A Semantics-Aware Evaluation Order for Abstract Argumentation Frameworks

Stefano Bistarelli[1], Carlo Taticchi[1,*]

[1]*Department of Mathematics and Computer Science - University of Perugia, Via Vanvitelli, 1 - 06123 Perugia, Italy*

Abstract

In Computational Argumentation, understanding how an Abstract Argumentation Framework (AF) is instantiated is crucial for capturing and possibly exploiting structural and dynamic aspects of the argumentative process. To facilitate the reconstruction of this process and, precisely, to determine the order in which arguments are presented, we introduce the notion of semantics-aware evaluation order. This approach relies on two fundamental concepts: syntactic dependence, derived from attack relations within the AF, and semantic dependence, which assesses the impact of an argument on the acceptability of others. Integrating these dependencies provides a more meaningful sequence for presenting arguments, improving its alignment with real-life scenarios and enriching the exploration of debate dynamics.

Keywords

Computational Argumentation, Abstract Argumentation Frameworks, Dependency Graphs, Concurrency

1. Introduction

Artificial Intelligence (AI) applications often produce results that may not seem transparent or reliable, reducing users' confidence in them. This is particularly crucial in sensitive areas like healthcare and finance, where understanding the rationale behind AI-generated outcomes is vital. Argumentation Theory [1] explores how to manage and reason with conflicting information, and Argumentation Frameworks have, in the last decades, become a consolidated tool to examine and replicate human reasoning for logically deriving conclusions starting from a set of premises. In general, non-monotonic reasoning offers the key advantages of flexibility and adaptability, mirroring real-world scenarios more closely by allowing conclusions to be revised or retracted in the face of new evidence. This approach is crucial for dealing with incomplete or evolving information, offering a practical way to instantiate and study complex, dynamic processes. In healthcare, argumentation-based systems have been proposed to support clinical decision-making [2, 3, 4]. For instance, they can be used to model the reasoning behind different diagnostic or treatment options, allowing medical professionals to compare the underlying logic of various choices and explaining these choices to patients, thereby enhancing trust in medical AI systems. In cybersecurity, Argumentation Theory finds application in risk assessment and management [5, 6, 7]. By simulating the logic underpinning security protocols and analysing potential vulnerabilities, the proposed systems increase the clarity of decision-making processes, identifying relevant risks and corresponding mitigations.

To simplify complex real-world reasoning processes and predispose them to computational and automated problem-solving approaches, one can abstract from the internal logic of arguments and focus solely on the interaction between them. The systems resulting from this abstraction are called Abstract Argumentation Frameworks (AFs) [8], visualised as directed graphs with nodes and edges representing arguments and their conflicts, respectively. Solving an argumentation problem formulated through an AF boils down to assessing the acceptability of its arguments through so-called argumentation semantics, i.e., criteria for selecting sets of arguments, known as *extensions*, that exhibit internal

CILC 2024: 39th Italian Conference on Computational Logic, June 26-28, 2024, Rome, Italy
*Corresponding author.
stefano.bistarelli@unipg.it (S. Bistarelli); carlo.taticchi@unipg.it (C. Taticchi)
0000-0001-7411-9678 (S. Bistarelli); 0000-0003-1260-4672 (C. Taticchi)

consistency and collectively withstand external challenges. These extensions encapsulate coherent positions or viewpoints within the framework. Consider, for example, the following three arguments:

a: "Everyone should eat more vegetables to reduce the risk of chronic diseases."
b: "Excess consumption of certain vegetables can lead to nutrient overdoses."
c: "Vegetable-induced nutrient overdoses are rare and manageable with a varied diet."

It is easy to see the conflict between them, i.e., a is attacked by b, which is, in turn, attacked by c. By abstracting from the sense of the individual arguments and considering only the relation between them, we can establish which arguments agree and can, therefore, be accepted together. In this example, a and c share the same point of view; in fact, c counters the attack of b against a. Therefore a and c together form an extension (with respect to the *complete* semantics [9]) for the examined AF.

While AFs offer clear benefits for the representation and resolution of argumentative problems, they also have drawbacks related to their inherent simplicity, which makes them ineffective in capturing certain fundamental aspects of human reasoning [10]. One particular disadvantage is that an AF provides only a static representation of a given argumentative process, thus not allowing an understanding of the dynamics that resulted in the establishment of such a framework. For instance, the argument c introduced earlier (nutrient overdoses are rare) is likely presented in response to argument b (vegetables can lead to nutrient overdoses), yet the abstraction inherent in AFs makes deducing the dependency between arguments non-trivial. Various studies [11, 12, 13, 14, 15] have explored how dynamic aspects can be incorporated into AFs via enhanced frameworks and systems that allow for the integration of new information or changes. Despite these advancements, the literature does not address the problem of tracing the process that leads to the generation of a specific AF.

To bridge this gap, we proposed in [16] to interpret AFs as dependency graphs in which the attacking arguments depend on the attacked ones. This approach aids in understanding the mechanism behind the generation and evolution of an AF, determining a *feasible evaluation order* in which arguments are introduced. In particular, we can establish a meaningful sequence for presenting the arguments without knowing their meaning or internal logic. In correspondence with cycles, following a feasible evaluation order produces some sequences of arguments that do not represent realistic scenarios. In fact, the choice of arguments within cycles to be evaluated first is nondeterministic. In this paper, we revisit the process of evaluating arguments within cycles by introducing semantic dependency as a key distinguishing factor. This concept is based on the notion that certain arguments conduct attacks that are crucial for determining the acceptability state of the arguments they target, whereas others may not have such a significant impact. In order to demonstrate the effectiveness of the new *semantics-aware evaluation order* that we introduced, we analyse two examples of instantiation of AFs modelling real situations. These examples show that our latest approach discards some meaningless sequences of arguments. We also provide an automated procedure for instantiating AFs following a semantics-aware evaluation order for the arguments.

With this work, therefore, we aim to address a limitation of AFs, namely the impossibility of representing the temporal evolution of a dialogue in which arguments come into play following precise dynamics dictated by the type of interaction. The static representation intrinsic to AFs limits the applicability of abstract argumentation in domains where the sequence and interdependence of arguments are crucial. By introducing a semantics-aware order of evaluation that integrates syntactic and semantic dependencies, we want to align AFs more closely with real-life argumentative processes and enhance the expressive power they can assume within critical AI-based applications.

The paper is organised as follows: Section 2 introduces basic notions of Computational Argumentation, along with the feasible evaluation order and an overview of CLA. Section 3 illustrated the concept of semantic dependency and the resulting semantics-aware evaluation order. Section 4 presents practical examples applying this order in real-world scenarios. Section 5 outlines a methodology for instantiating any AF following the semantics-aware evaluation order. The paper concludes in Section 6 with a discussion of the results and future research directions.

2. Background

This section briefly reviews the notion of AFs, argumentation semantics, and feasible evaluation order for arguments within an AF. Additionally, we cover aspects of the CLA syntax and its operational semantics, which are instrumental in implementing our proposed methodology.

2.1. Computational Argumentation

Argumentation Theory is focused on understanding and mimicking the natural way humans reason, particularly in handling uncertainties through non-monotonic (defeasible) reasoning. In his influential paper [8], Dung sets out the fundamental components of abstract argumentation.

Definition 1 (AFs). *An Abstract Argumentation Framework is a pair $\langle Arg, R\rangle$ where Arg is a finite set of arguments and R is a binary relation on Arg.*

For any two arguments $a, b \in Arg$, the notation $(a, b) \in R$ indicates an attack from argument a against argument b. Furthermore, we use a^+ and a^- to denote the sets of arguments that, respectively, are attacked by and attack a. Within an AF, our goal is to identify subsets of *acceptable* arguments, which are determined by applying specific criteria known as argumentation semantics. Arguments not deemed acceptable are consequently rejected. Various semantics have been devised to encapsulate desirable qualities for sets of arguments. Among the most extensively studied semantics are complete, stable, semi-stable, preferred, and grounded semantics [8, 9]. To practically identify acceptable arguments, one can use labelling-based semantics [17], which assign labels to arguments to assert their acceptability state.

Definition 2 (Labelling). *A labelling of an AF $F = \langle Arg, R\rangle$ is a function $L^F : Arg \rightarrow \{\mathsf{IN}, \mathsf{OUT}, \mathsf{UND}\}$. We will omit the referenced AF F when evident from the context. Moreover, we have that*

- *L is a complete labelling for F when, $\forall a \in Arg$*
  - $L(a) = \mathsf{IN} \iff \forall b \in Arg \mid (b, a) \in R. L(b) = \mathsf{OUT}$
  - $L(a) = \mathsf{OUT} \iff \exists b \in Arg \mid (b, a) \in R \land L(b) = \mathsf{IN}$
- *L is a grounded labelling for F when*
  - *L is a complete labelling, and*
  - *$\{a \in Arg \mid L(a) = \mathsf{IN}\}$ is minimal among all the complete labellings*

We will write L_σ to identify a labelling of an AF with respect to the semantics σ. Under the complete semantics, an argument receives the label IN if and only if every argument that attacks it is labelled OUT. Conversely, it is labelled OUT if it is attacked by at least one IN argument. In situations that do not meet these conditions, the argument is labelled UND. Specifically, arguments labelled IN are considered acceptable, whereas arguments with other labels are rejected. This approach to labelling is extended in [17] to define other argumentation semantics besides the complete and the grounded ones.

Besides computing the possible labellings with respect to a certain semantics σ, one of the most common tasks performed on AFs is to decide whether an argument a is accepted (labelled as IN) in some labelling or in all labellings. In the former case, we say that a is *credulously* accepted with respect to σ; in the latter, a is instead *sceptically* accepted with respect to σ.

2.2. Feasible Evaluation Order

In AFs, attacks among arguments establish a dependency relation between them. In particular, an argument depends on the other argument it attacks. Therefore, an AF can be viewed as a dependency graph $D = \langle S, T\rangle$, where S is a set of elements and T is the transitive reduction of a dependency relation $R \subseteq S \times S$. Such a graph allows for identifying a correct evaluation order that respects the dependencies, ensuring that if an argument a precedes argument b in the order, a does not depend on b.

Finding a correct evaluation order for the arguments of an AF can be thought of as mapping the process that led to the instantiation of the AF. This task is made challenging by possible cycles in the AF, which create circular dependencies where arguments depend on each other. We use the following notation to represent the set of arguments that form a cycle with a given argument a, encapsulating all potential circular dependencies for a.

Notation 1 (Circular Dependency). *Given an AF $F = \langle Arg, R\rangle$, we call cycle any subset of Arg whose elements form a cycle in F. Then, we denote with $Cycles(F)$ the set of all cycles in F, and with $ArC(F) = \bigcup_{C \in Cycles(F)} C$ the set of all arguments belonging to cycles in F. Moreover, $CiD(a) = \{b \in Arg \mid \exists C \in Cycles(F)$ such that $a, b \in C\}$ is the set of arguments in circular dependency with a.*

In the presence of circular dependencies, it is unclear which argument should come first in the order of evaluation. To overcome this issue, we consider each cycle as an agglomeration of arguments, focusing on their connections to adjacent arguments outside the cycle. The evaluation sequence is structured for arguments outside these cycles so that attacked arguments precede their attackers. Conversely, any ordering represents a viable solution for arguments within a cycle. This approach enables the formulation of a *feasible evaluation order* [16], where dependency among arguments within a cycle is not influential, and their ordering relative to arguments outside the cycle is defined.

Definition 3 (Feasible Evaluation Order). *A feasible evaluation order for an AF $F = \langle Arg, R\rangle$ is a numbering $n : Arg \to \mathbb{N}$ such that for any two arguments $a, b \in Arg$ with $n(a) < n(b)$, the following holds: if $a \notin CiD(b)$ then $\forall c \in \{a\} \cup CiD(a), d \in \{b\} \cup CiD(b).(c, d) \notin R$.*

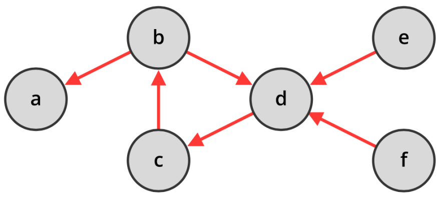

Figure 1: Example of an AF containing a cycle.

Example 1. *Referring to the AF shown in Figure 1, we can verify that (a, b, c, d, e, f) and (a, c, d, b, f, e) are two feasible evaluation orders. The first argument to be introduced is a, which is succeeded by b, c, d in any order, with e and f always following them. In particular, we observe that any argument between b, c, d can be nondeterministically chosen as the first argument of the cycle to be introduced, while e and f, which depend on the same argument, can be introduced concurrently in the AF immediately after d and the arguments of its circular dependencies have been added.*

Therefore, in order to model the instantiation process leading to the generation of this AF, we need a tool capable of manipulating AFs and supporting nondeterminism and parallel operations. To this end, we have chosen to use the Concurrent Language for Argumentation, described below, which, in addition to the features listed above, is already set up for the study of argument acceptability, a feature we will use with the aim of exploiting semantics to establish a refined order for evaluating the arguments.

2.3. Concurrent Language for Argumentation

The Concurrent Language for Argumentation (CLA) [18] is a tool devised for simulating concurrent interactions among agents engaged in reasoning and decision-making via argumentation processes. Agents utilise CLA constructs to access and manipulate a shared knowledge base encapsulated within an AF. In the following, we provide a shortened version of the CLA syntax and operational semantics, which we will use in the next section, referring to [18] for a thorough discussion of the language.

In Table 1, P denotes a generic process, C a sequence of clauses, A an agent and E a guarded agent. In a CLA process $P = C.A$, A is the initial agent to be executed within the context of the declarations C. The operational model of P is formally described by a transition system $T = (Conf, \rightarrow)$, where $Conf$ consists of a process and an AF $F = \langle Arg, R \rangle$ representing the shared knowledge base.

Table 1
Snippet of CLA syntax

$$\begin{aligned}
P &::= C.A \\
C &::= p(x) :: A \mid C.C \\
A &::= \mathit{success} \mid \mathit{failure} \mid \mathit{add}(\mathit{Arg}, R) \rightarrow A \mid E \mid A \| A \\
E &::= \mathit{check}^{w}(\mathit{Arg}, R) \rightarrow A \mid E + E
\end{aligned}$$

In Table 2, we give the definitions for the transition rules of addition (Add), check with waiting (Chw), parallelism (Par) and nondeterminism (Ndt) operators. The transition relation $\rightarrow \subseteq \mathit{Conf} \times \mathit{Conf}$ is the least relation satisfying those rules, and it characterises the system's evolution. In particular, $\langle A, F \rangle \rightarrow \langle A', F' \rangle$ represents a transition from a state in which we have the process $P = C.A$ and the AF F to a state in which we have the process $P = C.A'$ and the AF F'. An $\mathit{add}(\mathit{Arg}', R')$ results in the addition of a set of arguments Arg' and a set of attacks R' into the shared knowledge base. The operation $\mathit{check}^{w}(\mathit{Arg}', R')$ is used to verify whether the specified arguments and attacks are contained in the knowledge base, without introducing any further change. If the check is positive, the operation succeeds; otherwise, it suspends. The parallel operator $\|$ enables the specification of concurrent agents following the interleaving approach. This means that only one action is executed at a time in accordance with a scheduling imposed by the processor. The outcome of $A_1 \| A_2$ depends on the execution of A_1 and A_2: the parallel composition succeeds only if both succeed. Finally, any agent composed through nondeterminism (operator $+$) is chosen if its guards succeed. In detail, a guarded agent E_1 transits to agent A_1 whenever it can do so (first rule for (Ndt)); otherwise, both guarded agents are sent one step forward (second rule for (Ndt)). Indeed, a guarded agent can be followed by more guarded agents, all of whom must be satisfied for the operation to succeed. Until E_1 transits to A_1 (or E_2 to E_2), both guarded agents are run simultaneously to ensure true concurrency during execution.

Table 2
CLA operational semantics: add, check and parallel operators

$$\begin{array}{c}\langle \mathit{add}(\mathit{Arg}', R') \rightarrow A, \langle \mathit{Arg}, R \rangle \rangle \longrightarrow \langle A, \langle \mathit{Arg} \cup \mathit{Arg}', R \cup R'' \rangle \rangle \\ \text{with } R'' = \{(a, b) \in R' \mid a, b \in \mathit{Arg} \cup \mathit{Arg}'\}\end{array} \qquad \text{Add}$$

$$\frac{\mathit{Arg}' \subseteq \mathit{Arg} \wedge R' \subseteq R}{\langle \mathit{check}^{w}(\mathit{Arg}', R') \rightarrow A, \langle \mathit{Arg}, R \rangle \rangle \longrightarrow \langle A, \langle \mathit{Arg}, R \rangle \rangle} \qquad \text{Chw}$$

$$\frac{\langle A_1, F \rangle \longrightarrow \langle A_1', F' \rangle}{\begin{array}{c}\langle A_1 \| A_2, F \rangle \longrightarrow \langle A_1' \| A_2, F' \rangle \\ \langle A_2 \| A_1, F \rangle \longrightarrow \langle A_2 \| A_1', F' \rangle\end{array}} \qquad \frac{\langle A_1, F \rangle \longrightarrow \langle \mathit{success}, F' \rangle}{\begin{array}{c}\langle A_1 \| A_2, F \rangle \longrightarrow \langle A_2, F' \rangle \\ \langle A_2 \| A_1, F \rangle \longrightarrow \langle A_2, F' \rangle\end{array}} \qquad \text{Par}$$

$$\frac{\langle E_1, F \rangle \longrightarrow \langle A_1, F \rangle}{\begin{array}{c}\langle E_1 + E_2, F \rangle \longrightarrow \langle A_1, F \rangle \\ \langle E_2 + E_1, F \rangle \longrightarrow \langle A_1, F \rangle\end{array}} \qquad \frac{\langle E_1, F \rangle \longrightarrow \langle E_1', F \rangle, \langle E_2, F \rangle \longrightarrow \langle E_2', F \rangle}{\langle E_1 + E_2, F \rangle \longrightarrow \langle E_1' + E_2', F \rangle} \qquad \text{Ndt}$$

A web interface running a CLA interpreter [19] is also available.[1] To comply with the syntax of the tool, we will denote $\mathit{check}^{w}(Arg, R)$ by `checkw(Arg,R)` and $E + \cdots + E$ by `sum(E,...,E)`.

[1] CLA web interface: https://conarg.dmi.unipg.it/cla/.

3. Semantic Dependency of Arguments

In AFs, cycles represent intricate situations where arguments interact with each other in a circular manner, thus not providing an unambiguous interpretation for their acceptability. The circular dependency among arguments poses an issue even when attempting to reconstruct the argumentative process that led to the instantiation of a considered AF. An initial attempt to solve this problem is made by resorting to the feasible evaluation order of Definition 3, which involves treating arguments within the cycles as a single entity. Arguments attacked by this entity must always come before it, and those attacking it must always come after. Although the feasible evaluation order provides a workaround to the problem, choosing which argument to evaluate first within a cycle remains arbitrary. For instance, any feasible evaluation order for the AF of Figure 1 will place a as the first element and either e or f as the last, with b, c, and d appearing interchangeably right after a. In this section, we introduce a mechanism for refining the evaluation order within cycles through the notion of *semantic dependency*. The idea is that if the inclusion of an argument in a cycle does not alter the acceptability state of the other arguments within that cycle, then the argument can be introduced without being constrained by the dependency arising from the attacks it conducts.

The feasible evaluation order between two arguments a and b only considers syntactic dependency, i.e., whether an attack exists between a and b. This dependency does not involve reasoning at the semantic level, where the acceptability of arguments is asserted. However, studying how the acceptability state of argument changes as the debate unfolds is a key component for handling dynamics in AFs. When a new argument is proposed (added to the framework), its interaction with the already existing part of the AF may change the other arguments' acceptability state. With only the static representation of the debate, rendered through the AF, at our disposal, we can question which arguments, and in particular which attacks among them, are necessary to assert acceptability and which are negligible. In practice, if removing the attack conducted by an argument b towards another argument a changes the label of a, we conclude that a semantically depends on b. For the sake of a simpler notation, we will focus on the grounded semantics, which yields a unique labelling L. Regardless, extending the study to accommodate other semantics (like the complete one), which may involve multiple labellings, is straightforward.

Definition 4 (Semantic Dependency). *Given an AF* $F = \langle Arg, R\rangle$ *and two arguments* $a, b \in Arg$, *let* $G = \langle Arg, R \setminus \{(b, a)\}\rangle$. *We say that* a *semantically depends on* b *if and only if* $L^F(a) \neq L^G(a)$. *Otherwise, we say that* a *is semantically independent of* b.

Since the attack relation in AFs is asymmetric, semantic dependency is inherently asymmetric as well. Moreover, it is evident that an argument is semantically independent of all other arguments that do not directly or indirectly attack it.

Notation 2 (Invariant Attack). *Let* $F = \langle Arg, R\rangle$ *be an AF, where* $a, b \in Arg$ *are two arguments such that* $(b, a) \in R$. *We say that* (b, a) *is an invariant attack if* a *is semantically independent of* b.

Removing multiple invariant attacks is not guaranteed, in general, to leave the labelling unaltered. For instance, removing the invariant attacks (e, d) and (e, f) from the AF of Figure 1 changes the label of d from OUT to UND.

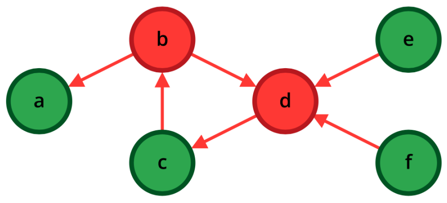

Figure 2: Example of an AF with the grounded labelling. Green arguments are IN, while red ones are OUT.

Example 2. *Consider now the cycle $\{b, c, d\}$ illustrated in Figure 2. From the syntactical perspective, we observe that d depends on c, c depends on b, and b, in turn, depends on d, creating a circular dependency. However, when analysing the situation at the semantic level, through the lens of grounded labelling, we find that only b semantically depends on c, whereas c and d do not semantically depend on d and b, respectively. In fact, excluding the attack (c, b) for the assessment of argument acceptability, b would receive the label* IN. *On the other hand, (b, d) and (d, c) are invariant attacks since d is rendered* OUT *by e, while c is* IN *in any case. Extending the analysis to arguments outside the cycle, we also find that d is semantically independent of e since excluding the attack (e, d) would still result in d being labelled as* OUT.

We use semantic dependency to refine the approach for evaluating arguments within cycles, specifically releasing syntactic dependencies that hinder the identification of the initial argument to be included. The idea is to evaluate first in each cycle one of the sink arguments (not attacking other arguments in the cycle) obtained by eliminating invariant attacks. Referring again to Figure 2, we want to select either b or d as the first argument to evaluate within the cycle. Indeed, they both conduct invariant attacks on other arguments in the cycle (b towards d and d towards c). In particular, we note that removing the attack (b, d) from the AF does not affect the labelling, indicating that b's syntactic dependence on d is irrelevant to the study of the semantics. Following this observation, we may disregard (b, d) and justifiably designate b as the first argument of the cycle to be evaluated.

Notation 3 (Invariant Attack Sets Within Cycles). *Consider an AF $F = \langle Arg, R\rangle$ and let $\overline{R} = \{(a, b) \mid (a, b) \in R \wedge b \in CiD(a)\}$ the set of attacks between arguments within a cycle. Then $I \subseteq \overline{R}$ is an invariant attack set within cycles of F if, given $G = \langle Arg, R \setminus I\rangle$, we have that $L^F(a) = L^G(a)$. Moreover, we denote with $\mathcal{I}$ the set of all invariant attack sets within cycles of F.*

Definition 5 (Semantics-Aware Evaluation Order). *Let $F = \langle Arg, R\rangle$ be an AF, and $\mathcal{I}$ the set of all invariant attack sets within cycles of F. A semantics-aware evaluation order for F is a numbering $n : Arg \to \mathbb{N}$ such that n is a feasible evaluation order for $G = \langle Arg, R \setminus I\rangle$, where $I \in \mathcal{I}$ and*

- *$\nexists I' \in \mathcal{I}$ such that $|ArC(\langle Arg, R \setminus I'\rangle)| < |ArC(G)|$;*
- *$\nexists I'' \in \mathcal{I}$ such that $I'' \subset I$ and $|ArC(\langle Arg, R \setminus I''\rangle)| = |ArC(G)|$.*

In the definition above, I represents the specific subset of attacks within the set $\mathcal{I}$ that, when removed from the set R, results in the minimum number of arguments involved in cycles within the AF $\langle Arg, R \setminus I\rangle$. Breaking a cycle by exploiting semantic dependency entails determining which argument to evaluate first. Once this argument has been found, all others in the cycle can be processed following their syntactic dependencies. However, in some instances, a cycle might include all arguments semantically dependent on each other, making it impossible to identify a specific attack to remove for rendering one of the arguments within this cycle a sink. Therefore, our goal is to minimise $ArC(G)$, i.e., the number of arguments in cycles, rather than trying to eliminate all the cycles, which may not be possible. Additionally, I is the minimal set with respect to set inclusion among all elements of $\mathcal{I}$ ensuring the smallest number of arguments in cycles is attained. This further minimisation allows the initial AF to change as little as possible to reconstruct its instantiation process.

Example 3. *Resuming from Example 2, we have that $\mathcal{I} = \{\emptyset, \{(b, d)\}, \{(d, c)\}, \{(b, d), (d, c)\}\}$ is the set of all invariant attack sets within cycles for the AF $F\langle Arg, R\rangle$ in Figure 2. In this set, both $\{(b, d)\}$ and $\{(d, c)\}$ are such that the removal of one of them eliminates the only cycle in the AF, and no smaller attack set does so. Following Definition 5, we can obtain two refined AFs: $G = \langle Arg, R \setminus \{(b, d)\}\rangle$, and $G' = \langle Arg, R\setminus\{(d, c)\}\rangle$. For G, we find two feasible evaluation orders, (a, b, c, d, e, f) and (a, b, c, d, f, e), while G' allows for 32 possible feasible evaluation orders, corresponding to any permutation of the six arguments in the AF following the partial order "a precedes b; b precedes c; d precedes b, e, and f". All these orders are semantics-aware evaluation orders for F.*

A thorough study of the complexity of calculating a semantics-aware evaluation order for an AF would require a more in-depth analysis, which is beyond the scope of this paper. As a preliminary consideration, we observe that the problem of finding invariant attacks is related to establishing a semantic equivalence between two AFs, which is generally intractable [20]. However, this problem might be simplified in the case where the compared AFs share the same set of arguments [21].

In general, feasible evaluation order and semantics-aware evaluation order are not related. Among all the semantics-aware evaluation orders for F, only (a, b, d, c, e, f), (a, b, d, c, f, e), (a, d, b, c, e, f), and (a, d, b, c, f, e) are also feasible evaluation orders for F since Definition 3 requires that a always be evaluated first and does not allow e and f to be evaluated before b and c. Moreover, considering F, (a, c, d, b, e, f) is a feasible evaluation order but not a semantics-aware evaluation order since the semantic dependency of b on c is not taken into account. Below, we show that for any AF it is always possible to find at least one feasible evaluation order that is also a semantics-aware evaluation order.

Proposition 1. *Given an AF $F = \langle Arg, R\rangle$, there exists a numbering $n : Arg \to \mathbb{N}$ such that n is a feasible evaluation order and a semantics-aware evaluation order for F.*

Proof. It is sufficient to observe that for any AF, it is always possible to find a semantics-aware evaluation order such that for every cycle C, all arguments belonging to C are evaluated before the arguments attacking it. If none of the attacks between arguments in C is invariant, the semantics-aware evaluation order collapses to a feasible evaluation order, as an argument in the cycle is chosen nondeterministically to be evaluated first. If C includes an argument a conducting an invariant attack towards another argument b within the cycle, disregarding the presence of (a, b), it is possible to evaluate the nodes within the cycle starting from a and ending at b. The order obtained in both cases is a feasible evaluation order by Definition 3. □

4. Semantics-Aware Evaluation Order in Small Real-World Scenarios

In this section, we illustrate with two examples of how the semantics-aware evaluation order can be employed to reconstruct the progression of a debate by studying the sequence in which arguments might have been introduced. The first example, taken from [22], reflects Anna's reasoning about the suitability of an apartment for rent, and was designed as a case study for a Temporal AF that incorporates the availability of arguments over time. Using an AF already tailored for dynamic contexts aligns perfectly with our objectives. Figure 3 shows the reference AF for this example, with the arguments listed below. Originally comprising five arguments, we expanded the example by incorporating arguments f and g, along with the attacks (b, f), (f, c) and (g, f) which generate a cycle. This addition further demonstrates the effectiveness of the proposed methodology. Note that the arguments we consider are abstract and are associated here with natural language sentences for exemplification only.

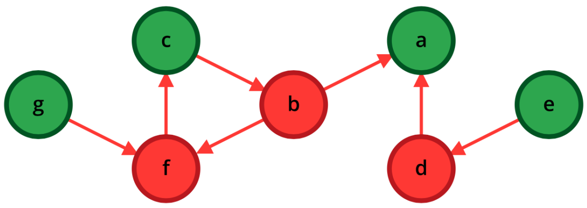

Figure 3: AF with grounded labelling presenting Anna's reasoning about renting an apartment.

a: Anna should rent the apartment she found.
b: The apartment seems to have humidity problems.
c: The owner is committed to solving structural problems in the apartment.
d: A nightclub is set to open nearby shortly.
e: Laws forbid the opening of nightclubs in the area.
f: The owner, planning to sell the property soon, is unlikely to fund long-term repairs.

g: Due to legal constraints, the apartment cannot be sold immediately.

Let us call F the AF in Figure 3 . The portion of F that is most interesting to analyse for this example is the cycle formed by the arguments b, c and f. The situation described is as follows: humidity problems compromise the plan to sell the apartment shortly – (b, f); intent on selling, the owner is not interested in structural work on the property – (f, c); the owner is committed on solving the humidity problem – (c, b). The presence of this cycle results in the identification of possible less meaningful argument orderings when only using the syntactic dependency generated by attacks. Take, for instance, the sequence (a, d, e, c, b, f, g), which represents an evaluation order for F. The placement of c before b in this sequence may not reflect a realistic scenario. Essentially, evaluating the owner's commitment (c) before acknowledging the humidity problems (b) may lead Anna to initially feel reassured about the apartment's issues, only to realise later on that there are specific, potentially unaddressed problems. This sequence might not provide the most logical flow for decision-making. It would be more realistic for Anna to first consider the potential issue (b) before contemplating the owner's commitment (c), thereby creating a more logical dialogue as she considers her options. We show that (a, d, e, c, b, f, g) is not a semantics-aware evaluation order for F. First, we identify the set $\mathcal{I} = \{\emptyset, \{(b, f)\}, \{(f, c)\}, \{(b, f), (f, c)\}\}$ of all invariant attack sets within cycles of F. It is easy to verify that all attacks in elements of $\mathcal{I}$ are either from OUT to IN arguments or from OUT to OUT. Since there is no $\overline{I} \in \mathcal{I}$ such that $(c, b) \in \overline{I}$, according to Definition 5, it is impossible for a semantics-aware evaluation order to evaluate argument c before b.

On the other hand, sequences such as (a, d, e, b, c, f, g), (a, d, e, f, g, b, c), and (f, g, a, b, c, d, e) qualify as semantics-aware evaluation orders. To substantiate this claim, we search for minimal sets $I \in \mathcal{I}$ for which $G = \langle Arg, R \setminus I \rangle$ is acyclic. Both $\{(b, f)\}$ and $\{(f, c)\}$ are viable candidates as minimal attack invariant sets. Considering the invariant attack (b, f) for removal, permits b to be evaluated directly after a, c after b, and f subsequent to c, giving rise to orders like (a, d, e, b, c, f, g). Alternatively, when removing (f, c), f becomes a sink argument in $G = \langle Arg, R \setminus \{(f, c)\} \rangle$, allowing its evaluation at any stage; thus, b follows a and f, g comes after f, and c after b. Consequently, orders such as (a, d, e, f, g, b, c) and (f, g, a, b, c, d, e) are possible under these conditions.

For the next example, we use a deliberation dialogue extracted from [23], focusing on modelling information exchange for managing shared resources. The dialogue features three participants: Alice, a farmer; Bob, an oyster farmer; and Carol, a government representative. They are engaged in a discussion concerning the effects of fertilisers on oysters, as delineated below. The reference AF, which we will call D, is shown in Figure 4. We adjusted the original example to enhance its relevance for our study, resulting in the emergence of two distinct cycles: $\{a, c, e, d\}$ and $\{a, f, g, b\}$.

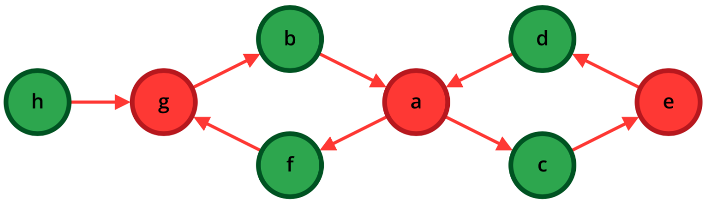

Figure 4: AF with grounded labelling exploring the impact of fertilisers on oysters.

a: (Alice) Using a lot of fertiliser helps to have a big yield.
b: (Bob) Using a lot of fertiliser pollutes the lake and harms the oyster.
c: (Carol) Low-income farms incur significant fertiliser costs, attracting authority attention.
d: (Carol) Using more fertiliser than the norm implies a fine.
e: (Alice) There is no risk of being controlled because of lack of means.
f: (Carol) Farms bear significant expenses to mitigate the harmful impacts of pesticides.
g: (Alice) Lake pollution is not linked to pesticides.

h: (Bob) The studies conducted on the groundwater were inaccurate.

To compute a semantics-aware evaluation order, we must identify beforehand a minimal invariant attack set within cycles of $D = \langle Arg, R \rangle$ capable of minimising, upon its removal, the number of arguments within cycles. Since we have two cycles in D, we want to remove exactly two invariant attacks, one from each cycle. Within $\{a, c, e, d\}$, we find three invariant attacks: (a, c), (d, a), and (e, d), while in $\{a, f, g, b\}$ we find four: (a, f), (b, a), (f, g), and (g, b). Only some pairs combining invariant attacks from each cycle belong to $\mathcal{I}$. For instance, the pair $\{(b, a), (d, a)\}$ is not an invariant attack set within cycles of D since $L^D(a) = \mathsf{OUT} \neq \mathsf{IN} = L^H(a)$ with $H = \langle Arg, R \setminus \{(b, a), (d, a)\} \rangle$. This illustrates why, to maintain a coherent debate flow where counterarguments are effectively positioned, Alice's argument (a) that using a lot of fertiliser helps to have a big yield should logically precede at least one between Bob's argument (b) that using a lot of fertiliser pollutes the lake and harms the oyster, or Carol's argument (d) that using more fertiliser than the norm implies a fine. In total, we find eleven minimal invariant attack set within cycles of D such that their removal produces an acyclic AF: $\{(a, c), (a, f)\}$, $\{(a, c), (f, g)\}$, $\{(a, c), (g, b)\}$, $\{(b, a), (a, c)\}$, $\{(b, a), (e, d)\}$, $\{(d, a), (a, f)\}$, $\{(d, a), (f, g)\}$, $\{(d, a), (g, b)\}$, $\{(e, d), (a, f)\}$, $\{(e, d), (f, g)\}$, $\{(e, d), (g, b)\}$.

For instance, if we select $\{(d, a), (g, b)\}$, we can obtain the semantics-aware evaluation order (d, e, c, g, h, f, a, b). Following this order, the debate would start with Carol highlighting the potential fines for using more fertiliser than normative levels (d), setting a regulatory tone. Alice quickly counters by questioning the effectiveness of such controls due to limited enforcement resources (e). Carol responds by pointing out the significant financial burden on low-income farms from high fertiliser costs, which attracts more regulatory scrutiny (c), contrasting with Alice's point about enforcement challenges. Alice then shifts the topic slightly by denying any link between lake pollution and pesticides (g), possibly trying to deflect from broader environmental concerns. Bob introduces a twist by questioning the accuracy of groundwater studies (h), which casts doubt on all environmental impact claims discussed. Carol continues by examining farms' economic burden in mitigating the negative effects of pesticides (f), countering Alice's previous assertions. Alice then praises the agricultural benefits of high fertiliser use, emphasising increased yields (a). Bob wraps up the discussion by focusing on the environmental damages of excessive fertiliser use, specifically highlighting the pollution of lakes and the harm to oysters (b), directly opposing Alice's utilitarian view and bringing the debate full circle to the consequences of agricultural practices on the environment.

5. Automatic Instantiation of AFs with CLA

Identifying invariant attacks is a key step in establishing a semantics-aware evaluation order. The problem of detecting modifications at the syntactic level of AFs that do not affect the semantics has already been addressed in the literature with different approaches and methodologies, e.g., [14, 21]. In [14], the authors discuss the persistence of semantics after the removal of attacks. For complete, preferred and grounded semantics, all attacks except IN to OUT and UND to UND can be removed without altering the label of any argument (attacks from IN to IN, from IN to UND, and from UND to IN are considered impossible because they contravene Definition 2). Attacks satisfying removal persistence are invariant but do not constitute the totality of invariant attacks of an AF. For example, an attack originating from an IN argument a to an OUT argument b is invariant if b is attacked by another IN argument c. Therefore, this method is not immediately viable, as it necessitates further research to develop a procedure capable of detecting all the attacks of interest.

In a separate line of work, Baumann has explored semantic equivalence properties between two AFs where the second framework is derived from specific modifications made to the first [21]. Among other aspects, his work examines semantic equivalence following the operation of local deletion, which involves the removal of attacks between arguments. Despite lacking an operational definition for identifying invariant attacks, we will leverage this concept to automate the process of identifying such attacks.

More in detail, we are interested in finding sets of attacks I with the properties described in Definition 5: I must be the smallest set whose removal minimises the number of arguments involved in cycles of an AF without changing the state of acceptability of the arguments. To approach this problem, we propose an implementation in Answer Set Programming (ASP) [24], a type of logic programming that operates based on the answer set semantics. In this framework, solutions to a specific problem are delineated as selected models, or answer sets, of the associated logic program. ASP is particularly well-suited for the specific task of minimising the number of arguments belonging to cycles of AFs while preserving argument acceptability due to its inherent strengths in handling complex search and optimisation problems within combinatorial decision-making environments.

5.1. ASP Optimization for Minimal Invariant Attack Sets

We outline the main passages of the proposed APS implementation, consisting of rules for determining paths and cycles, attacks to remove, grounded labelling, and semantic equivalence, alongside necessary minimisations. The complete code can be found in the appendix of this paper. The initial step in our program involves supplying an AF in the ASPARTIX format [25], which entails defining arguments and attacks using respectively the predicates `arg/1` and `att/2`. Afterwards, we define a choice rule `{ remove(X, Y) : att(X, Y) }` that allows any attack (X, Y) to be considered for removal based on the criteria set later in the program. We also include definitions for paths and membership in a cycle before and after the potential removal of attacks. For example, a path from X to Y in the modified AF exists if X attacks Y and the attack (X, Y) has not been removed.

To ensure that the labels assigned to arguments will not change after the attacks are removed, we must first obtain a grounded labelling for the AF before and after the modifications. We specify rules reflecting the conditions given in Definition 2 for labelling the AF before and after the changes. In particular, we explicitly declare that an argument is labelled UND if it is neither IN nor OUT. We also introduce a constraint imposing that exactly one label can be assigned to each argument. Then, we obtain the grounded labelling by minimising the number of IN labels assigned to arguments of the AF.

At this point, we impose the semantic equivalence constraint between the original AF and that obtained after removing the attacks. Concretely, we require that if an argument X has a certain label before removal, this label is maintained after removal. Next, we give two optimisation statements to minimise the number of arguments that are part of cycles and the number of attacks to be removed. Finally, we filter the output to display only instances of the `remove/2` predicate, showing which attacks are chosen for removal.

5.2. A CLA Program to Instantiate AFs

We now show how AFs can be instantiated through CLA constructs [18] which introduce arguments following a semantics-aware evaluation order. A key feature of CLA is its ability to handle concurrency and nondeterminism, enabling the simulation of the dynamic aspects of real-life debates. The CLA program provided in Listing 1 realises the AF of Figure 4 by taking into account syntactic and semantic dependencies between arguments. All arguments, except for d and g, are integrated into the debate only when their syntactic dependencies are satisfied (Listing 1, lines $3 - 10$). In detail, the instruction `checkw({c,f},{}) -> add({a},{(a,f),(a,c)}) -> success) -> success` adds a and its attacks into the shared knowledge base only when both c and f (the attacked arguments) are already present. No check is made for d and g, so these arguments can immediately be added (lines 6 and 9, respectively), ignoring their syntactic dependencies. $I = \{(d, a), (g, b)\}$ is a minimal invariant attack set in the sense of Definition 5, hence the invariant attacks (d, a) and (g, b) have initially been ignored to break the two cycles in the AF. They are incorporated afterwards (lines $1 - 2$) when all the arguments they involve have already been added. All the instructions are executed in parallel, enabling the addition of each argument as soon as the conditions in the guarded agent are satisfied. It follows that every sequence of additions obtained as an execution trace of the program in Listing 1 enumerates the arguments in a semantics-aware evaluation order.

Listing 1: CLA program instantiating the AF of Figure 4.

```
checkw({d,a},{}) -> add({},{(d,a)}) -> success ||
checkw({g,b},{}) -> add({},{(g,b)}) -> success ||
checkw({c,f},{}) -> add({a},{(a,f),(a,c)}) -> success ||
checkw({a},{}) -> add({b},{(b,a)}) -> success ||
checkw({e},{}) -> add({c},{(c,e)}) -> success ||
add({d},{}) -> success ||
checkw({d},{}) -> add({e},{(e,d)}) -> success ||
checkw({g},{}) -> add({f},{(f,g)}) -> success ||
add({g},{}) -> success ||
checkw({g},{}) -> add({h},{(h,g)}) -> success;
```

We provide a procedure, illustrated in Algorithm 1, for automatically obtaining a CLA program that builds an AF by adding arguments in semantics-aware evaluation order. Our input is an AF $F = \langle Arg, R\rangle$, while the output is a string corresponding to a CLA program. The procedure initiates by invoking `find_minimal_invariant_attack_set`(F) (line 2 of Algorithm 1), namely a call to the ASP program described in Section 5.1 used for identifying a set I of attacks that, when removed, left the AF with the minimum number of arguments in circular dependency. Subsequently, in line 3, we define G as the AF obtained from F by removing the attacks in I. Afterwards, the process starts to generate the CLA program. This phase is divided into two steps. Initially, we write CLA instructions to add all attacks within I into the shared knowledge base (lines $5-6$), provided that all arguments engaged in these attacks have already been added. Then, we call the procedure `gen_cla_prog`(G) described in [16], which generates a CLA program for instantiating G while ensuring the arguments are added in a feasible evaluation order.

Algorithm 1: Procedure for generating a CLA program to instantiate a specified AF.

```
Data: AF $F = \langle Arg, R\rangle$
Result: string $S$
1 procedure gen_cla_prog_saeo($F$):
2     $I$ = find_minimal_invariant_attack_set($F$)          // I: set of attacks
3     $G = \langle Arg, R \setminus I\rangle$
4     $S$ = “”
5     foreach $(a, b)$ in $I$ do
6         $S$ = $S$ + “checkw({$a, b$}, {}) −> add({}, {$(a, b)$}) −> success || ”
7     $S$ = $S$ + gen_cla_prog($G$)
```

The program resulting from the Algorithm 1 allows the shared knowledge base to be manipulated to shape the desired AF. This instantiation involves the addition of arguments according to the syntactic and semantic dependency relations dictated by the attacks within the framework. Due to the inclusion of parallel and nondeterministic operations, multiple unique executions may occur, leading to different (semantics-aware evaluation) orders for the addition of arguments.

6. Conclusion

A semantics-aware evaluation order introduces the arguments in a meaningful sequence, simulating what might have happened during the instantiation of the AF under consideration. Arguments that receive attacks are evaluated before those that initiate them, except for one argument per cycle, whose invariant attack is bypassed to allow the other arguments for proper evaluation. "The refined framework G, obtained as per Definition 5, can still contain cycles since it is possible that no invariant attack is identified within some cycles, necessitating the adoption of an arbitrary sequence provided by the feasible evaluation order for arguments involved in circular dependencies. Furthermore, multiple arguments conducting invariant attacks may be candidates within a cycle for initial evaluation. Even in such scenarios, there is no specific criterion for prioritising one argument over others for being evaluated first. Hence, the approach with the semantics-aware evaluation order might not always identify a single argument to evaluate first in every cycle. Nonetheless, such an order helps to limit the cases where the choice remains arbitrary, providing more meaningful insights into the possible sequence of arguments that constitutes the unfolding of a debate.

In the future, we plan to advance this work by studying a further evaluation approach based on three assumptions that better capture some aspects of real argumentative processes. Firstly, we can assume that a debate among multiple agents takes place through an exchange of arguments that replicate and counter the arguments already introduced. Each new argument inserted must therefore keep the graph connected. Secondly, we assume that when an agent inserts an argument, this will be labelled IN, as no one has had the chance to reply yet. This assumption implies a different use of semantic dependence from that presented in this paper, where instead the inserted arguments can have any label. Finally, it is reasonable to think that new arguments must change the acceptability of some other argument in the AF, otherwise there would have been no reason to insert such an argument. With these assumptions, we obtain sequences of arguments that could closely mirror argumentation processes with the outlined characteristics, though this approach may narrow the scope of application. For example, the semantics-aware evaluation order allows new arguments to be inserted without being connected to the previously instantiated part of the AF.

Acknowledgments

The authors are member of the INdAM Research group GNCS and of Consorzio CINI. This work has been partially supported by: GNCS-INdAM, CUP_E53C23001670001; GNCS-INdAM, CUP_E53C22001930001; EU PNRR MUR PRIN project EPICA: "Empowering Public Interest Communication with Argumentation" - Next Generation (J53D23007220006); EU PNRR MUR project SERICS - Next Generation (PE00000014); University of Perugia - Fondo Ricerca di Ateneo (2020, 2021, 2022) - Projects BLOCKCHAIN4FOODCHAIN, FICO, AIDMIX, "Civil Safety and Security for Society"; European Union - Next Generation EU NRRP-MUR - Project J97G22000170005 VITALITY: "Innovation, digitalisation and sustainability for the diffused economy in Central Italy"; Piano di Sviluppo e Coesione del Ministero della Salute 2014-2020 - Project I83C22001350001 LIFE: "the itaLian system Wide Frailty nEtwork" (Linea di azione 2.1 "Creazione di una rete nazionale per le malattie ad alto impatto" - Traiettoria 2 "E-Health, diagnostica avanzata, medical devices e mini invasività").

References

[1] C. Dutilh Novaes, Argument and Argumentation, in: The Stanford Encyclopedia of Philosophy, Fall 2022 ed., Metaphysics Research Lab, Stanford University, 2022.

[2] L. Caroprese, E. Vocaturo, E. Zumpano, Argumentation approaches for explanaible AI in medical informatics, Intell. Syst. Appl. 16 (2022) 200109.

[3] L. Xiao, Towards evidence-based argumentation graph for clinical decision support, in: 35th IEEE International Symposium on Computer-Based Medical Systems, CBMS 2022, Shenzen, China, July 21-23, 2022, IEEE, 2022, pp. 400–405.

[4] A. Vassiliades, N. Bassiliades, T. Patkos, Argumentation and explainable artificial intelligence: a survey, Knowl. Eng. Rev. 36 (2021) e5.

[5] E. Karafili, A. C. Kakas, N. I. Spanoudakis, E. C. Lupu, Argumentation-based security for social good, in: 2017 AAAI Fall Symposia, Arlington, Virginia, USA, November 9-11, 2017, AAAI Press, 2017, pp. 164–170.

[6] Y. Yu, V. N. L. Franqueira, T. T. Tun, R. J. Wieringa, B. Nuseibeh, Automated analysis of security requirements through risk-based argumentation, J. Syst. Softw. 106 (2015) 102–116.

[7] S. Bistarelli, F. Rossi, F. Santini, C. Taticchi, Towards visualising security with arguments, in:

Proceedings of the 30th Italian Conference on Computational Logic, Genova, Italy, July 1-3, 2015, volume 1459 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015, pp. 197–201.

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

[9] P. Baroni, M. Caminada, M. Giacomin, An introduction to argumentation semantics, Knowl. Eng. Rev. 26 (2011) 365–410.

[10] H. Prakken, M. D. Winter, Abstraction in argumentation: Necessary but dangerous, in: Computational Models of Argument - Proceedings of COMMA 2018, Warsaw, Poland, 12-14 September 2018, volume 305 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2018, pp. 85–96.

[11] G. Alfano, S. Greco, F. Parisi, Incremental computation in dynamic argumentation frameworks, IEEE Intell. Syst. 36 (2021) 80–86.

[12] S. Doutre, J. Mailly, Constraints and changes: A survey of abstract argumentation dynamics, Argument Comput. 9 (2018) 223–248.

[13] S. Bistarelli, F. Santini, C. Taticchi, On looking for invariant operators in argumentation semantics, in: Proceedings of the Thirty-First International Florida Artificial Intelligence Research Society Conference, FLAIRS 2018, Melbourne, Florida, USA. May 21-23 2018, AAAI Press, 2018, pp. 537–540.

[14] T. Rienstra, C. Sakama, L. W. N. van der Torre, Persistence and monotony properties of argumentation semantics, in: Theory and Applications of Formal Argumentation - Third International Workshop, TAFA 2015, Buenos Aires, Argentina, July 25-26, 2015, Revised Selected Papers, volume 9524 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 211–225.

[15] R. Baumann, What does it take to enforce an argument? minimal change in abstract argumentation, in: ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012, volume 242 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2012, pp. 127–132.

[16] S. Bistarelli, C. Taticchi, Deriving dependency graphs from abstract argumentation frameworks, in: AIxIA 2023 - Advances in Artificial Intelligence - XXIInd International Conference of the Italian Association for Artificial Intelligence, AIxIA 2023, Rome, Italy, November 6-9, 2023, Proceedings, volume 14318 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 17–29.

[17] M. Caminada, On the issue of reinstatement in argumentation, in: Logics in Artificial Intelligence, 10th European Conference, JELIA 2006, Liverpool, UK, September 13-15, 2006, Proceedings, volume 4160 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 111–123.

[18] S. Bistarelli, C. Taticchi, A concurrent language for modelling agents arguing on a shared argumentation space, Argument Comput. 15 (2024) 21–48.

[19] S. Bistarelli, C. Taticchi, Introducing a tool for concurrent argumentation, in: Logics in Artificial Intelligence - 17th European Conference, JELIA 2021, Virtual Event, May 17-20, 2021, Proceedings, volume 12678 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 18–24.

[20] R. Baumann, W. Dvorák, T. Linsbichler, S. Woltran, A general notion of equivalence for abstract argumentation, Artif. Intell. 275 (2019) 379–410.

[21] R. Baumann, Context-free and context-sensitive kernels: Update and deletion equivalence in abstract argumentation, in: ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014), volume 263 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2014, pp. 63–68.

[22] M. C. Budán, M. L. Cobo, D. C. Martínez, G. R. Simari, Bipolarity in temporal argumentation frameworks, Int. J. Approx. Reason. 84 (2017) 1–22.

[23] N. Paget, G. Pigozzi, O. Barreteau, Information sharing for natural resources management, 2013.

[24] V. Lifschitz, Answer Set Programming, Springer, 2019.

[25] U. Egly, S. A. Gaggl, S. Woltran, ASPARTIX: implementing argumentation frameworks using answer-set programming, in: Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings, volume 5366 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 734–738.

Appendix

Listing 2 provides an ASP implementation designed to identify the smallest set of attacks whose elimination minimises the number of arguments involved in cycles within an AF while maintaining the original acceptability state of the arguments.

Listing 2: ASP Implementation for Minimal Invariant Attack Sets.

```
% AF provided in a separate file
#include "af.lp".

% Choice rule for attacks to remove
{ remove(X, Y) : att(X, Y) }.

%Paths and cycles
path(X, Y) :- att(X, Y).
path(X, Y) :- path(X, Z), att(Z, Y).
path_after(X, Y) :- att(X, Y), not remove(X, Y).
path_after(X, Y) :- path_after(X, Z), att(Z, Y), not remove(Z, Y).
in_cycle(X) :- path(X, X).
in_cycle_after(X) :- path_after(X,X).

%Grounded labelling
in(X) :- arg(X), not has_non_out_attacker(X).
has_non_out_attacker(X) :- att(Y, X), arg(Y), not out(Y).
out(X) :- arg(X), att(Y, X), arg(Y), in(Y).
und(X) :- arg(X), not in(X), not out(X).
:- arg(X), not 1 { in(X); out(X); und(X) } 1.
#minimize { 1@4, X : in(X) }.

%Grounded labelling after the removal
in_after(X) :- arg(X), not has_non_o_a_a(X).
has_non_o_a_a(X) :- att(Y, X), arg(Y), not out_after(Y), not remove(Y, X).
out_after(X) :- arg(X), att(Y, X), arg(Y), in_after(Y), not remove(Y, X).
und_after(X) :- arg(X), not in_after(X), not out_after(X).
:- arg(X), not 1 { in_after(X); out_after(X); und_after(X) } 1.
#minimize { 1@3, X : in_after(X) }.

%Semantic equivalence
:- in(X), not in_after(X).
:- out(X), not out_after(X).
:- und(X), not und_after(X).

%Minimisation
#minimize { 1@2, X : in_cycle_after(X) }.
#minimize { 1@1, X, Y : remove(X, Y) }.

%Output
#show remove/2.
```