Argument & Computation 15 (2024) 121–159
DOI 10.3233/AAC-220018
IOS Press

Learning argumentation frameworks from labelings

Lars Bengel [a,*], Matthias Thimm [a] and Tjitze Rienstra [b]
[a] *Artificial Intelligence Group, FernUniversität in Hagen, Germany*
E-mails: lars.bengel@fernuni-hagen.de, matthias.thimm@fernuni-hagen.de
[b] *Department of Data Science and Knowledge Engineering, Maastricht University, The Netherlands*
E-mail: t.rienstra@maastrichtuniversity.nl

Abstract. We consider the problem of learning argumentation frameworks from a given set of labelings such that every input is a σ-labeling of these argumentation frameworks. Our new algorithm takes labelings and computes attack constraints for each argument that represent the restrictions on argumentation frameworks that are consistent with the input labelings. Having constraints on the level of arguments allows for a very effective parallelization of all computations. An important element of this approach is maintaining a representation of *all* argumentation frameworks that satisfy the input labelings instead of simply finding any suitable argumentation framework. This is especially important, for example, if we receive additional labelings at a later time and want to refine our result without having to start all over again. The developed algorithm is compared to previous works and an evaluation of its performance has been conducted.

Keywords: Abstract argumentation, labelings, semantics, learning

1. Introduction

An abstract argumentation framework due to Dung [15] is defined as a graph, where the nodes are arguments and the edges represent attacks between these arguments. An attack from an argument A to another argument B means that, if we consider the argument A to be acceptable, then we have to reject B, since A contradicts B. The goal of this approach is to model human argumentation in a formal manner and to use this model for reasoning. Abstract argumentation frameworks are interpreted using the notion of an *(argumentation) semantics*. A semantics characterizes acceptable sets of arguments (called *extensions*), which are supposed to model a valid outcome of the argumentation represented by an argumentation framework. In particular, they usually require that extensions are *conflict-free*, i.e., that there is no conflict between arguments of the extension, and that it defends itself, i.e., it counterattacks all its attackers (the latter property is called *admissibility*). While Dung proposed several different semantics in his seminal paper, like complete, grounded, preferred, and stable semantics, there have since been many proposals for new semantics [4], all with different goals in mind.

Extension-based semantics focus on the accepted arguments only, so, any extension with respect to some semantics only consists of the arguments that are considered acceptable. That means we also get the implicit knowledge that all other arguments are not accepted. However, we do not know for what reason these arguments are not accepted. They could be directly attacked by some acceptable argument,

*Corresponding author. E-mail: lars.bengel@fernuni-hagen.de.

meaning the extension actively contradicts them. In contrast to that they could also be not accepted because they are part of some conflict, unrelated to the acceptable arguments, which means these arguments could potentially also be compatible with the acceptable arguments. This additional information can be encoded in labelings [17]. In a labeling each argument of the argumentation framework receives a label, if it is considered acceptable the argument is labeled in, if the argument is attacked by some acceptable argument it gets the label out and otherwise it receives the label undec.

In *Inductive Logic Programming* [21] we derive from a given set of examples and background knowledge a hypothesized logic program that entails these examples. In a similar manner, algorithms for *Bayesian Network Learning* [13,18] are concerned with inducing a bayesian network from given data. Overall, both of these approaches are concerned with generalizing from observations or examples to a program or network which entails this knowledge and therefore also allows for inferring new knowledge in line with the input knowledge.

This approach is also interesting for the field of argumentation. In general, argumentation semantics allows us to perform *inference* with an abstract argumentation framework, by determining semantical information (extensions or labelings) from given syntactical information (in the form of an argumentation framework). In this work, we are, as already hinted at above, interested in the reverse direction, i.e., the process of *learning* a syntactic structure from semantical information. So, given a set of labelings we wish to infer a suitable attack relation that explains this input. That means, we want to find an argumentation framework that has at least those labelings.

This is interesting, if we look for example at the area of *argument mining* [20] which is concerned with extracting arguments and their relations from text. There, extracting the relations between arguments is especially hard [18]. This is one area where inductive learning approaches are useful, since they can aid us in constructing the attack relation for a set of arguments. Additionally, learning the attack relation for a set of arguments can also be helpful in terms of explainability [1,12] since this allows us to construct better argumentative explanations based on the underlying argumentation framework itself, see e.g., [16,27]. It will also allow us to better understand how the labelings originated and how we could challenge them, as the following example will highlight.

Assume a situation where we are able to meet up with a person A that has an anti-vax stance. We are able to discuss with them about the topic of the COVID-19 vaccine and our goal is to understand their personal reasoning process. Doing so may allow us to find better counterarguments in future discussions on the topic and it will also aid us in identifying false conclusions that A might have made, so that we can potentially clear them up. The internal reasoning process of A is represented as an argumentation framework, which is hidden to us. During the discussion, we may ask A for his opinion on different statements on the topic. From A's answers we can then infer a labeling, which essentially assigns to each argument their stance, i.e., either they agree (in), disagree (out) or their stance is unclear/unknown (undec). Given such a labeling, our goal is to reconstruct the hidden argumentation framework of A.

For example, lets consider the following four arguments:

(a) *The development of the COVID-19 vaccine has been rushed.*
(b) *The COVID-19 vaccine reduces the risk of infection.*
(c) *The COVID-19 vaccine does not work at all.*
(d) *The COVID-19 vaccine is dangerous and has various side effects.*

From our first discussion with A we may conclude that they agree with the arguments a and d while disagreeing with b and c. This would then translate to the complete labeling $\ell_1 = \{\mathsf{in} : \{a, d\}, \mathsf{out} : \{b, c\}, \mathsf{undec} : \emptyset\}$. A complete labeling is defined such that arguments not attacked by an out-labeled

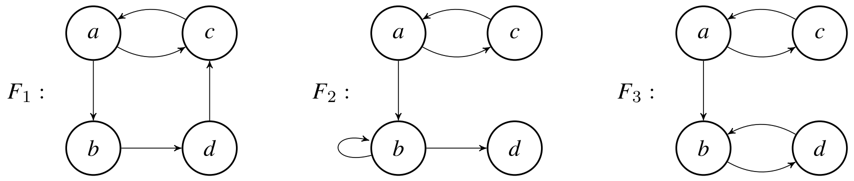


Fig. 1. Some argumentation frameworks that were reconstructed from the labelings ℓ_1 and ℓ_2 of the arguments $\{a, b, c, d\}$ as defined above. Note that these frameworks are not supposed to model factual information but rather the reasoning process of someone with potentially flawed logic or limited understanding of the topic at hand.

argument must be labeled in, all arguments attacked by an in-labeled arguments are labeled out and otherwise the argument is labeled undec. If we now try to reconstruct their internal argumentation framework, for example by simply brute-forcing every possible combination of attacks, we will find that multiple argumentation frameworks are consistent with this labeling. See for example the three argumentation frameworks in Fig. 1. At this point, it is impossible for us to decide which argumentation framework correctly represents the reasoning of A. If we were to choose one, we would have to make some additional assumptions and discard all other argumentation frameworks. This may lead to unintentionally discarding the actual framework that we are looking for. For that reason, it is sensible to maintain a representation of all of these argumentation frameworks. This allows us to easily incorporate additional information in the future.

For example, if we want to refine our understanding of A's reasoning, we could initiate another discussion with them so that we can obtain another complete labeling. After our second discussion, we may get the labeling $\ell_2 = \{\mathsf{in} : \{c\}, \mathsf{out} : \{a\}, \mathsf{undec} : \{b, d\}\}$. With this additional labeling, we can again reconstruct argumentation frameworks that are consistent with both ℓ_1 and ℓ_2. In our example, we will find that only F_2 and F_3 from Fig. 1 are consistent with both labelings, while F_1 is only consistent with ℓ_1. This again highlights the advantage of maintaining a representation of all consistent argumentation frameworks after each step.

With the argumentation framework(s) that we have learned from the labelings, we can then for example try to identify missing or nonsensical attacks (e.g. there should probably be an attack from b on c). Another thing we can do, is identify key arguments that we can disprove to sway the opinion of the person A. For example, if we present some evidence that the vaccine development has in fact not been rushed, A might change their opinion on the argument a and subsequently this could influence their stance on arguments b and c.

In this work, we will answer the following research question:

Research Question. *For a given set of labelings $L = \{\ell_1, \dots, \ell_n\}$ and a set of semantics $S = \{\sigma_1, \dots, \sigma_n\}$, how can we compute the set of argumentation frameworks $\mathbb{F}$ such that for all $F \in \mathcal{F}$ the following two conditions hold:*

(1) *if $F \in \mathbb{F}$, then $\forall \ell_i \in L : \ell_i$ is a σ_i-labeling of F, and*
(2) *if $F \notin \mathbb{F}$, then $\exists \ell_i \in L : \ell_i$ is* not *a σ_i-labeling of F.*

We explicitly allow the labelings to have different semantics in order to be more general, especially since different semantics have different advantages depending on the application [4]. In other words, for a given set of labelings L with respect to different semantics, we want to define a method to construct the set of *all* argumentation frameworks F, such that at least all labelings in L are labelings of F. In that case, we will also say that the argumentation framework F is *consistent* with L.

The general problem of learning argumentation frameworks has only received limited attention so far [23,25] and none of these works are able to address the exact scenario we outlined above. In their paper, Niskanen et al. [23] investigate the synthesis of argumentation frameworks from positive and negative examples. These examples are extensions with respect to conflict-free, admissible, complete or stable semantics. A positive example E is then encoded in some logical formula, representing the information in E. All positive examples are encoded with a corresponding formula for their semantics and the negative examples are encoded as the negation of those formulae. Their approach also allows for the attachment of weights to each example. The encoded examples, together with their weights, are given to a MaxSAT solver, which returns the optimal solution, i.e., an argumentation framework, which is consistent with all or most of the positive examples and the least of the negative examples.

On the other hand, Riveret et al. [25] propose an approach for learning attacks from labelings. Their algorithm is built around the grounded semantics and takes a stream of labelings as input. Since it is not possible to reconstruct the argumentation framework by only knowing the (uniquely determined) grounded extension, Riveret et al. decided to extend their scenario as follows. Their algorithm works with knowledge about the sub-frameworks of the argumentation framework, i.e., restrictions of the graph on different subsets of arguments. The input of their algorithm therefore consists of grounded labelings of different sub-frameworks. For the process of learning the hidden argumentation framework we start with a weighted argumentation framework, where every attack has an initial weight of 0. Then, for each input labeling we iterate over all combinations of two arguments and check whether they satisfy one of four rules, which essentially encode the different conditions a grounded labelings has to fulfill. If a rule is satisfied, the weight of the attacks between these two arguments are updated accordingly. After examining all input labelings we remove all attacks whose weight is below a certain threshold from the argumentation framework. The resulting argumentation framework is then the approximation of the original hidden argumentation framework.

While the above mentioned approaches can be used to construct argumentation frameworks from labelings or extensions, they only return a single argumentation framework as a result. This might not always be desirable, as we illustrated in the above example. In the case of the approach by Riveret et al. [25], their algorithm also relies on additional information about sub-frameworks.

Therefore, in this work, we propose an algorithm that better addresses this scenario by maintaining a representation of all consistent argumentation frameworks at all times. In general, our algorithm works according to the following procedure: We are given a set of arguments Arg as well as a set of input labelings L. For each input labeling $\ell \in L$ we compute a set of attack constraints $C_\ell = \{C_{a,\sigma}\}_{a \in \mathsf{Arg}}$, depending on its semantics. Afterwards we combine these constraints, so that we have one constraint C_a for each argument $a \in \mathsf{Arg}$. Finally, they can then be used to construct a set of argumentation frameworks $\mathbb{F}$ which satisfy all constraints. That means every argumentation framework $F \in \mathbb{F}$ is consistent with the set of input labelings L.

To summarise, the contributions of this paper are as follows:

(1) We introduce the concept of attack constraints and define semantic constraint functions that compute attack constraints for a given labeling (Section 3.1–3.2).
(2) We define a novel algorithm for learning argumentation frameworks from labelings that utilizes these constraints to maintain a representation of all consistent argumentation frameworks (Section 3.3).
(3) We conduct an experimental evaluation of our algorithm, comparing a naive and a parallelized implementation (Section 4).
(4) We discuss our algorithm and distinguish it from the existing work on the topic (Section 5).

In Section 2 we introduce the necessary background knowledge on abstract argumentation and labeling-based semantics and Section 6 concludes the paper. Proofs of technical results can be found in the Appendix.

2. Background

We consider *abstract argumentation frameworks* [15] defined as follows.

Definition 2.1. An *argumentation framework* (AF) is a pair $F = (\mathsf{Arg}, \mathsf{R})$ where Arg is a finite set of arguments and where $\mathsf{R} \subseteq \mathsf{Arg} \times \mathsf{Arg}$ is a relation of attack between arguments. We denote by $\mathcal{F}$ the set of all argumentation frameworks.

We say that an argument $a \in \mathsf{Arg}$ *attacks* an argument $b \in \mathsf{Arg}$ if $(a, b) \in \mathsf{R}$. Given a set $S \subseteq \mathsf{Arg}$ we say S *attacks* a if for some $b \in S$ we have $(b, a) \in \mathsf{R}$, and likewise a *attacks* S if for some $b \in S$ we have $(a, b) \in \mathsf{R}$.

A *labeling-based semantics* maps each argumentation framework to a set of *labelings* [8,17]. These are functions that assign to each argument a label indicating the acceptance status of the argument. The labeling-based semantics that we focus on in this paper use three labels: in indicates that the argument is accepted, out that the argument is rejected, and undec that the acceptance of the argument is undecided. In this paper we use the labeling-based approach since it allows us to distinguish not only accepted and non-accepted arguments, but also rejected arguments and undecided arguments. We will refer to a labeling-based semantics simply as a semantics.

Definition 2.2. A *labeling* of a set Arg of arguments is a total function $\ell : \mathsf{Arg} \to \{\mathsf{in}, \mathsf{out}, \mathsf{undec}\}$. We use $\mathbb{L}(\mathsf{Arg})$ to denote the set of labelings of Arg. A labeling of an argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$ is a labeling of Arg. Given a labeling ℓ we use $\mathsf{in}(\ell)$, $\mathsf{out}(\ell)$ and $\mathsf{undec}(\ell)$ to denote the set of arguments labeled in, out and undec, respectively.

A semantics σ is represented by a condition that determines whether a labeling ℓ of F is a σ-labeling. We now define the semantics that we focus on in this paper, namely the *complete*, *grounded*, *preferred* and *stable* semantics. While not typically considered to be a semantics, we also define the notion of a *conflict-free* and *admissible* labeling and treat these conditions as semantics. For that, we will use the definitions from [8] or definitions that are equivalent to those.

A labeling is conflict-free whenever no two arguments a and b exist such that a attacks b and both are labeled in.

Definition 2.3. A labeling ℓ of an argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$ is *conflict-free* if and only if:

(1) if a is labeled in then no attacker of a is labeled in.
(2) if a is labeled out then a has an attacker that is labeled in.

A labeling is admissible whenever it is conflict-free and, in addition, every argument a that is accepted is attacked only by arguments that are rejected. This amounts to requiring accepted arguments to be defended: in an admissible labeling, if a is accepted then all attackers of a are rejected and thus (as a consequence of Condition 2 in Definition 2.3) a is defended from these attackers by other arguments that are accepted.

Definition 2.4. A labeling ℓ of an argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$ is *admissible* if and only if:

(1) ℓ is conflict-free.
(2) if a is labeled in then all attackers of a are labeled out.

A labeling is complete whenever it is admissible and, in addition, (1) every argument attacked by an accepted argument is rejected, and (2) every argument whose attackers are rejected is accepted. This condition amounts to requiring defended arguments to be accepted: if all attackers of an argument a are attacked by some argument that is accepted, then all attackers of a are rejected and hence a is accepted.

Definition 2.5. A labeling ℓ of an argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$ is *complete* if and only if:

(1) ℓ is admissible.
(2) if some attacker of a is labeled in then a is labeled out.
(3) if all attackers of a are labeled out then a is labeled in.

The remaining semantics are all defined in terms of complete labelings. Firstly, a labeling is grounded if it is complete and if it is minimal with respect to accepted arguments. The grounded labeling of an argumentation framework is unique and represents the most skeptical point of view on which arguments to accept [8,15]. A preferred labeling is a complete labeling that is maximal with respect to accepted arguments. Preferred labelings represent maximally credulous points of view on which arguments to accept. Finally, a stable labeling is a complete labeling in which no argument is undecided.

Definition 2.6. Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and let ℓ be a labeling of F.

- ℓ is a grounded if it is complete and if there is no complete labeling ℓ' of F such that $\mathsf{in}(\ell') \subset \mathsf{in}(\ell)$.
- ℓ is a preferred if it is complete and if there is no complete labeling ℓ' of F such that $\mathsf{in}(\ell) \subset \mathsf{in}(\ell')$.
- ℓ is a stable if it is complete and if $\mathsf{undec}(\ell) = \emptyset$

In the following, we will denote with $\sigma(F)$ the set of all σ-labelings of F. The definitions in this section immediately imply the following relationships between the different semantics [8,15].

Proposition 2.1.

(1) *Every admissible labeling is conflict-free.*
(2) *Every complete labeling is admissible.*
(3) *Every grounded, preferred and stable labeling is complete.*
(4) *Every stable labeling is preferred.*

3. Learning from labelings

In this section, we introduce a novel algorithm for learning argumentation frameworks from labelings with the goal of constructing (or representing) *all* argumentation frameworks that are relevant to the input labelings. We start with formally defining the *AF learning* problem that we want to address. Afterwards, we define the concept of attack constraints and finally describe in detail our algorithm to solve this problem.

For the remainder of this work, we will consider the set of arguments Arg to be fixed. We also denote with $\mathbb{S}$ the set of all semantics. First, we define the notion of an *input*, which is a pair of the form (ℓ, σ) with a labeling ℓ and some semantics $\sigma \in \mathbb{S}$.

Definition 3.1. An *input pair* is a pair (ℓ, σ) where $\ell \in \mathbb{L}(\mathsf{Arg})$ is a labeling of Arg and $\sigma \in \mathbb{S}$ is a semantics.

We will also sometimes refer to ℓ as an input labeling. So, given a set of inputs $L = \{(\ell_1, \sigma_1), \ldots, (\ell_n, \sigma_n)\}$, our intention is to construct a set of argumentation frameworks $\mathbb{F}$, such that for all $F \in \mathbb{F}$ and each $i = 1, \ldots, n$, we have that ℓ_i is a σ_i-labeling of F. More formally, we are concerned with the following problem:

Problem:	AF learning
Input:	A set of inputs $L = \{(\ell_1, \sigma_1), \ldots, (\ell_n, \sigma_n)\}$
Output:	A set of argumentation frameworks $\mathbb{F} = \{F \mid \ell_i \text{ is a } \sigma_i\text{-labeling of } F, i = 1, \ldots, n\}$

The general idea behind our approach to solve the *AF learning* problem is as follows. We use propositional atoms of the form r_{ba} to express the existence of attacks $(b, a) \in \mathsf{R}$ in some argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$. Then, given an input (ℓ, σ), we construct a set of *attack constraints* $C_\ell = \{C_a \mid a \in \mathsf{Arg}\}$. For a labeling ℓ we will therefore have one attack constraint C_a for each argument $a \in \mathsf{Arg}$. These attack constraints depend on the semantics σ of the input as well as the labeling ℓ itself. Together, they express what an argumentation framework F has to satisfy such that ℓ is a σ-labeling of F. We will then use the fact that the models of these formulas directly correspond to argumentation frameworks by using the mapping functions described in Definition 3.3.

This approach can then also be extended to a set of input labelings $L = \{(\ell_1, \sigma_1), \ldots, (\ell_n, \sigma_n)\}$. So, for a given set of input labelings L we compute the constraints C_{ℓ_i} for each labeling ℓ_i, depending on its semantics σ_i. Then, each interpretation that satisfies all attack constraints of all input labelings represents an argumentation framework F where all labelings ℓ_i are σ_i-labelings of F. It follows, that the set of models that satisfy all attack constraints exactly corresponds to the set of argumentation frameworks $\mathbb{F}$ as specified in the above scenario description. In other words, we construct the set of all argumentation frameworks that have at least the given input as their labelings, but they can also have additional labelings.

3.1. Attack constraints

The key element of our algorithm are the so called *attack constraints*. Before we define the concept of attack constraints, we first introduce the propositional language that we use to represent attacks in argumentation frameworks.

Let At be a set of propositional atoms. With $\mathcal{L}(\mathsf{At})$ we denote the corresponding propositional language closed under the usual connectives. Furthermore, we will use the concept of an *interpretation* $\mathcal{A}$, a function that assigns to each atom a truth value. If an interpretation $\mathcal{A}$ satisfies a formula $f \in \mathcal{L}(\mathsf{At})$, i.e., it assigns the value *true* to it, we may also call $\mathcal{A}$ a *model* of f. With $\mathcal{M}(f)$ we denote the set of all models of f. If G is a set of formulas, we denote with $\mathcal{M}(G)$ the set of all models of the conjunction $g = \bigwedge_{f \in G} f$.

Definition 3.2. Let Arg be the fixed set of arguments and $a \in \mathsf{Arg}$ is an argument. Then we define the set of atoms representing all possible attacks in the argumentation framework F as follows:

$$attackAtoms(\mathsf{Arg}) = \{r_{ba} \mid a, b \in \mathsf{Arg}\}$$

and the set of atoms that represent all possible incoming attacks on the argument a as

$$inAttacks(a) = \{r_{ba} \mid b \in \mathsf{Arg}\}.$$

A positive occurrence r_{ba} represents an attack from b to a in the argumentation framework, i.e., $(b, a) \in \mathsf{R}$. On the other hand, a negative occurrence $\neg r_{ba}$ means there should be no attack from b on a in the argumentation framework, i.e., $(b, a) \notin \mathsf{R}$.

Due to the correspondence between the above defined atoms $attackAtoms(\mathsf{Arg})$ and the attacks of some argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$, we can also define a one-to-one mapping between the models of a formula and attack relations.

Definition 3.3. Arg is the set of arguments and $f \in \mathcal{L}(attackAtoms(\mathsf{Arg}))$ is a formula. We define the mapping from a model to an attack relation via the function $modelToAttacks : \mathcal{M}(f) \to 2^{\mathsf{Arg} \times \mathsf{Arg}}$ as:

$$modelToAttacks(\mathcal{A}) := \mathsf{R} \quad \text{with } \mathsf{R} = \left\{(a, b) | \mathcal{A}(r_{ab}) = true\right\}$$

and the mapping from an attack relation to a model $attacksToModel : 2^{\mathsf{Arg} \times \mathsf{Arg}} \to \mathcal{M}(f)$ as:

$$attacksToModel(\mathsf{R}) := \mathcal{A} \quad \text{with } \mathcal{A}(r_{ab}) = \begin{cases} true & \text{if } (a, b) \in \mathsf{R} \\ false & \text{otherwise.} \end{cases}$$

Together with the fixed set of arguments Arg, we can then also recover an argumentation framework $F = (\mathsf{Arg}, modelToAttacks(\mathcal{A}))$.

Example 1. Consider the set of arguments $\mathsf{Arg} = \{a, b, c, d\}$. Then, an attack constraint for the argument a could for instance look like this

$$C_a = \neg r_{aa} \wedge (r_{ca} \vee r_{da}).$$

For an argumentation framework, this constraint enforces that the argument a cannot be attacked by itself. Furthermore, a must be attacked by either c or d or both. Finally, an attack from b on a might also exist, since the constraint C_a does not impose any restriction on the variable r_{ba}. That means, there exist multiple different argumentation frameworks that satisfy the above attack constraint for a. These argumentation frameworks are represented by the set of models $\mathcal{M}(C_a)$.

Recalling Section 2, under any semantics an argument can only be accepted if it is not attacked by any other accepted argument (conflict-freeness). In addition to that, the acceptance of an argument also depends on the label of its attackers. If the attackers of the argument a are labeled out, then a can still be accepted (i.e., labeled in). On the other hand, if one of its attackers is labeled undec (or in), then a cannot be accepted under admissible, complete, grounded, preferred or stable semantics. It does not matter in this context what the outgoing attacks of the argument a are. To summarize, if we want to formulate a constraint for the acceptance of an argument, under most semantics we only need to consider its direct attackers and their label in a given labeling.

For that reason, we define the attack constraints from the local perspective of an argument.

Definition 3.4. Let $a \in \mathsf{Arg}$ be an argument. Then the *attack constraint* of a is a formula $C_a \in \mathcal{L}(inAttacks(a))$.

So, instead of specifying one constraint per input labeling, we will represent a labeling ℓ with a set of attack constraints. For this labeling constraint every model corresponds to an argumentation framework that has ℓ as a σ-labeling.

Definition 3.5. Let Arg be the fixed set of arguments and (ℓ, σ) is an input. Then we define the *labeling constraint* C_ℓ of the input labeling ℓ as a set of attack constraints as follows:

$$C_\ell = \{C_a \mid a \in \mathsf{Arg}\},$$

such that we have

$$\mathcal{A} \in \mathcal{M}(C_\ell) \implies \ell \text{ is a } \sigma\text{-labeling of } F = \big(\mathsf{Arg}, \mathit{modelToAttacks}(\mathcal{A})\big).$$

It is also worth noting that by defining the attack constraints as formulas over *inAttacks*(a), we ensure that the constraints for a labeling are mutually independent.

Proposition 3.1. *Let (ℓ, σ) be an input and C_ℓ is the labeling constraint of ℓ given σ. The attack constraints $C_a \in C_\ell$ are mutually independent, i.e., for any two attack constraints $C_a, C_b \in C_\ell$, we have that atoms$(C_a) \cap$ atoms$(C_b) = \emptyset$.*

Finally, we may also say that an argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$ satisfies a labeling constraint C_ℓ, if there exists a model $\mathcal{A}$ of C_ℓ that can be mapped to the attack relation R of F.

Definition 3.6. Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. Let (ℓ, σ) be an input and C_ℓ is the labeling constraint of ℓ. We say that F *satisfies* C_ℓ, written as $F \models C_\ell$, if and only if there is a model $\mathcal{A}$ of C_ℓ such that *attacksToModel*$(\mathsf{R}) = \mathcal{A}$.

3.2. Semantic constraint functions

To construct the constraints for an input (ℓ, σ), we need a method that given an argument a and a labeling ℓ computes a valid attack constraint for each argument. Under different semantics, we have different conditions under which arguments are accepted or rejected. Therefore, such a function should be different depending on the associated semantics σ of the input. To compute these constraints for each argument, we will use a *semantic constraint function $AttCon_\sigma$* with the following signature:

$$\mathit{AttCon}_\sigma : \mathsf{Arg} \times \mathbb{L}(\mathsf{Arg}) \longrightarrow \mathcal{L}\big(\mathit{inAttacks}(a)\big).$$

This function then takes the input labeling ℓ and an argument $a \in \mathsf{Arg}$ and returns the attack constraint for this argument. In other words, it computes a formula that represents the information about the acceptance of a that can be extracted from the labeling ℓ under the assumption that ℓ is a σ-labeling.

Recall that in our scenario we want to compute *all* argumentation frameworks F such that for all inputs (ℓ_i, σ_i) we have that ℓ_i is a σ_i-labeling of F. In order to satisfy this requirement, we have to ensure soundness and completeness in the following sense:

The set of arguments Arg is fixed. Let (ℓ, σ) be an input and AttCon_σ is a semantic constraint function for σ. The corresponding labeling constraint $C_{\ell,\sigma}$ for ℓ is then the set of attack constraints

$$C_{\ell,\sigma} = \big\{\mathit{AttCon}_\sigma(a, \ell) \mid a \in \mathsf{Arg}\big\}.$$

Whenever the semantics σ is not relevant or follows from context, we denote the labeling constraint for the labeling ℓ with just C_ℓ.

Then, the following two conditions must hold for all labelings:

(1) The semantic constraint function is *sound* for all labelings ℓ, i.e.,

$$\forall \mathcal{A} \in \mathcal{M}(C_{\ell,\sigma}) : \ell \text{ is a } \sigma\text{-labeling of } F = \big(\mathsf{Arg},\ \mathit{modelToAttacks}(\mathcal{A})\big).$$

(2) The semantic constraint function is *complete* for all labelings ℓ, i.e.,

$$\forall F = (\mathsf{Arg}, \mathsf{R}) : \ell \text{ is a } \sigma\text{-labeling of } F \implies \exists \mathcal{A} \in \mathcal{M}(C_{\ell,\sigma}) : \mathsf{R} = \mathit{modelToAttacks}(\mathcal{A}).$$

So, if there exists a semantic constraint function AttCon_σ for a semantics σ that satisfies the above two conditions, then we can use this function to learn argumentation frameworks as specified in the *AF learning* problem. On the other hand, if no such function exists, then labelings with respect to the semantics σ cannot be used to learn argumentation frameworks.

In the following, we will define the semantic constraint function for conflict-free, admissible, complete and stable semantics and give some examples to illustrate what kind of restrictions the constraints of each semantics impose on an argumentation framework.

3.2.1. Conflict-free semantics

First, we recall the conditions that a labeling ℓ has to satisfy in order to be considered conflict-free:

(1) if a is labeled in then no attacker of a is labeled in.
(2) if a is labeled out then a has an attacker that is labeled in.

From this we can derive what an attack constraint for a conflict-free input (ℓ, cf) should include. If an argument $a \in \mathsf{Arg}$ is labeled in or undec, then it cannot be attacked by any argument $b \in \mathsf{in}(\ell)$. The first case follows from the definition of conflict-freeness, as there cannot be any attacks between in-labeled arguments (see condition (1) from above). The case undec follows from the definition of labelings itself, i.e., if an argument is attacked by an in-labeled argument, then it would have to be labeled out (see condition (2)). Furthermore, if the argument is labeled out, then it has to be attacked by at least one argument $b \in \mathsf{in}(\ell)$.

In addition to that, every argument a can be attacked by any other argument c, that does not have the label in, without influencing the label of a. These attacks, which are not relevant in order to have ℓ as a σ-labeling are not contained in the attack constraint of a.

Definition 3.7. Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and (ℓ, cf) be an input. We then define the semantic constraint function for the conflict-free semantics as follows:

$$\mathit{AttCon}_{\mathit{cf}}(a, \ell) = \begin{cases} \bigwedge_{b \in \mathsf{in}(\ell)} \neg r_{ba} & \text{if } a \in \mathsf{in}(\ell) \\ \bigvee_{b \in \mathsf{in}(\ell)} r_{ba} & \text{if } a \in \mathsf{out}(\ell) \\ \bigwedge_{b \in \mathsf{in}(\ell)} \neg r_{ba} & \text{if } a \in \mathsf{undec}(\ell) \end{cases}$$

The attack constraints computed via the semantic constraint function $\mathit{AttCon}_{\mathit{cf}}$ are sound and complete for conflict-free input labelings in the above defined sense.

Theorem 3.1. *Let (ℓ, cf) be a conflict-free input and C_ℓ is the labeling constraint of ℓ computed via $AttCon_{cf}$. With $\mathbb{F}$ we denote the set of argumentation frameworks that satisfy C_ℓ, i.e for all $F \in \mathbb{F}$ we have that $F \models C_\ell$.*

The semantic constraint function is sound *for conflict-free input labelings ℓ, i.e.,*

$$\forall F \in \mathbb{F} : \ell \text{ is a } \mathsf{cf}\text{-labeling of } F$$

and the semantic constraint function is complete *for conflict-free input labelings ℓ, i.e.,*

$$\forall F = (\mathsf{Arg}, \mathsf{R}) : \ell \text{ is a } \mathsf{cf}\text{-labeling of } F \implies F \in \mathbb{F}.$$

Example 2. Consider the set of arguments $\mathsf{Arg} = \{a, b, c, d\}$ and the conflict free input labeling $\ell = \{\mathsf{in} = \{a, b\}, \mathsf{out} = \{c\}, \mathsf{undec} = \{d\}\}$ over these arguments. We can then use the above defined function $AttCon_{cf}(a, \ell)$ to compute the attack constraint of each argument in Arg:

$$C_a = AttCon_{cf}(a, \ell) = \neg r_{aa} \wedge \neg r_{ba}$$
$$C_b = AttCon_{cf}(b, \ell) = \neg r_{ab} \wedge \neg r_{bb}$$
$$C_c = AttCon_{cf}(c, \ell) = r_{ac} \vee r_{bc}$$
$$C_d = AttCon_{cf}(d, \ell) = \neg r_{ad} \wedge \neg r_{bd}$$

These conditions represent the input labeling ℓ. They capture that the in-labeled arguments a and b are conflict-free, i.e., there is no attack between them in any direction. In addition to that, the undec-labeled argument d is also not attacked by a and b, because that would mean d would have to be labeled out. The argument c is labeled out and thus it must be attacked by either a or b in order to satisfy the corresponding attack constraint.

We may also notice that the attack constraints contain no restrictions on attacks originating from c or d. That means these attacks are considered possible and we have to consider any combination of those when constructing the argumentation frameworks consistent with the conditions later.

3.2.2. Admissible semantics

Recalling the definition from Section 2, a labeling ℓ is considered admissible if (1) it is a cf-labeling and (2) we have that if an argument $a \in \mathsf{Arg}$ is labeled in then all attackers of a are labeled out.

As the above condition (1) suggests, the attack constraint for an admissible input (ℓ, ad) are a strengthened version of the conflict-free constraints. The attack constraint for arguments that are labeled out is the same as in the conflict-free case, it has to be attacked by at least one argument $b \in \mathsf{in}(\ell)$. Arguments that are labeled undec cannot be attacked by arguments that have the label in, just like it was the case for conflict-free labelings. From condition (2) it follows that if the argument is labeled in the attack constraint now includes that it cannot be attacked by any argument which is labeled in or undec. Disallowing attacks from in-labeled arguments captures the conflict-freeness and is the same as before. Additionally an argument which is accepted under admissible semantics can no longer be attacked by undec-labeled argument. This captures the defense property of admissibility. An argument with the label undec is per definition not attacked by any accepted argument and thus any attack from such an argument would be undefended.

However, an in-labeled argument can be attacked by any argument with the label out since these arguments are per definition attacked by some in-labeled argument. Furthermore, arguments that are labeled out or undec can additionally be attacked by any argument that does not have the label in, just like before.

Definition 3.8. Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and (ℓ, ad) be an input. We then define the semantic constraint function for the admissible semantics as follows:

$$AttCon_{ad}(a, \ell) = \begin{cases} \bigwedge_{b \in \mathsf{Arg} \setminus \mathsf{out}(\ell)} \neg r_{ba} & \text{if } a \in \mathsf{in}(\ell) \\ \bigvee_{b \in \mathsf{in}(\ell)} r_{ba} & \text{if } a \in \mathsf{out}(\ell) \\ \bigwedge_{b \in \mathsf{in}(\ell)} \neg r_{ba} & \text{if } a \in \mathsf{undec}(\ell) \end{cases}$$

The attack constraints computed via the semantic constraint function $AttCon_{ad}$ are sound and complete for admissible input labelings.

Theorem 3.2. *Let (ℓ, ad) be an admissible input and C_ℓ is the labeling constraint of ℓ computed via $AttCon_{ad}$. With $\mathbb{F}$ we denote the set of argumentation frameworks that satisfy C_ℓ, i.e for all $F \in \mathbb{F}$ we have that $F \models C_\ell$.*

The semantic constraint function $AttCon_{ad}$ is sound *for admissible input labelings ℓ, i.e.,*

$$\forall F \in \mathbb{F} : \ell \textit{ is an } \text{ad}\textit{-labeling of } F$$

and the semantic constraint function $AttCon_{ad}$ is complete *for admissible input labelings ℓ, i.e.,*

$$\forall F = (\mathsf{Arg}, \mathsf{R}) : \ell \textit{ is an } \text{ad}\textit{-labeling of } F \implies F \in \mathbb{F}.$$

Example 3. Let us consider the set of arguments $\mathsf{Arg} = \{a, b, c, d\}$ and the admissible input labeling $\ell = \{\mathsf{in} = \{a\}, \mathsf{out} = \{c\}, \mathsf{undec} = \{b, d\}\}$ over these arguments. We can then use the above defined function $AttCon_{ad}(a, \ell)$ to compute the attack constraint of each argument in Arg:

$$C_a = AttCon_{ad}(a, \ell) = \neg r_{aa} \wedge \neg r_{ba} \wedge \neg r_{da}$$
$$C_b = AttCon_{ad}(b, \ell) = \neg r_{ab}$$
$$C_c = AttCon_{ad}(c, \ell) = r_{ac}$$
$$C_d = AttCon_{ad}(d, \ell) = \neg r_{ad}$$

These conditions represent the input labeling ℓ and impose some restrictions on any argumentation framework that wants to be consistent with the labeling. For the only in-labeled argument a we have that it must not be attacked by itself, or any of the undec-labeled arguments b an d. Otherwise a would not be defended and thus not admissible. Like in the case of conflict-free labelings, the undec-labeled arguments b and d cannot be attacked by the in-labeled argument a. Finally, the argument c with the label out has to be attacked by a, since that is the only in-labeled argument in this example.

The attacks represented by all atoms $r \in attackAtoms(\mathsf{Arg})$ which are not mentioned in the attack constraints are again considered to be possible but not necessary.

3.2.3. Complete semantics

We recall that a labeling ℓ is considered complete if it satisfies the following three conditions:

(1) ℓ is admissible.
(2) if some attacker of a is labeled in then a is labeled out.
(3) if all attackers of a are labeled out then a is labeled in.

The attack constraints for a complete input (ℓ, co) are therefore a stronger version of the constraints for admissible labelings. The attack constraints for in-labeled arguments is the same as before, capturing both conflict-freeness and defense. The constraint for arguments with the label out are also identical and just require some attack from any in-labeled argument. However, the attack constraint for undec-labeled arguments is different for complete labelings and now consists of two parts. The first part states that an undec-labeled argument a cannot be attacked by an in-labeled argument and is the same as before. But in addition to that the argument a has to be attacked by some undec-labeled argument in order to be labeled undec. This can be an attack from some other argument or even the argument a itself. This additional constraint is necessary to capture the completeness property: Every argument that is defended by $\mathsf{in}(\ell)$ has to be included in $\mathsf{in}(\ell)$. Assuming that there is no attack from another undec-labeled argument, then the argument a can only be attacked by arguments with the label out. But, in that case the argument would be defended by $\mathsf{in}(\ell)$ and thus should be labeled in instead of undec.

Definition 3.9. Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and (ℓ, co) be an input. We then define the semantic constraint function for the complete semantics as follows:

$$AttCon_{co}(a, \ell) = \begin{cases} \bigwedge_{b \in \mathsf{Arg} \setminus \mathsf{out}(\ell)} \neg r_{ba} & \text{if } a \in \mathsf{in}(\ell) \\ \bigvee_{b \in \mathsf{in}(\ell)} r_{ba} & \text{if } a \in \mathsf{out}(\ell) \\ \bigwedge_{b \in \mathsf{in}(\ell)} \neg r_{ba} \wedge (\bigvee_{c \in \mathsf{undec}(\ell)} r_{ca}) & \text{if } a \in \mathsf{undec}(\ell) \end{cases}$$

The attack constraints computed via the semantic constraint function $AttCon_{co}$ are sound and complete for complete input labelings.

Theorem 3.3. *Let (ℓ, co) be a complete input and C_ℓ is the labeling constraint of ℓ computed via $AttCon_{co}$. With $\mathbb{F}$ we denote the set of argumentation frameworks that satisfy C_ℓ, i.e for all $F \in \mathbb{F}$ we have that $F \models C_\ell$.*

The semantic constraint function $AttCon_{co}$ is sound *for complete input labelings ℓ, i.e.,*

$$\forall F \in \mathbb{F} : \ell \textit{ is a } \text{co}\textit{-labeling of } F$$

and the semantic constraint function $AttCon_{co}$ is complete *for complete input labelings ℓ, i.e.,*

$$\forall F = (\mathsf{Arg}, \mathsf{R}) : \ell \textit{ is a } \text{co}\textit{-labeling of } F \implies F \in \mathbb{F}.$$

Example 4. Let us consider the set of arguments $\mathsf{Arg} = \{a, b, c, d\}$ and the complete input labeling $\ell = \{\mathsf{in} = \{c\}, \mathsf{out} = \{a\}, \mathsf{undec} = \{b, d\}\}$ over these arguments. We can then use the above defined function $AttCon_{co}(a, \ell)$ to compute the attack constraint of each argument in Arg:

$$C_a = AttCon_{co}(a, \ell) = r_{ca}$$

$$C_b = AttCon_{co}(b, \ell) = \neg r_{cb} \wedge (r_{bb} \vee r_{db})$$
$$C_c = AttCon_{co}(c, \ell) = \neg r_{bc} \wedge \neg r_{cc} \wedge \neg r_{dc}$$
$$C_d = AttCon_{co}(d, \ell) = \neg r_{cd} \wedge (r_{bd} \vee r_{dd})$$

Like before, the only out-labeled argument a must be attacked by the only in-labeled argument c. The argument c with the label in can then not be attacked by itself or any of the undec-labeled arguments b and d. Like in the example for admissible conditions, the arguments b and d are undecided. That means by definition they cannot be attacked by the in-labeled argument c. However, since the input labeling is complete in addition to that they have to be attacked by any undec-labeled argument.

We have again a few other possible attacks in this example, for instance the argument a can additionally be attacked by itself, b or d while still satisfying the attack constraints.

3.2.4. Stable semantics

A labeling ℓ is considered stable if it is a *co*-labeling and we have that $\mathsf{undec}(\ell) = \emptyset$. So, if we have a stable input (ℓ, st) we do not have any arguments in $\mathsf{undec}(\ell)$ since every argument of the argumentation framework is either in the corresponding stable extension or attacked by it. The constraints for in and out-labeled arguments are the same as in the conflict-free case. An argument that is labeled in cannot be attacked by any argument that is also labeled in. We longer need to include the defense part of the admissible or complete attack constraints, since there exists no undec-labeled argument per definition. If an argument is labeled out it has to be attacked by at least one argument $b \in \mathsf{in}(\ell)$.

Definition 3.10. Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and (ℓ, st) be an input. We then define the semantic constraint function for the stable semantics as follows:

$$AttCon_{st}(a, \ell) = \begin{cases} \bigwedge_{b \in \mathsf{in}(\ell)} \neg r_{ba} & \text{if } a \in \mathsf{in}(\ell) \\ \bigvee_{b \in \mathsf{in}(\ell)} r_{ba} & \text{if } a \in \mathsf{out}(\ell) \end{cases}$$

The attack constraints computed via the semantic constraint function $AttCon_{st}$ are sound and complete for stable input labelings.

Theorem 3.4. *Let (ℓ, st) be a stable input and C_ℓ is the labeling constraint of ℓ computed via $AttCon_{st}$. With $\mathbb{F}$ we denote the set of argumentation frameworks that satisfy C_ℓ, i.e for all $F \in \mathbb{F}$ we have that $F \models C_\ell$.*

The semantic constraint function $AttCon_{st}$ is sound *for stable input labelings ℓ, i.e.,*

$$\forall F \in \mathbb{F} : \ell \text{ is a st-labeling of } F$$

and the semantic constraint function $AttCon_{st}$ is complete *for stable input labelings ℓ, i.e.,*

$$\forall F = (\mathsf{Arg}, \mathsf{R}) : \ell \text{ is a st-labeling of } F \implies F \in \mathbb{F}.$$

3.2.5. Other semantics

Besides the already mentioned semantics, Dung also introduced the grounded and preferred semantics in his seminal paper [15]. However, for both of these semantics, there exists no semantic constraint function which is sound and complete. The reason for that lies in the minimality and maximality constraints w.r.t. set inclusion for these semantics. The semantic constraint functions are supposed to return local constraints, i.e., a formula in $\mathcal{L}(inAttacks(a))$. Clearly, such a local formula is not able to express minimality/maximality among multiple labelings.

Proposition 3.2. *There exists no sound and complete semantic constraint function for the grounded semantics.*

Proof. Assume the contrary. Let $AttCon_{gr}$ be the semantic constraint function that is sound and complete for gr. Consider the labeling $\ell = \{\mathsf{in} = \{a, c\}, \mathsf{out} = \{b\}, \mathsf{undec} = \emptyset\}$. One AF which has ℓ as its grounded labeling is the AF $(\{a, b, c\}, \{(a, b)\})$. If we add the attack (b, a) then ℓ is no longer the grounded labeling of this AF. Soundness then implies that we must have $AttCon_{gr}(a, \ell) \models \neg r_{ba}$. However, another AF which has ℓ as its grounded labeling is the AF $(\{a, b, c\}, \{(a, b), (c, b)\})$. This time, if we add the attack (b, a), then ℓ is still the grounded labeling of this AF. Completeness then implies that we have $AttCon_{gr}(a, \ell) \not\models \neg r_{ba}$. This is a contradiction, and hence there is no semantic constraint function that is sound and complete for the grounded semantics. □

Proposition 3.3. *There exists no sound and complete semantic constraint function for the preferred semantics.*

Proof. Assume the contrary. Let $AttCon_{pr} : \mathsf{Arg} \times \mathbb{L}(\mathsf{Arg}) \longrightarrow \mathcal{L}(inAttacks(a))$ be the semantic constraint function that is sound and complete for pr. Consider the labeling $\ell = \{\mathsf{in} = \emptyset, \mathsf{out} = \emptyset, \mathsf{undec} = \{a, b, c\}\}$. One AF which has ℓ as a preferred labeling is the AF $(\{a, b, c\}, \{(a, a), (a, b), (c, c)\})$. If we add the attack (b, a) then ℓ is no longer a preferred labeling of this AF. Soundness then implies that we must have $AttCon_{pr}(a, \ell) \models \neg r_{ba}$. However, another AF which has ℓ as a preferred labeling is the AF $(\{a, b, c\}, \{(a, a), (a, b), (c, c), (c, b)\})$. This time, if we add the attack (b, a), then ℓ is still a preferred labeling of this AF. Completeness then implies that we have $AttCon_{pr}(a, \ell) \not\models \neg r_{ba}$. This is a contradiction, and hence there is no semantic constraint function that is sound and complete for the preferred semantics. □

3.3. Algorithm

In the following we will introduce an algorithm for learning argumentation frameworks from labelings. The input of this algorithm is a set of arguments Arg and a set of inputs $L = \{(\ell_1, \sigma_1), \dots\}$. Our algorithm works in an iterative way. That means, each input labeling is processed independently one by one. So, the algorithm is able to consider a stream of input labelings and provide intermediate results after each processed labeling. The procedure for the algorithm is shown in Algorithm 1. In each iteration, the input parameters of the algorithm include a mapping C of the arguments to their current attack constraint. Before processing the first input labeling we assign to each argument a the initial attack constraint $C_a = \top$. So, by default no attacks are necessary to satisfy the constraint, which means all argumentation frameworks $F = (\mathsf{Arg}, \mathsf{R})$ are consistent with the empty set of conditions, as would be expected.

The main procedure of the algorithm can essentially be split into three parts:

(1) Computing the labeling constraint C_ℓ for the input labeling.
(2) Combining the labeling constraint C_ℓ with the input constraints C.
(3) Constructing the set of argumentation frameworks $\mathbb{F}$ from the constraints in C.

Algorithm 1 Iterative algorithm for learning argumentation frameworks from labelings

```
1: input mapping of constraints $C = \{a \rightarrow C_a \mid a \in \mathsf{Arg}\}$, set of arguments $\mathsf{Arg}$, an input $(\ell, \sigma)$.
2: output updated mapping of constraints $C$, set of argumentation frameworks $\mathbb{F}$ (optional)
3:
4: for each argument $a \in \mathsf{Arg}$ do
5:     $C_{a,\ell} \leftarrow AttCon_\sigma(a, \ell)$
6:     $C_a \leftarrow C_a \wedge C_{a,\ell}$
7: end for
8: $\mathbb{F} \leftarrow constructFrameworks(C)$    (optional)
9: return $C$, $(\mathbb{F})$.
```

In each iteration of the algorithm we take a mapping of constraints C, the set of arguments Arg and some input $(\ell, \sigma) \in L$ as input. When we reference the mapping as C_a we mean the constraint for argument a as stored in the mapping C.

For the input we compute the attack constraint $C_{a,\ell}$ of each argument $a \in \mathsf{Arg}$ via the semantic constraint function $AttCon_\sigma$.

In the second step, we combine the labeling constraint $C_{a,\ell}$ for each argument a with the existing attack constraint C_a. We want to compute argumentation frameworks that satisfy all input labelings, therefore the combined constraint for an argument a is just the logical conjunction of the constraints of the argument for each labeling. The updated constraint overwrites the constraint for the argument in the mapping C.

The third and final step is to construct the set of argumentation frameworks $\mathbb{F}$ for the updated attack constraint mapping C. For this, we use the mapping *modelToAttacks*$(\mathcal{A})$ described in Definition 3.3. In particular, we compute the set of models $\mathcal{M}_a$ for each argument $a \in \mathsf{Arg}$. It should be noted that the model computation for these constraints are independent of each other and thus can be done in parallel. If we then pick one model $\mathcal{A}_i \in \mathcal{M}_{a_i}$ for each argument, we obtain a consistent argumentation framework $F = (\mathsf{Arg}, modelToAttacks(\mathcal{A}_1 \cup \cdots \cup \mathcal{A}_n))$. If we consider all different combinations of models we obtain the set of all consistent argumentation frameworks $\mathbb{F}$.

This step is optional, since we only need the mapping of attack constraints C with the updated constraints for each argument for further learning. So, after each learning iteration the algorithm returns the updated set of attack constraints C and (if wanted) the set of constructed argumentation frameworks $\mathbb{F}$.

Regarding the complexity of constructing argumentation frameworks for a set of attack constraints. We can consider the constraints in clausal form, which follows easily from the definition of the semantic constraint functions. A very helpful property of the attack constraints in clausal form is, that all clauses contain at most one negated literal, i.e., each clause is a dual-horn clause. That means, the process of constructing an argumentation framework for a given set of attack constraints can be considered as a *Dual-Horn SAT* problem, which is **P**-complete [14]. Thus, the construction argumentation frameworks can be done in polynomial time, more precisely in linear time with respect to the size of the attack constraints which is in $O(n^2|L|)$, where n is the number of arguments and L is the set of learned labelings. This can be further improved by using parallel computations, as shown by our experiments in Section 4.

3.3.1. Properties

In the following we will look at some useful properties that the previously defined algorithm satisfies, namely soundness, completeness and monotonicity.

Theorem 3.5. *Let L be a set of inputs and $\mathbb{F}$ is the set of argumentation frameworks constructed by the procedure described in Algorithm 1 by iteratively processing L. For every σ_i in the set of inputs there exists a sound and complete semantic constraint function, i.e., $\sigma_i \in \{cf, ad, co, st\}$.*

The algorithm is sound *for all input labelings, i.e.,*

$$\forall F \in \mathbb{F} : \forall(\ell_i, \sigma_i) \in L : \ell \text{ is a } \sigma_i\text{-labeling of } F$$

The algorithm is complete *for all input labelings, i.e.,*

$$\forall G = (\mathsf{Arg}, \mathsf{R}) : \big(\forall(\ell_i, \sigma_i) \in L : \ell \text{ is a } \sigma_i\text{-labeling of } G\big) \implies G \in \mathbb{F}$$

As shown in Theorem 3.5 the algorithm based on the above defined attack constraints is sound and complete for any combination of inputs with respect to all four semantics. That means, the algorithm computes exactly the set of argumentation frameworks that are consistent with the input labelings. From the proof of Theorem 3.5 it also follows that the order in which the labelings are processed by the algorithm does not matter.

Proposition 3.4. *Let* Arg *be a set of arguments and ℓ_1, ℓ_2 are labelings. With $\mathbb{F}$ we denote the set of argumentation frameworks constructed by Algorithm 1 for the input labelings ℓ_1 and ℓ_2.*

Then, the order in which the labelings are learned has no influence on the constructed set of argumentation frameworks $\mathbb{F}$.

Another property of the algorithm is monotonicity (see Theorem 3.6), or more precisely the algorithm is monotonically refining. That means, if we have a set of labelings L_2 with $L_2 \supseteq L_1$ then the set of argumentation frameworks $\mathbb{F}_2$ constructed for L_2 will be a subset of the frameworks constructed for L_1. In other words, if we learn an additional labeling the number of argumentation frameworks that satisfy all attack constraints can only stay the same or become smaller.

Theorem 3.6. *Let* Arg *be a set of arguments and L_1, L_2 are sets of inputs. $\mathbb{F}_1$, $\mathbb{F}_2$ denote the sets of argumentation frameworks constructed by Algorithm 1 for the input labelings in L_1 and L_2 respectively.*

The algorithm for constructing argumentation frameworks from labelings is monotonically refining, *i.e.,*

$$\forall L_1, L_2 \in \mathbb{L}(\mathsf{Arg}) : L_1 \supseteq L_2 \implies \mathbb{F}_1 \subseteq \mathbb{F}_2$$

3.4. Example

This section will showcase the application of Algorithm 1. Assume the following situation. Unknown to us, there is the hidden argumentation framework as shown in Fig. 2. Given are only the set of arguments $\mathsf{Arg} = \{a, b, c, d, e\}$ and the set of inputs $L = \{(\ell_1, ad), (\ell_2, co), (\ell_3, cf)\}$. The three input labelings are shown in Fig. 3. Our goal is to construct all argumentation frameworks that are consistent with these three labelings.

We start the process of learning with the admissible labeling $\ell_1 = \{\mathsf{in} = \{a\}, \mathsf{out} = \{b, c\}, \mathsf{undec} = \{d, e\}\}$. The first step is computing the labeling-specific attack constraint for each argument. The labeling ℓ_1 is admissible, so we will use the semantic constraint function $AttCon_{ad}$ from Definition 3.8. The argument a is labeled in, that means it cannot be attacked by any in- or undec-labeled argument. b and c

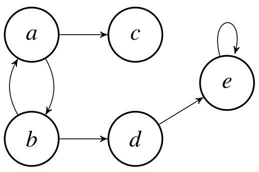
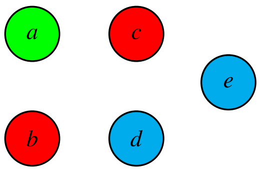
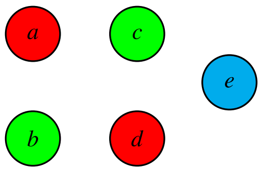
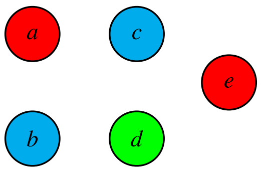


Fig. 2. The hidden argumentation framework from which the input labelings are generated.





$\ell_1 = \{\mathsf{in} = \{a\}, \mathsf{out} = \{b, c\}, \mathsf{undec} = \{d, e\}\}$ $\ell_2 = \{\mathsf{in} = \{b, c\}, \mathsf{out} = \{a, d\}, \mathsf{undec} = \{e\}\}$



$\ell_3 = \{\mathsf{in} = \{d\}, \mathsf{out} = \{e\}, \mathsf{undec} = \{a, b, c\}\}$

Fig. 3. The three input labelings: ℓ_1 is admissible, ℓ_2 is complete and ℓ_3 is conflict-free.

are labeled out, thus they have to be attacked by the only in-labeled argument a. The arguments d and e are undec which means they cannot be attacked by the in-labeled argument a. That leaves us with the following attack constraints after the first labeling:

$$C_a = \neg r_{aa} \wedge \neg r_{da} \wedge \neg r_{ea}$$
$$C_b = r_{ab}$$
$$C_c = r_{ac}$$
$$C_d = \neg r_{ad}$$
$$C_e = \neg r_{ae}$$

After processing the first labeling we know that the argument a must attack b and c. We also know that some attacks cannot exist, but there are still many possibilities for attacks. For example, the argument a can still be attacked by b or c and the resulting frameworks would still be consistent with ℓ_1. There can also be many different combinations of attacks between the arguments b, c, d and e. This iteration of the algorithm would then return the set of attack constraints $C = \{C_a, C_b, C_c, C_d, C_e\}$.

Lets consider the next labeling $\ell_2 = \{\mathsf{in} = \{b, c\}, \mathsf{out} = \{a, d\}, \mathsf{undec} = \{e\}\}$, which is complete. For this labeling we will use the semantic constraint function AttCon_{co} (see Definition 3.9). This labeling tells us, for example, that the out-labeled arguments a and d must be attacked by either b or c. The in-labeled arguments b and c cannot be attacked by any in- or undec-labeled arguments. The argument e is the only undec-labeled argument and since ℓ_2 is complete it follows that e has to attack itself. This gives us the following set of attack constraints C_{ℓ_2} for the labeling ℓ_2:

$$C_{a,\ell_2} = r_{ba} \vee r_{ca}$$

$$C_{b,\ell_2} = \neg r_{bb} \wedge \neg r_{cb} \wedge \neg r_{eb}$$

$$C_{c,\ell_2} = \neg r_{bc} \wedge \neg r_{cc} \wedge \neg r_{ec}$$

$$C_{d,\ell_2} = r_{bd} \vee r_{cd}$$

$$C_{e,\ell_2} = \neg r_{be} \wedge \neg r_{ce} \wedge r_{ee}$$

We now have to combine the constraints C from the last step with the contraints C_{ℓ_2} from this step. Since the argumentation frameworks we are looking for should be consistent with all input labelings we use a conjunction to combine the attack constraints. So, for each argument we set $C_a = C_a \wedge C_{a,\ell_2}$. If we do this for all arguments, we obtain the following attack constraints:

$$C_a = \neg r_{aa} \wedge \neg r_{da} \wedge \neg r_{ea} \wedge (r_{ba} \vee r_{ca})$$

$$C_b = r_{ab} \wedge \neg r_{bb} \wedge \neg r_{cb} \wedge \neg r_{eb}$$

$$C_c = r_{ac} \wedge \neg r_{bc} \wedge \neg r_{cc} \wedge \neg r_{ec}$$

$$C_d = \neg r_{ad} \wedge (r_{bd} \vee r_{cd})$$

$$C_e = \neg r_{ae} \wedge \neg r_{be} \wedge \neg r_{ce} \wedge r_{ee}$$

We have now reduced the number of possibilities even further. The argument a has to be attacked by either b or c and cannot be attacked by any other argument. The arguments b and c have to be attacked by a. They could also be attacked by d, but this attack is optional. d has to be attacked by b or c and optionally can also be attacked by itself or e. Finally, e has to attack itself and can also optionally be attacked by d. One observation we can make is that most of the uncertainty in the attack constraints is related to the argument d as it can still optionally attack b, c, e and itself. The algorithm then returns the updated set of attack constraints C.

The third and last input labeling $\ell_3 = \{\mathsf{in} = \{d\}, \mathsf{out} = \{e\}, \mathsf{undec} = \{a, b, c\}\}$ is conflict-free. In our current situation it is very helpful to learn this labeling, because the fact that d is labeled in allows us to clear the uncertainty regarding its outgoing attacks. The labeling tells us that d cannot attack a, b, c and itself. In addition to that, we now know that d has to attack the out-labeled argument e since d is the only argument with the label in. For the labeling ℓ_3 we obtain the following attack constraints:

$$C_{a,\ell_3} = \neg r_{da}$$

$$C_{b,\ell_3} = \neg r_{db}$$

$$C_{c,\ell_3} = \neg r_{dc}$$

$$C_{d,\ell_3} = \neg r_{dd}$$

$$C_{e,\ell_3} = r_{de}$$

Again, we combine the attack constraints C_ℓ with the constraints C from the previous step via a conjunction. When combining the constraints we can also apply simplifications, however this is not necessary in this example. So, the attack constraints after learning all three labelings look like this:

$$C_a = \neg r_{aa} \wedge \neg r_{da} \wedge \neg r_{ea} \wedge (r_{ba} \vee r_{ca})$$
$$C_b = r_{ab} \wedge \neg r_{bb} \wedge \neg r_{cb} \wedge \neg r_{db} \wedge \neg r_{eb}$$
$$C_c = r_{ac} \wedge \neg r_{bc} \wedge \neg r_{cc} \wedge \neg r_{dc} \wedge \neg r_{ec}$$
$$C_d = \neg r_{ad} \wedge \neg r_{dd} \wedge (r_{bd} \vee r_{cd})$$
$$C_e = \neg r_{ae} \wedge \neg r_{be} \wedge \neg r_{ce} \wedge r_{de} \wedge r_{ee}$$

These attack constraints now represent exactly the set of argumentation frameworks that are consistent with all three input labelings. The final step of the algorithm is now to construct these argumentation frameworks. Here, we use the fact that the atoms used in the attack constraints directly correspond to attacks in an argumentation framework, as described in Definition 3.3. First, we compute the models of the attack constraints of each argument:

$$\mathcal{M}_a = \{[r_{ba} \to \mathit{true}], [r_{ca} \to \mathit{true}], [r_{ba}, r_{ca} \to \mathit{true}]\}$$
$$\mathcal{M}_b = \{[r_{ab} \to \mathit{true}]\}$$
$$\mathcal{M}_c = \{[r_{ac} \to \mathit{true}]\}$$
$$\begin{aligned}\mathcal{M}_d = \{&[r_{bd} \to \mathit{true}], [r_{cd} \to \mathit{true}], [r_{bd}, r_{cd} \to \mathit{true}], [r_{bd}, r_{ed} \to \mathit{true}],\\ &[r_{cd}, r_{ed} \to \mathit{true}], [r_{bd}, r_{cd}, r_{ed} \to \mathit{true}]\}\end{aligned}$$
$$\mathcal{M}_e = \{[r_{de}, r_{ee} \to \mathit{true}]\}$$

The next step is constructing the corresponding partial attack relations for each model with the method *modelToAttacks*($\mathcal{A}$). This is quite simple, since every atom directly represents an attack. For example, the model $\mathcal{A}_{a,1} = [r_{ba} \to \mathit{true}]$ would have the corresponding partial attack relation $\mathsf{R}_{a,1} = \{(b, a)\}$, since r_{ba} is the only atom which is valuated as true in $\mathcal{A}_{a,1}$. Overall, we construct the following partial attack relations for each argument:

$$R_a = \{\{(b, a)\}, \{(c, a)\}, \{(b, a), (c, a)\}\}$$
$$R_b = \{\{(a, b)\}\}$$
$$R_c = \{\{(a, c)\}\}$$
$$R_d = \{\{(b, d)\}, \{(c, d)\}, \{(b, d), (c, d)\}, \{(b, d), (e, d)\}, \{(c, d), (e, d)\}, \{(b, d), (c, d), (e, d)\}\}$$
$$R_e = \{\{(d, e), (e, e)\}\}$$

Here, we can see that b, c and e only have one partial attack relation while a has 3 and the argument d has 6 corresponding partial attack relations. That means, overall we will have $3 * 6 = 18$ different argumentation frameworks. These frameworks are constructed by taking all possible combinations while taking one partial attack relation from each argument, i.e., we obtain the set of all attack relations $R_L = R_a \times R_b \times R_c \times R_d \times R_e$. We then take these attack relations $\mathsf{R}_i \in R_L$ and assemble the corresponding

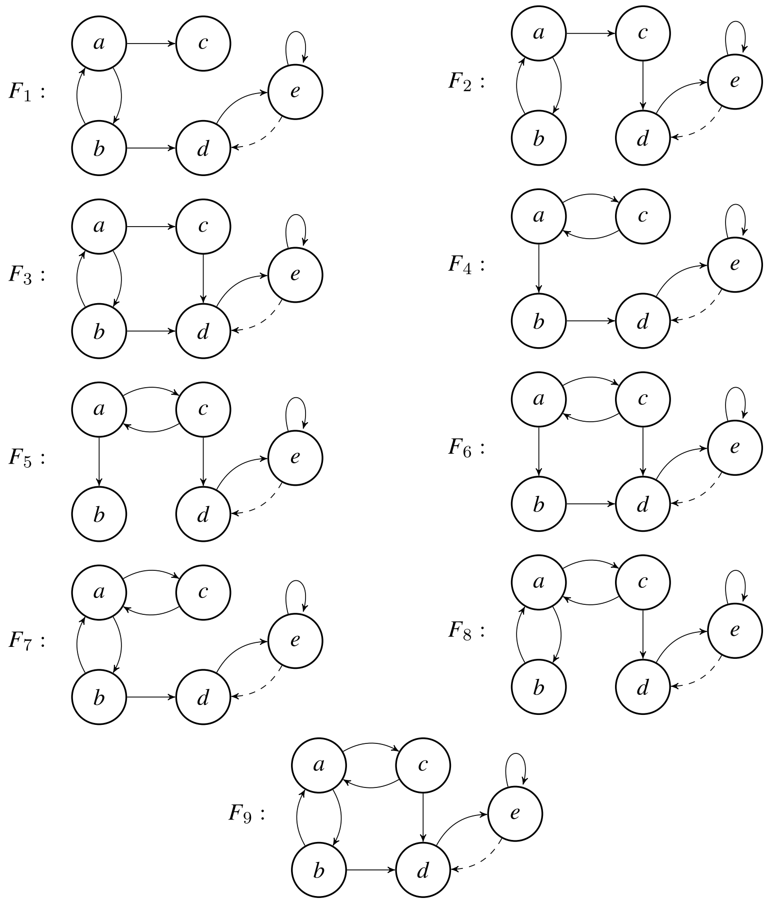


Fig. 4. All 18 constructed argumentation frameworks which are consistent with the labelings ℓ_1, ℓ_2 and ℓ_3. Each framework F_i represents two actual argumentation frameworks: F_i does not have the attack (e, d) and F_i' includes the optional attack (e, d).

argumentation framework $F_i = (\mathsf{Arg}, \mathsf{R}_i)$. The set $\mathbb{F}$ of all argumentation frameworks that are consistent with L then consists of exactly these argumentation frameworks.

All constructed argumentation frameworks are shown in Fig. 4. In the figure, each framework F_i represents two actual argumentation frameworks: F_i which is the depicted framework without the attack (e, d) and F_i' which does include the optional attack (e, d). All of these argumentation frameworks are consistent with the three input labelings.

4. Experiments

In this section we will conduct an experimental feasibility study of the approach introduced in Section 3. For that, we will consider both a naive implementation of the algorithm, as well as an optimized implementation with parallelized learning and construction steps. For the evaluation we will measure the performance of the algorithm, i.e., the time the algorithm takes to learn the input labelings and the time for constructing an argumentation framework that satisfies the learned constraints.

We will look at two different scenarios for our experiments. In the first scenario, we will use argumentation frameworks generated with the AFBenchGen2 generators [10] in order to measure how the algorithm's performance scales with the number of arguments. In the second scenario, we will use the benchmark argumentation frameworks from the ICCMA'19 competition [6] to measure the performance of the algorithm in a more realistic application scenario.

In the following, we briefly describe the system, both implementations and the two main metrics used for the evaluation.

System setup. The experiments were run on a machine with the Ubuntu 20.04 operating system. The machine has a 3.4 GHz Intel Xeon E5-2643v3 CPU with 12 cores and 24 threads and a total of 192 GB DDR4 memory.

Implementation. The implementation[1] has been done in Java as part of the *TweetyProject* library [26]. For the computation of the models of the attack constraints, needed for the construction of argumentation frameworks in the final step of the algorithm, the Java-based SAT-solver Sat4j[2] is used. In the *naive* version, all computations are done sequentially. On the other hand, in the *optimized* implementation, the learning of a labeling is done in parallel, i.e., the computation of the attack constraints for each argument. In addition to that, the computation of models of the constraints in the construction step is also done in parallel.

Metrics. In the following we define some questions that we want to answer with the experiments. Each question has a corresponding metric that will be investigated in order to answer it:

(1) *How does the time to learn a single input labeling scale with the number of arguments?*
The goal of this question is measuring how the algorithm's performance scales with the number of arguments that are considered. In order to do that, we consider the time t_{learn} the algorithm needs for learning all input labelings and set it in relation to the number of processed input labelings $|L|$. This removes the distortion of the results caused by the number of learned labelings. That means, we compute the learning time per labeling (TPL) as

$$t_{TPL} = \frac{t_{learn}}{|L|}.$$

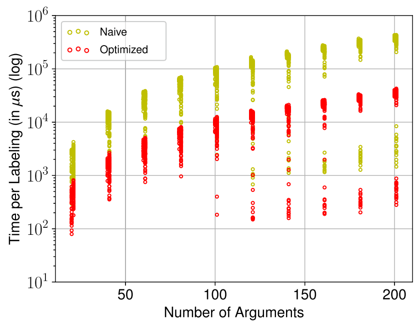
(2) *How much time does it take to construct an argumentation framework that is consistent with all input labelings?*
For this question we measure the time t_{constr} it takes to construct some argumentation framework $F \in \mathbb{F}$ that is consistent with the input labelings. This includes the computation of a model for each attack constraint. This allows us to get an idea of how the construction time scales with the number of arguments and also how effective the parallelization of the model computation is.

(3) *How close is the constructed argumentation framework to the original argumentation framework?*
For that, we compare the attack relation of the constructed argumentation framework $F' = (\mathsf{Arg}, \mathsf{R}')$ to that of the original framework $F = (\mathsf{Arg}, \mathsf{R})$ by counting the number of different attacks d_{att} between them normalized by the number of possible attacks, computed as

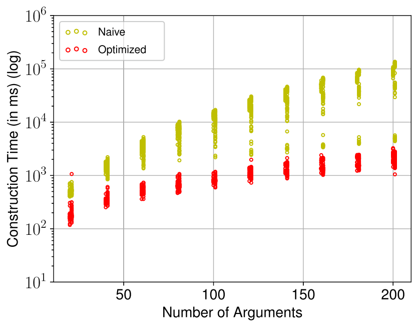
$$d_{att} = \frac{|\mathsf{R} \setminus \mathsf{R}'| + |\mathsf{R}' \setminus \mathsf{R}|}{|\mathsf{Arg} \times \mathsf{Arg}|}.$$

[1]Link to implementations: https://e.feu.de/aflearner.

[2]http://www.sat4j.org



(a) The learning time per labeling t_{TPL} in relation to the number of arguments |Arg|.



(b) The construction time t_{constr} in relation to the number of arguments |Arg|.

Fig. 5. Learning time per labeling and construction time in relation to the number of arguments for the AFs generated via AFBenchGen2.

Essentially, this will give us some insight on how many uncertainty is still left after learning the input labelings. If the constructed argumentation framework is very far from the original one, i.e., $d_{att} \gg 0$, then the input labelings leave many possibilities for attacks open. On the other hand, if they are close to each other, then we can assume not many argumentation frameworks are consistent with all input labelings.

4.1. *Experiment 1: AFBenchGen2*

The main goal of our first experiment is to investigate how our algorithm scales with the number of arguments. In particular, we are interested in how the parallelization affects the performance.

For this experiment, we generated a total of 1200 argumentation frameworks $F = (\mathsf{Arg}, \mathsf{R})$ with $n = |\mathsf{Arg}| = 20, 40, \ldots, 200$ arguments with the three AFBenchGen2 generators [10]. So, for each n we have 120 argumentation frameworks, one third generated by each of the generators: WattsStrogatz, ErdösRenyi and BarabasiAlbert. For each argumentation framework we compute all inputs (ℓ, σ) with $\sigma \in \{ad, co, st\}$. Then, we iteratively learn up to 1000 of these input labelings and measure the time the algorithm takes to learn them and to construct a consistent argumentation framework.

4.1.1. *Results*

Let us first consider the median learning time per labeling t_{TPL}. For the naive implementation, the median time per labeling scales from 2 ms for the instances with 20 arguments up to 361 ms for the instances with 200 arguments. While the number of arguments has increased tenfold, the time per labeling has increased by a factor of 172. Fig. 5(a) shows the learning time per labeling in relation to the number of arguments. Note the logarithmic scale of the vertical axis. We can quite clearly see that the time per labeling scales polynomially with the number of arguments for the naive implementation. This is also supported by the fact that the complexity of computing the labeling constraint is in $O(n^2)$, as we have to compute n attack constraints, one for each argument a, and for each computation we have to consider all n arguments once to determine their relation to a.

On the other hand, in the optimized implementation the median time per labeling ranges from 477 μs at 20 arguments to only 35 ms at 200 arguments, which is an increase by a factor of 73. That also means, for the smaller instances the parallelization reduces the computing time by approx. 77% and for the largest instances with 200 arguments by over 90%.

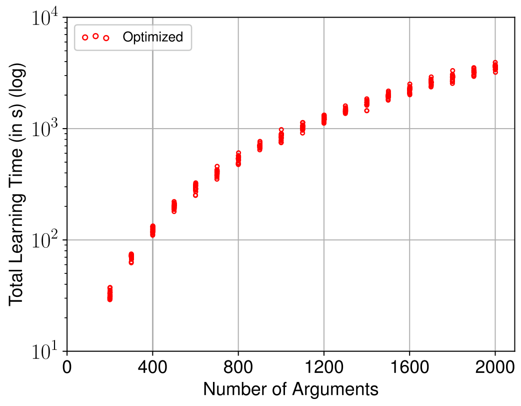


Fig. 6. The total learning time t_{learn} per instance in relation to the number of arguments |Arg| of the optimized version for larger AFs generated via AFBenchGen2.

For the instances with 120 or more arguments we can observe outliers with significantly lower learning times per labeling. In these cases the argumentation frameworks had only very few labelings and thus less than the intended 1000 labelings were learned. This suggest that there is a positive correlation between the number of learned labelings and the time per labeling t_{TPL}.

To ensure the observed scaling of the algorithm is accurate and not due to hardware, we consider larger and thus harder instances with up to 2000 arguments, generated in a similar fashion. The results for the total learning time t_{learn} for the optimized version of the algorithm are shown in Fig. 6. While the instances with 200 arguments have a total learning time of 13.4 s on average, the largest instances with 2000 arguments have an average total learning time of 1524 s. As before, we can observe a polynomial scaling of the learning time with the number of arguments, even for the harder instances.

We now look at the time it takes to construct an argumentation framework that is consistent with the attack constraints of the learned labelings. This includes computing a model for the attack constraint of each argument as well as converting these models to an attack relation. The time t_{constr} it takes to do this has been measured, depending on the number of arguments, and the results are shown in Fig. 5(b). For the naive implementation, the time for constructing ranges from a median of 62 ms for $n = 20$ to 9.7 s for the instances with 200 arguments. Similar to the time per labeling t_{TPL}, the construction time t_{constr} scales polynomially with the number of arguments as already discussed in Section 3.3. The optimized implementation scales a lot better with the number of arguments. The smaller instances with 20 arguments only require about 23 ms on average while the instances with 200 arguments take up to an average of 250 ms for the construction step. This is a near linear increase (factor of 11) of construction time from 20 to 200 arguments, while the naive implementation increased by a factor of 158 over the same span. Furthermore, at 200 arguments the optimized version reduces the computing time by over 97%.

Regarding the difference between original and learned argumentation framework we can observe the following. On average, less than 2% of the possible attacks were different for the instances in this experiment with a standard deviation of 0.01. The biggest difference was only 5.67%. In general, we can observe that the learned argumentation frameworks contained about 55% less attacks than the original one. This is to be expected, since in our scenario we are interested in the argumentation frameworks that produce *at least* the input labelings. That means, we explicitly allow other labelings to exist in the learned framework. This allows for a less restrictive attack relation, leading to fewer attacks being necessary to represent the labelings.

Overall, we can say that the paralellization is very effective in reducing the computation time for both processing a labeling as well as constructing a consistent argumentation framework in this artificial scenario. It is however more effective in the construction step than in the learning step.

4.2. Experiment 2: ICCMA'19

In the second experiment we will learn argumentation frameworks on the basis of some more realistic data in order to get an idea of the general performance of our algorithm. For that, we will use the argumentation frameworks from the ICCMA'19 competition [6]. In this experiment we consider only admissible, complete and stable labelings. It would not be feasible to construct all argumentation frameworks since there may be millions of frameworks that may be consistent with the input labelings, thus we only construct one argumentation framework that satisfies the attack constraints.

From the benchmark set, we consider all argumentation frameworks that have less than 1,000 arguments. Thus, we have 276 argumentation frameworks with the number of arguments ranging from 4 to 1,000. For each argumentation framework, we compute the inputs (ℓ, σ) with $\sigma \in \{ad, co, st\}$ and randomly learn from up to 1000 input labelings. After learning, we construct *one* argumentation framework $F \in \mathbb{F}$ that satisfies all learned attack constraints, i.e., we simply take the first model computed by the SAT solver.

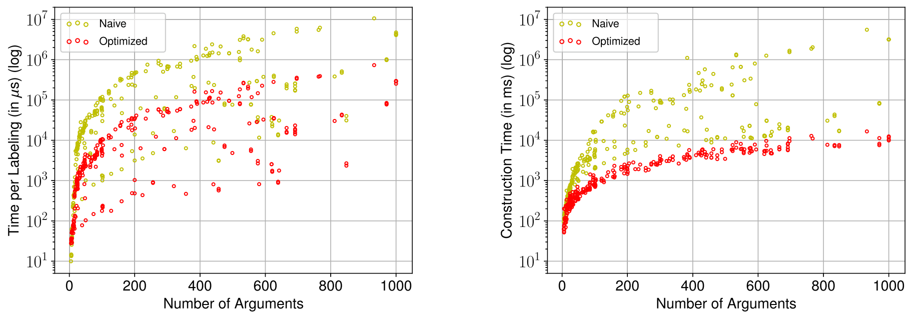
4.2.1. Results

In the following, the results of the first experiment are summarized and we look at each of the above defined questions and metrics.

We consider again the time per labeling $t_{TPL} = \frac{t_{learn}}{|L|}$. Fig. 7(a) shows the time per labeling in relation to the number of arguments of the corresponding ICCMA instance.

For the naive version, the time per labeling ranges from 10 μs to 10.6 s with a median of 42.25 ms per labeling. On the other hand the time per labeling ranges from 28 μs to 733 ms with a median of 3.82 ms per labeling for the optimized implementation.

On average, the optimization brings a decrease in computation time by a factor of about 10. The higher the number of arguments the higher the performance improvement. For the smallest instances with less than 10 arguments the naive implementation actually performs better. This is likely due to some



(a) The learning time per labeling t_{TPL} in relation to the number of arguments |Arg|.

(b) The construction time t_{constr} in relation to the number of arguments |Arg|.

Fig. 7. Learning time per labeling and construction time in relation to the number of arguments in the experiment with the ICCMA'19 instances.

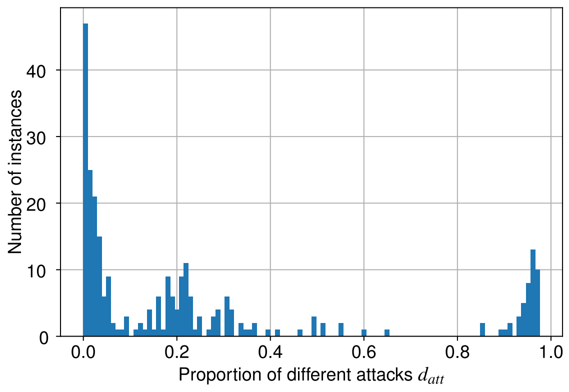


Fig. 8. The frequency distribution of the normalized number of different attacks d_{att} for the ICCMA'19 instances.

computational overhead of the parallelization, but already for instances with 10 or more arguments the optimized implementation consistently and significantly outperforms the naive version.

One observation we can make for both implementations is that there are some instances for which the time per labeling is significantly lower compared to other instances of similar size. Like in the previous experiment, these are instances where less than the intended 1000 labelings have been processed (simply due to the fact that there where not enough labelings for those instances). This is likely due to the fact, that with a growing number of input labelings the attack constraints get more complex and thus it takes longer to check satisfiability.

We now consider the construction times t_{constr} for the ICCMA'19 dataset which are shown in Fig. 7(b). The time for constructing a consistent argumentation framework ranges from 94.37 ms to 5527 s with a median of 6 s for the naive implementation. For the optimized version the times range from 51.75 ms to 16.6 s with a median of 996.8 ms. As we have already observed in the previous experiment the effect of the parallelization is even greater for the construction step where the optimized implementation is on average about 22 times faster than the naive version. Even for the smallest instances the optimized implementation is between 1.5 and 2 times faster while for the worst instances it reduces the construction time by a factor of over 200. It should be noted that we can again see the instances with less labelings learned, also being faster than comparable instances in the construction step. However, this seems to only effect the naive algorithm and the optimized algorithm has a lot less variance in its construction times.

Finally, Fig. 8 shows the distribution of the normalized number of different attacks d_{att} for the ICCMA'19 instances. On average We can see that for the majority of instances the constructed argumentation framework is quite close to the original framework. This suggests that after learning 1000 labelings for these instances, there is not much uncertainty left and only a few argumentation frameworks are consistent with these labelings. However, only very few instances yield the exact same argumentation framework. If we would want to reconstruct the exact argumentation framework, we would either need additional labelings or some other information, e.g., negative examples. Interestingly, there are also a number of instances with a proportion d_{att} of over 90%. They all have in common that the attack relation is very dense, i.e., over 90% of all possible attacks are present in the original argumentation framework. That also means these frameworks have only few labelings. Learning these labelings then means we only need very few attacks to ensure consistency with them, since we do not require there to be no further labelings for the constructed argumentation framework. In general, we can observe that the constructed argumentation frameworks contain on average 70% less attacks than the original framework. This is again related to the fact that we allow further labelings to exist in the learned argumentation frameworks.

Overall, this experiment on the ICCMA'19 dataset has confirmed the observation from the first experiment. Our findings suggest a time complexity within $O(n^2|L|)$ for learning argumentation frameworks from labelings. The parallelization of both the learning and construction step is very effective and significantly improves performance, especially for larger argumentation frameworks.

5. Discussion

In the previous sections we have introduced and discussed an algorithm for learning a set of argumentation frameworks from a given set of labelings. This algorithm computes a set of attack constraints for each labeling. These constraints represent the acceptability of each argument with regards to the labeling. Afterwards they are combined into a single attack constraint per argument. This set of attack constraints then represents all input labelings and can then be used to construct the argumentation frameworks that are consistent with the input labelings.

In this section we will take a closer look at our algorithm and discuss its strengths and weaknesses. Furthermore, we will discuss some existing algorithms from Riveret et al. [25] and Niskanen et al. [23] and the recent work of Mumford et al. [22] on the computational complexity of the AF learning problem.

We will first discuss some key advantages that distinguish our algorithm proposed in this work from the existing algorithms. One key point is the structure of the algorithms. Our algorithm works in an iterative way and can be applied on a stream of input labelings similarly to the algorithm of Riveret et al. [25]. This is an important difference to the algorithm of Niskanen et al. [23] which only works with a set of input extensions. An iterative approach means we are able to get intermediate results after each processed input labeling. It also means that we are able to refine our result at any point by learning more labelings. This would not be possible in the algorithm of Niskanen et al. [23] where we would have to start over and learn all labelings again. The iterative approach also allows us to choose the order in which the labelings are processed. This means we can add an additional layer to the algorithm by deciding which labeling to learn next based on the current status (i.e., the last intermediate result).

Another important difference is related to the conditions that are used to encode the labelings. The algorithm of Niskanen et al. [23] also uses conditions based on the input, however there is an important difference: these conditions model one extension as a whole. So, using the terminology introduced in Section 3, these conditions are formulae over the set $\mathit{attackAtoms}(\mathsf{Arg})$, i.e., the set of atoms representing all possible attacks over Arg. On the other hand, in our algorithm we have one condition per argument which are formulae over the set of atoms $\mathit{inAttacks}(a)$, i.e., all attacks onto the argument. In other words, the algorithm of Niskanen et al. [23] uses global conditions while our algorithm uses local conditions from the perspective of arguments. These local conditions are important because they enable us to define the iterative approach mentioned above, since we can easily combine formulae for the same argument. This is considerably harder in the algorithm of Niskanen et al. [23]. The fact that the conditions are local and from the perspective of the individual arguments also makes them easier to understand and modify. Furthermore, as we have seen in Section 4, the local conditions also lead to a significantly better performance both while learning a labeling as well as during the construction of an argumentation framework that satisfies the constraints.

The most important advantage of our algorithm compared to those of Riveret et al. and Niskanen et al. is the following: Our algorithm maintains a representation of all argumentation frameworks that are consistent with the input labelings while both other algorithms simply return one fitting argumentation framework. The algorithm of Riveret et al. [25] only maintains one representation of a weighted argumentation framework internally and returns this at the end. On the other hand, the algorithm of Niskanen et al. [23] computes one condition per labeling and as a result returns just one model that represent an argumentation framework. Our algorithm preserves all of the information that is given to it via the input labelings. This is done by translating the input labelings into a set of attack constraints and then combining and simplifying them. So, internally the algorithm only stores one set of mutually independent attack constraints that represent the set of argumentation frameworks. This approach enables us to incorporate additional input labelings at any point and refine the set of argumentation frameworks further. It also ensures that we do not discard any correct argumentation framework during the learning process, something that can happen in both the algorithms of Riveret et al. [25] and Niskanen et al. [23]. It should be mentioned however, that the algorithm of Niskanen et al. [23] can be modified to also return all consistent argumentation frameworks.

We will now look at some other minor differences between all three algorithms. The input of our algorithm is a set of arguments and a set of input labelings. That means, we assume that all arguments of the hidden framework are known. The same assumption is also made by both other algorithms. The second part of the input is the set of input labelings. In the iterative version this can also be a stream of input labelings, similarly to the algorithm of Riveret et al. [25]. This is a different approach compared to the algorithm of Niskanen et al. [23], which uses positive and negative examples in the form of extensions.

Furthermore, all approaches differ in the semantics that are accepted for the input. The algorithm of Riveret et al. [25] only allows grounded labelings as input, while the algorithm of Niskanen et al. [23] accepts conflict-free, admissible, complete, stable, grounded and preferred extensions. Our algorithm accepts conflict-free, admissible, complete and stable labelings. While the algorithm covers many semantics, it does not support grounded and preferred semantics. The reason for that is related to the definition of these semantics. The grounded labeling is defined as the minimal complete labeling and the preferred labelings are maximally complete. These constraints cannot be encoded in the attack constraints and thus it is not possible to learn these labelings with our algorithm, as already discussed in Section 3.2.5. The same applies to some other semantics proposed in the literature, such as semi-stable [7] which maximizes the attacked arguments of an extension. Similarly, the naive semantics and the semantics based on it, like CF2 [5], work with maximal conflict-free sets and thus cannot be modeled by the algorithm in its current form. However, it might be possible to address these issues by extending the algorithm with some form of global conditions. One possibility would be to use conditions similar to those used by Niskanen et al. [23] for modeling the grounded and preferred extensions and adjust them for the respective labelings. It should be mentioned, that this would of course weaken the advantages of our local constraints.

The algorithm of Riveret et al. [25] uses grounded sub-framework labelings as input. This considerably eases the task of learning from labelings since we can just consider all sub-frameworks with just two arguments and take the grounded labeling of these as input. Our algorithm does not rely on such labelings and is able to reconstruct argumentation frameworks from standard labelings.

Argumentation frameworks may contain self-attacking arguments. The algorithm of Riveret et al. [25] does have problems with such argumentation frameworks and there are cases where this algorithm is not able to reconstruct the hidden argumentation framework if there are self-attacking arguments. Our algorithm does not have any problems in that case and is able to work with self-attacking arguments.

Finally, we will also discuss some weaknesses of our algorithm compared to the existing work. The fact that our algorithm requires labelings instead of extensions as input can be considered a weakness. Labelings are more specific than extensions and hold more information about the argumentation framework that produced them. They make a distinction between directly rejected (out) and indirectly rejected (undecided) arguments. Without this distinction it would not be possible to compute the attack constraints in their current form. This is something that can potentially be worked on in the future.

Another observation regarding the input of the algorithm can be made. Our algorithm cannot handle input noise, i.e., some ‘wrong’ labelings that are not actually produced by the hidden argumentation framework. This is something that the algorithms of Riveret et al. [25] and Niskanen et al. [23] are able to handle. The algorithm of Riveret et al. [25] does this by using an internal weighted argumentation framework. This leaves more room to handle inconsistent input but might also lead to returning an argumentation framework which is not consistent with all of the correct input labelings. The algorithm of Niskanen et al. [23] uses weighted MaxSAT encodings for the input extensions. This allows them so find the argumentation framework which is consistent with the highest number of input extensions even if there are some noisy inputs. It might be possible to apply a similar concept to our attack constraints which is also something that can be tackled in the future. Alternatively, we could consider some paraconsistent logic [24] to interpret the attack constraints instead of the classical propositional logic.

The recent work of Mumford et al. [22] investigates the computational complexity of learning argumentation framework both from labelings and extensions. They focus on the complete semantics, but also consider grounded, preferred and stable semantics. Their results show that learning argumentation frameworks from extensions is **NP**-complete while learning from labelings is solvable in polynomial time. More specifically, they present an algorithm for constructing *one* consistent argumentation framework from labelings with a time complexity in $O(n^2|L|)$. This nicely complements our experimental findings, which suggest the same time complexity dimension for our algorithm. However, our algorithm can be parallelized, which significantly improves its performance. In addition to that, our approach is also capable of producing all consistent solutions instead of just one. This has one disadvantage however, as the algorithm by Mumford et al. has a space complexity within $O(n^2)$, while our algorithm needs more space to maintain a representation of multiple argumentation frameworks. In the worst case, our algorithm has a space complexity in $O(n^2|L|)$, which easily follows from the fact that we have one attack constraint C_a with at most n atoms for each of the n arguments and labelings. However, in practice we will often need less space than that since the attack constraints may be simplified due to overlap or contradictions in the learned labelings.

A final related work worthwhile to mention is [2], where an approach is presented that incrementally computes extensions when changes on the initial argumentation framework are observed, such as adding an attack. The problem faced in this work is somewhat orthogonal to ours, since we consider the argumentation framework to be unknown but fixed, and we process extensions (or more precisely labelings) incrementally.

6. Conclusion

We investigated the *AF learning* scenario, where we want to reconstruct argumentation frameworks from given labelings. The goal in this scenario is to find all argumentation frameworks that are consistent with at least all input labelings. To address this problem, we introduced a novel algorithm that iteratively processes labelings and translated them to attack constraints for each argument. These attack constraints are mutually independent and together they represent the set of all consistent argumentation frameworks. The independence of these constraints allows us to parallelize both the learning and construction part of the algorithm. Our experiments showed that this leads to a significant increase in performance. In particular, for the ICCMA'19 dataset the learning time has been improved by a factor of 10 while the construction time showed an even greater improvement with it being over 22 times faster. We also highlighted some of the key advantages of our approach compared to other similar existing approaches. These advantages are, among others, the ability to fully parallelize all computations and the fact that the attack constraints maintain a representation of all consistent argumentation frameworks at all times.

For future work, there are several possibilities. One such possibility is extending our algorithm to other AF-based formalisms, such as bipolar argumentation frameworks [9] or attack-support argumentation frameworks [11]. For these frameworks, there also exist acceptability semantics from whose labelings we can learn the respective frameworks. In particular, instances of these formalisms can be transformed into semantically equivalent meta argumentation frameworks [3]. That means, one only needs to deal with the meta arguments that are created by this transformation.

Another direction is to extend the capabilities of our algorithm. This could be done by defining semantic constraint functions for other semantics, e.g., for the strong admissible semantics [4]. We are however not limited to specific semantics. Alternatively, we can define semantic constraint functions for other concepts from the literature such as credulous or skeptical acceptance with respect to some semantics σ or properties like unattackedness. Another possibility would be to adapt the algorithm to be able to process negative examples, like the algorithm of Niskanen et al. [23]. It would also be interesting to convert the algorithm to a extension-based approach, if possible. One weakness of our approach is the inability to deal with noisy input. As already mentioned in Section 5, this could for example be addressed by using some paraconsistent logic [24].

Furthermore, it would also be interesting to consider a scenario where we can only obtain incomplete information in the form of a partial labeling wrt. some semantics where some of the arguments have no specified label. For a scenario like that, we would have to explore under which circumstances it is possible to recover information about the framework from the partial labeling and adapt the semantic constraint functions accordingly.

Another interesting thing to explore would be the relationship of the number of learned labelings and the difference between attacks in the hidden and the constructed argumentation framework. Meaning, we look at how many labelings we would need to learn in order to obtain construct exactly the hidden argumentation framework or some framework with at most k different attacks.

Finally, another interesting direction would be to make the scenario of learning argumentation frameworks from labelings more interactive, similar to the scenario of eliciting argumentation frameworks [19]. In that scenario the goal is to elicit a hidden argumentation framework from an entity by asking questions about the extensions of this framework. In our scenario, we could also include user input to control which labeling to learn next. This can then be used to steer the learning process to learn more "effectively", i.e., by choosing labelings that shrink the set of consistent argumentation frameworks the most.

Appendix. Proofs

Theorem 3.1. *Let (ℓ, cf) be a conflict-free input and C_ℓ is the labeling constraint of ℓ computed via $AttCon_{cf}$. With $\mathbb{F}$ we denote the set of argumentation frameworks that satisfy C_ℓ, i.e for all $F \in \mathbb{F}$ we have that $F \models C_\ell$.*

The semantic constraint function is sound *for conflict-free input labelings ℓ, i.e.,*

$$\forall F \in \mathbb{F} : \ell \text{ is a cf-labeling of } F$$

and the semantic constraint function is complete *for conflict-free input labelings ℓ, i.e.,*

$$\forall F = (\mathsf{Arg}, \mathsf{R}) : \ell \text{ is a cf-labeling of } F \implies F \in \mathbb{F}.$$

Proof of Theorem 3.1 (Soundness). Let (ℓ, cf) be a conflict-free input and Arg denotes the set of arguments. $C_\ell = \{C_a = AttCon_{cf}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of attack constraints computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C_ℓ.

We will proof by contradiction that the set of argumentation frameworks $\mathbb{F}$ consistent with the attack constraints (i.e., constructed by the algorithm) is sound for conflict-free labelings, i.e., every argumentation framework $F \in \mathbb{F}$ is consistent with the conflict-free labeling ℓ.

Assume there exists an argumentation framework $F = (\mathsf{Arg}, \mathsf{R}') \in \mathbb{F}$ which is not consistent with the labeling ℓ, i.e., ℓ is not a conflict-free labeling of F. Then, one of the following three cases must apply:

(1) There is an attack between two in-labeled arguments a and b in F.
(2) There is an out-labeled argument a which is not attacked by any in-labeled argument b in F.
(3) There is an undec-labeled argument a which is attacked by any in-labeled argument b in F.

Case 1: There is an attack between two in-labeled arguments a and b in F.

The algorithm constructs the argumentation frameworks based on the models of the attack constraints. We consider any argument $a \in \mathsf{in}(\ell)$. That means, there must be a model $\mathcal{A}$ of the attack constraint C_a such that $\mathcal{A}(r_{ba}) = true$. However, according to Definition 3.7 the attack constraint of a is defined as $C_a = \bigwedge_{b \in \mathsf{in}(\ell)} \neg r_{ba}$. Thus, it holds that $\mathcal{A}(r_{ba}) = false$ for any argument $b \in \mathsf{in}(\ell)$ and therefore there can be no attack between two in-labeled arguments in F.

Case 2: There is an out-labeled argument a which is not attacked by any in-labeled argument b in F.

That means, there exists an argument $a \in \mathsf{out}(\ell)$ such that for all arguments $b \in \mathsf{in}(\ell)$ it holds that $(b, a) \notin \mathsf{R}'$. So, there must be a model $\mathcal{A}$ of the attack constraint C_a such that $\mathcal{A}(r_{ba}) = false$ for all arguments $b \in \mathsf{in}(\ell)$. However, according to Definition 3.7 the attack constraint of a is defined as $C_a = \bigvee_{b \in \mathsf{in}(\ell)} r_{ba}$. From this it follows that there has to be at least one argument $b \in \mathsf{in}(\ell)$ for which $\mathcal{A}(r_{ba}) = true$. Thus, we have a contradiction and it holds that any argument $a \in \mathsf{in}(\ell)$ is always attacked by at least one in-labeled argument b.

Case 3: There is an undec-labeled argument a which is attacked by any in-labeled argument b in F.

That means, there exists an argument $a \in \mathsf{undec}(\ell)$ such that for any argument $b \in \mathsf{in}(\ell)$ it holds that $(b, a) \in \mathsf{R}'$. So, there must be a model $\mathcal{A}$ of the attack constraint C_a such that $\mathcal{A}(r_{ba}) = true$ for any arguments $b \in \mathsf{in}(\ell)$. However, according to Definition 3.7 the attack constraint of a is defined as $C_a = \bigwedge_{b \in \mathsf{in}(\ell)} \neg r_{ba}$. It follows that $\mathcal{A}(r_{ba}) = false$ for all arguments $b \in \mathsf{in}(\ell)$. Thus, we have a contradiction and it holds that every argument $a \in \mathsf{undec}(\ell)$ is not attacked by any in-labeled argument b.

All in all, it follows that every $F \in \mathbb{F}$ must produce the input labeling ℓ and thus the algorithm is *sound* for conflict-free input labelings. □

Proof of Theorem 3.1 (Completeness). Let (ℓ, cf) be a conflict-free input and Arg denotes the set of arguments. $C_\ell = \{C_a = AttCon_{cf}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of attack constraints computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C_ℓ.

We will proof by contradiction that the algorithm is complete for conflict-free labelings, i.e., for every argumentation framework F it holds that, if F is consistent with ℓ then $F \in \mathbb{F}$.

Assume there exists an argumentation framework $F = (\mathsf{Arg}, \mathsf{R}')$ which is consistent with the labeling ℓ and $F \notin \mathbb{F}$.

Then, according to Definition 2.3 the following three conditions hold for all arguments $a \in \mathsf{Arg}$ in F:

(1) If $a \in \mathsf{in}(\ell)$, then $\forall b \in \mathsf{Arg} : (b, a) \in \mathsf{R}' \to b \notin \mathsf{in}(\ell)$
(2) If $a \in \mathsf{out}(\ell)$, then $\exists b \in \mathsf{in}(\ell) : (b, a) \in \mathsf{R}'$
(3) If $a \in \mathsf{undec}(\ell)$, then $\forall b \in \mathsf{in}(\ell) : (b, a) \notin \mathsf{R}'$

According to the first condition, there can be no attack between any in-labeled arguments in F, i.e., $\forall a, b \in \mathsf{in}(\ell) : (a, b) \notin \mathsf{R}' \wedge (b, a) \notin \mathsf{R}'$. The attack (a, b) corresponds to the atom r_{ab} and (b, a) corresponds to r_{ba}. However, the attack constraint C_a for the in-labeled argument a is a conjunction and contains the literal $\neg r_{ba}$. This means that $\mathcal{A}(r_{ba}) = \mathit{false}$ for all models $\mathcal{A}$ of C_a. Similarly, the attack constraint for b enforces that $\mathcal{A}(r_{ab}) = \mathit{false}$ for all models $\mathcal{A}$. Thus, since F satisfies the first condition from above, it must also satisfy the attack constraint C_a for any in-labeled argument a.

Furthermore, every out labeled argument must be attacked by some in-labeled argument in F, i.e., $\forall a \in \mathsf{out}(\ell) : \exists b \in \mathsf{in}(\ell) : (b, a) \in \mathsf{R}'$. However, if that is the case, then F must also satisfy the attack constraint $C_a = \bigvee_{b \in \mathsf{in}(\ell)} r_{ba}$. It follows that F always satisfies the attack constraints of every out-labeled argument.

The third condition states that an undec-labeled argument cannot be attacked by an argument with the label in. Assume the undec-labeled argument a, then it holds for every argument $b \in \mathsf{in}(\ell)$ that the corresponding attack atom r_{ba} must be false. This matches exactly with the attack constraint for undec-labeled argument in conflict-free labelings $C_a = \bigwedge_{b \in \mathsf{in}(\ell)} \neg r_{ba}$. Thus every undec-labeled argument and its incoming attacks in F always satisfy all related attack constraints.

It follows that, if the argumentation framework F is consistent with the conflict-free labeling ℓ it also satisfies the attack constraints for all arguments as defined in Definition 3.7. Thus, it holds for all conflict-free labelings that, if F is consistent with ℓ, then $F \in \mathbb{F}$. □

Theorem 3.2. *Let (ℓ, ad) be an admissible input and C_ℓ is the labeling constraint of ℓ computed via $AttCon_{ad}$. With $\mathbb{F}$ we denote the set of argumentation frameworks that satisfy C_ℓ, i.e for all $F \in \mathbb{F}$ we have that $F \models C_\ell$.*

The semantic constraint function $AttCon_{ad}$ is sound *for admissible input labelings ℓ, i.e.,*

$$\forall F \in \mathbb{F} : \ell \textit{ is a } \mathrm{ad}\textit{-labeling of } F$$

and the semantic constraint function $AttCon_{ad}$ is complete *for admissible input labelings ℓ, i.e.,*

$$\forall F = (\mathsf{Arg}, \mathsf{R}) : \ell \textit{ is a } \mathrm{ad}\textit{-labeling of } F \implies F \in \mathbb{F}.$$

Proof of Theorem 3.2 (Soundness). Let (ℓ, ad) be an admissible input and Arg denotes the set of arguments. $C_\ell = \{C_a = AttCon_{ad}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of attack constraints computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C_ℓ.

We will proof by contradiction that the set of argumentation frameworks $\mathbb{F}$ constructed by the algorithm is sound for admissible labelings, i.e., every argumentation framework $F \in \mathbb{F}$ is consistent with the labeling ℓ.

Assume there exists an argumentation framework $F = (\mathsf{Arg}, \mathsf{R}') \in \mathbb{F}$ which is not consistent with the labeling ℓ, i.e., ℓ is not a admissible labeling of F. Then, one of the following four cases must apply:

(1) There is an attack between two in-labeled arguments a and b in F.
(2) There is an out-labeled argument a which is not attacked by any in-labeled argument b in F.
(3) There is an undec-labeled argument a which is attacked by any in-labeled argument b in F.
(4) There is an in-labeled argument a which is not defended by $\mathsf{in}(\ell)$ against any argument b.

Cases 1–3 are the same as for the conflict-free semantics. So, we only have to consider case 4 here.

Case 4: There is an in-labeled argument a which is not defended by $\mathsf{in}(\ell)$ against any argument b. That means, there exists an argument $a \in \mathsf{in}(\ell)$ such that $\exists b \in \mathsf{Arg} : (b, a) \in \mathsf{R}'$ and $\nexists c \in \mathsf{in}(\ell) : (c, b) \in \mathsf{R}'$. There are three possible labels for the argument b that we have to differentiate: in, out and undec. If $b \in \mathsf{in}(\ell)$, then we would have a conflict between in-labeled arguments, which is not possible as shown already in Case 1. If $b \in \mathsf{out}(\ell)$, then b cannot have any incoming attack from an argument $c \in \mathsf{in}(\ell)$. We have already shown in Case 2 that this is not possible. Finally, assume $b \in \mathsf{undec}(\ell)$ with $(b, a) \in \to'$ and there is no $c \in \mathsf{in}(\ell)$ with $(c, b) \in \mathsf{R}'$. Then there must be a model $\mathcal{A}$ of the attack constraint C_a such that $\mathcal{A}(r_{ba}) = \mathit{true}$. However, according to Definition 3.8 the attack constraint of a is defined as $C_a = \bigwedge_{b \in \mathsf{Arg} \setminus \mathsf{out}(\ell)} \neg r_{ba}$. Thus, it holds that $\mathcal{A}(r_{ba} = \mathit{false}$ for any argument $b \in \mathsf{undec}(\ell)$ and therefore there can be no attack from an undec-labeled argument on an in-labeled argument.

To summarize, it follows that every $F \in \mathbb{F}$ must be consistent with the input labeling ℓ and thus the algorithm is *sound* for admissible input labelings. □

Proof of Theorem 3.2 (Completeness). Let (ℓ, ad) be an admissible input and Arg denotes the set of arguments. $C_\ell = \{C_a = \mathit{AttCon}_{ad}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of attack constraints computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C_ℓ.

We proof by contradiction that the algorithm is complete for admissible labelings, i.e., for every argumentation framework F it holds that, if F is consistent with ℓ then $F \in \mathbb{F}$.

Assume there exists an argumentation framework $F = (\mathsf{Arg}, \mathsf{R}')$ which is consistent with the labeling ℓ and $F \notin \mathbb{F}$.

Then, according to Definition 2.4 the following three conditions hold for all arguments $a \in \mathsf{Arg}$ in F:

(1) If $a \in \mathsf{in}(\ell)$, then $\forall b \in \mathsf{Arg} : (b, a) \in \mathsf{R}' \to b \in \mathsf{out}(\ell)$.
(2) If $a \in \mathsf{out}(\ell)$, then $\exists b \in \mathsf{in}(\ell) : (b, a) \in \mathsf{R}'$.
(3) If $a \in \mathsf{undec}(\ell)$, then $\forall b \in \mathsf{in}(\ell) : (b, a) \notin \mathsf{R}'$.

Conditions 2 and 3 are the same as for conflict-free semantics. So, we only have to proof that the first condition also holds in F.

According to the first condition, any in-labeled argument can only be attacked by out-labeled arguments in F, i.e., $\forall a, b \in \mathsf{Arg} : (b, a) \in \mathsf{R}' \wedge a \in \mathsf{in}(\ell) \to b \in \mathsf{out}(\ell)$. The attack (b, a) corresponds to the atom r_{ba}. The attack constraint C_a for the in-labeled argument a is a conjunction and contains the literal $\neg r_{ba}$ for every argument b with the label in or undec. This means, for every argument $b \notin \mathsf{out}(\ell)$ it holds that $\mathcal{A}(r_{ba}) = \mathit{false}$ for all models $\mathcal{A}$ of C_a.

This is exactly what the first condition states and thus F must also satisfy the attack constraint C_a for any in-labeled argument a.

It follows that, if the argumentation framework F is consistent with the admissible labeling ℓ it also satisfies the attack constraints for all arguments as defined in Definition 3.8. Thus, it holds for all admissible labelings that, if F is consistent with ℓ, then $F \in \mathbb{F}$. □

Theorem 3.3. *Let (ℓ, co) be a complete input and C_ℓ is the labeling constraint of ℓ computed via $AttCon_{co}$. With $\mathbb{F}$ we denote the set of argumentation frameworks that satisfy C_ℓ, i.e for all $F \in \mathbb{F}$ we have that $F \models C_\ell$.*

The semantic constraint function $AttCon_{co}$ is sound *for complete input labelings ℓ, i.e.,*

$$\forall F \in \mathbb{F} : \ell \textit{ is a } \mathsf{co}\textit{-labeling of } F$$

and the semantic constraint function $AttCon_{co}$ is complete *for complete input labelings ℓ, i.e.,*

$$\forall F = (\mathsf{Arg}, \mathsf{R}) : \ell \textit{ is a } \mathsf{co}\textit{-labeling of } F \implies F \in \mathbb{F}.$$

Proof of Theorem 3.3 (Soundness). Let (ℓ, co) be a complete input and Arg denotes the set of arguments. $C_\ell = \{C_a = AttCon_{co}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of attack constraints computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C_ℓ.

The proof of the soundness of the algorithm for complete input labelings is similar to the proof for admissible input labelings. The only difference is that we have to consider an additional fifth possibility for the case distinction:

(5) There is an argument $a \notin \mathsf{in}(\ell)$ which is defended by $\mathsf{in}(\ell)$ in F.

Cases 1–4 have been shown in the proofs for conflict-free and admissible semantics.

Case 5: There is an argument $a \notin \mathsf{in}(\ell)$ which is defended by $\mathsf{in}(\ell)$ in F. That means, there exists an argument a with the label out or undec such that $\forall c \in \mathsf{Arg} : (c, a) \in \mathsf{R}' \to \exists d \in \mathsf{in}(\ell) : (d, c) \in \mathsf{R}'$.

Assume $a \in \mathsf{out}(\ell)$, then it follows that there must be an argument $c \in \mathsf{in}(\ell)$ that attacks a. Then, we also know from Case 1 earlier that there can be no argument $d \in \mathsf{in}(\ell)$ that attacks the in-labeled argument c. Thus, if a has the label out it is never defended by $\mathsf{in}(\ell)$ at the same time.

Assume $a \in \mathsf{undec}(\ell)$, then the attack constraint of a is defined as $C_a = \bigwedge_{b \in \mathsf{in}(\ell)} \neg r_{ba} \wedge (\bigvee_{c \in \mathsf{undec}(\ell)} r_{ca})$. From that it follows that in every model $\mathcal{A}$ of C_a there exists some argument $c \in \mathsf{undec}(\ell)$ such that $\mathcal{A}(r_{ca}) = \mathit{true}$. However, from the attack constraint C_c of c it would then follow that $\mathcal{A}(r_{dc}) = \mathit{false}$ for any in-labeled argument d, i.e., an undec-labeled argument is never attacked by an in-labeled argument. Thus, if a has the label undec it is never defended by $\mathsf{in}(\ell)$.

It follows, there is no argument $a \notin \mathsf{in}(\ell)$ that is defended by $\mathsf{in}(\ell)$ and thus it holds that every $F \in \mathbb{F}$ must be consistent with the input labeling ℓ, i.e., the algorithm is *sound* for complete input labelings. $\square$

Proof of Theorem 3.3 (Completeness). Let (ℓ, co) be a complete input and Arg denotes the set of arguments. $C_\ell = \{C_a = AttCon_{co}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of attack constraints computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C_ℓ.

We proof that the algorithm is complete for complete input labelings, i.e., for every argumentation framework F it holds that, if F is consistent with ℓ then $F \in \mathbb{F}$.

Assume there exists an argumentation framework $F = (\mathsf{Arg}, \mathsf{R}')$ which produces the labeling ℓ and $F \notin \mathbb{F}$.

Then, according to Definition 2.5 the following four conditions hold for all arguments $a \in \mathsf{Arg}$ in F:

(1) If $a \in \mathsf{in}(\ell)$, then $\forall b \in \mathsf{Arg} : (b, a) \in \mathsf{R}' \to b \in \mathsf{out}(\ell)$.
(2) If $a \in \mathsf{out}(\ell)$, then $\exists b \in \mathsf{in}(\ell) : (b, a) \in \mathsf{R}'$.
(3) If $a \in \mathsf{undec}(\ell)$, then $\forall b \in \mathsf{in}(\ell) : (b, a) \notin \mathsf{R}'$.
(4) If $a \in \mathsf{undec}(\ell)$, then $\exists b \in \mathsf{undec}(\ell) : (b, a) \in \mathsf{R}'$.

As shown in the proof for admissible and conflict-free labelings, from the conditions for in- and out-labeled arguments it follows that F must satisfy the respective attack constraints. We now show that from the third and fourth condition it follows that F also satisfies the attack constraint for complete semantics of any undec-labeled argument.

As shown in the proof for admissible labelings from the third condition it follows that F satisfies the admissible attack constraint $C_{a,1} = \bigwedge_{b\in\mathsf{in}(\ell)} \neg r_{ba}$ for any undec-labeled argument. This also equals the first part of the attack constraint for undec-labeled arguments under complete semantics. The fourth condition states that there must be some undec-labeled argument c that attacks the undec-labeled argument a. That means there exists an argument $c \in \mathsf{undec}(\ell)$ such that the corresponding atom r_{ca} is true, i.e., the formula $C_{a,2} \bigvee_{c\in\mathsf{undec}(\ell)} r_{ca}$ must be true. If we take the conjunction $C_a = C_{a,1} \wedge C_{a,2}$, then C_a is exactly the attack constraint for undec-labeled arguments in complete labelings as defined in Definition 3.9.

It follows that, if the argumentation framework F is consistent with the complete labeling ℓ it also satisfies the attack constraints for all arguments. Thus, it holds for all complete labelings that, if F is consistent with ℓ, then $F \in \mathbb{F}$. $\square$

Theorem 3.4. *Let (ℓ, st) be a stable input and C_ℓ is the labeling constraint of ℓ computed via $AttCon_{st}$. With $\mathbb{F}$ we denote the set of argumentation frameworks that satisfy C_ℓ, i.e for all $F \in \mathbb{F}$ we have that $F \models C_\ell$.*

The semantic constraint function $AttCon_{st}$ is sound *for stable input labelings ℓ, i.e.,*

$$\forall F \in \mathbb{F} : \ell \textit{ is a } \mathrm{st}\textit{-labeling of } F$$

and the semantic constraint function $AttCon_{st}$ is complete *for stable input labelings ℓ, i.e.,*

$$\forall F = (\mathsf{Arg}, \mathsf{R}) : \ell \textit{ is a } \mathrm{st}\textit{-labeling of } F \implies F \in \mathbb{F}.$$

Proof of Theorem 3.4 (Soundness). Let (ℓ, st) be a stable input and Arg denotes the set of arguments. $C_\ell = \{C_a = AttCon_{st}(a, \ell)\}_{a\in\mathsf{Arg}}$ is the set of attack constraints computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C_ℓ.

We proof by contradiction that the attack constraints and thus the algorithm are sound for stable input labelings. Assume there exists an argumentation framework $F = (\mathsf{Arg}, \mathsf{R}') \in \mathbb{F}$ which is not consistent with the labeling ℓ. Then, one of the following three cases must apply:

(1) There is an attack between two in-labeled arguments a and b in F.
(2) There is an out-labeled argument a which is not attacked by any in-labeled argument b in F.
(3) There is an argument a which is not labeled in and is also not attacked by any in-labeled argument.

Cases 1 and 2 are the same as for the conflict-free semantics. So, we will only look at the third case here.

Case 3: There is an argument a which is not labeled in and is also not attacked by any in-labeled argument. We know that F satisfies all attack constraints in C. Every attack constraint $C_a \in C$ is defined as $C_a = AttCon_{st}(a, \ell)$. Per Definition 3.10 the attack constraint for a is then either $C_a = \bigwedge_{b\in\mathsf{in}(\ell)} \neg r_{ba}$ or $C'_a = \bigvee_{b\in\mathsf{in}(\ell)} r_{ba}$. In the first case it follows for any model $\mathcal{A}$ of C_a that $\mathcal{A}(r_{ba}) = \mathit{true}$ for all argument $b \in \mathsf{in}(\ell)$. However, this means a can only be attacked by out labeled arguments and thus a must be part of any stable labeling of F.

For the second case with the attack constraint C'_a it holds for every model $\mathcal{A}$ that there exists some argument $b \in \mathsf{in}(\ell)$ such that $\mathcal{A}(r_{ba}) = \mathit{true}$. Thus, a is always attacked by some in-labeled argument and has to be labeled out.

To summarize, there can be no undec-labeled argument in an argumentation framework F that is consistent with the attack constraints C computed for a stable labeling ℓ. Thus, it follows that every $F \in \mathbb{F}$ must be consistent with the input labeling ℓ and the algorithm is *sound* for stable input labelings. □

Proof of Theorem 3.4 (Completeness). Let (ℓ, st) be a stable input and Arg denotes the set of arguments. $C_\ell = \{C_a = AttCon_{st}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of attack constraints computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C_ℓ.

We proof that the algorithm is complete for stable input labelings ℓ, i.e., for every argumentation framework F it holds that, if F is consistent with ℓ then $F \in \mathbb{F}$.

Assume there exists an argumentation framework $F = (\mathsf{Arg}, \mathsf{R}')$ which is consistent with the labeling ℓ and $F \notin \mathbb{F}$.

Then, according to Definition 2.6 the following two conditions hold for all arguments $a \in \mathsf{Arg}$ in F:

(1) If $a \in \mathsf{in}(\ell)$, then $\forall b \in \mathsf{Arg} : (b, a) \in \mathsf{R}' \rightarrow b \in \mathsf{out}(\ell)$.
(2) If $a \in \mathsf{out}(\ell)$, then $\exists b \in \mathsf{in}(\ell) : (b, a) \in \mathsf{R}'$.

This proof is rather trivial. From the proofs for conflict-free and admissible semantics we know that any F that satisfies the above conditions for in- and out-labeled must also satisfy the attack constraints of these arguments. Since ℓ is a stable labeling, there is no undec-labeled argument and thus we do not need to proof anything else.

That means, if the argumentation framework F is consistent with the stable labeling ℓ it also satisfies the attack constraints for all arguments as defined in Definition 3.10. Thus, it holds for all stable labelings that, if F is consistent with ℓ, then $F \in \mathbb{F}$. □

Theorem 3.6. *Let* Arg *be a set of arguments and L_1, L_2 are sets of inputs. $\mathbb{F}_1$, $\mathbb{F}_2$ denote the sets of argumentation frameworks constructed by Algorithm 1 for the input labelings in L_1 and L_2 respectively.*

The algorithm for constructing argumentation frameworks from labelings is monotonically refining, *i.e.,*

$$\forall L_1, L_2 \in \mathbb{L}(\mathsf{Arg}) : L_1 \supseteq L_2 \implies \mathbb{F}_1 \subseteq \mathbb{F}_2$$

Theorem 3.5. *Let L be a set of inputs and $\mathbb{F}$ is the set of argumentation frameworks constructed by the procedure described in Algorithm 1 by iteratively processing L. For every σ_i in the set of inputs there exists a sound and complete semantic constraint function, i.e., $\sigma_i \in \{cf, ad, co, st\}$.*

The algorithm is sound *for all input labelings, i.e.,*

$$\forall F \in \mathbb{F} : \forall (\ell_i, \sigma_i) \in L : \ell \textit{ is a } \sigma_i\textit{-labeling of } F$$

The algorithm is complete *for all input labelings, i.e.,*

$$\forall G = (\mathsf{Arg}, \mathsf{R}) : \big(\forall (\ell_i, \sigma_i) \in L : \ell \textit{ is a } \sigma_i\textit{-labeling of } G\big) \implies G \in \mathbb{F}$$

Proof of Theorem 3.5. It follows from Theorems 3.1, 3.2, 3.3 and 3.4 that the attack constraints are sound and complete for labelings with respect to conflict-free, admissible, complete and stable semantics. We need to prove that a set of attack constraints, and thus Algorithm 1, is sound and complete for any combination of input labelings with respect to different semantics. This follows from the fact that combining the attack constraints of each labeling is done by conjunction as shown below:

Let L be a set of inputs and $\mathbb{F}$ be the set of argumentation frameworks constructed by Algorithm 1 for the inputs L.

Consider two inputs $(\ell_1, \sigma_1), (\ell_2, \sigma_2) \in L$ with respect to different semantics. The corresponding attack constraints for any argument a are denoted C_{a,ℓ_1} and C_{a,ℓ_2}. The attack constraint for a in the algorithm would then be computed as $C_a = C_{a,\ell_1} \wedge C_{a,\ell_2}$. The models of C_a are then $\mathcal{M}_a = \mathcal{M}_{a,\ell_1} \cap \mathcal{M}_{a,\ell_2}$. We know that any argumentation framework $F \in \mathbb{F}$ satisfies C_a. Thus, it follows that F must also satisfy both C_{a,ℓ_1} and C_{a,ℓ_2}. We have proven that the attack constraints are sound and complete for labelings with respect to a semantics $\sigma \in \{cf, ad, co, st\}$. Therefore, the soundness and completeness also holds for any combination of labelings with respect to those semantics. $\square$

Proof of Theorem 3.6 (Monotonicity). Let L_1, L_2 be sets of input labelings and Arg denotes the set of arguments. C_1, C_2 are sets of attack constraints computed for L_1 and L_2 while $\mathbb{F}_1$ and $\mathbb{F}_2$ denote the set of argumentation frameworks consistent with C_1 and C_2 respectively.

We proof that the algorithm defined in Section 3.3 is monotonically refining. Consider any argument a. Per definition the attack constraint for a with respect to L_2 is then $C_{a,2} = \bigwedge_{\ell_\sigma \in L_2} AttCon_\sigma(a, \ell)$. That means, the models of $C_{a,2}$ are then computed as the intersection of the models of the conditions for the individual labelings, i.e.,

$$\mathcal{M}_{a,2} = \bigcap_{\ell_\sigma \in L_2} \mathcal{M}\big(AttCon_\sigma(a, \ell)\big).$$

Similarly, the set of all models of the attack constraint $C_{a,1}$ for the argument a with respect to the labelings L_1 is then defined as

$$\mathcal{M}_{a,1} = \bigcap_{\ell_\sigma \in L_1} \mathcal{M}\big(AttCon_\sigma(a, \ell)\big).$$

We know that $L_2 \subseteq L_1$ and thus can also write $L_1 = L_2 \cup L'$, where $L' = L_1 \setminus L_2$ is the set of labelings in L_1 but not in L_2. Then, we may also write

$$\mathcal{M}_{a,1} = \bigcap_{\ell_\sigma \in L_2 \cup L'} \mathcal{M}\big(AttCon_\sigma(a, \ell)\big).$$

We can now split up this formula and write it as

$$\mathcal{M}_{a,1} = \bigcap_{\ell_\sigma \in L_2} \mathcal{M}\big(AttCon_\sigma(a, \ell)\big) \cap \bigcap_{\ell_\sigma \in L'} \mathcal{M}\big(AttCon_\sigma(a, \ell)\big).$$

The first part is then exactly the set of models $\mathcal{M}_{a,2}$ of $C_{a,2}$. In short, we can also write $\mathcal{M}_{a,1} = \mathcal{M}_{a,2} \cap \mathcal{M}'_a$, where $\mathcal{M}'_a$ corresponds to the second part of the above formula for $\mathcal{M}_{a,1}$. The models of the argument a for the labelings L_1 are computed as the intersection of the models $\mathcal{M}_{a,2}$ for the labelings L_2 intersected with the models $\mathcal{M}'_a$ for the labelings L'. That means, every model of $C_{a,1}$ is a model of $C_{a,2}$ and has to additionally satisfy the attack constraint for every labeling $\ell \in L_1 \setminus L_2$.

From that we can easily see: it holds for every argument $a \in \mathsf{Arg}$ that the models for L_1 must be a subset of the models for L_2 for every attack constraint, i.e., $\mathcal{M}_{a,1} \subseteq \mathcal{M}_{a,2}$. Since the models of an attack constraint correspond per definition exactly to the set of argumentation frameworks the algorithm constructs, it follows that $\mathbb{F}_1 \subseteq \mathbb{F}_2$.

Thus, we have shown for all sets of labelings L_1, L_2, that $\mathbb{F}_1 \subseteq \mathbb{F}_2$ if $L_2 \subseteq L_1$, i.e., the algorithm is monotonically refining. $\square$

References

[1] A. Adadi and M. Berrada, Peeking inside the black-box: A survey on explainable artificial intelligence (XAI), *IEEE access* **6** (2018), 52138–52160. doi:10.1109/ACCESS.2018.2870052.

[2] G. Alfano, A. Cohen, S. Gottifredi, S. Greco, F. Parisi and G. Simari, Dynamics in abstract argumentation frameworks with recursive attack and support relations, in: *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI'20)*, 2020.

[3] G. Alfano, S. Greco, F. Parisi and I. Trubitsyna, On the semantics of abstract argumentation frameworks: A logic programming approach, *Theory and Practice of Logic Programming* **20**(5) (2020), 703–718. doi:10.1017/S1471068420000253.

[4] P. Baroni, M. Caminada and M. Giacomin, Abstract argumentation frameworks and their semantics, *Handbook of formal argumentation* **1** (2018), 157–234.

[5] P. Baroni and M. Giacomin, Solving semantic problems with odd-length cycles in argumentation, in: *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, Springer, 2003, pp. 440–451.

[6] S. Bistarelli, L. Kotthoff, F. Santini and C. Taticchi, A first overview of ICCMA'19, in: *AI^3@ AI*IA*, 2020, pp. 90–102.

[7] M. Caminada, Semi-stable semantics, *COMMA* **144** (2006), 121–130.

[8] M.W. Caminada and D.M. Gabbay, A logical account of formal argumentation, *Studia Logica* **93**(2) (2009), 109–145. doi:10.1007/s11225-009-9218-x.

[9] C. Cayrol and M.-C. Lagasquie-Schiex, On the acceptability of arguments in bipolar argumentation frameworks, in: *Symbolic and Quantitative Approaches to Reasoning with Uncertainty: 8th European Conference, ECSQARU 2005*, Proceedings 8, Barcelona, Spain, July 6–8, 2005, Springer, 2005, pp. 378–389.

[10] F. Cerutti, M. Giacomin and M. Vallati, Generating structured argumentation frameworks: AFBenchGen2, in: *COMMA*, 2016, pp. 467–468.

[11] A. Cohen, S. Gottifredi, A.J. García and G.R. Simari, An approach to abstract argumentation with recursive attack and support, *Journal of Applied Logic* **13**(4) (2015), 509–533. doi:10.1016/j.jal.2014.12.001.

[12] K. Čyras, A. Rago, E. Albini, P. Baroni and F. Toni, Argumentative XAI: a survey, 2021, arXiv preprint arXiv:2105.11266.

[13] L.M. de Campos and J.G. Castellano, Bayesian network learning algorithms using structural restrictions, *International Journal of Approximate Reasoning* **45**(2) (2007), 233–254. doi:10.1016/j.ijar.2006.06.009.

[14] W.F. Dowling and J.H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, *The Journal of Logic Programming* **1**(3) (1984), 267–284. doi:10.1016/0743-1066(84)90014-1.

[15] P.M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, *Artificial intelligence* **77**(2) (1995), 321–357. doi:10.1016/0004-3702(94)00041-X.

[16] X. Fan and F. Toni, On computing explanations in argumentation, in: *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[17] H. Jakobovits and D. Vermeir, Robust semantics for argumentation frameworks, in: *Journal of Logic and Computation*, Vol. 9, OUP, 1999, pp. 215–261.

[18] H. Kido and B. Liao, A Bayesian approach to direct and inverse abstract argumentation problems, 2019, arXiv preprint arXiv:1909.04319.

[19] I. Kuhlmann, Towards eliciting attacks in abstract argumentation frameworks, Online handbook of argumentation for AI 2021, 27.

[20] J. Lawrence and C. Reed, Argument mining: A survey, *Computational Linguistics* **45**(4) (2020), 765–818. doi:10.1162/coli_a_00364.

[21] S. Muggleton, Inductive logic programming, *New generation computing* **8**(4) (1991), 295–318. doi:10.1007/BF03037089.

[22] J. Mumford, I. Sassoon, E. Black and S. Parsons, On the complexity of determining defeat relations consistent with abstract argumentation semantics, in: *Proceedings of COMMA 2022: 9th International Conference on Computational Models of Argument*, IOS Press, 2022.

[23] A. Niskanen, J. Wallner and M. Järvisalo, Synthesizing argumentation frameworks from examples, *Journal of Artificial Intelligence Research* **66** (2019), 503–554. doi:10.1613/jair.1.11758.

[24] G. Priest, Paraconsistent logic, in: *Handbook of Philosophical Logic*, Springer, 2002, pp. 287–393. doi:10.1007/978-94-017-0460-1_4.

[25] R. Riveret and G. Governatori, On learning attacks in probabilistic abstract argumentation, in: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 2016, pp. 653–661.

[26] M. Thimm, Tweety – a comprehensive collection of Java libraries for logical aspects of artificial intelligence and knowledge representation, in: *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR'14)*, 2014.

[27] M. Ulbricht and J.P. Wallner, Strong explanations in abstract argumentation, in: *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35, 2021, pp. 6496–6504.