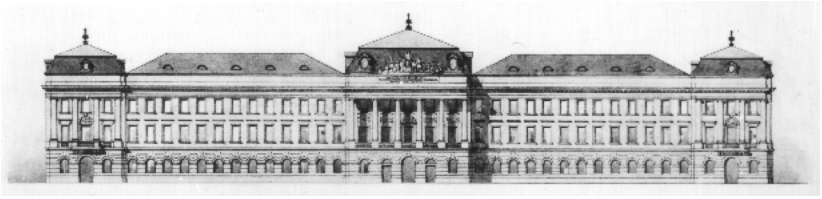
dbai

TECHNICAL REPORT



INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG DATENBANKEN UND ARTIFICIAL INTELLIGENCE

Implementing Abstract Argumentation A Survey

DBAI-TR-2013-82

Günther Charwat **Wolfgang Dvořák**
Sarah A. Gaggl **Johannes P. Wallner**
Stefan Woltran

DBAI TECHNICAL REPORT

2013

Institut für Informationssysteme
Abteilung Datenbanken und
Artificial Intelligence
Technische Universität Wien
Favoritenstr. 9
A-1040 Vienna, Austria
Tel: +43-1-58801-18403
Fax: +43-1-58801-18493
sekret@dbai.tuwien.ac.at
www.dbai.tuwien.ac.at

TU WIEN
TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DBAI TECHNICAL REPORT
DBAI TECHNICAL REPORT DBAI-TR-2013-82, 2013

Implementing Abstract Argumentation – A Survey

Günther Charwat [1] **Wolfgang Dvořák** [2] **Sarah A. Gaggl** [3]
Johannes P. Wallner [1] **Stefan Woltran** [1]

Abstract. Within the last decade, abstract argumentation has emerged as a central field in Artificial Intelligence. Besides serving as a core formalism for many advanced argumentation systems, this is mainly due to the fact that abstract argumentation has been shown to capture several nonmonotonic logics and other AI related principles. Although the idea of abstract argumentation is appealingly simple, several reasoning problems in this formalism suffer from a high computational complexity. This calls for advanced techniques when it comes to implementation issues, a challenge which has been recently faced from different angles. In this survey, we give an overview on different methods for solving abstract argumentation problems and compare their particular features. Moreover, we give links to available state-of-the-art systems for abstract argumentation, which put these methods to practice.

[1]Technische Universität Wien. E-mail: {gcharwat,wallner,woltran}@dbai.tuwien.ac.at
[2]Universität Wien. E-mail: wolfgang.dvorak@univie.ac.at
[3]Technische Universität Dresden. E-mail: sarah.gaggl@tu-dresden.de

Acknowledgements: This work has been supported by the Vienna Science and Technology Fund (WWTF) under grant ICT08-028, by the Austrian Science Fund (FWF) under grant P20704-N18, and by the Vienna University of Technology special fund "Innovative Projekte" (9006.09/008).

Contents

1 Introduction

Argumentation is a highly interdisciplinary field with links to psychology, linguistics, philosophy, legal theory, and formal logic. Since the advent of the computer age, formal models of argument have been materialized in different systems that implement — or at least support — creation, evaluation, and judgment of arguments. However, until Dung's seminal paper on *abstract argumentation* [51], the heterogeneity of these approaches was severely hampering a strong and joint development of a field like "computational argumentation". In fact, Dung's idea of evaluating arguments on an abstract level by taking only their inter-relationships into account, not only has been shown to underlie many of the earlier approaches for argumentation, but also uniformly captures several nonmonotonic logics. Not at least this second contribution located Argumentation as a sub-discipline of Artificial Intelligence [19] that has gained more and more significance over the last 15 years. This is witnessed by the biannual COMMA Conference on Computational Models of Argument[1], the IJCAI Workshop Series on Theory and Applications of Formal Argumentation (TAFA)[2], the 2010 established Journal of Argument and Computation[3], or the Textbook on *Argumentation in Artificial Intelligence* [117].

One particular feature of abstract argumentation frameworks is their simple structure. In fact, abstract argumentation frameworks are just directed graphs where vertices play the role of arguments and edges indicate a certain conflict between the two connected arguments. These argumentation frameworks are usually derived during an *instantiation process* (see, e.g., [22, 40]) where structured arguments are investigated with respect to their ability to contradict other such arguments; the actual notion of "contradicting" can be instantiated in many different forms (see, e.g., [94]). Having generated the framework in such a way, the process of "conflict-resolution", i.e., the search for jointly acceptable sets of arguments, is then delegated to semantics which operate on the abstract level. Thus, semantics for argumentation frameworks have also been referred to as *calculi of opposition* [30].

One direction of research in abstract argumentation was devoted to develop the "right" semantics (see, e.g., [9, 10, 11] where properties for argumentation semantics are proposed and evaluated). This has lead to what G. Simari has called a "*plethora of argumentation semantics*".[4] Today there seems to be agreement within the community that different semantics suite different applications, hence many of them are in use for a variety of application domains[5]. It is clear that this situation implies that successful systems for abstract argumentation are expected to offer not only a single semantics.

The central role of abstract argumentation frameworks also boosted the research for efficient procedures for this particular formalism. However, it was soon recognized that already these simple frameworks show high complexity (see, e.g., [49, 56, 72]); due to the link to nonmonotonic logic

[1] `http://www.comma-conf.org/`

[2] `http://homepages.abdn.ac.uk/n.oren/pages/TAFA-13/`

[3] `http://www.tandfonline.com/toc/tarc20/current`

[4] During the presentation of [105] at COMMA 2006.

[5] However, in connection with particular instantiation schemes, it is often claimed that only semantics which follow the principle of admissibility (arguments shall only be jointly accepted if each of the selected arguments is defended by the selected set; we will make the concepts more clear in Section 2) should be considered (see, e.g., [40]).

and to logic programming in particular, this came without a huge surprise. Together with the fact that many different semantics exist, general implementation methods for abstract argumentation thus require

- a certain level of generality, such that not only a single semantics can be treated; and
- a sufficient level of efficiency to face the high inherent complexity of the problem.

Scope of the Survey. In this article, we present a selection of evaluation methods for abstract argumentation which we believe to meet these requirements. We group the methods into two categories, the *reduction approach* and the *direct approach*.

The underlying idea of the *reduction approach* is to exploit existing efficient software which has originally been developed for other purposes. To this end, one has to formalize the problems she has in mind within other formalisms like constraint-satisfaction [47], propositional logic [23] or answer-set programming (ASP) [32]. In this setting, the resulting argumentation systems directly benefit from the high level of sophistication today's system for SAT (satisfiability in propositional logic) or ASP have reached. The reduction approach will be presented in detail in Section 3 of this article. Hereby, we will first focus on

- *SAT-based* argumentation systems. This direction has been advocated by Besnard and Doutre [20], and later extended by means of quantified propositional logic [3, 77]. We will first discuss the theoretical underpinnings of this approach and then continue with an introduction to the CEGARTIX system [65] which relies on iterative calls to SAT solvers for argumentation semantics of high complexity (i.e., being located on the second level of the polynomial hierarchy). The other reduction method we shall discuss in detail is the
- *ASP-based* approach. The use of this logic-programming paradigm to solve abstract argumentation problems has been initiated by several authors (the survey article by Toni and Sergot [123] provides a good overview). We focus here on the ASPARTIX approach [76] which in contrast to the aforementioned SAT methods relies on a query-based implementation where the argumentation framework to be evaluated is provided as an input database (from this point of view, the SAT methods can been seen as a compiler-like approach to abstract argumentation, while the ASP method acts like an interpreter). A large collection of such ASP queries is provided by the ASPARTIX system, which also offers a web front-end. We will discuss standard ways of ASP encodings, but also some recent methods which exploit advanced ASP techniques [63].

In the remainder of Section 3 we shall present the concepts behind other reduction-based approaches, for instance, with constraint satisfaction problems (CSP) as the target language, which lead to the development of the ConArg system [25].

In Section 4, we collect methods and algorithms which have been developed from scratch (instead of using another formalism like SAT or ASP). While the obvious disadvantage of this *direct approach* is that existing technology cannot be directly employed, such argumentation-tailored algorithms ease the incorporation of short-cuts that are specific to the argumentation domain. In detail, we will discuss the following ideas:

- The fundamental *labeling approach* [50, 107, 111, 124] gives a more fine-grained handle for the status of arguments when evaluated w.r.t. semantics and thus also provides a solid basis for dedicated algorithms. We present two different approaches for enumerating preferred extensions, one along the lines of [107] and another following [50] using improvements from [111]. Further we discuss an algorithm dedicated to credulous reasoning with preferred semantics following the work of [124]. Labeling-based algorithms have been materialized in the ArguLab system as well as in Verheij's COMPARG system.

- Characterizations via *Dialogue Games*. Here the acceptance status of an argument is given in terms of winning strategies in certain games on the argumentation framework. Typically such games are two player games where one player, the proponent, argues in favor of the argument in question and a second player, the opponent, argues against it. Such games can be used to design algorithms [107, 122] which are employed in systems like Dungine and Dung-O-Matic.

- Finally, we will have a look on *dynamic programming approaches* [69] which operate on decompositions of frameworks. Here, the main feature is that running times are not mainly dependent on the size of the given framework, but on a structural parameter. We focus here on the parameter tree-width and the concept of tree decomposition. This method was first advocated by Dunne in [54] and realized in the dynPARTIX system [44].

As we have already hinted above, many of the methods we present have found their way into an available software system. To this end, in this survey we will not only explain these methods, but also shall give the interested reader pointers to concrete tools which can be used to experiment.

The empirical evaluation of these systems is out of scope for this survey. Some of the systems have been evaluated and compared w.r.t. their performance (see e.g., [63, 111]), but no exhaustive performance comparisons have been done. In fact, an organized competition comparable to the ones from the areas of SAT [96] or ASP [35] have not yet been established.

To summarize, our goal with this article is to introduce a selection of methods for evaluating abstract argumentation systems; we shall explain the key concepts in detail for selected semantics and give pointers to the literature for the remaining semantics or when it comes to more subtle aspects like optimization. Concerning abstract argumentation itself, we only give a concise introduction in Section 2. For readers not familiar with abstract argumentation, we highly recommend the recent survey article by Baroni *et al.* [6].

Since the focus of this article is on the evaluation of semantics for Dung's abstract argumentation framework, advanced systems including instantiation (e.g., ASPIC [116] and Carneades [93]), assumption-based argumentation [52], or systems based on defeasible logic [87] are out of the scope of this article[6]. Likewise, we will not consider the vast collection of extensions to Dung's frameworks like value-based [18], bipolar [43], extended [106], constrained [2], temporal [34], practical [97], and fibring argumentation frameworks [81], as well as argumentation frameworks

[6] An overview on these approaches is given in [119].

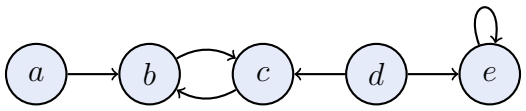


Figure 1: Example argumentation framework

with recursive attacks [7], argumentation context systems [31], and abstract dialectical frameworks [33]. We also exclude abstract argumentation with uncertainty or weights here; recent articles by Hunter [95] and respectively Dunne *et al.* [58] introduce these variants in detail.

2 Background

In this section we introduce (abstract) argumentation frameworks [51] and recall the semantics we study in this paper (see also [6, 11]).

Definition 1. *An* argumentation framework (AF) *is a pair $F = (A, R)$ where A is a set of arguments and $R \subseteq A \times A$ is the attack relation. The pair $(a, b) \in R$ means that a attacks b. We say that an argument $a \in A$ is* defended *(in F) by a set $S \subseteq A$ if, for each $b \in A$ such that $(b, a) \in R$, there exists a $c \in S$ such that $(c, b) \in R$.*

An argumentation framework can be represented as a directed graph.

Example 1. *Let $F = (A, R)$ be an AF with $A = \{a, b, c, d, e\}$ and $R = \{(a, b), (b, c), (c, b), (d, c), (d, e), (e, e)\}$. The corresponding graph representation is depicted in Fig. 1.*

Semantics for argumentation frameworks are given via a function σ which assigns to each AF $F = (A, R)$ a set $\sigma(F) \subseteq 2^A$ of extensions.

We consider for σ the functions $naive$, stb, adm, com, grd, prf, sem and stg which stand for naive, stable, admissible, complete, grounded, preferred, semi-stable and stage extensions, respectively. Towards the definition of these semantics we introduce a few more formal concepts.

Definition 2. *Given an AF $F = (A, R)$, the* characteristic function $\mathcal{F}_F : 2^A \to 2^A$ *of F is defined as $\mathcal{F}_F(S) = \{x \in A \mid x \text{ is defended by } S\}$. For a set $S \subseteq A$ and an argument $a \in A$, we write $S \rightarrowtail^R a$ (resp. $a \rightarrowtail^R S$) in case there is an argument $b \in S$, such that $(b, a) \in R$ (resp. $(a, b) \in R$). Moreover, for a set $S \subseteq A$, we denote the set of arguments attacked by S as $S^{\oplus}_R = \{x \mid S \rightarrowtail^R x\}$, and resp. $S^{\ominus}_R = \{x \mid x \rightarrowtail^R S\}$, and define the* range of S as $S^{+}_R = S \cup S^{\oplus}_R$ *and the* negative range of S as $S^{-}_R = S \cup S^{\ominus}_R$

Definition 3. *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is* conflict-free (in F)*, if there are no $a, b \in S$, such that $(a, b) \in R$. $cf(F)$ denotes the collection of conflict-free sets of F. For a conflict-free set $S \in cf(F)$, it holds that*

- *$S \in naive(F)$, if there is no $T \in cf(F)$ with $T \supset S$;*

- $S \in stb(F)$, *if* $S_R^+ = A$;
- $S \in adm(F)$, *if* $S \subseteq \mathcal{F}_F(S)$;
- $S \in com(F)$, *if* $S = \mathcal{F}_F(S)$;
- $S \in grd(F)$, *if* $S \in com(F)$ *and there is no* $T \in com(F)$ *with* $T \subset S$;
- $S \in prf(F)$, *if* $S \in adm(F)$ *and there is no* $T \in adm(F)$ *with* $S \subset T$;
- $S \in sem(F)$, *if* $S \in adm(F)$ *and there is no* $T \in adm(F)$ *with* $S_R^+ \subset T_R^+$;
- $S \in stg(F)$, *if there is no* $T \in cf(F)$, *with* $S_R^+ \subset T_R^+$.

We recall that for each AF F, the grounded semantics yields a unique extension, the grounded extension, which is the least fixed-point of the characteristic function $\mathcal{F}_F$.

Example 2. *Consider the AF from Example 1. Then:* $cf(F) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}, \{b, d\}\}$; $naive(F) = \{\{a, c\}, \{a, d\}, \{b, d\}\}$; $adm(F) = \{\emptyset, \{a\}, \{d\}, \{a, d\}\}$; *and* $stb(F) = com(F) = grd(F) = prf(F) = sem(F) = stg(F) = \{\{a, d\}\}$.

Labeling-based semantics. So far we have considered so-called extension-based semantics. However, there are several approaches defining argumentation semantics via certain kind of labelings instead of extensions. As an example we consider the popular approach by Caminada and Gabbay [41] and in particular their complete labelings. Basically such a labeling is a three-valued function that assigns one of the labels in, out and $undec$ to each argument with the intuition behind these labels being the following. An argument is labeled with: in if it is accepted; out if there are strong reasons to reject it, i.e., it is attacked by an accepted argument; $undec$ if the argument is undecided, i.e., neither accepted nor attacked by accepted arguments. We denote labeling functions $\mathcal{L}$ also by triples $(\mathcal{L}_{in}, \mathcal{L}_{out}, \mathcal{L}_{undec})$, where $\mathcal{L}_{in}$ is the set of arguments labeled by in, $\mathcal{L}_{out}$ is the set of arguments labeled by out and $\mathcal{L}_{undec}$ is the set of arguments labeled by $undec$.

As an example, we give the definition of complete labelings from [41].

Definition 4. *Given an AF* $F = (A, R)$, *a function* $\mathcal{L} : A \to \{in, out, undec\}$ *is a* complete labeling *iff the following conditions hold:*

- $\mathcal{L}(a) = in$ *iff for each* b *with* $(b, a) \in R$, $\mathcal{L}(b) = out$
- $\mathcal{L}(b) = out$ *iff there exists* b *with* $(b, a) \in R$, $\mathcal{L}(b) = in$

There is a one-to-one mapping between complete extensions and complete labelings, such that the set of arguments labeled with in corresponds to the complete extension and the arguments labeled with out correspond to the arguments attacked by the complete extension. Having complete labelings at hand we can also characterize preferred labelings as follows:

Definition 5. *Given an AF* $F = (A, R)$. *The* preferred labelings *are those complete labelings where* $\mathcal{L}_{in}$ *is* $\subseteq$*-maximal among all complete labelings.*

Right by the definitions, we have the same one-to-one mapping between preferred extensions and preferred labelings as for complete semantics. Making this one-to-one mapping formal one can use it to define labeling-based versions for all of our semantics (see [41]), but this is out of the scope of this survey.

Reasoning in Argumentation Frameworks. Extensions of AFs for a semantics are in general not unique, in fact most semantics yield multiple extensions, therefore reasoning in such frameworks offers different possibilities to cope with this behavior. We recall here the most important reasoning modes: Given an argumentation framework F and a semantics σ, the first mode is called $\mathsf{Enum}_\sigma(F)$, which results in an enumeration of all extensions. A simpler notion is $\mathsf{Count}_\sigma(F)$, which only counts the number of extensions. Query-based modes are $\mathsf{Cred}_\sigma(a, F)$ and $\mathsf{Skept}_\sigma(a, F)$ for deciding credulous (respectively skeptical) acceptance of an argument a. The former returns *yes* if a is contained in at least one extension under σ, while for the latter to return *yes* a must be contained in all extensions under σ. Finally we also consider the problem $\mathsf{Ver}_\sigma(S, F)$ of verifying a given extension, i.e., testing whether a given set S is a σ-extension of F, typically occurring as a subroutine of a reasoning procedure.

Definition 6. *Given an AF $F = (A, R)$, a semantics σ and an argument $a \in A$ then*

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

Example 3. *The reasoning problems for Example 1 result in $\mathsf{Enum}_{\mathsf{naive}}(F) = \big\{\{a, c\}, \{a, d\}, \{b, d\}\big\}$ and $\mathsf{Count}_{\mathsf{naive}}(F) = 3$. We could ask the following queries for the argument a: $\mathsf{Cred}_{\mathsf{naive}}(a, F) =$ yes and $\mathsf{Skept}_{\mathsf{naive}}(a, F) =$ no. If we consider preferred semantics we just get one extension $\mathsf{Enum}_{\mathsf{prf}}(F) = \big\{\{a, d\}\big\}$, $\mathsf{Count}_{\mathsf{prf}}(F) = 1$ and thus credulous and skeptical acceptance coincide, e.g., $\mathsf{Cred}_{\mathsf{prf}}(a, F) = \mathsf{Skept}_{\mathsf{prf}}(a, F) =$ yes.*

Next let us turn to the complexity of reasoning in abstract argumentation frameworks. We assume the reader has knowledge about standard complexity classes, i.e., P, NP and L (logarithmic space). Furthermore we briefly recapitulate the concept of oracle machines and related complexity classes. Let $\mathcal{C}$ denote some complexity class. By a $\mathcal{C}$-oracle machine we mean a (polynomial time) Turing machine which can access an oracle that decides a given (sub)-problem in $\mathcal{C}$ within one step.

Table 1: Computational complexity of reasoning in AFs.

σ	Cred_σ	Skept_σ	Ver_σ
$naive$	in L	in L	in L
stb	NP-c	coNP-c	in L
adm	NP-c	trivial	in L
com	NP-c	P-c	in L
grd	P-c	P-c	P-c
prf	NP-c	Π_2^P-c	coNP-c
sem	Σ_2^P-c	Π_2^P-c	coNP-c
stg	Σ_2^P-c	Π_2^P-c	coNP-c

We denote such machines as $\text{NP}^{\mathcal{C}}$ if the underlying Turing machine is nondeterministic. The class $\Sigma_2^P = \text{NP}^{\text{NP}}$ thus denotes the set of problems which can be decided by a nondeterministic polynomial time algorithm that has (unrestricted) access to an NP-oracle. The class $\Pi_2^P = \text{coNP}^{\text{NP}}$ is defined as the complementary class of Σ_2^P, i.e., $\Pi_2^P = \text{co}\Sigma_2^P$. The relation between the complexity classes is as follows:

$$\text{L} \subseteq \text{P} \subseteq \begin{matrix}\text{NP}\\ \text{coNP}\end{matrix} \subseteq \begin{matrix}\Sigma_2^P\\ \Pi_2^P\end{matrix}$$

The computational complexity of credulous and skeptical reasoning is extensively studied in the literature (see [59] for a starting point). Table 1 summarizes the computational complexity classifications of the defined decision problems [46, 49, 51, 56, 57, 72, 73, 86], where an entry $\mathcal{C}$-c denotes that the corresponding problem is complete for class $\mathcal{C}$.

3 Reduction-based Approaches

In this section we will discuss reduction-based approaches in abstract argumentation. Implied by the name, these methods reduce or translate a problem to another. From a computational point of view we require that this reduction is efficiently computable, i.e., achievable in polynomial time and that the new problem instance gives the same answer as the original one. Such methods offer the great benefit of exploiting existing and highly sophisticated solvers for well-known and well-studied problem domains.

Naturally reduction-based methods can be differentiated by the target system. In the literature many approaches have been studied for abstract argumentation ranging from propositional logic [3, 20, 65, 77], answer-set programming (ASP) [76, 110, 113, 126] and equational systems [83, 84] to Constraint Satisfaction Problems (CSP) [1, 24]. We will give an overview of these approaches and

in particular focus on the former two very prominent target systems, the reductions to propositional logic and ASP.

3.1 Propositional-Logic based Approach

Propositional logic is the prototypical target system for many approaches based on reductions, as the Boolean SAT problem is well studied and moreover accompanied with many mature and efficient solvers such as MiniSat [74] and GRASP [104].

First we recall the necessary background of Boolean logic and quantified Boolean formulae (QBF) since they serve as our target systems.

The basis of propositional logic is a set of propositional variables $\mathcal{P}$, to which we also refer to as atoms. Propositional formulae are built as usual from the connectives $\wedge, \vee, \rightarrow$ and $\neg$, denoting the logical conjunction, disjunction, (material) implication and negation respectively. As for truth constants, we use $\top$ for the value true and $\bot$ for false. In addition we consider quantified Boolean formulae with the universal quantifier $\forall$ and the existential quantifier $\exists$, that is, given a formula ϕ, then $Qp\phi$ is a QBF, with $Q \in \{\forall, \exists\}$ and $p \in \mathcal{P}$. Further $Q\{p_1, \ldots, p_n\}\phi$ is a shortcut for $Qp_1 \cdots Qp_n\phi$. The order of variables in consecutive quantifiers of the same type does not matter.

A propositional variable p in a QBF ϕ is free if it does not occur within the scope of a quantifier Qp and bound otherwise. If ϕ contains no free variable, then ϕ is said to be closed and otherwise open. Further we will write $\phi[p/\psi]$ to denote the result of uniformly substituting each free occurrence of p with ψ in the formula ϕ.

An interpretation $I \subseteq \mathcal{P}$ defines for each propositional variable a truth assignment where $p \in I$ indicates that p evaluates to true. This generalizes as usual for arbitrary formulae: Given a formula ϕ and an interpretation I, then ϕ evaluates to true under I (I satisfies ϕ) if one of the following holds, with $p \in \mathcal{P}$

- $\phi = p$ and $p \in I$
- $\phi = \neg p$ and $p \notin I$
- $\phi = \psi_1 \wedge \psi_2$ and both ψ_1 and ψ_2 evaluate to true under I
- $\phi = \psi_1 \vee \psi_2$ and one of ψ_1 and ψ_2 evaluates to true under I
- $\phi = \psi_1 \rightarrow \psi_2$ and ψ_1 evaluates to false or ψ_2 evaluates to true under I
- $\phi = \exists p\psi$ and one of $\psi[p/\top]$ and $\psi[p/\bot]$ evaluates to true under I
- $\phi = \forall p\psi$ and both $\psi[p/\top]$ and $\psi[p/\bot]$ evaluate to true under I.

If an interpretation I satisfies a formula ϕ, denoted by $I \models \phi$, we then say that I is a model of ϕ.

The approaches in Section 3.1.1 and Section 3.1.2 share the basic idea of translating a given AF, a semantics and a reasoning mode to a propositional formula, thereby reducing the problem to Boolean logic. In general this works by either inspecting the models of the resulting formula, which

are in correspondence to the extensions of the AF, or deciding whether a formula is satisfiable or unsatisfiable, to solve query-based reasoning. Note that we restrict ourselves here to the semantics which we consider to be sufficient for illustrating the main concepts. In general, the approaches can be applied to many other semantics.

3.1.1 Reductions to Propositional Logic

The first reduction-based approach [20, 77] we consider here encodes the problem of finding admissible sets via a propositional logic formula without quantifiers. Given an AF $F = (A, R)$, for each argument $a \in A$ a propositional variable v_a is constructed. Then $S \subseteq A$ is an extension under semantics σ iff $\{v_a \mid a \in S\} \models \phi$, with ϕ being a propositional formula that evaluates AF F under semantics σ (we will present below in detail how to translate AFs into formulas). Formally the correspondence between sets of extensions and models of a propositional formula can be defined as follows.

Definition 7. *Let $\mathcal{S} \subseteq 2^A$ be a collection of sets of arguments and let $\mathcal{I} \subseteq 2^{\mathcal{P}}$ be a collection of interpretations. We say that $\mathcal{S}$ and $\mathcal{I}$ correspond to each other, in symbols $\mathcal{S} \cong \mathcal{I}$, if*

1. *for each $S \in \mathcal{S}$, there exists an $I \in \mathcal{I}$, such that $\{a \mid v_a \in I, a \in A\} = S$;*
2. *for each $I \in \mathcal{I}$, it holds that $\{a \mid v_a \in I, a \in A\} \in \mathcal{S}$; and*
3. *$|\mathcal{S}| = |\mathcal{I}|$.*

The formula for reducing the problem of finding admissible sets of an AF (A, R) is built as follows.

$$adm_{A,R} := \bigwedge_{a \in A} ((v_a \rightarrow \bigwedge_{(b,a) \in R} \neg v_b) \wedge (v_a \rightarrow \bigwedge_{(b,a) \in R} (\bigvee_{(c,b) \in R} v_c)) \tag{1}$$

The models of formula $adm_{A,R}$ now correspond to the admissible sets of an AF $F = (A, R)$, i.e., $\mathsf{Enum}_{\mathsf{adm}}(F) \cong \{M \mid M \models adm_{A,R}\}$. The first line in (1) ensures that the resulting set of arguments is conflict-free, that is, whenever we accept an argument a (i.e., v_a evaluates to true under a model) then all its attackers cannot be selected anymore. The second line expresses the defense of arguments by stating that, when we accept a, then for all its attackers b, some defender c must be accepted as well. Note that an empty conjunction is treated as $\top$, whereas the empty disjunction is treated as $\bot$.

Example 4. *The propositional formula for admissible sets of the framework $F = (A, R)$ in Exam-*

ple 1 is given by

$$
\begin{aligned}
adm_{A,R} = &((v_a \rightarrow \top) \wedge && (2)\\
&(v_b \rightarrow (\neg v_a \wedge \neg v_c)) \wedge && (3)\\
&(v_c \rightarrow (\neg v_b \wedge \neg v_d)) \wedge && (4)\\
&(v_d \rightarrow \top) \wedge && (5)\\
&(v_e \rightarrow (\neg v_d \wedge \neg v_e))) \wedge && (6)\\
&((v_a \rightarrow \top) \wedge && (7)\\
&(v_b \rightarrow (\bot \wedge (v_b \vee v_d))) \wedge && (8)\\
&(v_c \rightarrow ((v_a \vee v_c) \wedge \bot)) \wedge && (9)\\
&(v_d \rightarrow \top) \wedge && (10)\\
&(v_e \rightarrow (\bot \wedge d))) && (11)
\end{aligned}
$$

The subformulae in lines 2 to 5 encode the conflict-free property, while lines 6 to 11 take care of defense of arguments. Consider for instance argument b. For ensuring that no conflicts occur, line 3 specifies that if we accept b we cannot accept a and c anymore. Likewise for admissibility we need a defender for all attackers of b. This is handled in line 8. For the attacker c we require either b itself or d to be accepted, but since a is not attacked, there is no model of $adm_{A,R}$ where v_b evaluates to true.

Another interesting translation to capture semantics of AFs within propositional logic is done by [82]. Here a deeper correspondence between AFs and propositional logic is shown via the Peirce–Quine dagger connective "$\downarrow$", which expresses $a \downarrow b \equiv \neg a \wedge \neg b$. The translation of AFs to propositional logic is a product of this correspondence. The basic idea is that if a set of arguments X attack an argument a, then we can apply, in a first step, a general form of the connective: $\Downarrow X = \bigwedge_{x \in X} \neg x$. Using certain mechanisms to cope with cycles, one can establish a faithful translation using either two-valued or three-valued interpretations depending on the semantics.

3.1.2 Reductions to Quantified Boolean Formulae

Problems beyond NP require a more expressive formalism than Boolean logic. Preferred semantics, for example, are based on subset-maximal admissible (or complete) sets. Intuitively, we can achieve this by computing the admissible sets and additionally checking that there is no proper superset which is also admissible. In order to express subset maximality directly inside the logic a universal (or, equivalently, a negated existential) quantifier is needed, making quantified Boolean formulae a well-suited formalism. It is possible to specify this in QBFs either via extension-based or labeling-based semantics.

For the former we encode the maximality check with an auxiliary formula. For convenience we denote by $A' = \{a' \mid a \in A\}$ the set of renamed arguments in A. Likewise we define a renaming for the attack relation: $R' = \{(a', b') \mid (a, b) \in R\}$. The following defines a useful shorthand for comparing to interpretations (i.e. sets of arguments) with respect to the subset-relation.

$$A < A' := \bigwedge_{a \in A} (v_a \rightarrow v_{a'}) \wedge \neg \bigwedge_{a' \in A'} (v_{a'} \rightarrow v_a) \tag{12}$$

In other words, this formula ensures that any model $M \models (A < A')$ satisfies $\{a \in A \mid v_a \in M\} \subset \{a \in A \mid v_{a'} \in M\}$. Now we can state the QBF for preferred extensions. Let the quantified variables be $A'_v = \{v_{a'} \mid a' \in A'\}$.

$$prf_{A,R} := adm_{A,R} \ \wedge \neg \exists A'_v((A < A') \wedge adm_{A',R'}) \tag{13}$$

Now for an arbitrary AF $F = (A, R)$ its preferred extensions are in a 1-to-1 correspondence to the models of $prf_{A,R}$, i.e., $\mathsf{Enum}_{\mathsf{prf}}(F) \cong \{M \mid M \models prf_{A,R}\}$. We simply check if the accepted arguments form an admissible set and if there exists a proper superset of it which is also admissible. If the latter does not exist, then we have found a preferred extension.

The second approach is based on complete labelings (see Definition 4) instead of extensions [3]. To this end we look at four-valued interpretations to express more than one state for each argument. In addition to the truth values true and false we have undecided and inconsistent. The three labelings in, out and undecided correspond to the former three truth values. The whole approach can be encoded in classical two-valued QBFs. Hereby the truth value of $p \in \mathcal{P}$ is encoded in $p^{\oplus}$ and $p^{\ominus}$. Now every classical two-valued interpretation assigns values to these two atoms as usual. For two variables we have four different cases, which correspond to the four truth values, i.e., $\{p^{\oplus}, p^{\ominus}\} \subseteq I$ is interpreted as assigning inconsistent to p, true (resp. false) is assigned to p if only $p^{\oplus}$ (resp. $p^{\ominus}$) is in I and the last case stands for undecided, if neither $p^{\oplus}$ nor $p^{\ominus}$ is in I.

For preferred semantics the encoding is more complex than (13), but the ideas are similar. We begin with formulae for the four truth values. Note that we slightly adapted the representation and formulae from [3] to better match the previous encodings, but the important concepts remain the same.

$$val(p, v) := \begin{cases} p^{\oplus} \wedge p^{\ominus} & \text{if } v = i \\ p^{\oplus} \wedge \neg p^{\ominus} & \text{if } v = t \\ \neg p^{\oplus} \wedge p^{\ominus} & \text{if } v = f \\ \neg p^{\oplus} \wedge \neg p^{\ominus} & \text{if } v = u \end{cases} \tag{14}$$

The formula $val(p, v)$ encodes the four possible truth values for a virtual atom p. Actually instead of p the auxiliary atoms $p^{\oplus}$ and $p^{\ominus}$ are present in the concrete formula. With the right choice of negation we can explicitly refer to the desired four-valued truth value of p on a sort of meta-level. Using this concept, we can proceed to the labeling formula for each argument in an AF $F = (A, R)$.

$$lab^t_{A,R}(a) := \ val(v_a, t) \rightarrow \bigwedge_{(b,a)\in R} val(v_b, f) \tag{15}$$

$$lab^f_{A,R}(a) := \ val(v_a, f) \rightarrow \bigvee_{(b,a)\in R} val(v_b, t) \tag{16}$$

$$lab^u_{A,R}(a) := \ val(v_a, u) \rightarrow \big((\neg \bigwedge_{(b,a)\in R} val(v_b, f)) \ \wedge (\neg \bigvee_{(b,a)\in R} val(v_b, t))\big) \tag{17}$$

Indeed, these formulae reflect Definition 4: The formulae (15), (16) and (17) encode the in, out and undecided labelings, respectively. For example (15) can be interpreted in the following way: If an argument a is set to true, then all its attackers must be false. (16) can be interpreted similarly, except that if an atom denotes that an argument is false, then one of its attackers must be true. The third formula, (17) now says that for any argument to which we assign undecided, it cannot be the case that all its attackers are false or one of them is true.

Three values are sufficient to reflect the three labelings. To avoid problems with the fourth truth value (inconsistent), we exclude it from occurring in the evaluation by the coherence formula.

$$coh_A := \bigwedge_{a\in A} \neg val(v_a, i) \tag{18}$$

Now complete extensions are characterized by the following formula. We will write an L as superscript in $com^L_{A,R}$ to denote that this formula handles labelings instead of extensions.

$$com^L_{A,R} := coh_A \wedge \bigwedge_{a\in A} (lab^t_{A,R}(a) \wedge lab^f_{A,R}(a) \wedge lab^u_{A,R}(a)) \tag{19}$$

The formula $com^L_{A,R}$ expresses that all the arguments are assigned either true, false or undecided. Furthermore for each of these three truth values we have a formula denoting that, for example, if a is true, then all its attackers must be false. Using this, one can encode complete labelings and hence complete extensions. Now preferred extensions, or labelings, are expressed as before by subset maximization with a similar construct. The difference is again in the four-valued interpretation encoded in QBFs.

$$\begin{aligned} A <^L A' := & \bigwedge_{a\in A} (val(v_a, t) \rightarrow val(v_{a'}, t)) \wedge \\ & \neg \bigwedge_{a'\in A'} (val(v_{a'}, t) \rightarrow val(v_a, t)) \end{aligned} \tag{20}$$

Then, as before, the preferred extensions, or their labelings, can be encoded with a QBF as follows, with the quantified atoms $A'_v = \{v^{\oplus}_{a'}, v^{\ominus}_{a'} \mid a' \in A'\}$.

$$prf^L_{A,R} := \ com^L_{A,R} \ \wedge \neg\exists A'_v((A <^L A') \wedge com^L_{A',R'}) \tag{21}$$

For an AF $F = (A, R)$ the following notion of correspondence holds: Let the projection of atoms evaluated to true under the four-valued interpretation be $M^t = \{p \mid p^{\oplus} \in M, p^{\ominus} \notin M\}$, then $\mathsf{Enum}_{\mathsf{prf}}(F) \cong \{M^t \mid M \models prf^L_{A,R}\}$. Note that $prf^L_{A,R}$ differs from $prf_{A,R}$ not only by using a labeling-based approach, but also by maximizing complete labelings rather than admissible sets.

Utilizing the added expressive power of quantifiers and the labeling approach, the authors of [3] also encode a range of other semantics, for instances semi-stable reasoning, where one can use the same idea as outlined above, but instead of maximizing the arguments that are in, the arguments that are labeled undecided are minimized.

This gives a general system for encoding many semantics, but one has to be careful with choosing the right target system. For example, grounded semantics can easily be specified in this formalism using a QBF, but computing the grounded extension can be done using an algorithm with polynomial running time. Thus an appropriate encoding would yield a QBF from a fragment which is known to be efficiently decidable, for instance, 2-QBF (the generalization of Krom forumalae to QBFs). However, we are not aware of any work which deals with such "complexity-sensitive" encodings in terms of QBFs.

3.1.3 Iterative Application of SAT Solvers

The last propositional-logic based approach we outline here is specifically well-suited for deciding credulous and skeptical acceptance of arguments. It is based on iterative applications of model checking for propositional formulae and has been presented in [65]. The idea is to use an algorithm which constructs a formula and a query, and based on the outcome modifies the query or generates new ones until a final decision is reached. This is in contrast to so-called monolithic encodings, which formulate the whole problem in a single formula. The iterative approach is suitable when the problem to be solved cannot in general be decided by the satisfiability of a single propositional formula (constructible in polynomial time) without quantifiers. This means that instead of reducing the problem to a single QBF formula, we delegate the solving task to an algorithm querying a SAT solver multiple times.

This approach aims at problems for which in general an exponential number of calls to the SAT solver is required. However, this can be avoided if the given AF satisfies certain properties and can thus be solved more efficiently. Questions of skeptical preferred acceptance can be solved with a number of SAT calls dependent on the number of preferred extensions of the given AF, see [65].

The basis of the algorithm is to compute the preferred extensions by falling back to a simpler semantics. For the preferred extensions we can compute admissible sets and iteratively extend them, which is achieved by iterative calls to the SAT solver. We will now briefly sketch this approach.

Algorithm 1 decides skeptical acceptance under preferred semantics of an argument a in an AF F. We discuss the algorithm description and the intuition behind the formulae, which are checked for satisfiability. The idea is to proceed from one preferred extension to the next and verifying that the argument a is in this extension. This is encoded in the main while loop, lines 2 to 11. The models of the formula ϕ represent the remaining admissible sets in the current state of the algorithm. In the beginning ϕ encodes all admissible sets of F (init(ϕ)). We start with an

Algorithm 1 $\mathsf{Skept}_{prf}(a, F)$

Require: AF $F = (A, R)$, argument $a \in A$,
Ensure: returns *yes* iff a is skeptically accepted under preferred semantics

```
1: init(φ)
2: while ∃I, I ⊨ φ do
3:    while ∃I', I' ⊨ ψ^I ∧ ¬v_a do
4:       I ← I'
5:    end while
6:    if ∄I, I ⊨ ψ^I then
7:       no
8:    else
9:       φ ← φ ∧ γ^I
10:   end if
11: end while
12: yes
```

admissible set and iteratively extend it while making sure that a is not accepted in this admissible set, via the second loop in lines 3 to 5 and adding $\neg v_a$ to the query. The formula ψ^I incorporates the model I and states that a model of it must still correspond to a admissible set, but also has to be a superset of the current one, specified by I.

If we cannot add arguments to the admissible set anymore, then we check if we could extend it with having a inside, in line 6. If this is the case, then every preferred extension, which is a superset of the current admissible set contains a and hence we can proceed to a different admissible extension not containing a. In case we cannot add a to the admissible set, we have found a preferred extension without a, hereby refuting its skeptical acceptance in F. In the former case the next iteration is ensured in line 9 by strengthening the main query ϕ by adding γ^I, stating that at least one argument currently not accepted in I must be accepted from now on. This states that we are not interested anymore in subsets of the current admissible set.

Example 5. *For the AF F from Example 1 we can check the skeptical acceptance of b. The condition of the first loop is satisfied as there exist the following admissible sets $\emptyset$, $\{a\}$, $\{d\}$, $\{a, d\}$ in F. Say we pick $\emptyset$. The second while loop then creates a subset maximal admissible set (excluding b) by two iterations, say first adding a and then d. As $\{a, d\}$ is now subset maximal, the second loop terminates. Since this set can also not be extended if we allow to also accept b, this must be a preferred extension. This means we refute the skeptical acceptance of b.*

3.1.4 Reasoning Problems

The first two presented reductions in Section 3.1.1 and Section 3.1.2 solve immediately questions of enumerating extensions. Deciding credulous and skeptical reasoning is typically easy to achieve. In order to decide $\mathsf{Cred}_\sigma(a, F)$ one can conjunctively add a to the formula. If there exists a model, a is credulously accepted. Adding negated a to the formula decides if a is not skeptically accepted,

i.e., if there exists a model then an extension does not contain a. Similarly one can switch to credulous reasoning for the algorithm presented in Section 3.1.3, by adding the atom to be queried positively instead of negatively.

Counting the number of extensions cannot be easily encoded in the formulae, but the SAT solver itself may offer this feature by counting the number of models.

3.1.5 Implementations

We conclude this section with a few words about concrete instantiations of such systems. Although SAT/QBF solvers [23, 114] nowadays provide very efficient solutions for solving hard problems like the typical problems arising in abstract argumentation, only the approach in Section 3.1.3 was implemented in the form of CEGARTIX. This system is available on the web[7] and focuses on problems hard for the second level of the polynomial hierarchy, namely acceptance under preferred, semi-stable and stage semantics. We note that, although the reductions to propositional logic lack fully implemented software systems, these approaches can be very quickly instantiated by essentially providing a parser, which rewrites AFs from the chosen input language to the formulae in the Boolean language of a SAT/QBF solver.

3.2 ASP-based Approach

Answer set programming (ASP, for short) [103, 109], also known as A-Prolog [5, 90], is a declarative problem solving paradigm, rooted in logic programming and non-monotonic reasoning. Due to continuous refinements over the last decade answer-set solvers (e.g., [89, 100]) nowadays not only support a rich language but also are capable of solving hard problems efficiently. Furthermore, the declarative approach of ASP leads to readable and maintainable code (compared to C code, for instance) thus allowing to define the problems at hand in a natural way.

Solving problems in abstract argumentation via ASP has been studied by several authors (see [123] for a survey), including the approach proposed by Nieves *et al.* [110] where the program is re-computed for every input instance, Wakaki and Nitta [126] which uses labeling-based semantics (see Section 2) and the approach by Egly *et al.* [76] which follows extension-based semantics. We focus here on the latter since this approach is put into practice by the ASPARTIX system which supports a wide range of different semantics and additionally offers a web frontend.

In the following we first give a brief introduction to ASP. We then present how the computation of admissible and preferred extensions can be encoded in ASP. In order to obtain preferred extensions it is necessary to check for subset-maximality of admissible sets. We sketch two approaches for this in ASP, one based directly on a certain *saturation technique* [78] (which is unfortunately hardly accessible for non-experts in ASP) and a second one which makes use of `metasp` encodings [88] (allowing to specify subset minimization via a single simple statement). Additionally we briefly discuss how reasoning problems can be specified and give links to current implementations of the ASP-based approach.

[7] `http://www.dbai.tuwien.ac.at/research/project/argumentation/cegartix/`

3.2.1 Answer Set Programming

We give a brief overview of the syntax and semantics of disjunctive logic programs under the answer-set semantics [91]; for further background, see [79, 100].

We fix a countable set $\mathcal{U}$ of *(domain) elements*, also called *constants*; and suppose a total order $<$ over the domain elements. An *atom* is an expression $p(t_1, \ldots, t_n)$, where p is a *predicate* of arity $n \geq 0$ and each t_i is either a variable or an element from $\mathcal{U}$. An atom is *ground* if it is free of variables. $B_\mathcal{U}$ denotes the set of all ground atoms over $\mathcal{U}$.

A *(disjunctive) rule* r with $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$ is of the form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_k, \ not\, b_{k+1}, \ldots, \ not\, b_m$$

where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms, and "not" stands for *default negation*. An atom a is a positive literal, while $not\, a$ is a default negated literal. The *head* of r is the set $H(r) = \{a_1, \ldots, a_n\}$ and the *body* of r is $B(r) = B^+(r) \cup B^-(r)$ with $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$. A rule r is *normal* if $n \leq 1$ and a *constraint* if $n = 0$. A rule r is *safe* if each variable in r occurs in $B^+(r)$. A rule r is *ground* if no variable occurs in r. A *fact* is a ground rule without disjunction and with an empty body. An *(input) database* is a set of facts. A program is a finite set of safe disjunctive rules. For a program π and an input database D, we often write $\pi(D)$ instead of $D \cup \pi$. If each rule in a program is normal (resp. ground), we call the program normal (resp. ground). Besides disjunctive and normal programs, we consider here the class of optimization programs, i.e., normal programs which additionally contain $\#minimize$ statements

$$\#minimize[l_1 = w_1@J_1, \ldots, l_k = w_k@J_k] \tag{22}$$

where l_i is a literal, w_i an integer weight and J_i an integer priority level.

For any program π, let U_π be the set of all constants appearing in π. $Gr(\pi)$ is the set of rules $r\tau$ obtained by applying, to each rule $r \in \pi$, all possible substitutions τ from the variables in r to elements of U_π. An *interpretation* $I \subseteq B_\mathcal{U}$ *satisfies* a ground rule r iff $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. I satisfies a ground program π, if each $r \in \pi$ is satisfied by I. A non-ground rule r (resp., a program π) is satisfied by an interpretation I iff I satisfies all groundings of r (resp., $Gr(\pi)$). $I \subseteq B_\mathcal{U}$ is an *answer set* of π iff it is a subset-minimal set satisfying the *Gelfond-Lifschitz reduct* $\pi^I = \{H(r) \leftarrow B^+(r) \mid I \cap B^-(r) = \emptyset, r \in Gr(\pi)\}$. For a program π, we denote the set of its answer sets by $\mathcal{AS}(\pi)$.

For semantics of optimization programs, we interpret the $\#minimize$ statement w.r.t. subset-inclusion: For any sets X and Y of atoms, we have $Y \subseteq^w_J X$, if for any weighted literal $l = w@J$ occurring in (22), $Y \models l$ implies $X \models l$. Then, M is a collection of relations of the form $\subseteq^w_J$ for priority levels J and weights w. A standard answer set (i.e., not taking the minimize statements into account) Y of π *dominates* a standard answer set X of π w.r.t. M if there are a priority level J and a weight w such that $X \subseteq^w_J Y$ does not hold for $\subseteq^w_J \in M$, while $Y \subseteq^{w'}_{J'} X$ holds for all $\subseteq^{w'}_{J'} \in M$ where $J' \geq J$. Finally a standard answer set X is an answer set of an optimization program π w.r.t. M if there is no standard answer set Y of π that dominates X w.r.t. M.

3.2.2 ASP in Argumentation

We now provide fixed queries for admissible and preferred extensions in such a way that the AF F is given as an input database $\widehat{F}$ and the answer sets of the combined program $\pi_e(\widehat{F})$ are in a certain one-to-one correspondence with the respective extensions, where $e \in \{adm, prf\}$. For an AF $F = (A, R)$, we define

$$\widehat{F} = \{\ \mathrm{arg}(a) \mid a \in A\} \cup \{\mathrm{att}(a,b) \mid (a,b) \in R\ \}.$$

We have to guess candidates for the selected type of extensions and then check whether a guessed candidate satisfies the corresponding conditions, where default negation is an appropriate concept to formulate such a guess within a query. In what follows, we use unary predicates $\mathrm{in}(\cdot)$ and $\mathrm{out}(\cdot)$ to perform a guess for a set $S \subseteq A$, where $\mathrm{in}(a)$ represents that $a \in S$.

Similarly as in Definition 7 we define the following notion of correspondence which is relevant for our purposes.

Definition 8. *Let $\mathcal{S} \subseteq 2^{\mathcal{U}}$ be a collection of sets of domain elements and let $\mathcal{I} \subseteq 2^{B_\mathcal{U}}$ be a collection of sets of ground atoms. We say that $\mathcal{S}$ and $\mathcal{I}$ correspond to each other, in symbols $\mathcal{S} \cong \mathcal{I}$, iff (i) for each $S \in \mathcal{S}$, there exists an $I \in \mathcal{I}$, such that $\{a \mid \mathrm{in}(a) \in I\} = S$; (ii) for each $I \in \mathcal{I}$, it holds that $\{a \mid \mathrm{in}(a) \in I\} \in \mathcal{S}$; and (iii) $|\mathcal{S}| = |\mathcal{I}|$.*

In what follows, we will stepwise introduce the rules from which our queries will be built. Let $F = (A, R)$ be an argumentation framework. The following program fragment guesses, when augmented by $\widehat{F}$, any subset $S \subseteq A$ and then checks whether the guess is conflict-free in F:

$$\begin{aligned}\pi_{cf} = \{\ & \mathrm{in}(X) \leftarrow \mathit{not}\ \mathrm{out}(X), \mathrm{arg}(X);\\ & \mathrm{out}(X) \leftarrow \mathit{not}\ \mathrm{in}(X), \mathrm{arg}(X);\\ & \leftarrow \mathrm{in}(X), \mathrm{in}(Y), \mathrm{att}(X,Y)\ \}.\end{aligned}$$

The program module π_{adm} for the admissibility test is as follows:

$$\begin{aligned}\pi_{adm} = \pi_{cf} \cup \{ & \mathrm{defeated}(X) \leftarrow \mathrm{in}(Y), \mathrm{att}(Y,X);\\ & \leftarrow \mathrm{in}(X), \mathrm{att}(Y,X), \mathit{not}\ \mathrm{defeated}(Y)\ \}.\end{aligned}$$

Sometimes we have to avoid the use of negation. This might either be the case for the saturation technique described below or if a simple program can be solved without a Guess&Check approach, e.g. for grounded semantics. Then, encodings typically rely on a form of loops where all domain elements are visited and it is checked whether a desired property holds for all elements visited so far. We will use this technique in our saturation-based encoding. For this purpose, an order $<$ over the domain elements (usually provided by common ASP solvers) is used together with a few helper predicates defined in the program $\pi_<$ below; in fact, predicates $\mathrm{inf}/1, \mathrm{succ}/2$ and $\mathrm{sup}/1$ denote infimum, successor and supremum of the order $<$.

$$\pi_{<} = \{\ \mathrm{lt}(X,Y) \leftarrow \mathrm{arg}(X), \mathrm{arg}(Y), X < Y;$$
$$\mathrm{nsucc}(X,Z) \leftarrow \mathrm{lt}(X,Y), \mathrm{lt}(Y,Z);$$
$$\mathrm{succ}(X,Y) \leftarrow \mathrm{lt}(X,Y), \mathit{not}\ \mathrm{nsucc}(X,Y);$$
$$\mathrm{ninf}(Y) \leftarrow \mathrm{lt}(X,Y);$$
$$\mathrm{inf}(X) \leftarrow \mathrm{arg}(X), \mathit{not}\ \mathrm{ninf}(X);$$
$$\mathrm{nsup}(X) \leftarrow \mathrm{lt}(X,Y);$$
$$\mathrm{sup}(X) \leftarrow \mathrm{arg}(X), \mathit{not}\ \mathrm{nsup}(X)\ \}.$$

3.2.3 Saturation Encodings

To compute the preferred extensions of an argumentation framework, we will use the saturation technique as follows: Having computed an admissible extension S (characterized via predicates $\mathrm{in}(\cdot)$ and $\mathrm{out}(\cdot)$ using our encoding $\pi_{adm}(\widehat{F})$), we perform a second guess using new predicates, say $\mathrm{inN}(\cdot)$ and $\mathrm{outN}(\cdot)$, to represent a further guess $T \supset S$. In order to check whether the first guess characterizes a preferred extension, we have to ensure that *no* guess of the second form (i.e., via $\mathrm{inN}(\cdot)$ and $\mathrm{outN}(\cdot)$) characterizes an admissible extension. The saturation module $\pi_{satpref}$ looks as follows.

$$\pi_{satpref} = \{\ \mathrm{inN}(X) \vee \mathrm{outN}(X) \leftarrow \mathrm{out}(X); \tag{23}$$
$$\mathrm{inN}(X) \leftarrow \mathrm{in}(X); \tag{24}$$
$$\mathrm{fail} \leftarrow \mathrm{eq}; \tag{25}$$
$$\mathrm{fail} \leftarrow \mathrm{inN}(X), \mathrm{inN}(Y), \mathrm{att}(X,Y); \tag{26}$$
$$\mathrm{fail} \leftarrow \mathrm{inN}(X), \mathrm{outN}(Y), \mathrm{att}(Y,X), \mathrm{undefeated}(Y); \tag{27}$$
$$\mathrm{inN}(X) \leftarrow \mathrm{fail}, \mathrm{arg}(X); \tag{28}$$
$$\mathrm{outN}(X) \leftarrow \mathrm{fail}, \mathrm{arg}(X); \tag{29}$$
$$\leftarrow \mathit{not}\ \mathrm{fail}\ \}. \tag{30}$$

Let us for the moment also assume that predicates eq (rule (25)) and $\mathrm{undefeated}(\cdot)$ (rule (27)) are defined (we give the additional rules for those predicates below in the modules π_{eq} and $\pi_{undefeated}$) and provide the following information:

- eq is derived if the guess S via $\mathrm{in}(\cdot)$ and $\mathrm{out}(\cdot)$ equals the second guess T via $\mathrm{inN}(\cdot)$ and $\mathrm{outN}(\cdot)$; in other words, eq is derived if $S = T$;
- $\mathrm{undefeated}(a)$ is derived if argument a is not defeated in F by the second guess T.

In what follows, we discuss the functioning of $\pi_{satpref}$ when conjoined with the program $\pi_{adm}(\widehat{F})$ for a given AF F. First, rule (23) guesses a set $T \subseteq A$ as already discussed above. Rule (24) ensures that the new guess satisfies $S \subseteq T$.

The task of the rules (25)–(27) is to check whether the new guess T is a proper superset of S and characterizes an admissible extension of the given AF F. If this is not the case, we derive the predicate fail. More specifically, rule (25) checks whether $S = T$, and in case this holds we derive fail; rule (26) checks whether T is not conflict-free in F, and in case this holds we derive fail; rule (27) checks whether T contains an argument not defended by T in F, and in case this holds we derive fail. In other words, we have not derived fail if $T \supset S$ and T is admissible in F. By definition, S then cannot be a preferred extension of F.

The remaining rules (28)–(30) saturate the guess in case fail was derived, and finally ensure that fail has to be in an answer set.

Let us illustrate now the behavior of $\pi_{satpref}$ for two scenarios. First, suppose the first guess S (via predicates $\mathrm{in}(\cdot)$ and $\mathrm{out}(\cdot)$) is a preferred extension of the given AF $F = (A, R)$. Hence, for each $T \supset S$, T is not admissible. But then, we have that *every* new guess T (via predicates $\mathrm{inN}(\cdot)$ and $\mathrm{outN}(\cdot)$) derives fail. Thus we have no interpretation, without predicate fail, which satisfies $\pi_{satpref}$. However, the saturated interpretation, which contains fail and both $\mathrm{inN}(a)$ and $\mathrm{outN}(a)$ for each $a \in A$, does satisfy the program and also becomes an answer set of the program.

Now suppose, the first guess S (via predicates $\mathrm{in}(\cdot)$ and $\mathrm{out}(\cdot)$) is an admissible but not a preferred extension of the given AF F. Then there exists a set $T \supset S$, such that T is admissible in F. If we consider the interpretation I characterizing T (i.e., we have $\mathrm{inN}(a) \in I$, for each $a \in T$, and $\mathrm{outN}(a) \in I$, for each $a \in A \setminus T$), then I does not contain fail and satisfies the rules (23)–(29). But this shows that we cannot have an answer set J which characterizes S. Due to rule (30) such an answer set J has to contain fail and by rules (28) and (29), J contains both $\mathrm{inN}(a)$ and $\mathrm{outN}(a)$ for each $a \in A$. Note that we thus have $I \subset J$ (if I and J characterize the same initial guess S). Moreover, I satisfies the reduct of our program with respect to J. This can be seen by the fact that the only occurrence of default negation is in rule (30). In other words, there is an $I \subset J$ satisfying the reduct and thus J cannot be an answer set. This however, is as desired, since the initial guess S characterized by J is not a preferred extensions.

We still have to define the rules for the predicates eq and $\mathrm{undefeated}(\cdot)$. Basically, these predicates would be easy to define, but as we have seen in the discussion above, default negation plays a central role in the saturation technique (recall the functioning of $\leftarrow \mathit{not}\ \mathrm{fail}$). We therefore have to find encodings which suitably define the required predicates only with a limited use of negation. In fact, we are only allowed to have stratified negation in these modules. Thus, both predicates eq and $\mathrm{undefeated}(\cdot)$ are computed predicates via predicates $\mathrm{eq\_upto}(\cdot)$ (resp., $\mathrm{undefeated\_upto}(\cdot,\cdot)$) in

the modules π_{eq} and $\pi_{undefeated}$, which are defined as follows.

$$
\begin{aligned}
\pi_{eq} = \{\ & \mathrm{eq\_upto}(X) \leftarrow \mathrm{inf}(X), \mathrm{in}(X), \mathrm{inN}(X);\\
& \mathrm{eq\_upto}(X) \leftarrow \mathrm{inf}(X), \mathrm{out}(X), \mathrm{outN}(X);\\
& \mathrm{eq\_upto}(X) \leftarrow \mathrm{succ}(Y,X), \mathrm{in}(X), \mathrm{inN}(X), \mathrm{eq\_upto}(Y);\\
& \mathrm{eq\_upto}(X) \leftarrow \mathrm{succ}(Y,X), \mathrm{out}(X), \mathrm{outN}(X), \mathrm{eq\_upto}(Y);\\
& \mathrm{eq} \leftarrow \mathrm{sup}(X), \mathrm{eq\_upto}(X)\ \};\\
\pi_{undefeated} = \{\ & \mathrm{undefeated\_upto}(X,Y) \leftarrow \mathrm{inf}(Y), \mathrm{outN}(X), \mathrm{outN}(Y);\\
& \mathrm{undefeated\_upto}(X,Y) \leftarrow \mathrm{inf}(Y), \mathrm{outN}(X), \mathit{not}\ \mathrm{att}(Y,X);\\
& \mathrm{undefeated\_upto}(X,Y) \leftarrow \mathrm{succ}(Z,Y), \mathrm{undefeated\_upto}(X,Z),\\
& \qquad\qquad\qquad\qquad \mathrm{outN}(Y);\\
& \mathrm{undefeated\_upto}(X,Y) \leftarrow \mathrm{succ}(Z,Y), \mathrm{undefeated\_upto}(X,Z),\\
& \qquad\qquad\qquad\qquad \mathit{not}\ \mathrm{att}(Y,X);\\
& \mathrm{undefeated}(X) \leftarrow \mathrm{sup}(Y), \mathrm{undefeated\_upto}(X,Y)\ \}.
\end{aligned}
$$

With these predicates at hand, we can now formally define the module for preferred extensions,

$$\pi_{prf} = \pi_{adm} \cup \pi_{<} \cup \pi_{eq} \cup \pi_{undefeated} \cup \pi_{satpref}.$$

Then, for any AF F, the answer sets of $\pi_{prf}(\widehat{F})$ are in a one-to-one correspondence with the preferred extensions of F.

3.2.4 `metasp` Encodings

The following encodings for preferred semantics are written using the $\#minimize$ statement when evaluated with the subset-minimization semantics provided by `metasp` [88]. For our encodings we do not need prioritization and weights, therefore these are omitted (i.e., set to default) in the minimization statements. The minimization technique is realized through meta programming techniques, which themselves are answer set programs. This works as follows: The ASP encoding to solve is given to the grounder `gringo` which reifies the program, i.e., outputs a ground program consisting of facts, which represent the rules and facts of the original input encoding. The grounder is then again executed on this output with the meta programs which encode the optimization. Finally, `claspD` computes the answer sets. Note that here we use the version of `clasp` which supports disjunctive rules. Therefore for a program π and an AF F we have the following execution.

```
gringo --reify π(F̂) | gringo - {meta.lp,metaO.lp,metaD.lp} \
       <(echo "optimize(1,1,incl).") | claspD 0
```

Here, `meta.lp`, `metaO.lp` and `metaD.lp` are the encodings for the minimization statement. The statement `optimize(incl,1,1)` indicates that we use subset inclusion for the optimization technique using priority and weight 1.

We now look at the encodings for the preferred semantics which are easy to encode using the minimization statement of `metasp`. We only need the module π_{adm} and minimize the $\mathrm{out}/1$ predicate. This in turn gives us the subset-maximal admissible extensions which captures the definition of preferred semantics.

$$\pi_{prf\_metasp} = \pi_{adm} \cup \{\#minimize[\mathrm{out}]\}.$$

Now it follows directly that, for any AF F, the answer sets of $\pi_{prf\_metasp}(\widehat{F})$ are in a one-to-one correspondence with the preferred extensions of F.

3.2.5 Reasoning Problems

As with other reduction-based approaches, the types of reasoning available depend on the ASP solver. Many of them feature enumeration of all solutions, as well as counting them and also credulous and skeptical reasoning. For the `metasp` variant, the meta encodings can be augmented with constraints to achieve credulous and skeptical reasoning.

3.2.6 Implementations

On the implementation side we mention the system ASPARTIX as a large collection of ASP encodings for abstract argumentation[8]. All encodings are fixed and the instance of an AF is given as input. The encodings from the system ASPARTIX are written in the general ASP syntax. It may be the case that one needs to adapt the encodings for some ASP solvers. The `metasp` encodings can only be performed with `gringo/claspD`. All semantics mentioned in this article are incorporated in ASPARTIX and `metasp` encodings are available for preferred, semi-stable and stage semantics. Furthermore, there exists a web-application of the system[9]. This is a user friendly tool which allows to run ASPARTIX without the need of downloading or installing any ASP solver or encodings. The platform is directly accessible from the web with any standard browser and provides a graphical representation of the input framework and the solutions.

3.3 Further Reduction-based Approaches

In the following we summarize further approaches for reduction-based methods.

3.3.1 Equational Approaches

Equational approaches for abstract argumentation map the given reasoning problem at hand to a set of equations. Solutions of such equations then directly represent solutions of the original problems. One such approach is proposed in [83, 84]. Here one has a system of equations where each argument is represented by a distinct variable and a domain of real numbers in the interval $[0, 1]$. Solutions to these systems of equations map to each variable a number from the domain. If

[8] `http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage/`
[9] `http://rull.dbai.tuwien.ac.at:8080/ASPARTIX/`

the variable of an argument a is mapped to 1, a is accepted and otherwise not. From this one can easily read off the extensions by inspecting the variable assignments. In order to represent an AF and a semantics, different kinds of equations are constructed. Conflict-freeness, for example, can be achieved by introducing for each argument an equation that incorporates its attackers.

3.3.2 Utilizing CSP

An approach that is inherently related to propositional logic reductions is based on using CSPs as the target system [1, 24, 25]. A CSP can generally be described by a triple (X, D, C), where X is the set of variables, D the possible domain and lastly a set C of constraints, which specify legal values for the variables. Again each variable is associated with an argument. Now the authors of [24] specify constraints for several concepts, like the conflict-free property or admissibility. For the former a constraint of the form $\neg(a_i = 1 \wedge a_j = 1)$ is introduced if there is an attack from a_i to a_j or vice versa. This constraint, here written in a logical style with the symbol $\wedge$ being the conjunction and $\neg$ the negation, specifies that not both of the variables may be set to 1. If the overall domain for each variable is restricted to 0 and 1, then such constraints result in solutions which correspond to conflict-free sets of the original AF. To capture admissibility, a different kind of constraint is introduced. First, if an argument a has no potential defender, i.e., there is no attacker for its attackers, then it will be automatically rejected by the constraint $a = 0$. Otherwise, the constraint $\neg(a_i = 1 \wedge a_{g1} = 0 \wedge \cdots \wedge a_{gk} = 0)$ specifies that the argument a_i can not be accepted if none of its defenders $a_{g1}, \ldots, a_{gk}$ is accepted as well. For ensuring maximality constraints the approach builds on utilizing a second CSP, e.g., finding first admissible sets and then via the second CSP finding preferred extensions. This approach was implemented in the tool ConArg[10] which also features several semantics, among them the grounded, complete, preferred, stage, semi-stable and stable semantics.

3.3.3 Monadic Second Order Logic

A reduction approach going beyond pure propositional logic is encoding the reasoning problems in monadic second order logic (MSO). In this expressive predicate logic we may quantify over variables and unary predicates. Given such an MSO formula and an interpretation I, the task is to check if I is a model of the formula. In [71] the authors encode several reasoning tasks for AFs into an MSO formula φ. A given AF is then transformed into an interpretation I and one decides the reasoning task by testing whether I is a model of φ. The work [71] introduces certain building blocks for such encodings, which enable straightforward reductions of the different semantics to MSO. While the first MSO-encodings for abstract argumentation [55, 72] were introduced to obtain complexity-theoretic results in terms of tree-width, the advent of efficient systems for MSO [26, 99] turns MSO-encodings into an interesting alternative to implement abstract argumentation via the reduction method.

[10] `http://www.dmi.unipg.it/francesco.santini/argumentation/conarg.zip`

4 Direct Approaches

In the previous section, we exhaustively discussed different reduction-based approaches for implementing abstract argumentation. But what about implementing procedures for abstract argumentation from scratch? While such an approach definitely requires more effort in implementation, it allows to access the framework directly, without having the overhead of transformation (and as a result a potential loss of structural information). Even more important, compared to the reduction approach, direct algorithms allow for an easy incorporation of short-cuts that are specific for the argumentation domain.

In the reduction-based approach the distinction between computing all extensions and performing specific reasoning tasks is often delegated to the reasoner of the target formalism and thus can be neglected. When using direct approaches we have to take care (and advantage) of specific reasoning tasks ourselves. Hence in this section we will distinguish more explicitly between algorithms for enumerating all extensions and, for instance, algorithms that are specially tailored for computing "witnesses" for certain queries.

Nowadays the most successful approaches for direct algorithms can be categorized in three groups. First there are so called *labeling-based algorithms* [36, 107, 124], which build on alternative characterizations for argumentation semantics using certain labeling functions of arguments. Second, we consider dialectical argument games, i.e., games played by two players alternating their arguments and where winning strategies ultimately characterize the acceptance status of an argument. Finally there are *dynamic programming algorithms*, which are based on graph decompositions and results from (parameterized) complexity analysis. In the following we present each of these approaches in detail.

4.1 Labeling-based Algorithms

The class of labeling-based algorithms builds on the concept of argument labelings, with probably the most prominent variant being the 3-valued labelings due to Caminada and Gabbay [41]. For the formal definitions of complete and preferred labelings we refer to Section 2 (Definitions 4 & 5).

First labeling-based algorithms have been proposed in [50]; many further materializations of this concept can be found in the literature (see, e.g., [107, 111, 124]). The central observation underlying all these approaches is the following: Whenever one fixes the label of one argument this has immediate implications for the possible labels of the neighbors of this argument. For instance if we are interested in complete labelings and label an argument a with in then all neighbors of a must be labeled out.

In what follows, we focus on labeling-based algorithms for preferred semantics and distinguish between two classes: (i) algorithms which aim to enumerate all preferred extensions of a given AF; and (ii) algorithms that are tailored to perform specific reasoning tasks like skeptical and credulous reasoning.

4.1.1 Enumerating Extensions

For enumerating extensions one can, in principle, simply enumerate all possible sets and check whether they are extensions. In general this is of course a quite inefficient approach. Therefore, labeling-based algorithms typically use a particular backtracking strategy to enumerate possible labelings, fixing the label of one argument in each step. In addition to the simple backtracking strategy, in each step the information of the new label is propagated to the neighbors of the argument. So, for instance, if we set an argument to in then all it predecessors must be labeled out. The different approaches to labeling-based algorithms have their own strategy for selecting the next arguments to be labeled as well as for the rules they apply for propagating labels. Algorithm 2 is an example for a labeling-based algorithm for computing preferred labelings in the spirit of [107].

Algorithm 2 pref-lab(F)

Require: AF $F = (A, R)$, a labeling $\mathcal{L}$
$S_{\mathcal{L}}$ global variable with candidate labelings
Ensure: $S_{\mathcal{L}}$ is the set of preferred labelings
1: $S_{\mathcal{L}} = \{(\emptyset, \emptyset, A)\}$, $\mathcal{L} = \langle A, \emptyset, \emptyset \rangle$
2: pref-lab($F, \mathcal{L}$)
3: **function** pref-lab($F, \mathcal{L}$)
4: **if** $\exists a \in \mathcal{L}_{in} : \exists b \notin \mathcal{L}_{out} : (b, a) \in R$ **then**
5: **for** all $a \in \mathcal{L}_{in}$ s.t. $\exists b \notin \mathcal{L}_{out} : (b, a) \in R$ **do**
6: set $\mathcal{L}' = \mathcal{L}$
7: set $\mathcal{L}'(a) = out$
8: **for** $y \in \{a\}^+$ **do**
9: **if** $y \notin \mathcal{L}'^{+}_{in}$ **then**
10: set $\mathcal{L}'(y) = undec$
11: **end if**
12: **end for**
13: pref-lab($F, \mathcal{L}'$)
14: **end for**
15: **else**
16: **for** $\mathcal{L}' \in S_{\mathcal{L}}$ **do**
17: **if** $\mathcal{L}_{in} \subseteq \mathcal{L}'_{in}$ **then**
18: break
19: **else if** $\mathcal{L}'_{in} \subset \mathcal{L}_{in}$ **then**
20: $S_{\mathcal{L}} = S_{\mathcal{L}} \setminus \{\mathcal{L}'\}$
21: $S_{\mathcal{L}} = S_{\mathcal{L}} \cup \{\mathcal{L}\}$
22: **end if**
23: **end for**
24: **end if**
25: **endFunction**

The main idea of Algorithm 2 is to start with the labeling that marks all arguments with in (the

set containing all arguments) and relabeling arguments to either out or $undec$ until the set becomes admissible. This strategy of considering candidates prevents the algorithm from considering all the (relatively small) admissible sets for being preferred extension like other algorithms do (compare Algorithm 3).

Let us explain Algorithm 2 in more detail. When applying the algorithm to an AF $F = (A, R)$, it first initializes the labeling $\mathcal{L}$ such that each argument is labeled with in, i.e., $\mathcal{L}_{in} = A$, and the set $S_{\mathcal{L}}$ of candidate solutions only contains the labeling $(\emptyset, \emptyset, A)$, corresponding to the empty set. Then in each step the algorithm picks an argument a which is labeled in but is not defended, i.e., there is a neighbor that is not labeled out, and relabels it. We call such a relabel step a transition step. In Algorithm 2 a transition step is due to the following rules. First the argument a is labeled to out and then all arguments in $y \in \{a\}^+$ are checked for being valid labeled out arguments, i.e., we test whether there is an argument labeled in and attacking y, and if not it is labeled $undec$. In [107] it is shown that each preferred extension can be obtained from the initial labeling that labels each argument to in by a finite sequence of such transition steps and further that each terminated sequence (which is indeed finite) corresponds to an admissible set.

This simple algorithm has several weaknesses which have been addressed in the literature. First consider line 5 of Algorithm 2. For each $a \in \mathcal{L}_{in}$ s.t. $\exists b \notin \mathcal{L}_{out} : (b, a) \in R$ one starts a transition and then recursively calls the procedure. This causes the branching in the search procedure and thus we want to minimize the number of arguments to be considered here. To this end [107] introduces a notion of so-called super-illegal arguments which form a subset of the above mentioned arguments and can be relabeled first without branching in the algorithm. That is, in case there is at least one super-illegal argument the algorithm first considers all of them (in arbitrary order) before branching among the other arguments. However, even with this improvement, it can happen that several branches of the algorithm may produce the same candidate extension. For instance consider the AF $F = (\{a, b, c\}, \{(a, b), (b, a), (b, c), (c, b)\})$. The preferred labeling $\langle\{b\}, \{a, c\}, \emptyset\rangle$ will be produced by two branches of the algorithm, by the branch choosing a in the first step (and assign out to a) and then choosing c in the second step (and assigning out as well), but also by the branch selecting c first and then a in the second step. As such duplicates are indeed a waste of computational resources this is a weak point. Other algorithms [50, 111] avoid such duplicates as they use a different strategy to branch in the search space (see, for instance, Algorithm 3).

Next consider lines 16-23 in the algorithm. This part ensures the $\subseteq$-maximality of the labelings in $S_{\mathcal{L}}$. As the set $S_{\mathcal{L}}$ can be of exponential size (even if the number of preferred labelings is small) testing whether a new candidate is $\subseteq$-maximal and updating the set $S_{\mathcal{L}}$ is costly. Hence alternative approaches to deal with $\subseteq$-maximality have been proposed. Firstly, [50] used a criterion for maximality that does not make use of the other extensions explicitly exploiting the observation that a complete labeling $\mathcal{L}$ is a preferred labeling iff there is no subset S of $\mathcal{L}_{undec}$ such that the set $\mathcal{L}_{in} \cup S$ is admissible. In particular for candidate labelings where all arguments are labeled either in or out this avoids an explicit check of maximality (such labelings corresponds to stable extensions). Secondly, in [111] the authors provide a smart traversal of the search space such that one can avoid deleting sets from $S_{\mathcal{L}}$, i.e., in each step one can decide whether the current candidate is preferred or not, by only using previously computed preferred labelings (see Algorithm 3).

Let us thus have a closer look on Algorithm 3 next. This algorithm for preferred semantics

Algorithm 3 pref-lab(F)

Require: AF $F = (A, R)$
$S_{\mathcal{L}}$ global variable with admissible labelings
Ensure: $S_{\mathcal{L}}$ is the set of preferred labelings
1: $S_{\mathcal{L}} = \emptyset$, $\mathcal{L} = \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$
2: pref-lab$(F, \mathcal{L})$
3: **function** pref-lab$(F, \mathcal{L})$
Require: a 4-valued labeling $\mathcal{L}$
4: **if** there is an unlabeled argument $a \in A$ **then**
5: $a =$ first unlabeled argument
6: **if** $\mathcal{L}_{in} \cup \{a\} \in cf(F)$ **then**
7: $\mathcal{L}'_{in} = \mathcal{L}_{in} \cup \{a\}$, $\mathcal{L}'_{out} = \mathcal{L}_{out} \cup \mathcal{L}'^{+}_{in}$
8: $\mathcal{L}'_{att} = (\mathcal{L}_{att} \cup \mathcal{L}'^{-}_{in}) \setminus \mathcal{L}'_{out}$
9: pref-lab$(F, \langle \mathcal{L}'_{in}, \mathcal{L}'_{out}, \mathcal{L}'_{att}, \mathcal{L}_{undec} \rangle)$
10: **end if**
11: pref-lab$(F, \langle \mathcal{L}_{in}, \mathcal{L}_{out}, \mathcal{L}_{att}, \mathcal{L}_{undec} \cup \{a\} \rangle)$
12: **else**
13: **if** $\mathcal{L}_{att} = \emptyset$ **then**
14: **if** $\mathcal{L}_{in}$ $\subseteq$-max among $\{\mathcal{L}_{in} \mid \mathcal{L} \in S_{\mathcal{L}}\}$ **then**
15: $S_{\mathcal{L}} = S_{\mathcal{L}} \cup \{\mathcal{L}\}$
16: **end if**
17: **end if**
18: **end if**
19: **endFunction**

follows the work of [50] and [111]. The main difference to Algorithm 2 is the way the search space is explored. Algorithm 3 starts with an unlabeled graph and in each step one (unlabeled) argument is considered branching now among the possible labels for this argument. Once a label is chosen it is never changed again and thus no labeling can be produced twice. Another apparent difference is that the algorithm uses four labels instead of just three labels. We denote such a four valued labeling $\mathcal{L}$ as quadruple $\langle \mathcal{L}_{in}, \mathcal{L}_{out}, \mathcal{L}_{att}, \mathcal{L}_{undec} \rangle$.[11] The intuition behind the labels in, out are the same as for three valued labelings while arguments which are labeled att are arguments that attack an in labeled argument but are not attacked by such an argument and arguments which are labeled $undec$ have no conflict with in labeled arguments.

This algorithm iterates over all admissible sets and tests whether they are $\subseteq$-maximal. As for each argument a the algorithm first tries to add an argument to $\mathcal{L}_{in}$ before considering the variant without a, we can be sure that supersets are always considered first. Hence we never have to remove a labeling from the set $S_{\mathcal{L}}$. The pitfall of Algorithm 3 is the potential exponential number of admissible labelings (even for a small number of preferred extensions) which are all considered

[11] Notice that we use different names for the labels compared to the original works [50, 111]. This is to present several algorithms in this survey in a uniform and easy comparable way.

by the algorithm.

Let us briefly compare Algorithm 2 and Algorithm 3 on the two extreme cases of (i) the AF $F_1 = (A, A \times A)$ with the total attack relation and (ii) the AF $F_2 = (A, \emptyset)$ with the empty attack relation. The empty set is the only admissible and thus also the only preferred extension of F_1. As there is just one admissible set, Algorithm 3 never branches and thus terminates after a linear number of steps. However Algorithm 2 has to update all arguments to $undec$. As this can be done in an arbitrary order we have $n!$ many branches producing the same extension. For F_2 there is just one preferred extension but Algorithm 3 considers all $2^{|A|}$ admissible sets. In contrast Algorithm 2 terminates immediately. As different labeling-based algorithms behave good on different kind of argumentation frameworks, empirical evaluations are an important issue. A first such evaluation of different labeling-based algorithms for preferred semantics is provided in [111].

Here we have only considered the case of preferred semantics, but for most of the semantics labeling-based algorithms have been proposed in the literature: an algorithm for grounded semantics is given in [107]; an algorithm for admissible labelings can be easily obtained from Algorithm 3 (by dropping the $\subseteq$-maximality test in Line 14); for complete semantics one can adapt Algorithm 2; for stable semantics, see [107]; algorithms for semi-stable and stage semantics can be found in [36, 38, 107].

4.1.2 Reasoning Problems

Having an algorithm for enumerating all extensions of an AF at hand one can immediately use them to answer reasoning problems by simple testing each extension for the queried argument. However, this is probably not the most efficient way. Given that we are only interested in the acceptance of a certain argument we might directly try to produce a witness (or counter-example) for this argument instead of computing all extensions. In this section we discuss dedicated algorithms for reasoning problems. As an example we review the work of Verheij [124] on a credulous acceptance algorithm for preferred semantics.

The idea behind the algorithm is that we start with the argument (or the set of arguments) for which we test credulous acceptance and iteratively add arguments to defend all arguments in our sets. The outlined Algorithm 4 starts with labeling the queried argument with in and all the other arguments with $undec$. Then it iterates the following two steps. First, check if the set $\mathcal{L}_{in}$ is conflict free and if so label all arguments attacking $\mathcal{L}_{in}$ with out. Otherwise terminate the branch of the algorithm. In the second step, for each argument a which is labeled out but not attacked by an argument labeled in, we pick an unlabeled argument b attacking a and label it with in. In the case there are several such arguments we start a new branch of the algorithm for each choice. If no such argument exists we terminate the branch of the algorithm. We stop a branch of the algorithm as soon as no more changes to labelings are made. In that case we have reached an admissible labeling acting as proof for the credulous acceptance of the queried argument.

We give a few more comments for Algorithm 4. In Line 1 of the algorithm one has to decide whether to use a Queue or a Stack for storing partialProofs. The choice determines the search strategy in the space of partial proofs: The former would give a breath first search (as suggested in [124]) while the latter yields a depth first search.

Algorithm 4 cred-pref(F, a)

Require: AF $F = (A, R)$, an argument $a \in A$
Ensure: $\mathcal{L}$ admissible labeling with $\mathcal{L}(a) = in$

1: Queue/Stack $partialProofs = \emptyset$
2: $partialProofs.push(\langle\{a\}, \emptyset, A \setminus \{a\}\rangle)$
3: **while** $\mathcal{L} = partialProofs.pop()$ **do**
4: **if** $\nexists x \in \mathcal{L}_{undec} : x \rightarrowtail \mathcal{L}_{in}$ **then**
5: **return** $\mathcal{L}$
6: **else**
7: **for** $x \in \mathcal{L}_{undec}$ s.t. $x \rightarrowtail \mathcal{L}_{in}$ **do**
8: set $\mathcal{L}(x) = out$
9: **end for**
10: **end if**
11: **for** $\mathcal{L}'_{in} \in cf(F) : \mathcal{L}'_{in} \supseteq \mathcal{L}_{in}$ is $\subseteq$-min s.t. $\mathcal{L}_{out} \subseteq \mathcal{L}'^{\oplus}_{in}$ **do**
12: $partialProofs.push(\langle\mathcal{L}'_{in}, \mathcal{L}_{out}, \emptyset\rangle)$
13: **end for**
14: **end while**

Next consider the sets $\mathcal{L}'$ in line 11. These are simply the sets where for each argument $a \in \mathcal{L}_{out} \setminus \mathcal{L}^{+}_{in}$ we pick one argument b attacking a and add b to $\mathcal{L}^{+}_{in}$. However, in each step there might be exponentially many such sets $\mathcal{L}'$. In case there is no such set we know the partial proof can not be expanded to a proof and we can close this branch of the search tree. Moreover it can happen that we consider the same partial proof twice, and thus it might be a good a idea to store already considered partial proofs.

Finally let us mention that beside the work of Verheij [124], Doutre and Mengin [50] suggest to start from an enumeration algorithm similar to Algorithm 3 but employing several shortcuts for credulous and skeptical reasoning.

4.1.3 Implementations

Labeling-based procedures for enumerating extensions have been implemented in the ArguLab[12] system [115]. ArguLab offers a web interface for evaluating argumentation frameworks w.r.t. stable, preferred, grounded, stage, and semi-stable semantics. The different labeling-based algorithms for preferred semantics are empirically compared in [111], but to the best of our knowledge the used implementations are not (yet) publicly available.

Algorithm 4 has been implemented in Verheij's COMPARG[13] system, where the partial proofs obtained by the algorithm are also used to actually compute stable, preferred, grounded and semi-stable extensions.

[12] `http://heen.webfactional.com/`
[13] `http://www.ai.rug.nl/~verheij/comparg/`

4.2 Dialogue Games

A popular approach for obtaining proof procedures for abstract argumentation is based on so called dialogue games (see, e.g., [107, 122]). Such games are played by two players, the proponent (Pro) and the opponent (Opp), on a given argumentation framework. The proponent and opponent alternate in raising arguments of the AF attacking arguments previously raised by the other player (according to certain rules). A player loses the game if she can not raise any argument. Typically, an argument a being accepted is equivalent to one player having a winning strategy when the opponent starts the dialogue game with a. However, in certain dialogue games it suffices that the proponent wins one of the possible dialogues starting with argument a to guarantee the acceptance of a. By their nature dialogue games are typically dedicated to a specific reasoning task, but sometimes they can be also used to actually compute extensions.

4.2.1 Games for Grounded and Preferred Semantics

In the following we consider games for grounded semantics and for credulous acceptance with preferred semantics, both are borrowed from [107]. In both cases the game is started by Pro raising the argument which is under question, and then Pro and Con alternately raise an argument attacking the previous argument in the dialogue. Finally a dialogue is won by the player making the last move, i.e., the player forcing the dialogue into a situation where the other player has no legal move left. The dialogue games correspond to our reasoning problems in the sense that Pro has a winning strategy in the game iff the argument is accepted. The games for the different semantics and reasoning tasks differ in the allowed moves for the players, where typically Pro and Con have different rule sets for legal moves.

A game for grounded semantics: First consider a game that provides us, given an AF $F = (A, R)$ and argument $a \in A$, a proof whether a is contained in the grounded extension of F. The game is given by the following rules of allowed moves of each player.

Legal Moves:

- For Pro: Any argument y that (i) attacks the last argument raised by Opp and (ii) is conflict-free with all arguments previously raised by Pro.
- For Opp: Any argument y that attacks the last argument raised by Pro.

One can easily show that a is in the grounded extension iff Pro has a winning strategy for the above game [107].

Example 6. *Consider the AF from Example 1. The grounded extension is $\{a, d\}$. Now if Pro starts a dialogue game with raising argument a, then as a is not attacked at all, Opp has no legal move to reply. Hence Pro has a winning strategy which reflects the fact that a is in the grounded extension. Next consider Pro starts a dialogue game with raising argument b (an argument not in the grounded extension). Then Opp has two legal moves, either raising a or c. In the first case Opp wins the game as a is not attacked at all and thus Pro has no legal moves. Hence Pro has no winning strategy when starting with b.*

A game for credulous preferred acceptance: Now let us consider a game that allows us, given an AF $F = (A, R)$ and argument $a \in A$, to prove whether a is contained in some preferred extension of F (or equivalently in some admissible set). The following game is quite similar to the game for grounded semantics, the only difference being that Opp is not allowed to repeat its moves. Restricting the legal moves of Opp makes it easier to have a winning strategy for Pro.

Legal Moves:

- For Pro: Any argument y that (i) attacks the last argument raised by Opp and (ii) is conflict-free with all arguments previously raised by Pro.
- For Opp: Any argument y that (i) attacks the last argument raised by Pro and (ii) was not previously used by Opp.

It can be shown that Pro has a winning strategy for the above game iff the argument a is in an admissible set iff a credulously accepted with preferred semantics [107].

Example 7. *Consider the very simple AF $F = (\{a, b\}, \{(a, b), (b, a)\})$ with the admissible sets $\{a\}$ and $\{b\}$. Now let us test for the credulous acceptance of a, i.e., Pro starts the game with raising a. Then the only option of Opp is to use b, Pro can use a again to defeat b. Now Opp has no legal move left, as it can not use b again. Hence Pro has a winning strategy for a. Notice that in the grounded game Pro and Opp would loop for ever with raising a and b.*

The thoughtful reader might observe that such dialogue games are indeed not algorithms. However it is more or less straight forward to build algorithms out of such games, using search procedures in the strategy space of these games branching along the possible moves (see [122, 125]). The resulting algorithms are indeed in a similar flavor as the previously discussed labeling-based algorithms.

4.2.2 Implementations

Algorithms based on dialogue games are implemented in the Dungine system [121], a Java library, which is a part of the ArgKit[14] package, as well as in the Dung-O-Matic[15] Java library. Dungine is limited to grounded and preferred semantics, while Dung-O-Matic captures all of the semantics considered here, except stage semantics. Both libraries can be tried out using OVAgen[16], an online visualization tool for abstract argumentation frameworks. Finally, let us mention here Visser's Epistemic and Practical Reasoner[17], a tool for argumentation with propositional languages that makes use of argumentation games for abstract argumentation (however, it is not a dedicated tool for abstract argumentation).

[14] `http://www.argkit.org/`
[15] `http://www.arg.dundee.ac.uk/?page_id=279`
[16] `http://ova.computing.dundee.ac.uk/ova-gen/`
[17] `http://www.wietskevisser.nl/research/epr/`

4.3 Dynamic-Programming based Approach

As discussed in Section 2 most of the reasoning problems in abstract argumentation were shown to be computationally intractable, i.e., NP-hard and even harder. Hence there is a lot of work on first classifying the exact (sources of) complexity of these problems and second on identifying problem classes that can be solved efficiently. Here we discuss algorithms based on ideas from computational complexity theory. The main observation is that binding a certain problem parameter to a fixed constant makes many of the, in general, intractable problems tractable. This property is referred to as *fixed-parameter tractability* (FPT) (see, e.g., [108]). The complexity class FPT consists of problems that can be computed in $f(k) \cdot n^{O(1)}$ where f is a function that depends on the problem parameter k, and n is the input size.

One important parameter for graph problems is the *tree-width* of a graph which is defined along so-called tree decompositions of the graph. Intuitively the tree-width measures the tree-likeliness of the graph, in particular connected graphs of tree-width 1 are exactly trees. As abstract argumentation frameworks can be seen as directed graphs the parameter tree-width can be directly applied to it. Indeed, many argumentation problems have been shown to be solvable in linear time for AFs of bounded tree-width [54, 71].

In this section we present a dynamic-programming based approach for abstract argumentation that is defined on tree decompositions. First introduced in [68], it especially aimed at the development of efficient algorithms that turn the complexity-theoretic results into practice. Credulous as well as skeptical acceptance for admissible and preferred semantics were analyzed. Furthermore, negative results for other graph parameters like bounded cycle-rank, directed path-width, and Kelly-width were obtained. Algorithms for further semantics (stable, complete), based on the same idea, were developed in [44]. Further fixed-parameter tractability results were obtained for argumentation frameworks with bounded clique-with [70] and in the work on backdoor sets for argumentation [67].

In the following we first introduce tree decompositions. We then present the general idea behind the dynamic-programming algorithms. Based on admissible semantics we exemplify the approach. Finally, we discuss how reasoning problems can be solved efficiently, i.e., without enumerating all extension, and refer to existing systems that implement this approach.

4.3.1 Tree Decompositions

A tree decomposition [118] is a mapping of a graph to a tree, defined as follows.

Definition 9. *A* tree decomposition *of an undirected graph $G = (V, E)$ is a pair $(\mathcal{T}, \mathcal{X})$ where $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$ is a tree, with vertices $V_\mathcal{T}$ and edges $E_\mathcal{T}$, and $\mathcal{X} : V_\mathcal{T} \to 2^V$ is a function that assigns to every vertex $t \in V_\mathcal{T}$ of the tree a so-called bag, i.e. a set $X_t \subseteq V$ of vertices from the original graph. These sets of vertices $(X_t)_{t \in V_\mathcal{T}}$ have to satisfy the following conditions:*

(i) $\bigcup_{t \in V_\mathcal{T}} X_t = V$

(ii) $(v_i, v_j) \in E \Rightarrow \exists t \in V_t : \{v_i, v_j\} \subseteq X_t$

(iii) $v \in X_{t_1} \wedge v \in X_{t_2} \wedge t_3 \in path(t_1, t_2) \Rightarrow v \in X_{t_3}$

A set X_t is also called the bag *for the vertex t.*

Condition (i) and (ii) guarantee that no information of the the original graph is lost, i.e., all vertices have to appear in at least one bag X_t and connected vertices have to appear together in some bag. Condition (iii) is the connectedness condition, ensuring that all bags containing the same vertex are connected in $\mathcal{T}$. The parameter tree-width is defined on the "best" tree decompositions one can get for a graph.

Definition 10. *The* width *of a tree decomposition $(\mathcal{T}, \mathcal{X})$ is defined as $max(|X_{t \in V_t}|) - 1$. The* tree-width *of a graph G is the minimum width of all possible tree decompositions of G.*

Here we will only consider *normalized* tree decompositions, which can be easily obtained from standard tree-decompositions [98]. Normalized tree decompositions comply with Definition 9, but only consist of the following four different node types:

1. *JOIN* node: A node t which has two children t' and t'', $X_t = X_{t'} = X_{t''}$.
2. *INTRODUCTION* node: A node t having exactly one child t' s.t. $|X_t| = |X_{t'}| + 1$ and $X_{t'} \subset X_t$.
3. *REMOVAL* node: A node t having exactly one child t' s.t. $|X_t| = |X_{t'}| - 1$ and $X_t \subset X_{t'}$.
4. *LEAF* node: A node t that has no child nodes.

Note that tree decompositions are defined on undirected graphs. We relate AFs (see Definition 1) to tree decompositions by defining that a tree decomposition of an AF $F = (A, R)$ is a tree decomposition of an undirected graph $G = (A, R')$ where A are the arguments of the AF and R' are the edges R without orientation. In Fig. 2 one possible normalized tree decomposition of our example AF from Fig. 1 is given. The width of this tree decomposition is 2. Note that the computation of an optimal tree decomposition (w.r.t. width) is known to be an NP-complete problem [4]. However, the problem is fixed parameter tractable w.r.t. treewidth [28] and there exist several heuristic-based algorithms that provide "good" tree decompositions in polynomial time (see, e.g., [29, 47, 48]).

4.3.2 Dynamic Programming

In the following we present a dynamic-programming algorithm for computing admissible extensions (and deciding credulous acceptance of arguments) as proposed in [68]. Algorithms for other semantics and reasoning modes can be defined similarly.

In a nutshell, the idea of dynamic programming as used here is as follows. First, a tree decomposition of the given problem instance (AF) is constructed. The tree of that decomposition is then traversed in bottom-up order. Due to the definition of tree decompositions it is possible to only work on the information that is locally available in the bags when traversing the tree. In every step

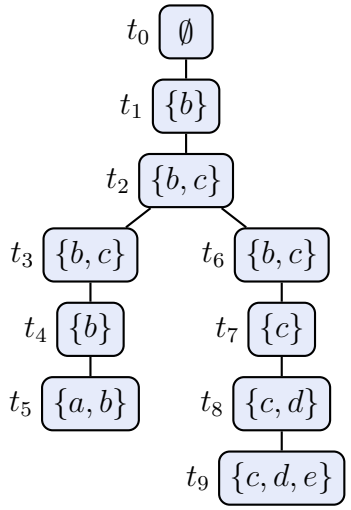
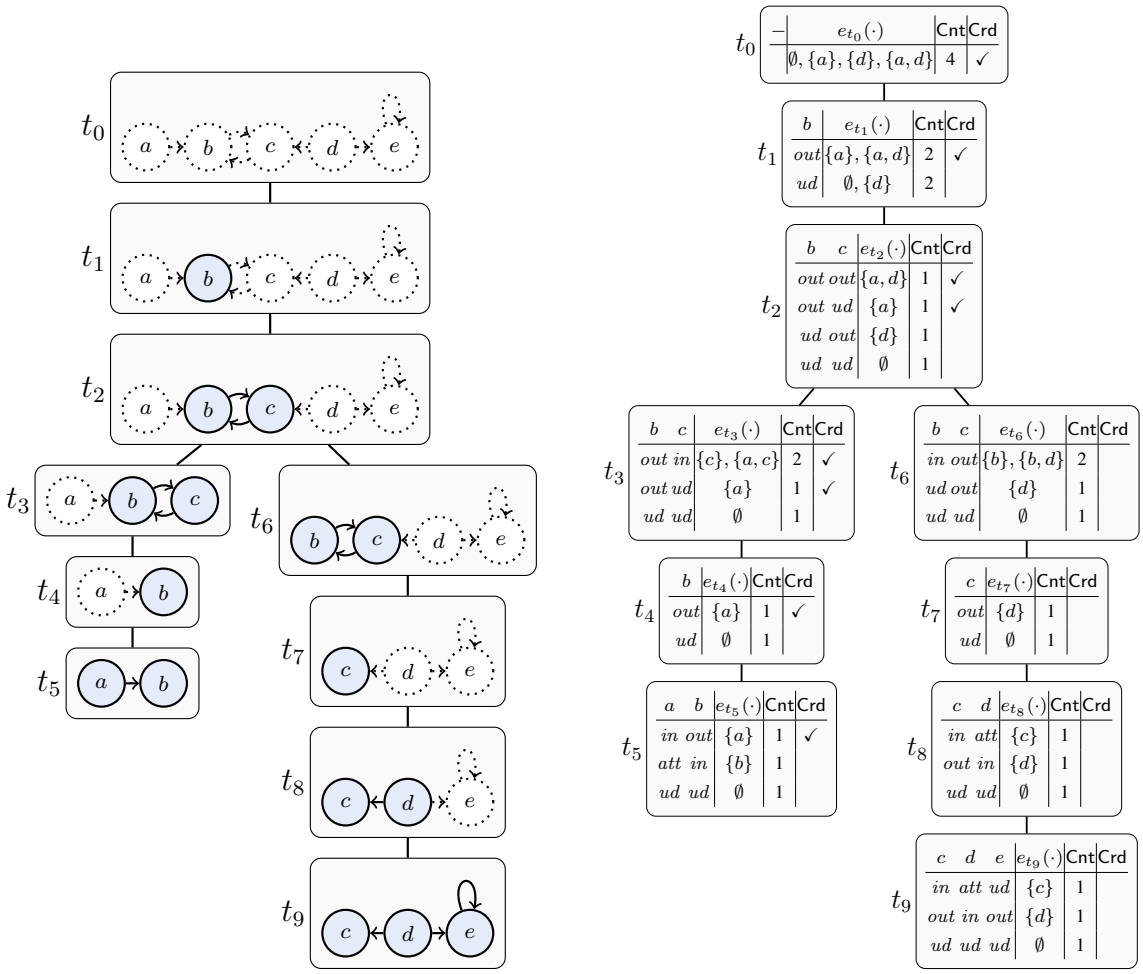


Figure 2: Normalized tree decomposition

we compute information that represents (partial) solutions for our problem. This information is computed based on the the arguments contained in the bag of the current node as well as the information from the child node(s) of the current node. The idea of dynamic programming is hereby realized as follows: Arguments that are removed from a bag of a node will never reappear in another bag later on during our traversal. We can therefore discard information for (partial) solutions in case it contains a removed argument that does not fulfill the properties of our semantics. The solutions for the whole input instance can be obtained by the final computation step in the root node.

Sub-frameworks: Towards the dynamic programming algorithm we have to introduce some notions that underlie the approach. First, for a tree decomposition $(\mathcal{T}, \mathcal{X})$ of an AF F let $t \in \mathcal{T}$. For a sub-tree of $\mathcal{T}$ that is rooted in t we define $X_{\geq t}$ as the union of all bags within this sub-tree, e.g., $X_{\geq t}$ contains all arguments of this sub-tree. Additionally, $X_{>t}$ denotes $X_{\geq t} \setminus X_t$, i.e., all arguments from the bags in the sub-tree excluding the arguments from the bag of t (even if they appear in an other bag). Furthermore, for a $t \in \mathcal{T}$ the *sub-framework in* t, denoted by F_t, consists of all arguments $x \in X_t$ and the attack relations (x_1, x_2) where $x_1, x_2 \in X_t$ and $(x_1, x_2) \in R$. The *sub-framework induced by the sub-tree rooted in* t, denoted by $F_{\geq t}$, consists of all arguments $x \in X_{\geq t}$ and the attack relations (x_1, x_2) where $x_1, x_2 \in X_{\geq t}$ and $(x_1, x_2) \in R$. Consider the tree decomposition given in Fig. 3(a). For each node t, the arguments that are contained in bag X_t are marked with solid cycles. The sub-framework F_t consists of the arguments in solid cycles and all solid attack arrows. In combination with the dotted parts we obtain the induced sub-frameworks $F_{\geq t}$.

Restricted sets: The idea is now to analyze the (sub)-framework $F_{\geq t}$ for every node t during our traversal. $X_{>t}$ denotes all arguments that were already removed from the bags of the sub-tree



(a) Sub-Frameworks (b) Tables for admissible semantics

Figure 3: Normalized tree decomposition in action

rooted at t. Hence these arguments are already completely processed by the algorithm and we can define $X_{>t}$*-restricted admissible sets*. Consider an $X_{>t}$-restricted admissible set S for a sub-framework $F_{\geq t}$. Here, S has to be conflict-free and it has to defend itself against the arguments in $X_{>t} \backslash S$. On the other hand, arguments in $X_t \cap S$ only have to be conflict-free but they can be attacked by arguments in $X_t \setminus S$ as they can still be defended by arguments appearing later, i.e., somewhere above in the tree decomposition.

Colorings: In order to represent the information that is computed within each node during traversal we need an appropriate data structure. We define so-called *colorings* that allow us to store information of relationships between arguments in $X_{\geq t}$ solely by assigning colors to arguments in X_t. For admissible semantics, this is described by a function $C : X_t \to \{in, out, att, ud\}$. Intuitively, in denotes that an argument is contained in the set S of selected arguments, out describes

that it is outside the set because it is attacked by S, att means that the argument attacks S but is not attacked by S and ud describes that the status is undecided (it is neither attacked nor attacks S). Notice that this definition is quite close to the definition of the labelings used in Algorithm 3. The main difference is that the aim of a labeling is to label the whole extension while colorings[18] are only applied to a small part of the extensions even if other parts have already been considered. Towards a more concise notion, for a coloring C, the set $[C]_{in}$ denotes all arguments that are colored with in.

We are in particular interested in colorings corresponding to at least one restricted admissible set, so-called *valid colorings*. Given a coloring C for node t, we define the *extensions of* C, $e_t(C)$, as the collection of $X_{>t}$-restricted admissible sets S for $F_{\geq t}$ which satisfy the following conditions for each $a \in X_t$:

$$C(a) = in \quad \text{iff } a \in S$$
$$C(a) = out \text{ iff } S \rightarrowtail a$$
$$C(a) = att \text{ iff } S \not\rightarrow a \text{ and } a \rightarrowtail S$$
$$C(a) = ud \text{ iff } S \not\rightarrow a \text{ and } a \not\rightarrow S$$

If $e_t(C) \neq \emptyset$, C is called a *valid coloring* for t.

Goal: Our overall goal is to compute the extensions of an AF for admissible semantics. The tree decomposition is traversed in bottom-up order. In each node we use our data structure of colorings and compute all valid colorings C for every node t. As shown in [68] there exists a one-to-one mapping between the extensions of C, $e_t(C)$, and the $X_{>t}$-restricted admissible sets for $F_{\geq t}$. Moreover, we assume that the root node r has an empty bag of arguments. Hence, by computing the valid colorings C for r we obtain the $X_{>r}$-restricted admissible sets for $F_{\geq r}$. As $X_{>r} = A$ these correspond to the admissible extensions for our original AF instance.

Node operations: In order to achieve tractability we have to compute valid colorings in bottom-up order without explicit computation of the corresponding restricted admissible sets $e_t(C)$. Hence we define operations for the computation of valid colorings which are applied recursively on the colorings computed at the child node(s). Detailed arguments for the correctness of these operations are given in [69], we shall just sketch the intuition behind them here.

Let $t \in \mathcal{T}$ be a node and t' and t'' be its possible children. Depending on the node type of t we apply the following operations:

LEAF node: Here we have that $F_t = F_{\geq t}$ and thus the restricted admissible sets are just the conflict-free sets. So we just compute the conflict-free sets of F_t and then build a coloring for each conflict free set S as follows: $C(x) = in$ if $x \in S$; $C(x) = out$ if $S \rightarrowtail x$; $C(x) = att$ if $x \rightarrowtail S$ and $S \not\rightarrow x$; $C(x) = ud$ in all other cases;

[18]Notice that we use different names for the colors than the original work [68]. This is to present the dynamic programming algorithm in a uniform setting with the labeling-based algorithms presented before.

REMOVAL node: In a removal node we have $X_t = X_{t'} \setminus \{a\}$ for some node a. For each valid coloring of t' with $C(a) \neq att$ we build a new coloring for node t by simple deleting the value for a and keeping all the remaining values. As we remove the argument a, by the connectedness of tree-decompositions, we know that we have already considered all neighbors of a. Now suppose C is a valid coloring for t', but has $C(a) = att$, i.e., a must be attacked to make the set admissible. As all neighbors of a where already considered we know that the corresponding sets can not be extended to an admissible set and thus we delete this coloring. If $C(a) \neq att$, then a does not cause a problem w.r.t. admissibility and as already all neighbors where considered will never do so.

INTRODUCTION node: For an introduction node we have $X_t = X_{t'} \cup \{a\}$. We build two colorings $C + a$ and $C \dot{+} a$ for t as described below. The first is always valid while the second is only valid if $[C \dot{+} a]_{in}$ is conflict-free.

$$(C + a)(b) = \begin{cases} C(b) & \text{if } b \in A \\ out & \text{if } b = a \text{ and } [C]_{in} \rightarrowtail a \\ att & \text{if } b = a \text{ and } [C]_{in} \not\rightarrowtail a \\ & \quad \text{and } a \rightarrowtail [C]_{in} \\ ud & \text{otherwise} \end{cases}$$

$$(C \dot{+} a)(b) = \begin{cases} in & \text{if } b = a \text{ or } C(b) = in \\ out & \text{if } a \neq b \text{ and} \\ & \quad ((a,b) \in F_t \text{ or } C(b) = out) \\ ud & \text{if } a \neq b \text{ and } C(b) = ud \text{ and} \\ & \quad (a,b) \notin F_t \text{ and } (b,a) \notin F_t \\ att & \text{otherwise} \end{cases}$$

In an introduction node we add a new argument to the framework. So for each extension we get two new candidates, one where we leave the argument a outside the extension (case $C + a$) and one where we add a to the extension (case $C \dot{+} a$). For the first coloring we just have to compute whether to color the new argument by out, att or ud while for the second coloring we first have to check that the set is still conflict-free and if so we have to update the colors of the old arguments according to their attacks with a.

JOIN node: A JOIN node has two child nodes t', t'' with $X_t = X_{t'} = X_{t''}$. We combine each valid coloring C of t' with each valid coloring D of t'' such that $[C]_{in} = [D]_{in}$ and build a new coloring as follows: All arguments in $[C]_{in}$ are colored in. An argument $x \in X_t$ is colored with out iff one of C, D colors it with out. The remaining arguments are colored with att iff one of C, D colors it with att and ud iff both C, D color it with ud.

The intuition behind this step is the following. The frameworks $F_{\geq t'}$ and $F_{\geq t''}$ are different parts of F that only intersect on X_t. So an extension of $F_{\geq t'}$ can be combined with an extension of $F_{\geq t''}$ as long as they coincide on the intersection. The join rule for the colorings corresponds to the fact that an argument attacks/is attacked by the union of two sets iff it attacks/is attacked by at least one of them.

Example 8. *Consider the leaf node t_5 with bag $\{a, b\}$ in Figure 3(b). The computed colorings represent the conflict-free (and $\emptyset$-restricted admissible) sets for $F_{\leq t_5}$. The next node t_4 is a removal node with $X_{t_4} = X_{t_5} \setminus \{a\}$. According to the definition for the computation of colorings in removal nodes the colorings for t_4 are obtained from the colorings of t_5 except for the second coloring C' (where $C'(a) = att$ and $C'(b) = in$). Here, argument b is not defended against the attack from a. Therefore, $\{b\}$ is not an $X_{>t_4}$ (or $\{a\}$)-restricted admissible set for $F_{\leq t_4}$. In node t_3 argument c is introduced. Consider the second coloring C' of t_4 where $C'(b) = ud$. Here we have two possibilities for adding c. If we do not add c to the set of selected arguments we obtain a coloring C_1 for t_3 where both arguments b and c are set to ud. On the other hand, we can add c to the set of selected arguments we obtain the coloring C_2 where $C_2(b) = out$ and $C_2(c) = in$. Note that the color of b changes in this case from ud to out as c attacks b. Furthermore note that this coloring coincides with the coloring obtained from C'' of t_4 with $C''(b) = out$ in case c is added to the set of selected arguments. Hence, C_2 represents both $\{a, c\}$ and $\{c\}$ which are $X_{>t_3}$ (or $\{a\}$)-restricted admissible sets for $F_{\leq t_3}$. In the join node t_2 two colorings C and D are combined in case $[C]_{in} = [D]_{in}$, i.e., they coincide on their in-colored arguments. Consider the second coloring C' of t_3 where $C'(b) = out$ and $C'(c) = ud$ as well as the second coloring D' of t_6 where $D'(b) = ud$ and $D'(c) = out$. Based on the definition of the join operator their combination results in a coloring C with $C(b) = out$ and $C(c) = out$ which represents one $X_{>t_2}$ (or $\{a, d, e\}$)-restricted admissible set for $F_{\leq t_2}$, namely $\{a, d\}$.*

4.3.3 Reasoning Problems

The dynamic-programming based approach can be used to solve several reasoning tasks.

Enumerating extensions: In order to enumerate all extensions for a semantics σ the tree decomposition is traversed a second time in top-down order after the initial bottom-up computation. Thereby only relevant solutions (the *extensions*) are considered. Note that we do not compute $e_t(C)$ explicitly during the first traversal as this would destroy tractability. In particular it is guaranteed that the second traversal only considers colorings that yield a solution. So enumerating extensions can be done with linear effort for each extension. For our running example AF F we obtain $\mathsf{Enum}_{adm}(F) = \{\emptyset, \{a\}, \{d\}, \{a, d\}\}$. In Fig. 3(b) this result is represented by the column $e_{t_0}(\cdot)$ in node t_0.

Counting extensions: In case we are only interested in the number of extensions a second traversal of the tree decomposition is not necessary. It is sufficient to calculate the number of $X_{>t}$-restricted admissible sets that are represented by the respective coloring immediately during the bottom-up traversal. The columns Cnt in Fig. 3(b) show the number of represented sets for each coloring. Consider for example coloring C of t_3 where $C(b) = out$ and $C(c) = in$: C represents two $X_{>t_3}$-restricted admissible sets as it results from the two colorings of t_4 where each represents one restricted set. At the root node we obtain $\mathsf{Count}_{adm}(F) = 4$.

Deciding credulous acceptance: Credulous acceptance of an argument x can be decided by storing an additional flag together with each coloring: In case $C(x)$ for a coloring C is set to in, C is marked. Additionally, this information is passed upwards the tree: If a coloring is constructed on basis of a marked coloring it is marked as well. Finally, in case the coloring at the root node is marked, we know that x is credulously accepted. In Fig. 3(b) this is represented by the columns Crd where we want to decide whether a is credulously accepted. For $\mathsf{Cred}_{adm}(a, F)$ we obtain yes. For skeptical acceptance, a dual approach can be employed e.g., for complete semantics.

4.3.4 Problems beyond NP

So far we have only considered admissible semantics but the dynamic programming approach is in no way limited to problems that are in NP. Harder problems, however, generally need a more complicated data structure. Consider preferred semantics where, for example, deciding Skept_{prf} is known to be Π_2^P-complete. We only give a rough outline of the ideas to extend the above algorithm for preferred semantics, for details the interested reader is referred to [69].

As preferred extensions are subset-maximal admissible extensions in order to guarantee subset maximality one can use pairs (C, Γ) as a data structure within a node t instead of colorings. Here, C is a coloring and Γ is a set of colorings, called *certificates*. The certificates characterize all $X_{>t}$-admissible sets which are strictly larger than the $X_{>t}$-admissible sets characterized by C. One can consider Γ as counter-examples for C representing subset-maximal $X_{>t}$-admissible sets. During the traversal of the tree decomposition, the colorings and certificates are computed analogously to the colorings for admissible semantics. At the root node r, one checks for each pair (C, Γ) whether $\Gamma = \emptyset$. If this is the case, C represents subset-maximal $X_{>r}$-admissible sets, which correspond to preferred extensions.

4.3.5 Implementations

Currently, two implementations that follow this dynamic-programming based approach are available: the stand-alone system dynPARTIX [66] and encodings for the dynamic programming interface D-FLAT [27]. Both systems share the same decomposition library, the SHARP framework[19], which provides heuristic-based tree decompositions and is responsible for handling the traversal of the tree during algorithm execution.

The *Dynamic Programming Argumentation Reasoning Tool* (or dynPARTIX[20]) is first presented in [66]. The original version is extended and improved in course of the work presented in [44]. dynPARTIX currently supports admissible, stable, complete and preferred semantics and the reasoning modes Enum, Count, Cred and Skept. dynPARTIX is entirely implemented in C++ and is intended as an easy-to-use high-performance tool for evaluating argumentation frameworks.

D-FLAT[21] stands for *Dynamic Programming Framework with Local Execution of ASP on Tree Decompositions*. In [27] a preliminary version is presented. The latest version is described in [26].

[19] `http://www.dbai.tuwien.ac.at/proj/sharp`
[20] `http://www.dbai.tuwien.ac.at/proj/argumentation/dynpartix`
[21] `http://www.dbai.tuwien.ac.at/proj/dynasp/dflat`

Here the user provides ASP encodings that define what is computed in the nodes of the decomposition; such encodings for abstract argumentation problems are available at the D-FLAT system homepage. Currently, ASP encodings for admissible, stable, complete and preferred semantics exist that can be used to obtain solutions for Enum and Count problems. Since the D-FLAT approach delegates the actual computation to powerful ASP solvers, the D-FLAT approach can be seen as a combination of a reduction-based and direct approach (since the dynamic programming algorithm inherently exploits argumentation specific information) for reasoning in abstract argumentation.

Comparing dynPARTIX and D-FLAT, the former exhibits higher performance whereas the latter allows to specify problems declaratively; this allows rapid development of new algorithms and results in easily readable and maintainable code. Generally speaking, the dynamic-programming based approach works particularly well in case the width of the tree decomposition is small, which reflects the theoretical results presented in [68].

5 Discussion

We conclude our survey on implementation of abstract argumentation with various issues we have not touched yet. This includes methods for further semantics (Section 5.1) and complementary aspects for evaluating abstract argumentation frameworks, for instance, pre-processing (Section 5.2). In Section 5.3, we give pointers to systems which are in a certain way concerned with abstract argumentation, but have a more general aim (in fact, methods as presented in this survey could be used *within* such systems). We then proceed with a global summary and discuss directions which we believe are important for future developments.

5.1 Further Semantics

In the interest of space, we have omitted a few prominent semantics in the main body of this survey. In what follows we give respective pointers to the literature and highlight systems implementing these semantics.

As shown by Baroni *et al.* [12] argumentation semantics can be defined on the basis of decomposing an AF in its strongly connected components (SCCs). This not only provides alternative definitions of some of the semantics which we have already discussed in the paper, but also leads to novel semantics, for instance cf2 [12] and stage2 [62] semantics. For both semantics, ASP encodings [61, 86] as well as labeling-based algorithms [61] have been presented, the former are integrated in the ASPARTIX system.

Moreover, there is the family of resolution-based semantics [8], with the resolution-based grounded semantics being the most popular instance. Different ASP encodings for resolution-based grounded semantics are studied in [63] and are incorporated to the ASPARTIX system, as well.

Finally, the unique-status semantics ideal [53] and eager [37] (for a general notion of parametric ideal semantics, see [60]) have been proposed to perform a prudent form of reasoning on the set of preferred extensions and semi-stable extensions, respectively. A characterization in terms of

labelings for ideal and eager semantics is given in [39] and labeling-based algorithms have been implemented in the ArguLab system. Also the Dung-O-Matic system allows for reasoning with ideal and eager semantics. In the ASP-setting a characterization for ideal semantics is given in [76] and is implemented in the ASPARTIX system.

5.2 Further Methods

Next, we briefly describe three concepts which can be considered to be used on top of argumentation systems as discussed in this survey. These methods can be seen as pre-processing or simplification steps before actually evaluating abstract argumentation frameworks.

First the idea of splitting allows to divide an argumentation framework F in (two) smaller argumentation frameworks F_1, F_2, such that there are no attacks from arguments in F_2 to arguments in F_1 [13, 17]. Then one can first compute the extensions of F_1 and then for each of its extension E compute the extensions for a slightly modified version F_2^E of F_2. The extensions of F can then be obtained by combining each extension E of F_1 with the extensions of the frameworks F_2^E. The benefit from this splitting approach comes from the fact that both F_1 and F_2 are smaller than the original AF F and thus can be evaluated faster (however, in the worst case an exponential number of AFs F_2^E has to be handled). The idea of splitting AFs has also been generalized by allowing a small number of attacks from arguments in F_2 to arguments in F_1, see [16]. In a recent paper, Liao and Huang have proposed a related method to evaluate only parts of a given framework when it comes to credulous or skeptical reasoning problems [101].

Second, the identification of redundant patterns might be used to simplify argumentation frameworks before evaluation. The notion of strong equivalence [85, 112] provides means to identify redundant attacks without analyzing the entire framework (an example are attacks between two self-attacking arguments; such attacks can be safely removed for most of the semantics). Relaxed notions of strong equivalence might be even more beneficial for this purpose, see, e.g., [14].

Finally, we mention the concept of intertranslatability between abstract argumentation semantics [73]. Here one is interested in translations from a semantics σ to another semantics τ, i.e., a function Tr that transforms arbitrary argumentation frameworks F such that $\sigma(F) = \tau(Tr(F))$. If this translation function Tr can be computed efficiently we can combine it with any system for semantics τ to build a system for σ. So translations between different semantics allow to expand the applicability of existing argumentation systems.

5.3 Further Systems

In this work we focused on systems that implement the evaluation of semantics on Dung's abstract argumentation framework directly. However, there exists a wide range of systems that extend these capabilities, in particular by additionally supporting instantiation of argumentation frameworks.

One approach is based on ASPIC^+ [116] which instantiates Dung-style frameworks. Arguments are represented as inference trees by applying strict and defeasible inference rules. TOAST (The Online Argument Structures Tool) [120] is an implementation of ASPIC^+ and is available

as web front-end[22]. The user-specified knowledge base, rule set, contrariness and preferences are used to construct an argumentation system which can currently be evaluated based on grounded, preferred, semi-stable and stable semantics.

The Carneades Web Service[23] is capable of "argument construction, storage, navigation, querying, evaluation, visualization and interchange" [92]. It is based on the ASPIC^+ model of structured argument but still preserves the features of the original version of Carneades system [93]. On the resulting Dung-style framework it applies grounded semantics.

An approach based on classical logic and argument instantiation is shown in [75]. Here arguments and possible counterarguments are constructed from a classical propositional knowledge base.

Finally, Vispartix[24] consists of a collection of ASP encodings [45] for obtaining Dung argumentation frameworks from a propositional knowledge base (and a set of predefined claims), based on the approach presented in [21]. The argumentation framework can then, for example, be evaluated by ASPARTIX.

Links to further systems can be found on Adam Wyner's web-page[25] as well as on the COMMA web-page[26]. Additionally, Simari's overview on argumentation systems [119] summarizes systems that focus on the construction of arguments. This includes approaches based on classical [21] and defeasible logic [87] and briefly introduces the systems ASPIC and CaSAPI[27] (which combines abstract and assumption-based argumentation).

5.4 Summary

The aim of this article was to provide the reader with a basic understanding of the different techniques used to implement the paradigm of abstract argumentation. We have grouped these techniques into two categories. The reduction-based techniques aim at transforming the argumentation problem at hand into an instance of a different problem (SAT, ASP, etc.) delegating the burden of computation to existing systems. On the other hand, the category of direct approaches refers to systems and methods implementing abstract argumentation "from scratch", thus allowing for tailored algorithms which explicitly realize argumentation specific optimizations.

We do not at all give any preference to one of these two categories over the other. In fact, the two categories are not as strictly separated as it might look like. The D-FLAT approach as discussed in Section 4.3.5 and also the CEGARTIX approach as introduced in Section 3.1.3 are examples which combine the advantages of the two categories. They are based on a dedicated algorithm for the argumentation problem at hand, but as a subroutine invoke existing systems (ASP and resp. SAT solvers).

[22] http://www.arg.dundee.ac.uk/toast/
[23] http://carneades.github.com/
[24] http://www.dbai.tuwien.ac.at/proj/argumentation/vispartix/
[25] http://wyner.info/LanguageLogicLawSoftware/index.php/software/
[26] http://comma.csc.liv.ac.uk/node/12
[27] http://www.doc.ic.ac.uk/~ft/CaSAPI/

5.5 Future Directions

Although significant progress has been made in the last years in implementing efficient systems for abstract argumentation, there is still a wide range of open issues.

On the one hand, several optimization methods which proved successful in other areas still have to be adapted for abstract argumentation systems. Methods including symmetry breaking, parallelization, heuristics and algorithm selection come to our mind. Even more important, benchmark suites are needed to evaluate and witness the value of such optimizations and, more generally, to compare the different approaches which are nowadays available on a broad and objective scope. Several ideas for establishing a benchmark library for abstract argumentation have been collected in [64].

On the other hand, we have to understand particularities in the argumentation domain to tune the systems towards more practical needs, in particular when used within an instantiation-based argumentation context. First, argumentation is inherently dynamic [15, 42, 80] and thus one expects that argumentation frameworks are continuously evolving. Consequently, methods which allow for incremental evaluation of frameworks (i.e., the system "remembers" the framework it has evaluated last time and tries to build the current solving on this prior results) are an important research direction. A first valuable theoretical contribution in this direction can be found in [102]. Second, many people in the community argue that abstract argumentation is not a stand-alone formalism. Consequently, the integration of "abstract" into "real" argumentation systems is central. In particular, the specific needs of these real argumentation systems have to be taken into account when abstract argumentation systems are improved. To this end, it has to be clarified whether such integrated systems lead to abstract frameworks of certain structure (in particular, in many cases, instantiations lead to particular symmetries in the resulting frameworks). Advanced abstract argumentation systems therefore should either be optimized towards such structures or provide interfaces which allow to feed additional information from the instantiation process to the system in order to guide heuristics or to prune the search space.

In conclusion, we believe that the challenge of implementing abstract argumentation systems is a perfect play-ground to apply and test different techniques on a set of uniform yet computationally manifold problems which are given by the different semantics for abstract argumentation. The future will show which techniques prove successful or whether completely novel methods will emerge in course of these investigations.

Acknowledgements

The authors would like to thank all colleagues who participated in the development of the systems ASPARTIX, CEGARTIX, D-FLAT and dynPARTIX. In particular, we would like to mention here Bernhard Bliem, Uwe Egly, Markus Hecher, Matti Järvisalo, Michael Morak, Clemens Nopp, Michael Petritsch, Reinhard Pichler and Paul Wandl.

References

[1] Leila Amgoud and Caroline Devred. Argumentation frameworks as constraint satisfaction problems. In Salem Benferhat and John Grant, editors, *Proceedings of the 5th International Conference on Scalable Uncertainty Management (SUM 2011)*, volume 6929 of *Lecture Notes in Computer Science*, pages 110–122. Springer, 2011.

[2] Leila Amgoud, Caroline Devred, and Marie-Christine Lagasquie-Schiex. A constrained argumentation system for practical reasoning. In Iyad Rahwan and Pavlos Moraitis, editors, *Proceedings of the 5th International Workshop on Argumentation in Multi-Agent Systems (ArgMAS 2008), Revised Selected and Invited Papers*, volume 5384 of *Lecture Notes in Computer Science*, pages 37–56. Springer, 2009.

[3] Ofer Arieli and Martin Caminada. A general QBF-based formalization of abstract argumentation theory. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the 4th Conference on Computational Models of Argument (COMMA 2012)*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 105–116. IOS Press, 2012.

[4] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8:277–284, April 1987.

[5] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2002.

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

[7] Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Giovanni Guida. AFRA: Argumentation framework with recursive attacks. *Int. J. Approx. Reasoning*, 52(1):19–37, 2011.

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

[9] Pietro Baroni and Massimiliano Giacomin. On principle-based evaluation of extension-based argumentation semantics. *Artif. Intell.*, 171(10-15):675–700, 2007.

[10] Pietro Baroni and Massimiliano Giacomin. A systematic classification of argumentation frameworks where semantics agree. In Philippe Besnard, Sylvie Doutre, and Anthony Hunter, editors, *Proceedings of the 2nd Conference on Computational Models of Argument (COMMA 2008)*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, pages 37–48. IOS Press, 2008.

[11] Pietro Baroni and Massimiliano Giacomin. Semantics of abstract argument systems. In Iyad Rahwan and Guillermo R. Simari, editors, *Argumentation in Artificial Intelligence*, pages 25–44. Springer, 2009.

[12] Pietro Baroni, Massimiliano Giacomin, and Giovanni Guida. SCC-Recursiveness: A general schema for argumentation semantics. *Artif. Intell.*, 168(1-2):162–210, 2005.

[13] Ringo Baumann. Splitting an argumentation framework. In James P. Delgrande and Wolfgang Faber, editors, *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2011)*, volume 6645 of *Lecture Notes in Computer Science*, pages 40–53. Springer, 2011.

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

[15] Ringo Baumann and Gerhard Brewka. Expanding argumentation frameworks: Enforcing and monotonicity results. In Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Guillermo R. Simari, editors, *Proceedings of the 3rd Conference on Computational Models of Argument (COMMA 2010)*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 75–86. IOS Press, 2010.

[16] Ringo Baumann, Gerhard Brewka, Wolfgang Dvořák, and Stefan Woltran. Parameterized splitting: A simple modification-based approach. In Esra Erdem, Joohyung Lee, Yuliya Lierler, and David Pearce, editors, *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*, pages 57–71. Springer, 2012.

[17] Ringo Baumann, Gerhard Brewka, and Renata Wong. Splitting argumentation frameworks: An empirical evaluation. In Sanjay Modgil, Nir Oren, and Francesca Toni, editors, *Proceedings of the 1st International Workshop on Theory and Applications of Formal Argumentation (TAFA 2011), Revised Selected Papers*, volume 7132 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2012.

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

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

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

[21] Philippe Besnard and Anthony Hunter. A logic-based theory of deductive arguments. *Artif. Intell.*, 128(1-2):203–235, 2001.

[22] Philippe Besnard and Anthony Hunter. *Elements of Argumentation*. MIT Press, 2008.

[23] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[24] Stefano Bistarelli and Francesco Santini. ConArg: A constraint-based computational framework for argumentation systems. In Taghi M. Khoshgoftaar and Xingquan (Hill) Zhu, editors, *Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2011)*, pages 605–612. IEEE Computer Society Press, 2011.

[25] Stefano Bistarelli and Francesco Santini. Modeling and solving AFs with a constraint-based tool: ConArg. In Sanjay Modgil, Nir Oren, and Francesca Toni, editors, *Proceedings of the 1st International Workshop on Theory and Applications of Formal Argumentation (TAFA 2011)*, volume 7132 of *Lecture Notes in Computer Science*, pages 99–116. Springer, 2012.

[26] Bernhard Bliem. Decompose, Guess & Check - Declarative problem solving on tree decompositions. Master's thesis, Vienna University of Technology, 2012.

[27] Bernhard Bliem, Michael Morak, and Stefan Woltran. D-FLAT: Declarative problem solving using tree decompositions and answer-set programming. *Theory and Practice of Logic Programming*, 12(4-5):445–464, 2012.

[28] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[29] Hans L. Bodlaender and Arie M.C.A. Koster. Treewidth computations I. Upper bounds. *Information and Computation*, 208(3):259 – 275, 2010.

[30] Gerhard Brewka. Nonmonotonic tools for argumentation. In Tomi Janhunen and Ilkka Niemelä, editors, *Proceedings of the 12th European Conference on Logics in Artificial Intelligence (JELIA 2010)*, volume 6341 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 2010.

[31] Gerhard Brewka and Thomas Eiter. Argumentation context systems: A framework for abstract group argumentation. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2009)*, volume 5753 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2009.

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

[33] Gerhard Brewka and Stefan Woltran. Abstract dialectical frameworks. In Fangzhen Lin, Ulrike Sattler, and Mirosław Truszczyński, editors, *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR 2010)*, pages 780–785. AAAI Press, 2010.

[34] Maximiliano C. Budán, Mauro J.G. Lucero, Carlos I. Chesñevar, and Guillermo R. Simari. Modelling time and reliability in structured argumentation frameworks. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, pages 578–582. AAAI Press, 2012.

[35] Francesco Calimeri, Giovambattista Ianni, Francesco Ricca, Mario Alviano, Annamaria Bria, Gelsomina Catalano, Susanna Cozza, Wolfgang Faber, Onofrio Febbraro, Nicola Leone, Marco Manna, Alessandra Martello, Claudio Panetta, Simona Perri, Kristian Reale, Maria Carmela Santoro, Marco Sirianni, Giorgio Terracina, and Pierfrancesco Veltri. The third answer set programming competition: Preliminary report of the system competition track. In James P. Delgrande and Wolfgang Faber, editors, *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2011)*, volume 6645 of *Lecture Notes in Computer Science*, pages 388–403. Springer, 2011.

[36] Martin Caminada. An algorithm for computing semi-stable semantics. In Khaled Mellouli, editor, *Proceedings of the 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2007)*, volume 4724 of *Lecture Notes in Computer Science*, pages 222–234. Springer, 2007.

[37] Martin Caminada. Comparing two unique extension semantics for formal argumentation: ideal and eager. In *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2007)*, pages 81–87, 2007.

[38] Martin Caminada. An algorithm for stage semantics. In Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Guillermo R. Simari, editors, *Proceedings of the 3rd International Conference on Computational Models of Argument (COMMA 2010)*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 147–158. IOS Press, 2010.

[39] Martin Caminada. A labelling approach for ideal and stage semantics. *Argument & Computation*, 2:1–21, 2011.

[40] Martin Caminada and Leila Amgoud. On the evaluation of argumentation formalisms. *Artif. Intell.*, 171(5-6):286–310, 2007.

[41] Martin Caminada and Dov M. Gabbay. A logical account of formal argumentation. *Studia Logica*, 93(2):109–145, 2009.

[42] Claudette Cayrol, Florence D. de Saint-Cyr, and Marie-Christine Lagasquie-Schiex. Change in abstract argumentation frameworks: Adding an argument. *J. Artif. Intell. Res.*, 38:49–84, 2010.

[43] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. Bipolar abstract argumentation systems. In Iyad Rahwan and Guillermo R. Simari, editors, *Argumentation in Artificial Intelligence*, pages 65–84. Springer, 2009.

[44] Günther Charwat. Tree-decomposition based algorithms for abstract argumentation frameworks. Master's thesis, Vienna University of Technology, 2012.

[45] Günther Charwat, Johannes P. Wallner, and Stefan Woltran. Utilizing ASP for generating and visualizing argumentation frameworks. In Michael Fink and Yuliya Lierler, editors, *Proceedings of the 5th International Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2012)*, pages 51–65, 2012.

[46] Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Symmetric argumentation frameworks. In Lluis Godo, editor, *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2005)*, volume 3571 of *Lecture Notes in Computer Science*, pages 317–328. Springer, 2005.

[47] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[48] Artan Dermaku, Tobias Ganzow, Georg Gottlob, Ben McMahan, Nysret Musliu, and Marko Samer. Heuristic methods for hypertree decomposition. In Alexander Gelbukh and Eduardo F. Morales, editors, *Proceedings of the 7th Mexican International Conference on Artificial Intelligence (MICAI 2008): Advances in Artificial Intelligence*, volume 5317 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2008.

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

[50] Sylvie Doutre and Jérôme Mengin. Preferred extensions of argumentation frameworks: Query answering and computation. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2001.

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

[52] Phan M. Dung, Robert A. Kowalski, and Francesca Toni. Assumption-based argumentation. In Iyad Rahwan and Guillermo R. Simari, editors, *Argumentation in Artificial Intelligence*, pages 25–44. Springer, 2009.

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

[54] Paul E. Dunne. Computational properties of argument systems satisfying graph-theoretic constraints. *Artif. Intell.*, 171(10-15):701–729, 2007.

[55] Paul E. Dunne. The computational complexity of ideal semantics. *Artif. Intell.*, 173(18):1559–1591, 2009.

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

[57] Paul E. Dunne and Martin Caminada. Computational complexity of semi-stable semantics in abstract argumentation frameworks. In Steffen Hölldobler, Carsten Lutz, and Heinrich Wansing, editors, *Proceedings of the 11th European Conference on Logics in Artificial Intelligence (JELIA 2008)*, volume 5293 of *Lecture Notes in Computer Science*, pages 153–165. Springer, 2008.

[58] Paul E. Dunne, Anthony Hunter, Peter McBurney, Simon Parsons, and Michael Wooldridge. Weighted argument systems: Basic definitions, algorithms, and complexity results. *Artif. Intell.*, 175(2):457–486, 2011.

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

[60] Wolfgang Dvořák, Paul E. Dunne, and Stefan Woltran. Parametric properties of ideal semantics. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 851–856. AAAI Press, 2011.

[61] Wolfgang Dvořák and Sarah A. Gaggl. Computational aspects of cf2 and stage2 argumentation semantics. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the 4th Conference on Computational Models of Argument (COMMA 2012)*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 273–284. IOS Press, 2012.

[62] Wolfgang Dvořák and Sarah A. Gaggl. Incorporating stage semantics in the scc-recursive schema for argumentation semantics. In *Proceedings of the 14th International Workshop on Non-Monotonic Reasoning*, 2012.

[63] Wolfgang Dvořák, Sarah A. Gaggl, Johannes P. Wallner, and Stefan Woltran. Making use of advances in answer-set programming for abstract argumentation systems. *CoRR*, abs/1108.4942, 2011.

[64] Wolfgang Dvořák, Sarah Alice Gaggl, Stefan Szeider, and Stefan Woltran. Benchmark libraries for argumentation. In Sascha Ossowski, editor, *Agreement Technologies*, volume 8 of *LGTS*, chapter The Added Value of Argumentation, pages 389–393. Springer, 2012.

[65] Wolfgang Dvořák, Matti Järvisalo, Johannes P. Wallner, and Stefan Woltran. Complexity-sensitive decision procedures for abstract argumentation. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, pages 54–64. AAAI Press, 2012.

[66] Wolfgang Dvořák, Michael Morak, Clemens Nopp, and Stefan Woltran. dynPARTIX - A dynamic programming reasoner for abstract argumentation. *CoRR*, abs/1108.4804, 2011.

[67] Wolfgang Dvořák, Sebastian Ordyniak, and Stefan Szeider. Augmenting tractable fragments of abstract argumentation. *Artificial Intelligence*, 186(0):157–173, 2012.

[68] Wolfgang Dvořák, Reinhard Pichler, and Stefan Woltran. Towards fixed-parameter tractable algorithms for argumentation. In Fangzhen Lin, Ulrike Sattler, and Mirosław Truszczyński, editors, *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR 2010)*, pages 112–122. AAAI Press, 2010.

[69] Wolfgang Dvořák, Reinhard Pichler, and Stefan Woltran. Towards fixed-parameter tractable algorithms for abstract argumentation. *Artif. Intell.*, 186:1–37, 2012.

[70] Wolfgang Dvořák, Stefan Szeider, and Stefan Woltran. Reasoning in argumentation frameworks of bounded clique-width. In Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Guillermo R. Simari, editors, *Proceedings of the 3rd Conference on Computational Models of Argument (COMMA 2010)*, Frontiers in Artificial Intelligence and Applications, pages 219–230. IOS Press, 2010.

[71] Wolfgang Dvořák, Stefan Szeider, and Stefan Woltran. Abstract argumentation via monadic second order logic. In Eyke Hüllermeier, Sebastian Link, Thomas Fober, and Bernhard Seeger, editors, *Proceedings of the 6th International Conference on Scalable Uncertainty Management (SUM 2012)*, volume 7520 of *Lecture Notes in Computer Science*, pages 85–98. Springer, 2012.

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

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

[74] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[75] Vasiliki Efstathiou and Anthony Hunter. Algorithms for generating arguments and counterarguments in propositional logic. *Int. J. Approx. Reasoning*, 52(6):672–704, 2011.

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

[77] Uwe Egly and Stefan Woltran. Reasoning in argumentation frameworks using quantified boolean formulas. In Paul E. Dunne and Trevor J. M. Bench-Capon, editors, *Proceedings of the 1st Conference on Computational Models of Argument (COMMA 2006)*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 133–144. IOS Press, 2006.

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

[79] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.

[80] Marcelo A. Falappa, Alejandro J. García, Gabriele Kern-Isberner, and Guillermo R. Simari. On the evolving relation between belief revision and argumentation. *Knowledge Eng. Review*, 26(1):35–43, 2011.

[81] Dov M. Gabbay. Fibring argumentation frames. *Studia Logica*, 93(2-3):231–295, 2009.

[82] Dov M. Gabbay. Dung's argumentation is essentially equivalent to classical propositional logic with the peirce-quine dagger. *Logica Universalis*, 5:255–318, 2011.

[83] Dov M. Gabbay. An equational approach to argumentation networks. *Argument & Computation*, 3(2-3):87–142, 2012.

[84] Dov M. Gabbay. The equational approach to cf2 semantics. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the 4th Conference on Computational Models of Argument (COMMA 2012)*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 141–152. IOS Press, 2012.

[85] Sarah A. Gaggl and Stefan Woltran. Strong equivalence for argumentation semantics based on conflict-free sets. In Weiru Liu, editor, *Proceedings of the 11th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2011)*, volume 6717 of *Lecture Notes in Computer Science*, pages 38–49. Springer, 2011.

[86] Sarah A. Gaggl and Stefan Woltran. The cf2 argumentation semantics revisited. *Journal of Logic and Computation*, 2012. To appear.

[87] Alejandro J. García and Guillermo R. Simari. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming*, 4(1-2):95–138, 2004.

[88] Martin Gebser, Roland Kaminski, and Torsten Schaub. Complex optimization in answer set programming. *TPLP*, 11(4–5):821–839, 2011.

[89] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius T. Schneider. Potassco: The Potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, 2011.

[90] Michael Gelfond. Representing knowledge in A-Prolog. In *Computational Logic: From Logic Programming into the Future*, volume 2408 of *Lecture Notes in Computer Science*, pages 413–451. Springer, 2002.

[91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[92] Thomas F. Gordon. The Carneades web service. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the 4th Conference on Computational Models of Argument (COMMA 2012)*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 517–518. IOS Press, 2012.

[93] Thomas F. Gordon, Henry Prakken, and Douglas Walton. The Carneades model of argument and burden of proof. *Artif. Intell.*, 171(10-15):875–896, 2007.

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

[95] Anthony Hunter. Some foundations for probabilistic abstract argumentation. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the 4th Conference on Computational Models of Argument (COMMA 2012)*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 117–128. IOS Press, 2012.

[96] Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The international sat solver competitions. *AI Magazine*, 33(1), 2012.

[97] Hiroyuki Kido and Katsumi Nitta. Practical argumentation semantics for socially efficient defeasible consequence. In Liz Sonenberg, Peter Stone, Kagan Tumer, and Pinar Yolum, editors, *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pages 267–274. IFAAMAS, 2011.

[98] Ton Kloks. *Treewidth: Computations and approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[99] Alexander Langer, Felix Reidl, Peter Rossmanith, and Somnath Sikdar. Evaluation of an mso-solver. In David A. Bader and Petra Mutzel, editors, *Proceedings of the 2012 Meeting on Algorithm Engineering & Experiments (ALENEX 2012)*, pages 55–63. SIAM / Omnipress, 2012.

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

[101] Beishui Liao and Huaxin Huang. Partial semantics of argumentation: Basic properties and empirical results. *Journal of Logic and Computation*, 2012. To appear.

[102] Beishui Liao, Li Jin, and Robert C. Koons. Dynamics of argumentation systems: A division-based method. *Artif. Intell.*, 175(11):1790–1814, 2011.

[103] Victor W. Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm – A 25-Year Perspective*, pages 375–398. Springer, 1999.

[104] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[105] Diego C. Martínez, Alejandro J. García, and Guillermo R. Simari. On acceptability in abstract argumentation frameworks with an extended defeat relation. In Paul E. Dunne and Trevor J. M. Bench-Capon, editors, *Proceedings of the 1st Conference on Computational Models of Argument (COMMA 2006)*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 273–278. IOS Press, 2006.

[106] Sanjay Modgil. Reasoning about preferences in argumentation frameworks. *Artif. Intell.*, 173(9-10):901–934, 2009.

[107] Sanjay Modgil and Martin Caminada. Proof theories and algorithms for abstract argumentation frameworks. In Iyad Rahwan and Guillermo R. Simari, editors, *Argumentation in Artificial Intelligence*, pages 105–132. Springer, 2009.

[108] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[109] Ilkka Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3–4):241–273, 1999.

[110] Juan C. Nieves, Mauricio Osorio, and Ulises Cortés. Preferred extensions as stable models. *Theory and Practice of Logic Programming*, 8(4):527–543, 2008.

[111] Samer Nofal, Paul E. Dunne, and Katie Atkinson. On preferred extension enumeration in abstract argumentation. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the 4th Conference on Computational Models of Argument (COMMA 2012)*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 205–216. IOS Press, 2012.

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

[113] Mauricio Osorio, Claudia Zepeda, Juan C. Nieves, and Ulises Cortés. Inferring acceptable arguments with answer set programming. In *Proceedings of the 6th Mexican International Conference on Computer Science (ENC 2005)*, pages 198–205. IEEE Computer Society, 2005.

[114] Claudia Peschiera, Luca Pulina, Armando Tacchella, Uwe Bubeck, Oliver Kullmann, and Inês Lynce. The seventh QBF solvers evaluation (qbfeval'10). In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT 2010)*, volume 6175 of *Lecture Notes in Computer Science*, pages 237–250. Springer, 2010.

[115] Mikolaj Podlaszewski, Martin Caminada, and Gabriella Pigozzi. An implementation of basic argumentation components. In Liz Sonenberg, Peter Stone, Kagan Tumer, and Pinar Yolum, editors, *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pages 1307–1308. IFAAMAS, 2011.

[116] Henry Prakken. An abstract framework for argumentation with structured arguments. *Argument & Computation*, 1(2):93–124, 2010.

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

[118] Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49 – 64, 1984.

[119] Guillermo R. Simari. A brief overview of research in argumentation systems. In Salem Benferhat and John Grant, editors, *Proceedings of the 5th International Conference on Scalable Uncertainty Management (SUM 2011)*, volume 6929 of *Lecture Notes in Computer Science*, pages 81–95. Springer, 2011.

[120] Mark Snaith and Chris Reed. TOAST: Online ASPIC$^+$ implementation. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the 4th Conference on Computational Models of Argument (COMMA 2012)*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 509–510. IOS Press, 2012.

[121] Matthew South, Gerard Vreeswijk, and John Fox. Dungine: A Java Dung reasoner. In Philippe Besnard, Sylvie Doutre, and Anthony Hunter, editors, *Proceedings of the 2nd Conference on Computational Models of Argument (COMMA 2008)*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, pages 360–368. IOS Press, 2008.

[122] Phan M. Thang, Phan M. Dung, and Nguyen D. Hung. Towards a common framework for dialectical proof procedures in abstract argumentation. *J. Log. Comput.*, 19(6):1071–1109, 2009.

[123] Francesca Toni and Marek Sergot. Argumentation and answer set programming. In Marcello Balduccini and Tran C. Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays in Honor of Michael Gelfond*, volume 6565 of *Lecture Notes in Computer Science*, pages 164–180. Springer, 2011.

[124] Bart Verheij. A labeling approach to the computation of credulous acceptance in argumentation. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 623–628, 2007.

[125] Gerard Vreeswijk. An algorithm to compute minimally grounded and admissible defence sets in argument systems. In Paul E. Dunne and Trevor J. M. Bench-Capon, editors, *Proceedings of the 1st Conference on Computational Models of Argument (COMMA 2006)*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 109–120, 2006.

[126] Toshiko Wakaki and Katsumi Nitta. Computing argumentation semantics in answer set programming. In *New Frontiers in Artificial Intelligence, JSAI 2008 Conference and Workshops, Revised Selected Papers*, volume 5447 of *Lecture Notes in Computer Science*, pages 254–269, 2008.