Argument & Computation 9 (2018) 41–72
DOI 10.3233/AAC-170031
IOS Press

Abstract solvers for Dung's argumentation frameworks

Remi Brochenin [a], Thomas Linsbichler [b,*], Marco Maratea [a], Johannes P. Wallner [b] and Stefan Woltran [b]

[a] *Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi, Università di Genova, Italy*
E-mails: r.brochenin@tue.nl, marco@dibris.unige.it
[b] *Institute of Information Systems, TU Wien, Austria*
E-mails: linsbich@dbai.tuwien.ac.at, wallner@dbai.tuwien.ac.at, woltran@dbai.tuwien.ac.at

Abstract. Abstract solvers are a quite recent method to uniformly describe algorithms in a rigorous formal way via graphs. Compared to traditional methods like pseudo-code descriptions, abstract solvers have several advantages. In particular, they provide a uniform formal representation that allows for precise comparisons of different algorithms. Recently, this new methodology has proven successful in declarative paradigms such as Propositional Satisfiability and Answer Set Programming. In this paper, we apply this machinery to Dung's abstract argumentation frameworks. We first provide descriptions of several advanced algorithms for the preferred semantics in terms of abstract solvers. We also show how it is possible to obtain new abstract solutions by "combining" concepts of existing algorithms by means of combining abstract solvers. Then, we implemented a new solving procedure based on our findings in CEGARTIX, and call it CEGARTIX+. We finally show that CEGARTIX+ is competitive and complementary in its performance to CEGARTIX on benchmarks of the first and second argumentation competition.

Keywords: Abstract argumentation, abstract solvers

1. Introduction

Dung's concept of abstract argumentation [23] is nowadays a core formalism in Artificial Intelligence [4,46]. The problem of solving certain reasoning tasks on such frameworks is the centerpiece of many advanced higher-level argumentation systems. The problems to be solved can however be intractable and might even be hard for the second level of the polynomial hierarchy [24,26]. Thus, efficient and advanced algorithms have to be developed in order to deal with real-world size data within reasonable performance bounds. The argumentation community is currently facing this challenge [17]: Already the second edition [27] of the solver competition [50,51] was held in 2017. Thus, the number of new algorithms and systems is steadily increasing, and we expect this to continue in the (near) future. Being able to precisely analyze and compare already developed and new algorithms is a fundamental step in order to understand the ideas behind such high-performance systems, and to build a new generation of more efficient algorithms and solvers.

Usually, algorithms are presented by means of pseudo-code descriptions, but other communities have experienced that analyzing such algorithms on this basis may not be fruitful. More formal descriptions,

*Corresponding author. E-mail: linsbich@dbai.tuwien.ac.at.

which allow, e.g. for a uniform representation, and are more amenable to comparison and to state formal results, have thus been developed: a successful approach in this direction is the concept of *abstract solvers* [44]. Hereby, one characterizes the possible states of computation as nodes of a graph, and the techniques (i.e., the computation steps in the algorithms) as arcs between nodes. In this way, the whole solving process amounts to a path in the graph. This concept proved successful for SAT [44], and also has been applied to several variants of Answer Set Programming [6,36,37].

In this paper, we apply abstract solvers to certain problems in Dung's argumentation frameworks. In order to understand whether abstract solvers are well suited also for this domain, we consider quite advanced algorithms for solving problems that are hard for the second level of the polynomial hierarchy – the considered algorithms range from dedicated [45] to reduction-based [13,25] approaches (see [19] for a survey). We show that abstract solvers allow for convenient algorithms design resulting in a clear and mathematically precise description. Moreover, formal properties of the algorithms (i.e. correctness) are easily specified by means of related graph properties (i.e. reachability). We then illustrate how abstract solvers allow to highlight in a more clear way similarities and differences among solving algorithms: This paves the way for a uniform view of the three solving algorithms mentioned above, thus simplifying the *combination* of concepts implemented in different solvers in order to define new abstract solutions. We propose one such combination and, in order to test its viability, we implemented the outcome of this combination inside the well-known CEGARTIX solver [25] and show that the resulting solver CEGARTIX+ is complementary in terms of performance w.r.t. CEGARTIX for certain tasks under the preferred semantics. We do so by using benchmarks of the first and second argumentation competition, as well as instances from earlier work. This is an interesting result which shows that a combination based on abstract solvers is proven to be also useful in practice (for similar observations, see [36,44]). We finally show (with focus on CEGARTIX), how reasoning tasks under further semantics, other than preferred, can be solved with this framework, and demonstrate how optimizations are easily added to our abstract solvers in a modular way.

To sum up, our main contributions are as follows:

- We provide a full formal description of recent algorithms [13,25,45] for reasoning tasks under the preferred semantics in terms of abstract solvers, thus enabling a comparison of these approaches at a formal level.
- We outline how our formal descriptions can be used to gain more insight into the algorithms, and how certain combinations can pave the way for new solutions.
- We implement such a new solution inside CEGARTIX and analyze its performance.
- We show how other semantics and optimizations can be incorporated to our abstract solvers.

The paper is structured as follows. Section 2 introduces the required preliminaries about abstract argumentation frameworks and abstract solvers. Then, Section 3 shows how our target algorithms are reformulated in terms of abstract solvers and introduces a new solving algorithm obtained from combining concepts from the target algorithms. Implementation and experimental analysis of the combined algorithm can be found in Section 4. Section 5 presents abstract solver representations of algorithms for reasoning tasks under other semantics, and indicates how shortcuts can be easily and modularly added. Section 6 provides a discussion of related research and Section 7 closes the paper with final remarks. We only include the full proofs of representative lemmata and theorems in the main body of the paper. The remaining proofs can be found in the Appendix.

The current paper extends and differs from an earlier version [8] as follows: (i) a new combination of abstract solvers is presented, which is easier to understand and more amenable to be implemented than

the one in [8], (ii) an implementation and experimental evaluation of the newly proposed algorithm are discussed, (iii) we apply additional and modified transition rules of algorithms for other semantics and optimizations (i.e. short-cuts) to the algorithms, with related formal results, and (iv) a detailed analysis of related work is provided.

2. Preliminaries

In this section we first review (abstract) argumentation frameworks [23] and their semantics (see [1] for an overview), and then introduce abstract solvers [44] on the concrete instance describing the Davis–Putnam–Logemann–Loveland (DPLL) procedure for SAT solving [20].

2.1. Abstract argumentation frameworks

An *argumentation framework (AF)* is a pair $F = (A, R)$ where A is a finite[1] set of arguments and $R \subseteq A \times A$ is the *attack relation*. Semantics for argumentation frameworks assign to each AF $F = (A, R)$ a set $\sigma(F) \subseteq 2^A$ of *extensions*. We consider here for σ the functions *adm*, *com*, and *prf*, which stand for admissible, complete, and preferred semantics. Towards the definitions of the semantics we need some formal concepts. For an AF $F = (A, R)$, 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 is a $c \in S$, such that $(c, b) \in R$ holds.

Definition 1. Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is *conflict-free (in F)*, denoted $S \in cf(F)$, if there are no $a, b \in S$ such that $(a, b) \in R$. For $S \in cf(F)$, it holds that

- $S \in adm(F)$ iff each $a \in S$ is defended by S;
- $S \in com(F)$ iff $S \in adm(F)$ and for each $a \in A$ defended by S, $a \in S$ holds;
- $S \in prf(F)$ iff $S \in adm(F)$ and there is no $T \in adm(F)$ with $T \supset S$, or equivalently,
- $S \in prf(F)$ iff $S \in com(F)$ and there is no $T \in com(F)$ with $T \supset S$.

Example 1. Consider the AF $F = (\{a, b, c, d\}, \{(a, b), (b, c), (b, d), (c, d), (d, c)\})$ depicted in Fig. 1 where nodes of the graph represent arguments and edges represent attacks. The extensions of F under admissible, complete, and preferred semantics are as follows: $adm(F) = \{\emptyset, \{a\}, \{a, c\}, \{a, d\}\}$, $com(F) = \{\{a\}, \{a, c\}, \{a, d\}\}$, and $prf(F) = \{\{a, c\}, \{a, d\}\}$.

Given an AF $F = (A, R)$, an argument $a \in A$, and a semantics σ, the problem of *skeptical acceptance* (Skept_σ) asks whether it is the case that a is contained in all σ-extensions of F; the problem of *credulous*

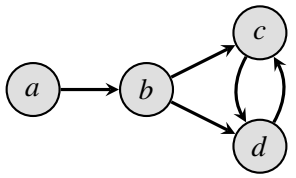


Fig. 1. AF F with $prf(F) = \{\{a, c\}, \{a, d\}\}$.

[1] In the literature also infinite AFs have been considered. We refer to [3] for the effects this has on semantics.

acceptance (Cred_σ) asks if a is contained in at least one σ-extension. While skeptical acceptance is trivial for *adm* and decidable in polynomial time for *com*, it is Π_2^P-complete[2] for *prf*, see [21,23,24].

2.2. Abstract solvers for SAT

Most SAT solvers are based on the Davis–Putnam–Logemann–Loveland (DPLL) procedure [20]. We give an abstract solver for DPLL following the work of Nieuwenhuis et al. [44]. The abstract solver is described by assigning a graph to each instance of the problem, where nodes and edges represent states and transitions of the actual solver, respectively. We start with basic notation for Boolean logic.

For a Conjunctive Normal Form (CNF) formula φ (resp. a set of literals M), we denote the set of atoms occurring in φ (resp. in M) by $atoms(\varphi)$ (resp. $atoms(M)$). A *literal* is an atom a or its negation $\neg a$. The *complement* $\overline{l}$ of literal l is defined as $\overline{a} = \neg a$ and $\overline{\neg a} = a$. We identify a consistent set E of literals (i.e. a set that does not contain a literal and its complement) with an assignment to $atoms(E)$ as follows: if $a \in E$ then a maps to *true*, while if $\neg a \in E$ then a maps to *false*. By $\mathrm{Sat}(\varphi)$ we refer to the set of satisfying assignments of φ.

We now introduce an abstract procedure for deciding whether a CNF formula is satisfiable. A *decision* literal is a literal annotated by d, as in l^d. An *annotated literal* is a literal, a decision literal or the false constant $\bot$. For a set X of atoms, a *record* relative to X is a string E composed of annotated literals over X without repetitions. For instance, $\emptyset$, $\neg a^d$ and $a\ \neg a^d$ are records relative to the set $\{a\}$. We say that a record E is *inconsistent* if it contains $\bot$ or both a literal l and its complement $\overline{l}$, and consistent otherwise. Moreover, by *unsat* we represent an inconsistent and decision-free record. We sometimes identify a record with the set containing all its elements without annotations, i.e. with an assignment to the atoms. For example, we identify the consistent record $b^d\ \neg a$ with the consistent set $\{\neg a, b\}$ of literals, and so with the assignment which maps a to *false* and b to *true*. Finally, $|E|$ denotes the number of literals in record E.

Each CNF formula φ determines its DPLL *graph* DP_φ. The set of nodes (states) of DP_φ consists of the records relative to $atoms(\varphi)$ and two distinguished states *Accept* and *Reject*. The edges of the graph DP_φ are specified by the transition rules presented in Fig. 2. A node in the graph is *terminal* if no edge originates from it; in practice, the terminal nodes are *Accept* and *Reject*. The *initial state* of the abstract solver is the empty record $\emptyset$. In solvers, generally the oracle rules are chosen with the preference order according to the order in which they are stated in Fig. 2. An exception is the failing rule, which has a higher priority than all the oracle rules.

Intuitively, every state of the DPLL graph represents some hypothetical state of the DPLL computation whereas a path in the graph is a description of a process of search for a satisfying assignment of a given CNF formula. The rule *Decide* asserts that we make an arbitrary decision to add a literal or, in other words, to assign a value to an atom. Since this decision is arbitrary, we are allowed to backtrack at a later point. The rule *UnitPropagate* asserts that we can add a literal that is a logical consequence of our previous decisions and the given formula. The rule *Backtrack* asserts that the present state of computation is failing but can be fixed: at some point in the past we added a decision literal whose value we can now reverse. The rule *Fail* asserts that the current state of computation has failed and cannot be fixed. The rule *Succeed* asserts that the current state of computation corresponds to a successful outcome.

[2]The class $\Pi_2^P = \mathrm{coNP}^{\mathrm{NP}}$ denotes the class of problems $\mathcal{P}$, such that the complementary problem $\overline{\mathcal{P}}$ can be decided by a nondeterministic polynomial time algorithm that has (unrestricted) access to an NP-oracle.

Oracle rules

Backtrack	$El^d E'' \Rightarrow E\bar{l}$	if	$El^d E''$ is inconsistent and E'' contains no decision literal
UnitPropagate	$E \Rightarrow El$	if	l does not occur in E and $C \vee l$ is a clause in φ and all the literals of $\overline{C}$ occur in E
Decide	$E \Rightarrow El^d$	if	E is consistent and neither l nor $\bar{l}$ occur in E

Failing rule

Fail	$E \Rightarrow \mathit{Reject}$	if	E is inconsistent and decision-free

Succeeding rule

Succeed	$E \Rightarrow \mathit{Accept}$	if	no other rule applies

Fig. 2. The transition rules of DP_φ.

Initial state:	$\emptyset$
Decide	$\Rightarrow a^d$
UnitPropagate	$\Rightarrow a^d\ c$
Decide	$\Rightarrow a^d\ c\ b^d$
Succeed	$\Rightarrow \mathit{Accept}$

Initial state:	$\emptyset$	
Decide	$\Rightarrow a^d$	
Decide	$\Rightarrow a^d\ \overline{c}^d$	$(*)$
UnitPropagate	$\Rightarrow a^d\ \overline{c}^d\ c$	
Backtrack	$\Rightarrow a^d\ c$	
Decide	$\Rightarrow a^d\ c\ b^d$	
Succeed	$\Rightarrow \mathit{Accept}$	

Fig. 3. Examples of paths in $DP_{\{a\vee b,\overline{a}\vee c\}}$.

To decide the satisfiability of a CNF formula it is enough to find a path in DP_φ leading from state $\emptyset$ to a terminal state. If it is *Accept*, then the formula is satisfiable, and if it is *Reject*, then it is unsatisfiable. Since there is no infinite path, a terminal state is always reached. The following result states this observation formally.

Theorem 2.1. *For any CNF formula φ, the graph DP_φ is finite and acyclic; any terminal state of DP_φ reachable from the initial state other than Reject is Accept; the record in the state preceding Accept corresponds to satisfying assignment of φ; and Reject is reachable if and only if φ is unsatisfiable.*

A proof of this theorem can be found in [36, Prop. 1] and (using a slightly different statement) in [44, Lemma 2.9]. The fact that *Accept* is reachable from the initial state iff φ is satisfiable follows directly.

Figure 3 presents two paths in DP_φ from the initial state $\emptyset$ to the terminal state *Accept*. Every edge is annotated on the left by the name of the transition rule that gives rise to this edge in DP_φ. Thus, Theorem 2.1 asserts that φ is satisfiable; moreover, the record where the *Succeed* rule is applied corresponds to a satisfying assignment of φ, i.e. $\{a, c, b\}$.

The difference between the paths in Fig. 3 is that only the path on the left will be indeed followed by SAT solvers, given it adheres with the ordering followed by SAT solvers, while the path on the right applies *Decide* (see (*)) where *UnitPropagate* is applicable.

2.3. Abstract solvers for computing maximal satisfying assignments

We now define a modification of the graph presented in the previous sub-section that will be useful in the definition of a new solving algorithm in Section 3.4.

The goal of this graph is to compute a maximal satisfying assignment of a CNF formula in the sense that the set of atoms mapped to true is $\subseteq$-maximal among all satisfying assignments. In order to do this it is enough to modify the graph DP_φ such that *Decide* always assigns the decision literal to true by default, i.e. to substitute rule *Decide* in Fig. 2 with the following rule $Decide^\prec$, where a represents an atom instead of a literal:

$$Decide^\prec \quad E \Rightarrow Ea^d \quad \text{if} \begin{cases} E \text{ is consistent and} \\ \text{neither } a \text{ nor } \neg a \text{ occur in } E \end{cases}$$

Let us call the resulting graph $DP_\varphi^\prec$, whose nodes correspond to the nodes of DP_φ graph. We can state the following theorem.

Theorem 2.2. *For any CNF formula φ, the graph $DP_\varphi^\prec$ is finite and acyclic; any terminal state of $DP_\varphi^\prec$ reachable from the initial state is either Reject or Accept; the record in the state preceding Accept corresponds to a maximal satisfying assignment of φ, and Reject is reachable if and only if φ is unsatisfiable.*

Proofs of this theorem can be found in [47, Theorem 2] and in [12, Prop. 1].

3. Algorithms for preferred semantics

In this section we first abstract two SAT-based algorithms for preferred semantics, namely PrefSat [13] (implemented in the tool ArgSemSat [14]) for extension enumeration, and an algorithm for deciding skeptical acceptance of cegartix [25]. Moreover, we abstract the dedicated approach for enumeration of [45]. Finally, in Section 3.4 we show how our graph representations can be used to develop a novel algorithm, by combining parts of cegartix and $DP^\prec$.

A key insight of the SAT-based algorithms is that preferred extensions can be found by iterative computation of certain extensions of a base semantics (admissible or complete): first, any extension of the base semantics is computed, and then, in each step, a strictly bigger (w.r.t. subset) one is computed. As these subproblems are in NP, each step is delegated to a SAT solver. The direct approach from [45], on the other hand, does not rely on external SAT solvers but uses a genuine procedure to compute preferred extensions. What the algorithms have in common is that they maintain a list of already found preferred extensions by which they constrain the search for new ones. All algorithms continue the search for new extensions until none can be found, the algorithm for skeptical acceptance just adds the constraint that the queried argument must not be contained.

We will present these algorithms in a uniform way via abstract solvers, abstracting from some minor tool-specific details. By presenting algorithms in such a uniform way, the relation among these algorithms becomes much clearer than by using, e.g. pseudo-code-based descriptions. In fact, common to all algorithms is a conceptual two-level architecture of computation, similar to Answer Set Programming solvers for disjunctive logic programs [6]. The lower level corresponds to a DPLL-like search subprocedure, while the higher level part takes care of the specific theory and drives the overall algorithm.

For PrefSat and CEGARTIX, the subprocedures actually are delegated to a SAT solver, while the dedicated approach carries out a tailored search procedure. Such common architecture is difficult to spot by looking at the related pseudo-code descriptions, while it will be clear by employing abstract solvers.

Each algorithm uses its own data structures, and, by slight abuse of notation, for a given AF $F = (A, R)$, the variables they use are denoted by $atoms(F)$. For this set it holds that $A \subseteq atoms(F)$, i.e. there is an atom for each argument. The states of all the graph representations we will define are either

(1) an annotated triple $(\epsilon, E', E)_i$ where $i \in \{out, base, max\}$, $\epsilon \subseteq 2^A$ is a set of sets of arguments, and both E' and E are records over $atoms(F)$; or
(2) $Ok(\epsilon)$ for $\epsilon \subseteq 2^A$; or
(3) a distinguished state *Accept* or *Reject*.

The intended meaning of a state $(\epsilon, E', E)_i$ is that ϵ is the set of already found preferred extensions of F (visited part of the search space), E' is a record representing the current candidate extension (which is admissible or complete in F and, for the SAT-based algorithms, has to be extended in the next iteration), E is a record that may be currently modified, and i refers to the current level of computation. Note that both E and E' are records, and they will be modified in the course of the computation; on the other hand found preferred extensions will be translated to a set of arguments before being stored in ϵ, and permanently left there unmodified. The annotation i denotes the current level of computation the procedure is in. Both annotations *base* and *max* correspond to different lower level computations, typically SAT calls, while *out* is used for states in which (simple) checks outside such procedures have to be made. Transition rules reflecting the higher level of computation shift these annotations, e.g. a shift from a *out* to *base* means that the algorithm is starting a call to a SAT solver. Transition rules mirroring rules "inside" a SAT solver do not modify i. A path through a graph made up of such states, representing a run of an algorithm, will then usually start in an *out* state, contain several subpaths consisting exclusively of either *base* states or *max* states, and finally end in the state $Ok(\epsilon)$ (for enumeration algorithms) or in one of the states *Accept* or *Reject* (for acceptance algorithms).

The remaining states denote terminated computation: $Ok(\epsilon)$ contains all solutions to the enumeration problem, while *Accept* or *Reject* denote an answer to a decision problem.

The SAT-based algorithms construct formulas by a function f s.t. $A \subseteq atoms(f(\epsilon, E, F, \alpha)) \subseteq atoms(F)$ for all possible arguments of f. The formulas $f(\epsilon, E, F, \alpha)$ are adapted from [5]. The argument α is relevant only for CEGARTIX to decide skeptical acceptance of α. Finally, we use $e(E) = E \cap A$ to project the arguments from a record E on the set of arguments A.

We now define a strict partial order on states, that will be used to show acyclicity of graphs later in this section. First, we define a particular representation of records used for lexicographic comparison.

Definition 2. Let E be a record. E can be written as $L^0 l^1 L^1 \dots l^p L^p$ where $L^0, \dots, L^p$ are strings of non-decision literals and $l^1, \dots, l^p$ are all the decision literals of E. We define the sequence representation of E as $s(E) = |L^0||L^1| \dots |L^p|$. For two sequence representations of records $s(E_1) = x_1\, x_2\, \dots\, x_{k_1}$ and $s(E_2) = y_1\, y_2\, \dots\, y_{k_2}$, we say that $s(E_1)$ is lexicographically smaller than $s(E_2)$, $s(E_1) <_{\text{lex}} s(E_2)$, if $x_n < y_n$ for the first index n where x_n and y_n differ with $n \leqslant \min(k_1, k_2)$, or if $k_1 < k_2$ and for all $n \leqslant \min(k_1, k_2)$ we have $x_n = y_n$.

Example 2. Consider the records $E_1 = \neg b c^d$, $E_2 = \neg b c^d d^d$, and $E_1 = b c^d \neg d$. The sequence representations of these records are given by $s(E_1) = 1\,0$, $s(E_2) = 1\,0\,0$, and $s(E_3) = 1\,1$. The lexicographic ordering is $s(E_1) <_{\text{lex}} s(E_2) <_{\text{lex}} s(E_3)$.

The orders on states are now defined in a way that the graphs produced by the abstract solvers presented in this section only feature edges between states ς_1 and ς_2 such that $\varsigma_1 < \varsigma_2$.

Definition 3. Let ϵ_1, ϵ_2 be sets of sets of arguments, E_1, E_2, E'_1, E'_2 be records, and $i_1, i_2 \in \{base, max, out\}$. We define the following strict partial orders (i.e. irreflexive and transitive binary relations):

$<_\epsilon$: $\epsilon_1 <_\epsilon \epsilon_2$ iff $\epsilon_1 \subset \epsilon_2$.
$<_{E'}$: $E'_1 <_{E'} E'_2$ iff $e(E'_1) \subset e(E'_2)$.
$<_E$: $E_1 <_E E_2$ iff $s(E_1) <_{\text{lex}} s(E_2)$, where $<_{\text{lex}}$ is the lexicographic order.
$<_i$: $base <_i max <_i out$.

The strict partial order $<$ on states is defined such that for any two states $\varsigma_1 = (\epsilon_1, E'_1, E_1)_{i_1}$ and $\varsigma_2 = (\epsilon_2, E'_2, E_2)_{i_2}$, $\varsigma_1 < \varsigma_2$ iff

(i) $\epsilon_1 <_\epsilon \epsilon_2$, or
(ii) $\epsilon_1 = \epsilon_2$ and $i_1 <_i i_2$, or
(iii) $\epsilon_1 = \epsilon_2$ and $i_1 = i_2$ and $E'_1 <_{E'} E'_2$, or
(iv) $\epsilon_1 = \epsilon_2$ and $i_1 = i_2$ and $E'_1 = E'_2$ and $E_1 <_E E_2$.

Example 3. Consider the states $\varsigma_1 = (\emptyset, out, c^d, b^d\neg d)$, $\varsigma_2 = (\{\{a\}\}, base, c^d, b^d\neg d)$, $\varsigma_3 = (\{\{a\}\}, max, b\neg c, b^d\neg d)$, $\varsigma_4 = (\{\{a\}\}, max, bd^d, \neg bc^d)$, and $\varsigma_5 = (\{\{a\}\}, max, bd^d, \neg bc^d d^d)$. It holds that $\varsigma_1 < \varsigma_2 < \varsigma_3 < \varsigma_4 < \varsigma_5$. First, $\varsigma_1 < \varsigma_2$ holds due to $\emptyset \subset \{\{a\}\}$. Moreover, $\varsigma_2 < \varsigma_3$ is because of $base <_i max$. Furthermore, $\varsigma_3 < \varsigma_4$ holds since $e(b\neg c) = \{b\} \subset \{b, d\} = e(bd^d)$. Finally, observe that $\neg bc^d d^d$ can be written, in the spirit of Definition 2, as $\neg b\, c^d\, \emptyset\, d^d\, \emptyset$, where $\emptyset$ denotes the empty string. Hence we obtain $s(\neg bc^d d^d) = 1\,0\,0$ and similarly $s(\neg bc^d) = 1\,0$. We get $\varsigma_4 < \varsigma_5$ since $s(\neg bc^d) = 1\,0 <_{\text{lex}} 1\,0\,0 = s(\neg bc^d d^d)$.

One can check that all orders on elements are transitive and irreflexive. Therefore the construction of $<$ also ensures these properties for the order on states.

3.1. SAT-based algorithm for enumeration

PrefSat (Algorithm 1 of [13]) is a SAT-based algorithm for finding all preferred extensions of a given AF. The algorithm maintains a list of visited preferred extensions. It first searches for a complete extension not contained in previously found preferred extensions. If such an extension is found, it is iteratively extended until we reach a subset-maximal complete extension, which is a preferred extension by definition. This preferred extension is stored, and we repeat the process.

This algorithm is realized by two subprocedures that are delegated to a SAT solver. The first has to generate a complete extension not contained in one of the enumerated preferred extensions, and the second searches for a complete extension that is a strict superset of a given one.

We now represent PrefSat as an abstract solver. The graph $\text{ENUM}^F_{\overline{f}}$ for an AF F and a vector of functions $\overline{f} = (f^{com}_{base}, f^{com}_{max})$ is defined by the states over $atoms(F)$ and the transition rules presented in Fig. 4. Its initial state is $(\emptyset, \emptyset, \emptyset)_{base}$. We assume the functions f^{com}_{base} and f^{com}_{max} that generate CNF formulas for $\epsilon \subseteq 2^A$, a record E, and an argument $\alpha \in A$ such that:

$$\left\{e(M) \mid M \in \text{Sat}\left(f^{com}_{base}(\epsilon, E, F, \alpha)\right)\right\} = \left\{S \in com(F) \mid \neg\exists S' \in \epsilon : S \subseteq S'\right\};$$

$$\left\{e(M) \mid M \in \text{Sat}\left(f^{com}_{max}(\epsilon, E, F, \alpha)\right)\right\} = \left\{S \in com(F) \mid e(E) \subset S\right\}.$$

i-oracle rules ($i \in \{base, max\}$)

$$Backtrack_i \quad (\epsilon, E', El^d E'')_i \Rightarrow (\epsilon, E', E\bar{l})_i \qquad \text{if} \begin{cases} El^d E'' \text{ is inconsistent and} \\ E'' \text{ contains no decision literal} \end{cases}$$

$$UnitPropagate_i \quad \big(\epsilon, E', E\big)_i \Rightarrow \big(\epsilon, E', El\big)_i \qquad \text{if} \begin{cases} l \text{ does not occur in } E \text{ and} \\ C \vee l \text{ is a clause in } f_i^{com}(\epsilon, E', F, \alpha) \text{ and} \\ \text{all the literals of } \overline{C} \text{ occur in } E \end{cases}$$

$$Decide_i \quad \big(\epsilon, E', E\big)_i \Rightarrow \big(\epsilon, E', El^d\big)_i \qquad \text{if} \begin{cases} E \text{ is consistent and} \\ \text{neither } l \text{ nor } \bar{l} \text{ occur in } E \end{cases}$$

Failing rules

$$Fail_{base} \quad \big(\epsilon, E', E\big)_{base} \Rightarrow Ok(\epsilon) \qquad \text{if} \{ E \text{ is inconsistent and decision-free}$$

$$Fail_{max} \quad \big(\epsilon, E', E\big)_{max} \Rightarrow \big(\epsilon \cup \{e(E')\}, \emptyset, \emptyset\big)_{base} \qquad \text{if} \{ E \text{ is inconsistent and decision-free}$$

Succeeding rules

$$Succeed_{base} \ \big(\epsilon, E', E\big)_{base} \Rightarrow (\epsilon, E, \emptyset)_{max} \qquad \text{if} \{ \text{no other rule applies}$$

$$Succeed_{max} \ \big(\epsilon, E', E\big)_{max} \Rightarrow (\epsilon, E, \emptyset)_{max} \qquad \text{if} \{ \text{no other rule applies}$$

Fig. 4. The rules of ENUM^F_f.

In words, the models of the formula $f^{com}_{base}(\epsilon, E, F, \alpha)$ represent the complete extensions of F such that no superset is contained in ϵ. Moreover, the models of $f^{com}_{max}(\epsilon, E, F, \alpha)$ represent the complete extensions of F strictly extending the extension represented by E. Hence, these are the formulas that are handed to a SAT solver in PrefSat in order to solve the necessary subprocedures.

We remark that α is not relevant for enumeration of extensions and only used for acceptance later on. In the interest of uniformity we keep it as parameter of the functions throughout the paper. Recall that in a state $(\epsilon, E', E)_i$ the set ϵ represents preferred extensions found as of now, E' is a record for the complete extension found in the previous oracle run and E is a record for the complete extension that the current oracle is trying to build. The annotation $i \in \{base, max\}$ corresponds to different kinds of SAT calls.

As the oracle rules with annotation $i \in \{base, max\}$ coincide with the ones of DP_φ (cf. Fig. 2), their role is to search for a satisfying assignment of f^{com}_i. That is, if a $Fail_i$ rule is applied to the state $(\epsilon, E', E)_i$ for i, the formula $f^{com}_i(\epsilon, E', F, \alpha)$ is unsatisfiable. Conversely, when a $Succeed_i$ rule is applied from that state, the formula $f^{com}_i(\epsilon, E', F, \alpha)$ is satisfied by E. Notice that $Fail_i$ and $Succeed_i$ might shift i to reflect a change of type of SAT calls. When $i = base$, the oracle searches for a complete extension that is not contained in a preferred extension that has been found before. In case of failure all the preferred extensions have been found. In case of success, it is necessary to search whether there are strictly larger complete extensions than the one found. This is handled by the computation within states annotated by max. In case of success, $Succeed_{max}$ is applied and the procedure is repeated, since the current complete extension might still not be maximal. Failure by $Fail_{max}$ means we have found a preferred extension.

Example 4. Again consider the AF F depicted in Fig. 1. We have seen in Example 1 that F has two preferred extensions, namely $\{a, c\}$ and $\{a, d\}$. Figure 5 shows a possible path in the graph ENUM^F_f. As expected, the computation terminates in the state $Ok(\{\{a, d\}, \{a, c\}\})$. Note that we abbreviate the parts of the path where we are "inside" the SAT-solver. Also, we only show literals over arguments of F, and

Initial state: $(\emptyset, \emptyset, \emptyset)_{base}$
base-oracle: $(\emptyset, \emptyset, E_1 \supseteq \{a, \neg b, \neg c, \neg d\})_{base}$
$Succeed_{base}$: $(\emptyset, E_1, \emptyset)_{max}$
max-oracle: $(\emptyset, E_1, E_2 \supseteq \{a, \neg b, \neg c, d\})_{max}$
$Succeed_{max}$: $(\emptyset, E_2, \emptyset)_{max}$
max-oracle: $(\emptyset, E_2, unsat)_{max}$
$Fail_{max}$: $(\{\{a, d\}\}, \emptyset, \emptyset)_{base}$
base-oracle: $(\{\{a, d\}\}, \emptyset, E_3 \supseteq \{a, \neg b, c, \neg d\})_{base}$
$Succeed_{base}$: $(\{\{a, d\}\}, E_3, \emptyset)_{max}$
max-oracle: $(\{\{a, d\}\}, E_3, unsat)_{max}$
$Fail_{max}$: $(\{\{a, d\}, \{a, c\}\}, \emptyset, \emptyset)_{base}$
base-oracle: $(\{\{a, d\}, \{a, c\}\}, \emptyset, unsat)_{base}$
$Fail_{base}$: $Ok(\{\{a, d\}, \{a, c\}\})$

Fig. 5. Path in $\text{ENUM}^F_{\tilde{f}}$ where F is the AF from Fig. 1.

do not state the extra literals that may have been assigned during the call to the SAT-solver. Recall that by *unsat* we represent an inconsistent and decision-free record.

It remains to show correctness of $\text{ENUM}^F_{\tilde{f}}$ by showing that we reach a terminal state containing all preferred extensions of F. First observe that the oracle rules are exactly taken from DP_φ of Fig. 2, working on the third element of the state-triple. Moreover, this working record is always initialized with $\emptyset$ when a transition rule outside the oracle rules is applied. Therefore, we can immediately follow from Theorem 2.1:

Lemma 3.1. *For any AF F and $i \in \{base, max\}$, if $Succeed_i$ is applied from state $(\epsilon, E', E)_i$ in the graph $\text{ENUM}^F_{(f^{com}_{base}, f^{com}_{max})}$ then $E \in \text{Sat}(f^{com}_i(\epsilon, E', F, \alpha))$; if $Fail_i$ is applied then $f^{com}_i(\epsilon, E', F, \alpha)$ is unsatisfiable.*

We continue with a lemma stating that only preferred extensions which have not been found at this point are added to ϵ.

Lemma 3.2. *For any AF F, if the rule $Fail_{max}$ is applied from state $(\epsilon, E', E)_{max}$ in the graph $\text{ENUM}^F_{(f^{com}_{base}, f^{com}_{max})}$ then $e(E') \in prf(F)$ and $e(E') \notin \epsilon$.*

Proof. Let $(\epsilon_1, E'_1, E_1)_{max}$ be the state from which $Fail_{max}$ is applied. This means, by Lemma 3.1, that $f^{com}_{max}(\epsilon_1, E'_1, F, \alpha)$ is unsatisfiable, hence, by the definition of formula f^{com}_{max}, there is no $S \in com(F)$ with $S \supset e(E'_1)$. To get $e(E'_1) \in prf(F)$ it remains to show that $e(E'_1) \in com(F)$. Observe that $Succeed_{base}$ is applied at least once, since every AF has a complete extension. Moreover, the value of E'_1 is only updated by applications of $Succeed_{base}$ or $Succeed_{max}$. In both cases $e(E'_1)$ corresponds to a complete extension of F, due to the definitions of the formula f^{com}_{base} or f^{com}_{max}, respectively, and Lemma 3.1. Therefore $e(E'_1)$ is a complete extension of F.

Since the initial state is $(\emptyset, \emptyset, \emptyset)_{base}$, an application of $Succeed_{base}$ must precede $Fail_{max}$. From this application of $Succeed_{base}$ it follows from Lemma 3.1 that there is a record E' such that $\neg\exists S \in \epsilon : e(E') \subseteq S$. Moreover every application of $Succeed_{max}$ updates E' by a proper superset of itself. Therefore $e(E'_1) \supseteq e(E')$ and also $\neg\exists S \in \epsilon : e(E'_1) \subseteq S$, in particular $e(E'_1) \notin \epsilon$. □

Now we are ready to show correctness of $\text{ENUM}^F_{\tilde{f}}$.

Theorem 3.3. *For any AF F, the graph* $\textsc{Enum}^{F}_{(f^{com}_{base}, f^{com}_{max})}$ *is finite, acyclic and the only terminal state reachable from the initial state is $Ok(\epsilon)$ where $\epsilon = prf(F)$.*

Proof. In order to show that $\textsc{Enum}^{F}_{\overline{f}}$ is finite, consider some state $(\epsilon, E', E)_i$ of $\textsc{Enum}^{F}_{\overline{f}}$. Since both E and E' are records over $atoms(F)$, and F is finite by definition, the number of possible records E and E' is finite. Similarly, there is only a finite number of sets of sets of arguments ϵ. Finally, $\textsc{Enum}^{F}_{\overline{f}}$ only contains states with $i \in \{base, max\}$. Thus the number of states is finite in the graph $\textsc{Enum}^{F}_{\overline{f}}$.

In order to show that it is acyclic recall the strict partial order $<$ on states from Definition 3. We show that each transition rule is increasing w.r.t. $<$ by referring to the conditions (i) to (iv) from Definition 3. To this end consider two states $\varsigma_1 = (\epsilon_1, E'_1, E_1)_{i_1}$ and $\varsigma_2 = (\epsilon_2, E'_2, E_2)_{i_2}$ representing the states before and after the application of a rule. First of all, the i-oracle rules (i.e. $Backtrack_i$, $UnitPropagate_i$, and $Decide_i$) fulfill $\varsigma_1 < \varsigma_2$ because of (iv). For all of these rules $\epsilon_1 = \epsilon_2$, $E'_1 = E'_2$ and $i_1 = i_2$, but $s(E_1)$ is lexicographically smaller than $s(E_2)$, therefore $E_1 <_E E_2$. Moreover, $Fail_{max}$ fulfills $\varsigma_1 < \varsigma_2$ due to (i) since $e(E'_1) \notin \epsilon_1$ by Lemma 3.2. $Succeed_{base}$ guarantees $\varsigma_1 < \varsigma_2$ because of (ii). Finally, $Succeed_{max}$ fulfills $\varsigma_1 < \varsigma_2$ due to (iii), since the max-oracle rules work on the formula f^{com}_{max} and the extension associated with a satisfying assignment $E_1 = E'_2$ of that formula must be a proper superset of $e(E'_1)$. Therefore, by transitivity of $<$, or any two states ς_1 and ς_n such that ς_n is reachable from ς_1 in $\textsc{Enum}^{F}_{\overline{f}}$ it holds that $\varsigma_1 < \varsigma_n$, showing that the graph is acyclic.

The only terminal state reachable from the initial state is $Ok(\epsilon)$ (via rule $Fail_{base}$) for some $\epsilon \subseteq 2^A$ since all states $(\epsilon, E, E')_i$ of $\textsc{Enum}^{F}_{\overline{f}}$ have $i \in \{base, max\}$ and for each $i \in \{base, max\}$ there is a rule $Succeed_i$ with the condition "no other rule applies". It remains to show that, when state $Ok(\epsilon)$ is reached, ϵ coincides with $prf(F)$. Since elements are only added to ϵ by application of the rule $Fail_{max}$ we know from Lemma 3.2 that for each $T \in \epsilon$ it holds that $T \in prf(F)$. To reach $Ok(\epsilon)$, the rule $Fail_{base}$ must have been applied from a state $(\epsilon, E', E)_{base}$. This means, by the definition of f^{com}_{base} and Lemma 3.1, that for each complete extension S of F there is some $T \in \epsilon$ such that $S \subseteq T$. Hence $\epsilon = prf(F)$. □

3.2. SAT-based algorithm for acceptance

CEGARTIX [25] is a SAT-based tool for deciding several acceptance questions for AFs. Here we focus on Algorithm 1 of [25] for deciding skeptical acceptance w.r.t. preferred semantics of an argument α. Similarly to PrefSat, CEGARTIX traverses the search space of a certain semantics, generates candidate extensions not contained in already visited preferred extensions, and maximizes the candidate until a preferred extension is found. The main differences to PrefSat are (1) the parametrized use of base semantics σ (the search space), which can be either admissible or complete semantics, and (2) the incorporation of the queried argument α. To prune the search space, α must not be contained in the candidate σ-extension before maximization. Again, we have two kinds of SAT-calls.

The graph $\textsc{Skept-prf}^{F,\alpha}_{\overline{f}}$ for an AF F, an argument α and a vector of functions $\overline{f} = (f^{\sigma}_{base}, f^{\sigma}_{max})$ is defined by the states over $atoms(F)$ and the rules in Fig. 4 replacing the rules $Fail_i$ for $i \in \{base, max\}$ and adding the rules $Fail_{out}$ and $Succeed_{out}$ as depicted in Fig. 6. The initial state is $(\emptyset, \emptyset, \emptyset)_{base}$. For $\sigma \in \{adm, com\}$ we assume the functions f^{σ}_{base} and f^{σ}_{max} such that:

$$\big\{e(M) \mid M \in \mathrm{Sat}\big(f^{\sigma}_{base}(\epsilon, E, F, \alpha)\big)\big\} = \big\{S \in \sigma(F) \mid \alpha \notin S \wedge \neg\exists S' \in \epsilon : S \subseteq S'\big\};$$

$$\big\{e(M) \mid M \in \mathrm{Sat}\big(f^{\sigma}_{max}(\epsilon, E, F, \alpha)\big)\big\} = \big\{S \in \sigma(F) \mid e(E) \subset S\big\}.$$

Failing rules

$Fail_{base}$ $(\epsilon, E', E)_{base} \Rightarrow Accept$ if $\{$ E is inconsistent and decision-free

$Fail_{max}$ $(\epsilon, E', E)_{max} \Rightarrow (\epsilon, E', \emptyset)_{out}$ if $\{$ E is inconsistent and decision-free

$Fail_{out}$ $(\epsilon, E', E)_{out} \Rightarrow (\epsilon \cup \{e(E')\}, \emptyset, \emptyset)_{base}$ if $\{$ $\alpha \in e(E')$

Succeeding rules

$Succeed_{out}$ $(\epsilon, E', E)_{out} \Rightarrow Reject$ if $\{$ $\alpha \notin e(E')$

Fig. 6. Changed transition rules for SKEPT-PRF$_{\overline{f}}^{F,\alpha}$.

Initial state: $(\emptyset, \emptyset, \emptyset)_{base}$
base-oracle: $(\emptyset, \emptyset, E_1 \supseteq \{a, \neg b, \neg c, \neg d\})_{base}$
$Succeed_{base}$: $(\emptyset, E_1, \emptyset)_{max}$
max-oracle: $(\emptyset, E_1, E_2 \supseteq \{a, \neg b, c, \neg d\})_{max}$
$Succeed_{max}$: $(\emptyset, E_2, \emptyset)_{max}$
max-oracle: $(\emptyset, E_2, unsat)_{max}$
$Fail_{max}$: $(\emptyset, E_2, \emptyset)_{out}$
$Fail_{out}$: $(\{\{a, c\}\}, \emptyset, \emptyset)_{base}$
base-oracle: $(\{\{a, c\}\}, \emptyset, E_3 \supseteq \{a, \neg b, \neg c, d\})_{base}$
$Succeed_{base}$: $(\{\{a, c\}\}, E_3, \emptyset)_{max}$
max-oracle: $(\{\{a, c\}\}, E_3, unsat)_{max}$
$Fail_{max}$: $(\{\{a, c\}\}, E_3, \emptyset)_{out}$
$Succeed_{out}$: *Reject*

Fig. 7. Reject-path for argument c in SKEPT-PRF$_{\overline{f}}^{F,c}$.

The graph SKEPT-PRF$_{\overline{f}}^{F,\alpha}$ is similar to ENUM$_{\overline{f}}^{F}$. Again, the models of the formulas $f_{base}^{com}(\epsilon, E, F, \alpha)$ and $f_{max}^{com}(\epsilon, E, F, \alpha)$ represent the complete extensions of F which are not contained in any element of ϵ and extending the extension represented by E, respectively. In addition, the query argument α is required not to be contained in the extensions represented by the models of $f_{base}^{com}(\epsilon, E, F, \alpha)$. The graph differs in case of failure in a state annotated by *base* or *max*. When all the preferred extensions have been enumerated, i.e. the *base*-oracle ends with an application of $Fail_{base}$, we can report a positive outcome with *Accept*, since we have ensured that α belongs to all of them. If we arrive at $Fail_{max}$, i.e. when a preferred extension has been found, it is necessary to check whether α belongs to it, and this is done via rules $Succeed_{out}$ and $Fail_{out}$ that either lead to *Reject* or give the control to the *base* level.

Example 5. Again consider the AF F from Fig. 1 and note that skeptical acceptance of argument c is rejected as c is not contained in the preferred extension $\{a, d\}$ of F. Accordingly, the possible path of the graph SKEPT-PRF$_{\overline{f}}^{F,c}$ which is depicted in Fig. 7 (with base semantics *adm*) terminates in the *Reject*-state.

On the other hand, argument a is skeptically accepted w.r.t. preferred semantics in F as it belongs to all preferred extensions enumerated in $\{\{a, d\}, \{a, c\}\}$. For checking whether a is skeptically accepted in F, a possible path in the graph SKEPT-PRF$_{\overline{f}}^{F,a}$ (again with base semantics *adm*) is shown in Fig. 8. As expected, the path terminates in the state *Accept*.

Again, we get the following from Theorem 2.1:

Initial state: $(\emptyset, \emptyset, \emptyset)_{base}$
base-oracle: $(\emptyset, \emptyset, E_1 \supseteq \{\neg a, \neg b, \neg c, \neg d\})_{base}$
$Succeed_{base}$: $(\emptyset, E_1, \emptyset)_{max}$
max-oracle: $(\emptyset, E_1, E_2 \supseteq \{a, \neg b, \neg c, \neg d\})_{max}$
$Succeed_{max}$: $(\emptyset, E_2, \emptyset)_{max}$
max-oracle: $(\emptyset, E_2, E_3 \supseteq \{a, \neg b, \neg c, d\})_{max}$
$Succeed_{max}$: $(\emptyset, E_3, \emptyset)_{max}$
max-oracle: $(\emptyset, E_3, unsat)_{max}$
$Fail_{max}$: $(\emptyset, E_3, \emptyset)_{out}$
$Fail_{out}$: $(\{\{a, d\}\}, \emptyset, \emptyset)_{base}$
base-oracle: $(\{\{a, d\}\}, \emptyset, unsat)_{base}$
$Fail_{base}$: *Accept*

Fig. 8. Accept-path for argument a in $\text{SKEPT-PRF}_{\overline{f}}^{F,a}$.

Lemma 3.4. *For any AF $F = (A, R)$, argument $\alpha \in A$, $\sigma \in \{adm, com\}$, and $i \in \{base, max\}$, if $Succeed_i$ is applied from state $(\epsilon, E', E)_i$ in the graph $\text{SKEPT-PRF}_{\overline{f}}^{F,\alpha}$ then $E \in \text{Sat}(f_i^\sigma(\epsilon, E', F, \alpha))$; if $Fail_i$ is applied then $f_i^\sigma(\epsilon, E', F, \alpha)$ is unsatisfiable.*

The proof of the following results is almost identical to the ones of Lemma 3.2 and Theorem 3.3 and can be found in the Appendix.

Lemma 3.5. *For any AF F, if the rule $Fail_{out}$ is applied from state $(\epsilon, E', E)_{out}$ in the graph $\text{SKEPT-PRF}_{(f_{base}^\sigma, f_{max}^\sigma)}^{F,\alpha}$ with $\sigma \in \{adm, com\}$ then $e(E') \in prf(F)$ and $e(E') \notin \epsilon$.*

Theorem 3.6. *For any AF $F = (A, R)$, argument $\alpha \in A$, and $\sigma \in \{adm, com\}$, the graph $\text{SKEPT-PRF}_{(f_{base}^\sigma, f_{max}^\sigma)}^{F,\alpha}$ is finite, acyclic and any terminal state reachable from the initial state is either Accept or Reject; Reject is reachable iff α is not skeptically accepted in F w.r.t. prf.*

Finally note that from Theorem 3.6 it follows that *Accept* is reachable from the initial state if and only if α is skeptically accepted by F, which completes the correctness statement for $\text{SKEPT-PRF}_{\overline{f}}^{F,\alpha}$.

3.3. Dedicated approach for enumeration

Algorithm 1 of [45] presents a direct approach for enumerating preferred extensions. One function is important for this algorithm, which is called IN-TRANS. Given an AF $F = (A, R)$, it marks an argument $a \in A$ as belonging to the currently built extension, and marks all attackers $\{b \mid (b, a) \in R\}$ and all attacked arguments $\{b \mid (a, b) \in R\}$ as outside of this extension. Intuitively, IN-TRANS *decides* to accept a, and then *propagates* the immediate consequences to the neighboring nodes. It actually does an additional task. It labels the attacked arguments as "attacked", and the attackers that are not yet labelled as attacked as "to be attacked": this allows later to easily check the admissibility of the extension by just looking whether there is any argument "to be attacked".

The algorithm is recursive, and stores the admissible extensions in a global variable. First, it checks whether all the arguments are marked as either belonging to or being outside the extension, and if so it returns after adding the extension to the global variable if the extension is actually admissible. Second, it applies the function IN-TRANS to some unmarked argument and calls itself recursively. Third, it

Oracle rules

$$Backtrack'_{max} \quad (\epsilon, \emptyset, Ea^d E'')_{max} \Rightarrow (\epsilon, \emptyset, E\neg a)_{max} \quad \text{if} \begin{cases} Ea^d E'' \text{ is inconsistent and} \\ E'' \text{ contains no decision literal} \end{cases}$$

$$Propagate'_{max} \quad (\epsilon, \emptyset, E)_{max} \Rightarrow (\epsilon, \emptyset, E\neg a)_{max} \quad \text{if} \begin{cases} e(E) \text{ attacks } a \text{ or } a \text{ attacks } e(E) \text{ and} \\ \neg a \text{ does not occur in } E \end{cases}$$

$$Decide'_{max} \quad (\epsilon, \emptyset, E)_{max} \Rightarrow \left(\epsilon, \emptyset, Ea^d\right)_{max} \quad \text{if} \begin{cases} E \text{ is consistent and} \\ \text{neither } a \text{ nor } \neg a \text{ occur in } E \text{ and} \\ Propagate'_{max} \text{ does not apply} \end{cases}$$

Succeeding and failing rules

$$Fail_{max} \quad (\epsilon, \emptyset, E)_{max} \Rightarrow Ok(\epsilon) \quad \text{if} \{ E \text{ is incons. and decision-free}$$

$$Succeed_{max} \quad (\epsilon, \emptyset, E)_{max} \Rightarrow (\epsilon, \emptyset, E)_{out} \quad \text{if} \{ \text{no other rule applies}$$

$$Fail_{out} \quad (\epsilon, \emptyset, E)_{out} \Rightarrow (\epsilon, \emptyset, E\bot)_{max} \quad \text{if} \begin{cases} \exists E' \in \epsilon : E \subseteq E' \text{ or} \\ \text{there is an argument } a \text{ s.t.} \\ e(E) \text{ does not attack } a \text{ and } a \text{ attacks } e(E) \end{cases}$$

$$Succeed_{out} \quad (\epsilon, \emptyset, E)_{out} \Rightarrow \left(\epsilon \cup \{e(E)\}, \emptyset, E\bot\right)_{max} \quad \text{if} \{ \text{no other rule applies}$$

Fig. 9. The rules of the graph DIRECT^F.

reverts the effects of IN-TRANS, marks the argument it chose as outside of this extension, and calls itself recursively.

We have defined an equivalent representation of this algorithm that follows the framework of abstract solvers with binary logics as previously used in this article. Binary truth values are sufficient to represent the arguments marked, but we see the labels "attacked" and "to be attacked" as an optimization as they can be easily recovered at the end of the algorithm. Indeed, they correspond to the condition "there is an argument a such that $e(E)$ does not attack a and a attacks $e(E)$" of the rule $Fail_{out}$.

The graph DIRECT^F for an AF F is defined by the states over $atoms(F)$ and the transition rules presented in Fig. 9. Its initial state is $(\emptyset, \emptyset, \emptyset)_{max}$. The structure of the graph is similar to that of ENUM^F_f. It differs from this graph in two ways. First, it has only one lower level of computation. Second, the rules of the oracle differ from the previous oracle rules since they are not a call to a SAT solver; we primed them to emphasize the difference.

More precisely, among the oracle rules, propagation (through $Propagate'_{max}$ rule) now only occurs so as to negatively add an atom if it attacks or is attacked by an atom of the extension being built. The $Decide'_{max}$ rule only adds atoms positively, which is useful in Algorithm 2 of [45] as it ensures maximality of final assignments. When a record assigning all arguments is found, the rule $Succeed_{max}$ is applied so as to allow the test of the outer rules to be carried on. Differently to the algorithms presented so far, the extension associated to this record is only guaranteed to be conflict-free at this point and not admissible (or complete, depending on the chosen base semantics). When the record corresponds to a preferred extension, it is stored through $Succeed_{out}$, and the process continues. In both $Succeed_{out}$ and $Fail_{out}$, the use of one of the rules $Backtrack'_{max}$ or $Fail_{max}$ is forced by making the record inconsistent. This way the process of browsing records is forced to continue.

A final comment is related to one of the main advantages of using abstract solvers, i.e. the fact that they allow to highlight in a more clear way similarities and differences among solving algorithms, as mentioned in Section 1. It is evident that our reformulation of the direct approach has allowed to present

$$
\begin{array}{ll}
\text{Initial state:} & (\emptyset, \emptyset, \emptyset)_{max} \\
Decide'_{max}: & (\emptyset, \emptyset, c^d)_{max} \\
Propagate'_{max}: & (\emptyset, \emptyset, c^d \neg b \neg d)_{max} \\
Decide'_{max}: & (\emptyset, \emptyset, c^d \neg b \neg d a^d)_{max} \\
Succeed_{max}: & (\emptyset, \emptyset, c^d \neg b \neg d a^d)_{out} \\
Succeed_{out}: & (\{\{a, c\}\}, \emptyset, c^d \neg b \neg d a^d \bot)_{max} \\
Backtrack'_{max}: & (\{\{a, c\}\}, \emptyset, c^d \neg b \neg d \neg a)_{max} \\
Succeed_{max}: & (\{\{a, c\}\}, \emptyset, c^d \neg b \neg d \neg a)_{out} \\
Fail_{out}: & (\{\{a, c\}\}, \emptyset, c^d \neg b \neg d \neg a \bot)_{max} \\
Backtrack'_{max}: & (\{\{a, c\}\}, \emptyset, \neg c)_{max} \\
Decide'_{max}: & (\{\{a, c\}\}, \emptyset, \neg c a^d)_{max} \\
Propagate'_{max}: & (\{\{a, c\}\}, \emptyset, \neg c a^d \neg b)_{max} \\
Decide'_{max}: & (\{\{a, c\}\}, \emptyset, \neg c a^d \neg b d^d)_{max} \\
Succeed_{max}: & (\{\{a, c\}\}, \emptyset, \neg c a^d \neg b d^d)_{out} \\
Succeed_{out}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c a^d \neg b d^d \bot)_{max} \\
Backtrack'_{max}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c a^d \neg b \neg d)_{max} \\
Succeed_{max}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c a^d \neg b \neg d)_{out} \\
Fail_{out}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c a^d \neg b \neg d \bot)_{max} \\
Backtrack'_{max}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c \neg a)_{max} \\
Decide'_{max}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c \neg a b^d)_{max} \\
Propagate'_{max}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c \neg a b^d \neg d)_{max} \\
Succeed_{max}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c \neg a b^d \neg d)_{out} \\
Fail_{out}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c \neg a b^d \neg d \bot)_{max} \\
Backtrack'_{max}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c \neg a \neg b)_{max} \\
Decide'_{max}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c \neg a \neg b d^d)_{max} \\
Succeed_{max}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c \neg a \neg b d^d)_{out} \\
Fail_{out}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c \neg a \neg b d^d \bot)_{max} \\
Backtrack'_{max}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c \neg a \neg b \neg d)_{max} \\
Succeed_{max}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c \neg a \neg b \neg d)_{out} \\
Fail_{out}: & (\{\{a, c\}, \{a, d\}\}, \emptyset, \neg c \neg a \neg b \neg d \bot)_{max} \\
Fail_{max}: & Ok(\{\{a, d\}, \{a, c\}\})
\end{array}
$$

Fig. 10. Path in DIRECT^F where F is the AF from Fig. 1.

this algorithm by modification of the previous two solving procedures, by explicitly viewing it in the light of a backtrack-search process in a search space, more similar to a SAT-based procedure. This would not be obvious by considering, e.g. the pseudo-code description of the direct approach.

Example 6. A possible path in the graph DIRECT^F for the AF F in Fig. 1 is shown in Fig. 10. One difference can be seen by the fact that the result of the modified oracle rules may be contained in an already found preferred extension. Then $\bot$ is added to the current record by $Fail_{out}$, followed by backtracking to the last decision literal, if any. Moreover note that in Fig. 10 we explicitly write the state transitions due to modified oracle rules, in order to emphasize the difference to the SAT oracle rules used in the previous graphs.

We give the correctness statement of the abstract solver representing the direct approach after providing an intermediate lemma; proofs can again be found in the Appendix.

Lemma 3.7. *For any AF F, if the rule Succeed$_{out}$ is applied from state $(\epsilon, \emptyset, E)_{out}$ in the graph* DIRECT^F *then $e(E) \in prf(F)$ and $e(E) \notin \epsilon$.*

Theorem 3.8. *For any AF F, the graph* DIRECT^F *is finite, acyclic and the only terminal state reachable from its initial state is $Ok(\epsilon)$ where $\epsilon = prf(F)$.*

3.4. Combining algorithms

We now use the insights gained by the graph representations of the algorithms from the literature to define a new algorithm for skeptical acceptance w.r.t. preferred semantics. We do so by defining a new abstract solver which incorporates the modified DPLL (cf. Section 2.3) into the graph representation of CEGARTIX (cf. Section 3.2). This gives rise to a new algorithm which is not only of theoretical interest, but also leads to a more efficient solving procedure, as we will show in Section 4.

Now recall that, in PrefSat and CEGARTIX, the two SAT calls annotated with $i \in \{base, max\}$ basically amount to finding a maximal satisfying assignment, i.e. a preferred extension. This is done by iteratively extending the satisfying assignment found by the SAT call annotated with *base* (which must not contain the queried argument), by means of a series of further SAT calls annotated with *max*.

But, given our result in Section 2.3, the "inner loop" of SAT calls for maximization is not strictly needed, and the two types of SAT calls can be substituted by a single modified call. More specifically, we replace *base* by *base*$'$ by abstaining from the condition that the queried argument must not be contained in the σ-extension. Hence, a single modified SAT call to *base*$'$ returns a preferred extension which has not been found already.

Given an AF F, an argument α and a base semantics $\sigma \in \{adm, com\}$, the graph MIX-PRF$^{F,\alpha}_{f^{\sigma}_{base'}}$ representing the new algorithm for deciding skeptical acceptance of α in F w.r.t. *prf* is defined by the states of $atoms(F)$ and the transition rules presented in Fig. 11. Its initial state is $(\emptyset, \emptyset, \emptyset)_{base'}$. As we can see, the graph describes exactly the intuition behind the new proposal. A new label *base*$'$ is employed to clearly differentiate with the other two-level architectures. Of course, in order to guarantee that the outcome of the modified SAT call is a preferred extension, we must assume the function $f^{\sigma}_{base'}(\epsilon, E, F, \alpha)$ such that:

$$\big\{e(M) \mid M \in \mathrm{Sat}\big(f^{\sigma}_{base'}(\epsilon, E, F, \alpha)\big)\big\} = \big\{S \in \sigma(F) \mid \neg\exists S' \in \epsilon : S \subseteq S'\big\}.$$

Then, the outcome of the *base*$'$ rules is treated similarly, through the *out* rules, to the graph SKEPT-PRF$^{F,\alpha}_{f}$ presented in Section 3.2.

Considering the fact that the new solution always adds positive atoms to the current assignment, it looks similar to the direct approach; but there is a notable difference between the new algorithm and the direct approach. The outcome of the oracle-rules of the direct approach (cf. Fig. 9) is a conflict-free set which is not necessarily maximal (and in other rules admissibility and maximality is checked), whereas the outcome of the oracle-rules in the new algorithm modifying SKEPT-PRF$^{F,\alpha}_{f}$ is guaranteed to be an admissible (and preferred) set.

From Theorem 2.2 we know that the *base*$'$-oracle rules give a maximal satisfying assignment of $f^{\sigma}_{base'}(\epsilon, E', F, \alpha)$:

Lemma 3.9. *For any AF $F = (A, R)$, argument $\alpha \in A$, and $\sigma \in \{adm, com\}$, if Succeed$_{base'}$ is applied from state $(\epsilon, E', E)_{base'}$ in the graph* MIX-PRF$^{F,\alpha}_{f^{\sigma}_{base'}}$ *then $E \in \mathrm{Sat}(f^{\sigma}_{base'}(\epsilon, E', F, \alpha))$ and $\neg\exists M \in \mathrm{Sat}(f^{\sigma}_{max}(\epsilon, E', F, \alpha))$ with $M \supset E$; if Fail$_{base'}$ is applied then $f^{\sigma}_{base'}(\epsilon, E', F, \alpha)$ is unsatisfiable.*

$base'$-oracle rules

$Backtrack_{base'}$	$(\epsilon, \emptyset, El^d E'')_{base'}$	$\Rightarrow (\epsilon, \emptyset, E\bar{l})_{base'}$	if $\begin{cases} El^d E'' \text{ is inconsistent and} \\ E'' \text{ contains no decision literal} \end{cases}$
$UnitPropagate_{base'}$	$(\epsilon, \emptyset, E)_{base'}$	$\Rightarrow (\epsilon, \emptyset, El)_{base'}$	if $\begin{cases} l \text{ does not occur in } E \text{ and} \\ C \vee l \text{ is a clause in } f^{\sigma}_{base'}(\epsilon, E', F, \alpha) \text{ and} \\ \text{all the literals of } \overline{C} \text{ occur in } E \end{cases}$
$Decide^{\prec}_{base'}$	$(\epsilon, \emptyset, E)_{base'}$	$\Rightarrow \left(\epsilon, \emptyset, Ea^d\right)_{base'}$	if $\begin{cases} E \text{ is consistent and} \\ \text{neither } a \text{ nor } \neg a \text{ occur in } E \end{cases}$

Failing rules

$Fail_{base'}$	$(\epsilon, \emptyset, E)_{base'}$	$\Rightarrow Accept$	if $\{ E$ is inconsistent and decision-free
$Fail_{out}$	$\left(\epsilon, E', E\right)_{out}$	$\Rightarrow \left(\epsilon \cup \{e(E')\}, \emptyset, \emptyset\right)_{base'}$	if $\{ \alpha \in e(E')$

Succeeding rules

$Succeed_{base'}$	$(\epsilon, \emptyset, E)_{base'}$	$\Rightarrow (\epsilon, E, \emptyset)_{out}$	if $\{$ no other rule applies
$Succeed_{out}$	$\left(\epsilon, E', E\right)_{out}$	$\Rightarrow Reject$	if $\{ \alpha \notin e(E')$

Fig. 11. The rules of MIX-PRF$^{F,\alpha}_{\overline{f}}$.

To be sure that maximal satisfying assignments correspond to preferred extensions, it has to hold that the atoms occurring in $f^{\sigma}_{base'}$ which do not correspond to arguments of the AF do not affect maximality. To this end we make the following assumption.

Assumption 1. Given an AF $F = (A, R)$, a set of sets of arguments ϵ, a record E relative to $atoms(F)$, and an argument $\alpha \in A$, for each $M_1, M_2 \in \text{Sat}(f^{\sigma}_{base}(\epsilon, E, F, \alpha))$, where $\sigma \in \{adm, com\}$, it holds that $M_1 \subseteq M_2$ iff $e(M_1) \subseteq e(M_2)$.

It is important to note that the concrete formulas used in CEGARTIX fulfill Assumption 1. Taking the assumption for granted in the rest of the paper, we are able to show correctness of the abstract solver representing the combined approach.

Lemma 3.10. *For any AF F, if the rule $Fail_{out}$ is applied from state $(\epsilon, E', E)_{out}$ in the graph* MIX-PRF$^{F,\alpha}_{f^{\sigma}_{base'}}$ *with $\sigma \in \{adm, com\}$ then $e(E') \in prf(F)$ and $e(E') \notin \epsilon$.*

Theorem 3.11. *For any AF $F = (A, R)$, argument $\alpha \in A$, and $\sigma \in \{adm, com\}$, the graph* MIX-PRF$^{F,\alpha}_{f^{\sigma}_{base'}}$ *is finite, acyclic and any terminal state reachable from the initial state is either Accept or Reject; Reject is reachable iff α is not skeptically accepted in F w.r.t. prf.*

Proof. (1) MIX-PRF$^{F,\alpha}_{f^{\sigma}_{base'}}$ is finite and acyclic: Finiteness follows in the same way as in Theorem 3.3. In order to show acyclicity we show that each transition rule is increasing w.r.t. the strict partial order $<$ from Definition 3 (with *base* replaced by $base'$). Consider two states $\varsigma_1 = (\epsilon_1, E'_1, E_1)_{i_1}$ and $\varsigma_2 = (\epsilon_2, E'_2, E_2)_{i_2}$ representing the states before and after the application of a rule. First of all, the $base'$-oracle rules (i.e. $Backtrack_{base'}$, $UnitPropagate_{base'}$, and $Decide^{\prec}_{base'}$) fulfill $\varsigma_1 < \varsigma_2$ because of (iv). For all of these rules $\epsilon_1 = \epsilon_2$, $E'_1 = E'_2$ and $i_1 = i_2$, but $s(E_1)$ is lexicographically smaller than $s(E_2)$, therefore $E_1 <_E E_2$. Moreover, $Fail_{out}$ fulfills $\varsigma_1 < \varsigma_2$ due to (i) since $e(E'_1) \notin \epsilon_1$ by Lemma 3.10.

$Succeed_{base}$ guarantees $\varsigma_1 < \varsigma_2$ because of (ii). Finally, $Succeed_{base'}$ fulfills $\varsigma_1 < \varsigma_2$ due to (ii), since $base <_i out$, and $Succeed_{out}$ and $Fail_{base'}$ result in terminal states. Therefore, by transitivity of $<$, or any two states ς_1 and ς_n such that ς_n is reachable from ς_1 in $\text{ENUM}^F_{\bar{f}}$ it holds that $\varsigma_1 < \varsigma_n$, showing that the graph is acyclic.

(2) Any terminal state of $\text{MIX-PRF}^{F,\alpha}_{f^\sigma_{base'}}$ reachable from the initial state is either *Reject* or *Accept*: This is immediate by the existence of the rule $Succeed_{base'}$ with condition "no other rule applied" and the fact that the rules $Fail_{out}$ and $Succeed_{out}$ are complete in the sense that if one rule does not apply the other rule applies and vice versa.

(3) *Reject* is reachable from the initial state iff α is not skeptically accepted by F w.r.t. *prf*.

($\Rightarrow$): Assume *Reject* is reachable. Hence also $(\epsilon, E', E)_{out}$ with $\alpha \notin e(E')$ is reachable. Moreover $Succeed_{base'}$ was applied at a state $(\epsilon, E'', E')_{base'}$, meaning, by Lemma 3.9, that $E' \in \text{Sat}(f^\sigma_{base'}(\epsilon, E'', F, \alpha))$ and $\neg\exists E''' \in \text{Sat}(f^\sigma_{max}(\epsilon, E'', F, \alpha))$ with $E''' \supset E'$. Taking into account Assumption 1 this means, by the definition of $f^\sigma_{base'}$, that $e(E')$ is a $\subseteq$-maximal σ-extension, i.e. a preferred extension. Since $\alpha \notin e(E')$ we get that α is not skeptically accepted.

($\Leftarrow$): Assume α is not skeptically accepted by F w.r.t. *prf*. Hence there is some $T \in prf(F)$ with $\alpha \notin T$. Now assume, towards a contradiction, that *Reject* is not reachable. This means by (1) and (2), that *Accept* is reachable. Hence $Fail_{base'}$ is applicable from a state $(\epsilon, E', E)_{base'}$. By the definition of $f^\sigma_{base'}$ and Lemma 3.9, this means that there is no σ-extension S of F such that $\neg\exists S' \in \epsilon : S \subseteq S'$. Now note that $Fail_{out}$ is the only rule where elements are added to ϵ. Moreover, by Lemma 3.10, we know that elements added are preferred extensions of F. But therefore for each $S \in \sigma(F)$ it holds that $\exists T \in prf(F) : S \subseteq T \wedge \alpha \in T$, a contradiction. □

Again it is important to note that from Theorem 3.11 it follows that *Accept* is reachable from the initial state if and only if α is skeptically accepted by F, which completes the correctness statement for $\text{MIX-PRF}^{F,\alpha}_{f^\sigma_{base'}}$.

Example 7. For the AF F from Fig. 1, a possible path in $\text{MIX-PRF}^{F,c}_{f^\sigma_{base'}}$ is depicted in Fig. 12. It correctly results in *Reject*, as c is not skeptically accepted in F w.r.t. *prf*. Figure 13, on the other hand, shows a possible path in $\text{MIX-PRF}^{F,a}_{f^\sigma_{base'}}$. Note that both paths are shorter than the ones of $\text{SKEPT-PRF}^{F,\alpha}_{\bar{f}}$ in Figs 7 and 8, respectively.

Of course, in principle it is not clear whether the new abstract solution leads to computational advantages (see, e.g. [30,31] for a related discussion); however, the experimental analysis in Section 4 shows that this is the case.

Initial state: $(\emptyset, \emptyset, \emptyset)_{base'}$
$base'$-oracle: $(\emptyset, \emptyset, E_1 \supseteq \{a, \neg b, c, \neg d\})_{base'}$
$Succeed_{base'}$: $(\emptyset, E_1, \emptyset)_{out}$
$Fail_{out}$: $(\{\{a, c\}\}, \emptyset, \emptyset)_{base'}$
$base'$-oracle: $(\{\{a, c\}\}, \emptyset, E_2 \supseteq \{a, \neg b, \neg c, d\})_{base'}$
$Succeed_{base'}$: $(\{\{a, c\}\}, E_2, \emptyset)_{out}$
$Succeed_{out}$: *Reject*

Fig. 12. Reject-path for argument c in $\text{MIX-PRF}^{F,c}_{f^\sigma_{base'}}$.

Initial state: $(\emptyset, \emptyset, \emptyset)_{base'}$
$base'$-oracle: $(\emptyset, \emptyset, E_1 \supseteq \{a, \neg b, \neg c, d\})_{base'}$
$Succeed_{base'}$: $(\emptyset, E_1, \emptyset)_{out}$
$Fail_{out}$: $(\{\{a, d\}\}, \emptyset, \emptyset)_{base'}$
$base'$-oracle: $(\{\{a, d\}\}, \emptyset, E_2 \supseteq \{a, \neg b, c, \neg d\})_{base'}$
$Succeed_{base'}$: $(\{\{a, d\}\}, E_2, \emptyset)_{out}$
$Fail_{out}$: $(\{\{a, d\}, \{a, c\}\}, \emptyset, \emptyset)_{base'}$
$base'$-oracle: $(\{\{a, d\}, \{a, c\}\}, \emptyset, unsat)_{base'}$
$Fail_{base'}$: $Accept$

Fig. 13. Accept-path for argument a in MIX-PRF$^{F,a}_{f^{\sigma}_{base'}}$.

4. Experiments

In order to test the viability of our proposed combination as introduced in Section 3.4, we have implemented an alternative version of CEGARTIX following the new approach. The choice of CEGARTIX is motivated by the fact that it has been one of the best AF solvers in both editions of the argumentation competition (http://argumentationcompetition.org). In particular, in the reasoning task of interest (skeptical acceptance under preferred semantics as in Section 3.2), CEGARTIX was 2nd out of 11 solvers entering the track in the first competition, and highly competitive also in the second event.[3]

4.1. Implementation

The main change done in CEGARTIX was thus replacing the two inner SAT calls with a single call to a SAT solver with modified heuristics, and we obtained this by changing the internal heuristics of the CLASP solver used by CEGARTIX in the 2015 competition. CLASP [29] is an ASP solver, but can act also as SAT solver with excellent results as shown in past SAT competitions, starting from 2009 to the most recent editions (see, e.g. http://www.satcompetition.org/). Moreover, for efficiency reasons, the implementation of CEGARTIX slightly differs from the algorithm in Section 3.2, given that the condition $\alpha \notin S$ is conjunctively added to $e(E) \subset S$ for $f^{\sigma}_{max}(\epsilon, E, F, \alpha)$, with the idea of guiding the search through counterexamples. Consequently, for comparing the two alternatives on the same implementation basis, the same is done for $f^{\sigma}_{base'}(\epsilon, E, F, \alpha)$.

The variants of CEGARTIX considered in our experiments are:

(1) **ceg**: version with *com* as a base semantics, which was the setting employed in both editions of the competition. Past experiments ([25], on different benchmark graphs) overall showed similar or better performance of this version compared to that with *adm* as a base semantics.
(2) **ceg+-com**: new version implementing the combination in Section 3.4 with *com* as a base semantics.
(3) **ceg+-adm**: new version implementing the combination in Section 3.4 with *adm* as a base semantics.

In our experiments, these three variants of CEGARTIX were run with the same parameters.

The version of CEGARTIX entering the competition included shortcuts, i.e. specific conditions that can lead the solver to find solutions earlier, before entering the main solving algorithm. Details for shortcuts will be presented in the next section. For this analysis, given the main goal is to test the new solution

[3] Comparisons of the performance of CEGARTIX and other solvers can be found in [18,50].

which applies to the core part of the algorithm and shortcuts could obfuscate the differences between algorithms, shortcuts have been disabled. Note, however, that the new variants can make use of the very same shortcuts. Therefore, it has to be noted that final implementations of the respective algorithms might show smaller gaps in performance, since they will all use the same shortcuts in the first place.

4.2. Benchmarks

For benchmarks, i.e., instances comprising of an AF and an argument for which one has to check skeptical acceptance under preferred semantics, we considered the following three benchmark sets[4]

- **ICCMA'15**: These are 192 AFs with three arguments to be queried for, from the competition in the year 2015 [50]. After cleaning this dataset from trivial queries (queried arguments not among the set of arguments in the AF), this resulted in 537 instances.
- **ICCMA'17**: In this dataset, from the competition in the year 2017 [27], we have 300 AFs, divided into four categories (according to expected difficulty), which were used to compare solvers for the task of checking skeptical acceptance under preferred semantics (this set is called set "A" in the competition). The hardest category has two arguments to be queried for per AF, while all other three categories have one query argument. This results in 350 instances.
- **CLIMA**: This is a set of AFs from our earlier work [53], which we included since it comprises of several AFs that were hard to solve by an earlier version of CEGARTIX. Here we have 320 AFs and one query argument per AF. The AFs $F = (A, R)$ were created as randomly generated digraphs, with a fixed set of arguments $A \in \{100, 150, 200, 225, 250, 300, 325, 350\}$ and a probability to include an attack (a, b) for each $a, b \in A$ with probability $p \in \{0.1, 0.2, 0.3, 0.4\}$. For each parameter choice 10 AFs were created.

For each of the three benchmark sets, the AFs were given in the original dataset, while the arguments to be queried for were only given for **ICCMA'15** and **ICCMA'17**. For the set **CLIMA**, we chose one argument in the AF to query at random, with uniform probability for each argument. Overall, this resulted in 1207 instances (AF and query).

4.3. Results

Experiments have been run on an AMD Opteron Processor 6308 3.5 GHz with 2 processors with each 2 physical cores; every of these cores puts at disposal 2 logical cores, for a sum of 192 GB (12×16 GB) of RAM. In our experiments we set a per-instance timeout of 600 sec.

We first show general runtime statistics in Table 1. More concretely, the table depicts median runtimes over the considered benchmark sets, as well as timeouts encountered in the runs. The last two columns list the number of instances that were uniquely solved by **ceg** or by the union of solved instances of **ceg+-com** and **ceg+-adm**, i.e., whether the original approach or the new approaches could contribute to uniquely solved instances.

This table indicates that, regarding median runtime and timeouts, the new approaches generally do not fare (much) better than the original version of CEGARTIX. In fact, median runtimes and timeouts overall increased when comparing new and original approaches, except for median running time of **ceg+-com** on

[4]Archives of the benchmark sets can be found at http://argumentationcompetition.org/2015/iccma2015_benchmarks.zip, http://www.dbai.tuwien.ac.at/iccma17/benchmarks/A.tar.gz, and http://www.dbai.tuwien.ac.at/research/project/argumentation/cegartix/files/clima.zip.

Table 1
General runtime statistics from experiments

	Median runtime			Timeouts			Uniquely solved	
Benchmark	**ceg**	**ceg+-com**	**ceg+-adm**	**ceg**	**ceg+-com**	**ceg+-adm**	by **ceg**	by **ceg+-com** or **ceg+-adm**
All	1.01	1.14	1.28	57	61	71	6	11
ICCMA'15	0.77	0.93	0.76	0	0	0	0	0
ICCMA'17	9.75	7.96	23.30	45	47	60	5	8
CLIMA	1.04	1.1	0.99	12	14	11	1	3

benchmark **ICCMA'17** and, to a small extent, **ceg+-adm** on benchmark **CLIMA**. A further observation is that the instances from benchmark **ICCMA'15** are rather easy to solve.

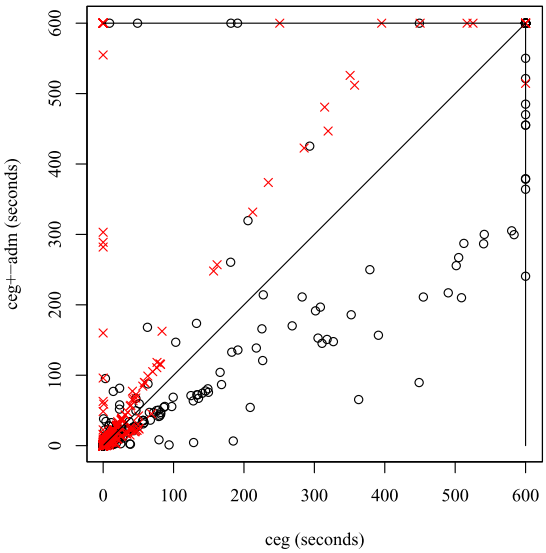
Nevertheless, the uniquely solved instances indicate differences of the approaches w.r.t. runtime performance. Looking closer at these uniquely solved instances, when **ceg+-com** or **ceg+-adm** could solve an instance within the timeout limit and **ceg** could not, it was always the case that **ceg+-adm** solved the instance, while this was only sometimes the case for **ceg+-com**. That is, **ceg+-adm** contributes to all of the uniquely solved instances, while **ceg+-com** only to five of the eleven instances.

We next illustrate, via Fig. 14, the different runtime behaviors of the three CEGARTIX implementations via scatter plots. In Fig. 14(a), the scatter plot between **ceg** and **ceg+-adm** is shown, while in Fig. 14(b), **ceg** is compared to **ceg+-com** and, finally, in Fig. 14(c), the scatter plot of the two new solvers is shown. Such plots show the running time of two solvers (on x and y axes) on each individual instance. A runtime directly on the diagonal implies the same runtime for both solvers on that instance.

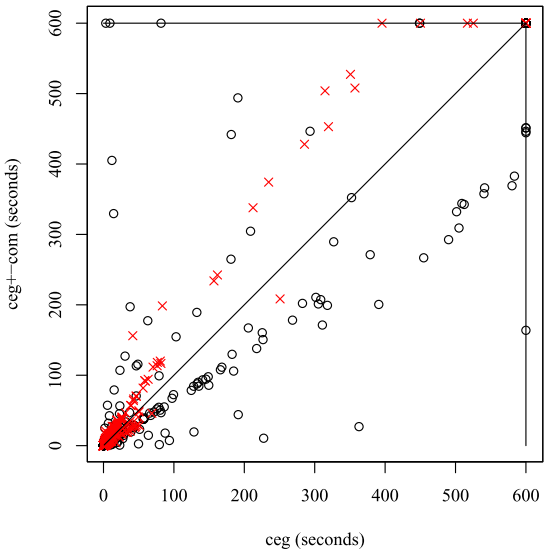
Closer inspection of the figures suggests that the solver **ceg** and the two solvers **ceg+-com** and **ceg+-adm** are rather complementary in their runtime behavior on many (non-easy) instances. That is, apart from the uniquely solved instances (these are the ones on the "timeout" lines for one of the solvers), also several further instances showed different runtime behavior: in both Fig. 14(a) and Fig. 14(b) several instances can be seen below or above the diagonal.

We hypothesize that a reason for the different runtimes, for original **ceg** and novel **ceg+-com** and **ceg+-adm**, stems from difficulties of **ceg** to find non-trivial (i.e., non-empty) admissible sets. To investigate towards this end, we have marked each instance of each scatter plot, in Figs 14(a)–(c), whether the corresponding AF has a non-empty grounded extension or not. When an AF has an empty grounded extension the corresponding symbol in the figure is a black circle, otherwise a red cross. An AF having a non-empty grounded extension can be seen as a kind of approximation of whether one can (easily) find a non-trivial admissible set. In Fig. 14(a) and Fig. 14(b), this categorization of the instances is, to some extent, reflected in the runtimes: many times when a novel solver outperformed **ceg** it is the case when the grounded extension is empty. When looking at Fig. 14(c), comparing running times of **ceg+-com** and **ceg+-adm**, the results suggest that on AFs with an empty grounded extension, **ceg+-adm** tends to be better w.r.t. running time, yet on AFs with a non-empty grounded extension, many instances, on that figure, are either in the diagonal or, in fact, trivial for **ceg+-com**, but not for **ceg+-adm**.

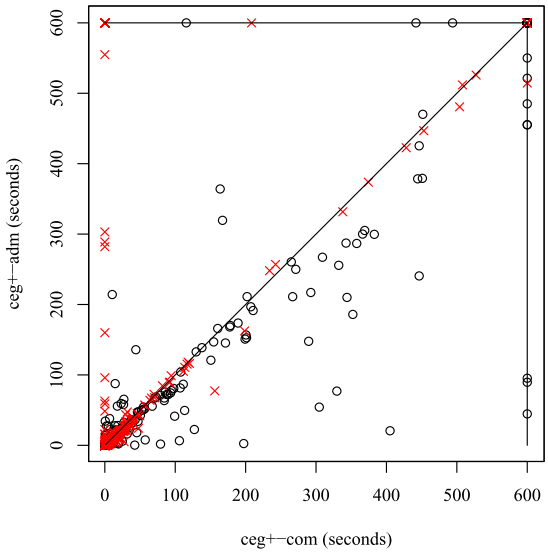
Although further research is needed, the characteristic of an AF having a (non-)empty grounded extension gives an indicator whether **ceg** or **ceg+-com** and **ceg+-adm** might be better to use for solving. This insight can be used, for instance, when compiling an algorithm selection for CEGARTIX, in line with techniques studied in [15], to first compute the grounded extension, and then choose which heuristic to apply.



(a) **ceg** vs. **ceg+-adm**.



(b) **ceg** vs. **ceg+-com**.



(c) **ceg+-com** vs. **ceg+-adm**.

Fig. 14. Scatter plots of our experimental analysis. Black circles indicate instances with an AF that has an empty grounded extension, while a red cross indicates a non-empty grounded extension.

5. Extensions to the framework

In Section 3 we have analyzed three prominent algorithms from the literature dealing with preferred semantics. In this section we show that the modularity of the abstract solver approach allows us to give the graph representation of related algorithms with little effort. First, we abstract the algorithms from

Table 2
Complexity of decision problems for AFs

σ	Cred_σ	Skept_σ
prf	NP-c	Π_2^P-c
sem	Σ_2^P-c	Π_2^P-c
stg	Σ_2^P-c	Π_2^P-c

[25] deciding skeptical (resp. credulous) acceptance w.r.t. other, namely stage [52] and semi-stable [11], semantics, and then we exemplify how to incorporate shortcuts into the graph-representations for the algorithms of the same paper.

5.1. Core algorithms for semi-stable and stage semantics

Other semantics involving reasoning tasks lying in the second level of the polynomial hierarchy are stage and semi-stable (cf. Table 2). Their definitions make use of the concept of the range of a set of arguments $S \subseteq A$ in an AF $F = (A, R)$, defined as $S_F^+ = S \cup \{a \in A \mid \exists b \in S : (b, a) \in R\}$, i.e. S together with all arguments it attacks. Stage (*stg*) and semi-stable (*sem*) semantics are then defined as follows:

Definition 4. Given an AF F,

- $S \in stg(F)$, if $S \in cf(F)$ and there is no $T \in cf(F)$ such that $T_F^+ \supset S_F^+$;
- $S \in sem(F)$, if $S \in adm(F)$ and there is no $T \in adm(F)$ such that $T_F^+ \supset S_F^+$, or equivalently,
- $S \in sem(F)$, if $S \in com(F)$ and there is no $T \in com(F)$ such that $T_F^+ \supset S_F^+$.

For semi-stable semantics the possible base semantics coincide with the ones for preferred semantics, that is admissible and complete, while stage semantics (yielding range-maximal conflict-free sets) uses conflict-free as base semantics. In other words, for the pairs $(\sigma, \theta) \in \{(adm, sem), (com, sem), (cf, stg)\}$, a uniform characterization of θ is as follows: Given an AF F, $S \in \theta(F)$ iff $S \in \sigma(F)$ and there is no $T \in \sigma(F)$ such that $T_F^+ \supset S_F^+$.

Algorithms for skeptical (resp. credulous) acceptance w.r.t. these semantics are presented in Algorithms 2 and 3 of [25] by adaptation of the algorithm for skeptical acceptance w.r.t. preferred semantics described in Section 3.2. Similar to the algorithm for preferred semantics, the general skeptical acceptance procedure for semantics θ and base semantics σ first makes use of two SAT oracles to find a range-maximal σ-extension. The main difference to the algorithm for preferred semantics is that the maximization is concerned with the range of extensions instead of the extensions themselves. Moreover, since there can be different σ-extensions with the same range, another oracle has to be consulted in order to check whether there is a σ-extension with a range equal to the maximal one found before, which does not contain the queried argument. If such an extension exists, the algorithm returns with a negative answer to the skeptical acceptance problem w.r.t. θ. For credulous acceptance, the algorithm returns with a positive answer if the oracle call finds such a σ-extension which *does* contain the queried argument.

The graph $\textsc{Skept}\text{-}\theta_{\overline{f}}^{F,\alpha}$ for a semantics $\theta \in \{sem, stg\}$, an AF F, and argument α, and a vector of functions $\overline{f}$ is defined by states over $atoms(F)$ and the transition rules of $\textsc{Skept-prf}_{\overline{f}}^{F,\alpha}$ (Figs 4 and 6) with additional oracle rules for index *out* (i.e. we have $i \in \{base, max, out\}$ for $Backtrack_i$, $UnitPropagate_i$, and $Decide_i$ now) and the rules $Fail_{out}$ and $Succeed_{out}$ changed according to Fig. 15. In contrast to the

Failing rules

$Fail_{out}$ $\quad (\epsilon, E', E)_{out} \Rightarrow (\epsilon \cup \{(e(E'))^+_F\}, \emptyset, \emptyset)_{base}$ if $\{$ E is inconsistent and decision-free

Succeeding rules

$Succeed_{out}$ $\quad \big(\epsilon, E', E\big)_{out} \Rightarrow Reject$ $\quad$ if $\{$ no other rule applies

Fig. 15. The transition rules of the graph SKEPT-$\theta^{F,\alpha}_{\overline{f}}$ that differ from SKEPT-PRF$^{F,\alpha}_{\overline{f}}$.

graphs presented so far, ϵ now contains the ranges of the extensions already found. Moreover, the decision whether to add the range of the current extension and continue the search or to reject the given instance is based on another set of oracle rules – the ones indexed by *out*.

The initial state of SKEPT-$\theta^{F,\alpha}_{\overline{f}}$ is $(\emptyset, \emptyset, \emptyset)_{base}$. For $\sigma \in \{adm, com, cf\}$ we assume functions f^{σ}_{base}, f^{σ}_{max} and f^{σ}_{out} such that:

$$\big\{e(M) \mid M \in \mathrm{Sat}\big(f^{\sigma}_{base}(\epsilon, E, F, \alpha)\big)\big\} = \big\{S \in \sigma(F) \mid \alpha \notin S \wedge \neg\exists S' \in \epsilon : S^+_F \subseteq S'\big\};$$

$$\big\{e(M) \mid M \in \mathrm{Sat}\big(f^{\sigma}_{max}(\epsilon, E, F, \alpha)\big)\big\} = \big\{S \in \sigma(F) \mid \big(e(E)\big)^+_F \subset S^+_F\big\};$$

$$\big\{e(M) \mid M \in \mathrm{Sat}\big(f^{\sigma}_{out}(\epsilon, E, F, \alpha)\big)\big\} = \big\{S \in \sigma(F) \mid \big(e(E)\big)^+_F = S^+_F \wedge \alpha \notin S\big\}.$$

Functions f^{σ}_{base} and f^{σ}_{max} coincide with the ones for preferred semantics, except that they compare ranges of extensions. The new function f^{σ}_{out} does the additional check described above.

Likewise, the graph CRED-$\theta^{F,\alpha}_{\overline{f}}$ abstracting the CEGARTIX-algorithm for credulous acceptance w.r.t. semi-stable and stage semantics coincides with SKEPT-$\theta^{F,\alpha}_{\overline{f}}$ with the exception that the outcomes of the rules $Fail_{base}$ and $Succeed_{out}$ are swapped, i.e. the application of $Fail_{base}$ leads to *Reject* and the application of $Succeed_{out}$ leads to *Accept*. That is since a found witness (a θ-extension containing α) means that α is credulously accepted, while if exhaustive search does not reveal such a witness, it α is not credulously accepted. As the algorithm searches for extensions containing the queried argument α, two functions have to differ; we assume g^{σ}_{base} and g^{σ}_{out}, which contain the condition $\alpha \in S$ instead of $\alpha \notin S$ compared to the functions f^{θ}_{base} and f^{θ}_{out} for skeptical acceptance:

$$\big\{e(M) \mid M \in \mathrm{Sat}\big(g^{\sigma}_{base}(\epsilon, E, F, \alpha)\big)\big\} = \big\{S \in \sigma(F) \mid \alpha \in S \wedge \neg\exists S' \in \epsilon : S^+_F \subseteq S'\big\};$$

$$\big\{e(M) \mid M \in \mathrm{Sat}\big(g^{\sigma}_{out}(\epsilon, E, F, \alpha)\big)\big\} = \big\{S \in \sigma(F) \mid \big(e(E)\big)^+_F = S^+_F \wedge \alpha \in S\big\}.$$

The following results show correctness of the abstract solvers for acceptance problems w.r.t. semi-stable and stage semantics described in this section. The proofs, which follow the same structure as previous proofs, can be found in the Appendix.

Lemma 5.1. *For $(\sigma, \theta) \in \{(adm, sem), (com, sem), (cf, stg)\}$, any AF $F = (A, R)$ and an argument $\alpha \in A$, if one of the rules $Fail_{out}$ or $Succeed_{out}$ is applied from state $(\epsilon, E', E)_{out}$ in the graph* SKEPT-$\theta^{F,\alpha}_{(f^{\sigma}_{out}, f^{\sigma}_{max}, f^{\sigma}_{base})}$ *(resp.* CRED-$\theta^{F,\alpha}_{(g^{\sigma}_{out}, f^{\sigma}_{max}, g^{\sigma}_{base})}$*) then $e(E') \in \theta(F)$ and $(e(E'))^+_F \notin \epsilon$.*

Theorem 5.2. *For $(\sigma, \theta) \in \{(adm, sem), (com, sem), (cf, stg)\}$, any AF $F = (A, R)$ and $\alpha \in A$, the graph* SKEPT-$\theta^{F,\alpha}_{(f^{\sigma}_{out}, f^{\sigma}_{max}, f^{\sigma}_{base})}$ *(resp.* CRED-$\theta^{F,\alpha}_{(g^{\sigma}_{out}, f^{\sigma}_{max}, g^{\sigma}_{base})}$*) is finite, acyclic and any terminal state reachable from the initial state is either Accept or Reject; Reject is reachable from the initial state iff α is not skeptically accepted (resp. not credulously accepted) in F w.r.t. prf.*

Failing rules
$Fail_{pre}$ $(\epsilon, E', E)_{pre} \Rightarrow (\emptyset, \emptyset, \emptyset)_{base}$ if $\{$ E is inconsistent and decision-free

Succeeding rules
$Succeed_{pre}$ $(\epsilon, E', E)_{pre} \Rightarrow Reject$ if $\{$ no other rule applies

Fig. 16. The transition rules of the graph FULL-SKEPT-PRF$^{F,\alpha}_{\overline{f}}$ added to SKEPT-PRF$^{F,\alpha}_{\overline{f}}$.

Finally note again that from Theorem 5.2 it follows that *Accept* is reachable from the initial state if and only if α is skeptically accepted (resp. credulously accepted) in F, which completes the correctness statement for SKEPT-$\theta^{F,\alpha}_{\overline{f}}$ (resp. CRED-$\theta^{F,\alpha}_{\overline{g}}$).

5.2. *Shortcuts for* CEGARTIX*-algorithms*

When defining abstract solvers for the algorithms of CEGARTIX in previous sections we restricted our attention to the core of the algorithm. In this section we show that the graphs presented so far can be easily extended in a modular way to capture the full algorithms.

We do so by abstracting the full Algorithm 1 of [25] for skeptical acceptance w.r.t. preferred semantics, including the shortcut computation performed at the beginning of the algorithm. By this shortcut, the algorithm immediately returns with a negative answer for the skeptical acceptance problem w.r.t. preferred semantics, if there is a σ-extension ($\sigma \in \{adm, com\}$) attacking the queried argument. To this end we define FULL-SKEPT-PRF$^{F,\alpha}_{\overline{f}}$ for a given AF $F = (A, R)$, an argument $\alpha \in A$ and a vector of oracle functions $\overline{f}$ as the graph SKEPT-PRF$^{F,\alpha}_{\overline{f}}$ from Section 3.2 with the following modifications:

- We add the transition rules presented in Fig. 16.
- Moreover, there is a set of oracle rules with index *pre*. For $\sigma \in \{adm, com\}$ we assume a function f^{σ}_{pre} such that

$$\{e(M) \mid M \in \mathrm{Sat}(f^{\sigma}_{pre}(\epsilon, E, F, \alpha))\} = \{S \in \sigma(F) \mid S \text{ attacks } \alpha\}.$$

- The initial state is $(\emptyset, \emptyset, \emptyset)_{pre}$.

To represent the shortcut, a third level has been added, which precedes levels *base* and *max*, so that we call this level *pre*. Note that ϵ and E' are always $\emptyset$ at level *pre*. If the oracle rules with index *pre* lead to a record corresponding to a satisfying assignment of f^{σ}_{pre} (i.e. a σ-extension attacking α), the application of $Succeed_{pre}$ leads to rejection; otherwise $Fail_{pre}$ leads to the state $(\emptyset, \emptyset, \emptyset)_{base}$, which means we have arrived at SKEPT-PRF$^{F,\alpha}_{\overline{f}}$. The resulting graph represents the full Algorithm 1 of [25].

Theorem 5.3. *For any AF $F = (A, R)$, argument $\alpha \in A$, and $\sigma \in \{adm, com\}$, the graph* FULL-SKEPT-PRF$^{F,\alpha}_{(f^{\sigma}_{base}, f^{\sigma}_{max}, f^{\sigma}_{pre})}$ *is finite, acyclic and any terminal state reachable from the initial state is either Accept or Reject; Accept is reachable iff α is skeptically accepted in F w.r.t. prf.*

6. Related work

The use of abstract solvers was initiated by Nieuwenhuis et al. [44]. In that work the authors first presented an abstract solver for SAT, similar to our introduction of abstract solvers in Section 2.2. Then, they

presented two extensions: (i) a description of Conflict-Driven Clause Learning SAT solving, i.e. involving backjumping and learning techniques, by means of modular addition of transition rules, but also by changing the definition of states to account for learned clauses, and (ii) they considered also Satisfiability Modulo Theories (SMT) problems with certain logics via a lazy approach [48], i.e. where the SAT calls are made to provide satisfying assignments of the Boolean abstraction of the SMT problem that are then checked for "SMT consistency". Lierler [35] imported this methodology to Answer Set Programming (ASP), by first designing abstract solvers for some backtracking-based ASP solvers for non-disjunctive ASP solving, and then enhancing her approach to include backjumping and learning techniques [36]. Another extension for describing CASP solvers, i.e. systems able to deal with a combination of ASP and constraint programming, a language useful to represent and reason on hybrid domains, has been put forward in [37]. Other works on abstract solvers are [38], where solvers for different formalisms, e.g. ASP and SAT with Inductive Definitions, are compared by means of comparison of the related graphs, and the following series of papers where, starting from a developed concept of modularity in answer set solving [39], abstract modeling of solvers for multi-logic systems are presented [40–42].

If we turn our attention to the usage of abstract solvers for dealing with reasoning tasks beyond NP, the situation is less developed and only very recently few works have been put forward. Abstract solvers for certain disjunctive answer set solvers implementing basic backtracking have been introduced by Brochenin et al. [6] and are studied in a more general way in [7]. Even more recently, abstract solvers for satisfiability of Quantified Boolean Formulas [9] and cautious reasoning in ASP [10] have been presented.

Only few of the aforementioned works [36,44] have already aimed for the implementation of combinations of algorithms based on abstract solvers; thus, our practical results are particularly remarkable.

As far as other algorithms for the preferred semantics in the literature are concerned, we mention [22, 43], where a labelling-based approach is employed. These algorithms differ in the initial labellings and how transitions are applied to argument labels. Moreover, [43] includes other semantics than preferred and also argument-based proof theories, another way to characterize an algorithm's behavior but whose goal is not to be the basis for an implementation.

The argumentation solver competition 2015 [51] had eleven participating systems in the task of deciding skeptical acceptance of an argument w.r.t. preferred semantics. The first two places were taken by ArgSemSAT and CEGARTIX, whose algorithms are described in terms of abstract solvers in Sections 3.1 and 3.2, respectively. The other solvers in the top five were LabSATSolver, ASPARTIX-V [28] and CoQuiAAS [32] (system descriptions of all participating solvers can be found in [49]). While LabSATSolver uses the same algorithm as ArgSemSAT for this particular task, ASPARTIX-V and CoQuiAAS are reduction-based approaches, using translations to ASP and a particular variant of Max-SAT, respectively. Thus, being based on reduction, their modeling via abstract solvers is less interesting for the abstract argumentation community, given that this would boil down to modeling, respectively, ASP and Max-SAT search algorithms. For this reason, they have not been considered in this paper.

7. Conclusions

In this paper we have shown the applicability and the advantages of using a rigorous formal way for describing certain algorithms for solving decision problems for abstract argumentation frameworks through graph-based abstract solvers instead of, e.g. pseudo-code-based descriptions. Both SAT-based and dedicated approaches for solving hard problems have been analyzed and compared, with focus

on algorithms for the preferred semantics. Moreover, by combining abstract solvers, we have obtained a novel algorithm for skeptical acceptance. The algorithm has been implemented taking CEGARTIX [25] as a starting point. An experimental analysis on the benchmark graphs of the first and second argumentation competition, as well as on graphs from earlier work, shows that the new algorithm is complementary to an existing algorithm in CEGARTIX. The above analysis has focused, as said, on the well-studied preferred semantics, and on core algorithms. We have then shown how the machinery can be employed to describe algorithms for different semantics, e.g. semi-stable and stage semantics, as employed in CEGARTIX, and for taking into account specific optimization techniques by means of modular addition of transition rules to the graphs describing the core parts of the algorithms.

As future work, we want to apply the concept of abstract solvers to other promising algorithms and optimization techniques for reasoning tasks in abstract argumentation. In particular, we plan to study certain approaches to the decomposition of AFs [2,16,33,34]. Moreover, we plan to extend our experimental analysis for the new version of CEGARTIX by studying on which classes of AFs the new version is performing better than existing algorithms.

Acknowledgements

We thank Benjamin Kaufmann, member of the Potassco team, for suggesting how to modify the version of CLASP employed in the experiments. This work has been funded by the Austrian Science Fund (FWF) through projects I1102, I2854 and P30168-N31, and by Academy of Finland through grants 251170 COIN and 284591.

Appendix. Proofs

Proof of Lemma 3.5. Let $S_{out} = (\epsilon, E', E)_{out}$ be the state from which $Fail_{out}$ is applied. The state S_{out} must have been achieved by the application of $Fail_{max}$ from a state $(\epsilon, E', \emptyset)_{max}$. This means, by the definition of formula f^{σ}_{max} and Lemma 3.4, that there is no $S \in \sigma(F)$ with $S \supset e(E')$. For $e(E') \in prf(F)$ it remains to show that $e(E') \in \sigma(F)$. Observe that an update of the value of E' is only done by an application of $Succeed_{base}$ or $Succeed_{max}$. In both cases $e(E')$ corresponds to a σ-extension of F, since it is a satisfying assignment of one of the formulas f^{σ}_{base} and f^{σ}_{max} and therefore guaranteed to be a σ-extension of F.

Since the initial state is $(\emptyset, \emptyset, \emptyset)_{base}$, an application of $Succeed_{base}$ must precede $Fail_{out}$. From this application of $Succeed_{base}$ it follows that there is some record E'' such that $\neg\exists S \in \epsilon : e(E'') \subseteq S$ holds. Moreover every application of $Succeed_{max}$ updates E'' by a proper superset of itself. Therefore $e(E') \supseteq e(E'')$ and also $\neg\exists S \in \epsilon : e(E') \subseteq S$, in particular $e(E') \notin \epsilon$. □

Proof of Theorem 3.6. (1) SKEPT-PRF$^{F,\alpha}_{f}$ is finite and acyclic: In order to show finiteness note that the states $(\epsilon, E', E)_i$ of SKEPT-PRF$^{F,\alpha}_{f}$ coincide with the states of ENUM$^{F}_{f}$, there is just an additional option *out* for i. Hence finiteness follows from Theorem 3.3. In order to show that SKEPT-PRF$^{F,\alpha}_{f}$ is acyclic we have to show that the rules that differ in SKEPT-PRF$^{F,\alpha}_{f}$ from ENUM$^{F}_{f}$ (i.e. the ones listed in Fig. 6) are increasing with respect to the ordering $<$ from Definition 3: $Fail_{out}$ fulfills $\varsigma_1 < \varsigma_2$ due to (i) by Lemma 3.5, $Fail_{max}$ guarantees $\varsigma_1 < \varsigma_2$ because of (ii), and $Fail_{base}$ and $Succeed_{out}$ end in terminal states.

(2) Any terminal state of $\text{SKEPT-PRF}_{\overline{f}}^{F,\alpha}$ reachable from the initial state is either *Reject* or *Accept*: Consider the state $\varsigma = (\epsilon, E, E')_i$. If $i \in \{base, max\}$ then there is a rule $Succeed_i$ with the condition "no other rule applies", hence ς cannot be a terminal state. If $i = out$, the rules $Fail_{out}$ and $Succeed_{out}$ are complete in the sense that if one rule does not apply the other rule applies and vice versa. Therefore only *Reject* and *Accept* can be terminal states.

(3) *Reject* is reachable from the initial state iff α is not skeptically accepted in F w.r.t. *prf*: ($\Rightarrow$): Assume *Reject* is reachable. Hence also $(\epsilon, E', E)_{out}$ with $\alpha \notin e(E')$ is reachable. Moreover $Fail_{max}$ was applied at a state $(\epsilon, E', E'')_{max}$, meaning, by Lemma 3.4, that $f^{\sigma}_{max}(\epsilon, E', F, \alpha)$ is unsatisfiable, i.e. there is no σ-extension S with $S \supset e(E')$. It remains to show that $e(E') \in \sigma(F)$. That is by the fact that there must be a preceding application of the rule $Succeed_{base}$ from some state $(\epsilon, E''', E')_{base}$ with $e(E')$ being a σ-extension of F by the definition of f^{σ}_{base} and Lemma 3.4. Now as $e(E') \in \sigma(F)$, $\neg\exists S \supset e(E') : S \in \sigma(F)$, and $\alpha \notin e(E')$, we have that α is not skeptically accepted by F w.r.t. *prf*. ($\Leftarrow$): Assume α is not skeptically accepted by F w.r.t. *prf*. Hence there is some $T \in prf(F)$ with $\alpha \notin T$. Now assume, towards a contradiction, that *Reject* is not reachable. This means by (1) and (2), that *Accept* is reachable. Hence $Fail_{base}$ is applicable from a state $(\epsilon, E', E)_{base}$. By the definition of f^{σ}_{base} and Lemma 3.4, this means that there is no σ-extension S of F with $\alpha \notin S$ and $\neg\exists S' \in \epsilon : S \subseteq S'$. Now note that $Fail_{out}$ is the only rule where elements are added to ϵ. Moreover, by Lemma 3.5, we know that elements added are preferred extensions of F. But therefore for each $S \in \sigma(F)$ with $\alpha \notin S$ it holds that $\exists T \in prf(F) : S \subseteq T \wedge \alpha \in T$, a contradiction. □

Proof of Lemma 3.7. The application of $Succeed_{out}$ from state $\varsigma_{out} = (\epsilon, \emptyset, E)_{out}$ must have been preceded by $Succeed_{max}$ from the state $\varsigma_{max} = (\epsilon, \emptyset, E)_{max}$ which only differs from ς_{out} in i. We now analyze the record E as it is constructed by the rules $Decide'_{max}$, $Propagate'_{max}$ and $Backtrack'_{max}$. The application of $Decide'_{max}$ adds literal a, literal $\neg b$ is added by $Propagate'_{max}$ for all b being in conflict with a in F. Therefore $e(E)$ is conflict-free in F. Moreover $e(E)$ is admissible since if "there is an argument α such that $e(E)$ does not attack α and α attacks $e(E)$", then $Fail_{out}$ is applied instead of $Succeed_{out}$. To get $e(E) \in prf(F)$ it remains to show that there is no $S \in adm(F)$ with $S \supset e(E)$. Assume there is such an $S \in adm(F)$. Then there must be some $a \in S$ with $a \notin e(E)$. Now observe that the graph first adds a to the record and afterwards $\neg a$. Therefore S must have been discovered in advance. But then $\exists S \in \epsilon : e(E) \subseteq S$, hence $Fail_{out}$ is applied instead of $Succeed_{out}$. □

Proof of Theorem 3.8. Since states of DIRECT^F consist of the same elements as states of $\text{ENUM}_{\overline{f}}^F$, finiteness of DIRECT^F follows in the same way as in Theorem 3.3.

To show that DIRECT^F is acyclic we will, again as in the proof of Theorem 3.3, show that each transition rule of DIRECT^F is increasing w.r.t. a strict partial order on states. To this end we define the strict partial order $<_D$ such that for any two states $\varsigma_1 = (\epsilon_1, \emptyset, E_1)_{i_1}$ and $\varsigma_2 = (\epsilon_2, \emptyset, E_2)_{i_2}$, $\varsigma_1 <_D \varsigma_2$ iff

(i) $\epsilon_1 <_\epsilon \epsilon_2$, or
(ii) $\epsilon_1 = \epsilon_2$ and $E_1 <_E E_2$, or
(iii) $\epsilon_1 = \epsilon_2$ and $E_1 = E_2$ and $i_1 <_i i_2$,

where $<_\epsilon$, $<_E$ and $<_i$ are the orderings from Definition 3. First of all, the oracle rules (i.e. $Backtrack'_{max}$, $UnitPropagate'_{max}$, and $Decide'_{max}$) and $Fail_{out}$ fulfill $\varsigma_1 <_D \varsigma_2$ because of (ii). For all of these rules $\epsilon_1 = \epsilon_2$, but $s(E_1)$ is lexicographically smaller than $s(E_2)$, therefore $E_1 <_E E_2$. Moreover, $Succeed_{out}$

fulfills $\varsigma_1 <_D \varsigma_2$ due to (i) since $e(E_1) \notin \epsilon_1$ by Lemma 3.7. $Succeed_{max}$ guarantees $\varsigma_1 <_D \varsigma_2$ because of (iii).

The only terminal state reachable from the initial state is $Ok(\epsilon)$ since all states $(\epsilon, \emptyset, E')_i$ of DIRECT^F have $i \in \{max, out\}$ and for each $i \in \{max, out\}$ there is a rule $Succeed_i$ with the condition "no other rule applies". It remains to show that, when state $Ok(\epsilon)$ is reached, ϵ is the set of preferred extensions of F. Since elements are only added to ϵ by rule $Succeed_{out}$ we know from Lemma 3.7 that for each $T \in \epsilon$ it holds that $T \in prf(F)$. On the other hand, the oracle rules guarantee that each conflict-free set S of F a set $(\epsilon, \emptyset, E)_{out}$ with $e(E) = S$ is reached. If S is then admissible and maximal w.r.t. ϵ (which contains only preferred extensions of F as observed before), S is added to ϵ. Therefore each $T \in prf(F)$ is contained in ϵ. □

Proof of Lemma 3.10. Let $S_{out} = (\epsilon, E', E)_{out}$ be the state from which $Fail_{out}$ is applied. The state S_{out} must have been achieved by the application of $Succeed_{base'}$ from a state $(\epsilon, E'', E')_{base'}$ This means, by the definition of formula $f^{\sigma}_{base'}$, Assumption 1, and Lemma 3.9, that $e(E') \in \sigma(F)$, $\neg\exists S \in \epsilon : e(E') \subseteq S$, and $e(E')$ is maximal with these properties. Since ϵ is initially empty and, as we argue, only preferred extensions of F are added, it follows that $e(E') \in prf(F)$ and $e(E') \notin \epsilon$. □

Proof of Lemma 5.1. Let $(\epsilon, E', E_0)_{out}$ be the state from which $Fail_{out}$ or $Succeed_{out}$ is applied. Other rules of index *out* have not changed E_0, hence $(\epsilon, E', \emptyset)_{out}$ was the outcome of the application of $Fail_{max}$. By definition of f^{σ}_{max} this means that $\neg\exists S \in \sigma(F) : S^+_F \supset (e(E'))^+_F$. To get $e(E') \in \theta(F)$ it remains to show that $e(E') \in \sigma(F)$. Observe that $Succeed_{base}$ is applied at least once, since every AF has a σ-extension. Moreover, the value of E' is only updated by applications of $Succeed_{base}$ or $Succeed_{max}$. In both cases $e(E')$ corresponds to a σ-extension of F, since E' is a satisfying assignment of the formula f^{σ}_{base} or f^{σ}_{max}, respectively. Therefore $e(E') \in \sigma(F)$.

Since the initial state is $(\emptyset, \emptyset, \emptyset)_{base}$, an application of $Succeed_{base}$ must precede $Fail_{out}$. From this application of $Succeed_{base}$ it follows that there is a record E'' such that $\neg\exists S \in \epsilon : (e(E''))^+_F \subseteq S$. Moreover every application of $Succeed_{max}$ updates E'' by a proper superset of itself. At some point, $Fail_{max}$ must be applied, leading to a state $(\epsilon, E', \emptyset)_{out}$ with $E' \supseteq E''$, hence again $\neg\exists S \in \epsilon : (e(E'))^+_F \subseteq S$. Finally, oracle rules with index *out* do not change E'', hence when $Fail_{out}$ is applied from state $(\epsilon, E', E)_{out}$ it holds that $(e(E'))^+_F \notin \epsilon$. □

Proof of Theorem 5.2. We show the result for SKEPT-$\theta^{F,\alpha}_{f}$, the proof for CRED-$\theta^{F,\alpha}_{g}$ very similar.

(1) SKEPT-$\theta^{F,\alpha}_{f}$ is finite and acyclic: Finiteness is immediate by Theorem 3.6, since SKEPT-$\theta^{F,\alpha}_{f}$ is defined over the same states as SKEPT-PRF$^{F,\alpha}_{f}$ – with the only exception of containing a set of extension-ranges instead of extensions, which makes no difference in this matter. For acyclicity all rules have to be increasing w.r.t. $<$. This was already shown for the rules in Figs 4 and 6. It also follows for the oracle rules for index *out* as the fact that oracle rules are increasing w.r.t. $<$ is independent from the index. Finally, the rule $Fail_{out}$ is increasing due to condition (i) in Definition 3 by Lemma 5.1 and $Succeed_{out}$ leads to a terminal state.

(2) Any terminal state of SKEPT-$\theta^{F,\alpha}_{f}$ reachable from the initial state is either *Reject* or *Accept*: For any possible state $\varsigma = (\epsilon, E, E')_i$ with $i \in \{base, max, out\}$ there is a rule $Succeed_i$ with the condition "no other rule applies", hence ς cannot be a terminal state. Therefore only *Reject* and *Accept* can be terminal states.

(3) *Reject* is reachable from the initial state iff α is not skeptically accepted in F w.r.t. θ: ($\Rightarrow$): Assume *Reject* is reachable. It must have been reached by application of $Succeed_{out}$ from a state $(\epsilon, E', E)_{out}$. By definition of f^{σ}_{out} this means that $e(E) \in \sigma(F)$, $\alpha \notin e(E)$ and $(e(E))^+_F = (e(E'))^+_F$. Moreover we know from Lemma 5.1 that $e(E') \in \theta(F)$, i.e. that there is no $S \in \sigma(F)$ with $S^+_F \supset (e(E'))^+_F$ and therefore also not with $S^+_F \supset (e(E))^+_F$. Hence $E \in \theta(F)$ and since $\alpha \notin E$, we conclude that α is not skeptically accepted in F w.r.t. θ. ($\Leftarrow$): Assume α is not skeptically accepted in F w.r.t. θ. Hence there is some $T \in \theta(F)$ with $\alpha \notin T$. Now assume, towards a contradiction, that *Reject* is not reachable, meaning, by (1) and (2), that *Accept* is reachable. Hence $Fail_{base}$ is applicable from a state $(\epsilon, E', E)_{base}$. By the definition of f^{σ}_{base}, this means that there is no σ-extension S of F such that $\alpha \notin S$ and $\neg\exists S' \in \epsilon : S^+_F \subseteq S'$. Now note that $Fail_{out}$ is the only rule where elements are added to ϵ. By Lemma 5.1, such elements are ranges of θ-extensions of F. But therefore for each $S \in \sigma(F)$ with $\alpha \notin S$ it holds that $\exists T \in \theta(F) : S^+_F \subseteq T^+_F \wedge \alpha \in T$, a contradiction to α not being skeptically accepted in F w.r.t. θ. □

Proof of Theorem 5.3. Finiteness is inherited from SKEPT-PRF$^{F,\alpha}_{\overline{f}}$. For acyclicity we consider $<$ from Definition 3, but extending $<_i$ by adding $pre <_i j$ for $j \in \{base, max, out\}$. With this, $Fail_{pre}$ is increasing due to (ii) as the set of extensions ϵ will stay empty during the application of rules of index *pre*. $Succeed_{pre}$ results in a terminal state and finally, also the oracle rules are increasing, as this is independent from the index.

A *pre*-state cannot be terminal since if "no other rule applies", $Succeed_{pre}$ is applied, resulting in *Reject*. Hence any terminal state reachable from the initial state is either *Accept* or *Reject*.

Since the shortcut can only reject instances it follows from Theorem 3.6 that if *Accept* is reachable then α is skeptically accepted in F w.r.t. *prf*. If, on the other hand, *Accept* is not reachable, then *Reject* is reached either by application of $Succeed_{out}$ or by application of $Succeed_{pre}$. In the first case we again know from Theorem 3.6 that α is not skeptically accepted (note that $Fail_{pre}$ leads to state $(\emptyset, \emptyset, \emptyset)_{base}$, which is just the initial state of SKEPT-PRF$^{F,\alpha}_{\overline{f}}$). In the second case there is some $S \in \sigma(F)$ which attacks α, therefore also a $T \in prf(F)$ which attacks α, hence $\alpha \notin T$. Again α is not skeptically accepted in F w.r.t. *prf*. □

References

[1] P. Baroni, M.W.A. Caminada and M. Giacomin, An introduction to argumentation semantics, *The Knowledge Engineering Review* **26**(4) (2011), 365–410. doi:10.1017/S0269888911000166.

[2] R. Baumann, Splitting an argumentation framework, in: *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2011*, J.P. Delgrande and W. Faber, eds, Lecture Notes in Computer Science, Vol. 6645, Springer, 2011, pp. 40–53.

[3] R. Baumann and C. Spanring, Infinite argumentation frameworks – On the existence and uniqueness of extensions, in: *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation – Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, T. Eiter, H. Strass, M. Truszczynski and S. Woltran, eds, Lecture Notes in Computer Science, Vol. 9060, Springer, 2015, pp. 281–295.

[4] T.J.M. Bench-Capon and P.E. Dunne, Argumentation in artificial intelligence, *Artificial Intelligence* **171**(10–15) (2007), 619–641. doi:10.1016/j.artint.2007.05.001.

[5] P. Besnard and S. Doutre, Checking the acceptability of a set of arguments, in: *Proceedings of the 10th International Workshop on Non-Monotonic Reasoning, NMR 2004*, J.P. Delgrande and T. Schaub, eds, 2004, pp. 59–64.

[6] R. Brochenin, Y. Lierler and M. Maratea, Abstract disjunctive answer set solvers, in: *Proceedings of the 21st European Conference on Artificial Intelligence, ECAI 2014*, T. Schaub, G. Friedrich and B. O'Sullivan, eds, Frontiers in Artificial Intelligence and Applications, Vol. 263, IOS Press, 2014, pp. 165–170.

[7] R. Brochenin, Y. Lierler and M. Maratea, Disjunctive answer set solvers via templates, *Theory and Practice of Logic Programming* **16**(4) (2016), 465–497. doi:10.1017/S1471068415000411.
[8] R. Brochenin, T. Linsbichler, M. Maratea, J.P. Wallner and S. Woltran, Abstract solvers for Dung's argumentation frameworks, in: *Proceedings of the 3rd International Workshop on Theory and Applications of Formal Argumentation, TAFA 2015, Revised Selected Papers*, E. Black, S. Modgil and N. Oren, eds, Lecture Notes in Computer Science, Vol. 9524, Springer, 2015, pp. 40–58.
[9] R. Brochenin and M. Maratea, Abstract solvers for quantified Boolean formulas and their applications, in: *Proceedings of the 14th International Conference of the Italian Association for Artificial Intelligence, AI*IA 2015*, M. Gavanelli, E. Lamma and F. Riguzzi, eds, Lecture Notes in Computer Science, Vol. 9336, Springer, 2015, pp. 205–217.
[10] R. Brochenin and M. Maratea, Abstract answer set solvers for cautious reasoning, in: *Proceedings of the Technical Communications of the 31st International Conference on Logic Programming, ICLP 2015*, M.D. Vos, T. Eiter, Y. Lierler and F. Toni, eds, CEUR Workshop Proceedings, Vol. 1433, CEUR-WS.org, 2015.
[11] M. Caminada, W. Carnielli and P.E. Dunne, Semi-stable semantics, *Journal of Logic and Computation* **22**(5) (2012), 1207–1254. doi:10.1093/logcom/exr033.
[12] T. Castell, C. Cayrol, M. Cayrol and D.L. Berre, Using the Davis and Putnam procedure for an efficient computation of preferred models, in: *Proceedings of the 12th European Conference on Artificial Intelligence, ECAI 1996*, W. Wahlster, ed., Wiley, Chichester, 1996, pp. 350–354.
[13] F. Cerutti, P.E. Dunne, M. Giacomin and M. Vallati, Computing preferred extensions in abstract argumentation: A SAT-based approach, in: *Proceedings of the 2nd International Workshop on Theory and Applications of Formal Argumentation, TAFA 2013, Revised Selected Papers*, E. Black, S. Modgil and N. Oren, eds, Lecture Notes in Computer Science, Vol. 8306, Springer, 2014, pp. 176–193.
[14] F. Cerutti, M. Giacomin and M. Vallati, ArgSemSAT: Solving argumentation problems using SAT, in: *Proceedings of the 5th International Conference on Computational Models of Argument, COMMA 2014*, S. Parsons, N. Oren, C. Reed and F. Cerutti, eds, Frontiers in Artificial Intelligence and Applications, Vol. 266, IOS Press, 2014, pp. 455–456.
[15] F. Cerutti, M. Giacomin and M. Vallati, Algorithm selection for preferred extensions enumeration, in: *Proceedings of the 5th International Conference on Computational Models of Argument, COMMA 2014*, S. Parsons, N. Oren, C. Reed and F. Cerutti, eds, Frontiers in Artificial Intelligence and Applications, Vol. 266, IOS Press, 2014, pp. 221–232.
[16] F. Cerutti, M. Giacomin, M. Vallati and M. Zanella, An SCC recursive meta-algorithm for computing preferred labellings in abstract argumentation, in: *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning, KR 2014*, C. Baral, G.D. Giacomo and T. Eiter, eds, AAAI Press, 2014, pp. 42–51.
[17] F. Cerutti, N. Oren, H. Strass, M. Thimm and M. Vallati, A benchmark framework for a computational argumentation competition, in: *Proceedings of the 5th International Conference on Computational Models of Argument, COMMA 2014*, S. Parsons, N. Oren, C. Reed and F. Cerutti, eds, Frontiers in Artificial Intelligence and Applications, Vol. 266, IOS Press, 2014, pp. 459–460.
[18] F. Cerutti, M. Vallati and M. Giacomin, Where are we now? State of the art and future trends of solvers for hard argumentation problems, in: *Computational Models of Argument – Proceedings of COMMA 2016*, P. Baroni, T.F. Gordon, T. Scheffler and M. Stede, eds, Frontiers in Artificial Intelligence and Applications, Vol. 287, IOS Press, 2016, pp. 207–218.
[19] G. Charwat, W. Dvořák, S.A. Gaggl, J.P. Wallner and S. Woltran, Methods for solving reasoning problems in abstract argumentation – A survey, *Artificial Intelligence* **220** (2015), 28–63. doi:10.1016/j.artint.2014.11.008.
[20] M. Davis, G. Logemann and D. Loveland, A machine program for theorem proving, *Communications of the ACM* **5**(7) (1962), 394–397. doi:10.1145/368273.368557.
[21] Y. Dimopoulos and A. Torres, Graph theoretical structures in logic programs and default theories, *Theoretical Computer Science* **170**(1–2) (1996), 209–244. doi:10.1016/S0304-3975(96)80707-9.
[22] S. Doutre and J. Mengin, Preferred extensions of argumentation frameworks: Query answering and computation, in: *Proceedings of the 1st International Joint Conference on Automated Reasoning, IJCAR 2001*, R. Goré, A. Leitsch and T. Nipkow, eds, Lecture Notes in Computer Science, Vol. 2083, Springer, 2001, pp. 272–288.
[23] P.M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, *Artificial Intelligence* **77**(2) (1995), 321–358. doi:10.1016/0004-3702(94)00041-X.
[24] P.E. Dunne and T.J.M. Bench-Capon, Coherence in finite argument systems, *Artificial Intelligence* **141**(1/2) (2002), 187–203. doi:10.1016/S0004-3702(02)00261-8.
[25] W. Dvořák, M. Järvisalo, J.P. Wallner and S. Woltran, Complexity-sensitive decision procedures for abstract argumentation, *Artificial Intelligence* **206** (2014), 53–78. doi:10.1016/j.artint.2013.10.001.
[26] W. Dvořák and S. Woltran, Complexity of semi-stable and stage semantics in argumentation frameworks, *Information Processing Letters* **110**(11) (2010), 425–430. doi:10.1016/j.ipl.2010.04.005.
[27] S.A. Gaggl, T. Linsbichler, M. Maratea and S. Woltran, Introducing the second international competition on computational models of argumentation, in: *Proceedings of the 1st International Workshop on Systems and Algorithms for Formal*

Argumentation (SAFA 2016), M. Thimm, F. Cerutti, H. Strass and M. Vallati, eds, 2016, pp. 4–9. https://www.dbai.tuwien.ac.at/iccma17/Introducing_ICCMA17.pdf.
[28] S.A. Gaggl, N. Manthey, A. Ronca, J.P. Wallner and S. Woltran, Improved answer-set programming encodings for abstract argumentation, *Theory and Practice of Logic Programming* **15**(4–5) (2015), 434–448. doi:10.1017/S1471068415000149.
[29] M. Gebser, B. Kaufmann and T. Schaub, Conflict-driven answer set solving: From theory to practice, *Artificial Intelligence* **187** (2012), 52–89. doi:10.1016/j.artint.2012.04.001.
[30] M. Järvisalo and T.A. Junttila, Limitations of restricted branching in clause learning, *Constraints* **14**(3) (2009), 325–356. doi:10.1007/s10601-008-9062-z.
[31] M. Järvisalo and I. Niemelä, The effect of structural branching on the efficiency of clause learning SAT solving: An experimental study, *Journal of Algorithms* **63**(1–3) (2008), 90–113. doi:10.1016/j.jalgor.2008.02.005.
[32] J. Lagniez, E. Lonca and J. Mailly, CoQuiAAS: A constraint-based quick abstract argumentation solver, in: *Proceedings of the 27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015*, IEEE Computer Society, 2015, pp. 928–935.
[33] B. Liao, Toward incremental computation of argumentation semantics: A decomposition-based approach, *Annals of Mathematics and Artificial Intelligence* **67**(3–4) (2013), 319–358. doi:10.1007/s10472-013-9364-8.
[34] B. Liao, *Efficient Computation of Argumentation Semantics*, Intelligent Systems Series, Academic Press, 2014. ISBN 978-0-12-410406-8. http://store.elsevier.com/product.jsp?isbn=9780124104068.
[35] Y. Lierler, Abstract answer set solvers, in: *Proceedings of the 24th International Conference on Logic Programming, ICLP 2008*, M.G. de la Banda and E. Pontelli, eds, Lecture Notes in Computer Science, Vol. 5366, Springer, 2008, pp. 377–391.
[36] Y. Lierler, Abstract answer set solvers with backjumping and learning, *Theory and Practice of Logic Programming* **11**(2–3) (2011), 135–169. doi:10.1017/S1471068410000578.
[37] Y. Lierler, Relating constraint answer set programming languages and algorithms, *Artificial Intelligence* **207** (2014), 1–22. doi:10.1016/j.artint.2013.10.004.
[38] Y. Lierler and M. Truszczynski, Transition systems for model generators – A unifying approach, *Theory and Practice of Logic Programming* **11**(4–5) (2011), 629–646. doi:10.1017/S1471068411000214.
[39] Y. Lierler and M. Truszczynski, Modular answer set solving, in: *Late-Breaking Developments in the Field of Artificial Intelligence*, AAAI Workshops, Vol. WS-13-17, AAAI, 2013.
[40] Y. Lierler and M. Truszczynski, Abstract modular inference systems and solvers, in: *Proceedings of the 16th International Symposium on Practical Aspects of Declarative Languages, PADL 2014*, M. Flatt and H. Guo, eds, Lecture Notes in Computer Science, Vol. 8324, Springer, 2014, pp. 49–64.
[41] Y. Lierler and M. Truszczynski, An abstract view on modularity in knowledge representation, in: *Proceedings of the 29th AAAI Conference on Artificial Intelligence, AAAI 2015*, B. Bonet and S. Koenig, eds, AAAI Press, 2015, pp. 1532–1538.
[42] Y. Lierler and M. Truszczynski, On abstract modular inference systems and solvers, *Artificial Intelligence* **236** (2016), 65–89. doi:10.1016/j.artint.2016.03.004.
[43] S. Modgil and M.W.A. Caminada, Proof theories and algorithms for abstract argumentation frameworks, in: *Argumentation in Artificial Intelligence*, I. Rahwan and G.R. Simari, eds, Springer, 2009, pp. 105–129.
[44] R. Nieuwenhuis, A. Oliveras and C. Tinelli, Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T), *Journal of the ACM* **53**(6) (2006), 937–977. doi:10.1145/1217856.1217859.
[45] S. Nofal, K. Atkinson and P.E. Dunne, Algorithms for decision problems in argument systems under preferred semantics, *Artificial Intelligence* **207** (2014), 23–51. doi:10.1016/j.artint.2013.11.001.
[46] I. Rahwan and G.R. Simari (eds), *Argumentation in Artificial Intelligence*, Springer, 2009. doi:10.1007/978-0-387-98197-0.
[47] E.D. Rosa, E. Giunchiglia and M. Maratea, Solving satisfiability problems with preferences, *Constraints* **15**(4) (2010), 485–515. doi:10.1007/s10601-010-9095-y.
[48] R. Sebastiani, Lazy satisfiability modulo theories, *Journal of Satisfiability, Boolean Modeling and Computation* **3**(3–4) (2007), 141–224.
[49] M. Thimm and S. Villata, System descriptions of the first International competition on computational models of argumentation (ICCMA'15), 2015. arXiv:1510.05373.
[50] M. Thimm and S. Villata, The first international competition on computational models of argumentation: Results and analysis, *Artificial Intelligence* **252** (2017), 267–294. doi:10.1016/j.artint.2017.08.006.
[51] M. Thimm, S. Villata, F. Cerutti, N. Oren, H. Strass and M. Vallati, Summary report of the first international competition on computational models of argumentation, *AI Magazine* **37**(1) (2016), 102–104. doi:10.1609/aimag.v37i1.2640.
[52] B. Verheij, Two approaches to dialectical argumentation: Admissible sets and argumentation stages, in: *Proceedings of the 8th Dutch Conference on Artificial Intelligence, NAIC 1996*, J.-J.C. Meyer and L.C. van der Gaag, eds, 1996, pp. 357–368.
[53] J.P. Wallner, G. Weissenbacher and S. Woltran, Advanced SAT techniques for abstract argumentation, in: *Proceedings of the 14th International Workshop on Computational Logic in Multi-Agent Systems, CLIMA 2013*, J. Leite, T.C. Son, P. Torroni, L. van der Torre and S. Woltran, eds, Lecture Notes in Computer Science, Vol. 8143, Springer, 2013, pp. 138–154.