y), \text{contrary}(S, Y). \\
& \leftarrow \text{in}(X), \text{defeated}(X).
\end{align*}
\]

The non-applicability of rules that are not defeated in the case of stable semantics is enforced with the rule

\[
\leftarrow \text{out}(R), \text{applicable\_by\_undefeated}(R).
\]

Here applicable\_by\_undefeated(R) captures the rules that the set of elements not defeated by the guessed assumption is able to use. It is derived analogously to applicable\_by\_in(R) in Listing 3.1. In addition to these changes, the encodings follow the format familiar from our ABA encodings in Section 3.1.2. Skeptical acceptance under preferred semantics can be decided with ASPRIN in the same way as for ABA.

### 4.1.3 Empirical Evaluation

Our encodings for acceptance under admissible, complete, preferred and stable semantics are available at [https://bitbucket.org/coreo-group/asforsaspic](https://bitbucket.org/coreo-group/asforsaspic). To test the scalability of our approach, we constructed synthetic benchmarks with a varying number of atoms \( N \) (atoms here exclude names of rules), with \( N \) varying from 1000 to 5500. The results are summarized in Table 4.2 with a time limit of 600 seconds. Our approach is able to solve instances with thousands of atoms, with a majority of instances being solved under admissible, complete and stable semantics for \( N \leq 3000 \), for both credulous and skeptical reasoning. For preferred semantics, we tested enumeration of preferred assumptions, as that is the mode ASPRIN most readily supports and skeptical acceptance is immediate, given all preferred assumption sets.

We compared our algorithmic approach against TWEETYPROJECT [154] in Article V on benchmarks generated with similar parameters as here. TWEETYPROJECT employs a translation to AFs and we observed that the
Table 4.2: Timeouts and mean runtimes (timeouts included as 600 seconds) for problems on ASPIC$^+$ frameworks without preferences. There are 25 instances per value of $N$.

<table>
<thead>
<tr>
<th>$N$</th>
<th>adm</th>
<th>cred</th>
<th>com</th>
<th>cred</th>
<th>stb</th>
<th>cred</th>
<th>stb</th>
<th>skept</th>
<th>prf</th>
<th>enum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0</td>
<td>(1)</td>
<td>0</td>
<td>(1)</td>
<td>0</td>
<td>(1)</td>
<td>0</td>
<td>(1)</td>
<td>0</td>
<td>(15)</td>
</tr>
<tr>
<td>1500</td>
<td>0</td>
<td>(18)</td>
<td>0</td>
<td>(16)</td>
<td>0</td>
<td>(22)</td>
<td>0</td>
<td>(28)</td>
<td>3</td>
<td>(243)</td>
</tr>
<tr>
<td>2000</td>
<td>0</td>
<td>(75)</td>
<td>2</td>
<td>(108)</td>
<td>2</td>
<td>(124)</td>
<td>2</td>
<td>(100)</td>
<td>13</td>
<td>(446)</td>
</tr>
<tr>
<td>2500</td>
<td>5</td>
<td>(179)</td>
<td>7</td>
<td>(235)</td>
<td>5</td>
<td>(197)</td>
<td>4</td>
<td>(150)</td>
<td>15</td>
<td>(499)</td>
</tr>
<tr>
<td>3000</td>
<td>8</td>
<td>(248)</td>
<td>9</td>
<td>(273)</td>
<td>8</td>
<td>(242)</td>
<td>6</td>
<td>(211)</td>
<td>20</td>
<td>(537)</td>
</tr>
<tr>
<td>3500</td>
<td>16</td>
<td>(421)</td>
<td>19</td>
<td>(501)</td>
<td>15</td>
<td>(374)</td>
<td>14</td>
<td>(374)</td>
<td>24</td>
<td>(590)</td>
</tr>
<tr>
<td>4000</td>
<td>17</td>
<td>(447)</td>
<td>20</td>
<td>(541)</td>
<td>20</td>
<td>(481)</td>
<td>18</td>
<td>(472)</td>
<td>25</td>
<td>(600)</td>
</tr>
<tr>
<td>4500</td>
<td>17</td>
<td>(463)</td>
<td>19</td>
<td>(501)</td>
<td>16</td>
<td>(410)</td>
<td>18</td>
<td>(437)</td>
<td>25</td>
<td>(600)</td>
</tr>
<tr>
<td>5000</td>
<td>23</td>
<td>(563)</td>
<td>24</td>
<td>(594)</td>
<td>17</td>
<td>(410)</td>
<td>17</td>
<td>(410)</td>
<td>25</td>
<td>(600)</td>
</tr>
<tr>
<td>5500</td>
<td>21</td>
<td>(523)</td>
<td>23</td>
<td>(553)</td>
<td>21</td>
<td>(505)</td>
<td>22</td>
<td>(529)</td>
<td>24</td>
<td>(592)</td>
</tr>
</tbody>
</table>

Table translation is not able to be completed on a majority of instances with 200 atoms within the time limit. In contrast, our approach is able to solve most instances with up to 3000 atoms (under semantics other than preferred) or 1500 atoms (under preferred semantics).

4.2 ASPIC$^+$ with Preferences

In Article V, we focus on ASPIC$^+$ with preferences over premises and defeasible rules, using the weakest-link principle and elitist ordering. We introduce a novel redefinition of stable semantics relying only on the notion of sets of premises and defeasible rules. The correspondence between the conventional semantics and our novel definition holds for well-formed argumentation theories (recall Definition 14). This theoretical advance allows us to develop algorithms that are appropriate for the complexity of the problems. In addition, we use our redefinition to prove that credulous and skeptical reasoning are computationally harder in this setting compared to ASPIC$^+$ with no preferences.

4.2.1 Redefining Stable Semantics

In contrast to ASPIC$^+$ with no preferences, there are five types of defeats (successful attacks) with different behaviour in ASPIC$^+$ when preferences are taken into account: undercut, contrary rebut and undermine, and con-
tradictory rebut and undermine. We prove in Article V that for all of these except for contraditory rebut, it holds that a defeat of a defeasible element generalizes to any argument that uses the defeasible element. Thus we refer to successful undercut, contraditory undermine, contrary undermine and contraditory rebut as individual defeats.

We define individual defeats from assumptions to defeasible elements such that given an assumption \((P,D)\) and an argument \(A\), there is an argument based on \((P,D)\) that defeats \(A\) with an individual defeat if and only if \((P,D)\) individually defeats a defeasible elements used in \(A\). First, we define the set \(X_{\not<}X' = \{x \in X | \exists x' \in X' : x \not< x'\}\) for sets of premises or defeasible rules \(X\) and \(X'\). In other words, \(X_{\not<}X'\) contains the elements of \(X\) that are not less preferred than every element in \(X'\). Under the elitist ordering, \(X_{\not<}X'\) is exactly the subset of \(X\) for which it holds that \(X_{\not<}X' \not< X'\). We say that \((P,D)\) individually defeats an \(x \in K_p \cup R_d\) if

- \(x\) is a premise and
  - a contrary of \(p\) is derivable from \((P,D)\) (contrary undermine),
  - a contraditory of \(p\) is derivable from \((P_{\not<}P', D)\), or

- \(x\) is a defeasible rule and
  - the contrary of \(r\) is derivable from \((P,D)\) (undercut), or
  - a contrary of head(\(r\)) is derivable from \((P,D)\) (contrary rebut).

To understand why individual defeats generalize to arguments containing them, note the following. First, undercut, contrary rebut and contrary undermine succeed regardless of preferences by definition. Moreover, contraditory undermine is defined on single premises: an argument is successfully contraditory undermined if and only if one of its premises is. Checking if an assumption contraditory undermines a premise is simple even under preferences. The assumption \((P,D)\) contraditory undermines a premise if and only if one can derive the contraditory of the premise from axioms and those elements of \(P\) that are not less preferred than the premise, using \(D\) and strict rules.

The fact that individual defeats generalize to all arguments using them implies polynomial time decidability for whether the arguments that are based on a given assumption defeat a given argument via an individual defeat. On the other hand, in contrast to when no preferences are present, contraditory rebuts do not generalize from a defeasible element to arguments containing the elements. Instead it is required to compare the preference relation between the attacker and all of the defeasible elements in the attacked argument.
Example 14. Let $T = (L, R, n, \neg, K, \leq)$ be an AT with $L = \{a, b, c, x, \neg x\}$, $K_p = \{a, b, c\}$, $K_n = \emptyset$, $R_s = \{b \rightarrow z\}$, and $R_d = \{r_1 : (z \Rightarrow x), r_2 : (a \Rightarrow \neg x)\}$. Further, let $a < b$, $r_2 < r_1$, and $x$ and $\neg x$ be contradictory.

There are two arguments concluding $x$ in $T$: $A_1 = [[b \rightarrow z] \Rightarrow x]$ and $A_2 = [[c \rightarrow z] \Rightarrow x]$. They both have $r_1$ as the top rule. However, the arguments based on the assumption $\{a\}$ (namely the argument for $\neg x$) only successfully contradictory rebuts $A_2$ and not $A_1$. This is because $\{a\}$ is less preferred than $\text{Prem}_d(A_1) = \{b\}$ but not $\text{Prem}_d(A_2) = \{c\}$, so the attack to the former fails and to the latter succeeds.

We thus define successful contradictory rebuts between assumptions (instead of from an assumption to a defeasible element, as for individual defeats). Given assumptions $(P, D)$ and $(P', D')$, we say that $(P, D)$ contradictory rebuts $(P', D')$ on $r \in D'$ if either

- $P'$ is not empty and the contradictory of head($r$) is derivable from either $(P, D_{\neg D'})$ or $(P_{\neg P'}, D)$, or
- $P'$ is empty and the contradictory of head($r$) is derivable from $(P, D_{\neg D'})$.

The sets $P_{\neg P'}$ and $D_{\neg D'}$ are chosen to reflect the elitist ordering. The two cases\(^2\) for when a contradictory rebut is successful reflect the weakest-link principle. Namely, if $P'$ is empty, then only defeasible rules are taken into account. In particular, the defeasible rules used in constructing the attack must not be less preferred than those of the attacked assumption in order for the rebut to be successful. Otherwise, per the definition of weakest-link principle, it suffices that either the defeasible rules or the premises of the attacking assumption are not less preferred than the corresponding ones of the attacked assumption. We prove in Article V that this definition captures contradictory rebuts in the sense that if an assumption $(P, D)$ contradictory rebuts an assumption $(P', D')$ on $r \in D'$, then each argument based on $(P', D')$ with $r$ as the top rule is successfully contradictory rebutted by an argument based on $(P, D)$.

Example 15. Consider the argumentation theory of Example 14 and the assumption $(P, D) = (\{a\}, \{r_2\})$ again. It holds that $\neg x$ can be derived from $(P_{\neg \{c\}}, D) = (P, D)$ but not from $(P_{\neg \{b\}}, D) = (\emptyset, D)$ or $(P, D_{\neg \{r_1\}}) = (P, \emptyset)$. Thus, using our definition of contradictory rebut between assumptions, $(P, D)$ defeats $(\{c\}, \{r_1\})$ but not $(\{b\}, \{r_1\})$. This corresponds to the

\(^2\)The cases are that either $P'$ is empty or neither $P'$ nor $D'$ are empty, as by definition $D'$ is not empty.
defeats between arguments, where the argument \([[[c] \rightarrow z] \Rightarrow x]\) (based on \((\{c\}, \{r_1\})\)) is defeated by the argument \([a \Rightarrow \neg x]\) (based on \((P, D)\)), but the argument \([[[b] \rightarrow z] \Rightarrow x]\) (based on \((\{b\}, \{r_1\})\)) is not.

Finally, we define stable semantics on assumptions, based on our definitions for defeats from assumptions. The definition first ensures conflict-freeness. Secondly, each premise not in the assumption must be defeated. Lastly, a more complicated condition states that for an assumption \((P, D)\) to be stable, there must not be another assumption whose defeasible elements are not defeated by \((P, D)\) (even by contradictory rebut).

**Definition 16.** Let \(T = (\mathcal{L}, \mathcal{R}, n, -, \mathcal{K}, \leq)\) be a well-formed AT. Assumption \((P, D)\) of \(T\) is stable if all \(D\) are applicable by \((P, D)\) and

1. \(\not\exists x, y \in \text{Th}_T(P, D)\) such that \(x\) is a contrary of \(y\), \(x\) and \(y\) are contradictory, or \(x \in \{n(r) \mid r \in D\}\),

2. \((P, D)\) individually defeats all \(p \in \mathcal{K}_P \setminus P\), and

3. \((P, D)\) contradictory rebuts all \((P', D')\) such that
   - each rule in \(D'\) is applicable by \((P', D')\),
   - \(D' \nsubseteq D\), and
   - \(P' \subseteq P\) and \(D' \subseteq D_U\), where \(D_U\) are the rules in \(\mathcal{R}_d\) that are not individually defeated by \((P, D)\).

A \((P', D')\) as defined in the third item would be a counterexample to \((P, D)\) corresponding to a stable extension if not contradictorily rebutted by \((P, D)\). To see this, first note that at least one argument can be constructed via \((P', D')\) as all rules in \(D'\) are applicable. Secondly, some argument based on \((P', D')\) is not based on \((P, D)\), since \(D' \nsubseteq D\). Lastly, by construction, this argument is not defeated by \((P, D)\) with an individual defeat or by contradictory rebut. Thus, from \((P', D')\) one can construct an argument that is not based on \((P, D)\) and is not defeated by it, and so the set of arguments based on \((P, D)\) is not a stable extension.

Similarly to the case with no preferences, the set of arguments that can be constructed from a stable assumption is a stable extension, and the defeasible elements used in a stable extension constitute a stable assumption. The acceptable conclusions drawn using assumptions and extensions similarly coincide. Thus we can use answer credulous and skeptical acceptance by finding stable assumptions.
Algorithm 4 Credulous justification

Require: Well-formed AT $T$ and queried atom $s \in \mathcal{L}$
Ensure: return YES if $s$ is credulously justified in $T$ under stable semantics, NO otherwise
1: $\pi \leftarrow \pi_{1,2}(T) \cup \{\leftarrow \text{not derived}(s)\}$
2: while $\pi$ has an answer set $M$ do
3: if $\pi_{\neg 3}(M)$ has no answer sets then return YES
4: else rule out subsets of candidate assumption in $\pi$
5: return NO

4.2.2 Algorithms for Acceptance under Stable Semantics

We detail an algorithm for credulous and skeptical acceptance under stable semantics, which also establishes that the problems are in $\Sigma^P_2$ and $\Pi^P_2$, respectively. We also prove the corresponding hardness results for both decision problems in Article V.

The theoretical advances outlined in the previous section enable us to develop algorithms for deciding the credulous and skeptical acceptance of claims by finding stable assumptions instead of constructing an abstract argumentation framework. On a high level, our approach is to non-deterministically guess an assumption (that either does or does not derive a queried atom, for credulous and skeptical acceptance, respectively) and then checking the conditions of Definition 16. Checking the conditions requires another non-deterministic guess, namely guessing a potential counterexample $(P', D')$ and checking the third condition of Definition 16.

We implement this approach with a CEGAR algorithm making iterative ASP calls. For generating candidates and checking for counterexamples, we introduce ASP encodings $\pi_{1,2}$ and $\pi_{\neg 3}$. The encodings, previously published only in an online appendix of Article V, are included in Appendix A. The approach for credulous acceptance (presented as Algorithm 4) works as follows:

1. A candidate solution is an assumption that satisfies conditions 1 and 2 from Definition 16, where each rule is applicable, and that derives the query. In other words, for a candidate $(P, D)$ it holds that each rule in $D$ is applicable by $(P, D)$, the candidate is conflict-free and individually defeats each premise not in $P$. A candidate satisfying these conditions is obtained on Line 2 of Algorithm 4.

2. A counterexample is another assumption that violates condition 3 of the definition. In particular, a counterexample contains only applica-
4.2 ASPIC$^+$ with Preferences

...ble rules, contains some rules not contained in $D$, contains no defeasible elements that are individually defeated by $(P, D)$, and lastly, is not contradictory rebutted by $(P, D)$. Line 3 checks for the existence of a counterexample for this particular candidate assumption$^3$. If a counterexample does not exist, the candidate is stable and thus the query is credulously justified.

3. If a counterexample is found, the current $\pi$ is refined to exclude each $(P'', D'')$ with $P'' \subseteq P$ and $D'' \subseteq D$ (Line 4). Excluding all subassumptions is valid, since if $(P, D)$ does not contradictory rebut a counterexample $(P', D')$, then no subassumption of $(P, D)$ can contradictory rebut $(P', D')$ either.

Skeptical acceptance can be decided with the familiar inversion: search for a stable assumption that does not contain the query, and report NO if such an assumption is found and YES otherwise.

4.2.3 Empirical Evaluation

We implemented the ASP-based CEGAR algorithms using the incremental Python interface of CLINGO [99]. The algorithms are available in open source at https://bitbucket.org/coreo-group/aspforaspic/. We generated synthetic benchmarks with the number of atoms $N$ varying from 100 to 800 per instance (atoms here exclude the names of rules). We also varied the number of rules deriving each atom ($rpa$) and the sizes of rule bodies ($rs$): for each non-premise atom, the number of rules deriving the atom was chosen at random from $[1, 5]$ or $[1, 10]$, as was the number of atoms in the body of each rule body.

Tables 4.3 and 4.4 give the number of timeouts and mean runtimes under a time limit of 600 seconds of our approach for each $N$ and choice of $rs$, $rpa$ for credulous and skeptical reasoning, respectively. Timed out instances are included as 600 seconds in the mean running times. The empirical hardness of the instances depend on the parameters and reasoning tasks. The instances with more rules (up to ten rules per atom) and smaller rule bodies (rule body size up to five) seem the hardest for both reasoning tasks while the instances with fewer and smaller rules are relatively easier.

We compared our implementation with TWEETYP'ROJECT [154], which employs a translation to AFs and explicitly generates the AF from a given AT as its first step. We observed that already the argument construction

---

$^3$The pseudocode as presented in Article V erroneously states the program for the counterexample check to be $\pi \cup \pi_{\neg 3}(M)$ instead of $\pi_{\neg 3}(M)$. 
Table 4.3: Timeouts and mean runtimes (timeouts included as 600 seconds) for credulous acceptance under stable semantics in ASPIC$^+$ with preferences. There are 10 instances for each value of $N$.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$rs=5, rpa=5$</th>
<th>$rs=5, rpa=10$</th>
<th>$rs=10, rpa=10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0 (1)</td>
<td>0 (16)</td>
<td>0 (6)</td>
</tr>
<tr>
<td>200</td>
<td>0 (8)</td>
<td>4 (321)</td>
<td>0 (49)</td>
</tr>
<tr>
<td>300</td>
<td>1 (85)</td>
<td>6 (428)</td>
<td>0 (182)</td>
</tr>
<tr>
<td>400</td>
<td>0 (64)</td>
<td>4 (491)</td>
<td>3 (474)</td>
</tr>
<tr>
<td>500</td>
<td>4 (316)</td>
<td>10 (600)</td>
<td>10 (600)</td>
</tr>
<tr>
<td>600</td>
<td>0 (222)</td>
<td>10 (600)</td>
<td>10 (600)</td>
</tr>
<tr>
<td>700</td>
<td>1 (405)</td>
<td>10 (600)</td>
<td>10 (600)</td>
</tr>
<tr>
<td>800</td>
<td>3 (556)</td>
<td>10 (600)</td>
<td>10 (600)</td>
</tr>
</tbody>
</table>

Table 4.4: Timeouts and mean runtimes (timeouts included as 600 seconds) for skeptical acceptance under stable semantics in ASPIC$^+$ with preferences. There are 10 instances for each value of $N$.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$rs=5, rpa=5$</th>
<th>$rs=5, rpa=10$</th>
<th>$rs=10, rpa=10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0 (2)</td>
<td>0 (74)</td>
<td>0 (7)</td>
</tr>
<tr>
<td>200</td>
<td>0 (51)</td>
<td>8 (509)</td>
<td>0 (114)</td>
</tr>
<tr>
<td>300</td>
<td>1 (190)</td>
<td>10 (600)</td>
<td>1 (279)</td>
</tr>
<tr>
<td>400</td>
<td>7 (445)</td>
<td>9 (582)</td>
<td>5 (531)</td>
</tr>
<tr>
<td>500</td>
<td>9 (567)</td>
<td>10 (600)</td>
<td>10 (600)</td>
</tr>
<tr>
<td>600</td>
<td>8 (524)</td>
<td>10 (600)</td>
<td>10 (600)</td>
</tr>
<tr>
<td>700</td>
<td>9 (580)</td>
<td>10 (600)</td>
<td>10 (600)</td>
</tr>
<tr>
<td>800</td>
<td>10 (600)</td>
<td>10 (600)</td>
<td>10 (600)</td>
</tr>
</tbody>
</table>

step fails (due to time or memory out) for all but seven of the benchmark instances (five of these instances had $N = 100$ and two $N = 200$). In comparison, our approach was able to solve the whole problem on instances that are multiple times larger.
Chapter 5

Conclusions

In this thesis, we considered computational aspects of argumentative reasoning starting from a knowledge base and ending with the identification of acceptable claims. Efficient algorithmic approaches for argumentative reasoning problems are essential for realizing the potential of formal argumentation in practice, especially because most of the central problems are NP-hard. From the point of view of practical efficiency, it is important to consider the whole process of accepting claims. The results of this thesis suggest that simply constructing an abstract argumentation framework and deferring to the algorithmic approaches to acceptance in abstract argumentation can be very inefficient compared to a direct approach. As the main contributions of this thesis, we developed practical algorithms based on the declarative paradigms of ASP and SAT, and proved complexity results for deciding the acceptance of claims in various instantiations of the central structured argumentation formalisms ABA and ASPIC$^+$. 

Concretely, focusing on various versions and semantics of ABA (namely LP-ABA, DL-ABA, and LP-ABA$^+$), we developed ASP and SAT-based declarative algorithms that non-deterministically guess a set of assumptions and check the conditions of a given semantics. To extend this approach to ASPIC$^+$, we defined central semantics without reference to abstract argumentation frameworks. For ASPIC$^+$, our approach covers admissible, complete, stable and preferred semantics when preferences are not included, and stable semantics when preferences are included under weakest-link principle and elitist ordering. We defined the semantics on sets of defeasible elements (premises and defeasible rules) and showed that a set of defeasible elements satisfies the conditions we presented for a semantics if and only if the arguments that are constructible using no other defeasible elements is an extension in the corresponding AF. This constitutes the foundation of our algorithmic approach to ASPIC$^+$, allowing for algorithms that non-
deterministically guess a set of premises and defeasible rules and check the conditions of the given semantics. Our algorithms avoid the construction of a potentially exponential-size abstract argumentation framework which is suggested by the conventional definition of ASPIC$^+$ semantics.

The algorithms we introduced are, to the best of our knowledge, the first to use modern declarative solving techniques without constructing an AF for ABA and ASPIC$^+$ problems. Depending on the instantiation and semantics we considered, the complexity of deciding the acceptance of a claim ranges from polynomial time decidability to completeness for the second level of the polynomial hierarchy. For the latter case, we developed CEGAR algorithms making multiple calls incrementally to ASP or SAT solvers. The algorithms first search for a candidate solution with a solver call and then check the conditions of the particular semantics with another solver call. Our approach clearly outperforms other available implementations of deciding acceptance in ABA and ASPIC$^+$ for each problem we considered.

We also provided complexity results for LP-ABA$^+$ and ASPIC$^+$. Notably, we showed that credulous reasoning under admissible, complete, stable and preferred semantics in ASPIC$^+$ without preferences is NP-complete, the same as for ABA and abstract argumentation. The similarity also holds for skeptical acceptance: under admissible and complete semantics it is in P, under stable semantics coNP-complete and under preferred semantics $\Pi_2^P$-complete. We also showed that when including preferences in ASPIC$^+$ with the weakest-link principle and elitist ordering, the complexity of credulous and skeptical acceptance under stable semantics is $\Sigma_2^P$-complete and $\Pi_2^P$-complete, respectively. For LP-ABA$^+$, we proved that acceptance under admissible semantics is harder than in LP-ABA, but acceptance under stable semantics has the same complexity for both instantiations.

There are multiple compelling directions for continuing the research of this thesis. The methodology introduced in this thesis paves the way for developing algorithms for further problems in rule-based structured argumentation. Notably our approach could be extended to other semantics than stable in ASPIC$^+$ with preferences. In addition, there are also other (well-behaved) mechanisms for preference handling, such as the last-link principle and democratic ordering [117, 121, 122]. Our approach of considering the acceptability of sets of defeasible elements rather than arguments could be extended to other semantics and preference instantiations to develop efficient algorithms for deciding the acceptance of claims under them. Similarly, the computational complexity of these variants is largely not established. If possible, it would be worthwhile to identify semantics and pre-
ference instantiations under which acceptance would be easier than under stable semantics with weakest-link principle and elitist ordering. Additionally, the complexity landscape of ASPIC$^+$ without preferences is not fully established. For example, the complexity of acceptance under grounded and ideal semantics are missing from our results, and likewise declarative algorithms for them.

There have been efforts to develop more efficient ways to instantiate AFs with structured argumentation [173, 174, 177, 178]. Recently, a preprocessing procedure for so-called non-circular LP-ABA frameworks has been identified, polynomially bounding the size of the AF that the framework gives rise to [145]. Evaluating the performance and benefits of different approaches, and possibly extending the preprocessing approach to larger classes of ABA frameworks or to ASPIC$^+$, are interesting directions for future research.

Beyond the instantiations we considered in this thesis, a promising avenue for future research is extending our algorithmic approach and complexity analysis to other forms of structured argumentation, including variants of ABA and ASPIC$^+$. Potential directions are, for example, non-flat instantiations of ABA, such as the autoepistemic logic instantiation [30, 123], or variants of ASPIC$^+$ where rationality properties are guaranteed to hold more generally [37, 39, 66, 108, 109]. In addition, similar algorithmic approaches to other structured argumentation formalisms, such as DeLP [25, 93, 94, 95] or deductive argumentation [24, 26], could be studied. Extending our approach to further problems building on acceptance of claims would be beneficial as well, for example in dynamic settings [120, 130, 131, 144]. Lastly, application case studies of the algorithmic approaches developed in this thesis offer yet another interesting direction for research beyond the scope of this thesis.
References


[35] Gerhard Brewka and Stefan Woltran. Abstract dialectical frameworks. In Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczynski,


References


References


References


Appendix A Details on ASP Encodings

We present the encodings $\pi_{1,2}(T)$ and $\pi_{-3}(M)$ used in our algorithms for deciding credulous and skeptical acceptance in ASPIC$^+$ with preferences (Algorithm 4). The former encodes the candidate generation and the latter the counterexample check.

Listing A.1 checks if there are assumptions in the given $AT$ that satisfy conditions 1 and 2 from Definition 15, and if so, detects this solution candidate and the defeasible elements are are not individually defeated by the candidate. The transitivity of preferences, and helper predicates for transitivity, are encoded (Lines 1-4). In Lines 5, the symmetry of the contradictory relation is enforced. Lines 6–10 encode a non-deterministic divide of the defeasible parts of the given $AT$ into elements that are in the candidate assumption $(P, D)$ and those that are out. Lines 11–16 compute atoms that are derivable from the candidate assumption. Line 17 enforces that all rules in $D$ are applicable. The individual defeats except for contradictory undermine are computed in Lines 18–20. For the purposes of checking conflict-freeness, Line 21 computes for which rules the contradictory of the head of said rule is derivable from the candidate. For the purposes of checking contradictory undermining, Lines 22–27 derive all atoms that are derivable from $(P', D)$, where $P' \subseteq P$ and $P'$ is not less preferred than a given premise. Line 28 obtains premises that are defeated via contradictory undermine. Lines 29–32 enforce conflict-freeness of the candidate. Finally, Lines 33–34 compute which defeasible elements are undefeated by the candidate, and Line 35 enforces that all premises not in the candidate must be defeated by the candidate.

Listing A.2 gets as input the candidate under consideration (predicate in) and defeasible elements that are not individually defeated by the candidate (predicate undefeated), and checks if there is a counterexample to the candidate being stable, according to the last item of Definition 15. Lines 1–6 are shared from Listing A.1. Lines 7–11 non-deterministically guess a subset of the defeasible parts that are not defeated, this constituting a possible counterexample $(P', D')$. Lines 12–18 compute atoms derivable from
the possible counterexample and enforce that all rules in the suspects must
be applicable by the possible counterexample. Lines 17 and 20 identify the
“not less preferred” premise and rule sets $P_{\not \leq P'}$ and $D_{\not \leq D'}$. In Lines 21–32
it is checked which atoms are derivable from $(P, D_{\not \leq D'})$ and $(P_{\not \leq P'}, D)$, and
(successful) contradictory rebuts are checked in Lines 34 (when $P' = \emptyset$) and 35. Lastly, Line 36 checks if the suspect elements are a subset of the
candidate, and Lines 37 and 38 rule out the suspect assumptions as a coun-
terexample if the suspects are a subset of the candidate or if the suspect
assumption is contradictory rebutted by the candidate, respectively.
Listing A.1: Module $\pi_{1,2}(T)$

1. \texttt{preferred}(X,Z) ← \texttt{preferred}(X,Y), \texttt{preferred}(Y,Z).
2. \texttt{strictly_less_preferred}(X,Y) ← \texttt{preferred}(Y,X), \texttt{not preferred}(X,Y).
3. \texttt{no_less_preferred}(X,Y) ← \texttt{not strictly_less_preferred}(X,Y),
   \texttt{premise}(X), \texttt{premise}(Y).
4. \texttt{no_less_preferred}(X,Y) ← \texttt{not strictly_less_preferred}(X,Y), \texttt{head}(X,\_),
   \texttt{head}(Y,\_).
5. \texttt{contradicts}(X,Y) ← \texttt{contradicts}(Y,X).
6. \texttt{in}(X) ← \texttt{axiom}(X).
7. \texttt{in}(X) ← \texttt{premise}(X), \texttt{not out}(X).
8. \texttt{out}(X) ← \texttt{premise}(X), \texttt{not in}(X).
9. \texttt{in}(R) ← \texttt{head}(R,\_), \texttt{not out}(R).
10. \texttt{out}(R) ← \texttt{head}(R,\_), \texttt{not in}(R).
11. \texttt{derived}(X) ← \texttt{axiom}(X).
12. \texttt{derived}(X) ← \texttt{premise}(X), \texttt{in}(X).
13. \texttt{derived}(X) ← \texttt{head}(R,X), \texttt{used_by_in}(R).
14. \texttt{derived}(X) ← \texttt{strict_head}(R,X), \texttt{used_by_in}(R).
15. \texttt{used_by_in}(R) ← \texttt{in}(R), \texttt{head}(R,\_), \texttt{derived}(X) : \texttt{body}(R,X).
16. \texttt{used_by_in}(R) ← \texttt{in}(R), \texttt{strict_head}(R,\_), \texttt{derived}(X) : \texttt{strict_body}(R,X).
17. \texttt{in}(R) ← \texttt{in}(R), \texttt{not used_by_in}(R), \texttt{head}(R,\_).
18. \texttt{defeated}(X) ← \texttt{derived}(Y), \texttt{contrary}(X,Y), \texttt{head}(X,\_).
19. \texttt{defeated}(X) ← \texttt{derived}(Y), \texttt{contrary}(X,Y), \texttt{premise}(X).
20. \texttt{defeated}(X) ← \texttt{head}(X,S), \texttt{derived}(Y), \texttt{contrary}(S,Y).
21. \texttt{contradict_rebut_conflict}(X) ← \texttt{head}(X,S), \texttt{derived}(Y), \texttt{contradicts}(S,Y).
22. \texttt{pref_derived}(X,Y) ← \texttt{no_less_preferred}(X,Y), \texttt{premise}(Y), \texttt{premise}(X),
   \texttt{in}(X).
23. \texttt{pref_derived}(X,Y) ← \texttt{axiom}(X), \texttt{premise}(Y).
24. \texttt{pref_derived}(X,Y) ← \texttt{head}(R,X), \texttt{used_by_pref_premises}(R,Y).
25. \texttt{pref_derived}(X,Y) ← \texttt{strict_head}(R,X), \texttt{used_by_pref_premises}(R,Y).
26. \texttt{used_by_pref_premises}(R,Y) ← \texttt{head}(R,\_), \texttt{in}(R), \texttt{premise}(Y),
   \texttt{pref_derived}(X,Y) : \texttt{body}(R,X).
27. \texttt{used_by_pref_premises}(R,Y) ← \texttt{strict_head}(R,\_), \texttt{premise}(Y), \texttt{in}(R),
   \texttt{pref_derived}(X,Y) : \texttt{strict_body}(R,X).
28. \texttt{defeated}(X) ← \texttt{pref_derived}(Y,X), \texttt{contradicts}(X,Y), \texttt{premise}(X).
29. \texttt{in}(X), \texttt{defeated}(X).
30. \texttt{in}(X), \texttt{contradict_rebut_conflict}(X).
31. \texttt{derived}(X), \texttt{derived}(Y), \texttt{contrary}(X,Y).
32. \texttt{derived}(X), \texttt{derived}(Y), \texttt{contradicts}(X,Y).
33. \texttt{undefeated}(X) ← \texttt{premise}(X), \texttt{not defeated}(X).
34. \texttt{undefeated}(R) ← \texttt{head}(R,\_), \texttt{not defeated}(R).
35. \texttt{premise}(X), \texttt{out}(X), \texttt{undefeated}(X).
Listing A.2: Module $\pi_\text{3}(M)$

1. preferred(X,Z) ← preferred(X,Y), preferred(Y,Z).
2. strictly_less_preferred(X,Y) ← preferred(Y,X), not preferred(X,Y).
3. no_less_preferred(X,Y) ← not strictly_less_preferred(X,Y), premise(X), premise(Y).
4. no_less_preferred(X,Y) ← not strictly_less_preferred(X,Y), head(X,\_), head(Y,\_).
5. contradicts(X,Y) ← contradicts(Y,X).
6. in(X) ← axiom(X).
7. suspect(X) ← axiom(X).
8. suspect(X) ← undefeated(X), not other(X).
9. other(X) ← premise(X), not suspect(X).
10. other(R) ← head(R,\_), not suspect(R).
11. other(R) ← strict_head(R,\_), not suspect(R).
12. derived_by_suspects(X) ← axiom(X).
13. derived_by_suspects(X) ← premise(X), suspect(X).
14. derived_by_suspects(X) ← head(R,X), used_by_suspects(R).
15. derived_by_suspects(X) ← strict_head(R,X), used_by_suspects(R).
16. used_by_suspects(R) ← suspect(R), head(R,\_), derived_by_suspects(X) : body(R,X).
17. used_by_suspects(R) ← suspect(R), strict_head(R,\_), derived_by_suspects(X) : strict_body(R,X).
18. ← suspect(R), not used_by_suspects(R), head(R,\_).
19. pref_premise(X) ← premise(X), in(X), no_less_preferred(X,Y), premise(Y), suspect(Y).
20. pref_rule(R) ← head(R,\_), in(R), no_less_preferred(R,Y), head(Y,\_), suspect(Y).
21. derived_by_pref_prem(X) ← pref_premise(X).
22. derived_by_pref_prem(X) ← axiom(X).
23. derived_by_pref_prem(X) ← head(R,X), used_by_pref_premises(R).
24. derived_by_pref_prem(X) ← strict_head(R,X), used_by_pref_premises(R).
25. used_by_pref_premises(R) ← head(R,\_), in(R), derived_by_pref_prem(X) : body(R,X).
26. used_by_pref_premises(R) ← strict_head(R,\_), in(R), derived_by_pref_prem(X) : strict_body(R,X).
27. derived_by_pref_rules(X) ← in(X), premise(X).
28. derived_by_pref_rules(X) ← axiom(X).
29. derived_by_pref_rules(X) ← head(R,X), used_by_pref_rules(R).
30. derived_by_pref_rules(X) ← strict_head(R,X), used_by_pref_rules(R).
31. used_by_pref_rules(R) ← pref_rule(R), derived_by_pref_rules(X) : body(R,X).
32. used_by_pref_rules(R) ← strict_rule(R), derived_by_pref_rules(X) : strict_body(R,X).
33. suspect_includes_premises ← suspect(X), premise(X).
34. rebutted_suspect ← contradicts(X,Y), head(R,X), suspect(R), derived_by_pref_prem(Y), suspect_includes_premises.
35. rebutted_suspect ← contradicts(X,Y), head(R,X), suspect(R), derived_by_pref_rules(Y).
36. subset ← in(X) : suspect(X).
37. ← subset.
38. ← rebutted_suspect.