Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
KR in the Wild

BAss: Symbolic Reasoning in Abstract Dialectical Frameworks

Samuel Pastva[1] , **Van-Giang Trinh**[2*]

[1]Faculty of Informatics, Masaryk University, Brno, Czechia
[2]Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology (HCMUT), VNU-HCM, Ho Chi Minh City, Vietnam
xpastva@fi.muni.cz, van-giang.trinh@hcmut.edu.vn

Abstract

We present `BAss` (BDD-based ADF symbolic solver), a novel analysis tool for *Abstract Dialectical Frameworks* (ADFs) based on *Binary Decision Diagrams* (BDDs). It supports the fully symbolic computation of all *admissible*, *complete*, and *preferred interpretations*, as well as *2-valued* and *stable models* of an ADF. Our approach is inspired by the recently discovered equivalence between *Boolean Networks* (BNs) and ADFs, significantly extending the capabilities of current BDD-based tools `bioLQM`, `aeon`, and `adf-bdd`. We conducted experiments on a large-scale collection of real-world models from both the BN and ADF communities. Our results show that `BAss` dramatically outperforms previous BDD-based tools and is competitive with (even significantly better in some cases) state-of-the-art SAT/ASP-based methods, particularly in scenarios involving large solution spaces. Notably, `BAss` is able to enumerate all fixed points or minimal trap spaces of certain biological networks beyond the reach of existing tools, thereby enabling new analyses and case studies in systems biology. These results highlight the practical relevance of symbolic reasoning for complex real-world applications, particularly in systems biology and formal argumentation.

1 Introduction

Argumentation has emerged as a central area in AI, offering formal models for reasoning with conflicting information and justifications (Bench-Capon and Dunne 2007; Atkinson et al. 2017). A foundational approach in this field is Dung's *Abstract Argumentation Frameworks* (AFs) (Dung 1995), which represent arguments as abstract entities and define their interactions via a binary attack relation. Despite their simplicity, AFs capture a wide range of non-monotonic reasoning patterns and have inspired extensive research on semantics, algorithms, and complexity (Charwat et al. 2015). Their computational appeal is reflected in the development of numerous solvers and the organization of the *International Competition on Computational Models of Argumentation* (ICCMA) (Thimm and Villata 2017), where SAT-based approaches have shown strong empirical performance (Niskanen and Järvisalo 2020).

While AFs have been successful, their conceptual simplicity—as directed graphs of abstract arguments and attacks—can be too restrictive for modeling complex dependencies (Strass 2013). *Abstract Dialectical Frameworks* (ADFs) address this by associating each argument with a propositional *acceptance condition* (Brewka and Woltran 2010; Brewka et al. 2013), allowing edges to represent arbitrary logical relationships rather than just attacks. This enables ADFs to subsume most classical argumentation semantics, including *complete*, *preferred*, and *stable* semantics (the latter inspired by the stable model semantics of logic programs (Gelfond and Lifschitz 1988)). Interpretations assign arguments one of three truth values (true, false, or unknown), and semantics is defined via a characteristic operator. ADFs thus form a highly expressive framework for argumentation, rule-based reasoning, and logic (Brewka et al. 2013; Brewka et al. 2017), with applications in legal reasoning (Al-Abdulkarim, Atkinson, and Bench-Capon 2016), dialog systems (Neugebauer 2017), text exploration (Cabrio and Villata 2016), and defeasible theories (Strass 2014). However, reasoning in ADFs is computationally hard: many problems are NP- or coNP-complete, and preferred or stable semantics reaches Σ_2^{P}-completeness and beyond (Strass and Wallner 2015; Linsbichler et al. 2022). This motivates ongoing efforts to develop efficient and scalable ADF solvers (Linsbichler et al. 2022; Ellmauthaler et al. 2022).

ADF-BN equivalence Recent work by (Heyninck, Knorr, and Leite 2024; Azpeitia et al. 2024) has established a formal correspondence between ADFs and *Boolean Networks* (BNs). As a straightforward yet powerful mathematical formalism, BNs have been extensively applied across science and engineering, particularly within systems biology (Thomas 1973; Rozum et al. 2023). Formally, a BN is a discrete dynamical system comprising n Boolean variables and n corresponding Boolean functions, which dictate discrete state updates according to a specified update scheme. Traditionally, the dynamical behavior of BNs has been analyzed through the lens of *attractors* (minimal inescapable sets of states) (Wang, Saadatpour, and Albert 2012). More recently, the concept of a *trap space* (an inescapable hypercube of states) was introduced (Klarner, Bockmayr, and Siebert 2015) and has since emerged as the central focus in BN analysis (Trinh et al. 2024; Trinh et al. 2025).

The aforementioned studies (Heyninck, Knorr, and Leite 2024; Azpeitia et al. 2024) demonstrate that ADFs inher-

*Corresponding author

ently function as BNs when their acceptance conditions are translated into Boolean functions. This translation reveals two formal bijections: 2-valued models correspond to *fixed points* (singleton attractors), while preferred interpretations map to *minimal* trap spaces. Building upon this equivalence, it follows that admissible interpretations of ADFs represent *all* BN trap spaces, and complete interpretations correspond to *fully percolated* trap spaces (Trinh et al. 2025).

This theoretical connection effectively bridges two previously disjoint fields—computational argumentation and systems biology—facilitating the reuse of algorithms and software tools (Azpeitia et al. 2024). Consequently, many ADF tools can be directly applied to BNs, and vice versa. Given their shared syntax and our primary focus on ADFs, this paper omits formal preliminaries for BNs; readers are referred to (Rozum et al. 2023) for comprehensive details.

Enumeration problems in ADFs and BNs This paper focuses on the enumeration of ADF interpretations or models under various semantics. Whereas decision problems have been widely studied in the literature (Linsbichler et al. 2022; Trinh et al. 2024), enumeration plays a crucial role in practical applications—such as model analysis, explanation, and verification—where exploring the entire solution space is essential (Ellmauthaler et al. 2022; Ellmauthaler and Gerlach 2023b; Ellmauthaler and Gerlach 2023a; Trinh, Benhamou, and Soliman 2023a). Moreover, enumeration is often the computational bottleneck in ADF and BN reasoning tools (Ellmauthaler et al. 2022; Trinh, Benhamou, and Soliman 2023a; Trinh et al. 2024; Azpeitia et al. 2024), where the number of solutions can be exponential and naive generation strategies quickly become infeasible. By targeting enumeration as a first-class objective, we aim to push the boundaries of scalability and completeness in ADF and BN solving.

Most existing ADF solvers rely on SAT or *Answer Set Programming* (ASP) encodings, which—thanks to highly optimized modern solvers—provide strong performance on many benchmarks. Symbolic methods based on *Binary Decision Diagrams* (BDDs) have also been explored, most notably in `adf-bdd` (Ellmauthaler et al. 2022), showing that BDDs can exploit structural regularities that are often missed by SAT/ASP-based approaches. In parallel, the BN community has developed efficient symbolic tools for fixed-point and trap space analysis, such as `bioLQM` (Naldi 2018), and for attractor analysis such as `aeon` (Benes et al. 2020). These address the solution space explosion problem common in BNs due to the large number of input variables, and can be adapted for use on ADFs.

While BDDs offer a promising foundation for symbolic reasoning in ADFs, enabling both efficient analysis and compact representation (Strass 2018; Ellmauthaler et al. 2022), current tools such as `adf-bdd` lack support for the preferred semantics and face scalability challenges (Ellmauthaler et al. 2022). Motivated by the recent ADF–BN equivalence, we introduce `BAss` (BDD-based ADF symbolic solver), a new BDD-based solver for ADFs. It represents each acceptance condition as a BDD, following the idea of `adf-bdd` (Ellmauthaler et al. 2022), but significantly extends the core BDD encoding of `bioLQM` (Naldi 2018) and applies the parametrized principle of the BN attractor analysis tool `aeon` (Benes et al. 2020) to achieve fully symbolic analysis and improved efficiency.

Contribution `BAss` adapts existing symbolic techniques towards the computation of all *admissible*, *complete*, and *preferred* interpretations, as well as *2-valued* and *stable* models. Notably, we introduce novel encoding approaches and optimization strategies, which enable `BAss` to be the first BDD-based ADF solver to support fully symbolic reasoning on the preferred and stable semantics.

We performed an extensive experimental evaluation over 1,200+ benchmark instances from both the BN and ADF domains. The results show that `BAss` dramatically outperforms current BDD-based methods (`adf-bdd`, `bioLQM`, and `aeon`) and is competitive with leading SAT/ASP-based methods, such as `k++adf`, `goDiamond`, `yadf`, `mpbn`, `fASP`, and `ts-conj`—often outperforming them on instances that require analysis of large solution sets.

In particular, `BAss` also brings concrete benefits to the analysis of biological regulatory networks. Thanks to the ADF-BN equivalence, it can enumerate all fixed points or minimal trap spaces of certain Boolean networks that remain unsolvable by existing BN tools. This opens the door to new case studies in systems biology (e.g., model-based phenotype discovery), where the complete characterization of dynamical behaviors is often critical.

The remainder of the paper is organized as follows. Section 2 reviews related work on ADF and BN solvers. Section 3 recalls basic notions of ADFs. Section 4 presents the BDD-based methods over different ADF semantics. Section 5 describes our experiments and evaluation of our method. It also discusses the biological applicability of our method. Section 6 concludes the paper.

2 Related Work

Research on ADF solvers has mainly followed two paradigms: (1) SAT and ASP-based encodings, and (2) symbolic BDD representations.

Among the SAT-based approaches, `k++adf` (Linsbichler et al. 2022) is one of the most prominent solvers. It translates ADF reasoning tasks into propositional satisfiability and supports enumeration under several semantics (admissible, complete, preferred, grounded, stable). It possesses efficient SAT encodings, evidenced by its strong empirical performance on models of the ICCMA competitions (Linsbichler et al. 2022). Other systems such as `goDiamond` (Strass and Ellmauthaler 2017) and `yadf` (Brewka et al. 2020) use ASP encodings, often excelling on small and medium-sized benchmarks but struggling with scalability (Linsbichler et al. 2022; Ellmauthaler et al. 2022).

The first BDD-based solver, `adf-bdd` (Ellmauthaler et al. 2022), represents each acceptance condition as a reduced ordered BDD, enabling the efficient application of the characteristic operator and model evaluation. It supports grounded, complete, and stable semantics,

and implements fixed-point iteration and model enumeration via symbolic restriction and heuristic-guided search; however, the preferred semantics has not been explored yet. While `adf-bdd` outperforms ASP-based solvers like `yadf` (Brewka et al. 2020) on stable semantics and matches SAT-based tools like `k++adf` (Linsbichler et al. 2022) on grounded semantics, it remains behind the fastest SAT solvers on complete and stable semantics tasks. The tool is implemented in Rust, is open-source, and offers both command-line and web interfaces (Ellmauthaler and Gerlach 2023b). In addition, BDDs enable straightforward graphical visualization of the underlying ADFs and their solutions in `adf-bdd`. This showcases the potential of BDDs towards better explainability and debugging of ADFs (Ellmauthaler and Gerlach 2023a).

In the BN community, tools such as `bioLQM` (Naldi 2018) and `pyboolnet` (Klarner, Bockmayr, and Siebert 2015; Klarner, Streck, and Siebert 2017) support the symbolic computation of fixed points and (minimal or maximal) trap spaces using BDDs or ASP encodings. `pyboolnet` also implements an *Integer Linear Programming* (ILP) encoding, and shows that this encoding is surpassed by the ASP one (Klarner, Bockmayr, and Siebert 2015). However, the ASP and ILP encodings of `pyboolnet` require the computation of prime implicants for each Boolean function, which is a bottleneck since computing even a single prime implicant is NP-hard and the total number of prime implicants can be exponential in the number of function inputs (Chandra and Markowsky 1978). `bioLQM` avoids the prime-implicant computation as it characterizes the set of generic trap spaces of a BN using a BDD, then filters this set to get the set of all minimal trap spaces. However, its main issue is that the number of such generic BN trap spaces may be significantly larger than the number of desirable (e.g., minimal) trap spaces (Naldi 2018; Trinh, Benhamou, and Soliman 2023b). This BDD-based filtering approach has also been implemented in the tool `aeon` (Benes et al. 2022) with specific optimizations targeting large solution spaces arising in networks with many input nodes. Recently, an efficient SAT-based encoding, called `H.E.`, has been proposed for BNs, but it is limited to fixed points (Higuchi et al. 2025).

Subsequent ASP encodings (Paulevé et al. 2020; Trinh, Benhamou, and Paulevé 2024; Trinh, Benhamou, and Soliman 2023b) that were proposed to overcome the prime-implicant bottleneck can still suffer from scalability and efficiency issues. For very large and complex models, these still require the *disjunctive normal forms* of all the Boolean functions (and their negations) of the original BN. For fixed points, the ASP encoding `fASP` (Trinh, Benhamou, and Soliman 2023a) uses a *negation normal form* for each Boolean function, which is much more efficient to obtain. This encoding was later generalized for minimal trap spaces in `ts-conj` (Trinh et al. 2024); however, when dealing with *unsafe formulas* (quite rare in real-world models), it still requires a disjunctive normal form to ensure correctness. Notably, in connection to ADFs, the experiments of (Azpeitia et al. 2024) show that `ts-conj` outperforms `k++adf` on large-scale BN and ADF models.

`BAss` builds on the strengths of both symbolic and logic-based approaches by integrating BDD-based reasoning with insights from the BN domain. Unlike `adf-bdd`, which applies BDDs directly to ADF semantics, `BAss` leverages the BN-ADF correspondence to import advanced computation techniques from tools like `aeon` (Benes et al. 2020) (dealing with the case of many input nodes) or `bioLQM`.

3 Preliminaries

In this section, we briefly recall the syntax and semantics of Abstract Dialectical Frameworks (ADFs).

Let $\mathbb{B}_\star = \{0, 1, \star\}$ be the 3-valued domain where 0, 1, and $\star$ denote the *false*, *true*, and *unknown* values, respectively. Then $\mathbb{B} = \{0, 1\}$ is the 2-valued (Boolean) domain. The logical connectives used in this paper include $\wedge$ (*and*), $\vee$ (*or*), $\neg$ (*negation*), $\Rightarrow$ (*implication*), and $\Leftrightarrow$ (*equivalence*).

Definition 1 (Brewka and Woltran 2010)**.** *An ADF is a tuple $D := (S, C)$ where S is a* finite *set of arguments and $C := \{\varphi_s\}_{s \in S}$ consists of acceptance conditions (one for each argument in S), corresponding to propositional formulas $\varphi_s := s' \in S \mid 0 \mid 1 \mid \neg\varphi \mid (\varphi \circ \varphi)$, such that $\circ$ can be one of the binary operators $\wedge, \vee, \Rightarrow, \Leftrightarrow$.*

The original definition of (Brewka and Woltran 2010) also assumes a link relation of dependencies between arguments to mirror the attack relation of AFs (Dung 1995). However, this relation can be inferred from the acceptance conditions and is therefore commonly omitted in recent literature.

While the semantics of AFs is based on sets of arguments (Dung 1995), that of ADFs is based on 3-valued interpretations (Brewka et al. 2013). A 3-valued *interpretation* is a mapping $I\colon S \to \mathbb{B}_\star$; it becomes a 2-valued interpretation if $I(s) \neq \star$ for each $s \in S$. An interpretation represents a belief assignment over the ADF arguments, with values mapping to "argument holds" (1), "argument does not hold" (0), and "unknown" ($\star$).

An *information ordering* $\leq_i$ is defined as the reflexive transitive closure of $<_i$ where $<_i$ is an ordering on $\mathbb{B}_\star$ as follows: $\star <_i 0$, $\star <_i 1$, and there is no other relation. Intuitively, values 0 and 1 carry more information than $\star$.

The information ordering extends to 3-valued interpretations by $I_1 \leq_i I_2$ iff $I_1(s) \leq_i I_2(s)$ for each $s \in S$, and $I_1 <_i I_2$ iff $I_1 \leq_i I_2$ and there is some s such that $I_1(s) <_i I_2(s)$. Intuitively, $I_1 \leq_i I_2$ if for every $s \in S$, either $I_1(s) = \star$, or $I_1(s) = I_2(s)$. That is, I_2 can only differ from I_1 in arguments that are unknown in I_1.

Then, the *partial evaluation* $\varphi[I]$ of φ w.r.t. I is defined by $\varphi[I] := \varphi[s/I(s) \mid s \in S,\ I(s) \in \mathbb{B}]$. That is, $\varphi[I]$ is the propositional formula obtained from φ by substituting each argument s that has a known Boolean value in I with this Boolean value $I(s)$. Note that if I is a 2-valued interpretation, $\varphi[I]$ always simplifies to 1 or 0. We then write $\models \varphi$ to denote that for every 2-valued interpretation I, $\varphi[I]$ simplifies to 1 (in other words, φ is a tautology on 2-valued interpretations).

Finally, the *characteristic operator* $\Gamma_D(I) = I'$ uniquely defines the 3-valued interpretation I' as follows: $I'(s) = 1$ iff $\models \varphi_s[I]$ ($\varphi_s[I]$ is a 2-valued tautology), $I'(s) = 0$ iff

$\models \neg\varphi_s[I]$ ($\varphi_s[I]$ is a 2-valued contradiction), and $I'(s) = \star$ otherwise. We are now able to define the semantics of ADFs:

Definition 2 (Brewka and Woltran 2010)**.** *Let $D = (S, C)$ be an ADF, and I a 3-valued interpretation; then I is:*

- Admissible *iff $I \leq_i \Gamma_D(I)$. Intuitively, any belief held by such I is fully justified by the acceptance conditions of D.*
- Complete *iff $I = \Gamma_D(I)$. Fixed point of Γ_D; includes all additional beliefs that logically follow from I itself.*
- Grounded *iff I is $\leq_i$-minimal complete interpretation. The least committed point of view admitting the maximum number of $\star$ arguments.*
- Preferred *iff I is $\leq_i$-maximal admissible interpretation. The most informative admissible interpretation with the least number of $\star$ arguments.*
- 2-valued model *iff I is complete and 2-valued. In other words, a complete interpretation with no $\star$ arguments.*

Finally, we recall the definition of a stable model, which is inspired by the *stable model semantics* in logic programming (Gelfond and Lifschitz 1988).

Definition 3 (Brewka and Woltran 2010)**.** *Let $D = (S, C)$ be an ADF and I be an arbitrary but fixed 2-valued interpretation of D. First, we define the* reduced ADF $D^I := (S^I, C^I)$ *where $S^I := \{s \in S \mid I(s) = 1\}$, $C^I := \{\varphi_s^I\}_{s \in S^I}$ and $\varphi_s^I = \varphi_s[s'/0 \text{ for } s' \in S \text{ s.t. } I(s') = 0]$. I is a stable model of D iff I is a 2-valued model of D and for the grounded interpretation G of D^I, we have that $I(s) = 1$ implies $G(s) = 1$. Intuitively, all held beliefs in such I are well-founded (i.e., free of circular reasoning). Every argument that holds is justified by an acyclic chain of support.*

We denote by $\texttt{2m}(D)$ and $\texttt{stb}(D)$ the sets of 2-valued and stable models of D, respectively; by $\texttt{ad}(D)$, $\texttt{co}(D)$, $\texttt{gr}(D)$, and $\texttt{pr}(D)$ we denote the sets of admissible, complete, grounded, and preferred interpretations of D, respectively. It has been shown that $\texttt{stb}(D) \subseteq \texttt{2m}(D) \subseteq \texttt{pr}(D) \subseteq \texttt{co}(D) \subseteq \texttt{ad}(D)$, as well as $\texttt{gr}(D) \subseteq \texttt{co}(D)$, and $|\texttt{gr}(D)| = 1$ (Brewka et al. 2013). For the 2-valued model semantics, the enumeration of all 2-valued models is generally intractable; its decision (resp. counting) version is NP-complete (resp. #P-complete) in general (Brewka et al. 2013). Deciding whether an ADF has a stable model is Σ_2^{P}-complete (Brewka et al. 2013), but it remains an open problem to precisely classify the enumeration complexity (or counting complexity) of stable models in ADFs. This applies to the complete and preferred semantics too. Overall, the enumeration problem in ADFs is very challenging, in particular for large-scale problem instances.

Example 1. *Consider the ADF $D = (S, C)$ with $S = \{a, b, c\}$ and $\varphi_a = 1, \varphi_b = \neg a \vee c, \varphi_c = b$. It has five admissible interpretations:*

$$I_1 = \{a \mapsto 1, b \mapsto 0, c \mapsto 0\}, I_2 = \{a \mapsto 1, b \mapsto 1, c \mapsto 1\},$$
$$I_3 = \{a \mapsto 1, b \mapsto \star, c \mapsto \star\},$$
$$I_4 = \{a \mapsto \star, b \mapsto \star, c \mapsto \star\}, I_5 = \{a \mapsto \star, b \mapsto 1, c \mapsto 1\}.$$

Here, I_1, I_2, and I_3 are complete interpretations, such that I_1 and I_2 are preferred, but also 2-valued models. The 2-valued model I_2 is not stable because the reduced ADF D^{I_2} where $D^{I_2} = D$ has the grounded interpretation $\{a \mapsto 1, b \mapsto \star, c \mapsto \star\}$, which does not satisfy the last condition of Definition 3. Only I_1 is a stable model of D.

4 Analysis Methods

In this section, we introduce the methods used by `BAss`. The primary goal of `BAss` in each case is to directly characterize the result set of the reasoning problem using a BDD, providing a compact representation of the full solution set while avoiding enumeration. Proofs are delegated to the *Supplementary Material* (Pastva and Trinh 2026).

4.1 Symbolic Representation of ADFs

A reduced, ordered BDD (Bryant 1986) is a rooted, directed, acyclic graph with two terminal nodes ($\mathbf{0}$ and $\mathbf{1}$) that encodes a Boolean function $f : \mathbb{B}^k \to \mathbb{B}$. Each non-terminal u (also called a *decision node*) has two successors, a positive and a negative one, denoted $pos(u)$ and $neg(u)$. Each such u also has an associated decision variable $var(u) \in \{1, \ldots, k\}$ corresponding to the inputs of f. On every path in this graph, each decision variable must appear at most once, and the variables respect a fixed linear order. Each u then represents a function $f_u : \mathbb{B}^k \to \mathbb{B}$ via the Shannon expansion:

$$f_u(x) = (x_{var(u)} \wedge f_{pos(u)}(x)) \vee (\neg x_{var(u)} \wedge f_{neg(u)}(x))$$

For the terminal nodes, we assume $f_{\mathbf{0}}(x) = 0$ and $f_{\mathbf{1}}(x) = 1$. A BDD F is a symbolic encoding of a Boolean function f if $f = f_{root(F)}$. Intuitively, to *evaluate* f using its BDD, we start at the root and, at each decision node u, pick the successor based on the value of $x_{var(u)}$ until we reach a terminal node, which determines the output. For additional details, we refer readers to (Clarke et al. 2018), which provides a comprehensive overview of the data structure and its applications in symbolic graph analysis.

Assuming a fixed variable ordering, a BDD representation is canonical (i.e., a specific f is always represented by the same BDD). In the following, we thus largely consider BDDs interchangeable with Boolean functions, and unless stated otherwise, each Boolean function is represented by its BDD. However, in the worst case, the size of the BDD (the number of its nodes) can be exponential in k (the number of function inputs). Given BDDs A and B, we can compute the BDD of $A \circ B$ in $\mathcal{O}(|A| \cdot |B|)$, where $\circ$ is any binary Boolean operator and $|A|$ denotes the number of nodes in A. While BDDs can often represent complex Boolean functions compactly, keeping the BDD size as small as possible is of utmost importance due to this quadratic complexity.

BDDs allow for the existential quantification of an input:

$$\exists x_i.\, f(x) = f(x[i/1]) \vee f(x[i/0])$$

Here, $x[i/b]$ is a copy of x with the i-th position set to b. Notice that $g(x) = \exists x_i.\, f(x)$ no longer depends on its i-th argument. Consequently, we can equivalently interpret g as a function over the domain $\mathbb{B}^{k-1}$. As we will discuss shortly, this is critical when symbolically encoding Boolean relations, because existential quantification allows us to perform projections onto specific components of the relation.

Symbolic ADF encoding Assume ADF $D = (S, C)$ with an arbitrary but fixed ordering of arguments $S = \{s_1, \ldots, s_n\}$. Then, we observe that a set $X \subseteq \mathbb{B}^n$ of 2-valued interpretations can be represented by its characteristic function $f_X : \mathbb{B}^n \to \mathbb{B}$ s.t. $f_X(x) = 1$ iff $x \in X$. In that regard, every acceptance condition $\varphi_s : \mathbb{B}^n \to \mathbb{B}$ also trivially has a corresponding BDD representation. To represent 3-valued interpretations, we define a *dual encoding* using additional Boolean variables $(s^\top, s^\bot)$ defined for each $s \in S$, such that $(1,0) \equiv 1$, $(0,1) \equiv 0$, and $(1,1) \equiv \star$. Here, $(0,0)$ is an invalid combination which we exclude by requiring $s^\top \vee s^\bot$ in all symbolic representations of 3-valued interpretations. Using this encoding, a set $Y \subseteq \mathbb{B}^n_\star$ can be represented as a function $f_Y : \mathbb{B}^{2n} \to \mathbb{B}$. Finally, an arbitrary relation R between 2-valued and 3-valued interpretations can be encoded via a BDD using a characteristic function $f_R : \mathbb{B}^n \times \mathbb{B}^{2n} \to \mathbb{B}$ (or $f_R : \mathbb{B}^{3n} \to \mathbb{B}$). For the sake of readability, we will consider interpretations to be interchangeable with their Boolean encodings: when I is an interpretation, we can write $f(I)$ to denote "function f evaluated on the Boolean encoding of I", as long as f is of the correct type (i.e., $f : \mathbb{B}^n \to \mathbb{B}$ for 2-valued, and $f : \mathbb{B}^{2n} \to \mathbb{B}$ for 3-valued interpretations).

On multi-valued decision diagrams Note that three-valued decision diagrams (DDs) could be also used to encode sets of 3-valued interpretations. Assuming $s^\top$ and $s^\bot$ follow each other in the variable ordering, a single three-valued node could replace up to three standard BDD nodes, resulting in a constant-factor DD size reduction. Such an approach is always a trade-off between DD size and internal overhead of the implementation, as considering additional levels complicates the decision node implementation. Here, we chose the BDD encoding mainly due to more mature and widespread support for BDD implementations compared to multi-valued DDs. Do note that this choice does not affect the asymptotic complexity of our algorithms in terms of the number of symbolic DD operations.

Symbolic computation of characteristic operator In addition to acceptance conditions and sets of interpretations, dual encoding allows us to directly represent the characteristic operator Γ_D. Specifically, let $f : \mathbb{B}^n \to \mathbb{B}$ be a function over the arguments of D. Then, let us define:

$$dual(f) = \exists s_1, \ldots, s_n. \left(f(s_1, \ldots, s_n) \wedge \left(\bigwedge_{s_i \in S} (s_i \Rightarrow s_i^\top) \wedge (\neg s_i \Rightarrow s_i^\bot) \right) \right) \quad (1)$$

Here, note that while the whole expression uses $s_i^\top$, $s_i^\bot$ as well as s_i, due to the existential quantification, $dual(f)$ only depends on $s_i^\top$ and $s_i^\bot$. As such, we can treat $dual(f)$ as a function $dual(f) : \mathbb{B}^{2n} \to \mathbb{B}$ which is satisfied by a 3-valued interpretation $I \in \mathbb{B}^n_\star$ iff there exists a 2-valued interpretation $I' \in \mathbb{B}^n$ s.t. $I \leq_i I'$ and $f(I') = 1$. In other words, $dual(f)$ is satisfied by every 3-valued interpretation that can be narrowed down to a 2-valued interpretation satisfying f. With this in mind, let us define $(\varphi_s^\top, \varphi_s^\bot)$ as $(dual(\varphi_s), dual(\neg\varphi_s))$ for every $s \in S$. Now, let I be a 3-valued interpretation and let $(a, b) = (\varphi_s^\top(I), \varphi_s^\bot(I))$. It then trivially follows that $\Gamma_D(I) = 1$ iff $(a, b) = (1, 0)$, $\Gamma_D(I) = 0$ iff $(a, b) = (0, 1)$, and $\Gamma_D(I) = \star$ iff $(a, b) = (1, 1)$. In other words, the pairs of functions $(\varphi_s^\top, \varphi_s^\bot)$ together define the dual encoding of $\Gamma_D : \mathbb{B}^n_\star \to \mathbb{B}^n_\star$.

Example 2. *Recall the ADF from Example 1, where $\varphi_a = 1$, $\varphi_b = \neg a \vee c$, and $\varphi_c = b$. The dual encoding of Γ_D is then defined by $(\varphi_a^\top, \varphi_a^\bot) = (1, 0)$, $(\varphi_b^\top, \varphi_b^\bot) = (a^\bot \vee c^\top, a^\top \wedge c^\bot)$, and $(\varphi_c^\top, \varphi_c^\bot) = (b^\top, b^\bot)$. Using this representation of Γ_D, it is easy to verify that $I_5 = \{a \mapsto \star, b \mapsto 1, c \mapsto 1\}$ is not complete, but is admissible: $(\varphi_a^\top(I_5), \varphi_a^\bot(I_5)) = (1, 0) \equiv 1 \not\equiv \star$ (but $\star \leq_i 1$), $(\varphi_b^\top(I_5), \varphi_b^\bot(I_5)) = (1, 0) \equiv 1$, and $(\varphi_c^\top(I_5), \varphi_c^\bot(I_5)) = (1, 0) \equiv 1$.*

This dual encoding of acceptance conditions is widely used in BN tools (`aeon`, `ts-conj`, `mpbn`, and `bioLQM` all use it to some extent). However, in all of these tools, the translation from f to $dual(f)$ is done *syntactically*, typically by constructing the DNF representation of f and then substituting positive literals for $s^\top$ and negative literals for $s^\bot$. Tools like `mpbn` and `ts-conj` try to avoid this translation in certain special cases, but in the worst case default to DNF. This translation of φ into DNF is often a bottleneck, as it can exponentially increase the size of the formula (Trinh et al. 2024). While conceptually simple, we believe to be the first in this context to define this *direct* propositional mapping (Equation 1). This allows us to construct the BDDs of $\varphi^\top$ and $\varphi^\bot$ fully symbolically by transforming φ instead of translating it into DNF. As an initial experiment, we implemented this singular optimization into the tool `aeon`, which is included in the final evaluation.

4.2 Computation of 2-valued Models

Given ADF $D = (S, C)$, the symbolic characterization of 2-valued models (i.e. the set $\texttt{2m}(D)$) is straightforward. The following Boolean function is satisfied exactly by all 2-valued models of D:

$$f_{\texttt{2m}(D)} = \bigwedge_{s_i \in S} s_i \Leftrightarrow \varphi_{s_i} \quad (2)$$

This characterization is a well-known fact both in the ADF and BN communities and its correctness trivially follows from the observation that $\Gamma_D(I)(s) = \varphi_s(I)$ when I is a 2-valued interpretation. Furthermore, the BDD of this formula can be constructed using just $\mathcal{O}(|S|)$ BDD operations.

Conjunction optimization Here, the intermediate results of applying the conjunction operator can be often much larger than the final BDD of $f_{\texttt{2m}(D)}$. To mitigate this problem, we utilize a simple greedy optimization scheme (Algorithm 1), which increases the number of BDD operations to $\mathcal{O}(|S|^2)$, but is in practice often much faster because it can maintain dramatically smaller intermediate BDD sizes. Usage of such optimization schemes is fairly well known in the BDD community. However, we are not aware of other

Algorithm 1 Greedy optimization algorithm for performing n-ary conjunction of BDDs.

Require: BDD collection Q (e.g. $\{a \Leftrightarrow \varphi_a \mid a \in S\}$)
1: $R \leftarrow \top$
2: **while** $Q \neq \emptyset$ **do**
3: $O \leftarrow X \in Q$ s.t. BDD size $|X \wedge R|$ is minimal
4: $R \leftarrow R \wedge O$
5: $Q \leftarrow Q \setminus \{O\}$
6: **end while**
7: **return** R

BDD-based tools in this area performing such optimization, thus we feel it should be highlighted. In the following, we use Algorithm 1 to construct the BDD of every similar n-ary conjunction operator.

4.3 Computation of Admissible and Complete Interpretations

Using the dual encoding defined at the beginning of this section, the characterization of admissible and complete interpretations (i.e. the sets $\mathtt{ad}(D)$ and $\mathtt{co}(D)$) is similar to that of 2-valued models:

$$f_{\mathtt{ad}(D)} = \bigwedge_{s_i \in S} ((\varphi_{s_i}^{\top} \Rightarrow s_i^{\top}) \wedge (\varphi_{s_i}^{\bot} \Rightarrow s_i^{\bot})) \tag{3}$$

$$f_{\mathtt{co}(D)} = \bigwedge_{s_i \in S} ((\varphi_{s_i}^{\top} \Rightarrow s_i^{\top}) \wedge (\varphi_{s_i}^{\bot} \Rightarrow s_i^{\bot}) \wedge ((s_i^{\top} \wedge s_i^{\bot}) \Rightarrow (\varphi_{s_i}^{\top} \wedge \varphi_{s_i}^{\bot}))) \tag{4}$$

Proposition 1. *The above BDD-based characterizations of $\mathtt{ad}(D)$ and $\mathtt{co}(D)$ are correct, in the sense that the models of $f_{\mathtt{ad}(D)}$ (resp. $f_{\mathtt{co}(D)}$) correspond exactly to the admissible (resp. complete) interpretations of the given ADF D.*

4.4 Computation of Preferred Interpretations

For preferred interpretations, the computation is more involved, as we need to select the $\leq_i$-maximal interpretations from the set $\mathtt{co}(D)$. A naive, often adopted approach, is to iteratively build a set C of candidate interpretations using the following method: select an interpretation I from $\mathtt{co}(D)$, remove all $I' \in C$ and $I' \in \mathtt{co}(D)$ such that $I' <_i I$, then insert I into C. Upon termination, the set C contains exactly the $\leq_i$-maximal interpretations. However, this procedure requires anywhere from $|\mathtt{pr}(D)|$ to $|\mathtt{co}(D)|$ iterations to converge, which can be intractable for large solution sets. Here, we instead present a symbolic algorithm which avoids this bottleneck, requiring only $|S|$ iterations.

Our method is based on the following procedures, which are individually not new, but represent a somewhat less common usage of BDDs:

- Given a Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ represented as a BDD B, it is possible to compute a satisfying valuation $x_1, \ldots, x_n$ (i.e. $f(x_1, \ldots, x_n) = 1$) with the least number of positive literals (i.e. $|\{i \mid x_i = 1\}|$ is minimal) in time $\mathcal{O}(|B|)$. We denote this operation $\mathit{leastVal}(f)$.

Algorithm 2 Iterative symbolic algorithm to compute the preferred interpretations $\mathtt{pr}(D)$ based on the known complete interpretations $\mathtt{co}(D)$.

Require: BDD representing the set $\mathtt{co}(D)$
1: $\mathtt{pr}(D) \leftarrow \bot$
2: **while** $\mathtt{co}(D) \neq \bot$ **do**
3: $I \leftarrow \mathit{leastVal}(\mathtt{co}(D))$
4: $k \leftarrow |\{i \mid I(s_i^{\top}) = 1 \wedge I(s_i^{\bot}) = 1\}|$
5: $\mathit{prf} \leftarrow \mathtt{co}(D) \wedge \#_k(s_1^{\top} \wedge s_1^{\bot}, \ldots, s_n^{\top} \wedge s_n^{\bot})$
6: $\mathtt{pr}(D) \leftarrow \mathtt{pr}(D) \vee \mathit{prf}$
7: $\mathtt{co}(D) \leftarrow \mathtt{co}(D) \wedge \neg \mathit{weakening}(\mathit{prf})$
8: **end while**
9: **return** $\mathtt{pr}(D)$

- Given Boolean functions $f_1, \ldots, f_m$, we can build the function $f_{\#k}$ satisfied iff exactly k functions from this list are satisfied, i.e. $f_{\#k}(x_1, \ldots, x_n) = (k = \sum_{i=1}^{m} f_i(x_1, \ldots, x_n))$. Note that when f_i are literals (i.e. $f_i = x_i$), the size and construction time of such BDD are both known to be in $\mathcal{O}(m \cdot k)$. We denote this operation $\#_k(f_1, \ldots, f_m)$.
- Given a Boolean function $f : \mathbb{B}^n \to \mathbb{B}$, we can compute (in $\mathcal{O}(n)$ BDD operations) a function $f' : \mathbb{B}^n \to \mathbb{B}$ such that $f'(y) = 1$ iff there exists some x such that $f(x) = 1$ and $\{i \mid x_i = 1\} \subseteq \{i \mid y_i = 1\}$. We denote this operation $\mathit{weakening}(f)$.

In the context of dual symbolic encoding of ADFs, we can assign a more intuitive meaning to these three operations: $\mathit{leastVal}(f)$ is $\leq_i$-maximal interpretation out of all interpretations satisfying f (specifically an interpretation with the least number of $\star$ values). Using $f_i = s_i^{\top} \wedge s_i^{\bot}$, the function $\#_k(f_1, \ldots, f_n)$ encodes exactly the 3-valued interpretations with k arguments set to $\star$. Finally, $\mathit{weakening}(f)$ encodes the set of interpretations that "weaken" the interpretations of f by replacing an arbitrary number of fixed values with $\star$. In other words, if $f(I) = 1$, then $\mathit{weakening}(f)(I') = 1$ for every $I' \leq_i I$. Using these operations, we can then build a fully symbolic optimization procedure (see Algorithm 2).

Proposition 2. *Algorithm 2 correctly computes the set of preferred interpretations $\mathtt{pr}(D)$ of an ADF D, based on its set of complete interpretations $\mathtt{co}(D)$, using symbolic operations over BDDs.*

4.5 Computation of Stable Models

Finally, we discuss our fully symbolic method for computing stable 2-valued models. This method follows the principles established in the computation of preferred models, but extends them to the stable 2-valued semantics. To start, we recall the following proposition from literature:

Definition 4. *Consider an ADF $D = (S, C)$. Given two interpretations I_1 and I_2 of D, we say $I_1 \leq_t I_2$ iff $I_1(s) = 1$ implies $I_2(s) = 1$ for all $s \in S$.*

Proposition 3 (Proposition 3.8 of (Strass 2013))**.** *Consider an ADF $D = (S, C)$. Given two stable models I_1 and I_2 of D, it holds that $I_1 \leq_t I_2$ implies $I_1 = I_2$.*

Algorithm 3 Symbolic algorithm computing the stable models $\texttt{stb}(D)$ based on the 2-valued models $\texttt{2m}(D)$.

Require: BDD representing the set $\texttt{2m}(D)$
$\texttt{stb}(D) \leftarrow \bot$
while $\texttt{2m}(D) \neq \bot$ **do**
  $I \leftarrow \mathit{leastVal}(\texttt{2m}(D))$
  $k \leftarrow |\{i \mid I(s_i) = 1\}|$
  $\mathit{min} \leftarrow \texttt{2m}(D) \wedge \#_k(s_1, \ldots, s_n)$
  $\texttt{stb}(D) \leftarrow \texttt{stb}(D) \vee \mathit{min}$
  $\texttt{2m}(D) \leftarrow \texttt{2m}(D) \wedge \neg \mathit{weakening}(\mathit{min})$
end while
$\mathit{Rel}_1 \leftarrow \bigwedge_{s \in S}(s \Rightarrow (s^\top \wedge s^\bot)) \wedge (\neg s \Rightarrow (\neg s^\top \wedge s^\bot))$
$\texttt{stb}(D) \leftarrow \texttt{stb}(D) \wedge \mathit{Rel}_1$
repeat
  for $s \in S$ **do**
    $U \leftarrow \texttt{stb}(D) \wedge (s^\top \wedge s^\bot) \wedge (\varphi_s^\top \wedge \neg \varphi_s^\bot)$
    $\texttt{stb}(D) \leftarrow (\texttt{stb}(D) \wedge \neg U) \vee U[s^\bot / \neg s^\bot]$
  end for
until fixed-point
$\mathit{Rel}_2 \leftarrow \bigwedge_{s \in S}(s \Rightarrow (s^\top \wedge \neg s^\bot))$
return $\exists s_1^\top, s_1^\bot, \ldots, s_n^\top, s_n^\bot . (\texttt{stb}(D) \wedge \mathit{Rel}_2)$

Intuitively, this proposition states that stable models are a subset of 2-valued models minimal in terms of their true arguments (that is, w.r.t. $\leq_t$). It is not necessarily true that every such minimal 2-valued model is a stable model; we still have to account for the grounded interpretation of the reduced ADF. However, we can speed up the stable model search significantly by pre-selecting 2-valued models minimal w.r.t. $\leq_t$. This provides an advantage compared to tools such as `k++adf`, which computes stable models by checking each 2-valued model. Furthermore, to search for such minimal models, we can adapt the symbolic optimization method that we propose for preferred interpretations.

To complete the procedure, we need to compute the grounded interpretation of the reduced ADF corresponding to each candidate 2-valued model. To do this, we combine the direct and dual encoding of ADFs within one BDD, allowing us to represent a relation between 2-valued models and their corresponding 3-valued interpretations in the reduced ADF. Using this combined representation we can compute grounded models G corresponding to 2-valued models I and only select those combinations where $I(s) = 1$ implies $G(s) = 1$. As the final step, we project our symbolic representation back to the direct encoding, obtaining the final stable models. The complete procedure is presented in Algorithm 3.

Proposition 4. *Algorithm 3 correctly computes the set of stable models $\texttt{stb}(D)$ of an ADF D, based on its set of 2-valued models $\texttt{2m}(D)$ and the grounded interpretations of their corresponding reduced ADFs.*

4.6 Implementation Notes

Free input arguments An ADF can contain a *free input* argument s where $\varphi_s = s$ (called an *input variable* in BNs). This is especially common in ADFs based on Boolean networks, where such arguments represent free external stimuli. For such s, we can simplify the optimization process used in Algorithms 2 and 3. In Algorithm 2, it is clear that no preferred interpretation can have such s set to $\star$ (only 0 or 1 are possible). Similarly, in Algorithm 3, it is clear that no stable model can have such s set to 1 (only 0 is possible). When computing $\texttt{pr}(D)$ and $\texttt{stb}(D)$, our implementation therefore restricts $\texttt{co}(D)$ and $\texttt{2m}(D)$ with these additional constraints. These directly propagate into the computation of $\texttt{co}(D)$ and $\texttt{2m}(D)$, simplifying the set of solutions. As such, `BAss` can sometimes compute $\texttt{pr}(D)$ or $\texttt{stb}(D)$ even if it fails to compute $\texttt{co}(D)$ or $\texttt{2m}(D)$.

Implementation The implementation of `BAss` originally started within the `aeon` toolbox, which we previously enhanced with the fully symbolic BDD transformation of acceptance conditions into dual encoding and greedy conjunction optimization. However, this is the first publication where this enhancement is presented and tested. Compared to this initial implementation, `BAss` also offers the presented fully symbolic computation of preferred and stable models, which are not present in `aeon`. Finally, `BAss` also adopts a novel BDD library called `ruddy` which implements CPU cache-friendly optimizations of the BDD data structure from (Pastva and Henzinger 2023). Compared to `aeon`, this makes `BAss` a fully-featured ADF solver. `BAss` is available on GitHub and is written entirely in Rust.

5 Evaluation

Tools To evaluate the performance of `BAss`, we compare its runtime against the current fastest ADF solvers: `k++adf` (Linsbichler et al. 2022) (version 2021-03-31), `goDiamond` (Strass and Ellmauthaler 2017) (version 2.0.2), `yadf` (Brewka et al. 2020) (version 2.2) and `adf-bdd` (Ellmauthaler et al. 2022) (version 0.3.1). Furthermore, we test the following state-of-the-art BN analysis tools that can be used to solve certain ADF problems: `aeon` (Benes et al. 2022) (version 1.2.5), `ts-conj` (Trinh et al. 2024) (version 2024-10-29), `fASP` (Trinh, Benhamou, and Soliman 2023a) (version 0.3.0), `mpbn` (Trinh, Benhamou, and Paulevé 2024) (version 0.4.1), `bioLQM` (Naldi 2018) (version 0.6.1, using BDDs) and `H.E.` (version 1.1) (Higuchi et al. 2025). Since these tools can typically only solve a subset of ADF problems, we provide an overview of the tested combinations in Table 1. All BDD-based tools order the variables as presented in the input file and we do not perform any specific tuning in this regard.

Test instances Our benchmark dataset consists of 1,237 ADF instances. Out of these, 245 are Boolean networks from the *Biodivine Boolean Models* (BBM) dataset (Pastva et al. 2023) converted into the ADF format. The rest are benchmarks previously used in (Brewka et al. 2020; Linsbichler et al. 2022; Ellmauthaler et al. 2022). Specifically, this includes (a) test instances based on the benchmarks from ICCMA 2017 (601 instances; from the `yadf`

	BAss	k++adf	goDiamond	yadf	adf-bdd	aeon	ts-conj	fASP	mpbn	bioLQM	H.E.
admiss.	✓	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗
complete	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✗
preferred	✓	✓	✓	✓	✗	✓	✓	✗	✓	✓	✗
2-val.	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
stable	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗

Table 1: Overview of the tested tools and ADF problems that they can solve out-of-the-box. Tools on the right-hand-side require translation into the `.bnet` BN format.

	BAss	k++adf	goDiam.	yadf	adf-bdd	aeon	ts-conj	fASP	mpbn	bioLQM	H.E.
admiss.	**695 (338)** **991s**	298 (0) 1796s	170 (0) 2063s	201 (0) 1992s	N/A	N/A	N/A	N/A	N/A	292 (0) 1807s	N/A
comp.	**974 (6+79)** **404s**	**883 (23)** **602s**	190 (0) 2023s	**535 (21)** **1320s**	476 (0) 1430s	963 (0) 432s	N/A	N/A	N/A	619 (0) 1136s	N/A
pref.	**982 (24+15)** **389s**	1081 (0) 181s	215 (0) 1958s	342 (0) 1732s	N/A	900 (0) 562s	**1103 (1)** **134s**	N/A	1065 (0) 219s	652 (0) 1067s	N/A
2-val.	**1142 (3)** **52s**	1116 (0) 102s	1130 (0) 74s	N/A	950 (0) 456s	1139 (0) 59s	1104 (0) 134s	1123 (0) 94s	1079 (0) 189s	1038 (0) 272s	1120 (0) 98s
stable	**1149 (50)** **36s**	**1115 (16)** **105s**	202 (0) 2002s	65 (0) 2269s	571 (0) 1249s	N/A	N/A	N/A	N/A	N/A	N/A

Table 2: Summary table of benchmark results. Each cell lists (a) the total number of completed benchmarks; (b) in parentheses, the number of benchmarks *uniquely* solved by that tool; (c) the PAR2 runtime score (in seconds). For `BAss`, we also list (using $+x$) the number of instances solved uniquely by `BAss` and `aeon`, but no other tool. Bold text highlights tools that uniquely solved at least one problem instance.

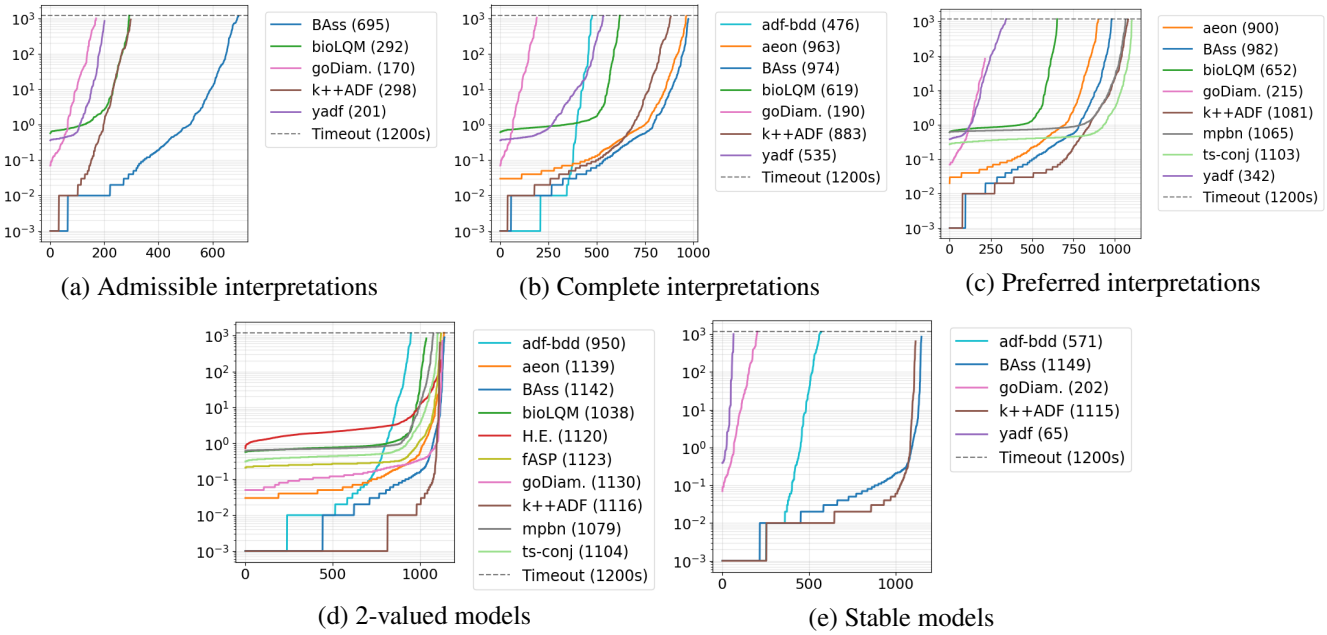
website[1]); (b) test instances based on the ICCMA 2019 competition (276 instances; from the `k++adf` repository[2]); (c) ADFs of metro networks and grids used in (Brewka et al. 2020) (115 instances; from the `yadf` website). We measured up to 1,076 arguments per benchmark instance (135 average; 80 median) and up to 923,346 links (i.e., dependencies between arguments) per benchmark instance (16,470 average; 280 median).

BN conversion For the BN tools, the ADF instances also had to be converted into the `.bnet` format. Here, we encountered a compatibility issue, as `.bnet` does not support the XOR operator, which is prominently used in some of the ADF benchmarks. We attempted to automatically rewrite these ADF instances without using XOR, but in 72 cases, this resulted in extremely large BNs that we had to exclude from the final analysis. To make the presented conclusions fair to all tools, we decided to completely exclude these 72 benchmarks from the final comparison, leaving 1,165 instances. However, we did test ADF tools (including `BAss`) on these inputs, and $52/72$ instances failed across all problems and tools. As such, ADF-specific tools are better at handling large XOR problem instances, but the advantage is not very significant on the benchmarks we have available.

Methodology We ran all experiments on an AMD EPYC 7713 64-core server with 1TB of RAM, such that up to 48 experiments were executed in parallel. For each experiment, we set a time limit of 1,200 seconds. We did observe out-of-memory errors and related segmentation faults for certain tools (namely `yadf`, `k++adf`, `H.E.`, and `bioLQM`). For simplicity, our main results count these as failures (the same way as timeouts), but they are logged separately in the raw dataset available in the attached reproducibility artifact. To compare the overall runtime, we use the PAR2 score, which is commonly used in solver competitions. The PAR2 score is the average runtime across all benchmarks, while counting timeouts or errors as $2 \cdot 1200$ seconds. To mitigate effects of disk I/O and printing operations, the benchmarks only count the solutions (i.e. results are not printed or written to disk). This ensures comparable experimental environment for all tools. As a separate experiment, we compared the full output of all tools for correctness using benchmarks where all tools finished in $\leq 10s$, ensuring tools agree in their output.

Results discussion The measured results are presented in summary Table 2 and Figure 1. First, let us compare the performance of BDD-based tools. Here, `bioLQM` and `adf-bdd` represent the most straightforward implementations, but do not achieve good performance compared to SAT/ASP-based tools. Meanwhile, `aeon` and `BAss` are often comparable to SAT/ASP. In particular, note that `BAss` solves unique benchmarks across all categories. Nevertheless, for complete, preferred, and stable interpretations, SAT/ASP still holds unique advantage in some problem instances (here, the two approaches appear somewhat complementary). Also notice that `BAss` solved the most instances

[1]https://www.dbai.tuwien.ac.at/proj/adf/yadf/
[2]https://bitbucket.org/andreasniskanen/k-adf/



(a) Admissible interpretations (b) Complete interpretations (c) Preferred interpretations

(d) 2-valued models (e) Stable models

Figure 1: Cactus plots showing the number of solved instances (x-axis) of individual tools across time (y-axis; seconds) considering different ADF problem categories.

BBM ID	$\|S\|$	Source	#Preferred	#Complete
222	206	(Ganguli et al. 2015)	687194767360	6699707176948710000
116	144	(Ostaszewski et al. 2020)	423188831391449000	9389013197839268000000000000
223	141	(Weinstein et al. 2017)	408333504	57879370298664
087	133	(Rex et al. 2016)	17179869184	16680058317523500
118	121	(Ostaszewski et al. 2020)	67545198297874400	54071328222060590000000000
112	112	(Ostaszewski et al. 2020)	135266304	7926623204940

Table 3: Largest BBM Boolean networks in terms of arguments (i.e., $|S|$) where `BAss` was uniquely able to identify the whole solution space of complete and preferred interpretations.

(and had the best PAR2 score) in the admissible, complete and stable model categories. For preferred interpretations, `BAss` is limited by its reliance on the full knowledge of $\mathtt{co}(D)$. Let us also compare `BAss` to `aeon` (which represents an experimental implementation of a subset of the presented methods). In the complete and 2-valued categories, `BAss` outperforms `aeon` slightly due to its more efficient BDD implementation, but otherwise relies on the same methodology. In the preferred category, we see a clear improvement due to the new fully symbolic procedure. Finally, we also list the number of benchmarks uniquely solved by `BAss` and `aeon`, but no other tool, as this collectively represents the contribution of methods presented in this paper compared to the remaining tools.

5.1 Biological Applicability

We introduced methods that reveal large sets of preferred interpretations (minimal trap spaces) and complete interpretations (percolated trap spaces). Table 3 showcases the six largest models that can only be handled by `BAss`.

Having these solution sets available provides a significant benefit to the systems biology community. Complete knowledge of minimal trap spaces is crucial when determining the landscape of biological phenotypes in a Boolean model (Klarner, Bockmayr, and Siebert 2015; Klarner et al. 2020) and detecting its *phenotype determining nodes*. Meanwhile, percolated trap spaces (and to some extent all trap spaces) are relevant for BN control, as each trap space matching a desired biological profile can be used to derive a control strategy that drives the network towards this phenotype (Trinh et al. 2025; Tonello and Paulevé 2024; Fontanals, Tonello, and Siebert 2020). Finally, percolated trap spaces are used to form *succession diagrams* (Rozum et al. 2021), which represent the gradual irreversible biological commitments of the model.

Even when the complete solution set is not required, or it is impossible to analyze each solution individually, having the solution set represented as a BDD provides a unique advantage: Due to its structure, one can *uniformly* sample solutions from a BDD. Meanwhile, sampling solutions using a solver is typically biased by the underlying algorithm. As such, BDDs allow statistical analysis on the solution space

not available to solver-based methods.

6 Conclusion

We have presented `BAss`, a new BDD-based reasoning tool for Abstract Dialectical Frameworks that leverages recent theoretical advances connecting ADFs with BNs. `BAss` builds on existing theoretical foundations of `aeon`, `bioLQM` and `adf-bdd`, but provides novel symbolic algorithms and optimization strategies for computing admissible, complete, and preferred interpretations as well as 2-valued and stable models. It is, to the best of our knowledge, the first BDD-based ADF solver to support fully symbolic reasoning on the preferred and stable semantics. Our extensive evaluation over 1,200+ ADF and BN instances demonstrates that `BAss` dramatically outperforms existing BDD-based tools and matches or exceeds state-of-the-art SAT/ASP-based solvers, particularly for models involving large numbers of solutions. In such cases, the BDD-based representation can replace full solution enumeration, allowing, for example, provably uniform random sampling of solutions. This property is crucial to maintain statistical interpretability in biological applications. Our results highlight the promise of symbolic methods as a robust and scalable foundation for ADF reasoning. Notably, we have successfully applied `BAss` to analyze previously unsolved biological networks.

Future work will improve performance by exploring dynamic variable reordering, hybrid symbolic-SAT strategies (e.g., eliminate the need for full computation of $\texttt{co}(D)$ when computing $\texttt{pr}(D)$), and BN reduction (Veliz-Cuba 2011; Tonello and Paulevé 2024). We also plan to expand support to additional semantics such as the naive, stage, and semi-stable semantics (Gaggl, Rudolph, and Straß 2021; Keshavarzi Zafarghandi, Verbrugge, and Verheij 2021).

Availability

`BAss` is available at https://github.com/sybila/biodivine-bass, with a benchmark reproducibility artifact at https://doi.org/10.5281/zenodo.17794231. The appendices are published on arXiv (Pastva and Trinh 2026).

Acknowledgments

We acknowledge Ho Chi Minh City University of Technology (HCMUT), VNU-HCM for supporting this study.

AI Declaration

Generative AI (Cursor) was used during the development and testing of auxiliary features of `BAss`; all core algorithms were written and tested manually. Generative AI (Gemini) was also utilized for proofreading the main text. However, no published content was directly generated by AI.

References

Al-Abdulkarim, L.; Atkinson, K.; and Bench-Capon, T. J. M. 2016. A methodology for designing systems to reason with legal cases using abstract dialectical frameworks. *Artif. Intell. Law* 24(1):1–49.

Atkinson, K.; Baroni, P.; Giacomin, M.; Hunter, A.; Prakken, H.; Reed, C.; Simari, G. R.; Thimm, M.; and Villata, S. 2017. Towards artificial argumentation. *AI Mag.* 38(3):25–36.

Azpeitia, E.; Gutiérrez, S. M.; Rosenblueth, D. A.; and Zapata, O. 2024. Bridging abstract dialectical argumentation and Boolean gene regulation. https://doi.org/10.48550/arXiv.2407.06106.

Bench-Capon, T. J. M., and Dunne, P. E. 2007. Argumentation in artificial intelligence. *Artif. Intell.* 171(10-15):619–641.

Benes, N.; Brim, L.; Kadlecaj, J.; Pastva, S.; and Safránek, D. 2020. AEON: attractor bifurcation analysis of parametrised Boolean networks. In *CAV*, 569–581. Springer.

Benes, N.; Brim, L.; Huvar, O.; Pastva, S.; Safránek, D.; and Smijáková, E. 2022. AEON.py: Python library for attractor analysis in asynchronous Boolean networks. *Bioinform.* 38(21):4978–4980.

Brewka, G., and Woltran, S. 2010. Abstract dialectical frameworks. In *KR*, 102–111. AAAI Press.

Brewka, G.; Strass, H.; Ellmauthaler, S.; Wallner, J. P.; and Woltran, S. 2013. Abstract dialectical frameworks revisited. In *IJCAI*, 803–809. IJCAI/AAAI.

Brewka, G.; Ellmauthaler, S.; Strass, H.; Wallner, J. P.; and Woltran, S. 2017. Abstract dialectical frameworks. an overview. *FLAP* 4(8).

Brewka, G.; Diller, M.; Heissenberger, G.; Linsbichler, T.; and Woltran, S. 2020. Solving advanced argumentation problems with answer set programming. *Theory Pract. Log. Program.* 20(3):391–431.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers* 35(8):677–691.

Cabrio, E., and Villata, S. 2016. Abstract dialectical frameworks for text exploration. In *ICAART*, 85–95. SciTePress.

Chandra, A. K., and Markowsky, G. 1978. On the number of prime implicants. *Discret. Math.* 24(1):7–11.

Charwat, G.; Dvorák, W.; Gaggl, S. A.; Wallner, J. P.; and Woltran, S. 2015. Methods for solving reasoning problems in abstract argumentation - A survey. *Artif. Intell.* 220:28–63.

Clarke, E. M.; Henzinger, T. A.; Veith, H.; Bloem, R.; et al. 2018. *Handbook of model checking*, volume 10. Springer.

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.

Ellmauthaler, S., and Gerlach, L. 2023a. ADF-BDD.DEV: Debug abstract dialectical frameworks with binary decision diagrams. In *XLoKR 2023*.

Ellmauthaler, S., and Gerlach, L. 2023b. ADF-BDD.DEV: insights to undecided statements in abstract dialectical frameworks. In *AI^3*. CEUR-WS.org.

Ellmauthaler, S.; Gaggl, S. A.; Rusovac, D.; and Wallner, J. P. 2022. Representing abstract dialectical frame-

works with binary decision diagrams. In *LPNMR*, 177–189. Springer.

Fontanals, L. C.; Tonello, E.; and Siebert, H. 2020. Control strategy identification via trap spaces in Boolean networks. In *CMSB*, 159–175. Springer.

Gaggl, S. A.; Rudolph, S.; and Straß, H. 2021. On the decomposition of abstract dialectical frameworks and the complexity of naive-based semantics. *J. Artif. Intell. Res.* 70:1–64.

Ganguli, P.; Chowdhury, S.; Bhowmick, R.; and Sarkar, R. R. 2015. Temporal protein expression pattern in intracellular signalling cascade during T-cell activation: A computational study. *J. Biosci.* 40:769–789.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *ICLP*, 1070–1080. MIT Press.

Heyninck, J.; Knorr, M.; and Leite, J. 2024. Abstract dialectical frameworks are Boolean networks. In *LPNMR*, 98–111. Springer.

Higuchi, R.; Soh, T.; Berre, D. L.; Magnin, M.; Banbara, M.; and Tamura, N. 2025. A SAT-based method for counting all singleton attractors in Boolean networks. In *IJCAI*, 2601–2609. ijcai.org.

Keshavarzi Zafarghandi, A.; Verbrugge, R.; and Verheij, B. 2021. Semi-stable semantics for abstract dialectical frameworks. In *KR*, 422–431.

Klarner, H.; Heinitz, F.; Nee, S.; and Siebert, H. 2020. Basins of attraction, commitment sets, and phenotypes of Boolean networks. *IEEE ACM Trans. Comput. Biol. Bioinform.* 17(4):1115–1124.

Klarner, H.; Bockmayr, A.; and Siebert, H. 2015. Computing maximal and minimal trap spaces of Boolean networks. *Nat. Comput.* 14(4):535–544.

Klarner, H.; Streck, A.; and Siebert, H. 2017. PyBoolNet: a python package for the generation, analysis and visualization of Boolean networks. *Bioinform.* 33(5):770–772.

Linsbichler, T.; Maratea, M.; Niskanen, A.; Wallner, J. P.; and Woltran, S. 2022. Advanced algorithms for abstract dialectical frameworks based on complexity analysis of subclasses and SAT solving. *Artif. Intell.* 307:103697.

Naldi, A. 2018. BioLQM: a Java toolkit for the manipulation and conversion of logical qualitative models of biological networks. *Front. Physiol.* 9:1605.

Neugebauer, D. 2017. Generating defeasible knowledge bases from real-world argumentations using D-BAS. In *AI*^{3}*@AI*IA*, 105–110. CEUR-WS.org.

Niskanen, A., and Järvisalo, M. 2020. μ-toksia: An efficient abstract argumentation reasoner. In *KR*, 800–804.

Ostaszewski, M.; Mazein, A.; Gillespie, M. E.; Kuperstein, I.; Niarakis, A.; Hermjakob, H.; Pico, A. R.; Willighagen, E. L.; Evelo, C. T.; Hasenauer, J.; et al. 2020. COVID-19 disease map, building a computational repository of SARS-CoV-2 virus-host interaction mechanisms. *Sci. Data* 7(1):1–4.

Pastva, S., and Henzinger, T. A. 2023. Binary decision diagrams on modern hardware. In *FMCAD*, 122–131. IEEE.

Pastva, S., and Trinh, V.-G. 2026. BAss: Symbolic reasoning in abstract dialectical frameworks. https://arxiv.org/abs/2604.27576.

Pastva, S.; Šafránek, D.; Beneš, N.; Brim, L.; and Henzinger, T. 2023. Repository of logically consistent real-world Boolean network models. *bioRxiv*.

Paulevé, L.; Kolčák, J.; Chatain, T.; and Haar, S. 2020. Reconciling qualitative, abstract, and scalable modeling of biological networks. *Nat. Commun.* 11(1):1–7.

Rex, J.; Albrecht, U.; Ehlting, C.; Thomas, M.; Zanger, U. M.; Sawodny, O.; Häussinger, D.; Ederer, M.; Feuer, R.; and Bode, J. G. 2016. Model-based characterization of inflammatory gene expression patterns of activated macrophages. *PLoS Comput. Biol.* 12(7):e1005018.

Rozum, J. C.; Gómez Tejeda Zañudo, J.; Gan, X.; Deritei, D.; and Albert, R. 2021. Parity and time reversal elucidate both decision-making in empirical models and attractor scaling in critical Boolean networks. *Sci. Adv.* 7(29):eabf8124.

Rozum, J. C.; Campbell, C.; Newby, E.; Nasrollahi, F. S. F.; and Albert, R. 2023. Boolean networks as predictive models of emergent biological behaviors. https://doi.org/10.48550/arXiv.2310.12901.

Strass, H., and Ellmauthaler, S. 2017. GoDIAMOND 0.6. 6–ICCMA 2017 system description, 2017. *Second International Competition on Computational Models of Argumentation: http://www. dbai. tuwien. ac. at/iccma17*.

Strass, H., and Wallner, J. P. 2015. Analyzing the computational complexity of abstract dialectical frameworks via approximation fixpoint theory. *Artif. Intell.* 226:34–74.

Strass, H. 2013. Approximating operators and semantics for abstract dialectical frameworks. *Artif. Intell.* 205:39–70.

Strass, H. 2014. Implementing instantiation of knowledge bases in argumentation frameworks. In *COMMA*, 475–476. IOS Press.

Strass, H. 2018. Instantiating rule-based defeasible theories in abstract dialectical frameworks and beyond. *J. Log. Comput.* 28(3):605–627.

Thimm, M., and Villata, S. 2017. The first international competition on computational models of argumentation: Results and analysis. *Artif. Intell.* 252:267–294.

Thomas, R. 1973. Boolean formalisation of genetic control circuits. *J. Theor. Biol.* 42:565–583.

Tonello, E., and Paulevé, L. 2024. Phenotype control and elimination of variables in Boolean networks. *Peer Community Journal* 4.

Trinh, V.-G.; Benhamou, B.; and Paulevé, L. 2024. mpbn: a simple tool for efficient edition and analysis of elementary properties of Boolean networks. https://doi.org/10.48550/arXiv.2403.06255.

Trinh, V.-G.; Benhamou, B.; and Soliman, S. 2023a. Efficient enumeration of fixed points in complex Boolean networks using answer set programming. In *CP*, 35:1–35:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Trinh, V.-G.; Benhamou, B.; and Soliman, S. 2023b. Trap spaces of Boolean networks are conflict-free siphons of their Petri net encoding. *Theor. Comput. Sci.* 971:114073.

Trinh, V.-G.; Benhamou, B.; Pastva, S.; and Soliman, S. 2024. Scalable enumeration of trap spaces in Boolean networks via answer set programming. In *AAAI*, 10714–10722. AAAI Press.

Trinh, V.-G.; Park, K. H.; Pastva, S.; and Rozum, J. C. 2025. Mapping the attractor landscape of Boolean networks with biobalm. *Bioinform.* 41(5):btaf280.

Veliz-Cuba, A. 2011. Reduction of Boolean network models. *J. Theor. Biol.* 289:167–172.

Wang, R.-S.; Saadatpour, A.; and Albert, R. 2012. Boolean modeling in systems biology: an overview of methodology and applications. *Phys. Biol.* 9(5):055001.

Weinstein, N.; Mendoza, L.; Gitler, I.; and Klapp, J. 2017. A network model to explore the effect of the microenvironment on endothelial cell behavior during angiogenesis. *Front. Physiol.* 8.