Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
KR and Constraints

Counterexample-Guided Abstraction Refinement for Assumption-based Argumentation

Jean-Marie Lagniez[1] , **Emmanuel Lonca**[1] , **Jean-Guy Mailly**[2]
[1]CRIL, Université d'Artois & CNRS
[2]IRIT, Université Toulouse Capitole & CNRS
{lagniez, lonca}@cril.fr, jean-guy.mailly@irit.fr

Abstract

Assumption-Based Argumentation (ABA) is a prominent formalism for structured argumentation, widely applied in domains such as healthcare, law, and robotics. Despite its inherent computational complexity, ABA has seen the development of effective techniques that successfully address key tasks, including evaluating the acceptability of literals and computing framework extensions. These approaches typically involve translating the initial ABA framework into an intermediate formalism, such as an Answer Set Program or an Abstract Argumentation Framework, which is then encoded into a Boolean satisfiability (SAT) problem. However, this translation can lead to large and complex intermediate representations, posing challenges for state-of-the-art SAT solvers. In this work, we propose a Counterexample-Guided Abstraction Refinement (CEGAR) approach that bypasses the initial translation step, at the cost of incrementally discovering certain ABA constraints that are not explicitly captured in the initial SAT encoding. We analyze the performance of our method and demonstrate that it outperforms state-of-the-art approaches on specific problem classes, while remaining competitive with the best existing solvers more broadly.

1 Introduction

Formal Argumentation (Baroni et al. 2018) focuses on representing arguments and their interactions, as well as developing reasoning techniques for assessing their acceptability. Several structured argumentation frameworks have been proposed (Besnard et al. 2014), typically defining arguments and attacks through logic-based methods. *Assumption-Based Argumentation* (ABA) is one such framework (Bondarenko et al. 1997; Toni 2014). An *Assumption-Based Argumentation Framework* (ABAF) consists of a set of inference rules over literals, a subset of which are designated as *assumptions*. Classical extension-based semantics (Dung 1995) identify sets of assumptions that are collectively acceptable, thereby yielding alternative "solutions" to the problem encoded by the ABAF. ABA subsumes several non-monotonic reasoning formalisms (Bondarenko et al. 1997) and has been successfully applied in diverse domains, including legal reasoning (Dung, Thang, and Hung 2010), healthcare (Fan et al. 2013; Cyras et al. 2021), and robotics (Fan et al. 2016).

The development of real-world applications based on argumentation formalisms requires efficient methods for solving hard instances of the underlying reasoning problems. Many such problems, including those arising in ABA, are known to be complete for the first or second levels of the polynomial hierarchy (Dvorák and Dunne 2018; Lehtonen, Wallner, and Järvisalo 2021a). Despite this high theoretical complexity, the International Competition on Computational Models of Argumentation (ICCMA) (Thimm 2025) has driven the emergence of highly effective computational techniques, initially for abstract argumentation frameworks (i.e., Dung's frameworks) (Dung 1995), and more recently (starting from the 6$^{\text{th}}$ edition in 2023) for ABAFs.

In the 2023 edition, several solvers competed in the ABA track, including ACBAR, ASPFORABA, ASTRA, CRUSTABRI, and FLEXABLE (Järvisalo, Lehtonen, and Niskanen 2025). Across all evaluated problems, ASPFORABA (Lehtonen, Wallner, and Järvisalo 2021a; Lehtonen, Wallner, and Järvisalo 2021b) achieved first place, followed by ACBAR (Lehtonen et al. 2023b) in second. As its name suggests, ASPFORABA is based on Answer Set Programming (ASP) (Gebser et al. 2012), while ACBAR transforms the ABA framework into an abstract argumentation framework, subsequently leveraging a state-of-the-art abstract argumentation solver. This transformation is carefully designed to mitigate the risk of exponential blowup.

While ASPFORABA and ACBAR have demonstrated strong performance, both approaches may be hindered by preprocessing steps that degrade efficiency on certain pathological instances. For ASPFORABA, the main bottleneck when running the underlying ASP solver lies in the grounding phase (Kaufmann et al. 2016) of the logic program derived from the ABA framework. Despite recent advances in grounding techniques (Besin, Hecher, and Woltran 2022), this step can still generate large propositional instances, affecting the subsequent SAT-solving phase of the ASP pipeline. In the case of ACBAR, the transformation from an ABA framework to an abstract argumentation framework can result in a formula of quadratic size in the presence of cycles, again leading to large and potentially inefficient SAT encodings.

In this paper, we propose a third approach that directly encodes an ABA framework into a SAT formula whose size is linear in the number of atoms and rules. While this initial encoding is an under-approximation, *i.e.* its models represent a superset of the actual solutions, we address this through a

Counterexample-Guided Abstraction Refinement (CEGAR) process. This algorithm incrementally refines the formula by eliminating spurious models, continuing until a valid solution is identified. In doing so, our method avoids the potentially expensive preprocessing phases required by existing approaches, replacing them with a lightweight, on-the-fly refinement strategy that adapts to the structure of the problem instance during solving.

The paper is organized as follows. Section 2.1 provides a concise background on Assumption-Based Argumentation. In Section 2.2, we introduce the necessary formal preliminaries on SAT solving, including an overview of the CEGAR framework. Our main contribution, a CEGAR algorithm tailored to solve ABAF-related reasoning tasks, is detailed in Section 3. In Section 4, we evaluate the performance of our approach against state-of-the-art ABA solvers. Finally, Section 5 presents our conclusions and discusses potential directions for future work.

2 Preliminaries

This section introduces the necessary background for our approach. We first formalize Assumption-Based Argumentation (ABA) and its semantics (Section 2.1), then outline the fundamentals of Boolean satisfiability (SAT) and the Counterexample-Guided Abstraction Refinement (CEGAR) paradigm, which form the computational core of our method (Sections 2.2 and 2.3). Finally, we review state-of-the-art ABA reasoning techniques, highlighting their methodologies and limitations (Section 2.4). These preliminaries provide the foundation for the CEGAR-based algorithm presented in the next sections.

2.1 Assumption-based Argumentation

Formal Argumentation (Baroni et al. 2018) studies how *arguments* and their interactions (especially conflicts through *attacks*), can be formally represented and how the acceptability of arguments can be systematically evaluated. In this work, we focus on *Assumption-Based Argumentation* (ABA) (Bondarenko et al. 1997; Toni 2014), with particular attention to its logic programming fragment, which has attracted increasing interest in recent years (Cyras et al. 2017; Lehtonen, Wallner, and Järvisalo 2021a; Lehtonen, Wallner, and Järvisalo 2021b; Järvisalo, Lehtonen, and Niskanen 2023; Rapberger and Ulbricht 2023; Lehtonen et al. 2024; Rapberger and Ulbricht 2024).

Definition 1 (Assumption-Based Argumentation Framework)**.** *An* Assumption-Based Argumentation Framework *(ABAF) is a tuple $\mathcal{D} = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, {}^{-} \rangle$ where:*

- *$\mathcal{L}$ is a set of* literals *(also referred to as* atoms*),*
- *$\mathcal{R}$ is a set of* inference rules*,*
- *$\mathcal{A} \subseteq \mathcal{L}$ is a (non-empty) set of* assumptions*,*
- *$^{-} : \mathcal{A} \to \mathcal{L}$ is a function that maps to each assumption $a \in \mathcal{A}$ its* contrary *$\overline{a}$.*

Each rule $r \in \mathcal{R}$ is of the form $r = h \leftarrow b_1, \ldots, b_n$, where $h, b_1, \ldots, b_n \in \mathcal{L}$ and $n \geq 0$. The element h is referred to as the head *of the rule, and the sequence $b_1, \ldots, b_n$ constitutes its* body*. We denote the head and body of a rule r by $\mathbf{h}(r)$ and $\mathbf{b}(r)$, respectively.*

In this paper, we restrict our attention to *flat* ABA frameworks (Cyras, Heinrich, and Toni 2021), that is, ABAFs in which the head of every rule is not an assumption.[1]

Example 1. *We adapt an example from (Toni 2014). Let $\mathcal{D} = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, {}^{-} \rangle$ be a (flat) ABA framework defined by:*

- *$\mathcal{L} = \{a, b, c, c_a, c_b, c_c, p\}$,*
- *$\mathcal{R} = \{c_a \leftarrow b, (c_b \leftarrow a, p), c_c \leftarrow b, c_c \leftarrow c, p \leftarrow\}$,*
- *$\mathcal{A} = \{a, b, c\}$,*
- *the contrariness function is defined as: $\overline{a} = c_a$, $\overline{b} = c_b$, and $\overline{c} = c_c$.*

The semantics of ABAFs are grounded in a notion of *derivation*. A literal $a \in \mathcal{L}$ is said to be *derivable* from a set of assumptions $X \subseteq \mathcal{A}$ using the rules $\mathcal{R}$, denoted $X \vdash_{\mathcal{R}} a$, if either:

- $a \in X$, or
- there exists a sequence of rules $r_1, \ldots, r_n \in \mathcal{R}$ such that $\mathbf{h}(r_n) = a$, and for every r_i in the sequence, each $x \in \mathbf{b}(r_i)$ is either in X or is the head of some earlier rule r_j with $j < i$.

Given $X \subseteq \mathcal{A}$, the set of literals derivable from X is called the *closure* of X.

Example 2 (Example 1 cont'd)**.** *Continuing Example 1, the literal p is derivable from $X_1 = \emptyset$ via the rule $(p \leftarrow)$. Similarly, c_b is derivable from $X_2 = \{a\}$ using the sequence of rules $(p \leftarrow)$ and $(c_b \leftarrow a, p)$.*

Extension-based semantics for ABAFs define sets of assumptions that satisfy specific properties, grounded in the notions of *attack* and *defense*. Given $A_1, A_2 \subseteq \mathcal{A}$:

- *A_1 attacks A_2* if there exists $a \in A_2$ such that $A_1 \vdash_{\mathcal{R}} \overline{a}$;
- *A_1 defends A_2* if, for every set $A_3 \subseteq \mathcal{A}$ that attacks A_2, it holds that A_1 attacks A_3.

With a slight abuse of terminology, we say that a set $A \subseteq \mathcal{A}$ attacks an assumption $a \in \mathcal{A}$ if it attacks the singleton $\{a\}$.

Definition 2 (Extension-based Semantics)**.** *Given $\mathcal{D} = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, {}^{-} \rangle$ an ABAF and $X \subseteq \mathcal{A}$ a set of assumptions,*

- *X is* conflict-free *($X \in \mathbf{cf}(\mathcal{D})$) if it does not attack any $a \in X$;*
- *X is* admissible *($X \in \mathbf{ad}(\mathcal{D})$) if $X \in \mathbf{cf}(\mathcal{D})$ and defends all its elements;*
- *X is* complete *($X \in \mathbf{co}(\mathcal{D})$) if $X \in \mathbf{ad}(\mathcal{D})$ and contains all the assumptions it defends;*
- *X is* preferred *($X \in \mathbf{pr}(\mathcal{D})$) if it is $\subseteq$-maximal in $\mathbf{co}(\mathcal{D})$;*
- *X is* stable *($X \in \mathbf{stb}(\mathcal{D})$) if $X \in \mathbf{cf}(\mathcal{D})$ and attacks every $a \in \mathcal{A} \setminus X$.*

Example 3 (Example 1 cont'd)**.** *Continuing the running example, the extensions of $\mathcal{D}$ under the considered semantics are as follows: $\mathbf{cf}(\mathcal{D}) = \mathbf{ad}(\mathcal{D}) = \mathbf{co}(\mathcal{D}) = \{\emptyset, \{a\}, \{b\}\}$; $\mathbf{pr}(\mathcal{D}) = \{\{a\}, \{b\}\}$; and $\mathbf{stb}(\mathcal{D}) = \{\{b\}\}$.*

[1] This restriction aligns with the focus of the current and upcoming editions of the ICCMA competition.

Classical reasoning tasks are *credulous acceptance* (verifying if a literal belongs to one σ extension, denoted by DC-σ), *skeptical acceptance* (verifying if a literal belongs to all σ extensions, DS-σ), or computing *some extension* (SE-σ).

As in abstract argumentation (Caminada 2006a), extension-based semantics for ABA frameworks can alternatively be characterized using *three-valued labellings*, where each assumption is labelled as accepted (IN), rejected (OUT), or undecided (UNDEC) (Schulz and Toni 2017).

Definition 3 (Labelling-based Semantics)**.** *Let us consider an ABAF $\mathcal{D} = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, {}^- \rangle$. An* assumption labelling *is a function $\mathcal{L}ab : \mathcal{A} \to \{\texttt{IN}, \texttt{OUT}, \texttt{UNDEC}\}$. For $X \in \{\texttt{IN}, \texttt{OUT}, \texttt{UNDEC}\}$, we denote by $X(\mathcal{L}ab) = \{a \in \mathcal{A} \mid \mathcal{L}ab(a) = X\}$ the set of assumptions labelled X.*

A labelling $\mathcal{L}ab$ is:

- admissible *if for every $a \in \mathcal{A}$:*
  - *if $\mathcal{L}ab(a) = \texttt{IN}$, then every set $A \subseteq \mathcal{A}$ that attacks a contains at least one b with $\mathcal{L}ab(b) = \texttt{OUT}$;*
  - *if $\mathcal{L}ab(a) = \texttt{OUT}$, then there exists a set $A \subseteq \mathcal{A}$ that attacks a and such that all $b \in A$ have $\mathcal{L}ab(b) = \texttt{IN}$;*
  - *if $\mathcal{L}ab(a) = \texttt{UNDEC}$, then for every set $A \subseteq \mathcal{A}$ that attacks a, there exists some $b \in A$ with $\mathcal{L}ab(b) \neq \texttt{IN}$.*
- complete *if it is admissible and for every $a \in \mathcal{A}$:*
  - *if $\mathcal{L}ab(a) = \texttt{UNDEC}$, then there exists a set $A \subseteq \mathcal{A}$ that attacks a and such that all $b \in A$ have $\mathcal{L}ab(b) \neq \texttt{OUT}$.*

  A complete labelling $\mathcal{L}ab$ is:
  - preferred *if $\texttt{IN}(\mathcal{L}ab)$ is $\subseteq$-maximal among all complete labellings;*
  - stable *if $\texttt{UNDEC}(\mathcal{L}ab) = \emptyset$.*

Example 4 (Example 1 cont'd)**.** *Continuing the running example, the complete labellings of $\mathcal{D}$ are:*

- $\mathcal{L}ab_1 = \{(a, \texttt{UNDEC}), (b, \texttt{UNDEC}), (c, \texttt{UNDEC})\}$,
- $\mathcal{L}ab_2 = \{(a, \texttt{IN}), (b, \texttt{OUT}), (c, \texttt{UNDEC})\}$,
- $\mathcal{L}ab_3 = \{(a, \texttt{OUT}), (b, \texttt{IN}), (c, \texttt{UNDEC})\}$.

Among these, $\mathcal{L}ab_2$ and $\mathcal{L}ab_3$ are preferred, and $\mathcal{L}ab_3$ is the only stable labelling.

Each σ-labelling (for $\sigma \in \{\mathbf{ad}, \mathbf{co}, \mathbf{pr}, \mathbf{stb}\}$) corresponds to a σ-extension via $\texttt{IN}(\mathcal{L}ab)$, and vice versa.

2.2 Preliminaries on Propositional Logic and SAT

Let us recall the basics of propositional logic. Let $\mathcal{V}$ be a finite set of Boolean variables, referred to as *atoms*. A *propositional formula* is defined recursively: any atom $x \in \mathcal{V}$ is a formula, and if ϕ and ψ are formulas, then so are the following: $\neg\phi$ (negation), $\phi \wedge \psi$ (conjunction), $\phi \vee \psi$ (disjunction), $\phi \Rightarrow \psi$ (implication), and $\phi \Leftrightarrow \psi$ (equivalence). Formulas are interpreted in the classical way: an *interpretation* ω assigns a truth value $\omega(x) \in \{0, 1\}$ to each variable $x \in \mathcal{V}$, where 0 denotes *false* and 1 denotes *true*. An interpretation ω is called a *model* of a formula ϕ if $\omega(\phi) = 1$. We use $\models$ to denote logical entailment and $\equiv$ for logical equivalence.

SAT solvers, tools that determine whether a propositional formula admits at least one model, typically require input in *Conjunctive Normal Form (CNF)*. A CNF formula is a conjunction of *clauses*, each of which is a disjunction of *literals*; a literal is either an atom or its negation. Any propositional formula can be converted into a query-equivalent CNF (i.e., one with the same models), using transformations that are linear in time and space with respect to the size of the original formula. A formula α is *query-equivalent* to a formula β if $\mathrm{Var}(\beta) \subseteq \mathrm{Var}(\alpha)$ and for every formula γ such that $\mathrm{Var}(\gamma) \subseteq \mathrm{Var}(\beta)$, we have $\alpha \models \gamma$ if and only if $\beta \models \gamma$. For further details on Boolean satisfiability and CNF transformations, we refer the reader to (Biere et al. 2021).

Example 5. *Let us consider the propositional formula Φ such that:*

$$\Phi = (x_1 \Leftrightarrow (x_2 \vee \neg x_3)) \wedge ((\neg x_1 \wedge x_4) \vee (x_2 \wedge \neg x_4))$$

This formula can be transformed into the query-equivalent *CNF formula Σ such that:*

$$\Sigma = \begin{pmatrix} \neg x_1 \vee x_2 \vee \neg x_3 & \neg x_2 \vee x_1 & x_3 \vee x_1 \\ \neg y_1 \vee \neg x_1 & \neg y_1 \vee x_4 & y_1 \vee x_1 \vee \neg x_4 \\ \neg y_2 \vee x_2 & \neg y_2 \vee \neg x_4 & y_2 \vee \neg x_2 \vee x_4 \\ y_1 \vee y_2 & & \end{pmatrix}$$

As we can see, every model ω of Σ also satisfies Φ, implying that Σ is query-equivalent to Φ with respect to the variables $\{x_1, x_2, x_3, x_4\}$. For instance, the model $\omega = \{\neg y_1, y_2, x_1, x_2, x_3, \neg x_4\}$ from Σ corresponds to the model $\{x_1, x_2, x_3, \neg x_4\}$ in Φ when restricted to these variables.

2.3 Counterexample-Guided Abstraction Refinement

Counterexample-Guided Abstraction Refinement (CEGAR) is an incremental technique for deciding the satisfiability of formulas in classical propositional logic. Originally introduced for model checking (Clarke, Gupta, and Strichman 2004), CEGAR addresses questions of the form "Does $S \models P$ hold?" or equivalently, "Is $S \wedge \neg P$ unsatisfiable?", where S describes a system and P a property to be verified. In such highly structured settings, it is often unnecessary to reason over the entire formula to reach a decision.

The core idea of CEGAR is to replace the original formula $\phi = S \wedge \neg P$ with a simplified abstraction ϕ' that is easier to solve in practice. Two types of abstractions are typically considered:

- An *over-approximation* $\hat{\phi}$ of ϕ, such that $\hat{\phi} \models \phi$, i.e., all models of $\hat{\phi}$ are also models of ϕ, hence $\hat{\phi}$ has *at most* as many models as ϕ;
- An *under-approximation* $\check{\phi}$ of ϕ, such that $\phi \models \check{\phi}$, i.e., all models of ϕ are models of $\check{\phi}$, so $\check{\phi}$ has *at least* as many models as ϕ.

In practice, the formula ϕ is typically expressed in CNF. In this paper, we adopt a CEGAR approach based on under-approximation, an abstraction of the original problem whose solution space is a superset of that of the target formula. Candidate solutions derived from the abstraction are then verified against the original formula: if a candidate satisfies the original problem, it is accepted; otherwise, it constitutes a counterexample that guides refinement, eliminating it from future consideration. SAT solvers are typically used

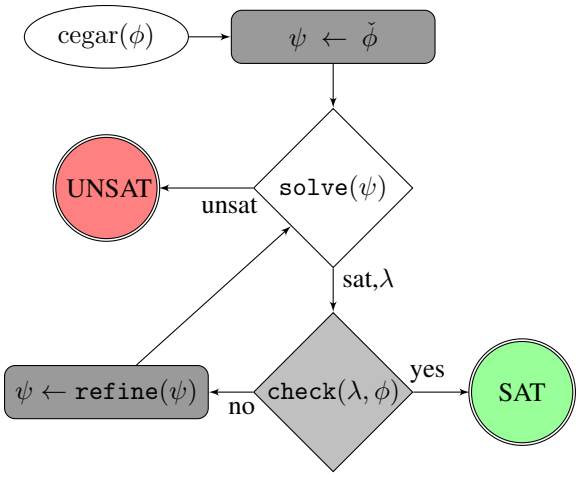

Figure 1: The CEGAR framework with under-abstraction

to search for candidate solutions and to validate them. An illustration of this framework is provided in Figure 1.

To ensure *soundness*, *completeness*, and *termination*, the CEGAR procedure must satisfy the following conditions:

1. The subroutine `solve` is sound, complete, and terminates;
2. If the current under-approximation $\check{\phi}$ is unsatisfiable, then the original formula ϕ is also unsatisfiable;
3. The routine `check`(λ, ϕ) returns `true` if and only if the candidate assignment λ is a model of ϕ;
4. There exists $n \in \mathbb{N}$ such that after n refinements, `refine`$^n(\check{\phi})$ is equisatisfiable with ϕ.

In the SAT community, CEGAR is a well-established approach for decision problems. In the CP/OR community, a closely related concept is Logic-based Benders Decomposition (LBBD) (Hooker 1994), often regarded as the optimization analogue of CEGAR. LBBD has been successfully applied in numerous domains, frequently outperforming state-of-the-art MIP solvers by orders of magnitude (Chu and Xia 2005; Hooker 2005; Tran and Beck 2012), just as CEGAR methods often surpass direct encodings. CEGAR can also be viewed as a variant of Lazy-SMT (de Moura, Rueß, and Sorea 2002; Ji and Ma 2012; Sebastiani 2007; Brummayer and Biere 2009), where problem-specific knowledge extracted from the abstraction guides the refinement, rather than relying on a theory solver.

CEGAR-style techniques have also proven effective for a wide range of computationally hard problems, including reasoning with Quantified Boolean Formulas (QBFs) (Janota et al. 2016), constraint programming (Stuckey 2010), explainable AI (XAI) (Audemard et al. 2024), biology (Riva et al. 2023), team formation (Schwind et al. 2023), computing preferred extensions in abstract argumentation (and other semantics requiring reasoning at the second level of the polynomial hierarchy or higher) (Dvorák et al. 2014), and handling argumentation dynamics (Wallner, Niskanen, and Järvisalo 2017).

2.4 State-of-the-art ABA Solvers

A natural approach to solving decision problems in ABA is to translate the given ABA framework into an Abstract Argumentation Framework (AF) (Dung 1995), and then apply one of the existing state-of-the-art abstract argumentation solvers, such as those based on SAT solving (Lagniez, Lonca, and Mailly 2015; Niskanen and Järvisalo 2020). However, as highlighted by Strass et al. (Strass, Wyner, and Diller 2019), this translation may yield an exponentially large number of abstract arguments with respect to the number of atoms in the underlying logical language of the ABAF. This exponential blow-up renders such a direct or "naive" transformation impractical in most cases.

To mitigate this issue, ACBAR (Lehtonen et al. 2023a) introduces an alternative translation strategy. It first converts the input ABAF into a so-called *atomic* ABAF, a restricted class of frameworks admitting a polynomial-size translation into an abstract AF. A critical step in this process involves eliminating cyclic dependencies, i.e., derivations in which the same atom occurs more than once. This is achieved through a rule and atom *cloning* procedure that ensures acyclicity while preserving semantic equivalence with the original framework. Once acyclicity is enforced, the resulting ABAF is translated into an atomic ABAF, which is then mapped to an abstract AF, and reasoning can be achieved using the solver μ-toksia (Niskanen and Järvisalo 2020).

An alternative line of work is represented by ASPFORABA (Lehtonen, Wallner, and Järvisalo 2021a; Lehtonen, Wallner, and Järvisalo 2021b), which employs Answer Set Programming (ASP) for reasoning within ABA. In this approach, key components of ABAFs (such as rules, assumptions, contraries, derivations, and attacks) are encoded as ASP predicates. Reasoning tasks are then performed using the ASP solver CLINGO (Gebser et al. 2011; Gebser et al. 2016). For computational problems located at the second level of the polynomial hierarchy, ASPFORABA leverages iterative ASP solving techniques that effectively implement a CEGAR-style algorithm.

3 SAT-based Algorithms for ABA

In this section, we present our SAT-based approach for reasoning within ABA. Our method builds upon the computational foundations discussed earlier, particularly SAT and CEGAR, and leverages them to efficiently handle reasoning tasks under various ABA semantics. We consider two scenarios: the *acyclic* case, where the underlying ABA framework contains no cyclic derivations, and the *general* case, where cycles may occur. The acyclic setting allows for more direct and lightweight encodings, often enabling efficient reasoning using standard SAT techniques. In contrast, the general case presents additional challenges due to potential recursion in derivations, requiring more sophisticated handling through iterative refinement and abstraction. We detail our algorithms for each case in the subsections that follow.

3.1 The Acyclic Case

We begin by considering the case where the ABAF $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ is *acyclic*. Additionally, we assume without

loss of generality that every rule in $\mathcal{R}$ is *derivable*, i.e., there exists a derivation for each rule under the given framework (non-derivable rules can be identified and removed in polynomial time). Consequently, every atom that appears as the head of a rule is also derivable. In this setting, we develop CNF encodings to capture three key ABA semantics: *conflict-free*, *complete*, and *stable*. Our encodings are based on a generalization of the labelling approach from (Schulz and Toni 2017), extending the original labelling scheme beyond assumptions to also encompass regular atoms and inference rules. For conflict-free and complete semantics, our encodings only track the elements (assumptions, atoms, and rules) that are labelled as `IN`.

Definition 4. *An **assumption** is labelled `IN` if and only if it is assigned the value `true`. A **regular atom** is labelled `IN` if and only if there exists a rule whose head is that atom and which is itself labelled `IN`. A **rule** is labelled `IN` if and only if all atoms in its body are labelled `IN`.*

We introduce a set of Boolean variables $V = V_{\mathcal{L}} \cup V_{\mathcal{R}} \cup V_{\mathcal{A}}$ such that there is a bijective correspondence between the elements of $\mathcal{L}$ (resp. $\mathcal{R}$, $\mathcal{A}$) and the variables in $V_{\mathcal{L}}$ (resp. $V_{\mathcal{R}}$, $V_{\mathcal{A}}$). Variables in $V_{\mathcal{A}}$ are named directly after the corresponding assumptions. Such a variable is set to `true` if and only if its associated assumption is selected (i.e., `IN`). Variables in $V_{\mathcal{L}}$ (denoted in_atom_x for $x \in \mathcal{L}$) and $V_{\mathcal{R}}$ (denoted in_rule_r for $r \in \mathcal{R}$) indicate whether the corresponding atom or rule is labelled `IN`. According to the ABA derivation semantics:

- in_atom_x is set to `true` for $x \in \mathcal{A}$ precisely when the assumption x is selected;
- for non-assumption atoms x, in_atom_x is set to `true` if at least one `IN` rule derives x;
- in_rule_r is set to `true` if and only if all atoms in the body of rule r are `IN`.

These conditions are captured by the following CNF encoding formulas:

$$\Phi_{\mathcal{L}} = \bigwedge_{x \in \mathcal{A}} in\_atom_x \Leftrightarrow x \quad \wedge \bigwedge_{x \in \mathcal{L} \setminus \mathcal{A}} in\_atom_x \Leftrightarrow \bigvee_{\substack{r \in \mathcal{R} \\ h(r)=x}} in\_rule_r$$

$$\Phi_{\mathcal{R}} = \bigwedge_{r \in \mathcal{R}} in\_rule_r \Leftrightarrow \bigwedge_{x \in b(r)} in\_atom_x$$

We note that the explicit definition of in_atom_x variables for assumptions is redundant, since they are equivalent to the assumption variables themselves, but we retain them to simplify the overall encoding structure and ensure uniform treatment of literals.

Given an ABAF, the derivability of rules and regular atoms is determined by the set of selected assumptions. In our encoding, this dependency is made explicit: the values of the variables corresponding to regular atoms (in_atom) and rules (in_rule) are fully determined once the truth values of the assumption variables are fixed. This property can be interpreted through the lens of logical definability: specifically, (explicit or implicit) definability in the sense of Beth's theorem (Beth 1953). When the values of assumption-related variables are set in the formula $\Phi = \Phi_{\mathcal{L}} \wedge \Phi_{\mathcal{R}}$, the values of the remaining variables are propagated deterministically by the constraints.

Lemma 1. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ be an acyclic ABAF, and let $\Phi = \Phi_{\mathcal{L}} \wedge \Phi_{\mathcal{R}}$ be the corresponding encoding over the variable set $V = V_{\mathcal{L}} \cup V_{\mathcal{R}} \cup V_{\mathcal{A}}$. Then all variables in $V_{\mathcal{L}} \cup V_{\mathcal{R}}$ are definable in terms of the variables in $V_{\mathcal{A}}$.*

Proof. Sketch of proof. Thanks to acyclicity, we can sort the atoms in such a way heads of rules are ranked higher than the atoms of the body. We construct Φ incrementally according to this order. (Base case): we consider no rules, thus no in_rule variables. in_atom_x variables are set in term of a single assumption (if $x \in \mathcal{A}$) or constant ($x \in \mathcal{L} \setminus \mathcal{A}$). (Inductive step): we introduce rules $r_i = h \leftarrow b_{i,1}, \ldots, b_{i,n_i}$ s.t. h has not be considered yet and b variables have been proved defined in terms of assumptions in a previous step. The new in_rule variables are set as equivalent to a conjunction of b variables, thus defined in terms of assumptions. The in_atom variable corresponding to h is no longer constant, but set as a disjunction of the new in_rule variable, thus defined in terms of assumptions. □

The definability of all variables in $V_{\mathcal{L}}$ and $V_{\mathcal{R}}$ in terms of $V_{\mathcal{A}}$ is central to establishing the soundness and completeness of our encoding. We now show a stronger result: there is a bijection between the models of Φ and the subsets of $\mathcal{A}$. Specifically, Φ has as many models as there are Boolean assignments over $V_{\mathcal{A}}$, and by Lemma 1, each assignment uniquely determines a full model.

Lemma 2. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ be an acyclic ABAF, and let $\Phi = \Phi_{\mathcal{L}} \wedge \Phi_{\mathcal{R}}$ be the corresponding encoding over $V = V_{\mathcal{L}} \cup V_{\mathcal{R}} \cup V_{\mathcal{A}}$. Then Φ has exactly $2^{|V_{\mathcal{A}}|}$ models, each corresponding uniquely to a subset of $\mathcal{A}$.*

Proof. Sketch of proof. By Lemma 1, all variables in $V_{\mathcal{L}} \cup V_{\mathcal{R}}$ are functionally determined by the variables in $V_{\mathcal{A}}$. *Forgetting* (i.e. existentially quantified out) these variables leads to tautology where the set of variables is $V_{\mathcal{A}}$. As established in (Lagniez, Lonca, and Marquis 2016; Lagniez, Lonca, and Marquis 2020) we can conclude that there are $2^{|V_{\mathcal{A}}|}$ models, one for each truth assignment to the assumption variables. □

The next step is to establish that each model of Φ corresponds to exactly one deductive closure of $\mathcal{D}$ under the given set of assumptions. This correspondence is captured through the in_atom variables, which encode the derivable literals.

Lemma 3. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ be an acyclic ABAF, and let $\Phi = \Phi_{\mathcal{L}} \wedge \Phi_{\mathcal{R}}$ be defined over the variable set $V = V_{\mathcal{L}} \cup V_{\mathcal{R}} \cup V_{\mathcal{A}}$. Then the models of Φ are in one-to-one correspondence with the deductive closures of $\mathcal{D}$ obtained from subsets of $\mathcal{A}$.*

Proof. Sketch of proof. By Lemma 2, it remains to show that the in_atom variables in this model represent exactly the deductive closure. Assume, for contradiction, that a model of Φ does not correspond to the correct closure. Then there is $x \in \mathcal{L}$ s.t.: (i) in_atom_x is set to `true`, but x is

not derivable from the $\mathcal{A}$ and $\mathcal{R}$; or (ii) in_atom_x is set to `false`, but x is derivable. However, by construction, each in_rule_r variable is `true` iff all in_atom variables corresponding to the atoms in the body of r are `true`, and each in_atom_x variable for $x \notin \mathcal{A}$ is `true` iff at least one in_rule_r variable for a rule with head x is `true`. □

Building on the previous lemmas, we have established that $\Phi_{\mathcal{L}} \wedge \Phi_{\mathcal{R}}$ encodes all possible subsets of assumptions and computes their corresponding deductive closures via the variables in $V_{\mathcal{L}}$. To capture conflict-freeness (the weakest among the semantics considered) we must further restrict the models of Φ to those whose closures do not include both an assumption and its contrary. This constraint can be efficiently expressed using the assumption variables in $V_{\mathcal{A}}$ and the derived atom variables in $V_{\mathcal{L}}$:

$$\Phi_{\overline{\mathcal{A}}} = \bigwedge_{x \in V_{\mathcal{A}}} \neg x \vee \neg in\_atom_{\overline{x}}$$

This formula ensures that for any assumption x selected (`true` in $V_{\mathcal{A}}$), its contrary $\overline{x}$ is not derivable (i.e., $in\_atom_{\overline{x}}$ is `false`). We now show that the resulting formula encodes conflict-freeness soundly and completely.

Proposition 1. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, {}^-)$ be an acyclic ABAF. The models of $\Phi_{CF} = \Phi_{\mathcal{L}} \wedge \Phi_{\mathcal{R}} \wedge \Phi_{\overline{\mathcal{A}}}$ are in one-to-one correspondence with the conflict-free extensions of $\mathcal{D}$.*

Proof. Sketch of proof. By Lemma 3, the formula $\Phi_{\mathcal{L}} \wedge \Phi_{\mathcal{R}}$ captures all possible deductive closures. The additional constraint $\Phi_{\overline{\mathcal{A}}}$ removes precisely those models in which a selected assumption x coexists with its contrary. This aligns with the conflict-freeness, preserving the bijection. □

Example 6 (Example 1 cont'd)**.** *The SAT encoding of the ABA Framework of Example 1 for the* **cf** *semantics is given at Table 1. The three models of the formula correspond to the* **cf** *extensions of the framework (in_a_x stands for in_atom_x and in_r_x stands for in_rule_x):*

$$\begin{aligned}
\omega_{\mathbf{cf},\emptyset} = \;& \neg a \wedge \neg b \wedge \neg c \wedge \neg in_a_a \wedge \neg in_a_b \wedge \neg in_a_c \\
\wedge \;& \neg in_a_{c_a} \wedge \neg in_a_{c_b} \wedge \neg in_a_{c_c} \wedge in_a_p \\
\wedge \;& \neg in_r_1 \wedge \neg in_r_2 \wedge \neg in_r_3 \wedge \neg in_r_4 \wedge in_r_5 \\
\omega_{\mathbf{cf},\{a\}} = \;& a \wedge \neg b \wedge \neg c \wedge in_a_a \wedge \neg in_a_b \wedge \neg in_a_c \\
\wedge \;& \neg in_a_{c_a} \wedge in_a_{c_b} \wedge \neg in_a_{c_c} \wedge in_a_p \\
\wedge \;& \neg in_r_1 \wedge in_r_2 \wedge \neg in_r_3 \wedge \neg in_r_4 \wedge in_r_5 \\
\omega_{\mathbf{cf},\{b\}} = \;& \neg a \wedge b \wedge \neg c \wedge \neg in_a_a \wedge in_a_b \wedge \neg in_a_c \\
\wedge \;& in_a_{c_a} \wedge \neg in_a_{c_b} \wedge in_a_{c_c} \wedge in_a_p \\
\wedge \;& in_r_1 \wedge \neg in_r_2 \wedge in_r_3 \wedge \neg in_r_4 \wedge in_r_5
\end{aligned}$$

Extending the encoding from conflict-free to stable semantics requires strengthening the condition on assumptions. Specifically, for each assumption a, either a or its contrary $\overline{a}$ must appear in the closure. In terms of the in_atom variables (i.e., the in_atom variables representing derived atoms), this condition can be directly encoded as:

$$\Phi_{in\_ST} = \bigwedge_{a \in \mathcal{A}} in\_atom_a \vee in\_atom_{\overline{a}}$$

This formula ensures that every assumption is either included in the extension or attacked by it.

Proposition 2. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, {}^-)$ be an acyclic ABAF. The models of $\Phi_{ST} = \Phi_{CF} \wedge \Phi_{in\_ST}$ are in one-to-one correspondence with the stable extensions of $\mathcal{D}$.*

Proof. Sketch of proof. From Proposition 1, we know that the models of Φ_{CF} correspond exactly to the conflict-free extensions of $\mathcal{D}$. The additional constraint Φ_{in_ST} removes precisely those models in which there exists an assumption a for which in_atom_a and $in\_atom_{\overline{a}}$ are both absent. This aligns with the definition of the stable semantics, preserving the bijection. □

Example 7 (Example 1 cont'd)**.** *The SAT encoding of the ABA Framework of Example 1 for the* **stb** *semantics is given at Table 1. The only model of this formula is $\omega_{\mathbf{cf},\{b\}}$ from Example 6.*

To capture the *complete semantics*, we must formalize the notion of *defense*, which requires reasoning not only about what is derived, but also about what can *not* be derived. This motivates introducing negative counterparts to the in_atom and in_rule variables used so far.

Definition 5. *An element of the framework is considered* `OUT` *under the following conditions:*

- *An* assumption $x \in \mathcal{A}$ *is* `OUT` *if and only if its negation is* `IN`*.*
- *A* regular atom $x \in \mathcal{L} \setminus \mathcal{A}$ *is* `OUT` *if and only if all rules with head x are* `OUT`*.*
- *A* rule $r \in \mathcal{R}$ *is* `OUT` *if and only if at least one atom in its body is* `OUT`*.*

We associate with each regular atom x a variable out_atom_x and with each rule r a variable out_rule_r, where these are true precisely when x or r is `OUT`, respectively. The corresponding constraints are defined as follows:

$$\begin{aligned}
\Phi_{\overline{L}} \;=\; & \bigwedge_{x \in \mathcal{A}} out_atom_x \Leftrightarrow in_atom_{\overline{x}} \\
\wedge \; & \bigwedge_{x \in \mathcal{L} \setminus \mathcal{A}} out_atom_x \Leftrightarrow \bigwedge_{\substack{r \in \mathcal{R} \\ h(r) = x}} out_rule_r
\end{aligned}$$

$$\Phi_{\overline{R}} \;=\; \bigwedge_{r \in \mathcal{R}} out\_rule_r \;\Leftrightarrow \bigvee_{x \in b(r)} out\_atom_x$$

These formulas encode the bottom-up propagation of `OUT` status based on the structure of the rules and the truth values of contraries. Notably, $\Phi_{\overline{L}}$ and $\Phi_{\overline{R}}$ are structurally dual to $\Phi_{\mathcal{L}}$ and $\Phi_{\mathcal{R}}$, but they propagate falsity (exclusion) rather than truth (inclusion). With these definitions in place, we are now prepared to formally encode the *defense condition* required for the complete semantics: an assumption x is defended by a set of assumptions Y if every set of assumptions that attacks x is itself attacked by Y. In terms of the encoding, this requires ensuring that any derivation of $\overline{x}$ is blocked, that is $in\_atom_{\overline{x}} = \texttt{false}$, whenever x is included in the extension.

If the contrary of an assumption x is a regular atom then there exists a set of assumptions that can derive $\overline{x}$ (as hypothesized at the beginning of the section). Therefore, in order to defend x, an extension must derive the contrary of $\overline{x}$. This means that if x is present in an extension, then $out\_atom_{\overline{x}}$

Encoding	Constraints
$\mathbf{cf}, \mathbf{stb}, \mathbf{co}$	$\Phi_{\mathcal{L}} = (in\_atom_a \Leftrightarrow a) \wedge (in\_atom_b \Leftrightarrow b) \wedge (in\_atom_c \Leftrightarrow c) \wedge (in\_atom_{c_a} \Leftrightarrow in\_rule_1)$ $\wedge \; (in\_atom_{c_b} \Leftrightarrow in\_rule_2) \wedge (in\_atom_{c_c} \Leftrightarrow in\_rule_3 \vee in\_rule_4) \wedge (in\_atom_p \Leftrightarrow in\_rule_5)$ $\Phi_{\mathcal{R}} = (in\_rule_1 \Leftrightarrow in\_atom_b) \wedge (in\_rule_2 \Leftrightarrow in\_atom_b \wedge in\_atom_p) \wedge (in\_rule_3 \Leftrightarrow in\_atom_b)$ $\wedge \; (in\_rule_4 \Leftrightarrow in\_atom_c) \wedge (in\_rule_5 \Leftrightarrow \top)$ $\Phi_{\overline{\mathcal{A}}} = (\neg a \vee \neg in\_atom_{c_a}) \wedge (\neg b \vee \neg in\_atom_{c_b}) \wedge (\neg c \vee \neg in\_atom_{c_c})$
$\mathbf{stb}$	$\Phi_{in\_ST} = (in\_atom_a \vee in\_atom_{c_a}) \wedge (in\_atom_b \vee in\_atom_{c_b}) \wedge (in\_atom_c \vee in\_atom_{c_c})$
$\mathbf{co}$	$\Phi_{\overline{\mathcal{L}}} = (out\_atom_a \Leftrightarrow in\_atom_{c_a}) \wedge (out\_atom_b \Leftrightarrow in\_atom_{c_b}) \wedge (out\_atom_c \Leftrightarrow in\_atom_{c_c})$ $\wedge \; (out\_atom_{c_a} \Leftrightarrow out\_rule_1) \wedge (out\_atom_{c_b} \Leftrightarrow out\_rule_2) \wedge (out\_atom_{c_c} \Leftrightarrow out\_rule_3$ $\wedge \; out\_rule_4) \wedge (out\_atom_p \Leftrightarrow \bot)$ $\Phi_{\overline{\mathcal{R}}} = (out\_rule_1 \Leftrightarrow out\_atom_b) \wedge (out\_rule_2 \Leftrightarrow out\_atom_b \vee out\_atom_p)$ $\wedge \; (out\_rule_3 \Leftrightarrow out\_atom_b) \wedge (out\_rule_4 \Leftrightarrow out\_atom_c) \wedge (out\_rule_5 \Leftrightarrow \top)$ $\Phi_{in\_CO} = (a \Leftrightarrow out\_atom_{c_a}) \wedge (b \Leftrightarrow out\_atom_{c_b}) \wedge (c \Leftrightarrow out\_atom_{c_c})$

Table 1: SAT Encodings for the ABA Framework of Example 1. Each row gives a set of constraints (second column) and the related semantics (first column). For example, the encoding for the $\mathbf{stb}$ semantics requires the constraints of the first and the second rows.

must be set to `true`. If the contrary $\overline{x}$ is an assumption, then we must consider the nature of its own contrary $\overline{\overline{x}}$. If $\overline{\overline{x}}$ is itself an assumption, this assumption is sufficient but also necessary to defend x: it must be present in every extension to which x belongs. If $\overline{\overline{x}}$ is a regular atom, it must be `IN` to defend x.

$$\begin{aligned}\Phi_{in\_CO} = & \bigwedge_{a \in \mathcal{A} \mid \overline{a} \notin \mathcal{A}} a \Leftrightarrow out\_atom_{\overline{a}} \\ \wedge & \bigwedge_{a \in \mathcal{A} \mid \overline{a} \in \mathcal{A} \wedge \overline{\overline{a}} \in \mathcal{A}} a \Leftrightarrow \overline{\overline{a}} \\ \wedge & \bigwedge_{a \in \mathcal{A} \mid \overline{a} \in \mathcal{A} \wedge \overline{\overline{a}} \notin \mathcal{A}} a \Leftrightarrow in\_atom_{\overline{\overline{a}}}\end{aligned}$$

Proposition 3. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, {}^-)$ by an acyclic ABAF. The models of $\Phi_{CO} = \Phi_{CF} \wedge \Phi_{in\_CO}$ are in one-to-one correspondence with the complete extensions of $\mathcal{D}$.*

Proof. As in the proof of Proposition 2, but using Φ_{in_CO} instead of Φ_{in_ST}. Φ_{in_CO} enforces the defense condition by requiring each atom's presence to be equivalent to it being defended. □

Example 8 (Example 1 cont'd)**.** *The SAT encoding of the ABA Framework of Example 1 for the* $\mathbf{co}$ *semantics is given at Table 1. The models of this formula are the following (out_a_x stands for out_atom_x; out_r_x stands for out_rule_x; refer to Example 6 for the $\omega_{\mathbf{cf},\mathbf{x}}$ terms):*

$$\begin{aligned}\omega_{\mathbf{co},\emptyset} = & \; \omega_{\mathbf{cf},\emptyset} \wedge \neg out\_a_a \wedge \neg out\_a_b \wedge \neg out\_a_c \\ & \wedge \; \neg out\_a_{c_a} \wedge \neg out\_a_{c_b} \wedge \neg out\_a_{c_c} \wedge \neg out\_a_p \\ & \wedge \; \neg out\_r_1 \wedge \neg out\_r_2 \wedge \neg out\_r_3 \wedge \neg out\_r_4 \\ & \wedge \; \neg out\_r_5 \\ \omega_{\mathbf{co},\{a\}} = & \; \omega_{\mathbf{cf},\{a\}} \wedge \neg out\_a_a \wedge out\_a_b \wedge \neg out\_a_c \\ & \wedge \; out\_a_{c_a} \wedge \neg out\_a_{c_b} \wedge \neg out\_a_{c_c} \wedge \neg out\_a_p \\ & \wedge \; out\_r_1 \wedge \neg out\_r_2 \wedge out\_r_3 \wedge \neg out\_r_4 \\ & \wedge \; \neg out\_r_5 \\ \omega_{\mathbf{co},\{b\}} = & \; \omega_{\mathbf{cf},\{b\}} \wedge out\_a_a \wedge \neg out\_a_b \wedge out\_a_c \\ & \wedge \; \neg out\_a_{c_a} \wedge out\_a_{c_b} \wedge \neg out\_a_{c_c} \wedge \neg out\_a_p \\ & \wedge \; \neg out\_r_1 \wedge out\_r_2 \wedge \neg out\_r_3 \wedge out\_r_4 \\ & \wedge \; \neg out\_r_5\end{aligned}$$

Preferred extensions are defined as the $\subseteq$-maximal complete extensions, as in standard abstract argumentation frameworks (Dung 1995). Thus, identifying preferred extensions reduces to computing complete extensions and selecting those that are maximal with respect to inclusion.

In this section, we have provided SAT-based encodings for the most commonly studied ABA semantics, as identified by the ICCMA competition (Järvisalo, Lehtonen, and Niskanen 2025), under the assumption that the ABA frameworks are acyclic. These encodings can be applied analogously to existing SAT encodings in abstract argumentation (Lagniez, Lonca, and Mailly 2015; Niskanen and Järvisalo 2020). To compute extensions, it suffices to enumerate models of the corresponding propositional formula. For preferred semantics, one must search for maximal models among the complete extensions. Moreover, these encodings facilitate reasoning tasks such as credulous and skeptical acceptance. Specifically, to determine whether a literal x is credulously (respectively, skeptically) accepted, we check whether the variable in_atom_a is `true` in some (respectively, each) model. This task reduces to a single SAT call in the case of stable semantics: we verify satisfiability of the formula $\Phi_{ST} \wedge in\_atom_x$ (for credulous acceptance), or $\Phi_{ST} \wedge \neg in\_atom_x$ (for skeptical acceptance). Having addressed the acyclic case, we now turn our attention to the general case, where ABA frameworks may include cycles.

3.2 The General Case

Consider the ABAF $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, {}^-)$ with $\mathcal{L} = \{p, q\}$, $\mathcal{R} = \{r_1 : q \leftarrow p,\ r_2 : p \leftarrow q\}$, $\mathcal{A} = \emptyset$, and ${}^- = \emptyset$. This framework admits a single conflict-free extension: the empty set $\emptyset$. However, the conflict-free encoding from the previous section yields two models:

$$\neg in\_atom_p \wedge \neg in\_atom_q \wedge \neg in\_rule_{r_1} \wedge \neg in\_rule_{r_2}$$

and

$$in\_atom_p \wedge in\_atom_q \wedge in\_rule_{r_1} \wedge in\_rule_{r_2}$$

The issue in cyclic frameworks is that atoms can be unjustifiably derived through circular dependencies among rules.

One way to address this is by transforming the ABAF into an acyclic, query-equivalent form using techniques like those in ACBAR (Lehtonen et al. 2023a). However, these transformations may lead to quadratic blow-up in size, adversely affecting performance. Instead, we propose an alternative approach. We observe that any spurious model of the encoding corresponds to an over-approximation: it includes atoms that are not justifiably derivable. These models improperly assign `true` to at least one in_atom variable. We illustrate this in the context of conflict-free semantics, though the same principle applies to other semantics as well.

Lemma 4. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ be any (possibly cyclic) ABAF. Then, for every conflict-free extension of $\mathcal{D}$, there exists a corresponding model of Φ_{CF}.*

Proof. Sketch of proof. Assume for contradiction that a conflict-free extension has no corresponding model in Φ_{CF}. This would imply that the interpretation violates some clause in Φ_{CF}. Such violation would mean either an atom is derived without an applicable rule, or a rule is applied while one of its premises is not derived, which is impossible. □

This lemma confirms that the models of Φ_{CF} cover all valid extensions, making SAT-based CEGAR based on under-approximation a viable strategy. To apply this approach, we begin by implementing the `check` function, which determines whether a given model corresponds to a valid extension. Fortunately, this step is straightforward: extract the assumptions set to `true` in the model, propagate them in the ABAF, and verify that the derived atoms match those in_atom variables assigned `true` in the model.

Next, we define the `refine` function, which, given an invalid model, returns clauses to exclude it from future consideration. Although negating invalid models is sufficient to prevent the SAT solver from returning them again, we take advantage of the nature of the problem to add a more restrictive set of clauses. This results in a potential decrease in the total number of SAT calls. The key insight here is that invalid models must include atoms marked as derived (in_atom variables set to `true`) that are not actually derivable from the assumptions.

Lemma 5. *Let $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ be a general ABAF. If a model of Φ_{CF} does not correspond to a conflict-free extension, then it includes at least one in_atom variable set to `true` whose atom is not derivable from the model's assumed literals.*

Proof. As shown in Lemma 3, any atom in the closure induced by the model's assumptions must be present in the model. If the model does not correspond to an extension, then at least one in_atom or in_rule variable (which implies an in_atom variable) must be incorrectly set to `true`, indicating the presence of a non-derivable atom. □

In other words, we can eliminate an invalid model by enforcing that any in_atom variable incorrectly set to `true` must instead be `false` under the current set of assumptions: that is, it must remain `false` unless one of the missing assumptions is included. This observation leads directly to the formulation of Algorithm 1.

Algorithm 1: Compute an extension

Input : An ABAF $\mathcal{D} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$, a semantics $\sigma \in \{\mathbf{cf}, \mathbf{co}, \mathbf{stb}\}$
Output: A σ-extension of $\mathcal{D}$ or $\varnothing$ if none

```
1  Φ ← encode(D, σ);
2  loop
3      ω ← computeModel(Φ);                    // solve
4      if ω = ∅ then return ∅;                 // UNSAT
       // check
5      assumptions ← modelAssumptions(ω);
6      inAbaf ← closure(D, assumptions);
7      inModel ← modelAtoms(ω);
8      if inModel \ inAbaf = ∅ then
9          return inAbaf
       // refine
10     foreach x in inModel \ inAbaf do
11         Φ ← Φ ∪ {¬x ∨ ⋁_{ℓ∈A\assumptions} ℓ};
```

Algorithm 1 starts by computing the initial formula Φ from the input ABAF and selected semantics, then enters the main CEGAR loop. At line 3, the SAT solver is invoked to determine whether Φ has a model. If no model is found, Lemma 4 guarantees that no extension exists, and the algorithm terminates accordingly (line 4). This step corresponds to the `solve` phase of the CEGAR framework in Figure 1. If a model is found, the algorithm enters the `check` phase by extracting the assumptions set to `true` (line 5). It then computes the deductive closure of this assumption set in the ABAF (line 6), and compares it with the set of atoms assigned `true` in the model (line 7). If these two sets match the closure forms a valid extension, which is returned (lines 8–9). Thus, lines 5 to 9 implement the `check` routine of the CEGAR framework.

If the model does not correspond to a valid extension, the algorithm enters the `refine` phase (lines 10–11), which adds constraints to eliminate the spurious model. The loop then restarts with the refined formula. Each refinement step eliminates at least one previously unseen incorrect model. Since the number of possible interpretations is finite, the loop must terminate after a finite number of iterations. Eventually, the formula will represent the problem precisely, with all spurious models eliminated. Since our algorithm follows the CEGAR framework and relies on a SAT solver that is sound, complete, and terminating, our approach is also guaranteed to be sound, complete, and to terminate.

4 Experimental Evaluation

We implemented our approach in Rust, releasing the solver publicly on GitHub[2]. For the CEGAR procedure, we selected the CaDiCaL SAT solver (Biere et al. 2020) as the underlying oracle. Maximal extension search for SE-**pr** and DS-**pr** problems was handled using a standard CEGAR al-

[2]https://github.com/crillab/scalop/releases/tag/kr25

gorithm (Dvorák et al. 2014). To optimize performance, we consistently used the same SAT solver instance incrementally across different problem variants. Leveraging literal assumptions allows temporary constraints to be added without discarding learned clauses, which significantly accelerates successive SAT solver calls common in CEGAR-based approaches. In accordance with the assumption stated at the start of Section 3, we preemptively identify and remove any atoms that cannot be derived from the ABA assumptions prior to computing extensions. This pruning step simplifies the problem and can improve overall solver efficiency.

We evaluated our approach against state-of-the-art solvers by replicating the experiments from the latest ICCMA competition (Järvisalo, Lehtonen, and Niskanen 2025). We selected ASPFORABA and ACBAR as benchmarks since they were the only solvers that participated in all ABA tracks of the competition. Notably, ASPFORABA and ACBAR ranked first and second across all tracks, respectively. Our evaluation covered the same six tracks (DC-**co**, DC-**stb**, DS-**pr**, DS-**stb**, SE-**pr**, and SE-**stb**), using the identical set of 400 instances, the same acceptance query arguments, and identical resource limits (1200 seconds of CPU time and 32GB of memory) as in the competition. Execution and resource enforcement were managed using the runsolver tool (Roussel 2011). Each solver ran on an Intel(R) Xeon(R) E5-2643 v4 CPU at 3.30 GHz, with Rocky Linux 9.5 (Linux kernel 5.14) as the operating system. The development environment included Rust 1.86, GCC 11.5, and Python 3.11.

For our experiments, we used the instance set from the ICCMA 2023 competition. However, as shown in Figure 2, most instances were either solved almost instantly by ASPFORABA (the competition winner) or were too challenging, resulting in a limited number of instances suitable for a meaningful comparison between ASPFORABA and our approach. These instances were generated by a script parameterized by the number of atoms, with maximum values of 2000 and 5000. Upon closer analysis, we found that instances with 2000 atoms were generally easy to solve, while ASPFORABA tended to time out on instances with 5000 atoms. To better evaluate performance on medium-difficulty problems, we generated additional instances with 3000 and 4000 atoms using the competition organizers' script, producing 160 new instances (resulting in 960 experiments across all tracks). We refer to the combined set of original and new instances as `ICCMA23+`. All three solvers were then tested on these newly generated medium-difficulty benchmarks.

The results in Table 2 report the number of instances solved by each approach. Across all benchmark sets, ASPFORABA consistently outperforms ACBAR, mirroring the competition results. On the original ICCMA23 test suite, the CEGAR approach shows comparable overall performance to ASPFORABA. For half of the tracks (DC-**co**, DC-**stb**, and SE-**stb**), ASPFORABA leads, while CEGAR performs better on the remaining tracks. Notably, for each track, the best solver is close to the Virtual Best Solver and each solver exhibits strengths on different benchmark tracks, suggesting that both methods have complementary strengths with no significant performance gaps. Looking at the ICCMA23+ results (Table 2 and Figure 2), our approach benefits from the addition of medium-difficulty instances. It surpasses ASPFORABA on DC-**co** and SE-**stb** tracks, although ASPFORABA retains a slight edge on DC-**stb**. Importantly, on tracks related to preferred semantics, our approach clearly outperforms ASPFORABA. We attribute this to the fully incremental use of the SAT solver in computing (SE-**pr**) and enumerating (DS-**pr**) preferred extensions. Both CEGAR loops share the same solver, which leverages learned clauses and literal assumptions. As a result, knowledge gained during maximal model search enhances extension computation, and vice versa, leading to improved overall efficiency.

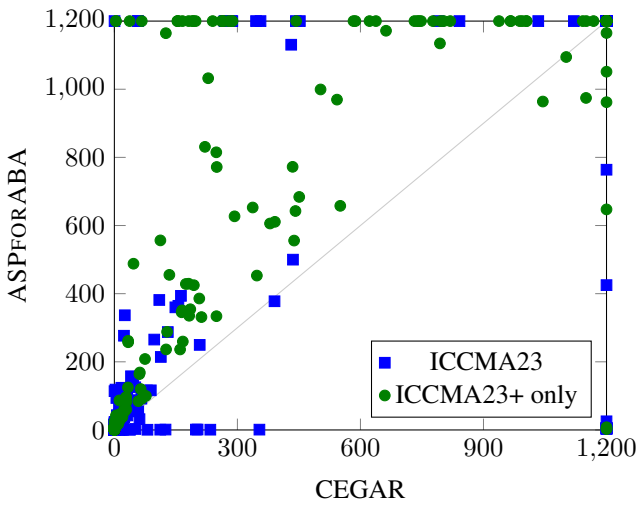

Figure 2: Scatter plot comparing the CPU times of ASPFORABA and CEGAR. For each instance, the x-coordinate (resp. y-coordinate) of the plot shows the CPU time required by CEGAR (resp. ASPFORABA). Plots on the top or right lines are instances for which at least one solver reached the timeout. "ICCMA23+ only" refers to ICCMA23+ $\setminus$ ICCMA23.

The effectiveness of our approach may depend on how many calls to the SAT oracle are required to obtain a correct extension or prove that none exist. To investigate this, we examined the number of calls necessary to obtain a correct extension (or conclude that none exist) across the 5,238 successful computations using our method. The results show that the number of SAT calls required is surprisingly low: no instance needed more than 10 calls. Specifically, the distribution is heavily skewed towards very few calls, with most instances requiring just 1 (4,828, half of them returning UNSAT) or 2 calls (335), and only a handful needing between 3 and 10 calls. This demonstrates that our approach efficiently converges in practice. The efficiency of our approach is partly due to the fact that nearly half of the successful solver calls yield UNSAT, effectively pruning the search. Theoretically, the number of calls could be as large as $|\mathcal{L}|/2$, for example in scenarios involving a single empty extension and $|\mathcal{L}|/2$ pairs of atoms that self-derive. However, the preprocessing step introduced in Section 3, which removes atoms that cannot be derived from the assumptions, often suffices

Solver	Instances	DC-**co**	DC-**stb**	DS-**pr**	DS-**stb**	SE-**pr**	SE-**stb**
ACBAR	ICCMA23	233 (1026.1)	237 (1001.5)	229 (1058.8)	238 (996.3)	230 (1047.2)	238 (997.7)
ASPFORABA	ICCMA23	**381 (119.7)**	**383 (104.4)**	375 (159.6)	381 (117.4)	377 (145.6)	**381 (118.3)**
CEGAR	ICCMA23	379 (130.3)	381 (115.0)	**380 (126.8)**	**382 (114.7)**	**381 (122.5)**	380 (123.5)
VBS	ICCMA23	381 (117.0)	383 (102.8)	381 (120.1)	383 (108.4)	383 (111,7)	382 (111,9)
ACBAR	ICCMA23+	233 (1418.6)	237 (1401.0)	229 (1442.0)	238 (1397.4)	230 (1433.7)	238 (1398.3)
ASPFORABA	ICCMA23+	514 (211.3)	**526 (162.4)**	499 (274.4)	519 (188.5)	504 (255.1)	517 (195.7)
CEGAR	ICCMA23+	**515 (204.1)**	523 (169.2)	**517 (206.0)**	**522 (173.5)**	**520 (193.6)**	**522 (175.2)**
VBS	ICCMA23+	519 (187.2)	526 (157.6)	519 (197.0)	524 (166.4)	524 (179.4)	524 (167.0)

Table 2: Number of instances (and average PAR2 scores) from the `ICCMA23` and `ICCMA23+` benchmark sets solved before the timeout. For each instance, the PAR2 score is equal to the solver's CPU time in case of success or twice the timeout otherwise.

to break such cycles. This preprocessing likely contributes significantly to the low number of SAT calls in practice.

Finally, we compared our `refine` implementation with the naïve one where the negation of the incorrect model is added to the CNF formula. From a theoretical standpoint, our counter-example construction can eliminate exponentially more spurious models than simply negating the current model. In practice, the results are somewhat more mixed: One benchmark instance was solved by our strategy but not by the naïve one. Average PAR2 scores were slightly worse (by less than one point) when using the naïve strategy, except under the preferred semantics, where the difference was more pronounced, six points for DS-**pr** and eight points for SE-**pr**. This discrepancy can be partially attributed to the fact that one fewer instance was solved. Regarding SAT oracle usage, most extension searches require very few calls: 91.85% of cases need only one call, and 99.25% require at most five. However, there are outlier cases with a higher number of calls (up to 33).

5 Conclusion and Perspectives

We have presented a novel approach for solving classical reasoning tasks in Assumption-Based Argumentation (ABA). Unlike state-of-the-art methods that translate ABA into other formalisms before applying SAT solving, our approach directly leverages SAT solvers within a CEGAR framework. This design effectively addresses challenges related to circularities in literal derivations inherent to ABA, avoiding incorrect results. Empirically, our method requires only a small number of calls to the SAT solver and demonstrates superior performance compared to other approaches on standard ABA benchmarks. Notably, it performs especially well on the more challenging preferred semantics, which are known for their computational complexity.

Looking forward, several avenues merit exploration. One promising direction is to investigate whether, in certain cases, subsequent SAT calls can be further optimized or even eliminated by simply retaining and managing the excess atoms identified during the refinement steps. Moreover, it would be valuable to adapt our approach to other difficult ABA semantics characterized by maximality notions, such as the semi-stable (Caminada 2006b), stage (Verheij 1996) or ideal (Dung, Mancarella, and Toni 2007) semantics. Additionally, understanding why some instances pose challenges for our solver while being efficiently solved by competitors like ASPFORABA could yield insights to substantially improve our solver's efficiency. Finally, extending our method beyond flat ABA frameworks to handle the more general and expressive non-flat ABA frameworks represents an exciting and important future challenge.

Acknowledgements

The third author is funded by the French National Research Agency under grants AGGREEY (ANR-22-CE23-0005) and AIDAL (ANR-22-CPJ1-0061-01).

References

Audemard, G.; Lagniez, J.; Marquis, P.; and Szczepanski, N. 2024. On the computation of example-based abductive explanations for random forests. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, 3679–3687. ijcai.org.

Baroni, P.; Gabbay, D.; Giacomin, M.; and van der Torre, L., eds. 2018. *Handbook of Formal Argumentation*, volume 1. College Publications.

Besin, V.; Hecher, M.; and Woltran, S. 2022. Body-decoupled grounding via solving: A novel approach on the ASP bottleneck. In Raedt, L. D., ed., *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, 2546–2552. ijcai.org.

Besnard, P.; García, A. J.; Hunter, A.; Modgil, S.; Prakken, H.; Simari, G. R.; and Toni, F. 2014. Introduction to structured argumentation. *Argument Comput.* 5(1):1–4.

Beth, E. 1953. On padoa's method in the theory of definition. *Indagationes Mathematicae (Proceedings)* 56:330–339.

Biere, A.; Fazekas, K.; Fleury, M.; and Heisinger, M. 2020. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Balyo, T.; Froleyks, N.; Heule, M.; Iser, M.; Järvisalo, M.; and Suda, M., eds., *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, 51–53. University of Helsinki.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2021. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Bondarenko, A.; Dung, P. M.; Kowalski, R. A.; and Toni, F. 1997. An abstract, argumentation-theoretic approach to default reasoning. *Artif. Intell.* 93:63–101.

Brummayer, R., and Biere, A. 2009. Lemmas on demand for the extensional theory of arrays. *J. Satisf. Boolean Model. Comput.* 6(1-3):165–201.

Caminada, M. 2006a. On the issue of reinstatement in argumentation. In Fisher, M.; van der Hoek, W.; Konev, B.; and Lisitsa, A., eds., *Logics in Artificial Intelligence, 10th European Conference, JELIA 2006, Liverpool, UK, September 13-15, 2006, Proceedings*, volume 4160 of *Lecture Notes in Computer Science*, 111–123. Springer.

Caminada, M. 2006b. Semi-stable semantics. In Dunne, P. E., and Bench-Capon, T. J. M., eds., *Computational Models of Argument: Proceedings of COMMA 2006, September 11-12, 2006, Liverpool, UK*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, 121–130. IOS Press.

Chu, Y., and Xia, Q. 2005. A hybrid algorithm for a class of resource constrained scheduling problems. In Barták, R., and Milano, M., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Second International Conference, CPAIOR 2005, Prague, Czech Republic, May 30 - June 1, 2005, Proceedings*, volume 3524 of *Lecture Notes in Computer Science*, 110–124. Springer.

Clarke, E. M.; Gupta, A.; and Strichman, O. 2004. Sat-based counterexample-guided abstraction refinement. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 23(7):1113–1123.

Cyras, K.; Fan, X.; Schulz, C.; and Toni, F. 2017. Assumption-based argumentation: Disputes, explanations, preferences. *FLAP* 4(8).

Cyras, K.; Oliveira, T.; Karamlou, A.; and Toni, F. 2021. Assumption-based argumentation with preferences and goals for patient-centric reasoning with interacting clinical guidelines. *Argument Comput.* 12(2):149–189.

Cyras, K.; Heinrich, Q.; and Toni, F. 2021. Computational complexity of flat and generic assumption-based argumentation, with and without probabilities. *Artif. Intell.* 293:103449.

de Moura, L. M.; Rueß, H.; and Sorea, M. 2002. Lazy theorem proving for bounded model checking over infinite domains. In Voronkov, A., ed., *Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002, Proceedings*, volume 2392 of *Lecture Notes in Computer Science*, 438–455. Springer.

Dung, P. M.; Mancarella, P.; and Toni, F. 2007. Computing ideal sceptical argumentation. *Artif. Intell.* 171(10-15):642–674.

Dung, P. M.; Thang, P. M.; and Hung, N. D. 2010. Modular argumentation for modelling legal doctrines of performance relief. *Argument Comput.* 1(1):47–69.

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

Dvorák, W., and Dunne, P. E. 2018. Computational problems in formal argumentation and their complexity. In Baroni, P.; Gabbay, D.; Giacomin, M.; and van der Torre, L., eds., *Handbook of Formal Argumentation*. College Publications. 631–688.

Dvorák, W.; Järvisalo, M.; Wallner, J. P.; and Woltran, S. 2014. Complexity-sensitive decision procedures for abstract argumentation. *Artif. Intell.* 206:53–78.

Fan, X.; Craven, R.; Singer, R.; Toni, F.; and Williams, M. 2013. Assumption-based argumentation for decision-making with preferences: A medical case study. In Leite, J.; Son, T. C.; Torroni, P.; van der Torre, L.; and Woltran, S., eds., *Computational Logic in Multi-Agent Systems - 14th International Workshop, CLIMA XIV, Corunna, Spain, September 16-18, 2013. Proceedings*, volume 8143 of *Lecture Notes in Computer Science*, 374–390. Springer.

Fan, X.; Liu, S.; Zhang, H.; Leung, C.; and Miao, C. 2016. Explained activity recognition with computational assumption-based argumentation. In Kaminka, G. A.; Fox, M.; Bouquet, P.; Hüllermeier, E.; Dignum, V.; Dignum, F.; and van Harmelen, F., eds., *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, 1590–1591. IOS Press.

Gebser, M.; Kaufmann, B.; Kaminski, R.; Ostrowski, M.; Schaub, T.; and Schneider, M. 2011. Potassco: The potsdam answer set solving collection. *AI Commun.* 24(2):107–124.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Wanko, P. 2016. Theory solving made easy with clingo 5. In Carro, M.; King, A.; Saeedloei, N.; and Vos, M. D., eds., *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA*, volume 52 of *OASIcs*, 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Hooker, J. N. 1994. Logic-based methods for optimization. In Borning, A., ed., *Principles and Practice of Constraint Programming, Second International Workshop, PPCP'94, Rosario, Orcas Island, Washington, USA, May 2-4, 1994, Proceedings*, volume 874 of *Lecture Notes in Computer Science*, 336–349. Springer.

Hooker, J. N. 2005. A hybrid method for the planning and scheduling. *Constraints An Int. J.* 10(4):385–401.

Janota, M.; Klieber, W.; Marques-Silva, J.; and Clarke, E. M. 2016. Solving QBF with counterexample guided refinement. *Artif. Intell.* 234:1–25.

Järvisalo, M.; Lehtonen, T.; and Niskanen, A. 2023. Design

of ICCMA 2023, 5th international competition on computational models of argumentation: A preliminary report (invited paper). In Cocarascu, O.; Doutre, S.; Mailly, J.; and Rago, A., eds., *Proceedings of the First International Workshop on Argumentation and Applications co-located with 20th International Conference on Principles of Knowledge Representation and Reasoning (KR 2023), Rhodes, Greece, September 2-8, 2023*, volume 3472 of *CEUR Workshop Proceedings*, 4–10. CEUR-WS.org.

Ji, X., and Ma, F. 2012. An efficient lazy SMT solver for nonlinear numerical constraints. In Reddy, S., and Drira, K., eds., *21st IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2012, Toulouse, France, June 25-27, 2012*, 324–329. IEEE Computer Society.

Järvisalo, M.; Lehtonen, T.; and Niskanen, A. 2025. Iccma 2023: 5th international competition on computational models of argumentation. *Artificial Intelligence* 342:104311.

Kaufmann, B.; Leone, N.; Perri, S.; and Schaub, T. 2016. Grounding and solving in answer set programming. *AI Mag.* 37(3):25–32.

Lagniez, J.-M.; Lonca, E.; and Mailly, J.-G. 2015. CoQuiAAS: A constraint-based quick abstract argumentation solver. In *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*, 928–935. IEEE Computer Society.

Lagniez, J.; Lonca, E.; and Marquis, P. 2016. Improving model counting by leveraging definability. In Kambhampati, S., ed., *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 751–757. IJCAI/AAAI Press.

Lagniez, J.; Lonca, E.; and Marquis, P. 2020. Definability for model counting. *Artif. Intell.* 281:103229.

Lehtonen, T.; Rapberger, A.; Ulbricht, M.; and Wallner, J. P. 2023a. Argumentation frameworks induced by assumption-based argumentation: Relating size and complexity. In *International Conference on Principles of Knowledge Representation and Reasoning*, 440–450. International Joint Conferences on Artificial Intelligence.

Lehtonen, T.; Rapberger, A.; Ulbricht, M.; and Wallner, J. P. 2023b. Argumentation frameworks induced by assumption-based argumentation: Relating size and complexity. In Marquis, P.; Son, T. C.; and Kern-Isberner, G., eds., *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023*, 440–450.

Lehtonen, T.; Rapberger, A.; Toni, F.; Ulbricht, M.; and Wallner, J. P. 2024. On computing admissibility in ABA. In Reed, C.; Thimm, M.; and Rienstra, T., eds., *Computational Models of Argument - Proceedings of COMMA 2024, Hagen, Germany, September 18-20, 2024*, volume 388 of *Frontiers in Artificial Intelligence and Applications*, 121–132. IOS Press.

Lehtonen, T.; Wallner, J. P.; and Järvisalo, M. 2021a. Declarative algorithms and complexity results for assumption-based argumentation. *J. Artif. Intell. Res.* 71:265–318.

Lehtonen, T.; Wallner, J. P.; and Järvisalo, M. 2021b. Harnessing incremental answer set solving for reasoning in assumption-based argumentation. *Theory Pract. Log. Program.* 21(6):717–734.

Niskanen, A., and Järvisalo, M. 2020. μ-toksia: An efficient abstract argumentation reasoner. In Calvanese, D.; Erdem, E.; and Thielscher, M., eds., *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, 800–804.

Rapberger, A., and Ulbricht, M. 2023. On dynamics in structured argumentation formalisms. *J. Artif. Intell. Res.* 77:563–643.

Rapberger, A., and Ulbricht, M. 2024. Repairing assumption-based argumentation frameworks. In Marquis, P.; Ortiz, M.; and Pagnucco, M., eds., *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024*.

Riva, S.; Lagniez, J.; López, G. M.; and Paulevé, L. 2023. Tackling universal properties of minimal trap spaces of boolean networks. In Pang, J., and Niehren, J., eds., *Computational Methods in Systems Biology - 21st International Conference, CMSB 2023, Luxembourg City, Luxembourg, September 13-15, 2023, Proceedings*, volume 14137 of *Lecture Notes in Computer Science*, 157–174. Springer.

Roussel, O. 2011. Controlling a solver execution with the runsolver tool. *J. Satisf. Boolean Model. Comput.* 7(4):139–144.

Schulz, C., and Toni, F. 2017. Labellings for assumption-based and abstract argumentation. *International Journal of Approximate Reasoning* 84:110–149.

Schwind, N.; Demirovic, E.; Inoue, K.; and Lagniez, J. 2023. Algorithms for partially robust team formation. *Auton. Agents Multi Agent Syst.* 37(2):22.

Sebastiani, R. 2007. Lazy satisability modulo theories. *J. Satisf. Boolean Model. Comput.* 3(3-4):141–224.

Strass, H.; Wyner, A.; and Diller, M. 2019. *EMIL*: Extracting meaning from inconsistent language: Towards argumentation using a controlled natural language interface. *Int. J. Approx. Reason.* 112:55–84.

Stuckey, P. J. 2010. Lazy clause generation: Combining the power of SAT and CP (and mip?) solving. In Lodi, A.; Milano, M.; and Toth, P., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings*, volume 6140 of *Lecture Notes in Computer Science*, 5–9. Springer.

Thimm, M. 2025. The international competition on computational models of argumentation website. https://argumentationcompetition.org [Accessed: 28/05/2025].

Toni, F. 2014. A tutorial on assumption-based argumentation. *Argument Comput.* 5(1):89–117.

Tran, T. T., and Beck, J. C. 2012. Logic-based benders decomposition for alternative resource scheduling with se-

quence dependent setups. In Raedt, L. D.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P. J. F., eds., *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 774–779. IOS Press.

Verheij, B. 1996. Two approaches to dialectical argumentation: admissible sets and argumentation stages. In *Proc. of BNAIC'96*.

Wallner, J. P.; Niskanen, A.; and Järvisalo, M. 2017. Complexity results and algorithms for extension enforcement in abstract argumentation. *J. Artif. Intell. Res.* 60:1–40.