Artificial Intelligence 220 (2015) 28–63

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Methods for solving reasoning problems in abstract argumentation – A survey

Günther Charwat [a], Wolfgang Dvořák [b], Sarah A. Gaggl [c], Johannes P. Wallner [a], Stefan Woltran [a,*]

[a] Vienna University of Technology, Institute of Information Systems, Austria
[b] University of Vienna, Faculty of Computer Science, Austria
[c] Technische Universität Dresden, Institute of Artificial Intelligence, Germany

ARTICLE INFO

Article history:
Received 29 March 2013
Received in revised form 19 November 2014
Accepted 28 November 2014
Available online 15 December 2014

Keywords:
Abstract argumentation
Algorithms
Argumentation systems

ABSTRACT

Within the last decade, abstract argumentation has emerged as a central field in Artificial Intelligence. Besides providing a core formalism for many advanced argumentation systems, abstract argumentation has also served to capture several non-monotonic logics and other AI related principles. Although the idea of abstract argumentation is appealingly simple, several reasoning problems in this formalism exhibit 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 reasoning problems in abstract argumentation and compare their particular features. Moreover, we highlight available state-of-the-art systems for abstract argumentation, which put these methods to practice.

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* [1], 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 non-monotonic logics. Yet this second contribution located Argumentation as a sub-discipline of Artificial Intelligence [2]. The increasing significance of argumentation as a research area of its own has also been witnessed by the biennial COMMA Conference on Computational Models of Argument,[1] which from the second meeting onwards provides sessions for software demonstrations of implemented systems, 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* [3].

* Corresponding author.
E-mail addresses: gcharwat@dbai.tuwien.ac.at (G. Charwat), wolfgang.dvorak@univie.ac.at (W. Dvořák), sarah.gaggl@tu-dresden.de (S.A. Gaggl), wallner@dbai.tuwien.ac.at (J.P. Wallner), woltran@dbai.tuwien.ac.at (S. Woltran).

[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.

http://dx.doi.org/10.1016/j.artint.2014.11.008

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., [4,5]), 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., [6]). 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* [7].

One direction of research in abstract argumentation was devoted to develop the "right" forms of semantics (see, e.g., [8–10] 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 suit 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 research on efficient procedures for this particular formalism. However, it was soon recognized that already these simple frameworks show high complexity (see, e.g., [12–14]); due to the link to non-monotonic logic 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 problems at hand.

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 these 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 reasoning problems within other formalisms like constraint-satisfaction problems (CSP) [15], propositional logic [16] or answer-set programming (ASP) [17]. In this approach, the resulting argumentation systems directly benefit from the high level of sophistication today's systems 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 [18], and later extended by means of quantified propositional logic [19,20]. We will first discuss the theoretical underpinnings of this approach and then continue with an introduction to the CEGARTIX system [21] and the ArgSemSAT system [22], which both rely on iterative calls to SAT solvers for argumentation semantics of high complexity (i.e., being located on the second level of the polynomial hierarchy).
- *CSP-based* approach. Reductions to CSP have been addressed by Amgoud and Devred [23] and Bistarelli and Santini [24–28]; the latter work led to the development of the ConArg system. We introduce the formalization of argumentation frameworks in terms of CSPs, where the arguments of the given framework represent the variables of the CSP with domains of 0 and 1. The constraints then depend on the specific semantics.
- *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 [29] provides a good overview). We focus here on the ASPARTIX approach [30] 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 or CSP methods can be 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. We will discuss standard ways of ASP encodings, but also some recent methods which exploit advanced ASP techniques [31].

In the remainder of Section 3 we shall present the concepts behind other reduction-based approaches, for instance, the *equational approach* as introduced by Gabbay in [32] and the reductions to *monadic second order logic* as proposed in [33].

In Section 4, we collect methods and algorithms which have been developed from scratch (instead of using another formalism like SAT or ASP). The obvious disadvantage of this *direct approach* is due to the fact that existing technology cannot be directly employed. On the other hand, 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 *labeling approach* [34–38] gives a more fine-grained handle on the status of arguments when evaluated w.r.t. semantics and also provides a solid basis for dedicated algorithms. We present two different proposals for implementing

[4] During the presentation of [11] at COMMA 2006.

[5] However, in connection with particular instantiation schemes, it is often claimed that only semantics that 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., [5]).

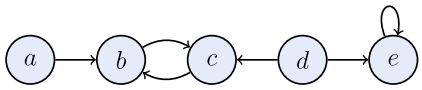


Fig. 1. Example argumentation framework.

the enumeration of preferred extensions, one along the lines of [35] and another following [34] using improvements from [36]. Furthermore, we discuss an algorithm dedicated to credulous reasoning with preferred semantics following the work of [38]. 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 [35,39], which are employed in systems like Dungine and Dung-O-Matic.
- Finally, we will take a look on *dynamic programming approaches* [40] which operate on decompositions of frameworks. Notably, the running times in this approach are not mainly dependent on the size of the given framework, but on a structural parameter. We focus on the parameter tree-width and the concept of tree decomposition. This method was first advocated by Dunne [41] and later realized in the dynPARTIX system [42].

As already hinted above, many of the methods we present have found their way into an available software system. Therefore, we will not only explain these methods in this survey, but shall also give the interested reader pointers to concrete systems which can be used to experiment. Section 5 contains a comparison of the systems w.r.t. their features (e.g. supported semantics and reasoning problems) and the underlying concepts. Some of the systems have been evaluated and compared w.r.t. their performance (see e.g., [31,36,43–45]), but no exhaustive performance comparisons have been done so far. In fact, an organized competition comparable to the ones from the areas of SAT [46] or ASP [47] is planned to take place in 2015 for the first time [48].[6] Thus, we abstain here from a systematic comparison of the systems' performance.

To summarize, our goal is to introduce a selection of methods for evaluating abstract argumentation frameworks; 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 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. [49].

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 [50] and Carneades [51]), assumption-based argumentation [52], or systems based on defeasible logic [53] are out of the scope of this article.[7] Likewise, we will not consider the vast collection of extensions to Dung's frameworks,[8] like value-based [56], bipolar [57], extended [58], constrained [59], temporal [60], practical [61], and fibring argumentation frameworks [62], as well as argumentation frameworks with recursive attacks [63], argumentation context systems [64], and abstract dialectical frameworks [65]. We also exclude abstract argumentation with uncertainty or weights here; recent articles by Hunter [66] and respectively Dunne et al. [67] introduce these variants in detail.

2. Background

In this section we introduce (abstract) argumentation frameworks [1] and recall the semantics we study in this paper (see also [10,49,68]).

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.

A semantics for argumentation frameworks is defined as a function σ which assigns to each AF $F = (A, R)$ a set $\sigma(F) \subseteq 2^A$ of extensions.

[6] See http://argumentationcompetition.org for further information.
[7] An overview on these approaches is given in [54].
[8] A brief overview on such approaches is given in [55].

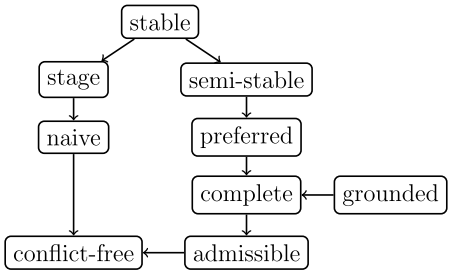


Fig. 2. Relations between argumentation semantics: An arrow from a semantics σ to another semantics τ denotes that each σ-extension is also a τ-extension.

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. $(a, b) \in R$). Furthermore, we write $S \not\rightarrowtail^R a$ (resp. $a \not\rightarrowtail^R S$) in case there is no 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 (resp. attacking) S as $S_R^{\oplus} = \{x \mid S \rightarrowtail^R x\}$ (resp. $S_R^{\ominus} = \{x \mid x \rightarrowtail^R S\}$), and define the *range of* S as $S_R^+ = S \cup S_R^{\oplus}$ and the *negative range of* S as $S_R^- = S \cup S_R^{\ominus}$.

The next definition formally defines the semantics we will focus on in this survey. All of them are based on conflict-free sets, i.e. it is not allowed to jointly accept arguments which are adjacent in the framework. Different additional criteria are then used for the concrete definition: naive extensions are just maximal (with respect to set-inclusion) conflict-free sets, stable extensions have to attack all other arguments, admissible sets defend themselves from attackers, complete extensions in addition have to contain all defended arguments. The grounded extension is given by the subset-minimal complete extension. Preferred extensions are subset-maximal admissible sets (equivalently: subset-maximal complete extensions); finally, semi-stable and stage extensions are characterized by maximizing the concept of range.

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 $\mathcal{F}_F$. Furthermore, Fig. 2 shows the relations between the aforementioned semantics. The figure is complete in the sense that if there is no arrow from semantics σ to semantics τ, then there is some AF F such that $\sigma(F) \not\subseteq \tau(F)$. For all semantics σ we introduced here, except stable semantics, it holds that for any AF F we have $\sigma(F) \neq \emptyset$.

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 kinds of labelings instead of extensions. As an example we consider the popular approach by Caminada and Gabbay [69] 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, i.e., it is defended by the *in* labeled arguments; *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 [69].

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 $(b, a) \in R$, $\mathcal{L}(b) = out$.
- $\mathcal{L}(a) = out$ 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. One can define labeling-based versions for all of our semantics (see [69]), but this is out of the scope of this survey.

Reasoning in argumentation frameworks We recall the most important reasoning problems for AFs: Given an argumentation framework F and a semantics σ, $\mathsf{Enum}_\sigma(F)$ results in an enumeration of all extensions. A simpler notion is $\mathsf{Count}_\sigma(F)$, which only counts the number of extensions. Query-based problems 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. This problem typically occurs 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

- $\mathsf{Enum}_\sigma(F) = \sigma(F)$
- $\mathsf{Count}_\sigma(F) = |\sigma(F)|$
- $\mathsf{Cred}_\sigma(a, F) = \begin{cases} \text{yes} & \text{if } a \in \bigcup_{S \in \sigma(F)} S \\ \text{no} & \text{otherwise} \end{cases}$
- $\mathsf{Skept}_\sigma(a, F) = \begin{cases} \text{yes} & \text{if } a \in \bigcap_{S \in \sigma(F)} S \\ \text{no} & \text{otherwise} \end{cases}$
- $\mathsf{Ver}_\sigma(S, F) = \begin{cases} \text{yes} & \text{if } S \in \sigma(F) \\ \text{no} & \text{otherwise} \end{cases}$

Example 3. Consider the AF F given in Example 1. For naive semantics, the reasoning problems result in $\mathsf{Enum}_{naive}(F) = \{\{a, c\}, \{a, d\}, \{b, d\}\}$ and $\mathsf{Count}_{naive}(F) = 3$. Furthermore, for argument a we obtain $\mathsf{Cred}_{naive}(a, F) = \text{yes}$ and $\mathsf{Skept}_{naive}(a, F) = \text{no}$. For preferred semantics, F has a single extension $\mathsf{Enum}_{prf}(F) = \{\{a, d\}\}$, $\mathsf{Count}_{prf}(F) = 1$, and thus credulous and skeptical acceptance coincide (e.g., $\mathsf{Cred}_{prf}(a, F) = \mathsf{Skept}_{prf}(a, F) = \text{yes}$).

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. We denote such machines as $\text{NP}^{\mathcal{C}}$ if the underlying Turing machine is non-deterministic. The class $\Sigma_2^P = \text{NP}^{\text{NP}}$ thus denotes the set of problems which can be decided by a non-deterministic 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 has been studied extensively in the literature (see [70] for a starting point). Table 1 summarizes the computational complexity classifications of the defined decision problems [68,12,1,13,71,14,72,73], where $\mathcal{C}$-c denotes that the corresponding problem is complete for class $\mathcal{C}$.

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

3. Reduction-based approaches

In this section we will discuss reduction-based approaches in abstract argumentation. As implied by the name, these methods reduce or translate a reasoning problem to another, typically to another formalism. From a computational point of view, we assure that this reduction is efficiently computable, i.e., achievable in polynomial time, and that the answer for the original problem instance can be immediately obtained from the answer to the new problem instance. 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 distinguished by the target system. Many such approaches have been studied for abstract argumentation ranging from propositional logic [19,18,21,20], constraint satisfaction problems (CSP) [27,23,26,28] and answer-set programming (ASP) [30,74–76] to equational systems [32,77]. We will give an overview of these approaches and in particular focus on the first three very prominent target systems, the reductions to propositional logic, CSP 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 [78] and GRASP [79].

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. We use the truth constants $\top$ to denote *true* and $\bot$ for *false*. In addition, we consider quantified Boolean formulae with the universal quantifier $\forall$ and the existential quantifier $\exists$ (both over atoms), that is, given a formula ϕ, then $Q\,p\phi$ is a QBF, with $Q \in \{\forall, \exists\}$ and $p \in \mathcal{P}$. Furthermore, $Q\,\{p_1, \ldots, p_n\}\phi$ is a shorthand for $Q\,p_1 \cdots Q\,p_n\phi$. 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 $Q\,p$ and bound otherwise. If ϕ contains no free variable, then ϕ is said to be closed and otherwise open. We will write $\phi[p/\psi]$ to denote the result of uniformly substituting each free occurrence of p with ψ in 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 while $p \notin I$ indicates that p evaluates to false. This generalizes to arbitrary formulae in the standard way: Given a formula ϕ and an interpretation I, then ϕ evaluates to true under I (i.e., 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 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 problem 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 [18,20] we consider here uses propositional logic formulae (without quantifiers) to encode the problem of finding admissible sets. Given an AF $F = (A, R)$, for each argument $a \in A$ a propositional variable v_a is used. 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 (below we will present in detail how to translate AFs into formulae). 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 $I \in \mathcal{I}$, there exists an $S \in \mathcal{S}$, such that $\{a \mid v_a \in I, a \in A\} = S$; and
3. $|\mathcal{S}| = |\mathcal{I}|$.

Given an AF $F = (A, R)$ the following formula can be used to solve the enumeration problem of admissible semantics.

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

The models of $adm_{A,R}$ now correspond to the admissible sets of F, i.e., $\mathsf{Enum}_{adm}(F) \cong \{M \mid M \models adm_{A,R}\}$. Taken into consideration that by definition a satisfiable formula has infinitely many models (thus violating item three in Definition 7), it is now possible to restrict the set of models to those containing only atoms occurring in the formula. The first conjunction 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), all its attackers cannot be selected any further. The second conjunct expresses the defense of arguments by stating that, if we accept a, then for each attacker 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 Example 1 is given by

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

Lines (2) to (6) encode the conflict-free property, while lines (7) to (11) ensure that arguments in an admissible set are defended. Note that for convenience, the conjuncts are arranged in a different order than in the definition of $adm_{A,R}$. Consider for instance argument b. Line (3) specifies that if we accept b we cannot accept a and c anymore (conflict-free property). Likewise, line (8) states that b can only be accepted in case it is defended against its attackers. For the attacker c either b itself or d must be accepted. However, since attacker a is not attacked by any other argument, 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 Gabbay [80]. Here, a correspondence between AFs and propositional logic is shown via the Peirce–Quine dagger ("nor") connective.

Furthermore, several papers deal with the converse translation, i.e. translating a Boolean formula in CNF to an AF. Similar as before, for each atom in a formula a corresponding argument is constructed. Accepting such an argument, e.g. under stable semantics, is then interpreted as setting the atom to true. The result is a correspondence between extensions under a specific semantics and satisfying assignments of the formula. Usually, these translations incorporate auxiliary arguments, which are used to simulate the logical connectives. In [12,13] and [81] such methods are studied and used to show complexity bounds or for translations between formalisms.

3.1.2. Reductions to quantified Boolean formulae

Problems beyond NP require a more expressive formalism than Boolean logic. Preferred semantics, for example, is defined as subset-maximal admissible (or complete) sets. Intuitively, we can compute a preferred extension by searching for an admissible set and additionally making sure 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 can be used, making quantified Boolean formulae a well-suited formalism. It is possible to specify preferred semantics in QBFs either via an extension-based, or a labeling-based approach.

First, we consider the extension-based approach from [20]. Here, 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 as $R' = \{(a', b') \mid (a, b) \in R\}$. The following defines a shorthand for comparing two sets of atoms an interpretation is defined upon with respect to the subset-relation.

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

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 $A'_v = \{v_{a'} \mid a' \in A'\}$.

$$prf_{A,R} = adm_{A,R} \wedge \neg \exists A'_v \big((A < A') \wedge adm_{A',R'} \big) \tag{12}$$

In short, we check whether the accepted arguments form an admissible set and whether there exists a proper superset of it which is also admissible. If the former check succeeds and in the latter no such set exists, then we have found a preferred extension. 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., $\mathsf{Enum}_{prf}(F) \cong \{M \mid M \models prf_{A,R}\}$.

The second approach is based on complete labelings (see Definition 4) instead of extensions [19].[9] To this end, we employ four-valued interpretations to express more than two possible states for each argument. In addition to the truth values true and false we also add values undecided and inconsistent. The three labelings in, out and undecided correspond to the first 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 via $p^{\oplus}$ 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: $\{p^{\oplus}, p^{\ominus}\} \subseteq I$ 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 undecided is assigned if neither $p^{\oplus}$ nor $p^{\ominus}$ is in I.

For preferred semantics the encoding is more complex than (12), but the ideas are similar. We begin with formulae for the four truth values. Note that we slightly adapted the representation and formulae from [19] to better match the previous encodings, but the important concepts remain the same.

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

Here, $val(p, v)$ encodes the four possible truth values for a virtual atom p that can be referred to on a sort of meta-level. Actually, instead of p the auxiliary atoms $p^{\oplus}$ and $p^{\ominus}$ are present in the concrete formula. Using this concept, we can specify the labeling formula for each argument in an AF $F = (A, R)$.

$$lab^{t}_{A,R}(a) = val(v_a, t) \rightarrow \bigwedge_{(b,a) \in R} val(v_b, f) \tag{13}$$

$$lab^{f}_{A,R}(a) = val(v_a, f) \rightarrow \bigvee_{(b,a) \in R} val(v_b, t) \tag{14}$$

$$lab^{u}_{A,R}(a) = val(v_a, u) \rightarrow \left(\left(\neg \bigwedge_{(b,a) \in R} val(v_b, f) \right) \wedge \left(\neg \bigvee_{(b,a) \in R} val(v_b, t) \right) \right) \tag{15}$$

These formulae reflect Definition 4: The formulae (13), (14) and (15) encode the in, out and undecided labelings, respectively. For example, (13) can be interpreted in the following way: If an argument a is set to true, then all its attackers must be false. (14) can be interpreted similarly, except that if an atom denotes that an argument is false, then one of its attackers must be true. Finally, (15) states that for any argument to which we assign undecided, it cannot be the case that all its attackers are false or that one of them is true.

[9] A similar approach was recently realized for abstract dialectical frameworks (and thus also for AFs) in the system QADF: http://www.dbai.tuwien.ac.at/proj/adf/qadf/.

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 following formula.

$$3val_A = \bigwedge_{a \in A} \neg val(v_a, i)$$

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

$$com^L_{A,R} := 3val_A \wedge \bigwedge_{a \in A} \left(lab^t_{A,R}(a) \wedge lab^f_{A,R}(a) \wedge lab^u_{A,R}(a) \right)$$

The formula $com^L_{A,R}$ expresses that all the arguments are assigned either true, false or undecided via the $3val_A$ subformula. For each argument, the three conjuncts on the right of the formula encode implications which ensure that the labels are assigned as specified for complete labelings. For example, if a is true, then all its attackers must be false. By applying this, one can encode complete labelings and hence complete extensions. Preferred extensions (or labelings) are expressed as before by subset maximization.

$$A <^L A' = \bigwedge_{a \in A} \left(val(v_a, t) \rightarrow val(v_{a'}, t) \right) \wedge \neg \bigwedge_{a' \in A'} \left(val(v_{a'}, t) \rightarrow val(v_a, t) \right) \tag{16}$$

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

$$prf^L_{A,R} = com^L_{A,R} \wedge \neg \exists A'_v \left(\left(A <^L A' \right) \wedge com^L_{A',R'} \right) \tag{17}$$

For an AF $F = (A, R)$ the following notion of correspondence holds: Let the set of atoms evaluated to true under the four-valued interpretation be $M^t = \{p \mid p^{\oplus} \in M, p^{\ominus} \notin M\}$, then $\mathsf{Enum}_{prf}(F) \cong \{M^t \mid M \models prf^L_{A,R}\}$. 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 expressive power of quantifiers and the labeling approach, the authors of [19] also encode a range of other semantics, for instance semi-stable reasoning, where one can apply the same idea as outlined above, but instead of maximizing the arguments that are in, the arguments that are labeled undecided are minimized.

This results in 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 formulae 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 based on the idea of iteratively searching for models of propositional formulae and has been instantiated in the systems ArgSemSAT [82,22] and CEGARTIX [21,83]. The idea is to use an algorithm which iteratively constructs formulae and searches for models of these formulae. A new formula is generated based on the model of the previous one (or based on the fact that the previous formula is unsatisfiable). At some point the algorithm reaches a final decision and terminates. This is in contrast to so-called monolithic encodings, which formulate the whole problem in a single formula. The encodings in previous sections are examples for such monolithic encodings. The iterative approach is suitable when the problem to be solved cannot be decided in general (under standard complexity theoretic assumptions) by the satisfiability of a single propositional formula (constructible in polynomial time) without quantifiers. This is, for instance, the case with skeptical acceptance under preferred semantics where the corresponding decision problem is Π^P_2 complete. Instead of reducing the problem to a single QBF formula, we delegate the solving task in the iterative scheme to an algorithm querying a SAT solver multiple times.

The algorithms for preferred semantics work roughly as follows. To compute preferred extensions we traverse the search space of a computationally simpler semantics. For instance, we can iteratively search for admissible sets or complete extensions and iteratively extend them until we reach a maximal set, which is a preferred extension. By generating a new candidate admissible set/complete extension, which is not contained in an already visited preferred extension we can enumerate all preferred extensions in this manner. This allows answering credulous and skeptical reasoning as well.

For deciding e.g. skeptical acceptance of an argument under preferred semantics one requires in general an exponential number of calls to the SAT solver (under standard complexity theoretic assumptions). However, the number of SAT calls in the iterative SAT scheme is dependent on the number of preferred extensions of the given AF, see [21].

In the following, we first sketch the CEGARTIX approach from [21] for skeptical acceptance under preferred semantics and subsequently outline the PrefSat approach [82], implemented in the ArgSemSAT system, for enumerating all preferred extensions. Again, we slightly adapted the algorithms for a uniform setting and presentation.

Algorithm 1 decides skeptical acceptance under preferred semantics of an argument a in an AF F. The idea is to proceed from one preferred extension to the next and checking whether a is in one of the extensions. This is encoded in the outer

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: $\phi \leftarrow adm_{A,R}$
2: **while** $\exists I, I \models \phi$ **do**
3: **while** $\exists I', I' \models \psi^I(A, R) \wedge \neg v_a$ **do**
4: $I \leftarrow I'$
5: **end while**
6: **if** $\exists I', I' \models \psi^I(A, R)$ **then**
7: $\phi \leftarrow \phi \wedge \gamma^I$
8: **else**
9: *no*
10: **end if**
11: **end while**
12: *yes*

while loop, lines 2 to 11. The models of formula ϕ represent the remaining admissible sets in the current state of the algorithm. In the beginning, ϕ encodes all admissible sets of F. We start with an admissible set and iteratively extend it while making sure that a is not accepted in this admissible set. This is done in the second loop (lines 3 to 5) and by 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 an admissible set, but also has to be a superset of the current one, specified by I.

If we cannot add further arguments to the admissible set, we check whether we can extend it with having a inside, in line 6. If this is the case, every preferred extension that is a superset of the current admissible set contains a. Hence, we can proceed to a different admissible set 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 (I does not represent a preferred extension) we strengthen the main query ϕ by adding γ^I in line 7, stating that at least one argument currently not accepted in I must be accepted from now on. This ensures that in future iterations we compute admissible sets that are not contained in previously found preferred extensions.

The formulae are defined as follows.

$$\psi^I(A, R) = adm_{A,R} \wedge \bigwedge_{a \in A, v_a \in I} v_a \wedge \left(\bigvee_{a \in A, v_a \notin I} v_a \right)$$

$$\gamma^I = \bigvee_{a \in A, v_a \notin I} v_a.$$

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 admissible sets $\emptyset$, $\{a\}$, $\{d\}$ and $\{a, d\}$ in F. The algorithm now non-deterministically selects one of the admissible sets. Say we pick $\emptyset$. The second while loop then creates a subset maximal admissible set (excluding b) in two iterations, say first adding a and then d. As $\{a, d\}$ is now subset maximal, the second loop terminates. Since this set cannot be extended if we allow to also accept b, we know that we have found a preferred extension. This means we refute the skeptical acceptance of b.

The PrefSat approach [82] is designed to enumerate all preferred extensions utilizing a similar idea. Hereby, also a simpler semantics for traversing the search space is used, but the encodings rely on the concept of labellings (see also Section 4.1). We outline the PrefSat procedure in Algorithm 2.

PrefSat encodes labelings of an AF $F = (A, R)$ by generating three variables per argument, i.e., the set of variables in the constructed formula are $\{I_a, O_a, U_a \mid a \in A\}$. In the final result these variables correspond naturally to a labeling. In particular, a three-valued labeling K corresponds to a model J if $K = (I, O, U)$ with $I = \{a \mid I_a \in J\}$, $O = \{a \mid O_a \in J\}$ and $U = \{a \mid U_a \in J\}$. The following constraint encodes that for every argument exactly one labeling is assigned.

$$\bigwedge_{a \in A} \big((I_a \vee O_a \vee U_a) \wedge (\neg I_a \vee \neg O_a) \wedge (\neg I_a \vee \neg U_a) \wedge (\neg O_a \vee \neg U_a) \big)$$

Furthermore, one can encode the conditions for a labeling to be complete by conjoining certain subformulae. For instance, the formula

$$\bigwedge_{x \in A, \exists (y,x) \in R} \left(\bigwedge_{(x',x) \in R} (\neg I_x \vee O_{x'}) \right)$$

encodes that we can accept x if each attacker is out. The remaining constraints for a labeling to be complete are encoded similarly. Several equivalent formulae for encoding complete labelings have been investigated by Cerutti et al. [82]. Let $com'_{A,R}$ be one of these choices for encoding complete labelings. The formulae for Algorithm 2 are then defined as follows.

Algorithm 2 $\mathsf{Enum}_{prf}(F)$.

Require: AF $F = (A, R)$, argument $a \in A$,
Ensure: returns $\mathsf{Enum}_{prf}(F)$
1: $\mathcal{S} \leftarrow \emptyset$
2: $\phi \leftarrow com'_{A,R}$
3: **while** $\exists J, J \models \phi$ **do**
4: **while** $\exists J', J' \models \psi^J(A, R)$ **do**
5: $J \leftarrow J'$
6: **end while**
7: $\mathcal{S} \leftarrow \mathcal{S} \cup \{\{a \mid I_a \in J\}\}$
8: $\phi \leftarrow \phi \wedge \gamma^J$
9: **end while**
10: return $\mathcal{S}$

$$\psi^J(A, R) = com'_{A,R} \wedge \bigwedge_{a \in A, I_a \in J} I_a \wedge \left(\bigvee_{a \in A, I_a \notin J} I_a \right)$$

$$\gamma^J = \bigvee_{a \in A, I_a \notin J} I_a.$$

Algorithm 2 traverses the space of complete labelings in two while loops. The outer while loop computes candidate complete labelings, which are not smaller than previously found preferred labelings (when comparing only the arguments assigned *in*). The inner while loop searches for a maximal preferred labeling. If the inner while loop terminates, then the corresponding preferred labeling/extension is added to the solution set $\mathcal{S}$. In line 8 we exclude smaller complete labelings from subsequent iterations. In [82] the algorithm for enumerating all preferred extensions contains further refinements, such as restricting the search space to non-empty complete extensions.

3.1.4. Reasoning problems

The first two reductions presented in Section 3.1.1 and Section 3.1.2 immediately solve problems of enumerating extensions. Deciding credulous and skeptical reasoning is typically easy to achieve. In order to decide $\mathsf{Cred}_\sigma(a, F)$ one can conjoin the base formula with an atom corresponding to the acceptance of argument a. If there exists a model, a is credulously accepted. Adding the atom for a in negated form to the formula decides if a is not skeptically accepted, i.e., if there exists a model, an extension does not contain a.

Similarly, the iterative SAT scheme of Algorithm 1 (see Section 3.1.3) can be adapted to solve credulous reasoning by adding the atom to be queried positively instead of negatively in line 3. Regarding the enumeration problem, Algorithm 2 enumerates all preferred extensions using iterative SAT.

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.

3.2. CSP-based approach

In this subsection we consider reductions to Constraint Satisfaction Problems (CSPs) [84], which allow solving combinatorial search problems. The approach of CSP is inherently related to propositional logic reductions as introduced in Subsection 3.1.1, see also [85] for a formal analysis of the relation between the two approaches.

A CSP can generally be described by a triple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is the set of variables, $D = \{D_1, \ldots, D_n\}$ is a set of finite domains for the variables and $C = \{c_1, \ldots, c_m\}$ a set of constraints. Each constraint c_i is a pair (h_i, H_i) where $h_i = (x_{i1}, \ldots, x_{ik})$ is a k-tuple of variables and H_i is a k-ary relation over D. In particular, H_i is a subset of all possible variable values representing the allowed combinations of simultaneous values for the variables in h_i. An assignment v is a mapping that assigns to every variable $x_i \in X$ an element $v(x_i) \in D_i$. An assignment v satisfies a constraint $((x_{i1}, \ldots, x_{ik}), H_i) \in C$ iff $(v(x_{i1}), \ldots, v(x_{ik})) \in H_i$. Finally, a solution is an assignment v to all variables such that all constraints are satisfied, denoted by $(v(x_1), \ldots, v(x_n))$.

Finding a valid assignment of a CSP is in general NP-complete. Nevertheless, several programming libraries support constraint programming, like ECLiPSe, SWI Prolog, Gecode, JaCoP, Choco, Turtle (just to mention some of them) and allow for efficient implementations of CSPs. These constraint programming solvers make use of techniques like backtracking and local search.

Computing argumentation semantics via constraint programming has been addressed mainly by Amgoud and Devred [23] and Bistarelli and Santini [26–28], where the latter provide the system ConArg which is able to compute a wide range of semantics for abstract argumentation frameworks.

3.2.1. Mappings of AFs to CSPs

Given an AF $F = (A, R)$, the associated CSP (X, D, C) is specified as $X = A$ and for each $a_i \in X$, $D_i = \{0, 1\}$. The constraints are formulated depending on the specific semantics σ. For example, solutions that correspond to conflict-free sets can be obtained by defining a constraint for each pair of arguments a and b with $(a, b) \in R$, where the two variables may not be

set to 1 at the same time. Here, the constraint is of the form $((a, b), ((0, 0), (0, 1), (1, 0)))$ which is equivalent to the cases when the propositional formula $(a \rightarrow \neg b)$ evaluates to true.

In the following, we will use the notation from [23], because it reflects the similarities between the CSP approach and the reductions to propositional logic as outlined above.

For admissible semantics we get the following constraints.

$$C_{adm} = \left\{ \left(a \rightarrow \bigwedge_{b:(b,a)\in R} \neg b \right) \wedge \left(a \rightarrow \bigwedge_{b:(b,a)\in R} \left(\bigvee_{c:(c,b)\in R} c \right) \right) \,\middle|\, a \in A \right\} \tag{18}$$

The first part ensures conflict-free sets and the second part encodes the defense of arguments. Then, for an AF $F = (A, R)$ and its associated admissible CSP (X, D, C_{adm}), $(v(x_1), \ldots, v(x_n))$ is a solution of the CSP iff the set $\{x_j, \ldots, x_k\}$ s.t. $v(x_i) = 1$ is an admissible set in F.

Example 6. For the AF of Example 1 on Page 30 we obtain the following admissible CSP (X, D, C_{adm}). $X = A$, for each $a_i \in X$ we have $D_i = \{0, 1\}$ and

$$\begin{aligned} C_{adm} = \big\{ & (a \rightarrow \top) \wedge (a \rightarrow \top), (b \rightarrow \neg a \wedge \neg c) \wedge (b \rightarrow \bot \wedge d), \\ & (c \rightarrow \neg b \wedge \neg d) \wedge \big(c \rightarrow (a \vee c) \wedge \bot\big), (d \rightarrow \top) \wedge (d \rightarrow \top), \\ & (e \rightarrow \neg d \wedge \neg e) \wedge (e \rightarrow \bot \vee d) \big\}. \end{aligned}$$

This CSP has the following solutions: $(0, 0, 0, 0, 0)$, $(1, 0, 0, 0, 0)$, $(0, 0, 0, 1, 0)$, $(1, 0, 0, 1, 0)$ which correspond to the admissible sets of F, namely $\{\}$, $\{a\}$, $\{d\}$ and $\{a, d\}$.

Most CSP solvers do not support subset maximization. Thus, for preferred semantics, the authors in [28] propose an approach that iteratively computes admissible/complete extensions and adds constraints to exclude certain sets, such that one finally obtains the preferred extensions.

3.2.2. Reasoning problems

Similarly to the reductions to propositional logic, one can decide the following reasoning problems with CSPs, namely verification, skeptical and credulous reasoning. Furthermore, enumeration is usually supported by modern CSP solvers.

3.3. ASP-based approach

Answer-set programming (ASP, for short) [86,87], also known as A-Prolog [88,89], 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., [90,91]) nowadays not only support a rich language but also are capable of solving hard problems efficiently. Furthermore, the declarative approach of ASP results in 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 [29] for a survey), including the approach proposed by Nieves et al. [74] where the program is re-computed for every input instance, Wakaki and Nitta [76] who use labeling-based semantics and the approach by Egly et al. [30] which follows extension-based semantics. Here, we focus 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 front-end.

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* [92] (which is unfortunately hardly accessible for non-experts in ASP) and a second one which makes use of `metasp` encodings [93] (allowing to specify subset minimization via a single simple statement). Additionally, we discuss how reasoning problems can be specified.

3.3.1. Answer-set programming

We give a brief overview of the syntax and semantics of disjunctive logic programs under the answer-set semantics [94]; for further background, see [95,91].

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

$$\#minimize[l_1 = w_1@J_1, \ldots, l_k = w_k@J_k] \tag{19}$$

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 (19), $Y \models l$ 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 [93].

3.3.2. Basic encodings

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

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

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 $a \in S$.

Similar to Definition 7, we define the subsequent 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

1. for each $S \in \mathcal{S}$, there exists an $I \in \mathcal{I}$, such that $\{a \mid \mathrm{in}(a) \in I\} = S$;
2. for each $I \in \mathcal{I}$, there exists an $S \in \mathcal{S}$, such that $\{a \mid \mathrm{in}(a) \in I\} = S$; and
3. $|\mathcal{S}| = |\mathcal{I}|$.

Next, we will successively introduce the rules our queries are composed of. 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:

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

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

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

For any AF $F = (A, R)$, the admissible sets of F correspond to the answer sets of π_{adm} augmented by $\widehat{F}$, i.e. $adm(F) \cong \mathcal{AS}(\pi_{adm}(\widehat{F}))$.

Sometimes we have to avoid the use of negation. This might either be the case for the saturation technique (described below), or if the problem can be encoded 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 lt/2, inf/1, succ/2 and sup/1 denote lower than, infimum, successor and supremum of the order $<$. The predicates ninf/1, nsup/1 and nsucc/2 are used to derive that an element is not the infimum or the supremum, or respectively an element is not the successor of the other w.r.t. the order $<$.

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

3.3.3. Saturation encodings

To compute the preferred extensions of an argumentation framework, we will use the saturation technique as follows: Having computed an admissible set S (characterized via predicates in$(\cdot)$ and out$(\cdot)$ using encoding $\pi_{adm}(\widehat{F})$), we perform a second guess with new predicates, say inN$(\cdot)$ and outN$(\cdot)$, to represent a 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 inN$(\cdot)$ and outN$(\cdot)$) characterizes an admissible set. The saturation module $\pi_{satpref}$ looks as follows.

$$\begin{aligned}\pi_{satpref} = \big\{&\mathrm{inN}(X) \vee \mathrm{outN}(X) \leftarrow \mathrm{out}(X); && (20)\\ &\mathrm{inN}(X) \leftarrow \mathrm{in}(X); && (21)\\ &\mathrm{fail} \leftarrow \mathrm{eq}; && (22)\\ &\mathrm{fail} \leftarrow \mathrm{inN}(X), \mathrm{inN}(Y), \mathrm{att}(X,Y); && (23)\\ &\mathrm{fail} \leftarrow \mathrm{inN}(X), \mathrm{outN}(Y), \mathrm{att}(Y,X), \mathrm{undefeated}(Y); && (24)\\ &\mathrm{inN}(X) \leftarrow \mathrm{fail}, \mathrm{arg}(X); && (25)\\ &\mathrm{outN}(X) \leftarrow \mathrm{fail}, \mathrm{arg}(X); && (26)\\ &\leftarrow \mathit{not}\ \mathrm{fail}\big\}. && (27)\end{aligned}$$

Let us for the moment also assume that predicates eq (rule (22)) and undefeated$(\cdot)$ (rule (24)) 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 in$(\cdot)$ and out$(\cdot)$ equals the second guess T via inN$(\cdot)$ and outN$(\cdot)$; in other words, eq is derived if $S = T$;
- undefeated(a) is derived if argument a is not defeated in F by the second guess T.

In the following, we discuss the functioning of $\pi_{satpref}$ when conjoined with the program $\pi_{adm}(\widehat{F})$ for a given AF F. First, rule (20) guesses a set $T \subseteq A$ as already discussed above. Rule (21) ensures that the new guess satisfies $S \subseteq T$.

The task of the rules (22)–(24) is to check whether the new guess T is a proper superset of S and characterizes an admissible set of the given AF F. If this is not the case, we derive the predicate fail. More specifically, we derive fail if either $S = T$ (rule (22)); T is not conflict-free in F (rule (23)); or T contains an argument not defended by T in F (rule (24)). 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 (25)–(27) 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 in$(\cdot)$ and out$(\cdot)$) is a preferred extension of the given AF $F = (A, R)$. Hence, for each $T \supset S$, T is not admissible. Consequently, *every* new guess T (via predicates inN$(\cdot)$ and outN$(\cdot)$) derives fail. Thus, we have no interpretation without predicate fail that satisfies $\pi_{satpref}$. However, the saturated interpretation, which contains fail and both inN(a) and 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 $T \supset S$, where 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 (20)–(26). But this shows that we cannot have an answer set J which characterizes S. Due to rule (27) such an answer set J has to contain fail and by rules (25) and (26), 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 (27). In other words, there is an $I \subset J$ satisfying the reduct and thus J cannot be an answer set. This, however, is the desired outcome, since the initial guess S characterized by J is not a preferred extension.

We still have to define the rules for the predicates eq and 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 undefeated$(\cdot)$ are computed via predicates eq_upto$(\cdot)$ (resp., undefeated_upto$(\cdot,\cdot)$) in the modules π_{eq} and $\pi_{undefeated}$, which are defined as follows.

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

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, i.e. $prf(F) \cong \mathcal{AS}(\pi_{prf}(\widehat{F}))$.

3.3.4. `metasp` *encodings*

The following encodings for preferred semantics use the *#minimize* statement when evaluated with the subset-minimization semantics provided by `metasp` [93]. 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 operates as follows: The ASP encoding to solve is given to the grounder `gringo`[10] 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 that 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 program execution call.

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

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

[10] For this technique one should use `gringo` version 3.0.5 or lower, as later versions don't support the `--reify` option.

As a result we get 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, i.e. $prf(F) \cong \mathcal{AS}(\pi_{prf\_metasp}(\widehat{F}))$.

3.3.5. 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 and also credulous and skeptical reasoning. For the `metasp` variant, the meta encodings can be augmented with constraints to achieve credulous and skeptical reasoning. Usually, when one uses an ASP program, like π_{prf} for preferred semantics, for an AF F, the ASP solver returns all answer sets of the program which correspond to all preferred extensions of F. For credulous or skeptical acceptance, one uses the respective reasoning problems of the ASP solver, which are sometimes referred to as *brave* or *cautious* reasoning. Note that the concrete usage depends on the particular solver.

We exemplify this option for the solver `dlv`. To compute all arguments which are skeptically accepted under preferred semantics for a given AF F, we invoke

$$\texttt{dlv -cautious}\ \pi_{prf}(\widehat{F}) \cup \{\mathrm{in}(X)?\}$$

3.4. Further reduction-based approaches

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

3.4.1. SAT extensions

Several extensions to the prototypical problem of SAT have been developed. Two interesting approaches include so-called *minimal correction sets* (MCSes) [96] and the *backbone* [97] of a Boolean formula in CNF. The former consists of a minimal subset of the clauses of an unsatisfiable formula, such that if we remove these clauses the resulting formula becomes satisfiable. A backbone of a satisfiable formula is simply the set of literals it logically entails. These concepts can be used to answer several reasoning problems for AFs and optimize existing algorithms. In [98] MCSes are applied to solve reasoning problems for semi-stable semantics by encoding the range of a set as satisfied clauses, and techniques for computing backbones are utilized to enhance the efficiency of the algorithms.

3.4.2. 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 was proposed by Gabbay [32,77]. Here, one receives a system of equations where each argument is represented by a distinct variable with a domain of real numbers in the interval $[0, 1]$. Solutions to these systems of equations assign to each variable a number from the domain. The variable assignment corresponds to a labeling. If the variable of an argument a is mapped to 1, a is in, if it is mapped to 0 then it is out and undecided otherwise. This allows for easy identification of the labelings by inspecting the variable assignments. For instance, to compute complete labelings in this way we can consider the equation $f(a) = 1 - max(f(x_1), \ldots, f(x_n))$ where $\{x_1, \ldots, x_n\}$ are the attackers of a. If all attackers of a are mapped to 0, then the value of a will be 1. If at least one of the attackers is "in", i.e. it is mapped to 1, then $f(a) = 0$. Otherwise, if all attackers are undecided with $1 < f(x_i) < 0$ for each $1 \leq i \leq n$, then a will be undecided.

Another work on equational approaches is by Osorio et al. [99,100]. They encode preferred and semi-stable semantics as integer programs and then use the solver Xpress[11] to compute the extensions.

3.4.3. Monadic second order logic

A reduction approach going beyond pure propositional logic is to encode 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 whether I is a model of the formula. In [33] the authors encode several reasoning problems for AFs into an MSO formula φ. A given AF is then transformed into an interpretation I and one decides the reasoning problem by testing whether I is a model of φ. In this work certain building blocks for such encodings are introduced, which facilitate straightforward reductions of the different semantics to MSO. While the first MSO-encodings for abstract argumentation [101,14] were introduced to obtain complexity-theoretic results in terms of tree-width, the advent of efficient systems for MSO [102,103] turns MSO-encodings into an interesting alternative to implement abstract argumentation via the reduction method.

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

[11] http://www.fico.com/en/products/fico-xpress-optimization-suite/.

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 problems 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* [104,35,38], 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 [69]. 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 [34]; many further materializations of this concept can be found in the literature (see, e.g., [35,38,36]). 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. 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 3 is an example for a labeling-based algorithm for computing preferred labelings in the spirit of [35].

The main idea of Algorithm 3 is to start with the labeling that marks all arguments with *in* (the set containing all arguments) and to re-label arguments to either *out* or *undec* until the set becomes admissible. This strategy prevents the algorithm from considering all the (relatively small) admissible sets as candidates for preferred extension like other algorithms do (compare Algorithm 4).

Let us explain Algorithm 3 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 an attacker that is not labeled *out*, and relabels it. We call such a relabel step a transition step. In Algorithm 3 a transition step is due to the following rules. First, the argument a is labeled to *out* and then all arguments $y \in \{a\}^+$ (a and all arguments attacked by a) are checked for being valid labeled *out*, i.e., in case y is neither labeled *in* nor is attacked by an argument labeled *in*, we change the label of y to *undec*. In [35] 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 3. 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, [35] 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

Algorithm 3 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** $a \in \mathcal{L}_{in}$ 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**

step. As such duplicates are indeed a waste of computational resources, this is a weak point. Other algorithms [34,36] avoid such duplicates as they use a different strategy to branch in the search space (see, for instance, Algorithm 4).

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, [34] used a criterion for maximality that does not make use of the other extensions explicitly. Instead, it exploits 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 correspond to stable extensions). Secondly, in [36] 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 4).

Let us thus have a closer look on Algorithm 4 next. This algorithm for preferred semantics follows the work of [34] and [36]. The main difference to Algorithm 3 is the way the search space is explored. Starting with all arguments being unlabeled in each step the algorithm chooses one (unlabeled) argument and branches between 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 $\langle \mathcal{L}_{in}, \mathcal{L}_{out}, \mathcal{L}_{att}, \mathcal{L}_{undec} \rangle$.[12] The intuition behind the labels *in* and *out* is the same as for three valued labelings, while arguments which attack an *in* labeled argument, but are not attacked by such an argument, are labeled *att*. Finally, 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 4 is the potential exponential number of admissible labelings (even for a small number of preferred extensions) which are all considered by the algorithm.

Let us briefly compare Algorithm 3 and Algorithm 4 on two extreme cases: (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. In F_1 the empty set is the only admissible set and thus also the only preferred extension. As there is just one admissible set, Algorithm 4 never branches and thus terminates after a linear number of steps. However Algorithm 3 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 4 considers all $2^{|A|}$ admissible sets. In contrast, Algorithm 3 terminates immediately. As different labeling-based algorithms behave good on different kind of argumentation frameworks, empirical evaluations are an important issue. A first step in

[12] In order to present several algorithms in this survey in a uniform and easily comparable way, we use different names for the labels compared to the original works [34,36].

Algorithm 4 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**

that direction is done in the work of Nofal et al. [105,36,37] where the performance of different labeling-based algorithms for preferred semantics is compared on randomly generated AFs.

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 [35]; an algorithm for admissible labelings can be easily obtained from Algorithm 4 (by dropping the $\subseteq$-maximality test in line 14); for complete semantics one can adapt Algorithm 3; for stable semantics, see [35]; algorithms for semi-stable and stage semantics can be found in [104,106,35]. Recently, [105] studied improved algorithms for enumerating grounded, complete, stable, semi-stable and stage semantics. Labeling-based algorithms are implemented in the ArguLab system [107] as well as in ArgTools [36] (see also Section 5.1).

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 simply 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 [38], a credulous acceptance algorithm for preferred semantics, which is implemented in the *CompArg* system (see Section 5.1).

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 5 starts with labeling the queried argument with *in* and all the other arguments with *undec*. Then, it iterates the following two steps. First, check whether 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, we pick for each argument a which is labeled *out* but not attacked by an argument labeled *in*, an *undec* labeled attacker b of a and label it with *in*. In 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 5. In contrast to the previous algorithm, we store several partial extensions (partial proofs) at the same time and also terminate as soon as we have found an admissible set. In line 1 of the algorithm one has to decide whether to use a queue or a stack for storing these partial proofs. The choice determines the search strategy in the space of partial proofs: The former would give a breadth-first search (as suggested in [38]) while the latter yields a depth-first search.

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 cannot 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 idea to store already considered partial proofs.

Finally, let us mention that beside the work of Verheij [38], Doutre and Mengin [34] suggest to start from an enumeration algorithm similar to Algorithm 4 but employing several shortcuts for credulous and skeptical reasoning.

Algorithm 5 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 = ∅
2: partialProofs.push(⟨{a}, ∅, A \ {a}⟩)
3: while L = partialProofs.pop() do
4:   if ∄x ∈ L_undec : x ⇀^R L_in then
5:     return L
6:   else
7:     for x ∈ L_undec s.t. x ⇀^R L_in do
8:       set L(x) = out
9:     end for
10:  end if
11:  for L'_in ∈ cf(F) : L'_in ⊇ L_in is ⊆-min s.t. L_out ⊆ L'^⊕_in do
12:    partialProofs.push(⟨L'_in, L_out, A \ (L'_in ∪ L_out)⟩)
13:  end for
14: end while
```

4.2. Dialectical proof-procedures (dialogue games)

A popular approach for obtaining proof procedures for abstract argumentation is based on so called dialogue games (see, e.g., [35,39]). 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 cannot raise any argument. Typically, an argument a being accepted is equivalent to one player having a winning strategy in the dialogue game when started 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 [108]. By their nature, dialogue games are typically dedicated to a specific reasoning problem, but sometimes they can also be used to actually compute extensions.

Such algorithms are implemented in the *Dungine* system [109], in the *Dung-O-Matic* Java library (see Section 5.1) and also are used in Visser's Epistemic and Practical Reasoner,[13] a tool for argumentation with propositional languages (however, it is not a dedicated tool for abstract argumentation).

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 borrowed from [35]. In both cases the game is started by Pro raising the argument in question, and then Pro and Opp 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 problems differ in the allowed moves for the players, where typically Pro and Opp have different rule sets for legal moves.

A game for grounded semantics First, consider a game that provides, 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 [35].

Example 7. 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.

[13] http://www.wietskevisser.nl/research/epr/.

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. The latter is well known to be equivalent to argument a being credulously accepted with preferred semantics [35].

Example 8. 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 cannot use b again. Hence, Pro has a winning strategy for a. Notice that in the grounded game Pro and Opp would loop forever with raising a and b.

The critical reader might observe that such dialogue games are indeed no 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 [39,110]). The resulting algorithms are indeed of a similar type as the previously discussed labeling-based algorithms.

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 or 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 parameterized complexity theory. The main observation is that binding a certain problem parameter to a fixed constant makes many of the intractable problems tractable. This property is referred to as *fixed-parameter tractability* (FPT) (see, e.g., [111]). The complexity class FPT consists of problems that can be computed in $f(k) \cdot n^{\mathcal{O}(1)}$ 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 that is defined along so-called tree decompositions. Intuitively, the tree-width measures the tree-likeliness of a graph, in particular connected graphs of tree-width 1 are exactly trees. AFs can be seen as directed graphs and therefore the parameter tree-width can be directly applied to them. Indeed, many argumentation problems have been shown to be solvable in linear time for AFs of bounded tree-width [41,33].

In this section we present a dynamic-programming based approach for abstract argumentation that is defined on tree decompositions. First introduced in [112], this approach especially aimed at the development of efficient algorithms that turn complexity-theoretic results into practice. The algorithms from [112] are capable of solving credulous and skeptical reasoning problems under admissible and preferred semantics. Later, this approach was extended to work with stable and complete semantics [42]. Further fixed-parameter tractability results were obtained for AFs with bounded clique-with [113] and in the work on backdoor sets for argumentation [114]. Negative results for other graph parameters like bounded cycle-rank, directed path-width, and Kelly-width can be found in [40].

The approach presented here is put into practice in the dynPARTIX tool [42,115] as well as in the D-FLAT system [102, 116]. While the former is a dedicated tool for argumentation (see also Section 5), the latter is a general framework that allows one to declaratively specify algorithms on tree decompositions by means of ASP.

In the following, we first introduce tree decompositions. We then present the general idea behind the dynamic-programming algorithms and provide an example based on admissible semantics. Finally, we discuss how the basic algorithm that enumerates all admissible sets can be adapted for other reasoning problems.

4.3.1. Tree decompositions

A tree decomposition [117] 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) $(v_i, v_j) \in E \Rightarrow \exists t \in V_\mathcal{T} : \{v_i, v_j\} \subseteq X_t$

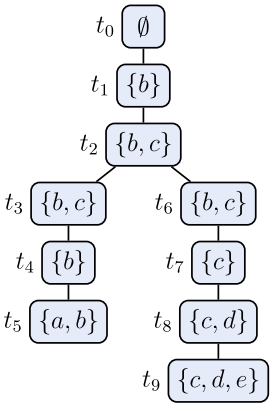


Fig. 3. Normalized tree decomposition.

(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.

Conditions (i) and (ii) guarantee that no information of 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 $\mathcal{T}$. In general a graph may have multiple tree decompositions. 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 [118]. Normalized tree decompositions comply with Definition 9, but only consist of the following four different node types:

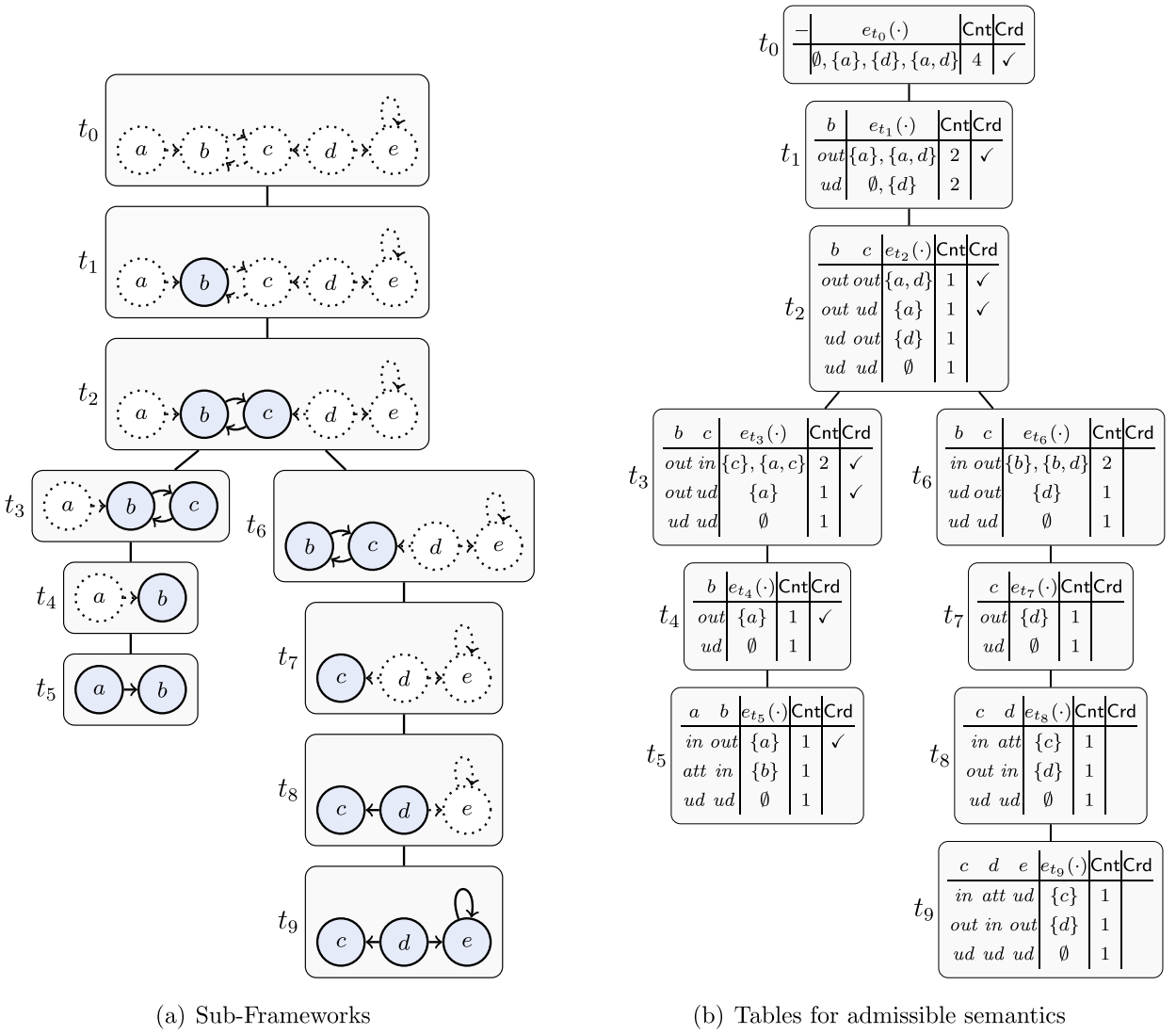
1. *JOIN* node: A node t which has two children t' and t'', $X_t = X_{t'} = X_{t''}$.
2. *INTRODUCTION* node: A node t having exactly one child t' s.t. $|X_t| = |X_{t'}| + 1$ and $X_{t'} \subset X_t$.
3. *REMOVAL* node: A node t having exactly one child t' s.t. $|X_t| = |X_{t'}| - 1$ and $X_t \subset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

Additionally, we assume that for root node r of the normalized tree-decomposition, we have $X_r = \emptyset$. 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. 3 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 [119]. However, the problem is fixed parameter tractable w.r.t. treewidth [120]. Thus, for AFs with small treewidth an optimal tree decomposition can be computed efficiently. Moreover, there exist several heuristic-based algorithms that provide "good" tree decompositions in polynomial time (see, e.g., [121,15,122]).

4.3.2. Dynamic programming

In the following, we present a dynamic-programming algorithm for computing admissible sets (and deciding credulous acceptance of arguments) as proposed in [112]. Algorithms for other semantics and reasoning problems 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 we compute so-called colorings, a data structure to represent (partial) solutions for our problem. These colorings are computed based on the arguments contained in the bag of the current node as well as colorings from the child node(s). The idea of dynamic programming is hereby realized as follows: If we encounter a REMOVAL node during bottom-up traversal then we can exploit the fact that the "removed" node will never reappear in another bag later on during the traversal. We can therefore discard information for (partial) solutions in case it contains a removed argument



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

Fig. 4. Normalized tree decomposition in action.

that does not fulfill the properties of our semantics. The solutions for the whole input instance can be obtained by a 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, i.e., $X_{\geq t}$ contains all arguments of this sub-tree. For instance in the tree decomposition from Fig. 3 we have $X_{\geq t_3} = X_{t_3} \cup X_{t_2} \cup X_{t_1} = \{a, b, c\}$. 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 another bag). Regarding the example we have $X_{>t_3} = (X_{t_2} \cup X_{t_1}) \setminus X_{t_3} = \{a\}$. 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. 4(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 rooted at t. Hence, these arguments are already completely processed by the algorithm and we can define $X_{>t}$*-restricted admissible sets*. We distinguish between attacks from arguments in $X_{>t}$ and X_t. While attacks from arguments X_t might be counter attacked by arguments appearing later, i.e.,

somewhere above in the tree decomposition, this cannot happen for arguments in $X_{>t}$. Thus, we define an $X_{>t}$-restricted admissible set S for a sub-framework $F_{\geq t}$ such that first S has to be conflict-free and second it has to defend itself against the arguments in $X_{>t} \setminus S$. So the $X_{>t}$-restricted admissible sets for a sub-framework $F_{\geq t}$ are all the sets that might become admissible in a framework $F_{\geq t'}$ for some node t' 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 4. The main difference is that a labeling labels the whole set of arguments while colorings are only applied to a small part of the set of arguments, even if other parts have already been considered.[14] 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 $F_{\geq t}$ that satisfy the following conditions for each $a \in X_t$:

$$
\begin{aligned}
C(a) &= in \quad \text{iff } a \in S \\
C(a) &= out \quad \text{iff } S \rightarrowtail^R a \\
C(a) &= att \quad \text{iff } S \not\rightarrowtail^R a \text{ and } a \rightarrowtail^R S \\
C(a) &= ud \quad \text{iff } S \not\rightarrowtail^R a \text{ and } a \not\rightarrowtail^R S
\end{aligned}
$$

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

Goal Our overall goal is to compute admissible sets of an AF. 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 [112], 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 sets 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 [40], we shall just sketch the intuition behind them here.

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

LEAF node: Here we have $F_t = F_{\geq t}$ and thus the restricted admissible sets are just the conflict-free sets. So we compute the conflict-free sets of F_t and then build a coloring for each conflict-free set S as follows:

$$
\begin{aligned}
C(x) &= in \quad \text{if } x \in S \\
C(x) &= out \quad \text{if } S \rightarrowtail^R x; \\
C(x) &= att \quad \text{if } x \rightarrowtail^R S \text{ and } S \not\rightarrowtail^R x \\
C(x) &= ud \quad \text{otherwise}
\end{aligned}
$$

Example 9. Consider the *leaf node* t_5 with bag $\{a, b\}$ in Fig. 4(b). The computed colorings represent the conflict-free (and $\emptyset$-restricted admissible) sets for $F_{\geq t_5}$. For instance, the second labeling C with $C(a) = att, C(b) = in$ corresponds to the $\emptyset$-restricted admissible/conflict-free set $\{b\}$, which however is not admissible for $F_{\geq t_5}$.

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 simply 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

[14] Notice that we use different names for the colors than the original work [112]. This is to present the dynamic programming algorithm in a uniform setting with the labeling-based algorithms presented before.

neighbors of a were already considered we know that the corresponding sets cannot 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 were considered will never do so.

Example 10. Node t_4 in Fig. 4(b) 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 $F_{\geq t_4}$.

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.

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

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

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. That is, if $(a,b) \in F_t$ then b is labeled *out* and if $(b,a) \in F_t$ and b is not already labeled *out* then it is labeled *att*.

Example 11. In node t_3 in Fig. 4(b) argument c is introduced, i.e. t_3 is an *introduction node*. 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, by adding 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 $F_{\geq t_3}$.

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 12. In the *join node* t_2 in Fig. 4(b) 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 $F_{\geq t_2}$, namely $\{a,d\}$.

4.3.3. Reasoning problems

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

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. 4(b) this result is represented by the column $e_{t_0}(\cdot)$ in node t_0.

Table 2
Argumentation systems: overview.

System name	URL	Section (type)	Reference
ArgSemSAT	http://sourceforge.net/projects/argsemsat/	3.1.3 (reduction, iterative SAT)	[82,22]
ArgTools	http://sourceforge.net/projects/argtools/	4.1 (direct, labelings)	[36]
ASPARTIX	http://www.dbai.tuwien.ac.at/proj/argumentation/systempage/	3.3 (reduction, ASP)	[30]
CEGARTIX	http://www.dbai.tuwien.ac.at/proj/argumentation/cegartix/	3.1.3 (reduction, iterative SAT)	[21]
CompArg	http://www.ai.rug.nl/~verheij/comparg/	4.1 (direct, labelings)	[38]
ConArg	http://www.dmi.unipg.it/conarg/	3.2 (reduction, CSP)	[26]
Dung-O-Matic	http://www.arg.dundee.ac.uk/?page_id=279	4.2 (direct, dialogue)	–
Dungine (ArgKit)	http://www.argkit.org/	4.2 (direct, dialogue)	[109]
dynPARTIX	http://www.dbai.tuwien.ac.at/proj/argumentation/dynpartix/	4.3 (direct, decomposition)	[115]
pyAAL (+ArguLab)	http://code.google.com/p/pyafl/	4.1 (direct, labelings)	[107]

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. 4(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. 4(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 (see [40]).

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 [40].

As preferred extensions are subset-maximal admissible sets 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.

5. System comparison

In this section we provide an overview on systems that implement the approaches presented earlier. We focus here on a comparison of the systems w.r.t. their features (e.g. supported semantics and reasoning problems) and underlying concepts. Our goal is to provide a comprehensive study of the strengths of each tool, where the reader can look up the appropriate tool for the problem at hand. The features of the presented systems are naturally subject to change in the future. We note that the landscape of currently available software is very heterogeneous: Some tools are tailored to graphical representations of the used algorithms and results whereas others are particularly tuned towards performance. Within the argumentation community there is currently no consensus on which instances are representative for comparing different implementations w.r.t. performance. Moreover, independent benchmark suites are not available. Hence, a systematic, fair and longer-term stable comparison w.r.t. run-time performance is currently not possible, and we refer to currently ongoing developments within the community that seek for a standardized system competition of argumentation systems [48,123,43]. Nevertheless, where available, we give references to articles that deal with a performance comparison of particular tools.

Table 2 summarizes systems for abstract argumentation. The URL links to the web page of the respective system, where source code and documentation or the web front-end (if available) can be found. Additionally, the table contains a reference to the section where the algorithms underlying the tool are discussed. The last column contains the reference to the main article of the tool. In case no particular article on the tool was published, we reference here the paper that presents the theoretical background of the tool.

Table 3 lists the technical characteristics of the considered systems. The “GUI” column not only indicates the availability of a full-fledged graphical user interface but also contains information about availability of front-ends for demonstration

Table 3
Argumentation systems: technical details.

System name	Platform	Language	GUI	Command line	Library
ArgSemSAT	independent	C++	–	yes	no
ArgTools	independent	C++	–	yes	no
ASPARTIX	as ASP solver	ASP	web	yes	no
CEGARTIX	Unix	C++	–	yes	no
CompArg	Windows	Delphi	stand-alone	no	no
ConArg	independent	Java, C++	stand-alone	yes	yes
Dung-O-Matic	independent	Java	web (via OVAGen)	no	yes
Dungine (ArgKit)	independent	Java	web (via OVAGen), demo GUI	no	yes
dynPARTIX	Unix	C++	–	yes	no
PyAAL (+ArguLab)	independent	Python	web (ArguLab)	yes	no

Table 4
System capabilities showing which system can reason C(redulously), S(keptically) or is able to E(numerate), respectively V(erify) a solution. Implicit reasoning support is denoted by the italic letters *C* and *S*.

	naive	*grd*	*adm*	*com*	*prf*	*stb*	*sem*	*stg*
ArgSemSAT		E		E				
ArgTools		E	*C*	*C*, E	C, S, E	E		
ASPARTIX	C, S, E		C, E		C, S, E			
CEGARTIX					S		C, S	
CompArg		E	*C*		C, E	E		
ConArg		E	C, E	C, S, E	V, E	C, S, E	E	
Dung-O-Matic		E						
Dungine		C, S	*C*	*C*, *S*	C			
dynPARTIX		*C*, *S*	C, E	C,S,E				
PyAAL		C,S,E	*C*	*C*, *S*	C,S,E			

purposes. Column “command line” denotes that the software is accessible via command line interface, and “library” specifies that the implementation can be accessed via a specified software interface.

In Table 4 we provide an overview on the supported semantics and reasoning problems of the systems. Note that this table only contains the semantics and reasoning problems we consider throughout this work (see Section 2). Additionally, we also include *implicit* reasoning support in the table, denoted by the italic letters *C* and *S*. That is, we know that credulous reasoning yields the same answer for any AF and argument w.r.t. preferred, complete and admissible semantics. Similarly, skeptical reasoning for complete and grounded semantics return the same result. Since the grounded extension is unique, credulous and skeptical reasoning are equivalent for grounded semantics.

The strengths of each tool are summarized in Section 5.1. There, we go into detail of system-specific characteristics, such as particular GUI-based features, support for additional reasoning problems or performance-relevant details. If a system is capable of computing further semantics, such as ideal [124], eager [125], cf2 [126], stage2 [127], resolution-based grounded [128] we note this in the corresponding system paragraph.

5.1. System properties

ArgSemSAT The high-performance system ArgSemSAT is built on top of modern SAT solvers in such that it incorporates an iterative SAT-procedure. In particular, it implements the PrefSat approach [82] from Section 3.1.3. The procedure relies on iteratively generating complete extensions/labelings and extending them iteratively to preferred extensions. A similar approach is taken by CEGARTIX, where skeptical acceptance of preferred semantics (among other query-based reasoning problems) is computed. The implementation of PrefSat showed good performance compared to ASPARTIX (even with its `metasp` approach) and ArgTools [82]. ArgSemSAT allows to choose between different SAT-solvers and processes input in the ASPARTIX input format. In future it is also supposed to include direct algorithms based on the SCC-recursive schema [22, 129].

ArgTools This system aims to provide a fast implementation of a labeling-based algorithm for enumerating all preferred extensions (cf. Algorithm 4). While the main focus of the research behind this tool is directed towards efficient enumeration of preferred semantics [36] there are several other results. First, enumeration algorithms for several other semantics, i.e. those depicted in Table 4 and ideal, where developed in [105]. Second, in [36] the authors present an implementation of optimizations for credulous and skeptical reasoning with preferred semantics. This line of research compares the performance of different labeling based algorithms and in particular gives empirical evidence, by comparing the algorithms on

randomly generated instances, that the newly proposed algorithms are the fastest ones. Some of the algorithms are also compared with ASPARTIX (using the DLV solver) and dynPARTIX where ArgTools again showed good performance.

ASPARTIX The "Answer Set Programming Argumentation Reasoning Tool" [30] is based on reductions to ASP as discussed in Section 3.3. It consists of a collection of ASP encodings, where each encoding, augmented by a given AF in form of ASP facts, can be given as input to an ASP solver in order to compute the extensions. For most semantics, ASPARTIX provides encodings for the solver DLV as well as gringo/clasp(D). Following the reduction approach, ASPARTIX's performance scales with new versions of these solvers. Furthermore, the system is platform independent in the sense that it runs on any system supporting the ASP solvers. ASPARTIX also offers a web front-end, where any argumentation framework as well as its extensions can be inspected graphically. A particularly useful feature of this system is that it supports many semantics and solves various reasoning problems. In addition to the semantics in Table 4, ASPARTIX supports ideal, cf2, stage2 and resolution-based grounded semantics. Since ASP solvers support enumeration and credulous as well as skeptical query-based reasoning, this can directly be utilized by ASPARTIX. The system is often used as a reference system in performance comparisons [21,82,31,36,37,83,98,105,130,45,44,131].

CEGARTIX The "Counter-Example Guided Argumentation Reasoning Tool" [21] is built on top of modern SAT solvers, and relies on an iterative procedure of SAT-calls (see Algorithm 1). As a command-line tool, CEGARTIX is built towards performance and computes the skeptical acceptance of an argument w.r.t. preferred, semi-stable and stage semantics and for the last two also credulous acceptance. Like ArgSemSAT [82], which relies on iterative SAT calls for enumerating preferred labelings, CEGARTIX can be seen as a sort of hybrid approach between direct and reduction-based methods, since only certain sub-tasks are delegated to a SAT solver. CEGARTIX is available online as a binary. The system allows the user to configure which SAT-solver she wants to use. Being based on a reduction approach, CEGARTIX scales with newer versions of SAT solvers and the system has been shown to be competitive w.r.t. ASPARTIX [21] and also processes the ASPARTIX input format.

CompArg CompArg [38] is intended for determining credulous acceptance of arguments w.r.t. preferred semantics and enumerating grounded, preferred, stable and semi-stable extensions. It implements the labeling-based approach as presented in Section 4.1 (Algorithm 4). Written in Delphi, the executable for Windows is publicly available. The system comes with many examples, which suits its main educational aim of illustration of the semantics. Due to this purpose it is not primarily built for high performance. The tool consists of a GUI that illustrates the computation of the acceptance status, either by providing proofs or refutations of arguments. Additionally, several example instances are provided. Therefore, CompArg is particularly useful when it comes to get a deeper understanding of the underlying algorithm. Besides deciding credulous acceptance, the resulting extensions can be enumerated.

ConArg The system ConArg [26] follows a reduction-based approach towards CSPs as presented in Section 3.2. Internally, the tool uses sophisticated Java implementations of CSP engines (JaCoP). Its performance thus scales with newer versions of these engines, and it is platform-independent through the use of Java. It supports the enumeration of extensions for many semantics (see Table 4) and is capable of verifying whether a given set is a preferred extension. The tool features a simple and intuitive graphical user interface for inspecting the AF and the extensions at hand. It supports the ASPARTIX input format for AFs, and also allows to generate random (weighted) argumentation frameworks. ConArg is also available as a web-interface with an interactive graphical representation. Recently, the second version of ConArg with some modifications was released [131,43,44]. To improve the performance, ConArg2 is now based on Gecode, an efficient C++ environment for constraint-based applications. ConArg2 is available as a pre-compiled command line tool for Linux. Besides the features of ConArg, ConArg2 also allows for credulous and skeptical reasoning for admissible, stable and complete semantics. ConArg and ConArg2 showed good performance compared to ASPARTIX and Dung-O-Matic [130,131].

Dung-O-Matic Dung-O-Matic is based on dialectical proof-procedures (see Section 4.2) and includes implementations for many different semantics. Besides support for most of the semantics listed in Table 4, ideal and eager extensions for a given AF can be computed. Implemented as a Java library, it can be flexibly used across platforms. For demonstration purposes, it is accessible via the tool OVAgen.[15] OVAgen is a web-based software where argumentation frameworks can be drawn graphically and the resulting extensions are visualized. A preliminary performance comparison against ASPARTIX and ConArg is published by Bistarelli et al. [130]. Although this work shows that the tool is outperformed by the other two systems when computing complete or stable extensions, one has to note that further comparisons and, in particular, real-world instances are necessary to gain a better picture of the tool's performance.

Dungine (part of ArgKit) Dungine [109] implements algorithms based on dialogue games, and currently provides native support for grounded and preferred semantics. It is part of ArgKit, a Java library intended for building custom software based on argumentation. For demonstration purposes, the ArgKit package includes examples of GUI applications. Additionally,

[15] http://ova.computing.dundee.ac.uk/ova-gen/.

similar to Dung-O-Matic, the software is integrated in the tool OVAgen. Since the source code is made publicly available under the LGPL license, it can be integrated in other projects.

dynPARTIX The concept underlying the "Dynamic Programming Argumentation Reasoning Tool" [42,115] is based on dynamic programming, where the instance is decomposed before solving (see Section 4.3). The tool exploits the structure of the given argumentation framework, where the decomposition is constructed based on heuristics. Hence, its run-time performance is particularly good for instances with tree-like structures. The tool, implemented in C++, is currently available as Linux executable. A special characteristic that differentiates it from the other systems presented here is its ability to provide the overall number of solutions without explicit enumeration.

PyAAL (+ArguLab) The "Python Abstract Argumentation Library" implements labeling-based procedures for determining the justification status of arguments and for enumerating the labelings for many semantics (see Section 4.1). In addition to the functionality summarized in Table 4, PyAAL is able to compute the ideal and eager labeling, as well as determining the corresponding justification status. ArguLab [107] is a web front-end that allows to demonstrate the capabilities of PyAAL. Within ArguLab, in a first step the argumentation framework is constructed. Next, based on the selected semantics the labelings associated with the arguments are visualized. The tool allows to interactively analyze the justification status of arguments. Note that ArguLab is designed for demonstration purposes only, but the underlying code of PyAAL can be used without restrictions (GPL licensed). Its particular strength lies in the fact that it supports a broad number of semantics and solves several reasoning problems.

5.2. Summary

The system comparison illustrates the diverse landscape of available tools for abstract argumentation: While some systems cover a wide range of different semantics (e.g., ASPARTIX, ConArg, Dung-O-Matic, and PyAAL), others are well-suited for illustration and demonstration purposes of the algorithms (e.g., CompArg and Dungine) or are tailored towards solving particular problems efficiently (e.g., ArgSemSAT, ArgTools, CEGARTIX and dynPARTIX). Also, diversity is observable when considering the supported semantics and solvable reasoning problems (see Table 4). Among the considered systems, no semantics and reasoning problem is supported by all tools. Additionally, to promote their functionality, several tools provide access to their systems via a web interface (e.g., ASPARTIX, ConArg, Dung-O-Matic, Dungine and PyAAL), which allows to test the system without the necessity to download or install software.

The available run-time comparisons do not indicate that one system outperforms all others. However, we observed that ASPARTIX is, in most cases, used as a base line system for performance comparisons. To give a clearer picture on the performance aspects of the tools, there is a need for independently created and publicly available benchmark suites. This (and also ideas on running even a public system competition for argumentation systems) is discussed within the community (see, e.g., [132]).

Additionally, besides run-time performance, many other aspects are important for a good system, including intuitive design, versatility, extendability, and also source code availability or ongoing support and development of the system. Each tool has its unique characteristics and advantages, therefore the choice for the right tool mainly depends on the problem at hand.

6. 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 6.1) and complementary aspects for evaluating abstract argumentation frameworks, for instance, pre-processing (Section 6.2). In Section 6.3, 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.

6.1. 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. [126] argumentation semantics can be defined on the basis of decomposing an AF into 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 [126] and stage2 [127] semantics. For both semantics, ASP encodings [127,133] as well as labeling-based algorithms [127] have been presented, the former are integrated in the ASPARTIX system.

Moreover, there is the family of resolution-based semantics [128], with the resolution-based grounded semantics being the most popular instance. Different ASP encodings for resolution-based grounded semantics are studied in [31] and are incorporated in the ASPARTIX system, as well.

Finally, the unique-status semantics ideal [124] and eager [125] (for a general notion of parametric ideal semantics, see [134]) 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 [135] 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 [30] and is implemented in the ASPARTIX system. Regarding other reduction-based systems, ConArg is also capable of computing the ideal extension of an AF.

6.2. 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 [136,137]. 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 [138]. 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 [139].

Second, the identification of redundant patterns might be used to simplify argumentation frameworks before evaluation. The notion of strong equivalence [73,140] 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., [141,142].

Finally, we mention the concept of intertranslatability between abstract argumentation semantics [72]. 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.

6.3. 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 [143], resp. ASPIC$^+$ [50], which instantiates Dung-style frameworks. Arguments are represented as inference trees by applying strict and defeasible inference rules. TOAST (The Online Argument Structures Tool) [144] is an implementation of ASPIC$^+$ and is available as web front-end.[16] 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 ASPIC argumentation engine demo[17] implements several instantiations of ASPIC and provides a web interface. Again the user can specify a knowledge base and a rule set to construct an argumentation system which then can be evaluated based on grounded and credulous preferred semantics. The Carneades Web Service[18] is capable of "argument construction, storage, navigation, querying, evaluation, visualization and interchange" [145]. It is based on the ASPIC$^+$ model of structured argument but still preserves the features of the original version of Carneades system [51]. On the resulting Dung-style framework it applies grounded semantics. An approach based on classical logic and argument instantiation is shown in [146]. Here, arguments and possible counterarguments are constructed from a classical propositional knowledge base. Furthermore, Vispartix[19] consists of a collection of ASP encodings [147] for obtaining Dung argumentation frameworks from a propositional knowledge base (and a set of predefined claims), based on the approach presented in [148]. The argumentation framework can then, for example, be evaluated by ASPARTIX. Another survey [54] summarizes systems that focus on the construction of arguments. This includes approaches based on classical [148] and defeasible logic [53] and briefly introduces the systems ASPIC and CaSAPI[20] (which combines abstract and assumption-based argumentation).

[16] http://www.arg.dundee.ac.uk/toast/.
[17] http://aspic.cossac.org/ArgumentationSystem/.
[18] http://carneades.github.com/.
[19] http://www.dbai.tuwien.ac.at/proj/argumentation/vispartix/.
[20] http://www.doc.ic.ac.uk/~ft/CaSAPI/.

There is also recent work on translating different argumentation models. In [149], a translation between Carneades and Dung AFs is studied and implemented in the functional programming language Haskell. The benefit of such systems is that one may re-use engines for AFs to compute results for Carneades and potentially other argumentation models. Regarding different argumentation models, a reduction-based approach was implemented for recursive, probabilistic defeasible logic programming (RP-DeLP). The resulting system uses ASP for computing the results [150].

Finally, a recent review on systems for argumentation in the Social Semantic Web [151] summarizes social web tools, and discusses how argumentation can be modeled in this context. It contains an exhaustive study of online tools that combine Web 2.0 and Semantic Web technologies. In the course of the review, it gives an comparative overview on current developments of practically applied argumentation research on the web.

6.4. 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.), thereby 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. The strengths of reduction-based systems are that they (i) directly scale with newer versions of solvers and (ii) can be easily adapted to specific needs, which is mirrored by a quite flexible support of reasoning problems and argumentation semantics. Ultimately, dedicated direct algorithms (when tuned sufficiently) outperform reduction-based ones (see e.g. [36]), but each reasoning problem and semantics needs its own full-fledged implementation. In conclusion, this suggests that the reduction-based approach provides a good basis for rapid prototyping, while systems that have to scale well for large problems require substantial design efforts.

However, the two categories are not as strictly separated as it might look like. For instance, the CEGARTIX approach as introduced in Section 3.1.3 is an example for a system that combines the advantages of the two categories. It is based on a dedicated algorithm for the argumentation problem at hand, but as a subroutine invokes existing systems (i.e., SAT solvers). Systems that combine these approaches are among the fastest systems, in particular significantly faster than pure reduction-based approaches and still directly benefit from improvements in the solvers. However, by this hybrid approach one loses the flexibility of pure reduction-based systems.

The current landscape of systems is very heterogeneous (see Section 5). Available systems for argumentation differ in their support for semantics and reasoning problems, summarized in Table 4. Each tool has its unique strengths and characteristics, be it graphical representation of algorithms or results, efficient solving in some setting, or providing broad support for problems to be solved in the context of argumentation.

6.5. 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. Likewise, the area of average-case and probabilistic algorithms has not been considered in combination with abstract argumentation yet. 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. To date, experiments are typically performed on some randomly generated frameworks. However, a better theoretical model for such frameworks is required in order to have a more meaningful picture when runtimes are measured. On the other hand, collections of structured instances stemming from real-world applications domains are lacking. Several ideas for establishing a benchmark library for abstract argumentation have first been collected in [152] and the upcoming first competition for argumentation systems, http://argumentationcompetition.org, will be very beneficial for the evolution of systems.

Moreover, 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 [153–155] 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 [156]. 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

This work has been supported by the Vienna Science and Technology Fund (WWTF) under grant ICT08-028, by the Austrian Science Fund (FWF): I1102, Y698 and P20704, and by the Vienna University of Technology special fund "Innovative Projekte" (9006.09/008).

The authors are grateful to the anonymous reviewers for their detailed reviews and helpful comments as well as to all colleagues who discussed their works and systems with the authors.

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, Paul Wandl and Christian Weichselbaum.

References

[1] P.M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, Artif. Intell. 77 (2) (1995) 321–358.
[2] T.J.M. Bench-Capon, P.E. Dunne, Argumentation in artificial intelligence, Artif. Intell. 171 (10–15) (2007) 619–641.
[3] I. Rahwan, G.R. Simari (Eds.), Argumentation in Artificial Intelligence, Springer, 2009.
[4] P. Besnard, A. Hunter, Elements of Argumentation, MIT Press, 2008.
[5] M. Caminada, L. Amgoud, On the evaluation of argumentation formalisms, Artif. Intell. 171 (5–6) (2007) 286–310.
[6] N. Gorogiannis, A. Hunter, Instantiating abstract argumentation with classical logic arguments: postulates and properties, Artif. Intell. 175 (9–10) (2011) 1479–1497.
[7] G. Brewka, Nonmonotonic tools for argumentation, in: T. Janhunen, I. Niemelä (Eds.), Proceedings of the 12th European Conference on Logics in Artificial Intelligence, JELIA 2010, in: Lecture Notes in Computer Science, vol. 6341, Springer, 2010, pp. 1–6.
[8] P. Baroni, M. Giacomin, On principle-based evaluation of extension-based argumentation semantics, Artif. Intell. 171 (10–15) (2007) 675–700.
[9] P. Baroni, M. Giacomin, A systematic classification of argumentation frameworks where semantics agree, in: P. Besnard, S. Doutre, A. Hunter (Eds.), Proceedings of the 2nd Conference on Computational Models of Argument, COMMA 2008, in: Frontiers in Artificial Intelligence and Applications, vol. 172, IOS Press, 2008, pp. 37–48.
[10] P. Baroni, M. Giacomin, Semantics of abstract argument systems, in: Rahwan and Simari [3], pp. 25–44.
[11] D.C. Martínez, A.J. García, G.R. Simari, On acceptability in abstract argumentation frameworks with an extended defeat relation, in: P.E. Dunne, T.J.M. Bench-Capon (Eds.), Proceedings of the 1st Conference on Computational Models of Argument, COMMA 2006, in: Frontiers in Artificial Intelligence and Applications, vol. 144, IOS Press, 2006, pp. 273–278.
[12] Y. Dimopoulos, A. Torres, Graph theoretical structures in logic programs and default theories, Theor. Comput. Sci. 170 (1–2) (1996) 209–244.
[13] P.E. Dunne, T.J.M. Bench-Capon, Coherence in finite argument systems, Artif. Intell. 141 (1/2) (2002) 187–203.
[14] W. Dvořák, S. Woltran, Complexity of semi-stable and stage semantics in argumentation frameworks, Inf. Process. Lett. 110 (11) (2010) 425–430.
[15] R. Dechter, Constraint Processing, Morgan Kaufmann, 2003.
[16] A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, IOS Press, 2009.
[17] G. Brewka, T. Eiter, M. Truszczyński, Answer set programming at a glance, Commun. ACM 54 (12) (2011) 92–103.
[18] P. Besnard, S. Doutre, Checking the acceptability of a set of arguments, in: J.P. Delgrande, T. Schaub (Eds.), Proceedings of the 10th International Workshop on Non-monotonic Reasoning, NMR 2004, 2004, pp. 59–64.
[19] O. Arieli, M. Caminada, A QBF-based formalization of abstract argumentation semantics, J. Appl. Log. 11 (2) (2013) 229–252.
[20] U. Egly, S. Woltran, Reasoning in argumentation frameworks using quantified boolean formulas, in: P.E. Dunne, T.J.M. Bench-Capon (Eds.), Proceedings of the 1st Conference on Computational Models of Argument, COMMA 2006, in: Frontiers in Artificial Intelligence and Applications, vol. 144, IOS Press, 2006, pp. 133–144.
[21] W. Dvořák, M. Järvisalo, J.P. Wallner, S. Woltran, Complexity-sensitive decision procedures for abstract argumentation, Artif. Intell. 206 (2014) 53–78.
[22] F. Cerutti, M. Giacomin, M. Vallati, ArgSemSAT: solving argumentation problems using SAT, in: S. Parsons, N. Oren, C. Reed, F. Cerutti (Eds.), Proceedings of the 5th International Conference on Computational Models of Argument, COMMA 2014, in: Frontiers in Artificial Intelligence and Applications, vol. 266, IOS Press, 2014, pp. 455–456.
[23] L. Amgoud, C. Devred, Argumentation frameworks as constraint satisfaction problems, in: S. Benferhat, J. Grant (Eds.), Proceedings of the 5th International Conference on Scalable Uncertainty Management, SUM 2011, in: Lecture Notes in Computer Science, vol. 6929, Springer, 2011, pp. 110–122.
[24] S. Bistarelli, D. Pirolandi, F. Santini, Solving weighted argumentation frameworks with soft constraints, in: J. Larrosa, B. O'Sullivan (Eds.), Proceedings of the 14th Annual ERCIM International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2009, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 6384, Springer, 2009, pp. 1–18.
[25] S. Bistarelli, F. Santini, A common computational framework for semiring-based argumentation systems, in: H. Coelho, R. Studer, M. Wooldridge (Eds.), Proceedings of the 19th European Conference on Artificial Intelligence, ECAI 2010, in: Frontiers in Artificial Intelligence and Applications, vol. 215, IOS Press, 2010, pp. 131–136.
[26] S. Bistarelli, F. Santini, ConArg: a constraint-based computational framework for argumentation systems, in: T.M. Khoshgoftaar, X.H. Zhu (Eds.), Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2011, IEEE Computer Society Press, 2011, pp. 605–612.
[27] S. Bistarelli, F. Santini, Modeling and solving AFs with a constraint-based tool: ConArg, in: S. Modgil, N. Oren, F. Toni (Eds.), Proceedings of the 1st International Workshop on Theory and Applications of Formal Argumentation, TAFA 2011, in: Lecture Notes in Computer Science, vol. 7132, Springer, 2012, pp. 99–116.
[28] S. Bistarelli, F. Santini, ConArg: a tool to solve (weighted) abstract argumentation frameworks with (soft) constraints, CoRR, arXiv:1212.2857.

[29] F. Toni, M. Sergot, Argumentation and answer set programming, in: M. Balduccini, T.C. Son (Eds.), Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays in Honor of Michael Gelfond, in: Lecture Notes in Computer Science, vol. 6565, Springer, 2011, pp. 164–180.
[30] U. Egly, S.A. Gaggl, S. Woltran, Answer-set programming encodings for argumentation frameworks, Argument Comput. 1 (2) (2010) 147–177.
[31] W. Dvořák, S.A. Gaggl, J.P. Wallner, S. Woltran, Making use of advances in answer-set programming for abstract argumentation systems, in: H. Tompits, S. Abreu, J. Oetsch, J. Pührer, D. Seipel, M. Umeda, A. Wolf (Eds.), Proceedings of the 19th International Conference on Applications of Declarative Programming and Knowledge Management, INAP 2011, Revised Selected Papers, in: Lecture Notes in Artificial Intelligence, vol. 7773, Springer, 2013, pp. 114–133.
[32] D.M. Gabbay, An equational approach to argumentation networks, Argument Comput. 3 (2–3) (2012) 87–142.
[33] W. Dvořák, S. Szeider, S. Woltran, Abstract argumentation via monadic second order logic, in: E. Hüllermeier, S. Link, T. Fober, B. Seeger (Eds.), Proceedings of the 6th International Conference on Scalable Uncertainty Management, SUM 2012, in: Lecture Notes in Computer Science, vol. 7520, Springer, 2012, pp. 85–98.
[34] S. Doutre, J. Mengin, Preferred extensions of argumentation frameworks: query answering and computation, in: R. Goré, A. Leitsch, T. Nipkow (Eds.), Proceedings of the 1st International Joint Conference on Automated Reasoning, IJCAR 2001, in: Lecture Notes in Computer Science, vol. 2083, Springer, 2001, pp. 272–288.
[35] S. Modgil, M. Caminada, Proof theories and algorithms for abstract argumentation frameworks, in: Rahwan and Simari [3], pp. 105–132.
[36] S. Nofal, K. Atkinson, P.E. Dunne, Algorithms for decision problems in argument systems under preferred semantics, Artif. Intell. 207 (2014) 23–51.
[37] S. Nofal, K. Atkinson, P.E. Dunne, Algorithms for argumentation semantics: labeling attacks as a generalization of labeling arguments, J. Artif. Intell. Res. 49 (2014) 635–668.
[38] B. Verheij, A labeling approach to the computation of credulous acceptance in argumentation, in: M.M. Veloso (Ed.), Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, 2007, pp. 623–628.
[39] P.M. Thang, P.M. Dung, N.D. Hung, Towards a common framework for dialectical proof procedures in abstract argumentation, J. Log. Comput. 19 (6) (2009) 1071–1109.
[40] W. Dvořák, R. Pichler, S. Woltran, Towards fixed-parameter tractable algorithms for abstract argumentation, Artif. Intell. 186 (2012) 1–37.
[41] P.E. Dunne, Computational properties of argument systems satisfying graph-theoretic constraints, Artif. Intell. 171 (10–15) (2007) 701–729.
[42] G. Charwat, Tree-decomposition based algorithms for abstract argumentation frameworks, Master's thesis, Vienna University of Technology, 2012, http://permalink.obvsg.at/AC07812654.
[43] S. Bistarelli, F. Rossi, F. Santini, Benchmarking hard problems in random abstract AFs: the stable semantics, in: S. Parsons, N. Oren, C. Reed, F. Cerutti (Eds.), Proceedings of the 5th International Conference on Computational Models of Argument, COMMA 2014, in: Frontiers in Artificial Intelligence and Applications, vol. 266, IOS Press, 2014, pp. 153–160.
[44] S. Bistarelli, F. Rossi, F. Santini, A first comparison of abstract argumentation reasoning-tools, in: T. Schaub, G. Friedrich, B. O'Sullivan (Eds.), Proceedings of the 21st European Conference on Artificial Intelligence, ECAI 2014, in: Frontiers in Artificial Intelligence and Applications, vol. 263, IOS Press, 2014, pp. 969–970.
[45] F. Cerutti, M. Giacomin, M. Vallati, Algorithm selection for preferred extensions enumeration, in: S. Parsons, N. Oren, C. Reed, F. Cerutti (Eds.), Proceedings of the 5th International Conference on Computational Models of Argument, COMMA 2014, in: Frontiers in Artificial Intelligence and Applications, vol. 266, IOS Press, 2014, pp. 221–232.
[46] M. Järvisalo, D.L. Berre, O. Roussel, L. Simon, The international SAT solver competitions, AI Mag. 33 (1) (2012) 89–94.
[47] M. Alviano, F. Calimeri, G. Charwat, M. Dao-Tran, C. Dodaro, G. Ianni, T. Krennwallner, M. Kronegger, J. Oetsch, A. Pfandler, J. Pührer, C. Redl, F. Ricca, P. Schneider, M. Schwengerer, L.K. Spendier, J.P. Wallner, G. Xiao, The fourth answer set programming competition: preliminary report, in: P. Cabalar, T.C. Son (Eds.), Proceedings of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2013, in: Lecture Notes in Artificial Intelligence, vol. 8148, Springer, 2013, pp. 42–53.
[48] F. Cerutti, N. Oren, H. Strass, M. Thimm, M. Vallati, A benchmark framework for a computational argumentation competition, in: S. Parsons, N. Oren, C. Reed, F. Cerutti (Eds.), Proceedings of the 5th International Conference on Computational Models of Argument, COMMA 20, in: Frontiers in Artificial Intelligence and Applications, vol. 266, IOS Press, 2014, pp. 459–460.
[49] P. Baroni, M. Caminada, M. Giacomin, An introduction to argumentation semantics, Knowl. Eng. Rev. 26 (4) (2011) 365–410.
[50] H. Prakken, An abstract framework for argumentation with structured arguments, Argument Comput. 1 (2) (2010) 93–124.
[51] T.F. Gordon, H. Prakken, D. Walton, The Carneades model of argument and burden of proof, Artif. Intell. 171 (10–15) (2007) 875–896.
[52] P.M. Dung, R.A. Kowalski, F. Toni, Assumption-based argumentation, in: Rahwan and Simari [3], pp. 25–44.
[53] A.J. García, G.R. Simari, Defeasible logic programming: an argumentative approach, Theory Pract. Log. Program. 4 (1–2) (2004) 95–138.
[54] G.R. Simari, A brief overview of research in argumentation systems, in: S. Benferhat, J. Grant (Eds.), Proceedings of the 5th International Conference on Scalable Uncertainty Management, SUM 2011, in: Lecture Notes in Computer Science, vol. 6929, Springer, 2011, pp. 81–95.
[55] G. Brewka, S. Polberg, S. Woltran, Generalizations of Dung frameworks and their role in formal argumentation, IEEE Intell. Syst. 29 (1) (2014) 30–38.
[56] T.J.M. Bench-Capon, Persuasion in practical argument using value-based argumentation frameworks, J. Log. Comput. 13 (3) (2003) 429–448.
[57] C. Cayrol, M.-C. Lagasquie-Schiex, Bipolar abstract argumentation systems, in: Rahwan and Simari [3], pp. 65–84.
[58] S. Modgil, Reasoning about preferences in argumentation frameworks, Artif. Intell. 173 (9–10) (2009) 901–934.
[59] L. Amgoud, C. Devred, M.-C. Lagasquie-Schiex, A constrained argumentation system for practical reasoning, in: I. Rahwan, P. Moraitis (Eds.), Proceedings of the 5th International Workshop on Argumentation in Multi-Agent Systems, ArgMAS 2008, Revised Selected and Invited Papers, in: Lecture Notes in Computer Science, vol. 5384, Springer, 2009, pp. 37–56.
[60] M.C. Budán, M.J. Lucero, C.I. Chesñevar, G.R. Simari, Modelling time and reliability in structured argumentation frameworks, in: G. Brewka, T. Eiter, S.A. McIlraith (Eds.), Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning, KR 2012, AAAI Press, 2012, pp. 578–582.
[61] H. Kido, K. Nitta, Practical argumentation semantics for socially efficient defeasible consequence, in: L. Sonenberg, P. Stone, K. Tumer, P. Yolum (Eds.), Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2011, IFAAMAS, 2011, pp. 267–274.
[62] D.M. Gabbay, Fibring argumentation frames, Stud. Log. 93 (2–3) (2009) 231–295.
[63] P. Baroni, F. Cerutti, M. Giacomin, G. Guida, AFRA: argumentation framework with recursive attacks, Int. J. Approx. Reason. 52 (1) (2011) 19–37.
[64] G. Brewka, T. Eiter, Argumentation context systems: a framework for abstract group argumentation, in: E. Erdem, F. Lin, T. Schaub (Eds.), Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2009, in: Lecture Notes in Computer Science, vol. 5753, Springer, 2009, pp. 44–57.
[65] G. Brewka, S. Woltran, Abstract dialectical frameworks, in: F. Lin, U. Sattler, M. Truszczyński (Eds.), Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning, KR 2010, AAAI Press, 2010, pp. 780–785.
[66] A. Hunter, Some foundations for probabilistic abstract argumentation, in: B. Verheij, S. Szeider, S. Woltran (Eds.), Proceedings of the 4th Conference on Computational Models of Argument, COMMA 2012, in: Frontiers in Artificial Intelligence and Applications, vol. 245, IOS Press, 2012, pp. 117–128.
[67] P.E. Dunne, A. Hunter, P. McBurney, S. Parsons, M. Wooldridge, Weighted argument systems: basic definitions, algorithms, and complexity results, Artif. Intell. 175 (2) (2011) 457–486.

[68] S. Coste-Marquis, C. Devred, P. Marquis, Symmetric argumentation frameworks, in: L. Godo (Ed.), Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2005, in: Lecture Notes in Computer Science, vol. 3571, Springer, 2005, pp. 317–328.
[69] M. Caminada, D.M. Gabbay, A logical account of formal argumentation, Stud. Log. 93 (2) (2009) 109–145.
[70] P.E. Dunne, M. Wooldridge, Complexity of abstract argumentation, in: Simari and Rahwan [3], pp. 85–104.
[71] P.E. Dunne, M. Caminada, Computational complexity of semi-stable semantics in abstract argumentation frameworks, in: S. Hölldobler, C. Lutz, H. Wansing (Eds.), Proceedings of the 11th European Conference on Logics in Artificial Intelligence, JELIA 2008, in: Lecture Notes in Computer Science, vol. 5293, Springer, 2008, pp. 153–165.
[72] W. Dvořák, S. Woltran, On the intertranslatability of argumentation semantics, J. Artif. Intell. Res. 41 (2011) 445–475.
[73] S.A. Gaggl, S. Woltran, The cf2 argumentation semantics revisited, J. Log. Comput. 23 (5) (2013) 925–949.
[74] J.C. Nieves, M. Osorio, U. Cortés, Preferred extensions as stable models, Theory Pract. Log. Program. 8 (4) (2008) 527–543.
[75] M. Osorio, C. Zepeda, J.C. Nieves, U. Cortés, Inferring acceptable arguments with answer set programming, in: Proceedings of the 6th Mexican International Conference on Computer Science, ENC 2005, IEEE Computer Society, 2005, pp. 198–205.
[76] T. Wakaki, K. Nitta, Computing argumentation semantics in answer set programming, in: New Frontiers in Artificial Intelligence, JSAI 2008 Conference and Workshops, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 5447, 2008, pp. 254–269.
[77] D.M. Gabbay, The equational approach to cf2 semantics, in: B. Verheij, S. Szeider, S. Woltran (Eds.), Proceedings of the 4th Conference on Computational Models of Argument, COMMA 2012, in: Frontiers in Artificial Intelligence and Applications, vol. 245, IOS Press, 2012, pp. 141–152.
[78] N. Eén, N. Sörensson, An extensible SAT-solver, in: E. Giunchiglia, A. Tacchella (Eds.), Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing, SAT 2003, in: Lecture Notes in Computer Science, vol. 2919, Springer, 2004, pp. 502–518.
[79] J.P. Marques-Silva, K.A. Sakallah, GRASP: a search algorithm for propositional satisfiability, IEEE Trans. Comput. 48 (5) (1999) 506–521.
[80] D.M. Gabbay, Dung's argumentation is essentially equivalent to classical propositional logic with the Peirce–Quine dagger, Log. Univers. 5 (2011) 255–318.
[81] G. Brewka, P.E. Dunne, S. Woltran, Relating the semantics of abstract dialectical frameworks and standard AFs, in: T. Walsh (Ed.), Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011, IJCAI/AAAI, 2011, pp. 780–785.
[82] F. Cerutti, P.E. Dunne, M. Giacomin, M. Vallati, Computing preferred extensions in abstract argumentation: a SAT-based approach, in: E. Black, S. Modgil, N. Oren (Eds.), Proceedings of the Second International Workshop on Theory and Applications of Formal Argumentation, Revised Selected Papers, TAFA 2013, in: Lecture Notes in Computer Science, vol. 8306, Springer, 2014, pp. 176–193.
[83] W. Dvořák, M. Järvisalo, J.P. Wallner, S. Woltran, Complexity-sensitive decision procedures for abstract argumentation, in: G. Brewka, T. Eiter, S.A. McIlraith (Eds.), Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning, KR 2012, AAAI Press, 2012, pp. 54–64.
[84] F. Rossi, P.v. Beek, T. Walsh, Handbook of Constraint Programming (Foundations of Artificial Intelligence), Elsevier Science Inc., 2006.
[85] T. Walsh, SAT v CSP, in: R. Dechter (Ed.), Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming, CP 2000, in: Lecture Notes in Computer Science, vol. 1894, Springer, 2000, pp. 441–456.
[86] V.W. Marek, M. Truszczyński, Stable models and an alternative logic programming paradigm, in: The Logic Programming Paradigm – A 25-Year Perspective, Springer, 1999, pp. 375–398.
[87] I. Niemelä, Logic programming with stable model semantics as a constraint programming paradigm, Ann. Math. Artif. Intell. 25 (3–4) (1999) 241–273.
[88] C. Baral, Knowledge Representation, Reasoning and Declarative Problem Solving, Cambridge University Press, 2002.
[89] M. Gelfond, Representing knowledge in A-Prolog, in: Computational Logic: From Logic Programming into the Future, in: Lecture Notes in Computer Science, vol. 2408, Springer, 2002, pp. 413–451.
[90] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, M.T. Schneider, Potassco: the Potsdam answer set solving collection, AI Commun. 24 (2) (2011) 107–124.
[91] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello, The DLV system for knowledge representation and reasoning, ACM Trans. Comput. Log. 7 (3) (2006) 499–562.
[92] T. Eiter, G. Gottlob, On the computational cost of disjunctive logic programming: propositional case, Ann. Math. Artif. Intell. 15 (3–4) (1995) 289–323.
[93] M. Gebser, R. Kaminski, T. Schaub, Complex optimization in answer set programming, Theory Pract. Log. Program. 11 (4–5) (2011) 821–839.
[94] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (3/4) (1991) 365–386.
[95] T. Eiter, G. Gottlob, H. Mannila, Disjunctive datalog, ACM Trans. Database Syst. 22 (3) (1997) 364–418.
[96] M.H. Liffiton, K.A. Sakallah, Algorithms for computing minimal unsatisfiable subsets of constraints, J. Autom. Reason. 40 (1) (2008) 1–33.
[97] P. Kilby, J.K. Slaney, S. Thiébaux, T. Walsh, Backbones and backdoors in satisfiability, in: M.M. Veloso, S. Kambhampati (Eds.), Proceedings of the 20th National Conference on Artificial Intelligence, AAAI 2005, AAAI Press/The MIT Press, 2005, pp. 1368–1373.
[98] J.P. Wallner, G. Weissenbacher, S. Woltran, Advanced SAT techniques for abstract argumentation, in: J. Leite, T.C. Son, P. Torroni, L. van der Torre, S. Woltran (Eds.), Proceedings of the 14th International Workshop on Computational Logic in Multi-Agent Systems, CLIMA 2013, in: Lecture Notes in Artificial Intelligence, vol. 8143, Springer, 2013, pp. 138–154.
[99] M. Osorio, J. Dıaz, A. Santoyo, Computing semi-stable semantics of af by 0-1 integer programming, in: Proceedings of the 9th Latin American Workshop on Logic/Languages, Algorithms and New Methods of Reasoning, LANMR 2014, 2014.
[100] M. Osorio, J. Díaz, A. Santoyo, Computing preferred semantics: comparing two ASP approaches vs an approach based on 0-1 integer programming, in: A.F. Gelbukh, F.C. Espinoza, S.N. Galicia-Haro (Eds.), Human-Inspired Computing and Its Applications – 13th Mexican International Conference on Artificial Intelligence, MICAI 2014, in: Lecture Notes in Computer Science, vol. 8856, Springer, 2014, pp. 419–430.
[101] P.E. Dunne, The computational complexity of ideal semantics, Artif. Intell. 173 (18) (2009) 1559–1591.
[102] B. Bliem, Decompose, guess & check – declarative problem solving on tree decompositions, Master's thesis, Vienna University of Technology, 2012, http://permalink.obvsg.at/AC07814499.
[103] A. Langer, F. Reidl, P. Rossmanith, S. Sikdar, Evaluation of an MSO-solver, in: D.A. Bader, P. Mutzel (Eds.), Proceedings of the 2012 Meeting on Algorithm Engineering & Experiments, ALENEX 2012, SIAM/Omnipress, 2012, pp. 55–63.
[104] M. Caminada, An algorithm for computing semi-stable semantics, in: K. Mellouli (Ed.), Proceedings of the 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2007, in: Lecture Notes in Computer Science, vol. 4724, Springer, 2007, pp. 222–234.
[105] S. Nofal, Algorithms for argument systems, Ph.D. thesis, University of Liverpool, 2013, http://research-archive.liv.ac.uk/12173/.
[106] M. Caminada, An algorithm for stage semantics, in: P. Baroni, F. Cerutti, M. Giacomin, G.R. Simari (Eds.), Proceedings of the 3rd International Conference on Computational Models of Argument, COMMA 2010, in: Frontiers in Artificial Intelligence and Applications, vol. 216, IOS Press, 2010, pp. 147–158.
[107] M. Podlaszewski, M. Caminada, G. Pigozzi, An implementation of basic argumentation components, in: L. Sonenberg, P. Stone, K. Tumer, P. Yolum (Eds.), Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2011, IFAAMAS, 2011, pp. 1307–1308.
[108] M. Caminada, W. Dvořák, S. Vesic, Preferred semantics as Socratic discussion, J. Log. Comput. (2014), http://dx.doi.org/10.1093/logcom/exu005, in press.
[109] M. South, G. Vreeswijk, J. Fox, Dungine: a Java Dung reasoner, in: P. Besnard, S. Doutre, A. Hunter (Eds.), Proceedings of the 2nd Conference on Computational Models of Argument, COMMA 2008, in: Frontiers in Artificial Intelligence and Applications, vol. 172, IOS Press, 2008, pp. 360–368.

[110] G. Vreeswijk, An algorithm to compute minimally grounded and admissible defence sets in argument systems, in: P.E. Dunne, T.J.M. Bench-Capon (Eds.), Proceedings of the 1st Conference on Computational Models of Argument, COMMA 2006, in: Frontiers in Artificial Intelligence and Applications, vol. 144, 2006, pp. 109–120.
[111] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, Oxford University Press, 2006.
[112] W. Dvořák, R. Pichler, S. Woltran, Towards fixed-parameter tractable algorithms for argumentation, in: F. Lin, U. Sattler, M. Truszczyński (Eds.), Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning, KR 2010, AAAI Press, 2010, pp. 112–122.
[113] W. Dvořák, S. Szeider, S. Woltran, Reasoning in argumentation frameworks of bounded clique-width, in: P. Baroni, F. Cerutti, M. Giacomin, G.R. Simari (Eds.), Proceedings of the 3rd Conference on Computational Models of Argument, COMMA 2010, in: Frontiers in Artificial Intelligence and Applications, IOS Press, 2010, pp. 219–230.
[114] W. Dvořák, S. Ordyniak, S. Szeider, Augmenting tractable fragments of abstract argumentation, Artif. Intell. 186 (2012) 157–173.
[115] W. Dvořák, M. Morak, C. Nopp, S. Woltran, dynPARTIX – a dynamic programming reasoner for abstract argumentation, in: H. Tompits, S. Abreu, J. Oetsch, J. Pührer, D. Seipel, M. Umeda, A. Wolf (Eds.), Proceedings of the 19th International Conference on Applications of Declarative Programming and Knowledge Management, INAP 2011, Revised Selected Papers, in: Lecture Notes in Artificial Intelligence, vol. 7773, Springer, 2013, pp. 259–268.
[116] B. Bliem, M. Morak, S. Woltran, D-FLAT: declarative problem solving using tree decompositions and answer-set programming, Theory Pract. Log. Program. 12 (4–5) (2012) 445–464.
[117] N. Robertson, P.D. Seymour, Graph minors. III. Planar tree-width, J. Comb. Theory, Ser. B 36 (1) (1984) 49–64.
[118] T. Kloks, Treewidth: Computations and Approximations, Lecture Notes in Computer Science, vol. 842, Springer, 1994.
[119] S. Arnborg, D.G. Corneil, A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Algebr. Discrete Methods 8 (1987) 277–284.
[120] H.L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (6) (1996) 1305–1317.
[121] H.L. Bodlaender, A.M. Koster, Treewidth computations I. Upper bounds, Inf. Comput. 208 (3) (2010) 259–275.
[122] A. Dermaku, T. Ganzow, G. Gottlob, B. McMahan, N. Musliu, M. Samer, Heuristic methods for hypertree decomposition, in: A. Gelbukh, E.F. Morales (Eds.), Proceedings of the 7th Mexican International Conference on Artificial Intelligence (MICAI 2008): Advances in Artificial Intelligence, in: Lecture Notes in Computer Science, vol. 5317, Springer, 2008, pp. 1–11.
[123] F. Cerutti, M. Giacomin, M. Vallati, Generating challenging benchmark AFs, in: S. Parsons, N. Oren, C. Reed, F. Cerutti (Eds.), Proceedings of the 5th International Conference on Computational Models of Argument, COMMA 2014, in: Frontiers in Artificial Intelligence and Applications, vol. 266, IOS Press, 2014, pp. 457–458.
[124] P.M. Dung, P. Mancarella, F. Toni, Computing ideal sceptical argumentation, Artif. Intell. 171 (10–15) (2007) 642–674.
[125] M. 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, 2007, pp. 81–87.
[126] P. Baroni, M. Giacomin, G. Guida, SCC-recursiveness: a general schema for argumentation semantics, Artif. Intell. 168 (1–2) (2005) 162–210.
[127] W. Dvořák, S.A. Gaggl, Stage semantics and the SCC-recursive schema for argumentation semantics, J. Log. Comput. (2014), http://dx.doi.org/10.1093/logcom/exu006, in press.
[128] P. Baroni, P.E. Dunne, M. Giacomin, On the resolution-based family of abstract argumentation semantics and its grounded instance, Artif. Intell. 175 (3–4) (2011) 791–813.
[129] F. Cerutti, M. Giacomin, M. Vallati, M. Zanella, An SCC recursive meta-algorithm for computing preferred labellings in abstract argumentation, in: C. Baral, G.D. Giacomo, T. Eiter (Eds.), Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning, KR 2014, AAAI Press, 2014, pp. 42–51.
[130] S. Bistarelli, F. Rossi, F. Santini, A first comparison of abstract argumentation systems: a computational perspective, in: D. Cantone, M.N. Asmundo (Eds.), Proceedings of the 28th Italian Conference on Computational Logic, CILC 2013, in: CEUR Workshop Proceedings, CEUR-WS.org, vol. 1068, 2013, pp. 241–245.
[131] S. Bistarelli, F. Rossi, F. Santini, Enumerating extensions on random abstract-AFs with ArgTools, Aspartix, ConArg2, and Dung-O-Matic, in: N. Bulling, L.W.N. van der Torre, S. Villata, W. Jamroga, W. Vasconcelos (Eds.), Proceedings of the 15th International Workshop on Computational Logic in Multi-Agent Systems, CLIMA XV, in: Lecture Notes in Computer Science, vol. 8624, Springer, 2014, pp. 70–86.
[132] S. Modgil, F. Toni, F. Bex, I. Bratko, C.I. Chesñevar, W. Dvořák, M.A. Falappa, X. Fan, S.A. Gaggl, A.J. García, et al., The added value of argumentation, in: Agreement Technologies, Springer, 2013, pp. 357–403.
[133] S.A. Gaggl, S. Woltran, cf2 semantics revisited, in: P. Baroni, F. Cerutti, M. Giacomin, G.R. Simari (Eds.), Proceedings of the 3rd Conference on Computational Models of Argument, COMMA 2010, in: Frontiers in Artificial Intelligence and Applications, vol. 216, IOS Press, 2010, pp. 243–254.
[134] W. Dvořák, P.E. Dunne, S. Woltran, Parametric properties of ideal semantics, in: T. Walsh (Ed.), Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011, IJCAI/AAAI, 2011, pp. 851–856.
[135] M. Caminada, A labelling approach for ideal and stage semantics, Argument Comput. 2 (2011) 1–21.
[136] R. Baumann, Splitting an argumentation framework, in: J.P. Delgrande, W. Faber (Eds.), Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2011, in: Lecture Notes in Computer Science, vol. 6645, Springer, 2011, pp. 40–53.
[137] R. Baumann, G. Brewka, R. Wong, Splitting argumentation frameworks: an empirical evaluation, in: S. Modgil, N. Oren, F. Toni (Eds.), Proceedings of the 1st International Workshop on Theory and Applications of Formal Argumentation, TAFA 2011, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 7132, Springer, 2012, pp. 17–31.
[138] R. Baumann, G. Brewka, W. Dvořák, S. Woltran, Parameterized splitting: a simple modification-based approach, in: E. Erdem, J. Lee, Y. Lierler, D. Pearce (Eds.), Correct Reasoning – Essays on Logic-Based AI in Honour of Vladimir Lifschitz, in: Lecture Notes in Computer Science, vol. 7265, Springer, 2012, pp. 57–71.
[139] B. Liao, H. Huang, Partial semantics of argumentation: basic properties and empirical results, J. Log. Comput. 23 (3) (2012) 541–562.
[140] E. Oikarinen, S. Woltran, Characterizing strong equivalence for argumentation frameworks, Artif. Intell. 175 (14–15) (2011) 1985–2009.
[141] R. Baumann, Normal and strong expansion equivalence for argumentation frameworks, Artif. Intell. 193 (2012) 18–44.
[142] P. Baroni, G. Boella, F. Cerutti, M. Giacomin, L. van der Torre, S. Villata, On the input/output behavior of argumentation frameworks, Artif. Intell. 217 (2014) 144–197.
[143] L. Amgoud, L. Bodenstaff, M. Caminada, P. McBurney, S. Parsons, H. Prakken, J. van Veenen, G. Vreeswijk, Final review and report on formal argumentation system, Deliverable D2. 6, Tech. rep., ASPIC IST-FP6-002307, 2006.
[144] M. Snaith, C. Reed, TOAST: online ASPIC$^+$ implementation, in: B. Verheij, S. Szeider, S. Woltran (Eds.), Proceedings of the 4th Conference on Computational Models of Argument, COMMA 2012, in: Frontiers in Artificial Intelligence and Applications, vol. 245, IOS Press, 2012, pp. 509–510.
[145] T.F. Gordon, The Carneades web service, in: B. Verheij, S. Szeider, S. Woltran (Eds.), Proceedings of the 4th Conference on Computational Models of Argument, COMMA 2012, in: Frontiers in Artificial Intelligence and Applications, vol. 245, IOS Press, 2012, pp. 517–518.
[146] V. Efstathiou, A. Hunter, Algorithms for generating arguments and counterarguments in propositional logic, Int. J. Approx. Reason. 52 (6) (2011) 672–704.
[147] G. Charwat, J.P. Wallner, S. Woltran, Utilizing ASP for generating and visualizing argumentation frameworks, in: M. Fink, Y. Lierler (Eds.), Proceedings of the 5th International Workshop on Answer Set Programming and Other Computing Paradigms, ASPOCP 2012, 2012, pp. 51–65.
[148] P. Besnard, A. Hunter, A logic-based theory of deductive arguments, Artif. Intell. 128 (1–2) (2001) 203–235.

[149] B. van Gijzel, H. Nilsson, Towards a framework for the implementation and verification of translations between argumentation models, in: R. Plasmeijer (Ed.), Proceedings of the 25th symposium on Implementation and Application of Functional Languages, IFL 2013, ACM, 2013, pp. 93–103.
[150] T. Alsinet, R. Béjar, L. Godo, F. Guitart, Using answer set programming for an scalable implementation of defeasible argumentation, in: Proceedings of the IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, IEEE, 2012, pp. 1016–1021.
[151] J. Schneider, T. Groza, A. Passant, A review of argumentation for the social semantic web, Semant. Web 4 (2) (2013) 159–218.
[152] W. Dvořák, S.A. Gaggl, S. Szeider, S. Woltran, Benchmark libraries for argumentation, in: S. Ossowski (Ed.), Agreement Technologies, in: LGTS, vol. 8, Springer, 2012, pp. 389–393, Ch. The Added Value of Argumentation.
[153] R. Baumann, G. Brewka, Expanding argumentation frameworks: enforcing and monotonicity results, in: P. Baroni, F. Cerutti, M. Giacomin, G.R. Simari (Eds.), Proceedings of the 3rd Conference on Computational Models of Argument, COMMA 2010, in: Frontiers in Artificial Intelligence and Applications, vol. 216, IOS Press, 2010, pp. 75–86.
[154] C. Cayrol, F.D. de Saint-Cyr, M.-C. Lagasquie-Schiex, Change in abstract argumentation frameworks: adding an argument, J. Artif. Intell. Res. 38 (2010) 49–84.
[155] M.A. Falappa, A.J. García, G. Kern-Isberner, G.R. Simari, On the evolving relation between belief revision and argumentation, Knowl. Eng. Rev. 26 (1) (2011) 35–43.
[156] B. Liao, L. Jin, R.C. Koons, Dynamics of argumentation systems: a division-based method, Artif. Intell. 175 (11) (2011) 1790–1814.