Abstract Solvers for Dung's Argumentation Frameworks

Remi Brochenin[1], Thomas Linsbichler[2], Marco Maratea[1], Johannes Peter Wallner[3], and Stefan Woltran[2]

[1] University of Genoa, Italy
[2] TU Wien, Austria
[3] HIIT, Department of Computer Science, University of Helsinki, Finland

Abstract. Abstract solvers are a quite recent method to uniformly describe algorithms in a rigorous formal way and have proven successful in declarative paradigms such as Propositional Satisfiability and Answer Set Programming. In this paper, we apply this machinery for the first time to a dedicated AI formalism, namely Dung's abstract argumentation frameworks. We provide descriptions of several advanced algorithms for the preferred semantics in terms of abstract solvers and, moreover, show how slight adaptions thereof directly lead to new algorithms.

1 Introduction

Dung's concept of abstract argumentation [12] is nowadays a core formalism in AI [2, 21]. 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 are however intractable and might even be hard for the second level of the polynomial hierarchy [13, 15]. Thus, efficient and advanced algorithms have to be developed in order to deal with real-world size data with reasonable performance. The argumentation community is currently facing this challenge [7] and a first solver competition[4] has been organized in 2015. Thus, a number of new algorithms and systems are currently under development. 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 algorithms on this basis may not be fruitful. More formal descriptions, which allow, e.g. for a uniform representation, have thus been developed: a recent and successful approach in this direction is the concept of *abstract solvers* [19]. Hereby, one characterizes the states of computation as nodes of a graph, the techniques as arcs between nodes, and the whole solving process as a path in the graph. This concept not only proved successful for SAT [19], but also has been applied for several variants of Answer-Set Programming [4, 16, 17].

[4] `http://argumentationcompetition.org`

In this paper, we make a first step to investigate the appropriateness of abstract solvers for dedicated AI formalisms and focus on certain problems in Dung's argumentation frameworks. In order to understand whether abstract solvers are powerful enough, we consider quite advanced algorithms – ranging from dedicated [20] to reduction-based [5, 14] approaches (see [8] for a recent survey) – for solving problems that are hard for the second level of the polynomial hierarchy. We show that abstract solvers allow for convenient algorithm design resulting in a clear and mathematically precise description, and how formal properties of the algorithms are easily specified by means of related graph properties. We also illustrate how abstract solvers simplify the *combination* of techniques implemented in different solvers in order to define new solving procedures. Consequently, our findings not only prove that abstract solvers are a valuable tool for specifying and analysing argumentation algorithms, but also indicate the broad range the novel concept of abstract solvers can be applied to. To sum up, our main contributions are as follows:

- We provide a full formal description of recent algorithms [5, 14, 20] for reasoning tasks under the preferred semantics in terms of abstract solvers, thus enabling a comparison of these approaches at a formal level.
- We give proofs illustrating how formal correctness of the considered algorithms can be shown with the help of descriptions in terms of abstract solvers.
- We outline how our reformulations can be used to gain more insight into the algorithms and how novel combinations of "levels" of abstract solvers might pave the way for new solutions.

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 the target algorithms. The paper ends in Section 4 with final remarks and possible topics for future research.

2 Preliminaries

In this section we first review (abstract) argumentation frameworks [12] and their semantics (see [1] for an overview), and then introduce abstract transition systems [19] on the concrete instance describing the DPLL-procedure [9].

Abstract Argumentation Frameworks. An *argumentation framework (AF)* is a pair $F = (A, R)$ where A is a finite 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.

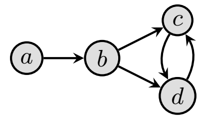

Fig. 1: AF F with $\mathit{prf}(F) = \{\{a,c\},\{a,d\}\}$.

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

- *$S \in \mathit{adm}(F)$ if each $a \in S$ is defended by S;*
- *$S \in \mathit{com}(F)$ if $S \in \mathit{adm}(F)$ and for each $a \in A$ defended by S, $a \in S$ holds;*
- *$S \in \mathit{prf}(F)$ if $S \in \mathit{adm}(F)$ (resp. $S \in \mathit{com}(F)$) and there is no $T \in \mathit{adm}(F)$ (resp. $T \in \mathit{com}(F)$) 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 Figure 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: $\mathit{adm}(F) = \{\emptyset, \{a\}, \{a, c\}, \{a, d\}\}$, $\mathit{com}(F) = \{\{a\}, \{a, c\}, \{a, d\}\}$, and $\mathit{prf}(F) = \{\{a, c\}, \{a, d\}\}$.

Given an AF $F = (A, R)$, an argument $a \in A$, and a semantics σ, the problem of skeptical acceptance asks whether it is the case that a is contained in all σ-extensions of F. While skeptical acceptance is trivial for *adm* and decidable in polynomial time for *com*, it is Π_2^P-complete for *prf*, see [10, 12, 13]. The class $\Pi_2^P = \text{coNP}^{\text{NP}}$ denotes the class of problems P, such that the complementary problem $\overline{P}$ can be decided by a nondeterministic polynomial time algorithm that has (unrestricted) access to an NP-oracle.

Abstract Solvers. Most SAT solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) procedure [9]. We give an abstract transition system for DPLL following the work of Nieuwenhuis et al. in [19], and start with basic notation.

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 $\mathit{atoms}(\varphi)$ (resp. $\mathit{atoms}(M)$). We identify a consistent set E of literals (i.e. a set that does not contain complementary literals such as a and $\neg a$) with an assignment to $\mathit{atoms}(E)$ as follows: if $a \in E$ then a maps to *true*, while if $\neg a \in E$ then a maps to *false*. For sets X and Y of atoms such that $X \subseteq Y$, we identify X with an assignment over Y as follows: if $a \in X$ then a maps to *true*, while if $a \in Y \setminus X$ then a maps to *false*. By $\mathrm{Sat}(\varphi)$ we refer to the set of satisfying assignments of φ.

We now introduce the 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

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 all the literals of $\overline{C}$ occur in E and $C \vee l$ is a clause in φ
Decide	$E \Rightarrow El^d$	if	E is consistent and neither l nor $\bar{l}$ occur in E

Failing rule

Fail	$E \Rightarrow$ *Reject*	if	E is inconsistent and decision-free

Succeeding rule

Succeed	$E \Rightarrow$ *Accept*	if	no other rule applies

Fig. 2: The transition rules of DP_φ.

to the set $\{a\}$. We say that a record E is *inconsistent* if it contains $\bot$ or both a literal l and its complement $\bar{l}$, and consistent otherwise. We sometimes identify a record with the set containing all its elements without annotations. 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*.

Each CNF formula φ determines its DPLL *graph* DP_φ. The set of nodes of DP_φ consists of the records relative to $atoms(\varphi)$ and the distinguished states *Accept* and *Reject*. A node in the graph is *terminal* if no edge originates from it; in practice, the terminal nodes are *Accept* and *Reject*. The edges of the graph DP_φ are specified by the transition rules presented in Figure 2. In solvers, generally the oracle rules are chosen with the preference order following the order in which they are stated in Figure 2, but 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.

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

3 Algorithms for Preferred Semantics

In this section we abstract two SAT-based algorithms for preferred semantics, namely PrefSat [5] (implemented in the tool ArgSemSat [6]) for extension enumeration, and an algorithm for deciding skeptical acceptance of cegartix [14]. Moreover, we abstract the dedicated approach for enumeration of [20]. In Section 3.4 we show how our graph representations can be used to develop novel algorithms, by combining parts of PrefSat and parts of the dedicated algorithm.

We will present these algorithms in a uniform way, abstracting from some minor tool-specific details. Moreover, even if abstract solvers are mainly conceived as a modeling formalism, in our solutions a certain level of systematicity can be outlined, that helps in the design of such abstract solvers. In fact, common to all algorithms is a conceptual two-level architecture of computation, similar to Answer Set Programming solvers for disjunctive logic programs [4]. The lower level corresponds to a dpll-like search subprocedure, while the higher level part takes care of the control flow 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.

Each algorithm uses its own data structures, and, by slight abuse of notation, for a given AF $F = (A, R)$ we denote their used variables in our graph representation by $atoms(F)$. For this set it holds that $A \subseteq atoms(F)$, i.e. the status of the arguments can be identified from this set of atoms. The states of our graph representations of all algorithms 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 E is a record that may be currently modified by a subprocedure. Note that both E and E' are records, since they will be modified by subprocedures, while found preferred extensions will be translated to a set of arguments before being stored in ϵ. The annotation i denotes the current (sub)procedure we are in. Both *base* and *max* correspond to different lower level computations, typically SAT calls, while *out* is used solely for (simple) checks outside such subprocedures. Transition rules reflecting the higher level of computation shift these annotations, e.g. from a terminated subprocedure *base* to subprocedure *max*, and transition rules mirroring rules "inside" a SAT solver do not modify i.

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

In order to show acyclicity of graphs later in this section, we define a strict partial order on states.

Definition 2. *Let E be a record. E can be written as $L^0 l_1^d L^1 \dots l_p^d L^p$ where $l_1^d, \dots, l_p^d$ are all the decision literals of E. We define $s(E) = |L^0|, |L^1|, \dots, |L^p|$.*

Definition 3. *Let* ϵ_1, ϵ_2 *be 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* $s(E_1) <_{lex} s(E_2)$*, where* $<_{lex}$ *is the lexicographic order.*
$<_i$*:* $base <_i max <_i out$*.*

The strict partial order $<$ *on states is defined such that for any two states* $S_1 = (\epsilon_1, E'_1, E_1)_{i_1}$ *and* $S_2 = (\epsilon_2, E'_2, E_2)_{i_2}$*,* $S_1 < S_2$ *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* $E_1 <_E E_2$*.*

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

The SAT-based algorithms construct formulas by an oracle function f s.t. $A \subseteq atoms(f(\epsilon, E, F, \alpha)) \subseteq atoms(F)$ for all possible arguments of f, in particular for $\alpha \in A$. The formulas $f(\epsilon, E, F, \alpha)$ are adapted from [3]. 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.

3.1 SAT-based Algorithm for Enumeration

PrefSat (Algorithm 1 of [5]) is a SAT-based algorithm for finding all preferred extensions of a given AF F. 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, i.e. a preferred extension. This preferred extension is stored, and we repeat the process.

In PrefSat we have 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 via abstract solver. The graph $\text{ENUM}^F_{\overline{f}}$ for an AF $F = (A, R)$ and a vector of oracle functions $\overline{f}$ is defined by the states over $atoms(F)$ and the transition rules presented in Figure 3. 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:

1. $\{e(M) \mid M \in \text{Sat}(f^{com}_{base}(\epsilon, E, F, \alpha))\} = \{E'' \in com(F) \mid \neg\exists E' \in \epsilon : E'' \subseteq E'\}$;
2. $\{e(M) \mid M \in \text{Sat}(f^{com}_{max}(\epsilon, E, F, \alpha))\} = \{E'' \in com(F) \mid e(E) \subset E''\}$.

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

$Backtrack_i$	$(\epsilon, E', El^d E'')_i \Rightarrow (\epsilon, E', E\bar{l})_i$	if	$\begin{cases} El^d E'' \text{ is inconsistent and} \\ E'' \text{ contains no decision literal} \end{cases}$
$UnitPropagate_i$	$(\epsilon, E', E)_i \Rightarrow (\epsilon, E', El)_i$	if	$\begin{cases} l \text{ does not occur in } E \text{ and} \\ \text{all the literals of } \overline{C} \text{ occur in } E \text{ and} \\ C \vee l \text{ is a clause in } f_i^{com}(\epsilon, E', F, \alpha) \end{cases}$
$Decide_i$	$(\epsilon, E', E)_i \Rightarrow (\epsilon, E', El^d)_i$	if	$\begin{cases} E \text{ is consistent and} \\ \text{neither } l \text{ nor } \bar{l} \text{ occur in } E \end{cases}$

Succeeding rules

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

Failing rules

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

Fig. 3: The rules of ENUM_f^F.

We remark that α is not relevant for enumeration of extensions and only used for acceptance later on. 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 subprocedure, 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.

If the conditions of a rule with annotation i check for consistency, we implicitly refer to the formula generated by f_i^{com}. That is, if a $Fail_i$ rule is applied to the state $(\epsilon, E', E)_i$ for $i \in \{base, max\}$, the formula $f_i^{com}(\epsilon, E', F, \alpha)$ is unsatisfiable. Conversely, when a $Succeed_i$ rule is applied, the formula $f_i^{com}(\epsilon, E', F, \alpha)$ 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 has not 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 subprocedure 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 2. Again consider the AF F depicted in Figure 1. We have seen in Example 1 that F has two preferred extensions, namely $\{a, c\}$ and $\{a, d\}$. Figure 4 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 A, and do not state the extra literals that may have been assigned during the

$$
\begin{array}{ll}
\text{Initial state}: & (\emptyset,\emptyset,\emptyset)_{base} \\
\textit{base}\text{-oracle}: & (\emptyset,\emptyset,E_1 \supseteq \{a,\neg b,\neg c,\neg d\})_{base} \\
\textit{Succeed}_{base}: & (\emptyset,E_1,\emptyset)_{max} \\
\textit{max}\text{-oracle}: & (\emptyset,E_1,E_2 \supseteq \{a,\neg b,\neg c,d\})_{max} \\
\textit{Succeed}_{max}: & (\emptyset,E_2,\emptyset)_{max} \\
\textit{max}\text{-oracle}: & (\emptyset,E_2,\textit{unsat})_{max} \\
\textit{Fail}_{max}: & (\{\{a,d\}\},\emptyset,\emptyset)_{base} \\
\textit{base}\text{-oracle}: & (\{\{a,d\}\},\emptyset,E_3 \supseteq \{a,\neg b,c,\neg d\})_{base} \\
\textit{Succeed}_{base}: & (\{\{a,d\}\},E_3,\emptyset)_{max} \\
\textit{max}\text{-oracle}: & (\{\{a,d\}\},E_3,\textit{unsat})_{max} \\
\textit{Fail}_{max}: & (\{\{a,d\},\{a,c\}\},\emptyset,\emptyset)_{base} \\
\textit{base}\text{-oracle}: & (\{\{a,d\},\{a,c\}\},\emptyset,\textit{unsat})_{base} \\
\textit{Fail}_{base}: & \textit{Ok}(\{\{a,d\},\{a,c\}\})
\end{array}
$$

Fig. 4: Path in $\textsc{Enum}^F_f$ where F is the AF from Figure 1.

call to the SAT-solver. By *unsat* we represent an inconsistent and decision-free record.

It remains to show that $\textsc{Enum}^F_f$ correctly describes PrefSat by showing that we reach a terminal state containing all preferred extensions of F. We begin with a lemma stating that we only add preferred extensions to ϵ which have not been found at this point.

Lemma 1. *For any AF F if the rule $Fail_{max}$ is applied from state $(\epsilon, E', E)_{max}$ in the graph $\textsc{Enum}^F_{(f^{com}_{base}, f^{com}_{max})}$ then $e(E') \in prf(F)$ and $e(E') \notin \epsilon$.*

Proof. Let $S_1 = (\epsilon_1, E'_1, E_1)_{max}$ be the state from which $Fail_{max}$ is applied. This means that f^{com}_{max} is unsatisfiable, hence, by the definition of formula f^{com}_{max}, there is no $C \in com(F)$ with $C \supset e(E'_1)$. 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 an application of $Succeed_{base}$ or $Succeed_{max}$. In both cases $e(E'_1)$ corresponds to a complete extension of F, since E'_1 is a satisfying assignment of the formula f^{com}_{base} or f^{com}_{max}, respectively. Therefore 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 that there is a record E' such that $\neg\exists C \in \epsilon : e(E') \subseteq C$. Moreover every application of $Succeed_{max}$ updates E' by a proper superset of itself. Therefore $e(E'_1) \supseteq e(E')$ and also $\neg\exists C \in \epsilon : e(E'_1) \subseteq C$, in particular $e(E'_1) \notin \epsilon$. □

Now we are ready to show correctness of $\textsc{Enum}^F_f$.

Theorem 1. *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 $\textsc{Enum}_{\overline{f}}^{F}$ is finite, consider some state $(\epsilon, E', E)_i$ of $\textsc{Enum}_{\overline{f}}^{F}$. 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, $\textsc{Enum}_{\overline{f}}^{F}$ only contains states with $i \in \{base, max\}$. Thus the number of states is finite in the graph $\textsc{Enum}_{\overline{f}}^{F}$.

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. $<$. To this end consider two states $S_1 = (\epsilon_1, E_1', E_1)_{i_1}$ and $S_2 = (\epsilon_2, E_2', E_2)_{i_2}$. First of all, the i-oracle rules (i.e. $Backtrack_i$, $UnitPropagate_i$, and $Decide_i$) fulfill $S_1 < S_2$ 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 $E_1 <_l E_2$. Moreover, $Fail_{max}$ fulfills $S_1 < S_2$ due to (i) since $e(E_1') \notin \epsilon_1$ by Lemma 1. $Succeed_{base}$ guarantees $S_1 < S_2$ because of (ii). Finally, $Succeed_{max}$ fulfills $S_1 < S_2$ due to (iii), since the max-oracle rules work on the formula f_{max}^{com} and the extension associated with a satisfying assignment $E_1 = E_2'$ thereof must be a proper superset of $e(E_1')$. Therefore, for any two states S_1 and S_n such that S_n is reachable from S_1 in $\textsc{Enum}_{\overline{f}}^{F}$ it holds that $S_1 < S_n$, showing that the graph is acyclic.

The only terminal state reachable from the initial state is $Ok(\epsilon)$ (via rule $Fail_{base}$) since all states $S = (\epsilon, E, E')_i$ of $\textsc{Enum}_{\overline{f}}^{F}$ 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 1 that for each $P \in \epsilon$ it holds that $P \in prf(F)$. To reach $Ok(\epsilon)$, the rule $Fail_{base}$ must have been applied from state $(\epsilon, E', E)_{base}$. This means, by the definition of f_{base}^{com}, that for each complete extension C of F there is some $P \in \epsilon$ such that $C \subseteq P$. Hence $\epsilon = prf(F)$. □

3.2 SAT-based Algorithm for Acceptance

CEGARTIX [14] is a SAT-based tool for deciding several acceptance questions for AFs. Here we focus on Algorithm 1 of [14] for deciding skeptical acceptance under preferred semantics of an argument α. Similarly as 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 to be contained in the candidate σ-extension before maximization. Again, we have two kinds of SAT-calls.

The graph $\textsc{Skept-prf}_{\overline{f}}^{F,\alpha}$ for an AF F, an argument α and a vector of oracle functions $\overline{f}$ is defined by the states over $atoms(F)$ and the rules in Figure 3 replacing the $Fail_i$ rules and adding the out rules as depicted in Figure 5. The initial state is $(\emptyset, \emptyset, \emptyset)_{base}$. For $\sigma \in \{adm, com\}$ we assume the functions f_{base}^{σ} and f_{max}^{σ} such that:

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. 5: Changed transition rules for $\textsc{Skept-prf}^{F,\alpha}_{\vec{f}}$.

1. $\{e(M) \mid M \in \text{Sat}(f^{\sigma}_{base}(\epsilon, E, F, \alpha))\} = \{E'' \in \sigma(F) \mid \alpha \notin E'' \wedge \neg\exists E' \in \epsilon : E'' \subseteq E'\}$;
2. $\{e(M) \mid M \in \text{Sat}(f^{\sigma}_{max}(\epsilon, E, F, \alpha))\} = \{E'' \in \sigma(F) \mid e(E) \subset E''\}$.

The graph $\textsc{Skept-prf}^{F,\alpha}_{\vec{f}}$ is nearly identical to $\textsc{Enum}^{F}_{\vec{f}}$. It differs only in case of failure in subprocedure *base* or *max*. When all the preferred extensions have been enumerated in subprocedure *base*, we can report a positive outcome with *Accept*, since we have ensured that α belongs to all of them. In subprocedure *max*, when a preferred extension has been found, it is here necessary to check whether α belongs to it. The *out* rules correspond to an if-then-else construct: if the condition $\alpha \notin E'$ holds then we follow the $Succeed_{out}$ rule else follow the $Fail_{out}$ rule. In other words, if α is not in the extension then the procedure can terminate with a negative answer; else proceed as in the previous graph: add the preferred extension to ϵ and search for a new one by going back to *base*.

Example 3. Again consider the AF F from Figure 1 and note that skeptical acceptance of argument c does not hold as c is not contained in the preferred extension $\{a, d\}$ of F. Accordingly, the possible path of the graph $\textsc{Skept-prf}^{F,c}_{\vec{f}}$ which is depicted in Figure 6a (with base semantics *adm*) terminates in the *Reject*-state.

On the other hand, argument a is skeptically accepted under 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 $\textsc{Skept-prf}^{F,a}_{\vec{f}}$ (again with base semantics *adm*) is shown in Figure 6b. As expected, the path terminates in the state *Accept*.

We will use the following lemma to show correctness of $\textsc{Skept-prf}^{F,\alpha}_{\vec{f}}$. Its proof is almost identical to the one of Lemma 1 and therefore omitted.

Lemma 2. *For any AF F, if the rule $Fail_{out}$ is applied from state $(\epsilon, E', E)_{out}$ in the graph $\textsc{Skept-prf}^{F,\alpha}_{(f^{\sigma}_{base}, f^{\sigma}_{max})}$ with $\sigma \in \{adm, com\}$ then $e(E') \in prf(F)$ and $e(E') \notin \epsilon$.*

Theorem 2. *For any AF $F = (A, R)$, argument $\alpha \in A$, and $\sigma \in \{adm, com\}$, the graph $\textsc{Skept-prf}^{F,\alpha}_{(f^{\sigma}_{base}, f^{\sigma}_{max})}$ is finite, acyclic and any terminal state reachable*

$$
\begin{array}{ll}
\text{Inital state}: & (\emptyset,\emptyset,\emptyset)_{base}\\
base\text{-oracle}: & (\emptyset,\emptyset,E_1\supseteq\{a,\neg b,\neg c,\neg d\})_{base}\\
Succeed_{base}: & (\emptyset,E_1,\emptyset)_{max}\\
max\text{-oracle}: & (\emptyset,E_1,E_2\supseteq\{a,\neg b,c,\neg d\})_{max}\\
Succeed_{max}: & (\emptyset,E_2,\emptyset)_{max}\\
max\text{-oracle}: & (\emptyset,E_2,unsat)_{max}\\
Fail_{max}: & (\emptyset,E_2,\emptyset)_{out}\\
Fail_{out}: & (\{\{a,c\}\},\emptyset,\emptyset)_{base}\\
base\text{-oracle}: & (\{\{a,c\}\},\emptyset,E_3\supseteq\{a,\neg b,\neg c,d\})_{base}\\
Succeed_{base}: & (\{\{a,c\}\},E_3,\emptyset)_{max}\\
max\text{-oracle}: & (\{\{a,c\}\},E_3,unsat)_{max}\\
Fail_{max}: & (\{\{a,c\}\},E_3,\emptyset)_{out}\\
Succeed_{out}: & Reject
\end{array}
$$

(a) Reject-path for argument c in $\textsc{Skept-prf}_f^{F,c}$.

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

(b) Accept-path for argument a in $\textsc{Skept-prf}_f^{F,a}$.

Fig. 6: Possible paths in $\textsc{Skept-prf}_f^{F,\alpha}$.

from the initial state is either Accept or Reject; Accept is reachable iff α is skeptically accepted in F w.r.t. prf.

Proof. (1) $\textsc{Skept-prf}_f^{F,\alpha}$ is finite and acyclic: In order to show finiteness note that the states $(\epsilon, E', E)_i$ of $\textsc{Skept-prf}_f^{F,\alpha}$ coincide with the states of $\textsc{Enum}_f^F$, there is just an additional option *out* for i. Hence finiteness follows from Theorem 1. In order to show that $\textsc{Skept-prf}_f^{F,\alpha}$ is acyclic we have to show that the rules that differ in $\textsc{Skept-prf}_f^{F,\alpha}$ from $\textsc{Enum}_f^F$ (i.e. the ones listed in Figure 5) are increasing with respect to the ordering $<$ from Definition 3: $Fail_{out}$ fulfills $S_1 < S_2$ due to (i) by Lemma 2, $Fail_{max}$ guarantees $S_1 < S_2$ because of (ii), and $Fail_{base}$ and $Succeed_{out}$ end in terminal states.

(2) Any terminal state of $\textsc{Skept-prf}_f^{F,\alpha}$ reachable from the initial state is either *Reject* or *Accept*: Consider the state $S = (\epsilon, E, E')_i$. If $i \in \{base, max\}$ then there is a rule $Succeed_i$ with the condition "no other rule applies", hence S 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 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 $Fail_{max}$ was applied at a state $(\epsilon, E', E'')_{max}$, meaning that $f^{\sigma}_{max}(\epsilon, E', F, \alpha)$ is unsatisfiable, i.e. there is no σ-extension C with $C \supset e(E')$. 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}. Now as $e(E') \in \sigma(F)$, $\neg\exists C \supset e(E') : C \in \sigma(F)$, and $\alpha \notin e(E')$, we have that α is not skeptically accepted by F w.r.t. *prf*. $\Leftarrow$: Assume α is skeptically rejected by F w.r.t. *prf*. Hence there is some $P \in prf(F)$ with $\alpha \notin P$. 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}, this means that there is no σ-extension C of F with $\alpha \notin C$ and $\neg\exists E'' \in \epsilon : C \subseteq E''$. Now note that $Fail_{out}$ is the only rule where elements are added to ϵ. Moreover, by Lemma 2, we know that elements added are preferred extensions of F. But therefore for each $C \in \sigma(F)$ with $\alpha \notin C$ it holds that $\exists E'' \in prf(F) : C \subseteq E'' \wedge \alpha \in E''$, a contradiction. □

Finally note that from Theorem 1 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 $\text{SKEPT-PRF}^{F,\alpha}_{f}$.

3.3 Dedicated Approach for Enumeration

Algorithm 1 of [20] presents a direct approach for enumerating preferred extensions. One function is important for this algorithm, which is called IN-TRANS. It marks an argument $x \in A$ as belonging to the extension which is currently constructed, and marks all attackers $\{y \mid (y, x) \in R\}$ and all attacked arguments $\{y \mid (x, y) \in R\}$ as outside of this extension. Intuitively, IN-TRANS *decides* to accept x, 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 reverts the effects of IN-TRANS, marks the argument it chose as outside of this extension, and calls itself recursively. This can be seen as a *backtrack*.

We have defined an equivalent representation of this algorithm that follows the framework of abstract solvers with binary logics as previously used in this

Oracle rules

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

$Fail_{max}$	$(\epsilon, \emptyset, E)_{max}$	$\Rightarrow Ok(\epsilon)$	if $\{ E$ is incons. and decision-free
$Succeed_{max}$	$(\epsilon, \emptyset, E)_{max}$	$\Rightarrow (\epsilon, \emptyset, E)_{out}$	if $\{$ no other rule applies
$Fail_{out}$	$(\epsilon, \emptyset, E)_{out}$	$\Rightarrow (\epsilon, \emptyset, E\bot)_{max}$	if $\begin{cases} \exists E' \in \epsilon : E \subseteq E' \text{ or} \\ \text{there is an argument } a \text{ s.t.} \\ E \text{ does not attack } a \text{ and} \\ a \text{ attacks } E \end{cases}$
$Succeed_{out}$	$(\epsilon, \emptyset, E)_{out}$	$\Rightarrow (\epsilon \cup \{e(E)\}, \emptyset, E\bot)_{max}$	if $\{$ no other rule applies

Fig. 7: The rules of the graph DIRECT^F.

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 does not attack a and a attacks 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 Figure 7. 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 subprocedure. 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 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 [20], but does not seem to be crucial here. 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. If the record corresponds to a preferred extension, then it is stored by $Succeed_{out}$ and the process of trying all possible records 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.

Example 4. A possible path in the graph DIRECT^F for the AF F in Figure 1 is shown in Figure 8. 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.

$$
\begin{array}{ll}
\text{Initial state}: & (\emptyset,\emptyset,\emptyset)_{max}\\
\mathit{Decide}'_{max}: & (\emptyset,\emptyset,c^d)_{max}\\
\mathit{Propagate}'_{max}: & (\emptyset,\emptyset,c^d\neg b\neg d)_{max}\\
\mathit{Decide}'_{max}: & (\emptyset,\emptyset,c^d\neg b\neg da^d)_{max}\\
\mathit{Succeed}_{max}: & (\emptyset,\emptyset,c^d\neg b\neg da^d)_{out}\\
\mathit{Succeed}_{out}: & (\{\{a,c\}\},\emptyset,c^d\neg b\neg da^d\bot)_{max}\\
\mathit{Backtrack}'_{max}: & (\{\{a,c\}\},\emptyset,c^d\neg b\neg d\neg a)_{max}\\
\mathit{Succeed}_{max}: & (\{\{a,c\}\},\emptyset,c^d\neg b\neg d\neg a)_{out}\\
\mathit{Fail}_{out}: & (\{\{a,c\}\},\emptyset,c^d\neg b\neg d\neg a\bot)_{max}\\
\mathit{Backtrack}'_{max}: & (\{\{a,c\}\},\emptyset,\neg c)_{max}\\
 & \ldots\\
\mathit{Succeed}_{max}: & (\{\{a,c\}\},\emptyset,\neg ca^d\neg bd^d)_{out}\\
\mathit{Succeed}_{out}: & (\{\{a,c\},\{a,d\}\},\emptyset,\neg ca^d\neg bd^d\bot)_{max}\\
\mathit{Backtrack}'_{max}: & (\{\{a,c\},\{a,d\}\},\emptyset,\neg ca^d\neg b\neg d)_{max}\\
\mathit{Succeed}_{max}: & (\{\{a,c\},\{a,d\}\},\emptyset,\neg ca^d\neg b\neg d)_{out}\\
\mathit{Fail}_{out}: & (\{\{a,c\},\{a,d\}\},\emptyset,\neg ca^d\neg b\neg d\bot)_{max}\\
\mathit{Backtrack}'_{max}: & (\{\{a,c\},\{a,d\}\},\emptyset,\neg c\neg a)_{max}\\
 & \ldots\\
\mathit{Succeed}_{max}: & (\{\{a,c\},\{a,d\}\},\emptyset,\neg c\neg a\neg b\neg d)_{out}\\
\mathit{Fail}_{out}: & (\{\{a,c\},\{a,d\}\},\emptyset,\neg c\neg a\neg b\neg d\bot)_{max}\\
\mathit{Fail}_{max}: & \mathit{Ok}(\{\{a,d\},\{a,c\}\})
\end{array}
$$

Fig. 8: Path in $\textsc{Direct}^F$ where F is the AF from Figure 1.

Then $\bot$ is added to the current record by Fail_{out}, followed by backtracking to the last decision literal.

Lemma 3. *For any AF F, if the rule $\mathit{Succeed}_{out}$ is applied from state $(\epsilon,\emptyset,E)_{out}$ in the graph $\textsc{Direct}^F$ then $e(E)\in \mathit{prf}(F)$ and $e(E)\notin\epsilon$.*

Proof. The application of $\mathit{Succeed}_{out}$ from state $S_{out}=(\epsilon,\emptyset,E)_{out}$ must have been preceded by $\mathit{Succeed}_{max}$ from the state $S_{max}=(\epsilon,\emptyset,E)_{max}$ which only differs from S_{out} in i. We now analyze the record E as it is constructed by the rules Decide'_{max}, $\mathit{Propagate}'_{max}$ and $\mathit{Backtrack}'_{max}$. The application of Decide'_{max} adds literal a, literal $\neg b$ is added by $\mathit{Propagate}'_{max}$ 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 a such that E does not attack a and a attacks E", then Fail_{out} is applied instead of $\mathit{Succeed}_{out}$. To get $e(E)\in\mathit{prf}(F)$ it remains to show that there is no $D\in\mathit{adm}(F)$ with $D\supset e(E)$. Assume there is such a $D\in\mathit{adm}(F)$. Then there must be some $a\in D$ with $a\notin e(E)$. Now observe that the graph first adds a to the record and afterwards $\neg a$. Therefore D must have been discovered in advance. But then $\exists D\in\epsilon : e(E)\subseteq D$, hence Fail_{out} is applied instead of $\mathit{Succeed}_{out}$. This condition is also the reason why $e(E)\notin\epsilon$ is guaranteed when $\mathit{Succeed}_{out}$ is applied from state S_{out}. □

Theorem 3. *For any AF F, the graph $\textsc{Direct}^F$ is finite, acyclic and the only terminal state reachable from its initial state is $\mathit{Ok}(\epsilon)$ where $\epsilon=\mathit{prf}(F)$.*

Proof. Since states of Direct^F consist of the same elements as states of Enum^F_f, finiteness of Direct^F follows in the same way as in Theorem 1.

To show that Direct^F is acyclic we will, again as in the proof of Theorem 1, 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 $S_1 = (\epsilon_1, \emptyset, E_1)_{i_1}$ and $S_2 = (\epsilon_2, \emptyset, E_2)_{i_2}$, $S_1 <_D S_2$ 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 orders from Definition 3. First of all, the oracle rules (i.e. $\mathit{Backtrack}'_{max}$, $\mathit{UnitPropagate}'_{max}$, and Decide'_{max}) and Fail_{out} fulfill $S_1 <_D S_2$ 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, $\mathit{Succeed}_{out}$ fulfills $S_1 <_D S_2$ due to (i) since $e(E_1) \notin \epsilon_1$ by Lemma 3. $\mathit{Succeed}_{max}$ guarantees $S_1 <_D S_2$ because of (iii).

The only terminal state reachable from the initial state is $\mathit{Ok}(\epsilon)$ 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 $\mathit{Succeed}_i$ with the condition "no other rule applies". It remains to show that, when state $\mathit{Ok}(\epsilon)$ is reached, ϵ is the set of preferred extensions of F. Since elements are only added to ϵ by rule $\mathit{Succeed}_{out}$ we know from Lemma 3 that for each $P \in \epsilon$ it holds that $P \in \mathit{prf}(F)$. On the other hand, the oracle rules guarantee that each conflict-free set C of F a set $(\epsilon, \emptyset, E)_{out}$ with $e(E) = C$ is reached. If C is then admissible and maximal w.r.t. ϵ (which contains only preferred extensions of F as observed before), C is added to ϵ. Therefore each $P \in \mathit{prf}(F)$ is contained in ϵ. □

3.4 Combining Algorithms

We can now define a new algorithm which is a combination of the PrefSat approach and the dedicated approach. In fact, it replaces the loop of SAT-calls for maximizing a complete extension of PrefSat by a part of the dedicated algorithm of [20]. In particular, instead of having subsequent oracle calls for maximization, we utilize the dedicated algorithm with a different initialization and stop already when the first preferred extension has been found. The graph mix-prf^F_f representing this algorithm consists of the oracle rules and the rules $\mathit{Succeed}_{max}$ and Fail_{out} of Direct^F, the *base*-oracle rules and the rule Fail_{base} of Enum^F_f and the rules in Figure 9. The initial state is $(\emptyset, \emptyset, \emptyset)_{base}$.

As in Enum^F_f, whenever a $\mathit{Succeed}_{base}$ rule is applied, a complete extension has been generated and it has to be validated or extended by the subprocedure identified with *max*. When $\mathit{Succeed}_{max}$ is applied, a preferred extension has been found and the search for another complete extension can be started. Whenever an extension has been found by procedure *base*, there is a preferred extension that is a superset of the found extension. Hence, there is no need for a rule Fail_{max}, since subprocedure *max* will always succeed.

Succeeding and failing rules

$$Succeed_{base}\ (\epsilon, \emptyset, E)_{base} \Rightarrow (\epsilon, \emptyset, e(E))_{max} \quad \text{if} \begin{cases} \text{no other rule applies} \end{cases}$$

$$Succeed_{out}\ (\epsilon, \emptyset, E)_{out} \Rightarrow (\epsilon \cup \{e(E)\}, \emptyset, \emptyset)_{base} \quad \text{if} \begin{cases} \text{no other rule applies} \end{cases}$$

Fig. 9: The rules of the graph MIX-PRF^F_f.

Lemma 4. *For any AF $F = (A, R)$, if the rule $Succeed_{out}$ is applied from state $(\epsilon, \emptyset, E)_{out}$ in the graph* MIX-PRF^F_f *then $e(E) \in prf(F)$ and $e(E) \notin \epsilon$.*

Proof. By the definition of f^{com}_{base} we know that when the rule $Succeed_{base}$ is applied at state $(\epsilon, \emptyset, E)_{base}$ it holds that $e(E)$ is a complete extension of F. By the same reasoning as in the proof of Lemma 3, $e(E) \in prf(F)$ and $e(E) \notin \epsilon$. □

Theorem 4. *For any AF F, the graph* MIX-PRF^F_f *is finite, acyclic and the only terminal state reachable from its initial state is $Ok(\epsilon)$ where $\epsilon = prf(F)$.*

Proof. Since states of MIX-PRF^F_f coincide with the ones of ENUM^F_f, finiteness of MIX-PRF^F_f follows in the same way as in Theorem 1.

For acyclicity of MIX-PRF^F_f we begin with the following observation. Consider a state $S = (\epsilon, \emptyset, E)_{max}$ and assume none of the *max*-oracle-rules apply. Then $Succeed_{max}$ with the condition "no other rule applies" applies and we only change the index of the state to then have just an if-else-decision between $Fail_{out}$ and $Succeed_{out}$. We get the same behavior when removing the rule $Succeed_{max}$ and changing $Fail_{out}$ and $Succeed_{out}$ in the following way:

$$Fail'_{out} \quad (\epsilon, \emptyset, E)_{max} \Rightarrow (\epsilon, \emptyset, E\bot)_{max} \quad \text{if} \begin{cases} \text{no other rule applies and} \\ (\exists E' \in \epsilon : E \subseteq E' \text{ or} \\ \text{there is an argument } a \text{ such that} \\ E \text{ does not attack } a \text{ and } a \text{ attacks } E) \end{cases}$$

$$Succeed'_{out}\ (\epsilon, \emptyset, E)_{max} \Rightarrow (\epsilon \cup \{e(E)\}, \emptyset, \emptyset)_{base} \quad \text{if} \begin{cases} \text{no other rule applies} \end{cases}$$

We show that the modified version MIX-PRF^F_f is acyclic and therefore get, by the considerations above, that MIX-PRF^F_f is acyclic. We do so by showing that all rules in the modified version of MIX-PRF^F_f are increasing w.r.t. the strict partial order $<$ from Definition 3. The *base*-oracle-rules and $Fail_{base}$ were shown to be increasing in the proof of Theorem 1. Moreover, $Succeed_{base}$ is increasing because of (ii). The *max*-oracle-rules and $Fail'_{out}$ are increasing due to (iv). Finally, $Succeed'_{out}$ is increasing by (i) (cf. Lemma 4).

The only terminal state reachable from the initial state is $Ok(\epsilon)$ since for each $i \in \{base, max, out\}$ there is a rule $Succeed_i$ with the condition "no other rule applies". It remains to show that ϵ coincides with the preferred extensions of F. Since extensions are exclusively added by the application of $Succeed_{out}$ it follows from Lemma 4 that $\epsilon \subseteq prf(F)$. The state $Ok(\epsilon)$ must have been reached by the application of $Fail_{base}$. Hence, by definition of f^{com}_{base}, we know that there is no complete extension C of F such that $\forall E' \in \epsilon : C \not\subseteq E'$. Therefore also for each $P \in \mathit{prf}(F)$, $\exists E' \in \epsilon : P \subseteq E'$. Since ϵ only contains preferred extensions of F we get $P \in \epsilon$, and finally $\mathit{prf}(F) \subseteq \epsilon$. □

4 Discussion and 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 AFs through graph-based abstract solvers instead of pseudo-code-based descriptions. Both SAT-based and dedicated approaches have been analyzed and compared. Moreover, by a combination of these approaches we have obtained a novel algorithm for enumeration of preferred extensions.

Our work shows the potential of abstract transition systems to describe, compare and combine algorithms also in the research field of abstract argumentation, as already happened in, e.g. SAT, SMT and ASP. In particular, the last feature, which allows the design of new solving procedures by combining reasoning modules from different algorithms, seems to be particularly appealing. However, we do not claim about the efficiency of a new tool built on this basis, given that it usually requires many iterations of theoretical analysis, practical engineering, and domain-specific optimizations to develop efficient systems.

We have focused on the well-studied preferred semantics and presented core algorithms. However, the machinery can be easily employed to describing algorithms for other reasoning tasks, such as credulous acceptance, or different semantics, e.g. semi-stable and stage semantics, as employed in CEGARTIX [14]. Moreover, specific optimization techniques can be taken into account by means of modular addition of transition rules to the graph describing the core parts of the algorithms. As future work we plan to make these points more concrete.

Moreover, we envisage to formally describe further algorithms for reasoning tasks within abstract argumentation (see e.g. [8, 11, 18]). In particular, the results of the competition suggest promising candidates for the application of the newly gained technique of algorithm combination via abstract solvers.

Acknowledgements

This work has been funded by the Austrian Science Fund (FWF) through project I1102, and by Academy of Finland through grants 251170 COIN and 284591.

References

1. Baroni, P., Caminada, M.W.A., Giacomin, M.: An introduction to argumentation semantics. The Knowledge Engineering Review 26(4), 365–410 (2011)
2. Bench-Capon, T.J.M., Dunne, P.E.: Argumentation in artificial intelligence. Artificial Intelligence 171(10-15), 619–641 (2007)
3. Besnard, P., Doutre, S.: Checking the Acceptability of a Set of Arguments. In: Delgrande, J.P., Schaub, T. (eds.) Proc. of the 10th International Workshop on Non-Monotonic Reasoning, NMR 2004. pp. 59–64 (2004)

4. Brochenin, R., Lierler, Y., Maratea, M.: Abstract disjunctive answer set solvers. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) Proc. of the 21st European Conference on Artificial Intelligence, ECAI 2014. FAIA, vol. 263, pp. 165–170. IOS Press (2014)
5. Cerutti, F., Dunne, P.E., Giacomin, M., Vallati, M.: Computing preferred extensions in abstract argumentation: A SAT-based approach. In: Black, E., Modgil, S., Oren, N. (eds.) Proc. of the 2nd International Workshop on Theory and Applications of Formal Argumentation, TAFA 2013. LNCS, vol. 8306, pp. 176–193. Springer (2014)
6. Cerutti, F., Giacomin, M., Vallati, M.: ArgSemSAT: Solving argumentation problems using SAT. In: Parsons, S., Oren, N., Reed, C., Cerutti, F. (eds.) Proc. of the 5th International Conference on Computational Models of Argument, COMMA 2014. FAIA, vol. 266, pp. 455–456. IOS Press (2014)
7. Cerutti, F., Oren, N., Strass, H., Thimm, M., Vallati, M.: A benchmark framework for a computational argumentation competition. In: Parsons, S., Oren, N., Reed, C., Cerutti, F. (eds.) Proc. of the 5th International Conference on Computational Models of Argument, COMMA 2014. FAIA, vol. 266, pp. 459–460. IOS Press (2014)
8. Charwat, G., Dvořák, W., Gaggl, S.A., Wallner, J.P., Woltran, S.: Methods for Solving Reasoning Problems in Abstract Argumentation - A Survey. Artificial Intelligence 220, 28–63 (2015)
9. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Communications of the ACM 5(7), 394–397 (1962)
10. Dimopoulos, Y., Torres, A.: Graph theoretical structures in logic programs and default theories. Theoretical Computer Science 170(1-2), 209–244 (1996)
11. Doutre, S., Mengin, J.: Preferred extensions of argumentation frameworks: Query answering and computation. In: Goré, R., Leitsch, A., Nipkow, T. (eds.) Proc. of the 1st International Joint Conference on Automated Reasoning, IJCAR 2001. LNCS, vol. 2083, pp. 272–288. Springer (2001)
12. Dung, P.M.: On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. Artificial Intelligence 77(2), 321–358 (1995)
13. Dunne, P.E., Bench-Capon, T.J.M.: Coherence in finite argument systems. Artificial Intelligence 141(1/2), 187–203 (2002)
14. Dvořák, W., Järvisalo, M., Wallner, J.P., Woltran, S.: Complexity-sensitive decision procedures for abstract argumentation. Artificial Intelligence 206, 53–78 (2014)
15. Dvořák, W., Woltran, S.: Complexity of semi-stable and stage semantics in argumentation frameworks. Information Processing Letters 110(11), 425–430 (2010)
16. Lierler, Y.: Abstract answer set solvers with backjumping and learning. Theory and Practice of Logic Programming 11(2-3), 135–169 (2011)
17. Lierler, Y.: Relating constraint answer set programming languages and algorithms. Artificial Intelligence 207, 1–22 (2014)
18. Modgil, S., Caminada, M.W.A.: Proof theories and algorithms for abstract argumentation frameworks. In: Rahwan, I., Simari, G.R. (eds.) Argumentation in Artificial Intelligence, pp. 105–129. Springer (2009)
19. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). Journal of the ACM 53(6), 937–977 (2006)
20. Nofal, S., Atkinson, K., Dunne, P.E.: Algorithms for decision problems in argument systems under preferred semantics. Artificial Intelligence 207, 23–51 (2014)
21. Rahwan, I., Simari, G.R. (eds.): Argumentation in Artificial Intelligence. Springer (2009)