Preferred Reasoning in ABA by Cycle-Breaking

Kiet Ngyuen Anh[1] , **Markus Ulbricht**[2]
ScaDS.AI Dresden/Leipzig, Leipzig University
[1]kietnguyen2023@hotmail.com [2]mulbricht@informatik.uni-leipzig.de

Abstract

We develop a fixed-parameter tractable (FPT) algorithm for skeptical preferred reasoning in assumption-based argumentation (ABA). To this end we make use of so-called *backdoors*, i.e., sets of assumptions that need to be evaluated s.t. the remaining ABA framework (ABAF) belongs to a computationally beneficial sub-class. In order to identify such target classes, we employ a suitable notion of a *dependency graph* of an ABAF. We show that these graphs can be constructed in polynomial time and that one can efficiently check sufficient properties ensuring that reasoning in the underlying ABAF is tractable. After establishing the theoretical foundations, we test our implementation against the ASPforABA solver which convincingly won the ABA track of the ICCMA'23 competition. As it turns out, our algorithm outperforms ASPforABA on instances with small backdoor sizes.

1 Introduction

Computational models of argumentation [Baroni *et al.*, 2018] play a central role in non-monotonic reasoning and have a wide range of applications in various fields such as law and healthcare [Atkinson *et al.*, 2017]. A well-known computational model of argumentation are so-called *abstract argumentation frameworks*, introduced by Dung [Dung, 1995]. In this work, Dung formalizes that a debate can be represented and evaluated by viewing arguments as atomic entities and attacks as a relation between them. This way, the discussion is represented as a directed graph $F = (A, R)$ where A is a set of *arguments* and $R \subseteq A \times A$ the *attack* relation. To study an AF in the field of formal argumentation, the notion of *semantics* was introduced, mapping an AF to sets of jointly acceptable arguments [Baroni *et al.*, 2011]. Several aspects of AFs have been studied extensively in the literature, e.g. enforcing arguments [Wallner *et al.*, 2017; Niskanen *et al.*, 2018], belief revision [Falappa *et al.*, 2009; Haret *et al.*, 2018], repairing a semantical collapse [Baumann and Ulbricht, 2019], or the role of formal argumentation for explainability in AI [Cyras *et al.*, 2021].

Since an AF $F = (A, R)$ can be interpreted as a directed graph, researchers make use of graph-theoretical properties in order to study computationally beneficial argumentation scenarios. For instance, the theoretical computational complexity has not only been studied extensively for AFs in general [Dvorák and Dunne, 2018], but also for sub-classes like e.g. i) AFs without odd-length cycles (odd-cycle free AFs), ii) AFs without even-length cycles (no-even AFs), iii) AFs without any cycle whatsoever (acyclic AFs). Moreover, means to compute models of F step-wise have been studied [Baroni *et al.*, 2005; Baumann, 2011; Baumann *et al.*, 2020].

Another line of research is concerned with so-called *fixed-parameter tractability* (FPT) for reasoning problems in AFs [Dunne, 2007; Dvorák *et al.*, 2012b; Dvorák *et al.*, 2022a] or e.g. probabilistic AFs [Liao *et al.*, 2018]. The idea is to identify a suitable parameter s.t. a given intractable problem becomes tractable whenever the parameter does not exceed a certain bound. In [Dvorák *et al.*, 2012a] the notion of *backdoors* has been considered: e.g. an acyclic backdoor is a set of argument s.t. their removal from the graph results in an acyclic AF. The authors show that, if the size of the smallest acyclic backdoor is bounded, then many important reasoning problems for AFs become tractable [Dvorák *et al.*, 2012a], i.e., these problems are FPT. This procedure has recently also been lifted to argumentation frameworks with collective attacks (SETAFs) [Dvorák *et al.*, 2022b].

One key aspect in the argumentation pipeline is the use of structured argumentation formalisms [Besnard *et al.*, 2014], which are employed to outline formal argumentative workflows from building blocks. Prominent approaches include assumption-based argumentation (ABA) [Bondarenko *et al.*, 1997], ASPIC$^+$ [Modgil and Prakken, 2013], defeasible logic programming DeLP [García and Simari, 2004], and deductive argumentation [Besnard and Hunter, 2008]. The reasoning process within these formalisms typically involves creating argument structures and identifying conflicts among them in a systematic manner from rule-based knowledge bases. The resulting arguments and conflicts can be captured by constructing a suitable AF. This AF can be utilized to determine the acceptability of arguments and draw conclusions based on the original knowledge bases.

In this paper, we focus on ABA as it is one of the key structured argumentation formalisms [Čyras *et al.*, 2018]. While the theoretical computational complexity of reasoning in ABA has also been investigated [Dvorák and Dunne, 2018] and solvers have been designed [Lehtonen *et al.*, 2021;

Lehtonen *et al.*, 2023], the study of FPT algorithms is not yet as extensive as in the case of AFs (one exception is a recent dynamic programming approach based on a bounded tree-width [Popescu and Wallner, 2023]). Since an ABA knowledge base is a rule-based system and no graph, means to identify suitable sub-classes as in the case of AFs are not immediate. Consequently, finding a notion of a no-even or acyclic backdoor in ABA is a more involved endeavor. In this paper, we approach this issue by identifying a suitable *dependency graph* for ABA which entails the properties we desire; for instance, we show that if there is no cycle in this graph, then reasoning in the corresponding ABA framework is tractable (a result similar to acyclic AFs). Building upon this cycle notion, we devise and implement an FPT-algorithm for skeptical preferred reasoning in ABA. Along the way, we establish several required theoretical results, which are interesting on their own, independent of our FPT procedure. Our main contributions can be summarized as follows.

- We study ABA classes induced by a polynomial-time computable *dependency graph*. Section 3
- We show how to perform computations in ABA in a modular way. Section 4
- We design and implement a backdoor algorithm for skeptical preferred reasoning in ABA. Section 5

The source code for our algorithm can be found online[1].

2 Background

Abstract Argumentation. An argumentation framework (AF) [Dung, 1995] is a directed graph $F = (A, R)$ where A represents a set of arguments and $R \subseteq A \times A$ models *attacks* between them. For two arguments $x, y \in A$, if $(x, y) \in R$ we say that x *attacks* y. We let $E^+ = \{x \in A \mid E \text{ attacks } x\}$ for a set $E \subseteq A$. A set $E \subseteq A$ is *conflict-free* in F iff for no $x, y \in E$, $(x, y) \in R$; E *defends* an argument x if E attacks each attacker of x. A conflict-free set E is *admissible* in F ($E \in ad(F)$) iff it defends all its elements. A *semantics* is a function with $F \mapsto \sigma(F) \subseteq 2^A$. This means, given an AF $F = (A, R)$ a semantics returns a set of subsets of A. These subsets are called *σ-extensions*. In this paper we consider so-called *complete*, *grounded*, *preferred*, and *stable* semantics (abbr. co, gr, pr, stb).

Definition 2.1. *Let $F = (A, R)$ be an AF and $E \in ad(F)$.*

- *$E \in co(F)$ iff E contains all arguments it defends;*
- *$E \in gr(F)$ iff E is $\subseteq$-minimal in $co(F)$;*
- *$E \in pr(F)$ iff E is $\subseteq$-maximal in $co(F)$;*
- *$E \in stb(F)$ iff $E^+ = A \setminus E$.*

Assumption-based Argumentation. We assume a deductive system $(\mathcal{L}, \mathcal{R})$, where $\mathcal{L}$ is a formal language, i.e., a set of sentences, and $\mathcal{R}$ is a set of inference rules over $\mathcal{L}$. A rule $r \in \mathcal{R}$ has the form $a_0 \leftarrow a_1, \ldots, a_n$ with $a_i \in \mathcal{L}$. We denote the head of r by $head(r) = a_0$ and the (possibly empty) body of r with $body(r) = \{a_1, \ldots, a_n\}$.

[1]https://github.com/kietGithubUser/ABA-Backdoor-Implementation

Definition 2.2. *An ABAF is a tuple $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$, where $(\mathcal{L}, \mathcal{R})$ is a deductive system, $\mathcal{A} \subseteq \mathcal{L}$ a set of assumptions, and a partial contrary function $^- : \mathcal{A} \to \mathcal{L}$.*

Note that we allow for assumptions without any contrary since $^-$ is partial. This does not make any conceptual difference, but is more convenient for our purpose.

Assumption 2.3. *In this work, we focus on ABA frameworks which are* flat*, i.e., for each rule $r \in \mathcal{R}$, $head(r) \notin \mathcal{A}$ (no assumption can be derived), and* finite*, i.e., $\mathcal{L}$, $\mathcal{R}$, $\mathcal{A}$ are finite; moreover, we assume $\mathcal{L}$ to be a set of atoms.*

We say that a sentence $p \in \mathcal{L}$ is *derivable* from assumptions $S \subseteq \mathcal{A}$ and rules $R \subseteq \mathcal{R}$, denoted by $S \vdash_R p$, if there is a finite rooted labeled tree T such that the root is labeled with p, the set of labels for the leaves of T is equal to S or $S \cup \{\top\}$, and for every inner node v of T there is a rule $r \in R$ such that v is labelled with $head(r)$, the number of successors of v is $|body(r)|$ and every successor of v is labelled with a distinct $a \in body(r)$ or $\top$ if $body(r) = \emptyset$

By $Th_{\mathcal{D}}(S) = \{p \in \mathcal{L} \mid \exists S' \subseteq S : S' \vdash_R p\}$ we denote the set of all conclusions derivable from an assumption set S in $\mathcal{D}$. Observe that $S \subseteq Th_{\mathcal{D}}(S)$ as $a \in \mathcal{A}$ is derivable from $\{a\} \vdash_{\emptyset} a$. For $S \subseteq \mathcal{A}$, let $\overline{S} = \{\overline{a} \mid a \in S\}$; moreover, for a derivation $S \vdash p$ we write $asms(S \vdash p) = S$ and for a set E of derivations we let $asms(E) = \bigcup_{x \in E} asms(x)$. Also, we often write $S \vdash_R p$ simply as $S \vdash p$.

A set $S \subseteq \mathcal{A}$ *attacks* a set $T \subseteq \mathcal{A}$, if for some $a \in T$ we have that $\overline{a} \in Th_{\mathcal{D}}(S)$. A set S is *conflict-free*, denoted $S \in cf(\mathcal{D})$, if it does not attack itself; it is *admissible*, denoted $S \in ad(\mathcal{D})$, if S is conflict-free and S *defends* itself, i.e., for each set $T \subseteq \mathcal{A}$, we have that if T attacks S, then S attacks T as well. With a little notational abuse we say S attacks a if S attacks the singleton $\{a\}$.

We next recall grounded, complete, preferred, and stable ABA semantics (abbr. gr, co, pr, stb).

Definition 2.4. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ be an ABAF. Further, let $S \subseteq \mathcal{A}$ be admissible in $\mathcal{D}$.*

- *$S \in co(\mathcal{D})$ iff S contains all assumptions it defends;*
- *$S \in gr(\mathcal{D})$ iff S is $\subseteq$-minimal in $co(\mathcal{D})$;*
- *$S \in pr(\mathcal{D})$ iff S is $\subseteq$-maximal in $co(\mathcal{D})$;*
- *$S \in stb(\mathcal{D})$ iff S attacks each $\{x\} \subseteq \mathcal{A} \setminus S$.*

We say that an assumption $a \in \mathcal{A}$ is *credulously accepted* (*skeptically accepted*) wrt. a semantics σ in an ABAF $\mathcal{D}$ iff $a \in \bigcup \sigma(\mathcal{D})$ (iff $\sigma(\mathcal{D}) \neq \emptyset$ and $a \in \bigcap \sigma(\mathcal{D})$). In this case, we write $a \in cred_\sigma(\mathcal{D})$ resp. $a \in skep_\sigma(\mathcal{D})$.

An ABAF induces an AF as follows.

Definition 2.5. *The associated AF $F_{\mathcal{D}} = (A, R)$ of an ABAF $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ is given by $A = \{S \vdash p \mid S \subseteq \mathcal{A}, p \in \mathcal{L}\}$ and attack relation $(S \vdash p, S' \vdash p') \in R$ iff $p \in \overline{S'}$.*

ABA and AFs are closely related (see [Čyras *et al.*, 2018]).

Proposition 2.6. *Given an ABAF $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$, its corresponding AF $F_{\mathcal{D}}$ and a semantics $\sigma \in \{gr, co, pr, stb\}$. If $E \in \sigma(F_{\mathcal{D}})$ then $asms(E) \in \sigma(\mathcal{D})$; if $S \in \sigma(\mathcal{D})$ then $\{S' \vdash p \mid \exists S' \subseteq S, R \subseteq \mathcal{R} : S' \vdash_R p\} \in \sigma(F_{\mathcal{D}})$.*

Since it suffices to instantiate only finitely many arguments (see e.g. [Lehtonen *et al.*, 2023]) we assume that the AF corresponding to some ABA $\mathcal{D}$ is finite, if not stated otherwise.

3 ABA Classes

In the context of AFs, several graph classes have been studied in the literature with corresponding computational benefits [Dvorák and Dunne, 2018]. For instance, AFs without any even-length cycle (also called *no-even* AFs) have exactly one complete extension (which, in this case, can be computed in polynomial time).

However, since ABA is a rule-based system, designing such algorithms in terms of suitable graph-classes is not immediate. A natural starting point would be to utilize Proposition 2.6 in order to obtain the desired properties from the instantiated AF $F_{\mathcal{D}}$. Indeed, the following observation is a direct corollary of known ABA properties.

Fact 3.1. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$ be an ABAF. If $F_{\mathcal{D}}$ is*

- *odd-cycle free, then $pr(\mathcal{D}) = stb(\mathcal{D})$;*
- *even-cycle free, then $gr(\mathcal{D}) = co(\mathcal{D}) = pr(\mathcal{D})$;*
- *acyclic, then $gr(\mathcal{D}) = co(\mathcal{D}) = pr(\mathcal{D}) = stb(\mathcal{D})$.*

This result is, however, impractical as it requires computing the (possibly exponential) argumentation graph $F_{\mathcal{D}}$. In this section, we seek to analyze a given ABAF $\mathcal{D}$ in a similar way, but thereby avoid computing $F_{\mathcal{D}}$.

3.1 ABA Dependency Graphs

A natural way to extract a graph from ABA is the notion of a *dependency graph*. Such graphs have been constructed in e.g. [Craven and Toni, 2016], but we need to differ from this notion to obtain a graph that is suitable for our purpose.

Our notion of dependency graphs is inspired by similar concepts for logic programming [Apt and Bol, 1994, Definition 2.2] and the more recent ABA dependency graph [Rapberger *et al.*, 2022]; we require, however, a different path notion. The idea is that the graph captures the dependencies between atoms, i.e., there is an edge from p to q whenever there is some rule with p in the body and q in its head. In addition, we draw an edge from $\overline{a}$ to a for each assumption $a \in \mathcal{A}$ to account for attacks.

Definition 3.2. *The* dependency graph *$\mathcal{G}_{\mathcal{D}} = (V, E, l)$ for a given ABA $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$ is an edge-labelled graph with*

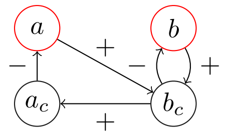
- *$V = \mathcal{L}$ is the set of vertices,*
- *there is an edge $e = (s, t) \in E$ iff i) there is some $r \in \mathcal{R}$ with $s \in body(r)$ and $head(r) = t$, in this case $l(e) = +$; or ii) $t \in \mathcal{A}$ and $\overline{t} = s$, in this case, $l(e) = -$.*

Recall that our goal is to obtain a result in the spirit of Fact 3.1 in terms of the dependency graph $\mathcal{G}_{\mathcal{D}}$. The question is thus how to define paths and cycles in $\mathcal{G}_{\mathcal{D}}$. Since the "$-$" labeled edges account for attacks in the ABAF, it makes senses to define the length of a path by the number of "$-$" labeled edges. However, there is some technical issue we need to take consideration, as illustrated in the following.

Example 3.3. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$ be the ABAF with $\mathcal{A} = \{a, b\}$, $\mathcal{L} = \mathcal{A} \cup \{a_c, b_c\}$, the expected contraries, i.e., $\overline{a} = a_c$ as well as $\overline{b} = b_c$, and the following rules $\mathcal{R}$:*

$$r_1 : b_c \leftarrow a \qquad r_2 : b_c \leftarrow b \qquad r_3 : a_c \leftarrow b_c$$

Then the dependency graph $\mathcal{G}_D$ is given as follows (we highlight the assumptions in the graph to enhance readability).



We can construct a path form a to b, namely a, b_c, b, but then we cannot continue this path to turn it into a cycle a, b_c, b, b_c, a_c, a since this sequence visits b_c twice. However, ABA is tailored to investigate the interaction of assumptions and ordinary atoms are mere intermediate steps in derivations. We would thus expect $\mathcal{G}_{\mathcal{D}}$ to contain an even-cycle.

The problem in the previous example was that visiting b_c twice prevented us from constructing an even-cycle from a to b and back. This can be circumvented by allowing to visit ordinary atoms multiple times. We thus define paths and their length in dependency graphs as follows.

Definition 3.4. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$ be an ABAF and $\mathcal{G}_{\mathcal{D}} = (V, E, l)$ its dependency graph. A* weak path *is a sequence $p_1, ..., p_n$ with $n \geq 1$, $p_i \in V$, and $(p_i, p_{i+1}) \in E$ for each $i < n$ and $p_i \neq p_j$ for $i \neq j$, whenever $p_i, p_j \in \mathcal{A}$. If $p_1 = p_n$, then the weak path is a* weak cycle. *The* length *of a weak path is the number of "$-$" labeled edges $e = (p_i, p_{i+1})$.*

Example 3.5. *The dependency graph $\mathcal{G}_{\mathcal{D}}$ from Example 3.3 contains a weak odd-cycle b, b_c, b of length 1 and a weak even-cycle b, b_c, a_c, a, b_c, b of length 2.*

It is clear that, in contrast to $F_{\mathcal{D}}$, the dependency graph $\mathcal{G}_{\mathcal{D}}$ can be computed in polynomial time. This provides us with the desired computational advantage.

Proposition 3.6. *$\mathcal{G}_{\mathcal{D}}$ can be computed in polynomial time.*

Now we define the classes of ABAFs based on the cycle properties of the underlying dependency graph.

Definition 3.7. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$ be an ABAF and $\mathcal{G}_{\mathcal{D}} = (V, E, l)$ its dependency graph. Then $\mathcal{D}$ is called*

- acyclic *iff $\mathcal{G}_{\mathcal{D}}$ contains no weak cycle of length $n \geq 1$;*
- odd-cycle free *iff $\mathcal{G}_{\mathcal{D}}$ contains no odd-length weak cycle;*
- even-cycle free *iff $\mathcal{G}_{\mathcal{D}}$ contains no even-length weak cycle of length $n \geq 2$.*

From now on we just say "cycle" instead of "weak cycle".

3.2 Computational Properties

The question arises whether or not cycles in $\mathcal{G}_{\mathcal{D}}$ are a weaker, stronger, or equivalent notion compared to cycles in $F_{\mathcal{D}}$. That is, can we lift Fact 3.1 to the underlying dependency graph? It turns out that under mild conditions, the dependency graph $\mathcal{G}_{\mathcal{D}}$ is capable of identifying the cycles we need. In fact, in a certain sense $\mathcal{G}_{\mathcal{D}}$ gives rise to an even a more efficient cycle notion since it detects fewer cycles, but induces all computational properties we seek.

To see that not every cycle in $F_{\mathcal{D}}$ induces a cycle in $\mathcal{G}_{\mathcal{D}}$, we consider the following example.

Example 3.8. *Let $D = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$ be an ABAF with $\mathcal{A} = \{a, b, c, d, e\}$, $\mathcal{L} = \mathcal{A} \cup \{a_c \mid a \in \mathcal{A}\}$, the expected contraries, and the following rules $\mathcal{R}$:*

$$d_c \leftarrow a. \quad b_c \leftarrow a. \quad c_c \leftarrow b. \quad a_c \leftarrow c. \quad e_c \leftarrow d. \quad a_c \leftarrow e.$$

The reader may verify that the induced AF $F_{\mathcal{D}}$ contains an even-cycle, where the dependency graph $\mathcal{G}_{\mathcal{D}}$ does not.

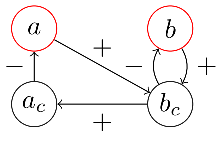
Now let us head to the other direction, i.e., suppose we are given a cycle in $\mathcal{G}_{\mathcal{D}}$. Here it might be the case that the cycle is due to atoms that can never be derived from any assumption. We exclude this case with the following notion.

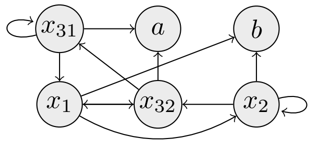
Definition 3.9. *An ABAF $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ is called* trim *if $\mathcal{L} = Th_{\mathcal{D}}(\mathcal{A})$.*

In a trim ABAF, each atom $p \in \mathcal{L}$ can be derived and thus, each rule $r \in \mathcal{R}$ is applicable. Note that a simple unit propagation suffices to turn any ABAF into a trim one in polynomial time. Given a trim $\mathcal{D}$, the following can be inferred.

Proposition 3.10. *If $\mathcal{D}$ is trim and there is a weak cycle of length $n \geq 1$ in $\mathcal{G}_{\mathcal{D}}$, then there is a cycle of length n in $F_{\mathcal{D}}$.*

Example 3.11. *Recall Example 3.3. We depict the instantiated AF $F_{\mathcal{D}}$ next to the dependency graph we already computed. In $F_{\mathcal{D}}$, x_i stems from rule r_i for each $i \in \{1, 2\}$ and x_{31} and x_{32} are due to combining x_1 and x_2 with r_3.*





The even-length cycle a, b_c, b, b_c, a_c, a in the dependency graph $\mathcal{G}_{\mathcal{D}}$ can be traced back to a entailing the contrary of b (via argument x_1) and b entailing the contrary of a (via argument x_{32}). Indeed, x_1 and x_{32} form an even-cycle in $F_{\mathcal{D}}$.

Putting together the previous observations shows that $\mathcal{G}_{\mathcal{D}}$ is more cautious in terms of cycles compared to $F_{\mathcal{D}}$. It is thus not clear whether this notion is strong enough for our purpose, i.e., whether Fact 3.1 can be lifted to ABAFs by means of these graphs. The following theorem answers this question affirmatively.

Theorem 3.12. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ be an ABAF. If $\mathcal{D}$ is*

- *odd-cycle free, then $pr(\mathcal{D}) = stb(\mathcal{D})$;*
- *even-cycle free, then $gr(\mathcal{D}) = co(\mathcal{D}) = pr(\mathcal{D})$;*
- *acyclic, then $gr(\mathcal{D}) = co(\mathcal{D}) = pr(\mathcal{D}) = stb(\mathcal{D})$.*

Intuitively, even-cycles in $\mathcal{G}_{\mathcal{D}}$ induce the non-determinism underlying co, pr, and stb; odd-cycles are responsible for the potential collapse of stb as they serve as constraints for acceptability; and without any cycle whatsoever, many reasoning tasks coincide with the simple gr reasoning.

As a final remark, we want to mention that our notion of cycles is in a certain sense independent of the tree derivations in a given ABAF: while the term "acyclic" in e.g. [Craven and Toni, 2016] is used to ensure that for each tree derivation $S \vdash p$, there is no cyclic dependency between atoms, our dependency graphs $\mathcal{G}_{\mathcal{D}}$ are not tailored to examine such dependencies. To illustrate this, we mention that even for acyclic ABAFs, there might be infinitely many tree-based arguments.

Proposition 3.13. *There is an acyclic ABAF $\mathcal{D}$ that gives rise to infinitely many tree-based arguments.*

To summarize, by inspecting $\mathcal{G}_{\mathcal{D}}$ we can anticipate the class to which $\mathcal{D}$ belongs, without computing $F_{\mathcal{D}}$. This is even the case for situations where $F_{\mathcal{D}}$ might be large, while $\mathcal{G}_{\mathcal{D}}$ on the other hand always has at most $|\mathcal{L}|$ nodes.

4 Partial Evaluations in ABA

We now have a target class of ABAFs for our backdoor algorithm: the idea is to guess the label of assumptions occurring in a cycle in $\mathcal{D}$ and then evaluate the remaining ABAF by propagating the labels. Doing so, however, requires us to have tools to partially evaluate a given ABAF. To this end we define the *reduct* as in the case for AFs [Baumann *et al.*, 2022] to conveniently study our backdoor algorithm in Section 5.

4.1 The ABA Reduct

First we define what we mean by the *range* $S^{\oplus}$ of a set S of assumptions. As in the case of AFs, S^+ is the set of assumptions attacked by S and the range is their union, i.e., we let $S^{\oplus} = S \cup S^+$.

Definition 4.1. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ be an ABAF. For $S \subseteq \mathcal{A}$ we let $S^+_{\mathcal{D}} = \{a \in \mathcal{A} \mid S \text{ attacks } a\}$ and call $S^{\oplus}_{\mathcal{D}} = S \cup S^+_{\mathcal{D}}$ the* range *of S.*

Now given an ABAF $\mathcal{D}$ we define the *reduct* w.r.t. some set S of assumptions in the way that $\mathcal{D}$ is (partially) evaluated w.r.t. S. That is, we assume S to be true and thus evaluate S^+ as false. Hence assumptions in S can be removed from rule bodies, whereas rules relying on assumptions in S^+ can be deleted entirely. This amounts to the following notion.

Definition 4.2. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ be an ABAF. The S-*reduct *of $\mathcal{D}$ is the ABAF $\mathcal{D}^S = (\mathcal{L}', \mathcal{R}', \mathcal{A}', ^{-'})$ where $\mathcal{L}' = \mathcal{L} \setminus S^{\oplus}_{\mathcal{D}}$, $\mathcal{A}' = \mathcal{A} \setminus S^{\oplus}_{\mathcal{D}}$,*

$$\mathcal{R}' = \{head(r) \leftarrow body(r) \setminus S \mid r \in \mathcal{R},\ body(r) \cap S^+_{\mathcal{D}} = \emptyset\},$$

and for $a \in \mathcal{A}'$, $\overline{a}' = \overline{a}$ iff $\overline{a} \in \mathcal{L}'$ and UNDEFINED *otherwise.*

Example 4.3. *Consider the ABAF $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ with $\mathcal{A} = \{a, b, c\}$, $\mathcal{L} = \mathcal{A} \cup \{a_c, b_c, c_c\}$, the expected contraries, and the following rules $\mathcal{R}$:*

$$r_1 : b_c \leftarrow a \qquad r_2 : a_c \leftarrow b \qquad r_3 : a_c \leftarrow c$$

Let $S = \{c\}$. In the reduct $\mathcal{D}^S$, c is set to true and since $a_c \in Th_{\mathcal{D}}(S)$, we set a to false. This yields the ABAF $\mathcal{D}^S$ given as $\mathcal{D}^S = (\{b, a_c, b_c, c_c\}, \mathcal{R}', \{b\}, ^{-'})$ with rules $\mathcal{R}' = \{r'_2 = (a_c \leftarrow b),\ r'_3 = (a_c \leftarrow)\}$. The only assumption is b.

4.2 The Modularization Property

Our backdoor algorithm requires us to evaluate a given ABAF partially and then compute a single assumption set S. The technical underpinning we will develop to this end is inspired by the modularization property for AFs [Baumann *et al.*, 2022]. In the previous Example 4.3, the assumption b is unattacked in the reduct $\mathcal{D}^S$ since a is rendered false (and $\mathcal{D}^S$ has been evaluated accordingly). Assuming that $S = \{c\}$ is compatible with b in some formal sense, we would expect that b can be added to S in order to obtain a novel extension $S \cup \{b\}$. Indeed, $S \cup \{b\} \in co(\mathcal{D})$ is true.

In the context of AFs, the so-called *modularization property* has been introduced in order to capture such step-wise computations of extensions [Baumann *et al.*, 2022]. Phrased within our setting, we obtain the following notion for ABAFs.

Definition 4.4. *A semantics σ satisfies the* modularization property *if for each ABAF $\mathcal{D}$, it holds that $S \in \sigma(\mathcal{D})$ and $S' \in \sigma(\mathcal{D}^S)$ implies $S \cup S' \in \sigma(D)$.*

Example 4.5. *Recall Example 4.3. We have that $S = \{c\} \in ad(\mathcal{D})$ and $S' = \{b\} \in ad(\mathcal{D}^S)$. Moreover, $S \cup S' \in ad(\mathcal{D})$ so the requirements for ad to satisfy modularization are satisfied for this ABAF.*

It turns out that this observation is no coincidence. In fact, like for AFs, all considered semantics satisfy modularization.

Proposition 4.6. *Each semantics $\sigma \in \{ad, co, pr, gr, stb\}$ satisfies modularization.*

For ad and co semantics, we can even show some kind of inverse statement, in the following sense.

Proposition 4.7. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$ be an ABAF and let $\sigma \in \{ad, co\}$. If $S = S' \cup S''$ with $S' \cap S'' = \emptyset$ and $S' \in \sigma(\mathcal{D})$ as well as $S \in \sigma(\mathcal{D})$, then $S'' \in \sigma(\mathcal{D}^{S'})$.*

Example 4.8. *Recall Example 4.3. We have that $S = \{b, c\} \in ad(\mathcal{D})$. Since $S' = \{c\} \in ad(\mathcal{D})$ we can be certain that $\{b\} \in ad(\mathcal{D}^{S'})$.*

Combining Propositions 4.6 and 4.7 yields the following characterizations of the standard ABA semantics.

Theorem 4.9. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$ be an ABAF, $S \in cf(\mathcal{D})$ and $\mathcal{D}^S = (\mathcal{L}', \mathcal{R}', \mathcal{A}', \overline{}')$ the reduct. It holds that*

- *$S \in stb(\mathcal{D})$ iff $\mathcal{D}^S = (\mathcal{L}', \mathcal{R}', \emptyset, \overline{}')$,*
- *$S \in ad(\mathcal{D})$ iff T attacks S in $\mathcal{D}$ implies $T \setminus S \not\subseteq \mathcal{A}'$,*
- *$S \in pr(\mathcal{D})$ iff $S \in ad(\mathcal{D})$ und $\bigcup ad(\mathcal{D}^S) = \emptyset$, and*
- *$S \in co(\mathcal{D})$ iff $S \in ad(\mathcal{D})$ und no assumption in $\mathcal{D}^S$ is unattacked.*

As a final remark in this section, we recall the characterization [Baumann and Ulbricht, 2021b, Theorem 5.7] for the existence of a stable extension: There is a stable extension $E \in stb(F)$ iff there is some admissible extension $E' \in ad(F)$ s.t. E' attacks each odd-cycle in F. This translates to ABA in the following way.

Theorem 4.10. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$ be a trim ABAF. Then $stb(\mathcal{D}) \neq \emptyset$ iff there is some $S \in ad(\mathcal{D})$ s.t. each assumption a occurring in an odd-cycle in $\mathcal{G}_\mathcal{D}$ satisfies $a \in S^\oplus$.*

5 Breaking Cycles: A Backdoor Algorithm

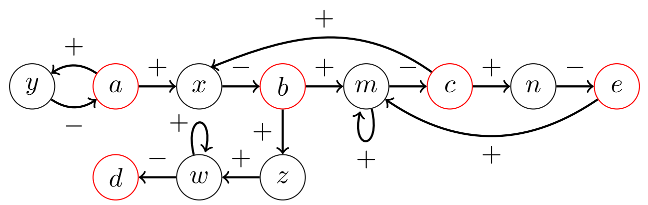
We are ready to design our backdoor algorithm. Our idea is inspired by the recent work on backdoor algorithms for SETAFs [Dvorák *et al.*, 2022b] which are similar in spirit to ABA [König *et al.*, 2022]. We will guess a set B of assumptions in such a way that after their acceptance status is evaluated, the remaining ABAF is acyclic. Since acyclic ABAFs only have a unique complete extension, we can propagate the effect of this evaluation through the framework. We then have to check whether this propagation is compatible with our initial guess (implying we found an extension) or not (implying we have to backtrack).

We consider the following example to illustrate this idea.

Example 5.1. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$ be the ABAF with $\mathcal{A} = \{a, b, c, d, e\}$, $\mathcal{L} = \mathcal{A} \cup \{x, y, w, z, m, n\}$, contraries $\overline{a} = y$, $\overline{b} = x$, $\overline{c} = m$, $\overline{d} = w$, $\overline{e} = n$, and rules $\mathcal{R}$:*

$$x \leftarrow a. \quad y \leftarrow a. \quad z \leftarrow b. \quad w \leftarrow z. \quad w \leftarrow w.$$
$$m \leftarrow b. \quad x \leftarrow c. \quad m \leftarrow e. \quad m \leftarrow m. \quad n \leftarrow c.$$

The dependency graph is given as follows.



From the dependency graph we can extract that e.g. removing b and e renders the ABAF even-cycle free. That is, if we fix the labels of these two assumptions, then the non-determinism is removed from $\mathcal{D}$. Consequently, the remaining labels follow by propagating the effect of guessing b and e.

Suppose e.g. we set b and e to true. Then, by applying the given rules, z is entailed and thus w as well which is the contrary of d; hence it follows that d is defeated. Moreover, from e we entail m which means c is rendered defeated as well. Since c is the only attacker of e (by entailing n), we obtain that $\{b, e\}$ indeed defends e. Nonetheless, $\{b, e\}$ does not induce an admissible extension since the attack from a against b (by entailing x) cannot be defended. We thus have to correct the status of b and remove it from our extension again. This, however, has no effect on d and we find the admissible extension $E = \{e\}$.

The idea is thus to find a set of assumptions breaking all cycles in $\mathcal{D}$ (called a *backdoor*), guessing their acceptance status and then propagating the consequences of this guess. Due to our results from Theorem 3.12 we know that the non-determinism is due to the even-cycles in $\mathcal{G}_\mathcal{D}$ and Section 4 provides us with convenient tools to partially evaluate $\mathcal{D}$.

5.1 Theoretical Foundations

Our procedure works by propagating labels on assumptions, denoting whether they are in an extension ("in"), attacked by the given extension ("out"), or neither ("undec"). To this end we need to formally introduce labels for ABA. We will only recall the necessary technical tools here; a comprehensive discussion of labeling-based semantics in ABA can be found in [Schulz and Toni, 2017].

Definition 5.2. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$ be an ABAF. A labeling is a mapping $\lambda : \mathcal{A} \to \{\textsc{In}, \textsc{Out}, \textsc{Undec}\}$. We let $\textsc{In}(\lambda) = \{a \in \mathcal{A} \mid \lambda(a) = \textsc{In}\}$, $\textsc{Out}(\lambda) = \{a \in \mathcal{A} \mid \lambda(a) = \textsc{Out}\}$, $\textsc{Undec}(\lambda) = \{a \in \mathcal{A} \mid \lambda(a) = \textsc{Undec}\}$, and $\textsc{Def}(\lambda) = \textsc{In}(\lambda) \cup \textsc{Out}(\lambda) \cup \textsc{Undec}(\lambda)$.*

A complete labeling is defined s.t. it corresponds to some complete extension in the expected way [Schulz and Toni, 2017, Theorem 4].

Definition 5.3. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$ be an ABAF. A labeling λ is called* complete *whenever:*

- *if $a \in \textsc{In}(\lambda)$, then for each set $S \subseteq \mathcal{A}$ attacking a, it holds that $S \cap \textsc{Out}(\lambda) \neq \emptyset$;*
- *if $a \in \textsc{Out}(\lambda)$, then there is some $S \subseteq \mathcal{A}$ attacking a with $S \subseteq \textsc{In}(\lambda)$;*
- *if $a \in \textsc{Undec}(\lambda)$, then i) for each set $S \subseteq \mathcal{A}$ attacking a, it holds that some $b \in S$ satisfies $b \notin \textsc{In}(\lambda)$ and ii) there is some $S' \subseteq \mathcal{A}$ attacking a s.t. $S' \cap \textsc{Out}(\lambda) = \emptyset$.*

A labeling λ is called preferred *if there is no complete labeling λ' s.t. $\textsc{In}(\lambda) \subsetneq \textsc{In}(\lambda')$.*

Since our backdoor algorithm is based on partially evaluating $\mathcal{D}$, we require the notion of a partial labeling.

Definition 5.4. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ })$ be an ABAF. A partial labeling is a partial mapping $\lambda : \mathcal{A} \to \{\textsc{In}, \textsc{Out}, \textsc{Undec}\}$.*

Now we are ready to describe our procedure. Our first step will be to select a set B of assumptions in a way that $\mathcal{G}_\mathcal{D}$ belongs to a certain target class of graphs whenever the nodes corresponding to B are removed. For this we define the restriction of a dependency graph $\mathcal{G}_\mathcal{D}$ in the natural way.

Definition 5.5. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ })$ be an ABAF and $\mathcal{G}_\mathcal{D} = (V, E, l)$ its dependency graph. Then $(\mathcal{G}_\mathcal{D})\downarrow_{V\setminus B} = (V', E', l')$ is given via $V' = V \setminus B$, $E' = E \cap (V' \times V')$, and $l'(e) = l(e)$ for each $e \in E'$.*

Now B is called a *$\mathcal{C}$-backdoor* whenever $(\mathcal{G}_\mathcal{D})\downarrow_{V\setminus B}$ belongs to the class $\mathcal{C}$ of graphs, i.e., if removing B from $\mathcal{G}_\mathcal{D}$ turns it into a graph with the desired properties.

Definition 5.6. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ })$ be an ABAF and let $\mathcal{C}$ be a class of graphs. A set $B \subseteq \mathcal{A}$ of assumptions is called a $\mathcal{C}$*-backdoor *if $(\mathcal{G}_\mathcal{D})\downarrow_{V\setminus B}$ belongs to the class $\mathcal{C}$ of graphs.*

By ACYC_{DG} we denote the class of acyclic dependency graphs and by $\text{NOEVEN}_{\text{DG}}$ the class of no-even ones.

Example 5.7. *In Example 5.1, $B = \{b, e\}$ is a $\text{NOEVEN}_{\text{DG}}$-backdoor since $(\mathcal{G}_\mathcal{D})\downarrow_{V\setminus B}$ does not contain any even-cycle. It is, however, not an ACYC_{DG}-backdoor since it still contains the self-attacker a.*

We are now ready to formalize how to treat the ABAF $\mathcal{D}$ after we guessed the labels on the backdoor B. Recall from Example 5.1 that we propagated the consequences of our guess. In general, we apply the following rules.

Definition 5.8. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ })$ be an ABAF and let λ be a partial labeling. We define λ^* via*

1. *set $\lambda^*(a) = \textsc{Out}$ if $\overline{a} \in Th_\mathcal{D}(\textsc{In}(\lambda^*))$ and*
2. *set $\lambda^*(a) = \textsc{In}$ if $\overline{a} \notin Th_\mathcal{D}(\mathcal{A} \setminus \textsc{Out}(\lambda^*))$.*

We denote the result of applying these rules to λ (i.e., for each $a \in \mathcal{A} \setminus \textsc{Def}(\lambda)$) until a fixed point is reached as PROPAGATEIO*($\mathcal{D}$,λ). Then we define $\lambda^\dagger$ via*

3. *set $\lambda^\dagger(a) = \textsc{Undec}$ if $\lambda^\dagger(a) = \textsc{In}$ and $\overline{a} \in Th_\mathcal{D}(\mathcal{A} \setminus OUT(\lambda^\dagger))$ and*
4. *set $\lambda^\dagger(a) = \textsc{Undec}$ if $\lambda^\dagger(a) = \textsc{Out}$ and $\overline{a} \notin Th_\mathcal{D}(IN(\lambda^\dagger))$.*

We denote the result of applying these rules to λ (i.e., for each $a \in \textsc{In}(\lambda) \cup \textsc{Out}(\lambda)$) until a fixed point is reached as PROPAGATEUNDEC*($\mathcal{D}$,λ).*

Intuitively, the first rule simply formalizes that an assumption a must be labelled out whenever its contrary can be entailed from the assumptions we already assume to be true (e.g. in Example 5.1 d being labelled $\textsc{Out}$ is a consequence of b being labelled $\textsc{In}$). In the same vein, if the contrary of a can only be deduced from assumptions that are already labelled $\textsc{Out}$, then a can surely be labelled $\textsc{In}$. Then, rules 3. and 4. are necessary in case our guess on the backdoor assumptions is not consistent with the performed propagation (for instance, if we guess a to be true in Example 5.1 it will turn out that this must be corrected as it attacks itself). Thus, rule 3. corrects labels from $\textsc{In}$ to $\textsc{Undec}$ if necessary; and finally, rule 4. corrects wrongly labelled $\textsc{Out}$ assumptions.

The induced procedure can be found in Algorithm 1. The following theorem formalizes that Algorithm 1 finds all preferred extensions of the input ABAF $\mathcal{D}$. It also finds some complete sets which are not preferred, which is why $pr^*(\mathcal{D}, B)$ does not coincide with $pr(\mathcal{D})$.

Algorithm 1 Computing Admissible Candidates

```
Input: ABAF $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ })$, $\text{NOEVEN}_{\text{DG}}$-backdoor $B$
Output: admissible candidates of $D$: $pref^*(D, B)$
 1: procedure GETADMISSIBLECANDIDATES($\mathcal{D}$,$B$)
 2:     $pref^*(\mathcal{D}, B) = \emptyset$
 3:     foreach $I \subseteq B$ do
 4:         let $\lambda$ be a partial labeling
 5:         set $\lambda(a) = \textsc{In}$ for $a \in I$
 6:         set $\lambda(a) = \textsc{Out}$ for $a \in B \setminus I$
 7:         $\lambda^* = $ PROPAGATEIO($\mathcal{D}$,$\lambda$)
 8:         set $\lambda^*(a) = \textsc{Undec}$ for $a \in \mathcal{A} \setminus \textsc{Def}(\lambda^*)$
 9:         $\lambda^\dagger = $ PROPAGATEUNDEC($\mathcal{D}$,$\lambda^*$)
10:         if $\textsc{In}(\lambda^\dagger) \cap B == I$ then
11:             $pref^*(\mathcal{D}, B) = pref^*(\mathcal{D}, B) \cup \{\textsc{In}(\lambda^\dagger)\}$
12:         end if
13:     end for
14:     return $pref^*(D, B)$
15: end procedure
```

Theorem 5.9. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ })$ be a trim ABAF and $\mathcal{G}_\mathcal{D}$ be its dependency graph. Let $\mathcal{C} \in \{\text{ACYC}_{\text{DG}}, \text{NOEVEN}_{\text{DG}}\}$ and let B be a $\mathcal{C}$-backdoor. For the output $pr^*(\mathcal{D}, B)$ of Algorithm 1 it holds that $pr(\mathcal{D}) \subseteq pr^*(\mathcal{D}, B)$.*

We now establish that this procedure is fixed-parameter tractable for ACYC_{DG}-backdoors. First of all, the algorithm itself runs in time $2^{|B|} \times \mathcal{O}(p(n))$ for some polynomial p, independent of whether B is a ACYC_{DG}-backdoor or a $\text{NOEVEN}_{\text{DG}}$-backdoor.

Proposition 5.10. *Algorithm 1 runs in time $2^k \times \mathcal{O}(p(n))$ where p is some polynomial, n is the size of the input ABAF, and k the size of the (ACYC_{DG}/$\text{NOEVEN}_{\text{DG}}$)-backdoor B.*

In Algorithm 1, the backdoor B is given. Thus the question arises whether or not a backdoor can be computed efficiently. Due to our dependency graph results, this boils down to finding such backdoors in directed graphs. For $\text{NOEVEN}_{\text{DG}}$-backdoors the fastest known procedure requires time $n^{\mathcal{O}(k)}$ [Dvorák *et al.*, 2012a] (where k is the backdoor size). For ACYC_{DG}-backdoors, however, we find the following.

Proposition 5.11. *Let $\mathcal{D}$ be an ABAF, k be an integer, p be some polynomial and n be size of $\mathcal{D}$. There is a function f s.t. we can find an ACYC_{DG}-backdoor of size at most k in time $f(k) \times p(n)$ or detect if no such backdoor exists.*

As a corollary, we obtain that skeptical reasoning for preferred semantics is fixed-parameter tractable in ABA w.r.t. the size of the smallest ACYC_{DG}-backdoor of $\mathcal{D}$.

Corollary 5.12. *Skeptical acceptance for $\sigma \in \{pr, stb\}$ of a query assumption is fixed-parameter tractable w.r.t. the size of the smallest ACYC_{DG}-backdoor of an ABAF.*

5.2 Empirical Evaluation

We now discuss some computational shortcuts we utilize for our actual implementation. We first note that before delving into the cycles of our given ABAF, we can compute the grounded extension G of $\mathcal{D}$ as well as the reduct $\mathcal{D}^G$, and then remove unnecessary atoms and rules in order to make it trim. Soundness of this is a corollary of Proposition 4.6 and the fact that $G \subseteq E$ holds for each preferred extension $E \in pr(\mathcal{D})$.

Corollary 5.13. *Let $\mathcal{D}$ be an ABAF and $G \in gr(\mathcal{D})$. Then $E \in pr(\mathcal{D})$ iff $E \setminus G \in pr(\mathcal{D}^G)$.*

This way we can apply our algorithm to $\mathcal{D}^G$ which admits smaller backdoors in general compared to $\mathcal{D}$.

We also consider two further shortcuts in Algorithm 1: First, if line 9 labels one of the assumptions in I as UNDEC, we can immediately skip the guess (due to line 10). Second, if line 9 does not correct any label in B to UNDEC, then we know that the propagation in line 7 must have been correct as well. So we always iterate through the backdoor B first and if no label is changed in line 9, we can immediately return the IN labeled assumptions.

Setup. We implemented the backdoor approach in Python. We used the ASP-solver Clingo [Gebser *et al.*, 2018] to compute a minimal ACYC$_{\mathrm{DG}}$-backdoor. Since we have to compute the deductive closure $Th_D(S)$ several times for different assumption sets S, we utilized the Glucose SAT-solver [Audemard and Simon, 2018] based on MiniSat [Eén and Sörensson, 2004]. This SAT-solver is accessed through the PySAT library [Ignatiev *et al.*, 2018].

Computations were done using resources of the Leipzig University Computing Center. There we used the Paul Cluster with a memory limit of 32GB ram and with the CPU: 2x AMD EPYC 7713 @ 2.0GHz - Turbo 3.7GHz (64 cores). As experimental data, 400 ABAFs from the ICCMA 2023 competition benchmark generator[2] were used. The ABA track was won convincingly by the ASPforABA solver [Lehtonen *et al.*, 2021] against which we evaluated our backdoor algorithm. As problem, we enumerated all preferred extensions for some arbitrary ABAF $\mathcal{D}$. A timeout of 90s is used.

Results. Our evaluation shows that, as expected, the performance of the backdoor algorithm is quite sensitive to the size of the smallest ACYC$_{\mathrm{DG}}$. We divided the instances into three groups, backdoor size between 0 and 5, between 6 and 10, and greater than 10. Most of the 400 instances (220) belong to the first group with small backdoor sizes in which the backdoor procedure outperforms the state-of-the-art ASPforABA solver (first row in Table 1). There are only few instances with backdoor size $5-10$. Both algorithms are quite fast in these instances, but ASPforABA is better by a small margin (second row). For larger backdoor sizes, however, the ASPforABA solver is much better due to the large search space the backdoor algorithm has to handle here (last row).

	Approach	Av.	Total	Solved Instances
$k \leq 5$	Backdoor	0.17	37.29	100% (220/220)
	ASPforABA	1.28	282.30	100% (220/220)
$k \in [6, 10]$	Backdoor	0.17	4.08	100% (24/24)
	ASPforABA	0.06	1.46	100% (24/24)
$11 \leq k$	Backdoor	89.31	13932.87	1.28% (2/156)
	ASPforABA	22.40	3494.80	80.13% (125/156)

Table 1: Runtimes of our Algorithm 1 and ASPforABA in s

6 Conclusion

We developed and implemented an FPT-algorithm for enumerating preferred extensions in ABA. To this end we considered acyclic *backdoors* w.r.t. a suitable dependency graph notion. The main theoretical observation for this algorithm is that preferred reasoning in ABAFs is fixed-parameter tractable w.r.t. the size of the smallest ACYC$_{\mathrm{DG}}$-backdoor. More broadly, our study shows that the dependency graph can be utilized to examine computational beneficial sub-classes of ABA, similar in spirit to AFs (see e.g. [Dvořák and Dunne, 2018]). This is in contrast to common cycle notions for ABA focusing on tree derivations [Craven and Toni, 2016], but similar in spirit to logic programming dependency graphs [Apt and Bol, 1994]. Our empirical evaluation demonstrates that our procedure is competitive with state-of-the-art solvers up until an acyclic backdoor size of around 10, which is the case for a significant proportion of the ICCMA'23 instances.

Exploiting graph-specific properties for incremental computations has also been done for AFs [Baroni *et al.*, 2005; Baumann and Ulbricht, 2021a; Alfano *et al.*, 2023; Liao, 2013; Liao, 2014]. For instance in [Liao, 2013; Liao, 2014] the AF is divided into sub-graphs in order to utilize the fact that these sub-graphs might fall into certain computationally beneficial fragments. Our modularization notion does not make explicit use of the topology of the dependency graph, but the backdoor algorithm could benefit from a similar decomposition of the knowledge base in order to exploit properties of its subsets. The dependency graph lays the foundation for such a procedure in the context of ABA.

Similar in spirit to our work is [Alfano *et al.*, 2021] in the context of defeasible logic programming. In order to analyze dynamic reasoning environments, a hypergraph is used to examine the dependency of literals within the given knowledge base. This work differs from ours as notions such as "acyclic" or "no-even" knowledge bases are not studied to gain a computational boost, because the focus is on (efficiency of) dynamic reasoning. Our work, however, is a potential first step towards pushing such an analysis into the realm of ABA.

In future work, our theoretical analysis could be continued by considering further ABA classes and theoretical properties induced by dependency graphs. For instance, research for AFs based on the topology of the given argumentation graph can now be rigorously extended to ABA, e.g. the directionality principle [Baroni and Giacomin, 2007] or semantics based on strongly connected components SCCs [Baroni *et al.*, 2005]. Since the backdoor size appears to be the bottleneck for our procedure, another natural avenue for future work consist in adjusting it to NOEVEN$_{\mathrm{DG}}$-backdoors. Here, the main challenge will be finding the backdoor efficiently. Moreover, our approach could be generalized to non-flat ABAFs or other rule-based formalisms like ASPIC+ [Modgil and Prakken, 2013].

[2]https://iccma2023.github.io/benchmarks.html

Acknowledgements

This word was funded by the Federal Ministry of Education and Research of Germany and by Sächsische Staatsministerium für Wissenschaft, Kultur und Tourismus in the programme Center of Excellence for AI-research "Center for Scalable Data Analytics and Artificial Intelligence Dresden/Leipzig", project identification number: ScaDS.AI.

References

[Alfano *et al.*, 2021] Gianvincenzo Alfano, Sergio Greco, Francesco Parisi, Gerardo I. Simari, and Guillermo Ricardo Simari. Incremental computation for structured argumentation over dynamic delp knowledge bases. *Artif. Intell.*, 300:103553, 2021.

[Alfano *et al.*, 2023] Gianvincenzo Alfano, Marco Calautti, Sergio Greco, Francesco Parisi, and Irina Trubitsyna. Explainable acceptance in probabilistic and incomplete abstract argumentation frameworks. *Artif. Intell.*, 323:103967, 2023.

[Apt and Bol, 1994] Krzysztof R. Apt and Roland N. Bol. Logic programming and negation: A survey. *J. Log. Program.*, 19/20:9–71, 1994.

[Atkinson *et al.*, 2017] Katie Atkinson, Pietro Baroni, Massimiliano Giacomin, Anthony Hunter, Henry Prakken, Chris Reed, Guillermo Ricardo Simari, Matthias Thimm, and Serena Villata. Towards artificial argumentation. *AI Magazine*, 38(3):25–36, 2017.

[Audemard and Simon, 2018] Gilles Audemard and Laurent Simon. On the glucose sat solver. *International Journal on Artificial Intelligence Tools*, 27(01):1840001, 2018.

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

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

[Baroni *et al.*, 2011] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *The Knowledge Engineering Review*, 26:365–410, 2011.

[Baroni *et al.*, 2018] Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors. *Handbook of Formal Argumentation*. College Publications, 2018.

[Baumann and Ulbricht, 2019] Ringo Baumann and Markus Ulbricht. If nothing is accepted - repairing argumentation frameworks. *J. Artif. Intell. Res.*, 66:1099–1145, 2019.

[Baumann and Ulbricht, 2021a] Ringo Baumann and Markus Ulbricht. Choices and their consequences - explaining acceptable sets in abstract argumentation frameworks. In *Proc. KR*, pages 110–119, 2021.

[Baumann and Ulbricht, 2021b] Ringo Baumann and Markus Ulbricht. On cycles, attackers and supporters - A contribution to the investigation of dynamics in abstract argumentation. In *Proc. IJCAI*, pages 1780–1786. ijcai.org, 2021.

[Baumann *et al.*, 2020] Ringo Baumann, Gerhard Brewka, and Markus Ulbricht. Comparing Weak Admissibility Semantics to their Dung-style Counterparts – Reduct, Modularization, and Strong Equivalence in Abstract Argumentation. In *Proc. KR*, pages 79–88, 9 2020.

[Baumann *et al.*, 2022] Ringo Baumann, Gerhard Brewka, and Markus Ulbricht. Shedding new light on the foundations of abstract argumentation: Modularization and weak admissibility. *Artif. Intell.*, 310:103742, 2022.

[Baumann, 2011] Ringo Baumann. Splitting an argumentation framework. In *Proc. LPNMR*, volume 6645 of *Lecture Notes in Computer Science*, pages 40–53. Springer, 2011.

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

[Besnard *et al.*, 2014] Philippe Besnard, Alejandro Garcia, Anthony Hunter, Sanjay Modgil, Henry Prakken, Guillermo Simari, and Francesca Toni. Introduction to structured argumentation. *Argument Comput.*, 5(1):1–4, 2014.

[Bondarenko *et al.*, 1997] Andrei Bondarenko, Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artif. Intell.*, 93:63–101, 1997.

[Craven and Toni, 2016] Robert Craven and Francesca Toni. Argument graphs and assumption-based argumentation. *Artif. Intell.*, 233:1–59, 2016.

[Čyras *et al.*, 2018] Kristijonas Čyras, Xiuyi Fan, Claudia Schulz, and Francesca Toni. Assumption-based argumentation: Disputes, explanations, preferences. In *Handbook of Formal Argumentation*, chapter 7, pages 365–408. College Publications, 2018.

[Cyras *et al.*, 2021] Kristijonas Cyras, Antonio Rago, Emanuele Albini, Pietro Baroni, and Francesca Toni. Argumentative XAI: A survey. In *Proc. IJCAI*, pages 4392–4399. ijcai.org, 2021.

[Dung, 1995] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–357, 1995.

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

[Dvorák and Dunne, 2018] Wolfgang Dvorák and Paul E. Dunne. Computational problems in formal argumentation and their complexity. In *Handbook of Formal Argumentation*. College Publications, February 2018.

[Dvorák *et al.*, 2012a] Wolfgang Dvorák, Sebastian Ordyniak, and Stefan Szeider. Augmenting tractable fragments of abstract argumentation. *Artif. Intell.*, 186:157–173, 2012.

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

[Dvorák *et al.*, 2022a] Wolfgang Dvorák, Markus Hecher, Matthias König, André Schidler, Stefan Szeider, and Stefan Woltran. Tractable abstract argumentation via backdoor-treewidth. In *Proc. AAAI*, pages 5608–5615. AAAI Press, 2022.

[Dvorák *et al.*, 2022b] Wolfgang Dvorák, Matthias König, and Stefan Woltran. Deletion-backdoors for argumentation frameworks with collective attacks. In *Proc. SAFA*, volume 3236 of *CEUR Workshop Proceedings*, pages 98–110. CEUR-WS.org, 2022.

[Eén and Sörensson, 2004] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing*, pages 502–518, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Falappa *et al.*, 2009] Marcelo Alejandro Falappa, Gabriele Kern-Isberner, and Guillermo Ricardo Simari. Belief revision and argumentation theory. In *Argumentation in Artificial Intelligence*, pages 341–360. 2009.

[García and Simari, 2004] Alejandro Javier García and Guillermo Ricardo Simari. Defeasible logic programming: An argumentative approach. *Theory Pract. Log. Program.*, 4(1-2):95–138, 2004.

[Gebser *et al.*, 2018] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot asp solving with clingo, 2018.

[Haret *et al.*, 2018] Adrian Haret, Johannes Peter Wallner, and Stefan Woltran. Two sides of the same coin: Belief revision and enforcing arguments. In *Proc. IJCAI*, pages 1854–1860, 2018.

[Ignatiev *et al.*, 2018] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018.

[König *et al.*, 2022] Matthias König, Anna Rapberger, and Markus Ulbricht. Just a matter of perspective. In *Proc. COMMA*, volume 353 of *Frontiers in Artificial Intelligence and Applications*, pages 212–223. IOS Press, 2022.

[Lehtonen *et al.*, 2021] Tuomo Lehtonen, Johannes Peter Wallner, and Matti Järvisalo. Harnessing incremental answer set solving for reasoning in assumption-based argumentation. *CoRR*, abs/2108.04192, 2021.

[Lehtonen *et al.*, 2023] Tuomo Lehtonen, Anna Rapberger, Markus Ulbricht, and Johannes Peter Wallner. Argumentation frameworks induced by assumption-based argumentation: Relating size and complexity. In *Proc. KR*, pages 440–450, 2023.

[Liao *et al.*, 2018] Beishui Liao, Kang Xu, and Huaxin Huang. Formulating semantics of probabilistic argumentation by characterizing subgraphs: theory and empirical results. *J. Log. Comput.*, 28(2):305–335, 2018.

[Liao, 2013] Beishui Liao. Toward incremental computation of argumentation semantics: A decomposition-based approach. *Ann. Math. Artif. Intell.*, 67(3-4):319–358, 2013.

[Liao, 2014] Beishui Liao. *Efficient Computation of Argumentation Semantics*. Intelligent systems series. Academic Press, 2014.

[Modgil and Prakken, 2013] Sanjay Modgil and Henry Prakken. A general account of argumentation with preferences. *Artif. Intell.*, 195:361–397, 2013.

[Niskanen *et al.*, 2018] Andreas Niskanen, Johannes Peter Wallner, and Matti Järvisalo. Extension enforcement under grounded semantics in abstract argumentation. In *Proc. KR*, pages 178–183, 2018.

[Popescu and Wallner, 2023] Andrei Popescu and Johannes Peter Wallner. Reasoning in assumption-based argumentation using tree-decompositions. In *Proc. JELIA*, volume 14281 of *Lecture Notes in Computer Science*, pages 192–208. Springer, 2023.

[Rapberger *et al.*, 2022] Anna Rapberger, Markus Ulbricht, and Johannes Peter Wallner. Argumentation frameworks induced by assumption-based argumentation: Relating size and complexity. In *Proc. NMR*, volume 3197 of *CEUR Workshop Proceedings*, pages 92–103. CEUR-WS.org, 2022.

[Schulz and Toni, 2017] Claudia Schulz and Francesca Toni. Labellings for assumption-based and abstract argumentation. *Int. J. Approx. Reason.*, 84:110–149, 2017.

[Wallner *et al.*, 2017] Johannes Peter Wallner, Andreas Niskanen, and Matti Järvisalo. Complexity results and algorithms for extension enforcement in abstract argumentation. *J. Artif. Intell. Res.*, 60:1–40, 2017.