April 2026

On the Complexity of Initial Models in Abstract Dialectical Frameworks

Hannes STRASS [a] and Johannes P. WALLNER [b]
[a] *TUD Dresden University of Technology*
[b] *Graz University of Technology*
ORCiD ID: Hannes Strass https://orcid.org/0000-0001-6180-6452, Johannes P. Wallner https://orcid.org/0000-0002-3051-1966

Abstract. Abstract dialectical frameworks (ADFs) constitute one of the most general formal approaches within abstract argumentation. Recently, Bengel and Thimm (2025) proposed so-called initial models in ADFs as a suitable concept for "serializing" reasoning, i.e., decomposing reasoning in ADFs step by step. They showed several complexity results of reasoning with initial models. Our contributions in this paper are (i) that we close a gap in their complexity results of verifying initial models by showing DP-completeness, and (ii) based on our complexity results we provide an answer set programming (ASP) implementation to support systems utilizing initial models.

1. Introduction

With roots in diverse areas such as legal reasoning, formal logics, non-monotonic reasoning, and informal reasoning, computational argumentation [1,2] provides approaches to represent argumentative reasoning within the broader field of Artificial Intelligence (AI). Computational argumentation, also sometimes referred to as formal argumentation, finds diverse applications [3], such as in medical and legal reasoning.

Applications of formal models in argumentation usually rely on a principled workflow, or argumentation process [4], in which arguments are instantiated based on existing (semi-)formal knowledge bases. Typically, such arguments consist of premises and a conclusion, and can incorporate various other components. These arguments are linked together, with an attack relation (or counter-argument relation) being the most prominent one to be used in subsequent reasoning.

Looking at the constructed arguments and relations at an abstract level, we arrive at what is commonly referred to as abstract argumentation. A set of abstract arguments together with directed attacks between them constitute an argumentation framework (AF) [5]. Argumentation semantics then drive reasoning on these abstracted AFs, and, e.g., state which sets of arguments can be collectively accepted together, i.e., such a set provides a coherent stance to argue in favour of each of its member arguments. A common condition is that of admissibility, the requirement for a set of arguments to be internally conflict-free and to defend itself against counter-arguments.

To allow for more sophisticated relations between arguments, several generalizations of AFs have been studied [6], with abstract dialectical frameworks (ADFs) [7,8]

being among the most general. These ADFs allow for flexible specification of relations between arguments via acceptance conditions, e.g., representable as Boolean formulas. Reflecting the heterogeneity of computational argumentation, ADFs found several application directions, such as in legal reasoning [9,10,11], online dialog systems [12], text exploration [13], and instantiation of defeasible theories [14]. Recently, ADFs were also connected to reasoning in Boolean networks [15,16,17].

Reasoning in argumentation inherently features a "dialectical" flavour: arguing against or in favour of claims under scrutiny in a possibly iterative fashion. Standard (computational) reasoning tasks in ADFs (and AFs) usually focus on sets of arguments being acceptable together, or more precisely, three-valued interpretations in the case of ADFs: assigning accepted (true), rejected (false), or undecided to each argument.

Nevertheless, a more step-by-step dialectical explication or explanation can be useful for understanding the underlying reasoning process. Towards such dialectical explications, e.g., game-based discussions have been proposed for AFs [18] and ADFs [19,20].

Recently, so-called initial sets in AFs [21] and their counterpart in ADFs, initial models [22], were found to be a useful concept towards dialectical explanations. The idea is to "serialize" [23,22] reasoning, such that a given "acceptable" three-valued interpretation, e.g., an admissible interpretation, is serialized into several distinguishable steps, with each step again representing an admissible interpretation. An initial model is then a possible starting point of such a serialization process; more formally, an *initial model* is an admissible interpretation with no non-trivial smaller admissible interpretation. That is, such an initial model cannot directly be decomposed further (into admissible chunks).

Reasoning in (abstract) argumentation usually exhibits high computational complexity [24], and this is mirrored also for reasoning with initial models in ADFs. In particular, Bengel and Thimm [22] showed several complexity results for initial models. For the so-called verification task, i.e., "Given a three-valued interpretation v, is v an initial model?", they showed coNP-hardness and membership in Θ_2^P. There remains a gap: problems in coNP can be solved via a Boolean SAT(isfiability) solver, while Θ_2^P seems to suggest that several SAT calls are needed (either in parallel or a logarithmic number in an adaptive fashion).

We take up the work by Bengel and Thimm and close the complexity gap. In particular, we show that the verification problem of initial models is, in fact, DP-complete, signaling that this task can be solved with exactly *two* SAT solver calls (one addressing an "NP" part and one solving a "coNP" part). This aligns verification complexity of initial models with that of complete interpretations in ADFs, and shows that initial models are more complex (unless $\mathsf{P} = \mathsf{NP}$) than admissible interpretations [25].

Finally, to complement our main theoretical result of showing DP-completeness, we provide an implementation in answer set programming (ASP) [26] of the "NP" part of the verification problem. Together with other systems already available for the "coNP" part [27], our implementation can be used for verification of initial models.

2. Background

We recall main preliminaries in this section.

2.1. Complexity Theory

We assume familiarity with the basic complexity class NP, and with hardness under polynomial-time many-one reductions and completeness of problems for complexity classes. We refer the reader to standard textbooks on this topic [28,29].

We will also make use of the classes coNP and DP. The former is defined as the complement of NP, i.e., a problem is in NP iff the complement of this problem is in coNP. Complementarity means here that an instance is a "yes" instance of a problem iff the same instance is a "no" instance of the complementary problem.

The class DP is defined by containing all decision problems L such that there exist a problem $L_1 \in \mathsf{NP}$ and a problem $L_2 \in \mathsf{coNP}$ such that $L = L_1 \cap L_2$. Intuitively, L is the logical "AND" of a problem in NP and a problem in coNP, i.e., contains "yes" instances iff the same instance is a "yes" instance of a problem in NP and one in coNP. The canonical DP-complete problem is SAT-UNSAT, where we are given a pair of Boolean formulas (ϕ, ψ). Such a pair is a "yes" instance of SAT-UNSAT iff ϕ is satisfiable and ψ is unsatisfiable. Thus, SAT-UNSAT is the intersection of the NP problem $\{(\phi,\psi) \mid \phi \text{ is satisfiable}\}$ and the coNP problem $\{(\phi,\psi) \mid \psi \text{ is unsatisfiable}\}$.

2.2. Abstract Dialectical Frameworks

Abstract dialectical frameworks (ADFs) were proposed by Brewka and Woltran [7], and have been subsequently studied in quite some depth [8].

An ADF is a triple, with a set of (abstract) arguments, a set of links between arguments, and an acceptance condition for each argument.

Definition 1. *An ADF is a triple $D = (A, L, C)$ with A a set of arguments, $L \subseteq A \times A$, and $C = \{C_a\}_{a \in A}$ a collection of acceptance conditions $C_a \colon 2^{par_D(a)} \to \{\mathbf{t}, \mathbf{f}\}$ for each argument, with the parents of a defined as $par_D(a) = \{b \in A \mid (b,a) \in L\}$. It is customary to represent such an acceptance condition, a Boolean function C_a, as a propositional formula φ_a with the understanding that $C_a(M) = \mathbf{t}$ iff $M \models \varphi_a$, for all $M \subseteq par_D(a)$.*

Semantics of ADFs are defined on three-valued interpretations v assigning each argument true (**t**), false (**f**), or undecided (**u**). A three-valued interpretation is two-valued if no argument is assigned **u**. Truth values are ordered according to their information content by $\mathbf{u} \leq_i \mathbf{t}$ and $\mathbf{u} \leq_i \mathbf{f}$; **t** and **f** are incomparable w.r.t. $\leq_i$. Moreover, $x \leq_i x$ holds for each $x \in \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$. The ordering $\leq_i$ extends to three-valued interpretations by $v \leq_i v'$ if $v(a) \leq_i v'(a)$ for each argument a. The strict counterpart $<_i$ is defined by $\mathbf{u} <_i x$ for $x \in \{\mathbf{t}, \mathbf{f}\}$. Consequently, for interpretations $v, w \colon A \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ it is the case that $v <_i w$ if $v \leq_i w$ and there is an argument $a \in A$ with $v(a) <_i w(a)$.

As we are mainly interested in admissible(-based) semantics, we recall it next.

Definition 2. *Let $D = (A, L, C)$ be an ADF, and v a three-valued interpretation over A. For any formula φ, we denote by φ^v the formula that is obtained by replacing in φ:*

- *every occurrence of an $a \in A$ with $v(a) = \mathbf{t}$ by $\top$ (denoting truth), and*
- *every occurrence of an $a \in A$ with $v(a) = \mathbf{f}$ by $\bot$ (denoting falsity).*

An interpretation $v \colon A \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ is admissible *for D iff for all $a \in A$:*

- *if $v(a) = \mathbf{t}$, then φ_a^v is irrefutable (a tautology);*

- *if* $v(a) = \mathbf{f}$*, then* φ_a^v *is unsatisfiable (a contradiction).*

Let us recall a standard reasoning task on ADFs (such tasks are typically parameterized by one semantics). Given an ADF and considering a semantics σ (e.g., admissible semantics), the verification task asks whether a given three-valued interpretation is a σ-interpretation for the given ADF. We denote the associated decision problem by $\texttt{Ver}_\sigma$.

Let us exemplify admissible interpretations next. For conciseness, we only show arguments mapped to true or false. Links between arguments are implicitly represented by syntactical occurrence of arguments in acceptance formulas.

Example 1. *Consider a simple ADF with three arguments, a, b, and c. Their acceptance conditions are given by* $\varphi_a = b$*,* $\varphi_b = a$*, and* $\varphi_c = c$*. That is, a and b are acceptable provided that the other is accepted, while c can be accepted without needing further arguments. Three admissible interpretations here are* $v_1 = \{a \mapsto \mathbf{t}, b \mapsto \mathbf{t}, c \mapsto \mathbf{t}\}$*,* $v_2 = \{a \mapsto \mathbf{t}, b \mapsto \mathbf{t}\}$*, and* v_3 *mapping all arguments to undecided.*

We showed in previous work [25] that the verification task under admissible semantics is coNP-complete. Intuitively, to verify an interpretation as admissible for an ADF whose acceptance conditions are represented by propositional formulas, one has to verify irrefutability of the propositional formula $\bigwedge_{a \in A, v(a) \neq \mathbf{u}} (a \leftrightarrow \varphi_a)^v$.

Initial models [22] extend an associated notion of initial sets in AFs [21], and can be compactly characterized as admissible interpretations such that no strictly smaller admissible interpretation exists, except for the trivial one $v_\mathbf{u}$ (assigning all arguments $\mathbf{u}$).

Definition 3. *Let D be an ADF. An interpretation v is an* initial model *of D iff:*

1. *v is admissible for D,*
2. $v_\mathbf{u} <_i v$*, that is, v is* non-trivial*, and*
3. *there is no admissible* v' *for D with* $v_\mathbf{u} <_i v' <_i v$*.*

Example 2. *Continuing Example 1,* v_1 *is not an initial model, since* $v_2 <_i v_1$ *and* v_2 *is admissible. Intuitively, a and b can be accepted depending on each other, but "explaining" acceptance of c is independent. Conversely,* v_2 *is an initial model – while* v_3 *is admissible with* $v_3 <_i v_2$*, this is fine because* $v_3 = v_\mathbf{u}$ *is trivial.*

Bengel and Thimm [22] showed that the complexity of the verification task for initial models, denoted by $\texttt{Ver}_{\mathsf{is}}$, is coNP-hard and in Θ_2^P. The later class contains decision problems which can be solved in polynomial time with access to an NP oracle, however this oracle can only be used a logarithmic number of times. In the next section, we will improve both bounds and settle the open complexity.

3. Main Result: Complexity of Initial Models Verification

In this section we show our main result, namely the tight complexity bounds for the verification task for initial models in ADFs. We refine the previous result by Bengel and Thimm [22], who showed that verifying initial models is coNP-hard and in Θ_2^P, by showing that this verification task is DP-complete. It holds that $\mathsf{coNP} \subseteq \mathsf{DP} \subseteq \Theta_2^P$.

For clarity, we separate the completeness statement, and proof, into a statement for hardness and one for membership. We start with hardness, as this is the direction that is arguably easier to establish.

Proposition 1 (Hardness)**.** $\texttt{Ver}_{\mathsf{is}}$ *is* DP*-hard.*

Proof. We reduce from the DP-complete problem SAT-UNSAT. Let (ϕ, ψ) be an instance of SAT-UNSAT, where ϕ and ψ are propositional formulas over disjoint vocabularies P and Q, respectively. We define the following ADF $D_{\phi,\psi}$ to be given as follows:

$$
\begin{aligned}
A &:= P \cup Q \cup \{x, y\} && \text{with } x, y \notin P \cup Q \\
\varphi_p &:= p && \text{for each } p \in P \\
\varphi_q &:= q && \text{for each } q \in Q \\
\varphi_x &:= \phi \wedge y \\
\varphi_y &:= \psi \vee x
\end{aligned}
$$

We now proceed to prove that the interpretation $v := \{x \mapsto \mathbf{f}, y \mapsto \mathbf{f}\}$ is an initial model of $D_{\phi,\psi}$ if and only if ϕ is satisfiable and ψ is unsatisfiable.

"if": Let ϕ be satisfiable and ψ be unsatisfiable. Then $\varphi_x^v = \phi \wedge \bot \equiv \bot$ is unsatisfiable and $\varphi_y^v = \psi \vee \bot \equiv \psi$ is unsatisfiable by assumption. Thus v is admissible. Regarding minimality, the only candidates for strictly $\leq_i$-smaller non-trivial admissible interpretations are $v_1 = \{x \mapsto \mathbf{f}\}$ and $v_2 = \{y \mapsto \mathbf{f}\}$. We consider each in turn.

- For $v_1 = \{x \mapsto \mathbf{f}\}$, we obtain that $\varphi_x^{v_1} = \varphi_x = \phi \wedge y$ is satisfiable because ϕ is satisfiable by assumption and y does not occur in ϕ. Thus v_1 is not admissible.
- For $v_2 = \{y \mapsto \mathbf{f}\}$, we get that $\varphi_y^{v_2} = \psi \vee x$ is satisfiable. Thus v_2 is not admissible.

Therefore, no non-trivial interpretation v' with $v' <_i v$ is admissible and v is an initial model of $D_{\phi,\psi}$.

"only if": Let v be an initial model of $D_{\phi,\psi}$. Then in particular v is admissible and we get that $\varphi_y^v = \psi \vee \bot \equiv \psi$ is unsatisfiable. Moreover, $v_1 = \{x \mapsto \mathbf{f}\} <_i v$ is not admissible, thus $\varphi_x^{v_1}$ is not unsatisfiable, that is, $\varphi_x^{v_1} = \phi \wedge y$ is satisfiable, whence in particular ϕ is satisfiable. □

Example 3. *We show an illustration of the hardness construction of the preceding proof in Figure 1, for* $\phi = p_1 \vee p_2$ *and* $\psi = q_1 \wedge (\neg q_1 \vee q_2) \wedge \neg q_2$*. The former formula is satisfiable and the latter is unsatisfiable. Then* $v := \{x \mapsto \mathbf{f}, y \mapsto \mathbf{f}\}$ *is admissible in the constructed ADF. Removal of either assignment to false would lead to the acceptance condition of the other argument to be satisfiable (violating a condition of admissibility).*

Towards our membership result of $\texttt{Ver}_{\mathsf{is}}$ in DP, we first provide a non-deterministic algorithm that shows that a given three-valued interpretation has no strictly less informative non-trivial admissible interpretation.

In other words, we first show the "NP" part of $\texttt{Ver}_{\mathsf{is}}$, while the "coNP" part will be dedicated to checking admissibility of a given three-valued interpretation.

We first remark that for the subproblem of verifying that there is no non-trivial strictly less informative admissible interpretation, a "standard" guess & check approach for showing membership in NP seems not very direct. For instance, "guessing" a strictly smaller non-trivial three-valued interpretation and a subsequent check of admissibility is not adequate, since the latter check is already coNP-hard, yielding an overall procedure in Σ_2^P. Our containment result below establishes that there is a more efficient way.

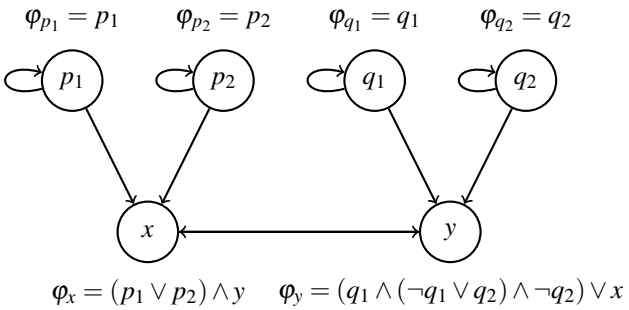

Figure 1. Constructed ADF in the proof of Proposition 1 with $\phi = p_1 \vee p_2$ and $\psi = q_1 \wedge (\neg q_1 \vee q_2) \wedge \neg q_2$.

In the algorithm, we use the notational shorthands below, where $v\colon A \to \{\mathbf{t},\mathbf{f},\mathbf{u}\}$ is an interpretation: we define

- $\mathrm{dom}_2(v) := \{a \in A \mid v(a) \in \{\mathbf{t},\mathbf{f}\}\}$ to be the set of arguments mapped to **t** or **f** in v,
- $[v]_2 := \{w\colon A \to \{\mathbf{t},\mathbf{f}\} \mid v \leq_i w\}$ to be the set of two-valued completions of v, and
- $v[a \mapsto \mathbf{u}]$ to be the interpretation that is like v only that it maps a to **u**.

Algorithm 1: no-strictly-smaller-non-trivial-admissible

Input : an ADF D over A, a three-valued interpretation $w\colon A \to \{\mathbf{t},\mathbf{f},\mathbf{u}\}$
Output: "yes" if all non-trivial $v <_i w$ are not admissible, "no" otherwise

```
1  for a ∈ dom2(w) do
2      w_a := w[a ↦ u]
3      v_1 := w_a
4      for j from 1 to |dom2(w_a)| do
5          v_{j+1} := v_j
6          for c ∈ dom2(v_j) do
7              guess some v_c ∈ [v_j]_2
8              if v_c(c) ≠ v_c(φ_c) then
9                  v_{j+1} := v_{j+1}[c ↦ u]
10         if v_{j+1} = v_j and dom2(v_{j+1}) ≠ ∅ then
11             return "no"
12 return "yes"
```

The intuition behind Algorithm 1 is to iteratively search for counterexamples that a three-valued interpretation strictly smaller than a given one is admissible (and non-trivial). In Lines 1–2 of Algorithm 1 we "reduce" the given three-valued interpretation w by re-assigning one argument to **u**. We loop over all arguments and perform subsequent checks. This is to ensure that we only search among those interpretations that are strictly smaller than the given one. In the for-loop in Lines 4–11 we non-deterministically construct a $v_c \in [v_j]_2$ (a "completion" of the current three-valued interpretation to a two-valued one) and check in Line 8, whether this completion constitutes a counter-witness

to admissibility of v_j. If it does, we reduce the three-valued interpretation further (since c cannot have the value $v_j(c) \in \{\mathbf{t}, \mathbf{f}\}$ due to the counterexample). If we iterate for each $a \in A$ in Line 1 and reduce them to the trivial interpretation, we conclude that no admissible non-trivial interpretation strictly smaller than the given one exists.

Example 4. *Let us revisit Example 3 and use the same ADF and the initial model $v = \{x \mapsto \mathbf{f}, y \mapsto \mathbf{f}\}$. Let us use Algorithm 1 to check whether there is a strictly smaller admissible interpretation w in this ADF with $w <_i v$.*

Let us start with $w_y = \{x \mapsto \mathbf{f}\}$. Then the two-valued domain of $v_1 = w_y$ consists only of x. We can guess a two-valued interpretation $v_x = \{x \mapsto \mathbf{f}, y \mapsto \mathbf{t}, p_1 \mapsto \mathbf{t}, p_2 \mapsto \mathbf{t}, q_1 \mapsto \mathbf{t}, q_2 \mapsto \mathbf{f}\}$. It holds that $v_x(x) = \mathbf{f} \neq v_x(\varphi_x) = \mathbf{t}$. We modify v_1 to v_2 by assigning x to $\mathbf{u}$. Then Line 10 of Algorithm 1 does not apply (v_2 is trivial). The case for w_x in the first for loop leads to a trivial interpretation, as well. Thus, Algorithm 1 returns "yes".

Consider a different example, with the same ADF and given $v = \{p_1 \mapsto \mathbf{t}, p_2 \mapsto \mathbf{t}\}$. This is not an initial model, but an admissible interpretation. If we apply Algorithm 1, we find that, when starting with w_{p_1}, we can find no guessed $v_{p_2} \in [w_{p_1}]_2$ (more precisely $v_{p_2} \in [v_1]_2$, however $v_1 = w_{p_1}$) such that $\mathbf{t} = v_{p_2}(p_2) \neq v_{p_2}(\varphi_{p_2}) = \mathbf{t}$, since the evaluation of p_2 only depends on itself. Thus, Algorithm 1 returns "no".

We next prove that Algorithm 1 is correct, that is, sound and complete. The main insight behind soundness of the algorithm is that the algorithm cannot "jump over" counterexamples. If there is a counterexample (a strictly smaller admissible interpretation), the algorithm must "hit it" with some v_j and then return "no". Regarding completeness, if there is no counterexample, then every strictly smaller interpretation is not admissible; for this, in turn, there exist witnessing completions that the algorithm can guess.

Proposition 2. *Algorithm 1 answers "yes" on input (D, w) if and only if all non-trivial $v <_i w$ are not admissible for D.*

Proof. We show the two directions individually.

Soundness: Assume that there exists a non-trivial $w' <_i w$ that is admissible for D. We have to show that the algorithm will return "no". To this end, we first prove the following loop invariant about the for-loop spanning Lines 4–11:

> For all $j \geq 1$, if $w' \leq_i v_j$ then $w' \leq_i v_{j+1}$.

Consider $j \geq 1$ and assume $w' \leq_i v_j$. Let $c \in \mathrm{dom}_2(w')$. Then $w'(c) = v_j(c)$, and by $w' \leq_i v_j$ it holds that $[v_j]_2 \subseteq [w']_2$. Now for any $v_c \in [v_j]_2$ that the algorithm could potentially guess in Line 7, we obtain $v_j(c) = w'(c) = v_c(c) = v_c(\varphi_c)$ since $v_c \in [w']_2$ and w' is admissible. Therefore, $v_{j+1}(c) = v_j(c)$. Since c was chosen arbitrarily, we obtain $w' \leq_i v_{j+1}$. This establishes the loop invariant.
Next, it is easy to see that if $w' <_i w$ then there exists an $a \in \mathrm{dom}_2(w)$ such that $w' \leq_i w_a$. Thus for this $v_1 := w_a$ we can employ that w' is non-trivial and obtain that $1 \leq |\mathrm{dom}_2(w')| \leq |\mathrm{dom}_2(v_1)|$. Additionally, it is clear by design that for all $j \geq 1$, after the for-loop in Line 9, either $v_{j+1} <_i v_j$ or $v_{j+1} = v_j$. Therefore, by the loop invariant and a pigeonhole argument there exists a $j \leq |\mathrm{dom}_2(v_1)|$ such that (after Line 9) $v_{j+1} = v_j$. Furthermore, v_j is non-trivial as $w' \leq_i v_j$ and w' is non-trivial. Thus $\mathrm{dom}_2(v_{j+1}) = \mathrm{dom}_2(v_j) \neq \emptyset$ and the algorithm returns "no".

Completeness: Assume that every non-trivial $v <_i w$ is not admissible. We have to show that the algorithm will answer "yes". To this end, we show that for any $a \in A$ and $1 \leq j \leq |\mathrm{dom}_2(w_a)|$, the algorithm will not return "no" in Line 11.
Consider $a \in \mathrm{dom}_2(w)$ and $1 \leq j \leq |\mathrm{dom}_2(w_a)|$ arbitrary. If v_j is trivial, the test in Line 10 will fail (hence the algorithm will not return "no"), so let v_j be non-trivial. Then, by assumption, as $v_j <_i w$, we find that v_j is not admissible. Thus there exists a $c \in \mathrm{dom}_2(v_j)$ such that for some $v_c \in [v_j]_2$, we find $v_c(c) \neq v_c(\varphi_c)$. Hence, there is a run of the algorithm where this v_c is guessed in Line 7, and in this run we obtain $v_{j+1} <_i v_j$ whence the test "$v_{j+1} = v_j$" on Line 10 will fail and the algorithm will not return "no" for v_{j+1} at this point in the run.
Since a and j have been chosen arbitrarily and the guesses are all independent of each other, the algorithm will not return "no" for any $a \in \mathrm{dom}_2(w)$ and $j \geq 0$, whence the algorithm will return "yes" on Line 12. □

Next we show that Algorithm 1 is, in fact, an algorithm which we can utilize to show membership in NP for the problem of checking whether a non-trivial strictly smaller three-valued interpretation exists, given a three-valued interpretation. That is, Algorithm 1 must run in non-deterministic polynomial time. Non-determinism can be seen directly by Line 7 (a non-deterministic "guess").

Proposition 3. *Algorithm 1 runs in non-deterministic polynomial time.*

Proof. The algorithm consists of three nested for-loops, with the number of iterations in each loop being bounded by the number of arguments in D (actually in $\mathrm{dom}_2(w)$), thus linear in the length of the input. Each guess only produces an interpretation that is also of size at most linear in the input. □

Finally, we utilize Algorithm 1, together with correctness statements and runtime results, to show membership of $\mathtt{Ver}_{\mathsf{is}}$ in DP.

Proposition 4 (Containment)**.** $\mathtt{Ver}_{\mathsf{is}}$ *is in* DP.

Proof. Given $w\colon A \to \{\mathbf{t},\mathbf{f},\mathbf{u}\}$, we can verify that w is admissible in coNP and verify that w is non-trivial $\leq_i$-minimal with respect to being admissible in NP according to Proposition 2 and Proposition 3. □

Taking the preceding results together, we arrive at the main result.

Theorem 1. $\mathtt{Ver}_{\mathsf{is}}$ *is* DP*-complete.*

Proof. Containment is shown in Proposition 4, hardness in Proposition 1. □

4. Answer Set Programming Implementation

In this section we provide an implementation of Algorithm 1 in answer set programming (ASP) [26,30,31].

We briefly recall basics of ASP. A normal ASP program π consists of rules r of the form $b_0 \leftarrow b_1,\ldots,b_k,not\,b_{k+1},\ldots,not\,b_m$, where each b_i is an atom. A rule without head b_0 is a constraint and a shorthand for $a \leftarrow b_1,\ldots,b_k,not\,b_{k+1},\ldots,\ not\,b_m,not\,a$ for

a fresh a. An atom b_i is of the form $p(t_1,\ldots,t_n)$ with p an n-ary predicate symbol and each t_j either a constant or a variable. An answer set program is ground if it is free of variables. For a non-ground program, GP is the set of rules obtained by applying all possible substitutions from the variables to the set of constants appearing in the program. An interpretation I, that is, a subset of all the ground atoms, satisfies a positive rule $r = h \leftarrow b_1,\ldots,b_k$ if and only if all positive body atoms $b_1,\ldots,b_k$ being in I implies that the head atom is in I. For a program π consisting only of positive rules, let $Cl(\pi)$ be the uniquely determined interpretation I that satisfies all rules in π and no subset of I satisfies all rules in π. Interpretation I is an answer set of a ground program π if $I = Cl(\pi^I)$ where $\pi^I = \{h \leftarrow b_1,\ldots,b_k \mid (h \leftarrow b_1,\ldots,b_k, not\, b_{k+1},\ldots,not\, b_m) \in \pi, \{b_{k+1},\ldots,b_m\} \cap I = \emptyset\}$ is the reduct of π w.r.t. I. Similarly, I is an answer set of a non-ground program π if I is an answer set of GP of π.

We first encode a given ADF $D = (A,L,C)$ and a given three-valued interpretation v over A as a set of facts (atoms), as follows. We assume that a given acceptance condition $\varphi_a = c_1 \wedge \ldots \wedge c_n$, for $a \in A$ is represented as a propositional formula in conjunctive normal form (CNF). We give all clauses c occurring in acceptance conditions a unique identifier, represented by $id(c)$. The facts (rules with empty bodies) encoding the ADF D are now:

$\mathbf{arg}(a).$	for $a \in A$
$\mathbf{ac}(a,i).$	for $i = id(c)$ and $c \in \varphi_a$
$\mathbf{clausePos}(i,x).$	for $i = id(c)$ and $x \in c$
$\mathbf{clauseNeg}(i,x).$	for $i = id(c)$ and $\neg x \in c$
$\mathbf{givenInt}(a,y).$	for $v(a) = y$

Example 5. *We show the ASP encoding of the ADF from Example 3 in Listing 1.*

Listing 1: ASP Encoding of the ADF from Example 3.

```
1  % arguments
2  arg(p1). arg(p2). arg(q1). arg(q2). arg(x). arg(y).
3  % acceptance conditions in CNF
4  ac(p1,1). clausePos(1,p1). ac(p2,2). clausePos(2,p2).
5  ac(q1,3). clausePos(3,q1). ac(q2,4). clausePos(4,q2).
6  ac(x,5). clausePos(5,p1). clausePos(5,p2). ac(x,6). clausePos(6,y).
7  ac(y,7). clausePos(7,q1). clausePos(7,x).
8  ac(y,8). clauseNeg(8,q1). clausePos(8,q2). clausePos(8,x).
9  ac(y,9). clauseNeg(9,q2). clausePos(9,x).
10 % given three-valued interpretation
11 givenInt(p1,u). givenInt(p2,u).
12 givenInt(q1,u). givenInt(q2,u).
13 givenInt(x,f). givenInt(y,f).
```

We present our ASP encoding of Algorithm 1 in Listing 2. This ASP encoding is satisfiable, if supplied with an instance as in Listing 1 that permits no strictly smaller non-trivial admissible interpretation. This encoding follows Algorithm 1.

Lines 2–3 initialize the number of arguments in the given ADF and start "iterations", represented by a binary predicate **iter**(IA, I), with IA being one of the arguments assigned true or false in the given three-valued interpretation (simulating Line 1 of Algorithm 1) and an iteration ID starting with 1 (and ending in the number of arguments).

Next, in Lines 5–6, we construct v_1 for each "initial argument" (IA) where we assign the same value to the arguments, except the chosen argument is assigned undecided.

Lines 8–10 perform the non-deterministic guess of a two-valued extension. Subsequently, in Lines 12–15, we check satisfaction of clauses and acceptance conditions. After a check for trivial interpretations, we calculate which of the assigned values of arguments need to change (if the evaluation of their acceptance condition is different than the value assigned by the current interpretation, for arguments not assigned **u**).

An iteration is here "successful" (Lines 23–24) if the three-valued interpretation is trivial or we change a value. Finally, we update the interpretation to the "next" iteration (Lines 28–30), and verify that each iteration, for each choice of initially unassigned argument, leads to a success (i.e., to a trivial interpretation).

Finally, note that our ASP encoding does not make use of advanced ASP language features, i.e., is (almost) a normal ASP program, except for an aggregate statement (Line 2), which could also be provided as input, and some simple arithmetic comparisons and calculations.

Listing 2: ASP Encoding of Algorithm 1

```
1  %%% initialization and number of arguments
2  numArgs(N) ← N=#count{ A: arg(A) }.
3  iter(IA,1) ← givenInt(IA,V), V!=u.
4  %%% Construct v1 for each argument assigned true or false initially
5  int(IA,1,A,V) ← givenInt(A,V), givenInt(IA,VIA), VIA!=u, A!=IA.
6  int(IA,1,IA,u) ← givenInt(IA,V), V!=u.
7  %%% Guess a two−valued extension for each iteration
8  guessExt(IA,I,A,V) ← int(IA,I,A,V), V!=u.
9  guessExt(IA,I,A,t) ← int(IA,I,A,V), V=u, not guessExt(IA,I,A,f).
10 guessExt(IA,I,A,f) ← int(IA,I,A,V), V=u, not guessExt(IA,I,A,t).
11 %%% Clause satisfaction
12 satClause(IA,I,C) ← clausePos(C,A), guessExt(IA,I,A,t).
13 satClause(IA,I,C) ← clauseNeg(C,A), guessExt(IA,I,A,f).
14 unsatAC(IA,I,A) ← iter(IA,I), ac(A,C), not satClause(IA,I,C).
15 satAC(IA,I,A) ← iter(IA,I), arg(A), not unsatAC(IA,I,A).
16 %%% Check for trivial interpretations
17 nonTrivial(IA,I) ← int(IA,I,A,V), V!=u.
18 ← not nonTrivial(_,1).
19 %%% Does an acceptance condition evaluate differently under an extension?
20 changeVal(IA,I,A) ← satAC(IA,I,A), int(IA,I,A,f).
21 changeVal(IA,I,A) ← unsatAC(IA,I,A), int(IA,I,A,t).
22 %%% Iteration is successful if we have a trivial interpretation or change a value
```

23 **success**(IA,I) ← **iter**(IA,I), *not* **nonTrivial**(IA,I).
24 **success**(IA,I) ← **changeVal**(IA,I,A).
25 *%%% Next iterations*
26 **iter**(IA,J) ← **success**(IA,I), I < N, **numArgs**(N), J=I+1.
27 *%%% Update interpretations from previous iterations*
28 **int**(IA,J,A,V) ← **int**(IA,I,A,V), **iter**(IA,J), J=I+1, V!=u, *not* **changeVal**(IA,I,A).
29 **int**(IA,J,A,V) ← **int**(IA,I,A,V), **iter**(IA,J), J=I+1, V=u.
30 **int**(IA,J,A,u) ← **int**(IA,I,A,V), **iter**(IA,J), J=I+1, V!=u, **changeVal**(IA,I,A).
31 *%%% Last iteration must be successful*
32 ← **numArgs**(N), **givenInt**(IA,V), V!=u, *not* **success**(IA,N).

It follows that this encoding also provides the first implementation of initial models for ADFs. Given that ASP encodings for verifying admissibility already exist [32], using two independent solver calls in accordance with Theorem 1 will correctly verify whether a given interpretation is an initial model.

5. Conclusion

In this work we closed the complexity bounds left open in previous work by Bengel and Thimm [22] for the verification task for initial models in ADFs. We showed that this task is DP-complete. Our underlying proof resulted in a (non-deterministic) algorithm, which we implemented in ASP for checking whether a non-trivial admissible interpretation exists that is strictly smaller than a given one.

For future work, we think it is worthwhile to extend our ASP-based implementation to general reasoning with initial models and serializability in ADFs, and AFs. This could be possible by devising a saturation-based encoding in disjunctive ASP [33] or employing the guess-and-check methodology of Eiter and Polleres [34].

References

[1] Baroni P, Gabbay D, Giacomin M, van der Torre L, editors. Handbook of Formal Argumentation. College Publications; 2018.
[2] Gabbay D, Giacomin M, Simari GR, Thimm M, editors. Handbook of Formal Argumentation. vol. 2. College Publications; 2021.
[3] Atkinson K, Baroni P, Giacomin M, Hunter A, Prakken H, Reed C, et al. Towards Artificial Argumentation. AI Magazine. 2017;38(3):25-36.
[4] Caminada M, Amgoud L. On the evaluation of argumentation formalisms. Artif Intell. 2007;171(5-6):286-310.
[5] Dung PM. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. Artif Intell. 1995;77(2):321-58.
[6] Brewka G, Polberg S, Woltran S. Generalizations of Dung Frameworks and Their Role in Formal Argumentation. IEEE Intell Syst. 2014;29(1):30-8.
[7] Brewka G, Woltran S. Abstract Dialectical Frameworks. In: Lin F, Sattler U, Truszczynski M, editors. Proc. KR. AAAI Press; 2010. p. 102-11.
[8] Brewka G, Ellmauthaler S, Strass H, Wallner JP, Woltran S. Abstract Dialectical Frameworks. An Overview. FLAP. 2017;4(8). Available from: `http://www.collegepublications.co.uk/downloads/ifcolog00017.pdf`.
[9] Al-Abdulkarim L, Atkinson K, Bench-Capon TJM. A methodology for designing systems to reason with legal cases using Abstract Dialectical Frameworks. Artif Intell Law. 2016;24(1):1-49.

[10] Al-Abdulkarim L, Atkinson K, Bench-Capon TJM, Whittle S, Williams R, Wolfenden C. Noise induced hearing loss: Building an application using the ANGELIC methodology. Argument & Computation. 2019;10(1):5-22.

[11] Mumford J, Atkinson K, Bench-Capon TJM. Reasoning with Legal Cases: A Hybrid ADF-ML Approach. In: Francesconi E, Borges G, Sorge C, editors. Proc. JURIX. vol. 362 of Frontiers in Artificial Intelligence and Applications. IOS Press; 2022. p. 93-102.

[12] Neugebauer D. Generating Defeasible Knowledge Bases from Real-World Argumentations using D-BAS. In: Proc. AI^3@AI*IA. vol. 2012 of CEUR Workshop Proc. CEUR-WS.org; 2017. p. 105-10.

[13] Cabrio E, Villata S. Abstract Dialectical Frameworks for Text Exploration. In: van den Herik HJ, Filipe J, editors. Proc. ICAART. SciTePress; 2016. p. 85-95.

[14] Strass H. Implementing Instantiation of Knowledge Bases in Argumentation Frameworks. In: Parsons S, Oren N, Reed C, Cerutti F, editors. Proc. COMMA. vol. 266 of FAIA. IOS Press; 2014. p. 475-6.

[15] Dimopoulos Y, Dvořák W, König M. Connecting Abstract Argumentation and Boolean Networks. In: Reed C, Thimm M, Rienstra T, editors. Proc. COMMA. Frontiers in Artificial Intelligence and Applications. IOS Press; 2024. p. 85-96.

[16] Heyninck J, Knorr M, Leite J. Abstract Dialectical Frameworks are Boolean Networks. In: Dodaro C, Gupta G, Martinez MV, editors. Proc. LPNMR. LNCS. Springer; 2024. p. 98-111.

[17] Azpeitia E, Gutiérrez SM, Rosenblueth DA, Zapata O. Bridging abstract dialectical argumentation and Boolean gene regulation. CoRR. 2024;abs/2407.06106.

[18] Caminada M. A Discussion Game for Grounded Semantics. In: Black E, Modgil S, Oren N, editors. Proc. TAFA. Lecture Notes in Computer Science. Springer; 2015. p. 59-73.

[19] Keshavarzi Zafarghandi A, Verbrugge R, Verheij B. A Discussion Game for the Grounded Semantics of Abstract Dialectical Frameworks. In: Prakken H, Bistarelli S, Santini F, Taticchi C, editors. Proc. COMMA. Frontiers in Artificial Intelligence and Applications. IOS Press; 2020. p. 431-42.

[20] Keshavarzi Zafarghandi A, Verbrugge R, Verheij B. Strong admissibility for abstract dialectical frameworks. Argument Comput. 2022;13(3):249-89.

[21] Xu Y, Cayrol C. Initial sets in abstract argumentation frameworks. J Appl Non Class Logics. 2018;28(2-3):260-79.

[22] Bengel L, Thimm M. Initial Models and Serialisability in Abstract Dialectical Frameworks. In: Proc. IJCAI. ijcai.org; 2025. p. 4365-73.

[23] Thimm M. Revisiting initial sets in abstract argumentation. Argument Comput. 2022;13(3):325-60.

[24] Dvořák W, Dunne PE. Computational Problems in Formal Argumentation and their Complexity. In: Baroni P, Gabbay D, Giacomin M, van der Torre L, editors. Handbook of Formal Argumentation. College Publications; 2018. p. 631-88.

[25] Strass H, Wallner JP. Analyzing the computational complexity of abstract dialectical frameworks via approximation fixpoint theory. Artif Intell. 2015;226:34-74.

[26] Gelfond M, Lifschitz V. The Stable Model Semantics for Logic Programming. In: Proc. ICLP/SLP. MIT Press; 1988. p. 1070-80.

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

[28] Papadimitriou CH. Computational complexity. Academic Internet Publ.; 2007.

[29] Arora S, Barak B. Computational Complexity - A Modern Approach. Cambridge University Press; 2009.

[30] Niemelä I. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. Ann Math Artif Intell. 1999;25(3-4):241-73.

[31] Brewka G, Eiter T, Truszczynski M. Answer set programming at a glance. Commun ACM. 2011;54(12):92-103.

[32] Ellmauthaler S, Strass H. The DIAMOND System for Computing with Abstract Dialectical Frameworks. In: Parsons S, Oren N, Reed C, Cerutti F, editors. Computational Models of Argument - Proceedings of COMMA 2014, Atholl Palace Hotel, Scottish Highlands, UK, September 9-12, 2014. Frontiers in Artificial Intelligence and Applications. IOS Press; 2014. p. 233-40.

[33] Eiter T, Gottlob G. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. Ann Math Artif Intell. 1995;15(3-4):289-323.

[34] Eiter T, Polleres A. Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. Theory Pract Log Program. 2006;6(1-2):23-60.