UNIVERSITÄT KOBLENZ · LANDAU
Faculty 4: Computer Science

WeST People and Knowledge Networks
Institute for Web Science and Technologies

Learning Argumentation Frameworks from Labelings

Master's Thesis

in partial fulfillment of the requirements for
the degree of Master of Science (M.Sc.)
in Informatik

submitted by
Lars Bengel

First supervisor:	Prof. Dr. Matthias Thimm Artifical Intelligence Group FernUniversität in Hagen
Second supervisor:	Dr. Tjitze Rienstra Department of Data Science and Knowledge Engineering Maastricht University

Koblenz, September 2021

Statement

I hereby certify that this thesis has been composed by me and is based on my own work, that I did not use any further resources than specified – in particular no references unmentioned in the reference section – and that I did not submit this thesis to another examination before. The paper submission is identical to the submitted electronic version.

	Yes	No
I agree to have this thesis published in the library.	☐	☐
I agree to have this thesis published on the Web.	☐	☐
The thesis text is available under a Creative Commons License (CC BY-SA 4.0).	☐	☐
The source code is available under a GNU General Public License (GPLv3).	☐	☐
The collected data is available under a Creative Commons License (CC BY-SA 4.0).	☐	☐

....

(Place, Date) (Signature)

Note

- If you would like us to contact you for the graduation ceremony,
  please provide your personal E-mail address: ....
- If you would like us to send you an invite to join the WeST Alumni
  and Members group on LinkedIn, please provide your LinkedIn ID : ....

Zusammenfassung

Ein zentraler Aspekt der abstrakten Argumentation sind seit ihrer Einführung durch Dung im Jahr 1995 die Semantiken. Diese Semantiken befassen sich mit der Bestimmung von zulässigen Mengen von Argumenten, die bestimmte Kriterien erfüllen, welche auf der Struktur der Angriffen im Argumentationsgraphen basieren. Es gibt zwei verschiende Ansätze für die Semantiken zwischen denen unterschieden werden kann: Erweiterungen und Beschriftungen. Im Rahmen dieser Arbeit werden wir uns hauptsächlich mit beschriftungs-basierte Semantiken beschäftigen.
Ein Thema, welchhes bisher eher wenig Aufmerksamkeit erhalten hat, beschäftigt sich mit der entgegengesetzten Richtung der Semantiken: Die Konstruktion von Argumentationsgraphen aus einer gegebenen Menge von Beschriftungen mit dem Ziel, dass diese Argumentationsgraphen alle diese Beschriftungen erzeugen.
In dieser Masterarbeit wird ein neuartiger Ansatz zum Lernen von Argumentationgraphen von Beschriftungen vorgestellt. Ein wichtiges Element dieses Ansatzes ist es, *alle* Argumentationsgraphen zu berechnen, die die Beschriftungen erzeugen können, anstatt einfach nur einen beliebigen geeigneten Argumentationsgraphen zu finden. Dies ist zum Beispiel besonders wichtig, wenn wir zu einem späteren Zeitpunkt zusätzliche Beschriftungen erhalten und unser Ergebnis verfeinern wollen, ohne ganz von vorne beginnen zu müssen. Der entwickelte Algorithmus wird mit bestehenden Arbeiten zu diesem Thema verglichen und es wird eine Bewertung seiner Performanz durchgeführt.

Abstract

Since its introduction by Dung in 1995, a central aspect of abstract argumentation has been the argumentation semantics. These semantics are concerned with computing sets of arguments which satisfy certain criteria based on the structure of attacks in the argumentation framework. Two approaches can be distinguished for the output of argumentation semantics: extensions and labelings. In the scope of this work we will mainly focus on labeling-based argumentation semantics.
A topic which has not received as much attention so far is going in the opposite direction of semantics: Constructing argumentation frameworks from a given set of labelings such that these argumentation frameworks produce all input labelings.
This master thesis presents a novel approach for learning argumentation frameworks from labelings. An important element of this approach is computing *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 will be compared to the existing work on this topic and an evaluation of its performance has been conducted.

Contents

1 Introduction **1**
- 1.1 Scenario . . . 5
- 1.2 Research Question . . . 5
- 1.3 Methodology & Structure . . . 6

2 Background **7**
- 2.1 Abstract Argumentation Frameworks . . . 7
- 2.2 Extension-based Semantics . . . 8
- 2.3 Labeling-based Semantics . . . 11
- 2.4 Propositional Logic . . . 15

3 Related Work **17**
- 3.1 Learning Attacks from Labelings . . . 17
  - 3.1.1 Preliminaries . . . 17
  - 3.1.2 Algorithm . . . 19
- 3.2 Synthesizing Argumentation Frameworks from Examples . . . 21
- 3.3 Eliciting Argumentation Frameworks . . . 23
- 3.4 A Bayesian Approach to the Inverse Problem . . . 24

4 Learning Argumentation Frameworks **25**
- 4.1 Scenarios . . . 25
  - 4.1.1 Type of Input . . . 25
  - 4.1.2 Positive and Negative Examples . . . 26
  - 4.1.3 Knowledge about Sub-Frameworks . . . 26
  - 4.1.4 Type of Algorithm . . . 26
- 4.2 Basic Definitions . . . 27

5 Learning Argumentation Frameworks from Labelings **29**
- 5.1 Requirements . . . 29
- 5.2 General Approach . . . 29
- 5.3 Acceptance Conditions . . . 31
  - 5.3.1 Conflict-free Semantics . . . 33
  - 5.3.2 Admissible Semantics . . . 34
  - 5.3.3 Complete Semantics . . . 36
  - 5.3.4 Stable Semantics . . . 38
  - 5.3.5 Properties . . . 38
- 5.4 Algorithm . . . 40
  - 5.4.1 Combining Acceptance Conditions . . . 41
  - 5.4.2 Constructing Argumentation Frameworks from Conditions . . . 42
  - 5.4.3 Iterative Approach . . . 45
  - 5.4.4 Properties . . . 46
- 5.5 Example . . . 48

6 Discussion **53**
6.1 Acceptance Conditions 56

7 Evaluation **59**
7.1 Experiment Setup 59
7.2 Experiment 1 59
7.2.1 Results 61
7.3 Experiment 2 64
7.3.1 Results 65
7.4 Experiment 3 68
7.4.1 Results 69

8 Future Work **72**
8.1 Open Questions 72
8.2 Extending the Approach 72
8.3 Eliciting Argumentation Frameworks From Labelings 74

9 Conclusion **77**

A Proofs **78**
A.1 Conflict-free Semantics 78
A.2 Admissible Semantics 80
A.3 Complete Semantics 82
A.4 Stable Semantics 84
A.5 Monotonicity 86

B Contents of the attached CD **88**

1. Introduction

Since its introduction by Dung in 1995 the research field of abstract argumentation [9] has received great attention in the literature. An abstract argumentation framework is defined as a graph, where the nodes are arguments and the edges represent attacks between arguments (see Figure 2). 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 use this model for knowledge reasoning.

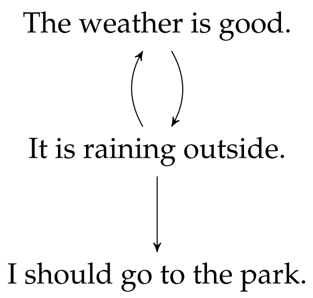


Figure 1: A hypothetical argumentation scenario regarding the weather.

Example 1.1. *Consider the three statements in Figure 1.* The weather is good *and* It is raining outside *are both observations about the current weather conditions that we can make. It is also clear that these two statements directly contradict each other. If it rains, we would not consider the weather to be good and vice versa. Thus, in the context of argumentation we consider these statements, also called arguments, to both be attacking each other. The third statement is* I should go to the park. *Now, we would probably only go to the park if the weather is good. That means, if it rains we would rather stay inside. For that reason we say that the argument* It is raining outside *attacks the argument* I should go to the park.
This structure then enables us to reason with these arguments. For example, if we observe that the weather is good, it follows from our model that we must reject the statement It is raining outside. *That would then allow us to conclude that we should go to the park. We can then also say that the set containing these two arguments is an extension of the above argumentation framework. On the other hand, if we observe that it is raining we would reject both other arguments and rather stay inside.*
In abstract argumentation we would ignore the underlying statements and reason only with the attacks and abstract arguments.

Building on Dung's idea, there are many extensions of the original approach that have since been proposed. Some of them include the addition of a support relation [8] or adding weights to the attacks between the arguments [10]. Another area which has received increased attention are the argumentation semantics. A semantics is a function which, given an abstract argumentation framework, computes sets of arguments (called extensions) which are consistent or acceptable with respect to the attack relation of the framework. That means for example, that there is no conflict between arguments of the extension and it counterattacks all arguments that attack some argument in the extension (see Example 1.2). While Dung proposed several different semantics in his seminal paper, like admissible, complete or stable semantics, there have since been many proposals for new semantics [1], all with different goals in mind.

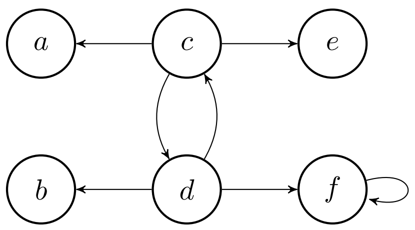


Figure 2: An abstract argumentation framework.

Example 1.2. *In Figure 2 we have an argumentation framework consisting of six arguments. For example, we have the argument A_3 and A_4 that are attacking each other and also the argument A_6 attacking itself. We can also see that the argument A_1 is attacked by A_3, but it is defended by A_4, since A_4 attacks A_3. One of the semantics proposed by Dung is the complete semantics, which states that the extension has to be conflict-free and it must contain exactly the arguments that are defended by it. In this example we have three complete extensions. The first complete extension is $\{A_2, A_3\}$. A_3 is only attacked by A_4 and defends itself while A_2 is also attacked by A_4 and thus defended by A_3. Even though A_6 is also defended by A_3 from A_4 it is not in the extension since the self-attack means it is not conflict-free. The other complete extensions in this argumentation framework are the empty set and $\{A_1, A_4, A_5\}$.*

The goal of all argumentation semantics is finding extensions within an argumentation framework. 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 rejected. However, we do not know for what reason these arguments are rejected. They could be directly attacked by some acceptable argument, meaning the extension actively contradicts them. But they could also be rejected 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 [13]. 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.

Example 1.3. *The argumentation framework in Figure 2 has the three complete extensions $\emptyset$, $\{A_2, A_3\}$ and $\{A_1, A_4, A_5\}$. For example, if we take the extension $\{A_2, A_3\}$, we can construct the corresponding labeling if we also know the argumentation framework. The arguments A_2 and A_3 are labelled* in, *because they are in the extension. The arguments A_1, A_4 and A_5 are labelled* out, *since they are attacked by A_3 and finally, A_6 gets the label* undec *because it is self-attacking and not attacked by an* in*-argument. So, the complete labeling corresponding to this extension is:* $\{\mathsf{in} : \{A_2, A_3\}, \mathsf{out} : \{A_1, A_4, A_5\}, \mathsf{undec} : \{A_6\}\}$.

So far we only looked at the process of having an argumentation framework and computing extensions or labelings from them. However, there also exists the possibility of starting with a set of extensions or labelings with respect to some semantics and trying to construct an argumentation framework, which is consistent with them. This research topic has not received much attention as of yet and therefore only few approaches to this problem exist, e.g. [20] or [17].

In their paper Niskanen et al. [17] 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 produces all or most of the positive examples and the least of the negative examples.

On the other hand Riveret et al. [20] propose an approach for learning attacks from labelings. Their algorithm is built around the grounded semantics and takes a stream of labelings as input. The grounded extension can be defined as the minimal complete extension of the argumentation framework. An argumentation framework always has exactly one grounded extension. Since it is not possible to reconstruct the argumentation framework by only knowing the grounded extension Riveret et al. decided to come up with an additional idea. Their algorithm works with knowledge about the sub-frameworks of the argumentation framework. A sub-framework means we only consider a subset of the arguments and the attacks between them. So, the input stream 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 some rules. 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.

Both of these algorithms address the above described problem, but they only return a single argumentation framework as a result. This might not always be desirable, as illustrated in Example 1.4. Assume a situation where we are communicating with some entity. This entity has some argumentation framework, hidden to us, which is used in its reasoning process. That means if we ask the entity a question it will return a labeling corresponding to some acceptable position with respect to some semantics. We can request as many labelings as we want, limited only by the number of valid labelings that exist in the hidden argumentation framework, and thus can accumulate a collection of labelings. If our goal is to understand the reasoning of this entity, we will have to reconstruct the underlying argumentation framework using the previously obtained collection of labelings.

Example 1.4. *We are given the complete labelings* $\{\mathsf{in} : \{A_1, A_4\}, \mathsf{out} : \{A_2, A_3\}, \mathsf{undec} : \emptyset\}$ *and* $\{\mathsf{in} : \{A_3\}, \mathsf{out} : \{A_1\}, \mathsf{undec} : \{A_2, A_4\}\}$ *with the goal of reconstructing the hidden argumentation framework from which they originated. A trivial algorithm for this problem would be to iterate over all argumentation frameworks containing the given arguments and compute their complete labelings. If the two input labelings are labelings of one of them, we stop and return the argumentation framework. However, there are cases where multiple frameworks can produce these labelings and in those cases only returning one might not be desired. This is also the case in our example and two of those argumentation frameworks are shown in Figure 3. As we can see, both frameworks produce both of the above complete labelings.*

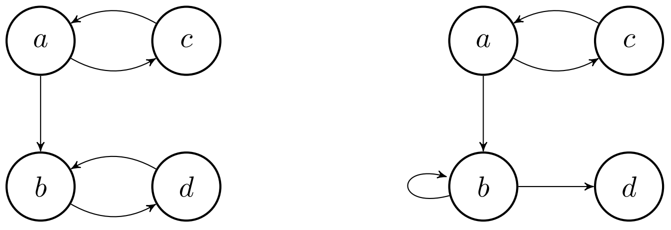


Figure 3: Two argumentation frameworks that can be reconstructed from the labelings in Example 1.4.

That means, by only returning one argumentation framework as the result, we make an additional assumption which can lead to unintentional discarding of the hidden argumentation framework that we are looking for. If we now obtain an additional labeling we would no longer be able to correctly reconstruct the hidden argumentation framework of the entity. For that reason an algorithm for learning argumentation frameworks from labelings should return all possible results after a learning step.

1.1. Scenario

In Example 1.4 we have seen that it might not be enough to learn a single argumentation framework for a given set of input labelings. Instead, for a more accurate result, we should learn a set of argumentation frameworks. More specifically, given a set of labelings we want to find all argumentation frameworks from which these labelings can be computed. So, at any point during the algorithm we want to keep all information that has been provided to us in the form of labelings. If we then obtain additional information, i.e. additional labelings, we can refine our result instead of having to start all over again.
Before formulating the exact scenario addressed in this thesis, we introduce the notion of *produces* (see also Definition 4.3). We say that an argumentation framework F *produces* a set of labelings L, if all labelings in L are labelings of F. That means the labelings of F are a superset of L, i.e. F can still have additional labelings.
Formally, the scenario addressed in this master thesis can then be described as follows:

Input: A set of arguments Arg, a set of labelings L.

Output: The set of argumentation frameworks $\mathbb{F}$ that produce the input labelings L.

Requirements: The input labelings can be in the form of an input stream, which means the algorithm should be able to process them iteratively. The algorithm should always retain all of the provided information.

1.2. Research Question

In the following we will formulate the central research question that shall be answered in this master thesis. This research question is concerned with the scenario described above.

RQ1: How can we effectively compute the set of all argumentation frameworks that produce a given set of labelings?
When given a set of labelings we can try to reconstruct an argumentation framework that produced these labelings. Depending on the input labelings there can be more than one argumentation framework that fulfills this condition. Just reconstructing one of these argumentation frameworks means we make some additional assumptions and lose information. It shall be investigated under which circumstances we are able to reconstruct all argumentation frameworks that produce the input labelings. For the investigation we will consider the semantics introduced by Dung [9] for the input labelings. With the obtained information an algorithm shall be created that is able to construct the set of argumentation frameworks which produce the set of input labelings.

1.3. Methodology & Structure

At the beginning of this work a literature review has been conducted in order to get an overview of the area of learning argumentation frameworks. Following that, a novel approach for learning argumentation frameworks was developed. This approach was then further refined and implemented. Finally, on the basis of this implementation an evaluation has been conducted to measure the performance of the developed approach.

This masters thesis is structured as follows: Section 2 will introduce the necessary background that is needed in the remainder of this work. Following that in Section 3, we have on overview of the related work in the area of learning argumentation frameworks. After that Section 4 summarizes the most important distinctions that can be made between the existing approaches for learning argumentation frameworks and introduces some basic definitions that are required in the later sections. The main contribution of this thesis is described in Section 5. Here, the concept of acceptance conditions is introduced and an algorithm that addresses the main research question is proposed. It follows a comparison of the proposed algorithm to the existing work in Section 6 as well as a closer look at the acceptance condition and the information they hold. The proposed algorithm has subsequently been implemented and an evaluation of this implementation has been conducted (see section 7). Furthermore, Section 8 gives an overview of some of the possible future extensions to the presented idea. Finally, it follows a conclusion in Section 9 that summarizes this thesis.

2. Background

This section will introduce important concepts from the literature that are necessary for the remainder of this thesis. First, we will look at the notion of an abstract argumentation framework [9]. Afterwards all argumentation semantics [9] that are relevant in the scope of this thesis will be defined. These argumentation semantics are first defined on the basis of extensions. Following in subsection 2.3 is an alternative definition of these semantics based on labelings [19, 13, 7]. Finally, we have a brief overview of some basic concepts of propositional logic [12] which will also be needed later.

2.1. Abstract Argumentation Frameworks

The most important concept for this thesis is that of an *argumentation framework*. The following definitions are taken from Dung [9]. An argumentation framework consists of a set of arguments Arg and a binary relation R over Arg called *attack relation*. For the scope of this master thesis we consider the set of arguments to be finite. The idea of abstract argumentation is to reason on the basis of the relations between arguments without knowing their content.

Definition 2.1. *An* argumentation framework *(AF) is a directed graph $F = (\mathsf{Arg}, \mathsf{R})$ where Arg is the set of nodes and $\mathsf{R} \subseteq \mathsf{Arg} \times \mathsf{Arg}$ is a set of links between these nodes. The nodes represent abstract arguments and the directed links represent attacks from one argument to another. We denote by $\mathbb{F}$ the set of all argumentation frameworks.*

We say that an argument $a \in \mathsf{Arg}$ *attacks* another argument $b \in \mathsf{Arg}$ if $(a, b) \in \mathsf{R}$. In that case we may also write $a \rightarrow b$. This notation can also be extended onto sets of arguments $S \subseteq \mathsf{Arg}$. We consider that S *attacks* an argument $a \in \mathsf{Arg}$ if for some $b \in S$ we have that b *attacks* a. Alternatively, we say that a *attacks* S if there is some $b \in S$ such that a *attacks* b. Furthermore, $a \nrightarrow b$ denotes that a does not attack b.

An important concept in abstract argumentation is *defense*. Informally, we say a set of arguments S *defends* an argument a if and only if S attacks all attackers of a.

Definition 2.2. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. Then a set of arguments $E \subseteq \mathsf{Arg}$* defends *an argument $a \in \mathsf{Arg}$, if and only if for all $b \in \mathsf{Arg}$ with $(b, a) \in \mathsf{R}$ there is an argument $c \in E$ with $(c, b) \in \mathsf{R}$.*

Example 2.1. *Lets consider the argumentation framework in Figure 4. The argumentation framework consists of six arguments and eight attacks. Their are multiple occasions where arguments defend each other in the argumentation framework. For example, lets look at the arguments a and c. While a is not attacked by any other argument, the argument c is attacked by b and d. However, the argument b is attacked by a and thus the argument a defends c against the attack from b. Furthermore, the argument c defends itself against the attack from d by directly counterattacking d. Similarly the argument d is defended by a*

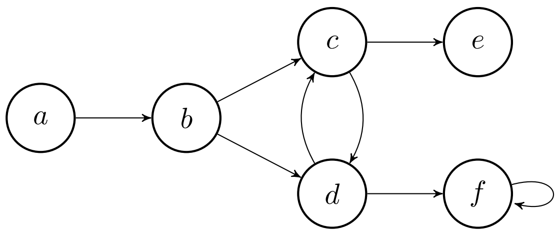


Figure 4: An argumentation framework.

against b and also defends itself against the attack from c. The argument e is attacked by c but it can be defended in two different ways, either by the argument b or d. The argument f can however not be defended since it attacks itself and thus every defender would have to attack and defend f at the same time, which is not possible.

2.2. Extension-based Semantics

This section will introduce the argumentation semantics that are considered in the scope of this master thesis. For that we will look at all the semantics proposed by Dung [9], namely *conflict-free*, *admissible*, *complete*, *grounded*, *preferred* and *stable* semantics. A semantics is defined as a function that, given an argumentation framework, returns a set of extensions. An extension is a set of arguments, i.e. the semantics returns a set of sets of arguments. Each extension then represents a position within the argumentation framework which satisfies certain conditions depending on the semantics.

The most basic concept that has been defined by Dung is the *conflict-free* property. We say that a set of arguments E is conflict-free if and only if there exists no conflict, i.e. there is no attack, between any of its elements. The idea of this property is simply that an extension should represent a consistent position and thus there should be no conflict between arguments of that position.

Definition 2.3. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. Then a set of arguments $E \subseteq \mathsf{Arg}$ is called* conflict-free*, if and only if for all $a, b \in E : (a, b) \notin R$. The set of* conflict-free *extensions of F is denoted by $\mathrm{cf}(F)$.*

Example 2.2. *Lets consider the argumentation framework in Figure 4. We are looking for the conflict-free extensions. A set of arguments is called conflict-free, if there is no attack between any of its arguments. Trivially the empty set is always conflict-free. A singleton set $\{x\}$ is conflict-free if x is not self-attacking. In our case the singleton sets $\{a\}$, $\{b\}$, $\{c\}$, $\{d\}$ and $\{e\}$ are thus conflict-free, while $\{f\}$ is not. Furthermore, the sets $\{a, c\}$, $\{a, d\}$, $\{a, e\}$, $\{b, e\}$, $\{d, e\}$ and $\{a, d, e\}$ are also conflict-free.*

We will also consider another basic property which builds on the above defined conflict-freeness. This property is called *admissibility*, which combines conflict-freeness with the concept of defense introduced in Definition 2.2. A set of arguments E is then called *admissible* if it is conflict-free and also defends all its arguments against attacks from outside arguments (see Definition 2.4). This means that all admissible extensions are per definition also conflict-free, while the reverse does not necessarily hold. While not technically considered to be argumentation semantics we will consider conflict-freeness and admissibility as such in the scope of this master thesis.

Definition 2.4. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. Then a set of arguments $E \subseteq \mathsf{Arg}$ is called* admissible, *if and only if E is conflict-free and E defends every argument $a \in E$. The set of* admissible *extensions of F is denoted by* $\mathrm{ad}(F)$.

Example 2.3. *We will again consider the argumentation framework in Figure 4. A set of arguments is called admissible if it is conflict-free and defends all its elements. That means, we will only have to check all conflict-free extensions in order to find the admissible ones. Trivially the empty set is also admissible. For the singleton sets we can easily see that only $\{a\}$ is admissible, since there is no attack on a at all. All other arguments have at least one attack that they do not defend themselves against. For example the arguments c and d. They both cannot defend against the attack from b. However, since a defends them against b the sets $\{a, c\}$ and $\{a, d\}$ are admissible. On the other hand, the conflict-free set $\{a, e\}$ is not admissible because the attack from c on e is undefended. The sets $\{b, e\}$ and $\{d, e\}$ are also not admissible. While both sets defend against the attack from c on e there is another attack which they do not defend against: the attack from a on b and the attack from b on e respectively. Finally, the set $\{a, d, e\}$ is admissible, as the addition of a defends d against b.*

Building on the idea of admissibility we can also define the *complete* semantics. A complete extension is always admissible, and thus also conflict-free, but in addition to that it has to contain exactly the arguments that it defends. The complete semantics can thus be considered as a stronger version of the admissible semantics.

Definition 2.5. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. Then a set of arguments $E \subseteq \mathsf{Arg}$ is called* complete, *if and only if E is admissible and for all $a \in \mathsf{Arg}$ if E defends a, then we have $a \in E$. The set of* complete *extensions of F is denoted by* $\mathrm{co}(F)$.

Example 2.4. *We are again considering the argumentation framework F in Figure 4. A complete extension is admissible and contains all arguments that are defended by it. That means we will only have to consider the admissible extension and check which arguments they defend. We start with the empty set. In our argumentation framework we have the unattacked argument a which means it is defended by any conflict free set. Thus, the empty set is not complete in F. The set $\{a\}$ is complete. While a defends c and d against the argument b both arguments have an additional attacker d and c respectively which a does not defend against. The set $\{a, c\}$ is also complete. However, the set $\{a, d\}$ is not complete. The reason for that is the argument e which is only attacked by c. Since d defeats c, the set $\{a, d\}$ does not contain all arguments defended by it. But this also means if we include e we obtain the third and last complete extension of the argumentation framework: $\{a, d, e\}$.*

Finally, using the above defined semantics we can define further semantics by imposing additional constraints on their extensions. First, we have the *preferred* and *grounded* semantics. The *preferred* extensions of an argumentation framework are defined as the admissible extensions that are also maximal with respect to set inclusion. However, it also holds that every preferred extension is complete.

Definition 2.6. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. Then a set of arguments $E \subseteq \mathsf{Arg}$ is called* preferred*, if and only if E is admissible and there exists no $E' \in \mathrm{ad}(F)$ with $E \subset E'$, i.e. E is maximal in $\mathrm{ad}(F)$ with respect to set inclusion. The set of* preferred *extensions of F is denoted by $\mathrm{pr}(F)$.*

Similarly, the *grounded* extension is defined as the minimal complete extension. There is always exactly one minimal extension which means the grounded extension of any argumentation framework is unique. The unique grounded extension is then a superset of the initial arguments of the argumentation framework, i.e. the arguments that are not attacked by any other argument in F.

Definition 2.7. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. Then a set of arguments $E \subseteq \mathsf{Arg}$ is called* grounded*, if and only if E is complete and there exists no $E' \in \mathrm{co}(F)$ with $E' \subset E$, i.e. E is minimal in $\mathrm{co}(F)$ with respect to set inclusion.*

The last semantics that is considered in this thesis is the *stable* semantics. A set of arguments E is called stable if it is complete and also attacks every other argument in the argumentation framework. That means there can be cases where an argumentation framework has no stable extensions. This happens in the presence of odd attack cycles in the framework (see Example 2.5). Every *stable* extension is per definition also complete, but it also holds that every *stable* extension is preferred. A full overview over the relations between different semantics can be found in Figure 5.

Definition 2.8. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. Then a set of arguments $E \subseteq \mathsf{Arg}$ is called* stable*, if and only if E is complete and for every argument $a \in \mathsf{Arg} \setminus E$ we have that E attacks a. The set of* stable *extensions of F is denoted by $\mathrm{st}(F)$. It should be noted that a stable extension does not always exist.*

Example 2.5. *Consider the argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$ in Figure 4. As we have determined already in Example 2.4, the complete extensions of this argumentation framework are: $\{a\}$, $\{a, c\}$ and $\{a, d, e\}$. Based on this, we can easily see that $\{a\}$ must be the grounded extension of F, since we have $\{a\} \subset \{a, c\}$ as well as $\{a\} \subset \{a, d, e\}$. On the other hand, it follows that $\{a, c\}$ and $\{a, d, e\}$ are the preferred extensions of F since there exists no complete superset of each of them.*

Finally, the only stable extension of F is $\{a, d, e\}$. We have that $\{a, d, e\}$ is complete and it attacks every argument in Arg that is not in it. On the other hand, $\{a, c\}$ is not stable, because it does not include the argument f and does also not attack it. The set $\{a\}$ cannot be stable, because it is not preferred.

$$\begin{tikzcd}
\text{Stable semantics} \arrow[dr] & & & \\
& \text{Preferred semantics} \arrow[dr] & & \text{Grounded semantics} \arrow[dl] \\
& & \text{Complete semantics} \arrow[d] & \\
& & \text{Admissible semantics} \arrow[d] & \\
& & \text{Conflict-free semantics} &
\end{tikzcd}$$

Figure 5: Relations between the argumentation semantics. $\sigma_1 \rightarrow \sigma_2$ means every σ_1-extension is also a σ_2-extension.

2.3. Labeling-based Semantics

Instead of using extensions, we can also consider the use of labelings for the semantics. This approach was first proposed by Pollock [19] as well as Jakobovits and Vreeswijk [13]. Later, Caminada [7] provided definitions for the above introduced semantics, which we will use in this thesis.

In contrast to extensions a labeling provides more information about the underlying argumentation framework. An extension only tells us the set of accepted arguments and implicitly the set of rejected arguments. On the other hand a labeling associates with each argument exactly one label: in, out or undec. The label in means that the argument is accepted, the label out indicates that an argument is actively rejected and the label undec means that the argument is undecided, i.e. it is neither accepted nor rejected.

Definition 2.9. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. A* labeling *is a total function $\ell : \mathsf{Arg} \rightarrow \{\mathsf{in}, \mathsf{out}, \mathsf{undec}\}$.*

In the following $\mathsf{in}(\ell)$ denotes the set of arguments that are labeled in. While $\mathsf{out}(\ell)$ denotes the set of arguments that are labeled out and $\mathsf{undec}(\ell)$ denotes the set of arguments that are labeled undec. Furthermore, $\mathbb{L}(\mathsf{Arg})$ denotes the set of all labelings over the set of arguments Arg.

It should be noted that there exists a correspondence between labelings and extensions. Given a labeling ℓ there is always exactly one extension associated with it. This extension is simply the set $\mathsf{in}(\ell)$. There also exists a relation for the other direction. However, the number of corresponding labelings for an extension depends on the semantics. In addition to that we would also need to know the argumentation framework, i.e. its attack relation, to compute the corresponding labelings for an extension. Otherwise it is not possible to decide between the labels out and undec for an argument. The following definitions are taken from [7].

We can now define a labeling-based approach for the above introduced semantics. The most basic form of labelings are the *conflict-free* labelings. A labeling is considered to be conflict-free if and only if there exists no attack between two in-labeled arguments in the argumentation framework.

Definition 2.10. *Let* $F = (\mathsf{Arg}, \mathsf{R})$ *be an argumentation framework. A* conflict-free *labeling* ℓ *is a labeling such that for every* $a \in \mathsf{Arg}$ *it holds that:*

- *if* a *is labeled* in *then no attacker of* a *is labeled* in*.*
- *if* a *is labeled* out *then* a *has an attacker that is labeled* in*.*
- *otherwise* a *is labeled* undec*.*

The next semantics we define in the context of labelings is the *admissible* semantics. For an admissible labeling we require the defense of all in-labeled arguments in addition to the conflict-freeness. This is achieved by adjusting the first part of the definition for conflict-free labelings. So for an admissible labeling we require an in-labeled argument to only have attackers with the label out (see Definition 2.11).

Caminada et al. have shown that there is a many-to-one relationship between admissible labelings and admissible sets. That means, each admissible set can be associated with one or more admissible labelings.

Definition 2.11. *Let* $F = (\mathsf{Arg}, \mathsf{R})$ *be an argumentation framework. An* admissible *labeling* ℓ *is a labeling such that for every* $a \in \mathsf{Arg}$ *it holds that:*

- *if* a *is labeled* in *then all attackers of* a *are labeled* out*.*
- *if* a *is labeled* out *then* a *has an attacker that is labeled* in*.*
- *otherwise* a *is labeled* undec*.*

For the *complete* labelings we have to strengthen the undec label. If a labeling is complete, then the restrictions for in and out are the same as in the admissible case. However, an argument is now only labeled undec if it is attacked by some other undec-labeled argument and is not attacked by any in-labeled argument. This change reflects the fact that a complete extension has to contain every argument that it defends (see Definition 2.12).

In the case of the complete semantics we have a one-to-one relationship between compete labelings and complete extensions. That means there is exactly one complete labeling that can be associated with a given complete extension. However, it is important to note that this requires us to know the attack relation of the argumentation framework. Without knowing the attack relation it is not possible to find the corresponding complete labeling for an extension. This is especially relevant in the context of this master thesis as we are working with a scenario in which we do not know the argumentation framework. That means if we are reconstructing argumentation frameworks from labelings or extensions these two forms of defining semantics are not equivalent.

Definition 2.12. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. A* complete *labeling ℓ is a labeling such that for every $a \in \mathsf{Arg}$ it holds that:*

- *if a is labeled* in *then all attackers of a are labeled* out*.*
- *if a is labeled* out *then a has an attacker that is labeled* in*.*
- *if a is labeled* undec *then it has at least one attacker that is labeled* undec *and it does not have an attacker that is labeled* in*.*

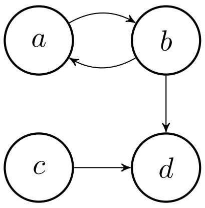


Figure 6: An argumentation framework F with four arguments.

Example 2.6. *Lets consider the argumentation framework F in Figure 6. We start by looking at the admissible labelings of F, which are depicted in Figure 7. ℓ_1, where all arguments are labeled* undec *corresponds to the empty set which is, as we know, always admissible. Furthermore, the singleton set $\{a\}$, $\{b\}$ and $\{c\}$ are also admissible. $\{a\}$ and $\{b\}$, with the corresponding labelings ℓ_2 and ℓ_3 both defend itself against each other. Thus, in ℓ_2 the argument b is labeled* out *and in ℓ_3 we have a labeled* out*. In both labelings the arguments c and d have the label* undec*. The argument c is unattacked in F and attacks only d. That gives us the labeling ℓ_4, with $\mathsf{in}(\ell_4) = \{c\}$. Finally, we also have the admissible labelings ℓ_5 and ℓ_6, with the associated admissible extensions $\{a, c\}$ and $\{b, c\}$ respectively. In both labelings, the remaining arguments have the label* out*.*

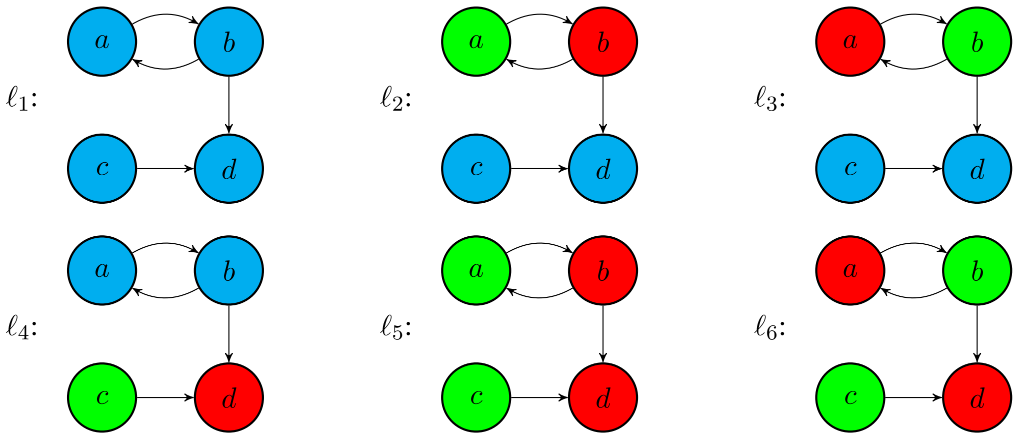


Figure 7: The six admissible labelings of F.

The argumentation framework F also has three complete labelings. Since c is unattacked, it has to be labeled in *in every complete labeling. From that, it follows that d will be labeled* out *in every complete labeling. The arguments a and b attack each other and have no other incoming attack. That leaves us with three possibilities on how to label them. They can both be undecided, or one of them is labeled* in *and the other must then be labeled* out*. That gives us the three complete labelings of F: ℓ_4, ℓ_5 and ℓ_6.*

Based on the complete labelings we can now define grounded, preferred and stable labelings. For this we will use definitions that are equivalent to those in [7]. The *grounded* labeling ℓ of an argumentation framework is defined as the complete labeling where $\mathsf{undec}(\ell)$ is maximal. This would be equivalent to demanding that $\mathsf{in}(\ell)$ or $\mathsf{out}(\ell)$ are minimal among the complete labelings. For the labeling-based semantics it also holds that the grounded labeling is unique for any argumentation framework.

Definition 2.13. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. A labeling ℓ is called* grounded *if it holds that:*

- *ℓ is a complete labeling and*
- $\mathsf{undec}(\ell)$ *is maximal with respect to set inclusion.*

Similarly we can define the *preferred* labelings. We say that a complete labeling ℓ is called preferred if it holds that $\mathsf{in}(\ell)$ is maximal with respect to set inclusion among the complete labelings.

Definition 2.14. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. A labeling ℓ is called* preferred *if it holds that:*

- *ℓ is a complete labeling and*
- $\mathsf{in}(\ell)$ *is maximal with respect to set inclusion.*

Finally, the *stable* labelings can also be defined as a special case of complete labelings. A complete labeling ℓ is called stable if $\mathsf{undec}(\ell)$ is empty. That means all arguments in the argumentation framework are either labeled in (accepted) or they are labeled out (attacked by accepted argument), which is exactly how the stable semantics has been defined by Dung [9].

Furthermore, just like it is the case with the extension-based semantics every stable labeling is also a preferred labeling. All in all, the same relations between the semantics also hold for the labeling-based approach (see Figure 5).

Definition 2.15. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. A labeling ℓ is called* stable *if it holds that:*

- *ℓ is a complete labeling and*
- $\mathsf{undec}(\ell) = \emptyset$.

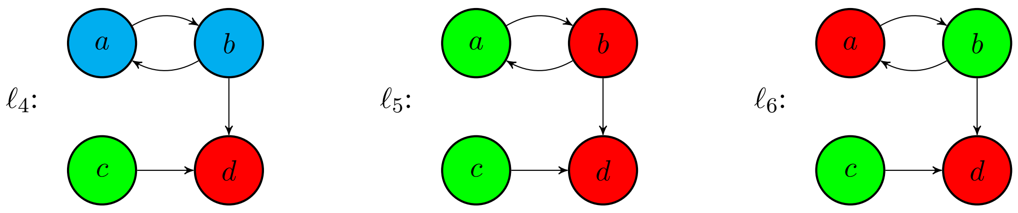


Figure 8: The complete labelings of F. ℓ_4 is also grounded, while ℓ_5 and ℓ_6 are preferred and stable.

Example 2.7. *We consider again the argumentation framework in Figure 6 with its three complete labelings ℓ_4, ℓ_5 and ℓ_6 as shown in Figure 8*

A labeling ℓ is grounded, if it is complete and it holds that $\mathsf{undec}(\ell)$ *is maximal. For the three labelings we have that* $\mathsf{undec}(\ell_4) = \{a, b\}$ *and* $\mathsf{undec}(\ell_5) = \mathsf{undec}(\ell_6) = \emptyset$*. Thus, ℓ_4 is the unique grounded labeling of F.*

For the preferred semantics we take the complete labelings where $\mathsf{in}(\ell)$ *is maximal. This is the case for ℓ_5 as well as ℓ_6.*

A labeling ℓ is called stable, if it is complete and we have that $\mathsf{undec}(\ell) = \emptyset$*. As noted before, this is the case for the labelings ℓ_5 and ℓ_6 which means these are exactly the stable labelings of F.*

2.4. Propositional Logic

This section will describe the necessary concepts from propositional logic [12] and introduce the terminology used in this work. A *propositional calculus* is defined as a formal system $PL = (\mathsf{At}, \Omega)$. This includes a set of atoms At and a set of logical operators Ω. To define a propositional calculus we only need the basic logical operators $\top$, $\bot$, $\neg$, $\land$ and $\lor$. Based on these we can also define some additional operators: the implication $p \to q := \neg p \lor q$ and the equivalence $p \leftrightarrow q := (p \to q) \land (q \to p)$.

Definition 2.16. *A* propositional calculus *is formal system $PL = (\mathsf{At}, \Omega)$, where* At *is a propositional signature (a set of atoms) and $\Omega = \{\top, \bot, \neg, \land, \lor\}$ is the set of logical operators. Then we define the propositional language $\mathcal{L}(\mathsf{At})$ or the set of formulas of PL as follows:*

1. *$\top$ and $\bot$ are formulae.*
2. *every $p \in$ At is a formula.*
3. *if p is a formula, then $\neg p$ is a formula.*
4. *if p and q are formulae, then $\circ(p, q)$ is a formula (with $\circ \in \{\land, \lor, \to, \leftrightarrow\}$).*

Furthermore, we define the *interpretation* $\mathcal{A}$ (see Definition 2.17) of a set of atoms $\mathcal{A}$ as a function that assigns to each atom $p \in$ At a truth value. We also define the *satisfaction relation* $\models$. We say an interpretation $\mathcal{A}$ satisfies a formula f over At ($\mathcal{A} \models f$ in short) if $\mathcal{A}$ assigns the value $true$ to the formula (see Definition 2.18). A formula f is then called *satisfiable* if there exists an interpretation $\mathcal{A}$ with $\mathcal{A} \models f$. Otherwise it is called *unsatisfiable*.

Definition 2.17. *Let* At *be a propositional signature. An* interpretation $\mathcal{A}$ *is a function* $\mathcal{A} : \mathsf{At} \to \{true, false\}$. Int_{At} *denotes the set of all interpretations of* At.

Definition 2.18. *Let* At *be a propositional signature,* $\mathcal{A} \in Int_{At}$ *is an interpretation and* $f, g \in \mathcal{L}(\mathsf{At})$ *are formulae. Then we define the* satisfaction relation $\models$ *recursively as follows:*

$\mathcal{A} \models f \wedge g$	*iff* $\mathcal{A} \models f$ *and* $\mathcal{A} \models g$
$\mathcal{A} \models f \vee g$	*iff* $\mathcal{A} \models f$ *or* $\mathcal{A} \models g$
$\mathcal{A} \models \neg f$	*iff* $\mathcal{A} \not\models f$
$\mathcal{A} \models f$	*iff* $f \in \mathsf{At}$ *and* $\mathcal{A}(f) = true$

Finally, we define the set of models $\mathcal{M}(f)$ of a formula f. A *model* is an interpretation $\mathcal{A}$, such that $\mathcal{A} \models f$.

Definition 2.19. *Let* At *be a propositional signature and* $f \in \mathcal{L}(\mathsf{At})$ *is a formula. The set of models* $\mathcal{M}(f)$ *of* f *is defined as:*

$$\mathcal{M}(f) = \{\mathcal{A} \in Int_{At} \mid \mathcal{A} \models f\}$$

Example 2.8. *Let* $\mathsf{At} = \{P, Q, R, S, T\}$ *be a propositional signature. We consider the corresponding formal system* $PL = (\mathsf{At}, \{\top, \bot, \neg, \wedge, \vee\})$.

Based on this we can then define some formulae $f, g \in \mathcal{L}(\mathsf{At})$*, for example:*

$$f = (P \vee \neg Q) \wedge (\neg P \wedge Q)$$

$$g = (P \vee Q \vee R) \wedge (P \vee \neg R) \wedge \neg Q$$

We can now compute the interpretations of f *and* g *as described in Definition 2.18. For the formula* f*, we have* $\mathcal{A}(f) = \mathcal{A}(P \vee \neg Q) \wedge \mathcal{A}(\neg P \wedge Q)$*. Going further, the models of the first subformula are then defined as* $\mathcal{A}(P \vee \neg Q) = \mathcal{A}(P) \vee \mathcal{A}(\neg Q)$*. That means, either* P *has to be true or* Q *has to be false to satisfy this subformula. Conversely, in order to satisfy the second subformula* P *has to be false and* Q *has to be true. Since these formulae are connected via a conjunction we obtain a contradiction and thus,* f *is unsatisfiable.*

Consider the formula g*. We may regard this as a conjunction of three subformulae* $g_1 = (P \vee Q \vee R)$, $g_2 = (P \vee \neg R)$ *and* $g_3 = \neg Q$*. We can then compute the models of each of these:* $\mathcal{M}(g_1)$, $\mathcal{M}(g_2)$ *and* $\mathcal{M}(g_3)$*. Since* g *is a conjunction, we may then obtain the set of models of* g *as the intersection of the models of the subformulae:* $\mathcal{M}(g) = \mathcal{M}(g_1) \cap \mathcal{M}(g_2) \cap \mathcal{M}(g_3)$*. This would then give us the models* $\mathcal{M}(g) = \{[P \to true], [P, R \to true]\}$*. Thus,* g *is satisfiable.*

3. Related Work

In this section we will take a look at some related work. All of these works describe a scenario that is similar to the one described in this thesis. First, we take a closer look at the scenario of learning attacks from labelings introduced by Riveret et al. [20]. They also provide an algorithm that addresses this scenario. After that we have a look at the problem of argumentation framework synthesis introduced by Niskanen et al. [17] and the associated algorithm that they defined. Both approaches have some similarities to what is done in this thesis, but also differ in some important aspects (For a detailed comparison of these algorithms see Section 6).
Following that, we take a look at the scenario of argumentation framework elicitation [15]. This can be considered as a more advanced version of learning argumentation frameworks where we are able to ask questions to an agent. The goal here is trying to minimize the number of questions that have to be asked to obtain the original argumentation framework.
Finally, we also consider an approach by Kido et al. [14]. They call the scenario of constructing argumentation frameworks from extensions the inverse problem and propose a bayesian approach to address this problem.

3.1. Learning Attacks from Labelings

In 2016 Riveret and Governatori [20] introduced an approach for learning attacks from labelings. Their algorithm works with grounded 4-valued labelings as input. Internally it uses rules based on the input labelings to update a weighted argumentation framework. It then returns an argumentation framework such that every attack satisfies a certain weight threshold. The following definitions are taken from [20] and adapted to suit the terminology and notion used in this thesis. Before describing the algorithm, we begin with some preliminary definitions.

3.1.1. Preliminaries

The Riveret algorithm uses 4-valued grounded labelings, for that we need the concept of sub-frameworks. A sub-framework G of an argumentation framework F is defined as a subset of the arguments $\mathsf{Arg}_G \subseteq \mathsf{Arg}$ together with all attacks of F which are between two arguments of Arg_G.

Definition 3.1. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. Then, we define the* sub-framework *G of F as an argumentation framework $G = (\mathsf{Arg}_G, \mathsf{R}_G)$, where $\mathsf{Arg}_G \subseteq \mathsf{Arg}$ and $\mathsf{R}_G = \mathsf{R} \cap \mathsf{Arg}_G \times \mathsf{Arg}_G$.*

We then define the 4-valued labeling which assigns to each argument of an argumentation framework one of the labels: $\mathsf{in}, \mathsf{out}, \mathsf{undec}, \mathsf{off}$. The first three labels are the same as the standard labelings and the label off is assigned to arguments which are ignored by the semantics.

Definition 3.2. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. A* 4-valued labeling *is a total function $\ell : \mathsf{Arg} \rightarrow \{\mathsf{in}, \mathsf{out}, \mathsf{undec}, \mathsf{off}\}$.*

The Riveret algorithm uses grounded 4-valued labelings. These labelings can also be described as grounded labelings of sub-frameworks. For that, the label off is assigned to all arguments which are not part of the sub-framework that is considered. All other arguments are then labeled according to the grounded labeling of the sub-framework G.

Definition 3.3. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and G and induced sub-framework of F. A grounded 4-valued labeling of F is a labeling, such that for all arguments $a \in \mathsf{Arg}$ it holds that:*

- *a is labeled according to the grounded labeling of* G.
- *a is labeled* off *iff $a \in \mathsf{Arg} \setminus \mathsf{Arg}_G$.*

Internally, the algorithm works with weighted argumentation frameworks. A weighted argumentation framework F is defined as an argumentation framework together with a function $w : \mathsf{R} \rightarrow \mathbb{R}$ which assigns to each attack in F a weight.

Definition 3.4. *A weighted argumentation framework* [10] *is a triple $(\mathsf{Arg}, \mathsf{R}, w)$ such that $(\mathsf{Arg}, \mathsf{R})$ is an argumentation framework and $w : \mathsf{R} \rightarrow \mathbb{R}$ is a function assigning weights to every attack.*

We also consider the following notion to describe the attacks in a weighted argumentation framework based on their weight. This will later be used by the algorithm to decide which attacks are necessary to produce the input labelings and which attacks have to be discarded for that:

Definition 3.5. *Let $\mathsf{F} = (\mathsf{Arg}, \mathsf{R}, w)$ be a weighted argumentation framework.*
Any attack $(a, b) \in \mathsf{R}$ is considered:

- discarded, *if $w_{(a,b)} < 0$*
- confirmed, *if $w_{(a,b)} > 0$*
- undecided, *if $w_{(a,b)} = 0$*

3.1.2. Algorithm

In this section we take a closer look at the actual algorithm proposed by Riveret et al. to learn argumentation frameworks from grounded 4-valued labelings. First, we define the scenario that is addressed by the algorithm as follows:

Input: set of arguments Arg, set of grounded 4-valued labelings L.

Output: some argumentation framework F, such that F produces as many labelings $\ell \in L$ as possible.

Internally, the algorithm will learn the labelings one by one. For each labeling ℓ the weights of an internal weighted argumentation framework will be adjusted. This is done according to the following four rules:

Definition 3.6. *Let $F = (\mathsf{Arg}, \mathsf{R}, w)$ be a weighted complete argumentation framework and ℓ be any grounded labeling of F. For any pair of arguments $a, b \in \mathsf{Arg}$:*

Rule 1. If $a, b \in \mathsf{in}(\ell)$, then

$$w_{(b,a)} := w_{(b,a)} - 1$$

Rule 2. If $a \in \mathsf{in}(\ell)$ and $b \in \mathsf{undec}(\ell)$, then

$$w_{(b,a)} := w_{(b,a)} - 1$$
$$w_{(a,b)} := w_{(a,b)} - 1$$

Rule 3. If $a, b \in \mathsf{undec}(\ell)$ and for any possible attacker c (with $c \neq a$ and $c \neq b$) of a or b, we have $c \in \mathsf{out}(\ell)$ or $c \in \mathsf{off}(\ell)$, then

$$w_{(b,a)} := w_{(b,a)} + 1$$
$$w_{(a,b)} := w_{(a,b)} + 1$$

Rule 4. If $a \in \mathsf{out}(\ell)$, $b \in \mathsf{in}(\ell)$ and for any possible attacker c (with $c \neq a$ and $c \neq b$) of a, we have $c \notin \mathsf{in}(\ell)$, then

$$w_{(b,a)} := w_{(b,a)} + 1$$
$$w_{(a,b)} := w_{(a,b)} - 1$$

That means, for each labeling we will consider all pairs of arguments $a, b \in \mathsf{Arg}$. Then, according to Rule 1 if both arguments are labeled in the weight of the attack (b, a) is decreased by 1. This represents the fact that there cannot be attacks between two in-labeled arguments. Rule 2 then decreases the weight of attacks in both directions between in- and undec-labeled arguments. The third rule only relevant if both arguments are labeled undec. If that is the case and if there exists an argument $c \in \mathsf{Arg}$ which attacks a or b with the label out or off, then we increase the weight of both attacks between a and b. This models the fact that arguments which are labeled undec have to be attacked by some undec-labeled arguments in order to be considered undec under grounded semantics. If that were not the case (and they are only attacked by out or off) they would have to be labeled in instead since they would be defended against all incoming attacks. Finally, the fourth rule applies when a is labeled out and b is labeled in. If there is an attacker $c \in \mathsf{Arg}$ of a, which is not labeled in, then the weight of the attack (b, a) is increased and the weight of the attack (a, b) is decreased. This models the fact that out-labeled arguments have to be attacked by at least one in-labeled argument.

Algorithm 1 Algorithm for learning from labelings from Riveret et al. [20]

```
1: **input** a set of arguments Arg, a set of labelings $L$.
2: $F = (\mathsf{Arg}, \mathsf{R}, w)$.
3: $\mathsf{R} \leftarrow \mathsf{Arg} \times \mathsf{Arg}$.
4: $\forall a, b \in \mathsf{Arg}$: initialize $w_{(a,b)} = 0$.
5: **while** there is an undecided attack *or* within computational budget **do**
6:     Get a labeling $\ell$.
7:     Update weights $w$ using the rules 1-4.
8:     $F \leftarrow prune(F)$. (optional)
9: **end while**
10: $F \leftarrow prune(F)$.
11: **return** F.
12:
```

Based on these four rules Riveret et al. have defined an algorithm (see Algorithm 1) which learns an attack relation (resp. an argumentation framework) from a given set of grounded 4-valued labelings.
As input the algorithm takes a set of arguments Arg and a set of grounded 4-valued labelings L. We initialize the internal weighted argumentation framework $F = (\mathsf{Arg}, \mathsf{R}, w)$ with *all* attacks and assign each attack an initial weight of 0. We then iterate over the labelings until either every attack is considered confirmed or discarded (see Definition 3.5) or we exceed a fixed computational budget. For each labeling $\ell \in L$ we iterate over all pairs of arguments and check the previously defined Rules 1-4. If a rule applied we update the weights of the relevant attacks. After updating all relevant attacks we can optionally prune the internal weighted argumentation

framework. That means we dump every attack that is labeled as discarded, i.e. it has a weight < 0. This means we will not have to consider it in future iterations. After reaching the end of the main loop we have to prune the internal framework and remove all discarded or undecided attacks. If we ignore the weights we obtain an argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$ which contains all confirmed attacks and should produce most of the input labelings.

While this algorithm works fine in most cases, there are some problems in certain scenarios. One problem is related to self-attacking arguments. If the argumentation framework F needs self-attacking arguments to produce all input labelings, the Riveret algorithm may not always find the correct solution. Another important point to note is that constructing grounded 4-valued labelings for the input requires a lot of knowledge about the original argumentation framework already, as we need to know sub-framework to construct these labelings. The algorithm also does not support other semantics, like admissible, complete, preferred or stable.

3.2. Synthesizing Argumentation Frameworks from Examples

In their paper Niskanen et al. [17] consider extension-based semantics and introduce the *AF synthesis problem*. Their work is related to the study of realizability [11], i.e. whether a semantics allows for exact representation of a set of extensions as an argumentation framework. The AF synthesis problem can be considered as an extension of realizability. In this scenario our goal is to "synthesize argumentation frameworks that are semantically closest to the knowledge at hand even when no argumentation frameworks exactly representing the knowledge exist" [17]. They also provide an MaxSAT-based algorithm that addresses this scenario.

We now look at the algorithm that addresses the AF synthesis problem. First, we formalize the AF synthesis problem introduced by Niskanen et al. as follows:

Input: set of arguments Arg, set of positive examples E^+, set of negative examples E^-, a semantics σ.

Output: some argumentation framework F, such that $cost(F, E^+, E^-)$ is minimal.

The positive examples are extensions, that shall be produced be the synthesized argumentation framework and the negative examples are extensions which shall not be produced by the argumentation framework. σ is the semantics which the positive examples should satisfy and the negative examples should not satisfy. Each example, positive and negative, also has a weight attached to it. These weights are used so compute the cost $cost(F, E^+, E^-)$ of a solution, i.e. the sum of the weights of all examples that are not satisfied by the synthesized argumentation framework. The goal of this algorithm is to find the argumentation framework which minimizes this cost function.

The algorithm proposed by Niskanen et al. works with MaxSAT encodings. That means, we have a set of hard clauses and a set of soft clauses. The hard clauses must be satisfied by any solution while the soft clauses each have a weight assigned and do not have to be satisfied. A solution of such a MaxSAT instance is an interpretation $\mathcal{A}$ which has a cost, i.e. the sum of weights of the unsatisfied soft clauses. Any solution is called optimal if it is minimal among all solutions.

Definition 3.7. *Let $P = (\mathsf{Arg}, E^+, E^-, \sigma)$ be an AF synthesis problem. We then define a propositional signature At as follows:*

$$\forall e \in E^+ \cup E^- : Ext^e_\sigma \in At$$

$$\forall a, b \in \mathsf{Arg} : r_{ab} \in At$$

The propositional signature At we will use for the AF synthesis problem is described in Definition 3.7. For any interpretation $\mathcal{A}$ with the synthesized argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$, we declare that $\mathcal{A}(Ext^e_\sigma) = true$ indicates $e \in \sigma(F)$ and $\mathcal{A}(r_{ab}) = true$ indicates $(a, b) \in \mathsf{R}$. Based on this we can define the hard clauses of the MaxSAT instance of the AF synthesis problem as follows:

Definition 3.8. *Let $P = (\mathsf{Arg}, E^+, E^-, \sigma)$ be an AF synthesis problem. Then we define the* hard clause *for each example $e \in E^+ \cup E^-$ as:*

$$Ext^e_\sigma \leftrightarrow \varphi_\sigma(e),$$

where $\varphi_\sigma(e)$ encodes the fact that e is a σ-extension.

The constraint $\varphi_\sigma(e)$ encodes all which attacks need to be present or not in an argumentation framework to ensure that e will be a σ extension of that framework.

Definition 3.9. *Let e be an example with respect to σ semantics. Then we define the constraint $\varphi_\sigma(e)$ as follows:*
If σ = conflict-free:

$$\varphi_{cf}(e) = \bigwedge_{a,b \in e} \neg r_{ab},$$

if σ = admissible:

$$\varphi_{ad}(e) = \varphi_{cf}(e) \wedge \bigwedge_{a \in e} \bigwedge_{b \in \mathsf{Arg} \setminus e} (r_{ba} \rightarrow \bigvee_{c \in e} r_{cb}),$$

if σ = complete:

$$\varphi_{co}(e) = \varphi_{ad}(e) \wedge \bigwedge_{a \in \mathsf{Arg} \setminus e} (\bigvee_{b \in \mathsf{Arg}} (r_{ba} \wedge \bigwedge_{c \in e} \neg r_{cb})),$$

and if σ = stable:

$$\varphi_{st}(e) = \varphi_{cf}(e) \wedge \bigwedge_{a \in \mathsf{Arg} \setminus e} (\bigvee_{b \in e} r_{ba}).$$

The soft clauses for the MaxSAT instance are quite simple: If $e \in E^+$ we have the soft clause Ext^e_σ together with the weight of e and if $e \in E^-$ we have the soft clause $\neg Ext^e_\sigma$ together with the weight of e. The synthesized argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$ is then obtained by computing the optimal solution $\mathcal{A}$ for the MaxSAT instance by including an attack (a, b) in the attack relation R if $\mathcal{A}(r_{ab}) = true$.

The algorithm proposed by Niskanen et al. is able to construct an argumentation framework for a given set of positive and negative input extensions with weights. This argumentation framework then produces as many positive examples as possible and it produces as few negative examples as possible with respect to their weights. The algorithm generates one constraint $\varphi_\sigma(e)$ per input example. These constraints get quite complex with increasing numbers of arguments and are difficult for humans to understand. The algorithm also only returns one argumentation framework that satisfies all generated constraints. That means, we are not able to refine the results further by considering additional input examples. If we would want to do that, we would have to create a new MaxSAT instance including all input examples and compute everything again. Instead, it would be desirable that we only need to learn the new input examples to get an updated argumentation framework.

3.3. Eliciting Argumentation Frameworks

In their recent work Kuhlmann [15] introduces the problem of *argumentation framework elicitation*. This problem can be considered as an extension of the idea of learning argumentation frameworks where we are able to interact with an agent and try to uncover his hidden argumentation framework.

Before describing the scenario, we define the notion of σ-equivalence which is relevant for the elicitation. Two argumentation frameworks are considered σ-equivalent if they have exactly the same σ-extensions, as noted in the following definition taken from [15].

Definition 3.10. *Let σ be a semantics. Two argumentation frameworks $F = (\mathsf{Arg}, \mathsf{R})$ and $F' = (\mathsf{Arg}, \mathsf{R})$ are σ-equivalent if they possess the same σ-extensions, i.e. $\sigma(F) = \sigma(F')$.*

The elicitation problem can now be described as the following interactive scenario: There is an agent that possesses a hidden argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$. We know the set of arguments Arg, but the set of attacks R is hidden to us. Our goal is now to elicit the attack relation by asking the agent a series of questions about its hidden argumentation framework. The agent then answers these questions truthfully. We consider our attempt to be successful if we are able to construct an argumentation framework F', such that F and F' are σ-equivalent.

There are different types of questions that we can ask the agent. We may differentiate two categories: syntactical and semantical questions. Possible syntactical questions are for instance: *"Is there an attack between the arguments a and b?"*, *"Is there a self-attacking argument?"* or *"How many attacks are there?"*.

On a semantical level we could for example ask the agent *"Is $E \subseteq \mathsf{Arg}$ a σ-extension of F?"*, *"What is the set of all σ-extensions of F?"* or *"Is the argument a contained in every σ-extension of F?"*.

Each question provides different types of information and the idea is now to come up with an algorithm to construct a σ-equivalent argumentation framework based on these various kinds of information.

There are also many other interesting open points of research related to the argumentation framework elicitation problem. It can for example be explored how useful the different questions are and if it is at all possible to uncover the hidden argumentation framework with them. For instance, the question *"Is there an attack between the arguments a and b?"* makes it quite easy to elicit the hidden argumentation framework, whereas other question provide less clear information about the argumentation framework. Furthermore, it could be investigated what the minimal number of questions is that we have to ask, in order to obtain a σ-equivalent argumentation framework. How the elicitation scenario could be adapted to the learning argumentation frameworks from labelings problem might also be an interesting topic to explore in the future (see Section 8).

3.4. A Bayesian Approach to the Inverse Problem

In their paper Kido and Liao define the *inverse problem* of abstract argumentation as follows [14]:

Direct Problem. Given an argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$, a direct problem aims to find sentiments regarding acceptability of the arguments, i.e. find extensions or labelings with respect to some semantics.
Inverse Problem. Given noisy sentiments regarding acceptability of arguments, an inverse problem aims to find an attack relation R explaining the sentiments well in terms of the semantics.

While the direct problem describes the standard idea behind argumentation semantics, the inverse problem describes a scenario of learning an attack relation from a given set of semantical information, e.g. in the form of extensions or labelings. A possible application of this scenario is using human judgment of arguments, e.g. extracted from the web, as input sentiment in order to find an attack relation that explains this sentiment.

Kido and Liao propose a probabilistic approach to address the inverse problem. Their approach includes a generative model which is used to trace the causality of the acceptability sentiment back to the argumentation framework [14]. This is done by considering the posterior probability over attack relations given acceptability values. Essentially, the approach computes an approximate distribution of attack relations that explain the input sentiments.

4. Learning Argumentation Frameworks

The research field of learning argumentation frameworks has not received much attention in the literature as of now and only few approaches [17, 20, 14] have been proposed to address the scenario.

In general, the idea behind learning argumentation frameworks is simple. In abstract argumentation we are interested in extracting semantic information from syntactic structures. This is typically done by computing the extensions (via semantics) of an argumentation framework. The idea is now to reverse this process and construct an attack relation (the syntactical structure) based on some given semantic information.

This section will give an overview of the distinctions that can be made in the scenarios addressed in the existing work. Following that, we will make some basic definitions that will be needed for this thesis.

4.1. Scenarios

The approaches for learning argumentation frameworks presented in the previous section show that we can consider different scenarios for learning. The approach by Niskanen et al. [17] as well as the approach by Riveret et al. [20] consider the set of arguments Arg to be known, while the attack relation R of the hidden argumentation framework is unknown. The goal is then to reconstruct this attack relation of this hidden argumentation framework with the help of some additional input. This input can differ in the amount and type of knowledge we are given for reconstructing the hidden argumentation framework. The following will give an overview of the different possibilities.

4.1.1. Type of Input

As mentioned earlier an acceptable position in an argumentation framework can be represented by either an extension or a labeling. If we know the argumentation framework then extensions and labelings can be considered equivalent as there is a one-to-one correspondence between complete extensions and complete labelings [7]. However, in the *Learning Argumentation Frameworks* scenario we do not have any knowledge about the hidden argumentation framework besides the set of arguments. For that reason extensions and labelings provide different amounts of information about the structure of the hidden argumentation framework. An extension only contains the acceptable arguments, while a labeling also distinguishes between two types of rejected arguments. Arguments that are directly attacked by acceptable arguments and arguments that are undecided. For that reason we can consider two different approaches: *Learning from Extensions* which is addressed by [17] and *Learning from Labelings* which has been addressed by the algorithm in [20]. The additional information from the labelings makes it easier to reconstruct the hidden argumentation framework, but that requires the knowledge to be used effectively.

4.1.2. Positive and Negative Examples

Besides the extensions as input we can also consider negative examples, like it was done in [17]. In the case of extensions a negative example is a set of arguments, that is not considered acceptable with respect to a semantics, i.e. it either contains a conflict or some other property of the semantics is violated by it. In the case of labelings the same applies, but in addition to that the labeling could also contain a violation of its definition, i.e. an argument is directly attacked by an in-labeled argument, but it is not labeled out.

4.1.3. Knowledge about Sub-Frameworks

The approach in [20] requires labelings of sub-frameworks to work effectively. That means we would have to be able to tell the entity to ignore certain arguments in its argumentation framework and give us an acceptable position in this restricted framework (sub-framework). Having this possibility is very powerful and would allow us to restrict the framework to all combinations of two arguments and ask for the extensions/labelings. This makes the task of reconstructing the argumentation framework much easier and thus should not be required for the approach developed in this master thesis.

4.1.4. Type of Algorithm

The approaches mentioned in Section 3 also differ in their procedure. The approach in [17] takes all the input and computes one argumentation framework. That means if there exists more than one fitting argumentation framework the algorithm will only return one of them. This can lead to problem as already mentioned in the introduction. If we execute the algorithm once it returns some argumentation framework which produces the given input. However, if we want to include some additional later obtained input we cannot used the previously generated argumentation framework and instead have to run the algorithm with the whole input again.

On the other hand, the input of the approach in [20] is a stream of labelings. With an approach like this it is possible to process additional inputs without starting from scratch. A downside of this approach is that we can only reliably use the information encoded in a labeling itself. Since we only process one labeling at a time we cannot rely on information that could be extracted by looking at the interrelations of labelings.

4.2. Basic Definitions

In the following we will introduce some basic concepts and notions that will be used in the remainder of this work. First, we denote with $args(\ell)$ the set of arguments that has been labeled by a labeling ℓ.

Definition 4.1. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and ℓ be a labeling of F. Then we define the set of arguments that appear in ℓ as follows:*

$$args(\ell) = \mathsf{in}(\ell) \cup \mathsf{out}(\ell) \cup \mathsf{undec}(\ell)$$

We also introduce the notion of input labelings (see Definition 4.2). An input labeling ℓ_σ is a labeling ℓ together with its corresponding semantics σ. In the scope of this work we consider all labelings to be input labelings. However, if the semantics of a labeling is not relevant or follows from context, we can omit it and simply write ℓ instead of ℓ_σ. The association of labelings and semantics will be relevant later when we compute conditions based on labelings and their associated semantics.

Definition 4.2. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. Then we define an* input labeling *ℓ_σ as a labeling ℓ of F which is valid in F with respect to the semantics σ.*

As mentioned already in Section 1 we use the notion of *produces* to describe whether a labeling can be obtained from an argumentation framework or not. We say that an argumentation framework F produces an input labeling ℓ_σ if ℓ_σ labels all arguments of F and the set of in-labeled arguments is a σ-extension of F. This is formalized in the following definition:

Definition 4.3. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and ℓ_σ be an input labeling. We say that F* produces *ℓ_σ if it holds that:*

1. *$args(\ell_\sigma) = \mathsf{Arg}$ and*
2. *$in(\ell_\sigma) \in \sigma(F)$.*

We can also extend this definition on sets of argumentation frameworks and sets of input labelings. That means, we say an argumentation framework F produces a set of labelings L, if F produces every input labeling $\ell_\sigma \in L$ with respect to their semantics. In that case the labelings in L may have different semantics. Furthermore, we say a set of argumentation frameworks $\mathbb{F}$ produces an input labeling ℓ_σ, if every argumentation framework $F \in \mathbb{F}$ produces ℓ_σ. Lastly, we say a set of argumentation frameworks $\mathbb{F}$ produces a set of labeling L, if every framework $F \in \mathbb{F}$ produces every labeling $\ell_\sigma \in L$.

We now define some concepts which are needed for the acceptance conditions that will be introduced in the following section. For that, we will use a propositional calculus. We define a set of atoms $attackAtoms(\mathsf{Arg})$ for a set of arguments as described in the following definition:

Definition 4.4. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework. Then we define the set of atoms representing all possible attacks in the argumentation framework F as follows:*

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

We denote with $\mathcal{L}(attackAtoms(\mathsf{Arg}))$ the set of all formulae over the set of atoms $attackAtoms(\mathsf{Arg})$.
Let us consider the argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$ and the set of atoms $attackAtoms(\mathsf{Arg})$ for the same set of arguments Arg. We can then say that there is a one-to-one correspondence between the attacks $(b, a) \in \mathsf{R}$ and the atoms $r_{ba} \in attackAtoms(\mathsf{Arg})$. That means, we could describe the argumentation framework F via the set of atoms $attackAtoms(\mathsf{Arg})$ as follows: If there is an attack from $b \in \mathsf{Arg}$ to $a \in \mathsf{Arg}$, then the corresponding atom r_{ba} must be true, otherwise we consider the atom r_{ba} as false. This concept will later be used to construct argumentation frameworks from a set of formulae over $attackAtoms(\mathsf{Arg})$.

Based on the above we define the set of atoms $inAttacks(a)$ for some argument a (see Definition 4.5). This set of atoms represents all possible incoming attacks an the argument a. With $\mathcal{L}(inAttacks(a))$ we denote the set of all formulae over the set of atoms $inAttacks(a)$. It holds for every argument $a \in \mathsf{Arg}$ that $inAttacks(a) \subseteq attackAtoms(\mathsf{Arg})$. Furthermore, it also holds for any set of arguments Arg that $attackAtoms(\mathsf{Arg}) = \bigcup\limits_{a \in \mathsf{Arg}} inAttacks(a)$.

Definition 4.5. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and $a \in \mathsf{Arg}$ is an argument. Then we define the set of atoms that represent all possible incoming attacks on the argument a as follows:*

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

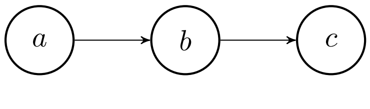


Figure 9: An argumentation framework.

Example 4.1. *Lets consider the argumentation framework in Figure 9 and the complete input labeling $\ell_{co} = \{\mathsf{in} = \{a, c\}, \mathsf{out} = \{b\}, \mathsf{undec} = \emptyset\}$. We can then say that F produces ℓ_{co}, because $args(\ell_{co}) = \mathsf{Arg}$ and indeed we have that $\{a, c\} \in co(F)$.*
Furthermore, we can also compute the set of attacks atoms of F as $attackAtoms(F) = \{r_{aa}, r_{ab}, r_{ac}, r_{ba}, r_{bb}, r_{bc}, r_{ca}, r_{cb}, r_{cc}\}$. Finally, the set of possible incoming attacks on the argument a would then be $inAttacks(a) = \{r_{aa}, r_{ba}, r_{ca}\}$.

5. Learning Argumentation Frameworks from Labelings

In this section we will introduce a novel algorithm for learning argumentation frameworks from labelings that addresses *RQ1*: How can we effectively compute the set of all argumentation frameworks that produce a given set of labelings?. In Section 3 we looked at some existing algorithms that address the problem of learning argumentation frameworks. However, these algorithms do have some problems and leave room for improvement. In the following we will formulate some requirements that shall be met by a new algorithm. Afterwards, we introduce an algorithm that is able to learn argumentation frameworks from labelings and fulfills the requirements.

5.1. Requirements

This section list the requirements that should be satisfied by an algorithm for learning argumentation frameworks. Requirement 1 describes the main goal of the algorithm. The Requirements 2, 3 and 4 are concerned with some additional restrictions regarding the input and the procedure of the algorithm.

Requirement 1. The algorithm shall take a set of labelings L as input and use them to compute the set of *all* argumentation frameworks $\mathbb{F}$ such that every $F \in \mathbb{F}$ produces L. This requirement is directly related to *RQ1*.

Requirement 2. The algorithm shall work with positive examples as input. That means negative examples are not considered as input. However, it can be considered to extend the approach later by including this possibility. For that we might also consider using negative examples in the form of extensions instead of labelings.

Requirement 3. The algorithm shall not require knowledge about sub-frameworks of the hidden argumentation framework. That means the algorithm does not require labelings of sub-frameworks in order produce a result like the algorithm in [20]. So, all input labelings are labelings of the whole argumentation framework and not just parts of it.

Requirement 4. The created approach shall be able to work with a stream of labelings as input, like the approach in [20]. That means after processing each input labeling we can see an intermediate result, representing the current set of argumentation frameworks which produce the input labelings processed so far.

5.2. General Approach

The idea for this algorithm is inspired by the *abstract dialectical frameworks*[1]. An abstract dialectical framework is a generalization of Dung's argumentation framework. Instead of just using attacks between arguments abstract dialectical frameworks use logical conditions to model the acceptability of arguments. We have one

[1]ADFs were introduced by Brewka et al. [4] [5]

acceptance condition per argument in the framework which tells us under which circumstances the argument can be accepted.

Having these acceptance conditions instead of just a simple attack relation allows for more complex acceptance restrictions. For example, lets say the acceptance condition for an argument a is $C_a = \neg b \vee \neg c$. That means, a can be accepted if either b or c are rejected. This concept is rather hard to model in a standard argumentation framework. Since this possibility is quite powerful it can be also be adapted and used to model the acceptability of an argument if we have incomplete information. This is the case when learning argumentation frameworks from labelings. If we are given a labeling, we might for example conclude that the argument a has to be attacked by either b or c based on the information from the labeling. Now, we normally would have to make a decision and choose one option while discarding the other possibility. This could lead to problems later if we look at another labeling and it turns out we chose the wrong option. Using the concept of acceptability conditions however, we can delay this decision until we obtain further information. In addition to that, if we end up with conditions that leave open more than one possibility (i.e. there is more than model for the acceptance condition formula) after processing all labelings, we can simply consider each model to represent a separate argumentation framework. That way we can in fact learn a set of argumentation frameworks from labelings.

For the algorithm we will only consider labelings with respect to a semantics $\sigma \in \mathbb{S}$ with $\mathbb{S} = \{\textit{conflict-free(cf)}, \textit{admissible(ad)}, \textit{complete(co)}, \textit{stable(st)}\}$. Other semantics, like grounded or preferred are not considered for now. The reason for that will be discussed in Section 6.

We describe the scenario addressed by the proposed algorithm as follows:
Given a set of input labelings L such that for every input labeling $\ell_\sigma \in L$ it holds that $\sigma \in \mathbb{S}$, we want to construct *all* argumentation frameworks F that produce L.

Input:
A set of arguments Arg and a set of input labelings $L = \{\ell_\sigma \mid \sigma \in \mathbb{S}\}$.

Output:
Compute the set of argumentation frameworks:

$$\mathbb{F} = \{F = (\mathsf{Arg}, \mathsf{R}) \mid F \text{ produces } L\}$$

The general procedure of the algorithm is as follows:
We are given a set of arguments Arg as well as a set of input labelings L. For each input labeling $\ell_\sigma \in L$ we compute a set of acceptance conditions $C_\ell = \{C_{a,\sigma}\}_{a \in \mathsf{Arg}}$.

Afterwards we combine these conditions, so that we have one condition C_a for each argument $a \in \mathsf{Arg}$.
Finally, the conditions are then used to construct a set of argumentation frameworks $\mathbb{F}$ which fulfill all conditions. That means every argumentation framework $F \in \mathbb{F}$ produces the given set of input labelings L.

5.3. Acceptance Conditions

For the algorithm we want to compute acceptance conditions for each argument based on the input labelings. We will use the following notation for the acceptance conditions of the algorithm. The condition for an argument a consists of literals that represent attacks in an argumentation framework. These literals have the form r_{ba} or $\neg r_{ba}$ with $a, b \in \mathsf{Arg}$. The positive occurrence r_{ba} in the condition C_a means there should be 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. $(b, a) \notin \mathsf{R}$.

Example 5.1. *Lets consider the set of arguments* $\mathsf{Arg} = \{a, b, c, d\}$*. Then, an acceptance condition for the argument* a *could for instance look like this*

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

This would mean that the argument a *should not be attacked by itself and the argument* b*. Furthermore,* a *must be attacked by either* c *or* d *or both. So, we have multiple different argumentation frameworks that satisfy the above acceptance condition for* a*.*

In this example, for the acceptance condition of a *we already used only literals representing incoming attacks on* a*. In the following, we will learn why this is a good idea and why this is required for the acceptance conditions used in the proposed algorithm.*

The goal is that the acceptance conditions encode the relevant information that can be extracted from the input labelings. As the name suggests the acceptance condition for an argument a is concerned with the acceptance of that argument. Recalling Section 2, an argument can be accepted under any semantics if it is not attacked by any other accepted argument. 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. What that means is, if we want to formulate a condition for the acceptance of an argument we only need to consider its direct attackers and their label in a given labeling. It does not matter in this context what the outgoing attacks of the argument a are. For that reason, we define the *acceptance condition* C_a for an argument a with respect to a input labeling ℓ_σ as a formula over the set of atoms $inAttacks(a)$. This formula is computed via the family of functions $AccCond_\sigma(a, \ell)$. These functions take an argument a and an input labeling ℓ_σ and return the acceptance condition. The specifics of this function depend on the semantics σ of the input labeling and are defined in Sections 5.3.1-5.3.4.

Definition 5.1. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and ℓ_σ be an input labeling. Then we define a family of functions $AccCond_\sigma(a, \ell)$ that computes the* acceptance condition *of an argument $a \in \mathsf{Arg}$ with respect to the input labeling ℓ_σ, which has the following signature:*

$$AccCond_\sigma(a, \ell) : \mathsf{Arg} \times \mathbb{L}(\mathsf{Arg}) \longrightarrow \mathcal{L}(inAttacks(a))$$

The algorithm for learning argumentation frameworks from labelings will use the above introduced concept of acceptance conditions. For each input labeling ℓ_σ we compute a set of acceptance conditions $C_\ell = \{C_{a,\sigma} = AccCond_\sigma(a, \ell)\}_{a \in \mathsf{Arg}}$. We have one acceptance condition $C_{a,\sigma}$ for each argument a present in the labeling ℓ_σ. From a set of acceptance conditions we can then obtain consistent argumentation frameworks thanks to the correspondence between the atoms and attacks. The exact process of constructing these argumentation frameworks will be described in detail in Section 5.4.2. However, we note already in the following definition how these argumentation frameworks are distinguished:

Definition 5.2. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework, ℓ be a labeling and C_ℓ is the set of acceptance conditions for ℓ. We say that F is consistent with C_ℓ if and only if there is a model $\mathcal{A}$ of C_ℓ, such that $\mathcal{A}(r_{ba})$ is true for all attacks $(b, a) \in \mathsf{R}$, otherwise $\mathcal{A}(r_{ba})$ is false.*

An important characteristic of the acceptance condition C_a for an argument a is that they per definition only contain literals concerning the *incoming* attacks on a. As a reminder, C_a is defined as a formula over the set of atoms $inAttacks(a) = \{r_{ba} \mid b \in \mathsf{Arg}\}$. With this approach we ensure that an atom r_{ba} for any argument $b \in args(\ell)$ only occurs in the acceptance condition of the argument a and nowhere else. This leads likely to a better performance since we can compute a model for the acceptance condition of each argument independently.

This acceptance condition C_a contains the atoms concerning all incoming attacks that are *necessary* to ensure the constructed argumentation framework will produce the input labeling ℓ_σ. On the other hand, atoms concerning incoming attacks that do not influence the arguments label will not be contained in the acceptance condition. So, an acceptance condition C_a may not contain all atoms in $inAttacks(a)$. If an attack $(b, a) \in \mathsf{R}$ is not explicitly covered by the acceptance condition, i.e. r_{ba} is not contained in C_a, then we consider this attack to be possible, but not necessary in the argumentation framework producing the labeling ℓ. These attacks will still have to be considered when we compute the set of argumentation frameworks in the final step.

As explained above, the acceptability of an argument only depends on the label of its attackers in a given labeling. However, the exact reasoning behind the acceptability differs between most semantics. For example, in a conflict-free labeling an argument can be accepted even if it is attacked by an undec-labeled argument. This is however not the case for all other semantics that are considered in this master thesis. For that reason the acceptance conditions for an argument will be different, depending on the semantics of the input labeling. In the following we will define the acceptance conditions for conflict-free, admissible, complete and stable semantics.

5.3.1. Conflict-free Semantics

For input labelings ℓ_{cf} (in the following only denoted ℓ) with respect to conflict-free semantics the acceptance conditions are quite simple. If an argument $a \in args(\ell)$ 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. 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. If the argument is labeled out, then it has to be attacked by at least on argument $b \in \mathsf{in}(\ell)$.

Optionally 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 are not contained in the acceptance condition of a.

The acceptance conditions are sound and complete for conflict-free input labelings.

Definition 5.3. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and ℓ be an input labeling with respect to conflict-free semantics. We then define for each argument $a \in args(\ell)$ the acceptance condition C_a with respect to the conflict-free input labeling ℓ via the function $AccCond_{cf}(a, \ell)$ as follows:*

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

Theorem 5.1. *Let ℓ be a conflict-free input labeling and $C = \{C_a = AccCond_{cf}(a, \ell) \mid a \in \mathsf{Arg}\}$ is the set of acceptance conditions for the input labeling ℓ. With $\mathbb{F}$ we denote the set of argumentation frameworks consistent with C.*

The acceptance conditions are sound *for conflict-free input labelings, i.e.*

$$\forall F \in \mathbb{F} : \ell \textit{ is conflict-free} \implies F \textit{ produces } \ell$$

The acceptance conditions are complete *for conflict-free input labelings, i.e.*

$$\forall G = (\mathsf{Arg}, \mathsf{R}) : G \textit{ produces } \ell \implies G \in \mathbb{F}$$

Proof. see Appendix A.1

□

Example 5.2. *Lets consider the set of arguments* $\mathsf{Arg} = \{a, b, c, d\}$ *and the conflict free input labeling* $\ell_{cf} = \{\mathsf{in} = \{a, b\}, \mathsf{off} = \{c\}, \mathsf{undec} = \{d\}\}$ *over these arguments. We can then use the above defined function* $AccCond_{cf}(a, \ell)$ *to compute the acceptance condition of each argument in* Arg*:*

$$\begin{aligned}
C_a = AccCond_{cf}(a, \ell_{cf}) = &\quad \neg r_{aa} \wedge \neg r_{ba} \\
C_b = AccCond_{cf}(b, \ell_{cf}) = &\quad \neg r_{ab} \wedge \neg r_{bb} \\
C_c = AccCond_{cf}(c, \ell_{cf}) = &\quad r_{ac} \vee r_{bc} \\
C_d = AccCond_{cf}(d, \ell_{cf}) = &\quad \neg r_{ad} \wedge \neg r_{bd}
\end{aligned}$$

These conditions represent the input labeling ℓ_{cf}. 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 acceptance condition.*

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

5.3.2. Admissible Semantics

For a labeling ℓ_{ad} (in the following only denoted ℓ) with respect to admissible semantics the conditions are fairly similar. This is related to the fact that every admissible labeling is also conflict-free. The acceptance condition 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. If the argument is labeled in the condition 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. This kind of attack is not necessary and thus not included in the acceptance condition for the argument a. 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.

The acceptance conditions are sound and complete for admissible input labelings.

Definition 5.4. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and ℓ be an input labeling with respect to admissible semantics. We then define for each argument $a \in args(\ell)$ the acceptance condition C_a with respect to the admissible input labeling ℓ via the function $AccCond_{ad}(a, \ell)$ as follows:*

$$AccCond_{ad}(a, \ell) = \begin{cases} \bigwedge\limits_{b \in \mathsf{Arg} \setminus \mathsf{out}(\ell)} \neg r_{ba} & a \in \mathsf{in}(\ell) \\ \bigvee\limits_{b \in \mathsf{in}(\ell)} r_{ba} & a \in \mathsf{out}(\ell) \\ \bigwedge\limits_{b \in \mathsf{in}(\ell)} \neg r_{ba} & a \in \mathsf{undec}(\ell) \end{cases}$$

Theorem 5.2. *Let ℓ be an admissible input labeling and $C = \{C_a = AccCond_{ad}(a, \ell) \mid a \in \mathsf{Arg}\}$ is the set of acceptance conditions for the input labeling ℓ. With $\mathbb{F}$ we denote the set of argumentation frameworks consistent with C.*

The acceptance conditions are sound *for admissible input labelings, i.e.*

$$\forall F \in \mathbb{F} : \ell \text{ is admissible} \implies F \text{ produces } \ell$$

The acceptance conditions are complete *for admissible input labelings, i.e.*

$$\forall G = (\mathsf{Arg}, \mathsf{R}) : G \text{ produces } \ell \implies G \in \mathbb{F}$$

Proof. see Appendix A.2 □

Example 5.3. *Lets consider the set of arguments* $\mathsf{Arg} = \{a, b, c, d\}$ *and the admissible input labeling* $\ell_{ad} = \{\mathsf{in} = \{a\}, \mathsf{off} = \{c\}, \mathsf{undec} = \{b, d\}\}$ *over these arguments. We can then use the above defined function* $AccCond_{ad}(a, \ell)$ *to compute the acceptance condition of each argument in* Arg*:*

$$\begin{aligned} C_a = AccCond_{ad}(a, \ell_{ad}) = &\quad \neg r_{aa} \land \neg r_{ba} \land \neg r_{da} \\ C_b = AccCond_{ad}(b, \ell_{ad}) = &\quad \neg r_{ab} \\ C_c = AccCond_{ad}(c, \ell_{ad}) = &\quad r_{ac} \\ C_d = AccCond_{ad}(d, \ell_{ad}) = &\quad \neg r_{ad} \end{aligned}$$

These conditions represent the input labeling ℓ_{ad}*. 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 acceptance conditions are again considered to be optional.*

5.3.3. Complete Semantics

The conditions for complete input labelings ℓ_{co} (in the following only denoted ℓ) are building on the conditions for admissible labelings. The acceptance condition for in-labeled arguments is the same as before, capturing both conflict-freeness and defense. The condition for arguments with the label out are also identical and just require some attack from any in-labeled argument. However, the acceptance condition 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.

Optionally an argument with the label undec can still be attacked by arguments that are labeled out since these attacks do not influence the label of a.

The acceptance conditions are sound and complete for complete input labelings.

Definition 5.5. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and ℓ be an input labeling with respect to complete semantics. We then define for each argument $a \in args(\ell)$ the acceptance condition C_a with respect to the complete input labeling ℓ via the function $AccCond_{co}(a, \ell)$ as follows:*

$$AccCond_{co}(a,\ell) = \begin{cases} \bigwedge\limits_{b \in \mathsf{Arg} \setminus \mathsf{out}(\ell)} \neg r_{ba} & a \in \mathsf{in}(\ell) \\ \bigvee\limits_{b \in \mathsf{in}(\ell)} r_{ba} & a \in \mathsf{out}(\ell) \\ \bigwedge\limits_{b \in \mathsf{in}(\ell)} \neg r_{ba} \wedge (\bigvee\limits_{c \in \mathsf{undec}(\ell)} r_{ca}) & a \in \mathsf{undec}(\ell) \end{cases}$$

Theorem 5.3. *Let ℓ be a complete input labeling and $C = \{C_a = AccCond_{co}(a, \ell) \mid a \in \mathsf{Arg}\}$ is the set of acceptance conditions for the input labeling ℓ. With $\mathbb{F}$ we denote the set of argumentation frameworks consistent with C.*

The acceptance conditions are sound *for complete input labelings, i.e.*

$$\forall F \in \mathbb{F} : \ell \textit{ is complete } \implies F \textit{ produces } \ell$$

The acceptance conditions are complete *for complete input labelings, i.e.*

$$\forall G = (\mathsf{Arg}, \mathsf{R}) : G \textit{ produces } \ell \implies G \in \mathbb{F}$$

Proof. see Appendix A.3 □

Example 5.4. *Lets consider the set of arguments $\mathsf{Arg} = \{a, b, c, d\}$ and the complete input labeling $\ell_{co} = \{\mathsf{in} = \{c\}, \mathsf{off} = \{a\}, \mathsf{undec} = \{b, d\}\}$ over these arguments. We can then use the above defined function $AccCond_{co}(a, \ell)$ to compute the acceptance condition of each argument in Arg:*

$$\begin{aligned} C_a = AccCond_{co}(a, \ell_{co}) = &\quad r_{ca} \\ C_b = AccCond_{co}(b, \ell_{co}) = &\quad \neg r_{cb} \wedge (r_{bb} \vee r_{db}) \\ C_c = AccCond_{co}(c, \ell_{co}) = &\quad \neg r_{bc} \wedge \neg r_{cc} \wedge \neg r_{dc} \\ C_d = AccCond_{co}(d, \ell_{co}) = &\quad \neg r_{cd} \wedge (r_{bd} \vee r_{dd}) \end{aligned}$$

Like before, the only off*-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 optional attacks in this example, for instance the argument a can optionally be attacked by itself, b or d while still satisfying the acceptance conditions.

5.3.4. Stable Semantics

If we have a stable input labeling ℓ_{st} (in the following only denoted ℓ) 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 other conditions 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 previous conditions 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 $b \in \mathsf{in}(\ell)$.

In addition to that, every argument can be attacked by any out-labeled argument without influencing its label.

The acceptance conditions are sound and complete for stable input labelings.

Definition 5.6. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework and ℓ be an input labeling with respect to stable semantics. We then define for each argument $a \in args(\ell)$ the acceptance condition C_a with respect to the stable input labeling ℓ via the function $AccCond_{st}(a, \ell)$ as follows:*

$$AccCond_{st}(a,\ell) = \begin{cases} \bigwedge\limits_{b \in \mathsf{in}(\ell)} \neg r_{ba} & a \in \mathsf{in}(\ell) \\ \bigvee\limits_{b \in \mathsf{in}(\ell)} r_{ba} & a \in \mathsf{out}(\ell) \end{cases}$$

Theorem 5.4. *Let ℓ be a stable input labeling and $C = \{C_a = AccCond_{st}(a, \ell) \mid a \in \mathsf{Arg}\}$ is the set of acceptance conditions for the input labeling ℓ. With $\mathbb{F}$ we denote the set of argumentation frameworks consistent with C.*

The acceptance conditions are sound *for stable input labelings, i.e.*

$$\forall F \in \mathbb{F} : \ell \textit{ is stable} \implies F \textit{ produces } \ell$$

The acceptance conditions are complete *for stable input labelings, i.e.*

$$\forall G = (\mathsf{Arg}, \mathsf{R}) : G \textit{ produces } \ell \implies G \in \mathbb{F}$$

Proof. see Appendix A.4 □

5.3.5. Properties

After defining the acceptance conditions for input labelings with respect to different semantics, we make some observations. Usually, the acceptance condition C_a for an argument a will not contain all atoms $r_{ba} \in inAttacks(a)$. As mentioned before there can be attacks on the argument a that do not influence the label of a. These attacks are optional but we need to consider them when constructing the set of argumentation frameworks that produce the set of input labelings. Which of the attacks are considered optional depends on the semantics of the input labeling and the labels of the arguments in that labeling. However, we can easily infer which attacks are optional from the acceptance condition C_a.

That means we can define an *optional acceptance condition* C_a^{opt} for each argument a that captures the optional attacks as described in the following definition.

Definition 5.7. *Let $F = (\mathsf{Arg}, \mathsf{R})$ be an argumentation framework, ℓ is a labeling of F and $a \in args(\ell)$. C_a is the acceptance condition of the argument a.*
The acceptance condition contains the atom r_{ba} for each argument b in a set $B \subseteq \mathsf{Arg}$. Then, the optional acceptance condition $C_a^{opt} \in \mathcal{L}(inAttacks(a))$ *can be defined as follows:*

$$C_a^{opt} = \bigvee_{c \in \mathsf{Arg} \setminus B} r_{ca}$$

Example 5.5. *Consider the set of arguments $\mathsf{Arg} = \{a, b, c, d\}$. Lets look at the acceptance condition for the argument a from Example 5.2:*

$$C_a = \neg r_{aa} \wedge \neg r_{ba}.$$

Then, the acceptance condition contains the atoms $\{r_{aa}, r_{ba}\}$. That means, the atoms $\{r_{ca}, r_{da}\}$ have to be considered for the optional acceptance condition. According to the above definition we then obtain the optional acceptance condition of a:

$$C_a^{opt} = r_{ca} \vee r_{da}.$$

The optional condition then tells us that the argument a can optionally be attacked by c or d or both without influencing the acceptability of the arguments, i.e. even with these attacks the resulting argumentation framework would still produce the labeling for which the acceptance conditions where computed.

For the computations during the algorithm we do not need the optional acceptance condition. However, we can use it to get an overview of the attacks on the argument a which are still open, i.e. all attacks that are still possible but not necessary.

Another thing that should be noted is that the acceptance condition for an argument with the label out is always the same, regardless of the semantics of the labeling. This makes sense, since the label out does not depend on the semantics of the labeling at all. For an argument with the label out it is only relevant to know that it is attacked by some accepted argument and thus the acceptance condition of these arguments is independent of the semantics.

Furthermore, we can observe that the above definitions of acceptance conditions work for argumentation frameworks with self-attacks, without further adjustments. A self-attacking argument will be labeled undec for all semantics, unless it is attacked by an in-labeled argument, in which case it will be labeled out.

5.4. Algorithm

In the following we will introduce an algorithm for learning argumentation frameworks from labelings. The procedure for the algorithm is shown in Algorithm 2. At first, we receive a set of arguments Arg and a set of input labelings L as the input. For every input labeling $\ell_\sigma \in L$ it holds that $args(\ell_\sigma) \subseteq \mathsf{Arg}$. Furthermore, each labeling is with respect to a different semantics $\sigma \in \{\textit{conflict-free}, \textit{admissible}, \textit{complete}, \textit{stable}\}$. Following that, we initialize the set of argumentation frameworks $\mathbb{F}$ as the empty set. We also initialize an acceptance condition for each argument a as $C_a = \top$. That means, by default no attacks are necessary. The main procedure of the algorithm can essentially be split into three parts:

1. Computing the acceptance conditions $C_{a,\ell}$ for each argument and labeling.
2. Combining the conditions into one acceptance condition C_a per argument.
3. Constructing the set of argumentation frameworks $\mathbb{F}$ from the acceptance conditions C_L.

Algorithm 2 Algorithm for learning argumentation frameworks from labelings.

```
1: input set of arguments Arg, set of input labelings $L$.
2: $\mathbb{F} \leftarrow \emptyset$
3: $C_L \leftarrow \{C_a = \top\}_{a \in \mathsf{Arg}}$
4: for each input labeling $\ell_\sigma \in L$ do
5:     for each argument $a \in \mathsf{Arg}$ do
6:         $C_{a,\ell} \leftarrow AccCond_\sigma(a, \ell_\sigma)$
7:     end for
8: end for
9: for each argument $a \in \mathsf{Arg}$ do
10:    $C_a \leftarrow combineConditions(\{C_{a,\ell}\}_{\ell \in L})$
11: end for
12: $\mathbb{F} \leftarrow constructFrameworksFromConditions(C_L)$
13: return $\mathbb{F}$.
14:
```

First, we iterate over all input labelings in L. For each input labeling ℓ_σ we compute an acceptance condition $C_{a,\ell}$ of each argument $a \in \mathsf{Arg}$ with respect to that labeling ℓ_σ. In order to do this we have to consider the labeling as well as the semantics σ it is associated with. We use the family of functions $AccCond_\sigma(a, \ell)$, which has been defined in Section 5.3, to compute the acceptance conditions and return the correct condition for the given argument a in the input labeling ℓ_σ. The semantics σ can be the conflict-free, admissible, complete or stable semantics.

In the second step we iterate over all arguments in Arg. For each argument a we take the acceptance conditions $C_{a,\ell}$ from all input labelings and combine them into a single acceptance condition C_a per argument. This is done in the abstract method $combineConditions(C)$ and is further described in Section 5.4.1.

In the third and final step we look at the previously computed set of acceptance conditions C_L and construct the set of argumentation frameworks $\mathbb{F}$. For this we will consider the models of each acceptance condition C_a and use the fact that each atom represents an attack in the argumentation framework. This part of the algorithm is described in Section 5.4.2. After the construction the algorithm returns the set of argumentation frameworks $\mathbb{F}$.

5.4.1. Combining Acceptance Conditions

The process of combining a set of acceptance conditions $C = \{C_{a,\ell}\}_{\ell \in L}$ for an argument a and reducing them to a single condition C_a is fairly simple. The combined condition for the argument is just the logical conjunction of all conditions $c \in C$. In addition to that, we can apply standard logical simplifications to reduce the complexity of the acceptance condition and remove unnecessary elements.

Definition 5.8. *Let C be a set of acceptance conditions. Then, the we define the method $combineConditions(C)$ to return the following logical formula:*

$$C_a = \bigwedge_{c \in C} c.$$

Example 5.6. *Let* Arg $= \{a, b, c, d\}$ *be a set of arguments. We consider the acceptance conditions for the input labelings ℓ_{cf} and ℓ_{co} from Examples 5.2 and 5.4:*

$$\begin{array}{rclrcl}
C_{a,\ell_{cf}} & = & \neg r_{aa} \wedge \neg r_{ba} & C_{a,\ell_{co}} & = & r_{ca} \\
C_{b,\ell_{cf}} & = & \neg r_{ab} \wedge \neg r_{bb} & C_{b,\ell_{co}} & = & \neg r_{cb} \wedge (r_{bb} \vee r_{db}) \\
C_{c,\ell_{cf}} & = & r_{ac} \vee r_{bc} & C_{c,\ell_{co}} & = & \neg r_{bc} \wedge \neg r_{cc} \wedge \neg r_{dc} \\
C_{d,\ell_{cf}} & = & \neg r_{ad} \wedge \neg r_{bd} & C_{d,\ell_{co}} & = & \neg r_{cd} \wedge (r_{bd} \vee r_{dd})
\end{array}$$

So, in order to obtain the acceptance condition representing both input labelings for each argument we combine them via conjunction. For the argument a this is straight forward

$$C_a = \neg r_{aa} \wedge \neg r_{ba} \wedge r_{ca}.$$

For the argument b we first obtain the formula $C_b = \neg r_{ab} \wedge \neg r_{bb} \wedge \neg r_{cb} \wedge (r_{bb} \vee r_{db})$. We can however simplify this by removing the literal r_{bb} from the disjunction, since we know from the condition $C_{b,\ell_{cf}}$ that $\mathcal{A}(r_{bb})$ must be false. That gives us the simplified acceptance condition for b

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

For the argument c we can again simplify by removing the literal $r_b c$ from the disjunction, because it is already stated in $C_{c,\ell_{co}}$ that $\mathcal{A}(r_{bc})$ must be false:

$$C_c = r_{ac} \land \neg r_{bc} \land \neg r_{cc} \land \neg r_{dc}.$$

Finally, the acceptance condition for d can also be simplified by removing the literal r_{bd} and dissolving the disjunction like before:

$$C_d = \neg r_{ad} \land \neg r_{bd} \land \neg r_{cd} \land r_{dd}.$$

5.4.2. Constructing Argumentation Frameworks from Conditions

In the final step of the algorithm we need to construct a set of argumentation frameworks from a given set of acceptance conditions C_L. These argumentation frameworks $F = (\mathsf{Arg}, \mathsf{R})$ have to satisfy the acceptance condition $C_a \in C_L$ of each argument $a \in \mathsf{Arg}$. That way we ensure that every constructed argumentation framework F produces the set of input labelings L.

Assume $a, b \in \mathsf{Arg}$ are arguments. We know that the atom r_{ba} can only occur in the acceptance condition of the argument a. Because of this property, we can look at the acceptance condition of each argument individually, compute partial attack relations $\mathsf{R} \subset \mathsf{Arg} \times \mathsf{Arg}$ and combine them afterwards into different argumentation frameworks.

Algorithm 3 Algorithm for constructing a set of argumentation frameworks from a set of acceptance conditions.

```
1: input set of arguments Arg, set of acceptance conditions $C = \{C_a\}_{a \in \mathsf{Arg}}$.
2: $\mathbb{F} \leftarrow \emptyset$
3: for each argument $a \in \mathsf{Arg}$ do
4:     $\mathcal{M}_a \leftarrow \{\mathcal{A} \in Int_{inAttacks(a)} \mid \mathcal{A} \models C_a\}$
5:     $R_a \leftarrow \{\mathsf{R}_{a,i} = toR(\mathcal{A}_i) \mid \mathcal{A}_i \in \mathcal{M}_a\}$
6: end for
7: $R_L \leftarrow R_a \times \cdots \times R_z$ with $a, \ldots, z \in \mathsf{Arg}$
8: $\mathbb{F} \leftarrow \{F = (\mathsf{Arg}, \mathsf{R}) \mid \mathsf{R} = \bigcup_{\mathsf{R}_i \in R'} \mathsf{R}_i$ with $R' \in R_L\}$
9: return $\mathbb{F}$.
```

The procedure for constructing the argumentation frameworks, done by the method $constructFrameworks(C)$, is shown in Algorithm 3. We start with the set of acceptance conditions C. The set of arguments Arg also follows implicitly from this, as there is exactly one acceptance condition per argument. For every argument $a \in \mathsf{Arg}$ we consider the acceptance condition C_a. We then compute all interpretations $\mathcal{A}$ for which C_a is true. So, for each argument a we obtain the set of all models $\mathcal{M}_a$, i.e. all interpretations that satisfy the acceptance condition of a. We might then also say $\mathcal{A}$ is a model of a, if $\mathcal{A}$ is a model of the acceptance condition of a.

Definition 5.9. *Let $a \in \mathsf{Arg}$ be an argument and C_a is the acceptance condition of a. Then we define the set of all models $\mathcal{M}_a$ of a as the set of all interpretations $\mathcal{A}$ that satisfy C_a:*

$$\mathcal{M}_a = \{\mathcal{A} \in Int_{inAttacks(a)} \mid \mathcal{A} \models C_a\}$$

It is important to note, that we need to consider the optional attacks that are not included in the acceptance conditions. This is relevant because the goal is to compute all argumentation frameworks that produce the input labelings. That includes argumentation frameworks which contain attacks that are not necessary to produce the set of input labelings. If we would not include all of these possibilities we lose information and might not be able to reconstruct the correct argumentation framework in a later step. However, all of these possibilities are covered by Definition 5.9. The definition explicitly considers all interpretations of the set of atoms $inAttacks(a)$. That means all attack atoms on the argument a are considered and not only the ones included in the acceptance condition. That means, the set of all models $\mathcal{M}_a$ implicitly includes all configurations of attacks and no extra step is needed to address the optional attacks.

After computing the models of an argument a we proceed to the next step in line 5 of the algorithm which is computing a corresponding partial attack relation for each model. Per definition an atom r_{ba} represents the attack from the argument b on the argument a. That means, we can easily construct the corresponding partial attack relation for every model. This process is done by the function $toR(\mathcal{A})$ and defined in the following.

Definition 5.10. *Let $a, b \in \mathsf{Arg}$ be arguments and $\mathcal{A} \in \mathcal{M}_a$ is a model of a. Then we define the function $toR : Int_{inAttacks(a)} \rightarrow \mathsf{Arg} \times \mathsf{Arg}$ as follows:*

$$toR(\mathcal{A}) = \{(b, a) \mid \mathcal{A}(r_{ba}) = true\}$$

We then store all partial attack relation for the argument a in the set R_a. In the next step we will combine the partial attack relations into argumentation frameworks (see line 7-8). That means, we take one partial attack relation $\mathsf{R}_{a,i} \in R_a$ for each argument a, and take the union in order to obtain a full attack relation. Together with the set of arguments Arg this can now be assembled to the argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$ and added to the set $\mathbb{F}$. This is done for every possible combination and finally we return the set of constructed argumentation frameworks $\mathbb{F}$. With this procedure we obtain different attack relations that all satisfy every acceptance condition and thus also produce the set of input labelings L.

Example 5.7. *We again consider the set of arguments* $\mathsf{Arg} = \{a, b, c, d\}$ *and the acceptance conditions computed in Example 5.6 by combining the conditions for the two input labelings* $\ell_{cf} = \{\mathsf{in} = \{a, b\}, \mathsf{off} = \{c\}, \mathsf{undec} = \{d\}\}$ *and* $\ell_{co} = \{\mathsf{in} = \{c\}, \mathsf{off} = \{a\}, \mathsf{undec} = \{b, d\}\}$*:*

$$\begin{aligned} C_a = &\quad \neg r_{aa} \wedge \neg r_{ba} \wedge r_{ca} \\ C_b = &\quad \neg r_{ab} \wedge \neg r_{bb} \wedge \neg r_{cb} \wedge r_{db} \\ C_c = &\quad r_{ac} \wedge \neg r_{bc} \wedge \neg r_{cc} \wedge \neg r_{dc} \\ C_d = &\quad \neg r_{ad} \wedge \neg r_{bd} \wedge \neg r_{cd} \wedge r_{dd} \end{aligned}$$

In order to construct the corresponding argumentation frameworks, we first compute the models for each acceptance condition. For the argument a *we have the two models*

$$\mathcal{M}_a = \{[r_{ca} \to true], [r_{ca}, r_{da} \to true]\}.$$

As we can see, since we consider interpretations $\mathcal{A} \in Int_{inAttacks(a)}$*, one of the models includes the optional attack from* d *on* a*. For the arguments* b*,* c *and* d *we then only have one model each:*

$$\begin{aligned} \mathcal{M}_b &= \{[r_{db} \to true]\} \\ \mathcal{M}_c &= \{[r_{ac} \to true]\} \\ \mathcal{M}_d &= \{[r_{dd} \to true]\} \end{aligned}$$

Based on these models, we may then compute the corresponding partial attack relations as described in Definition 5.10. After doing that, we obtain the two partial attack relations $\{(c, a)\}$ *and* $\{(c, a), (d, a)\}$ *for the argument* a*. The other arguments then have one partial attack relation each:* $\{(d, b)\}$ *for* b*,* $\{(a, c)\}$ *for* c *and* $\{(d, d)\}$ *for* d*.*

Finally, we assemble the complete attack relations: $\mathsf{R}_1 = \{(c, a), (d, b), (a, c), (d, d)\}$ *and* $\mathsf{R}_2 = \{(c, a), (d, b), (a, c), (d, d), (d, a)\}$*. Together with the set of arguments* Arg*, we have two resulting argumentation frameworks* $F_1 = (\mathsf{Arg}, \mathsf{R}_1)$ *and* $F_2 = (\mathsf{Arg}, \mathsf{R}_2)$ *in this example because of the two possibilities for the condition* C_a *of* a*. The two argumentation frameworks are depicted in Figure 10.*

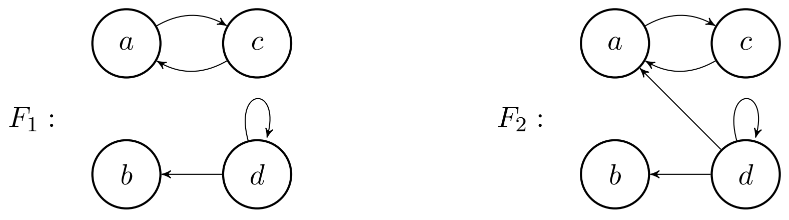


Figure 10: The only argumentation frameworks F_1 and F_2 that produce the input labelings ℓ_{cf} and ℓ_{co}.

5.4.3. Iterative Approach

The algorithm presented in the previous section already satisfies some of the requirements. The algorithm addresses the scenario described in Requirement 1. It also satisfies Requirement 2 as the algorithm can process positive examples as input, i.e. in the form of labelings. Furthermore, Requirement 3 is satisfied since the proposed algorithm does not rely on sub-frameworks labelings.

On the other hand, the algorithm does not satisfy Requirement 4 which states that the algorithm should be able to consider a stream of input labelings and provide intermediate results after each processed labeling. However, this can easily be achieved by restructuring the above introduced approach as shown in Algorithm 4. This iterative approach and the approach in Algorithm 2 are equivalent, i.e. they construct the same set of argumentation frameworks for the same input.

Algorithm 4 Iterative version of Algorithm 2.

```
1: input set of conditions $C = \{C_a \mid a \in \mathsf{Arg}\}$, set of arguments Arg, input labeling $\ell_\sigma$.
2: for each argument $a \in \mathsf{Arg}$ do
3:     $C_{a,\ell} \leftarrow AccCond_\sigma(a, \ell_\sigma)$
4:     $C_a \leftarrow combineConditions(\{C_a, C_{a,\ell}\})$
5: end for
6: $\mathbb{F} \leftarrow constructFrameworks(C)$    (optional)
7: return $C$, $(\mathbb{F})$.
```

In the iterative approach we take a set of acceptance conditions C, the set of arguments Arg and some input labeling ℓ_σ as input. Now, similar to before, we iterate over all arguments $a \in \mathsf{Arg}$. For each argument we compute the acceptance condition $C_{a,\ell}$ of a with respect to the input labeling ℓ_σ. Then, we combine it with the existing acceptance condition C_a as was specified in Section 5.4.1. If we are interested in an intermediate result, we can then construct the set of argumentation frameworks from the conditions (see line 6). Otherwise we skip this step as it not needed for the algorithm, since we only need the set of acceptance conditions C with the updated conditions for each argument.

Finally, the algorithm returns the updated set of acceptance conditions C and (if wanted) the set of constructed argumentation frameworks $\mathbb{F}$. The returned set of acceptance conditions C represents all information that has been learned and can then be used to learn further labelings.

An important note is, that we do not need to actually construct the set of argumentation frameworks in order to get an idea of its size during the learning process. It is possible to determine the number of argumentation frameworks in the set based on the acceptance conditions alone. This will be discussed in more detail in Section 6.1.

5.4.4. Properties

In the following we will look at some useful properties that the previously defined algorithm satisfies, like soundness, completeness and monotonicity. We will also define the notion of minimal constructed argumentation frameworks.

Theorem 5.5. *Let L be a set of input labelings and $\mathbb{F}$ is the set of argumentation frameworks constructed by the procedure described in Algorithm 2 for the input labelings L.*

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

$$\forall F \in \mathbb{F} : F \textit{ produces } L$$

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

$$\forall G = (\mathsf{Arg}, \mathsf{R}) : G \textit{ produces } L \implies G \in \mathbb{F}$$

Proof of Theorem 5.5. It follows from Theorems 5.1 to 5.4 that the acceptance conditions are sound and complete for labelings with respect to conflict-free, admissible, complete and stable semantics. We need to prove that a set of acceptance conditions, and thus Algorithm 2, is sound and complete for any combination of input labelings with respect to different semantics. This follows from the fact that combining the acceptance conditions of each labeling is done by conjunction as shown below:

Let L be a set of input labelings and $\mathbb{F}$ be the set of argumentation frameworks constructed by Algorithm 2 for the input L.

Consider two input labelings $\ell_1, \ell_2 \in L$ with respect to different semantics σ_1 and σ_2. The corresponding acceptance conditions for any argument a are denoted C_{a,ℓ_1} and C_{a,ℓ_2}. The acceptance condition 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 acceptance conditions 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. □

As shown in Theorem 5.5 the algorithm based on the above defined acceptance conditions is sound and complete for all four semantics. That means, the algorithm computes exactly the set of argumentation frameworks that produce the input labelings. From the proof of Theorem 5.5 it also follows that the order in which the labelings are processed by the algorithm does not matter.

Proposition 5.1. *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 4 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 the monotonicity (see Theorem 5.6). 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 set of argumentation frameworks that satisfies all acceptance conditions can only stay the same or become smaller.

Theorem 5.6. *Let* Arg *be a set of arguments and* L_1, L_2 *are sets of labelings.* $\mathbb{F}_1$*,* $\mathbb{F}_2$ *denote the sets of argumentation frameworks constructed by the algorithm defined in Section 5.4 for the input labelings* L_1 *and* L_2 *respectively.*

The algorithm for constructing argumentation frameworks from labelings is monotonous.

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

Proof. see Appendix A.5 □

Lets say F_n denotes the set of all argumentation frameworks with n arguments. Then, the cardinality of this set is $|\mathsf{F}_n| = 2^{n^2}$. For our scenario this means the number of argumentation frameworks we have to consider grows exponentially with n^2. That also means the set of constructed argumentation frameworks $\mathbb{F}$ can be quite big as well. For example, for $n = 5$ there would already be over 33 million possible argumentation frameworks to consider. While the acceptance conditions can easily represent any number of argumentation frameworks, it might not be feasible to construct all of them. In that case we can consider the *minimal constructed argumentation frameworks* $\mathbb{F}_{min}$:

Definition 5.11. *Let L be a set of labelings and $\mathbb{F}$ denotes the set of argumentation frameworks constructed by Algorithm 2 for the input L.*

Then we define the set of minimal constructed argumentation frameworks $\mathbb{F}_{min}$ *as follows:*

$$\mathbb{F}_{min} = \{F = (\mathsf{Arg}, \mathsf{R}) \in \mathbb{F} \mid \forall G = (\mathsf{Arg}, \mathsf{R}') \in \mathbb{F} : \mathsf{R} \subseteq \mathsf{R}'\}$$

They are defined as the set of all argumentation frameworks $F \in \mathbb{F}$ that are minimal among the constructed argumentation frameworks. That means, these argumentation frameworks will only contain attacks that are necessary to satisfy all acceptance conditions and thus they will not contain any of the optional attacks.

If we are only interested in some argumentation framework that produces all input labelings instead of all frameworks, we could then just chose any argumentation framework $F \in \mathbb{F}_{min}$. This also raises an interesting question for future work (see Section 8): Can we efficiently construct some argumentation framework $F \in \mathbb{F}_{min}$?

5.5. Example

This section will showcase the application of the above introduced algorithm. For that, we will use the iterative version which is described in Algorithm 4. Unknown to us, there is the hidden argumentation framework as shown in Figure 11. Given are the set of arguments $\mathsf{Arg} = \{a, b, c, d, e\}$ and the set of input labelings $L = \{\ell_1, \ell_2, \ell_3\}$. The three input labelings are shown in Figure 12. Our goal is to construct all argumentation frameworks that produce these three labelings.

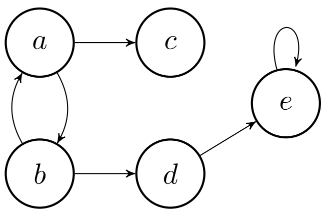
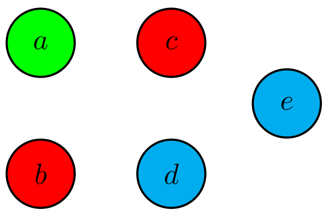
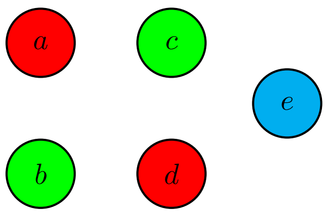


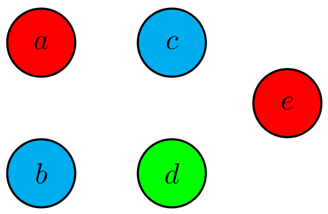
Figure 11: The hidden argumentation framework from which the input labelings are generated.



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



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



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

Figure 12: The three input labelings: ℓ_1 is admissible, ℓ_2 is complete and ℓ_3 is conflict-free.

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 acceptance condition for each argument. The labeling ℓ_1 is admissible, so we will use the method $AccCond_{ad}$ from Definition 5.4. The argument a is labeled in, that means it cannot be attacked by any in- or undec-labeled argument. b and c 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 acceptance conditions after the first iteration:

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

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. Recalling Definition 5.7, all atoms not present in the acceptance conditions are considered to be optional attacks. For example, the argument a can still be attacked by b or c and the resulting frameworks would still produce ℓ_1. There can also be many different combinations of attacks between the arguments b, c, d and e. The iterative algorithm would then return the set of acceptance conditions $C = \{C_a, C_b, C_c, C_d, C_e\}$.

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 method $AccCond_{co}$. 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 acceptance conditions C_{ℓ_2} for the labeling ℓ_2:

$$\begin{aligned}
C_{a,\ell_2} &= r_{ba} \vee r_{ca} \\
C_{b,\ell_2} &= \neg r_{bb} \wedge \neg r_{cb} \wedge \neg r_{eb} \\
C_{c,\ell_2} &= \neg r_{bc} \wedge \neg r_{cc} \wedge \neg r_{ec} \\
C_{d,\ell_2} &= r_{bd} \vee r_{cd} \\
C_{e,\ell_2} &= \neg r_{be} \wedge \neg r_{ce} \wedge r_{ee}
\end{aligned}$$

We now have to combine the conditions C from the last step with the conditions C_{ℓ_2} from this step. Since the argumentation frameworks we are looking for should produce all input labelings we use a conjunction to combine the acceptance conditions. 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 acceptance conditions:

$$\begin{aligned}
C_a &= \neg r_{aa} \land \neg r_{da} \land \neg r_{ea} \land (r_{ba} \lor r_{ca}) \\
C_b &= r_{ab} \land \neg r_{bb} \land \neg r_{cb} \land \neg r_{eb} \\
C_c &= r_{ac} \land \neg r_{bc} \land \neg r_{cc} \land \neg r_{ec} \\
C_d &= \neg r_{ad} \land (r_{bd} \lor r_{cd}) \\
C_e &= \neg r_{ae} \land \neg r_{be} \land \neg r_{ce} \land r_{ee}
\end{aligned}$$

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 can 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 acceptance conditions 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 acceptance conditions 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 acceptance conditions:

$$\begin{aligned}
C_{a,\ell_3} &= \neg r_{da} \\
C_{b,\ell_3} &= \neg r_{db} \\
C_{c,\ell_3} &= \neg r_{dc} \\
C_{d,\ell_3} &= \neg r_{dd} \\
C_{e,\ell_3} &= r_{de}
\end{aligned}$$

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

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

These acceptance conditions now represent exactly the set of argumentation frameworks that produce all three input labelings. The final step of the algorithm is now to construct these argumentation frameworks. Here, we use the the fact that the atoms used in the acceptance conditions directly correspond to attacks in an argumentation framework. The procedure for this step has been described in Algorithm 3. First, we compute the models of the acceptance conditions of each argument:

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

The next step is constructing the corresponding partial attack relations for each model with the method $toR(\mathcal{A})$ (see Definition 5.10). This is quite simple, since every atom directly represents an attack. For example, the model $\mathcal{A}_{a,1} = [r_{ba} \rightarrow true]$ 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:

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

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 argumentation framework $F_i = (\mathsf{Arg}, \mathsf{R}_i)$. The set $\mathbb{F}$ of all argumentation frameworks that produce L then consists of exactly these argumentation frameworks.
All constructed argumentation frameworks are shown in Figure 13. 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 produce the three input labelings. The minimal constructed argumentation frameworks, according to Definition 5.11, are the frameworks F_1, F_2, F_4 and F_5 as they only contain the minimal set of attacks that is necessary to satisfy all acceptance conditions.

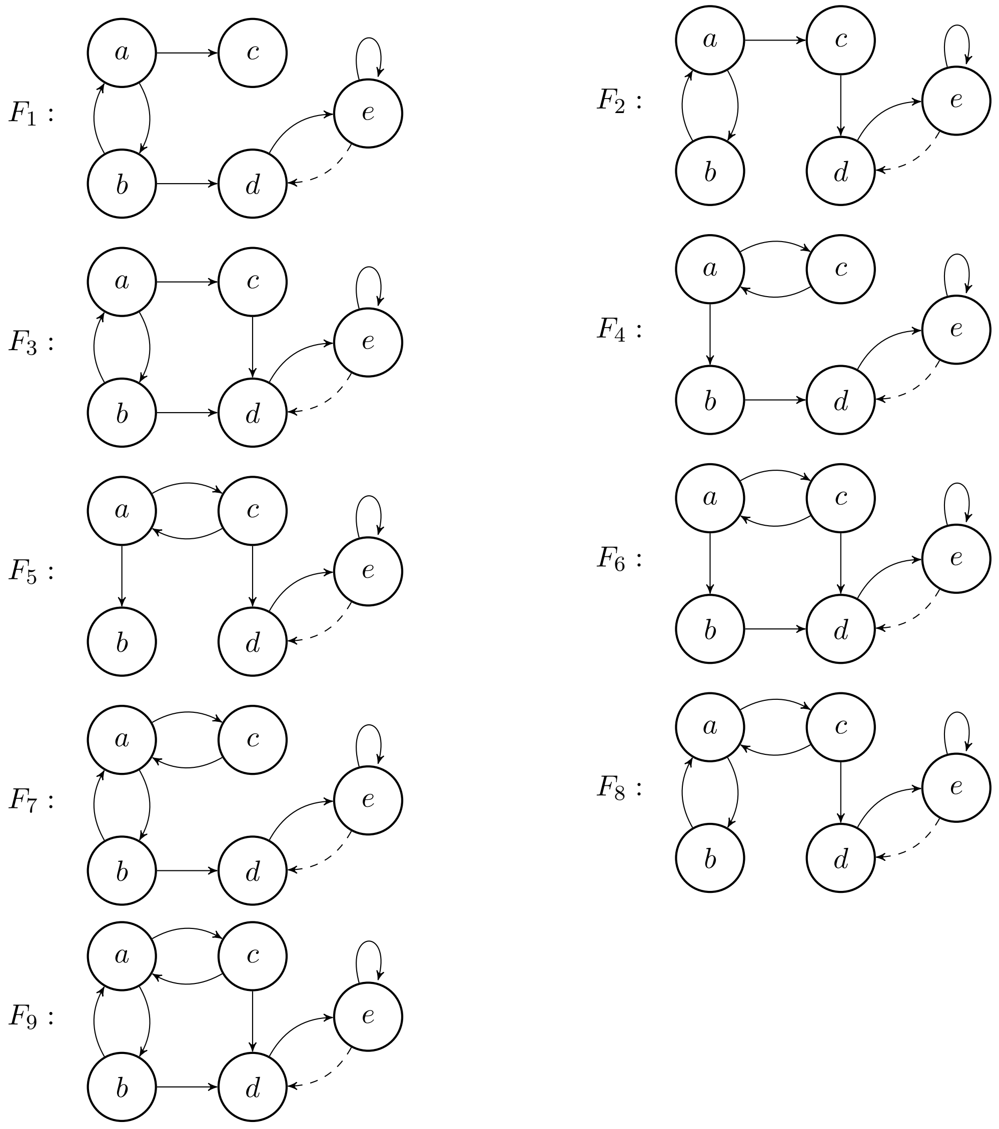


Figure 13: All 18 constructed argumentation frameworks which produce the labelings ℓ_1, ℓ_2 and ℓ_3. Each framework F_i represents two actual argumentation frameworks: F_i does not have the attack (e, d) and F_i' includes the optional attack (e, d).

6. Discussion

In the previous section we have introduced an algorithm for learning a set of argumentation frameworks from a given set of labelings. The algorithm computes a set of acceptance conditions for each labeling. These conditions represent the acceptability of each argument with regards to the labeling. Afterwards they are combined into one overall acceptance condition per argument. This set of overall acceptance conditions then represents all input labelings and can then be used to construct all argumentation frameworks that produce the input labelings.

In this section we will take a closer look on the proposed algorithm and discuss its strengths and weaknesses. Furthermore, we will highlight some differences to the existing algorithms from Riveret et al. [20] and Niskanen et al. [17]. Finally, we take a closer look at acceptance conditions and what information they hold.

We will first discuss some key advantages that distinguish the algorithm proposed in this thesis from the existing algorithms.

One key point is the procedure of the algorithms as highlighted in Section 4.1.4. The proposed algorithm works in an iterative way and can be applied on a stream of input labelings similarly to the Riveret algorithm. This is an important difference to the Niskanen algorithm 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 Niskanen algorithm where we would have to start over and learn all labelings again. The iterative approach also allows us to chose 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). This will be explored further in Section 8.

Another important difference is related to the conditions that are used to encode the labelings. The Niskanen algorithm also uses condition 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 4.2, these conditions are formulae over the set $attackAtoms(\mathsf{Arg})$, 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 $inAttacks(a)$, i.e. all attacks onto the argument. In other words, the Niskanen algorithm uses global conditions while the algorithm in this thesis 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 Niskanen algorithm. Furthermore, the local conditions will likely lead to better performance when computing the models that represent the argumentation frameworks.

The most important advantage of the proposed algorithm compared to the Riveret and Niskanen is the following: The proposed algorithm computes the set of all argumentation frameworks that produce the input labelings while both other algorithms only return one of these argumentation frameworks. The proposed 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 acceptance conditions and then combining and simplifying them. So, internally the algorithm only stores one set of mutually independent acceptance conditions 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 Riveret and Niskanen algorithm.

However, if we would want a similar result to the Riveret and Niskanen algorithms, we can at any point take some argumentation framework $F \in \mathbb{F}_{min}$ of the minimal constructed ones and consider this to be a solution of the addressed problem.

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 the Riveret and Niskanen 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 Riveret algorithm. This is a different approach compared to the Niskanen algorithm, 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 Riveret algorithm only allows grounded labelings as input, while the Niskanen algorithm accepts conflict-free, admissible, complete, stable, grounded and preferred extensions. The algorithm proposed in this thesis however only 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 their definition. The grounded labeling is defined as the minimal complete labeling and the preferred labelings are maximally complete. These cardinality constraints cannot be encoded in the acceptance conditions and thus it is not possible to learn these labelings with the proposed algorithm. The same applies to other semantics proposed in the literature, such as semi-stable [6] which maximizes the attacked arguments of an extension. Similarly, the naive semantics and the semantics based on it, like CF2 [2], 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 (see Section 8).

The Riveret algorithm uses grounded sub-framework labelings as input. As mentioned in Section 4.1 this somewhat trivializes 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. The algorithm proposed in this thesis does not rely on such labelings and is able to reconstruct argumentation frameworks from standard labelings.

Argumentation frameworks may contain self-attacking arguments. The Riveret algorithm 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. The algorithm proposed in this thesis 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 the proposed algorithm compared to the existing work.
The fact that the proposed 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 acceptance conditions 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. The algorithm proposed in this thesis cannot handle input noise, i.e. some 'wrong' labelings that are not actually produced by the hidden argumentation framework. This is something that the Riveret and Niskanen algorithm are able to handle. The Riveret algorithm 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 does not produce all of the correct input labelings. The Niskanen algorithm uses weighted MaxSAT encodings for the input extensions. That allows them so find the argumentation framework which produces the most of the extensions even if there are some noisy input extensions. It might be possible to apply a similar concept to the acceptance conditions which is also something that can be tackled in the future.

6.1. Acceptance Conditions

In the following we take a closer look at the acceptance conditions and what informaton they hold. In our algorithm the acceptance conditions are used to represent all information that can be extracted from an input labeling. They are then combined so that we obtain and keep one acceptance condition per argument internally. This set of acceptance conditions then essentially represents a set of argumentation frameworks. We can then construct these argumentation frameworks with the procedure described in Section 5.4.2. However, this is not necessary for the algorithm as we only need the acceptance conditions to represent and refine the corresponding set of argumentation frameworks.

Lets consider the iterative approach described in Algorithm 4. After any iteration we have a set of acceptance conditions C that represents the current state after learning a number of input labelings. Now, without constructing the corresponding argumentation frameworks we can extract a lot of useful information from them.

For example, lets consider the acceptance conditions from the example in Section 5.5 after learning the second labeling:

$$\begin{aligned}
C_a &= \neg r_{aa} \land \neg r_{da} \land \neg r_{ea} \land (r_{ba} \lor r_{ca})\\
C_b &= r_{ab} \land \neg r_{bb} \land \neg r_{cb} \land \neg r_{eb}\\
C_c &= r_{ac} \land \neg r_{bc} \land \neg r_{cc} \land \neg r_{ec}\\
C_d &= \neg r_{ad} \land (r_{bd} \lor r_{cd})\\
C_e &= \neg r_{ae} \land \neg r_{be} \land \neg r_{ce} \land r_{ee}
\end{aligned}$$

One observation we can make is that the acceptance conditions provide a good human readability since they only contain conjunctions and disjunctions. Furthermore, the acceptance conditions are by default in conjunctive normal form. That means we can easily determine attacks which are absolutely necessary to satisfy an acceptance condition. For example, the acceptance condition C_b tells us that there has to be an attack from a on b because r_{ab} is the only literal in one of the clauses. In addition to that, we can also easily see that the attacks (b, b), (c, b) and (e, b) cannot be in any argumentation framework that satisfies all acceptance conditions.

What we can also learn from this acceptance condition is that the attack (d, b) is still open. If we look at every acceptance condition we can then compute a set of attacks R_o that are still open, i.e. not included in any form in the acceptance conditions. From this set of attacks we can then obtain some useful information. First of all we can say that this set R_o represents the uncertainty in our system by compiling all optional attacks. We can also count the occurrences O_a of each argument $a \in \mathsf{Arg}$ in the set of attacks R_o. Then, the value O_a tells us how many optional attacks are associated with the argument a. In the current form of the algorithm this information is not used. However, this information could be useful if we are able to actively chose the next labeling to learn. One possibility would be for example: compute the value O_a for each argument $a \in \mathsf{Arg}$. We then chose the argument a with the highest value O_a. As the next input labeling we can then chose a labeling ℓ, where the argument a is labeled in, if possible. The idea here is that such a labeling would be more helpful, i.e. reduces more of the uncertainty as simply learning a random labeling. This idea could be combined with the elicitation scenario proposed in [15] where we are able to ask questions to some entity which knows the hidden argumentation framework. Some ideas on how the algorithm proposed in this thesis can be extended to the elicitation scenario in the future will be discussed in Section 8.

If we are only interested in some argumentation framework that satisfies the acceptance conditions we can quite easily obtain this by just computing some model $\mathcal{A}_a$ for each condition and combining them. The corresponding attack relation can then be assembled to an argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$ that satisfies all acceptance conditions, i.e. it produces all learned input labelings. This will be considerably faster than computing all argumentation frameworks.

Finally, it is also possible to compute the number of argumentation frameworks that produce the processed input labelings without actually constructing them. For that, we look at each argument $a \in \mathsf{Arg}$. The number of optional attackers N_a^o is then the number of atoms in $inAttacks(a)$ that are not present in the acceptance condition C_a, i.e. $N_a^o = |inAttacks(a) - atoms(C_a)|$. We then also compute the number of models N_a^m of C_a, i.e. $N_a^m = |\{\mathcal{A} \in Int_{atoms(C_a)} \mid \mathcal{A} \models C_a\}|$. The number of attack relations N_a^{at} associated with the argument a is then computed as $N_a^{at} = N_a^m * 2^{N_a^o}$. We have to use $2^{N_a^o}$ here because there are 2^n possible models for the optional acceptance condition (see Definition 5.7). The number of argumentation frameworks that satisfy the acceptance conditions is then given by the formula $N_{AF} = \prod_{a \in \mathsf{Arg}} N_a^{at}$. This is also described in the following proposition:

Proposition 6.1. *Let* Arg *be a set of arguments and* $C = \{C_a\}_{a \in \mathsf{Arg}}$ *is a set of acceptance conditions. Then we compute the number of argumentation frameworks* N_{AF} *that correspond to the set of acceptance conditions* C *as follows:*

$$N_{AF} = \prod_{a \in \mathsf{Arg}} N_a^m * 2^{N_a^o},$$

with $N_a^m = |\{\mathcal{A} \in Int_{atoms(C_a)} \mid \mathcal{A} \models C_a\}|$ *and* $N_a^o = |inAttacks(a) - atoms(C_a)|$.

This computation can be useful during the algorithm to get an overview of how many argumentation frameworks are still possible and also to decide whether we made progress after processing a labeling or not.

Example 6.1. *Lets consider the acceptance conditions C from the example in Section 5.5 after learning all three input labelings:*

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

Using the approach described above we can compute the following values for each argument:

$$\begin{aligned}
&a\text{:} \quad N_a^m = 3, \quad N_a^o = 0 \\
&b\text{:} \quad N_b^m = 1, \quad N_b^o = 0 \\
&c\text{:} \quad N_c^m = 1, \quad N_c^o = 0 \\
&d\text{:} \quad N_d^m = 3, \quad N_d^o = 1 \\
&e\text{:} \quad N_e^m = 1, \quad N_e^o = 0
\end{aligned}$$

The number of argumentation frameworks that can be constructed from these acceptance conditions, i.e. the number of argumentation frameworks that produce the input labelings is then given as $N_{AF} = \prod_{a \in \mathsf{Arg}} N_a^m * 2^{N_a^o} = 18$, *which is exactly the number of argumentation frameworks that have been constructed in the example in Section 5.5 (compare Figure 13).*

7. Evaluation

In this section we will conduct an evaluation of the approach introduced in Section 5. For that, we will consider the iterative version described in Algorithm 4. For the evaluation we will focus on the performance of the algorithm, i.e. the time the algorithm takes to learn the input labelings and to compute the set of argumentation frameworks that produce them. We will consider three different scenarios for the evaluation.
In the first scenario we will use some of the benchmark argumentation frameworks from the ICCMA'19 competition [3] to measure the performance of the algorithm in a more realistic application scenario.
In the second scenario we will use randomly generated argumentation frameworks with a fixed number of arguments in order to measure how the algorithm's performance scales with the number of arguments.
Finally, the third experiment also uses randomly generated argumentation framework. However, in this scenario we learn input labelings until there is only one argumentation framework left that produces the input labelings.

7.1. Experiment Setup

In the following we will describe the experiment setup that has been used for the evaluation of the approach for learning from labelings defined in this thesis.

System Setup. The experiments have been run on a machine with the operating system Windows 10. The machine has a 3.6 GHz AMD Ryzen 5 3600 CPU with 6 cores and a total of 32 GB of memory.

Implementation. For the evaluation the iterative version of the algorithm for learning argumentation frameworks from labelings has been implemented (see Algorithm 4). The implementation has been done in Java and is based on the *Tweety* library [21]. For the computation of the models of the acceptance conditions, needed for the construction of argumentation frameworks in the final step of the algorithm, the Java-based SAT-solver Sat4j [2] is used.

7.2. Experiment 1

In this experiment we will learn argumentation frameworks on the basis of some real data in order the get an idea of the general performance of the proposed algorithm. For that, we will use a subset of the argumentation frameworks of the ICCMA'19 benchmark [3]. In this experiment we consider only complete and stable labelings. We do not consider conflict-free or admissible labelings, since there tend to be too many of these labelings for argumentation frameworks of this size and it

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

takes too much time to compute them all. It would not be feasible to construct all argumentation framework since there will millions of frameworks that can produce the input labelings, so we only construct one argumentation framework that satisfies the acceptance conditions.

Scenario. From the benchmark set, we select all argumentation frameworks that have less than 1,000 arguments. We then have 276 argumentation frameworks with the number of arguments ranging from 4 to 1,000. For these argumentation frameworks, the number of complete labelings ranges from 1 to 178,682 while the number of stable labelings is between 0 and 2,552. For each argumentation framework, we compute the input labelings ℓ_σ with $\sigma \in \{co, st\}$ and randomly learn up to 200 input labelings. After learning we construct *one* argumentation framework $F \in \mathbb{F}$ that produces the input labelings.

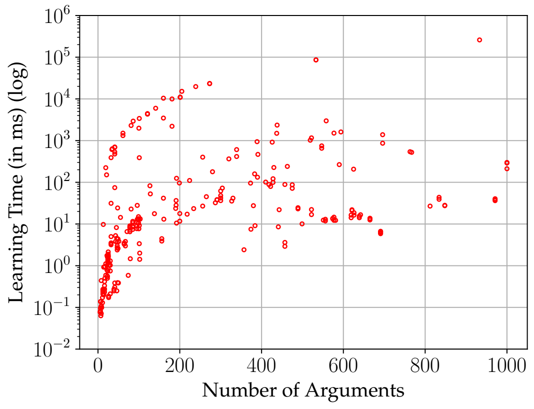
Metrics. In the following we define some questions that we want to answer with this experiment. Each question has a corresponding metric that will be investigated in order to answer the question:

1. *How much time does it take to learn all input labelings?*
   In order to answer this question we will measure the time t_{learn} it takes to process the input labelings. That means, we do not include the time for constructing the argumentation framework here. So, what me measure for this metric is the time it takes to assemble and combine the acceptance conditions for each argument and input labeling. We also do not apply any transformation (like CNF or DNF) to the acceptance conditions in this step.

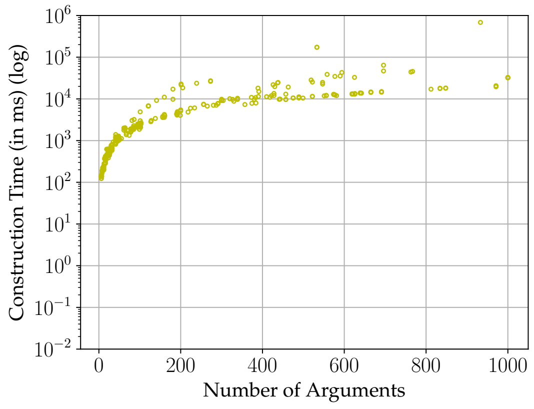
2. *How much time does it take to construct an argumentation framework that produces all input labelings?*
   For this question we measure the time t_{constr} it takes to construct some argumentation framework $F \in \mathbb{F}$ that produces the input labelings. We explicitly only construct one of these argumentation framework. This allows us to get an idea of how the construction time scales with the number of arguments.

3. *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 $|L|$, i.e. we compute the learning time per labeling (TPL)

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



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



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

Figure 14: Learning and construction time in relation to the number of arguments in the experiment with the ICCMA'19 instances.

7.2.1. Results

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

How much time does it take to learn all input labelings?
To answer this question we look at the time t_{learn} needed to process all input labelings. The total learning time in relation to the number of arguments in the hidden argumentation framework is shown in Figure 14(a). Note the logarithmic scale of the vertical axis. Each of the red dots represents one of the 276 instances that have been used for the experiment.
These instances have a median of 101 arguments and a median learning time of only $\approx 13\ ms$. For instances with less than 100 arguments the learning time ranges from 63 μs to $2.93\ s$. The longest learning time of $259.7\ s$ was measured for the instance with 933 arguments. In that case the maximum number of 200 labelings was learned, which explains this outlier.

Another thing we can observe is the cluster in the upper left. In this cluster, the learning time is larger by about a factor of 10^3 compared to other instances with a similar number of arguments. Upon closer inspection we find that in these instances 200 input labelings have been learned while the overall median number of learned labelings is 7.

Overall, we can see lots of outliers in both directions. An explanation for this might be the following: In the learning step, the complexity of assembling the acceptance condition depends mainly on how the arguments are labeled. This applies especially to complete input labelings. In that case, the acceptance condition for undec-labeled arguments is the most complex of all defined acceptance conditions.

However, if there are very few or no undec-labeled arguments in an input labeling then we do not have to compute this more complex condition. This can easily happen if we have argumentation frameworks without self-attacks or other odd-cycles.

Apart from these outliers we can still clearly see a trend of increased learning time with an increasing number of arguments. This is to be expected, since more arguments means more acceptance conditions and also more atoms in each acceptance condition.

How much time does it take to construct an argumentation framework that produces all input labelings?
In this experiment we constructed some argumentation framework that produces all input labelings. That means we have to compute some model of each acceptance condition and assemble the corresponding partial attack relations and combine them into one argumentation framework. The time t_{constr} it takes to do this has been measured and the results are shown in Figure 14(b). Note again the logarithmic scale of the vertical axis.
The time for constructing a consistent argumentation framework ranges from 123.5 ms to 682.7 s with a median of 2.63 s. Compared to the learning the, the constructing times are on average larger by a factor of about $6 \cdot 10^2$. So, in this experiment it takes considerably longer to construct an argumentation framework compared to learning the input labelings. This can be explained by the need to compute a model for each acceptance condition, which is by far the larger factor contributing to the construction time. This is due to the fact that the acceptance conditions are unstructured (i.e. not in DNF or CNF) and computing a model for such formulae takes more time.

We can again observe the same cluster of instances in the upper left. Like before, in these instances the maximum of 200 input labelings has been learned. This also influences the construction time, because more labelings means the acceptance condition for each argument is longer, as the acceptance condition for an argument is the conjunction of the acceptance conditions of the argument for each input labeling. However, we can observe that there are less outliers overall in any direction for the construction time compared to the learning time.

Ignoring these outliers we can clearly see an increase of construction time with an increasing number of arguments, which makes sense. Let n be the number of arguments of the instance. Then, the complexity of computing the models of all acceptance conditions is in $O(n^2)$ and the complexity of constructing an argumentation framework from a given model is also in $O(n^2)$. So, overall we can hypothesize a complexity in $O(n^2)$ for constructing an argumentation framework for a given set of acceptance conditions.

How does the time to learn a single input labeling scale with the number of arguments?
In order to remove the factor of the number of learned labelings $|L|$, we consider the time per labeling $t_{TPL} = \frac{t_{learn}}{|L|}$. Figure 15 shows the time per labeling in relation to

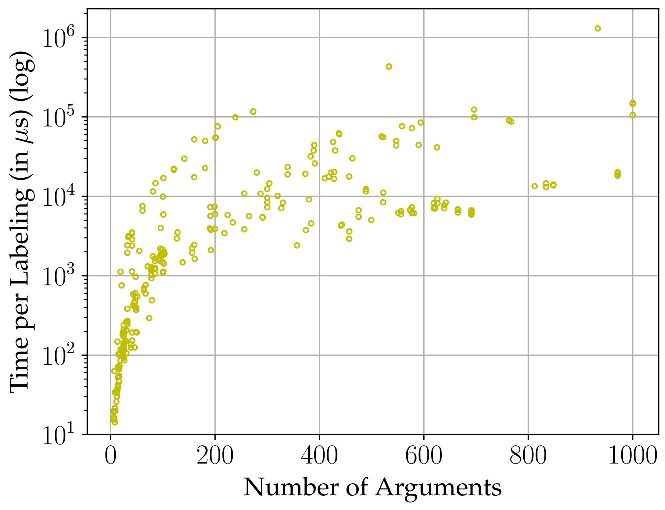


Figure 15: Learning time per labeling t_{TPL} in relation to the number of arguments in the experiment with ICCMA'19 instances.

the number of arguments of the corresponding instance. Note the logarithmic scale of the vertical axis.

The time per labeling ranges from $14.28\ \mu s$ to $1.3\ s$ with a median of $2.9\ ms$ per labeling. The variance is still quite high, however not as high as in the case of the total learning time.

We still have the same heavy outliers as well as the cluster in the upper left. So, even if we consider the time per labeling, instances where we learn the maximum of 200 input labelings have a worse performance than instances with less labelings. That means, there must be some correlation between the number of complete/stable labelings of an argumentation frameworks and the complexity of computing the acceptance conditions. A possible explanation might be that a higher number of labelings usually means we have less in-labeled arguments and more undec-labeled arguments. The acceptance conditions for undec-labeled arguments are more difficult to compute and thus, the time per labeling increases the more undec-labeled arguments there are in an input labeling.

Despite these outliers we can see a clear correlation between the time per labeling t_{TPL} and the number of arguments n. Similarly to before, a complexity of $O(n^2)$ can be hypothesized. This makes sense, since for each labeling we have to compute and combine the acceptance conditions for all n arguments and each acceptance condition consists of up to n atoms.

A possibility for future work is to utilize parallelization when computing the acceptance condition. Since there is no overlap between the atoms of each acceptance condition, we could easily parallelize the learning step to further improve performance (see Section 8).

7.3. Experiment 2

The main goal of this experiment is to investigate how the proposed algorithm scales with the number of arguments. For that, we learn admissible, complete and stable labelings. We do not consider conflict-free labelings in this experiment.

Scenario. For the experiment, we randomly generate 100 argumentation frameworks $F = (\mathsf{Arg}, \mathsf{R})$ with $n = |\mathsf{Arg}| = 4, 8, 12, 16$ arguments. The set of attacks R is generated as follows: for each pair of arguments a, b the attack (a, b) has a probability of $p_{att} = \frac{1}{n}$ to be included in R. For each argumentation framework we compute all input labelings ℓ_σ with $\sigma \in \{ad, co, st\}$. Then, we iteratively learn all of these input labelings and measure the time the algorithm takes to learn them.

Metrics. We define the following questions that we want to answer with this experiment. Each question has a specific metric that will be investigated in order to answer the question:

1. *Is it always possible to reconstruct exactly the hidden argumentation framework?*
   In this experiment we learn all admissible, complete and stable labelings. That raises the question whether there is always exactly one argumentation framework that produces these labelings or not. If there were only one argumentation framework, then the set of argumentation frameworks $\mathbb{F}$ that are consistent with the acceptance conditions would only contain one element. So, to answer this question we observe the number of argumentation frameworks

   $$n_{afs} = |\mathbb{F}|$$

   that can be constructed after learning the input labelings.

2. *How many argumentation frameworks can still be constructed after learning all input labelings?*
   This question is related to the previous one. In cases where there is more than one argumentation framework that produces the input labelings, we consider the number of argumentation frameworks

   $$n_{afs} = |\mathbb{F}|$$

   that can be constructed. This allows us to get an idea of how the number of consistent argumentation frameworks scales with the number of arguments $|\mathsf{Arg}|$.

3. *How much time does is take to learn all input labelings?*
   In order to answer this question we will measure the time t_{learn} it takes to process the input labelings. We do not construct any argumentation framework

in this experiment, as we are only interested in the number of consistent argumentation frameworks. So, we only measure the time it takes to assemble and combine the acceptance conditions for each argument and input labeling.

4. *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. Since the number of arguments is fixed to $n = 4, 8, 12, 16$ we can get a clearer view on how the performance scales compared to the previous experiment. As a metric, 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 $|L|$, i.e we compute the learning time per labeling (TPL)

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

7.3.1. Results

In the following we look at the results of the second experiment and answer the above defined questions.

Is it always possible to reconstruct exactly the hidden argumentation framework?
In the experiment we learned all admissible, complete and stable labelings of the hidden argumentation framework. That raises the question if this is enough information to reconstruct exactly the hidden argumentation framework. For that, we consider the number of argumentation frameworks $n_{afs} = |\mathbb{F}|$ that can be constructed after learning the input labelings. The experiment for argumentation frameworks with n=4 arguments shows already that we are not guaranteed to achieve this. Only in 5 out of 100 cases were we able to find exactly the hidden argumentation framework, i.e. $n_{afs} = 1$. In all other cases we could still construct multiple argumentation frameworks after learning.

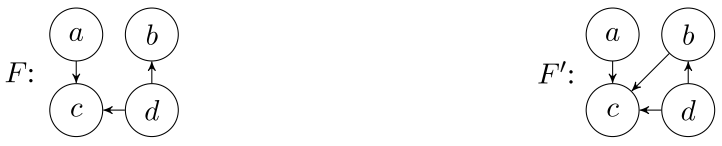


Figure 16: The hidden argumentation framework F and some constructed argumentation framework F' that also produces all ad, co and st labelings of F.

Consider for example one of the instances for n=4, which is depicted in Figure 16. F is the randomly generated hidden argumentation framework. However, after learning all the input labelings we can still construct 128 different argumentation frameworks, including F, that all produce these input labelings. One of these argumentation frameworks is F'.

This result shows why the main idea of this approach: "Learning *all* argumentation frameworks that produce the input labelings" is relevant. When learning labelings there can be multiple correct solutions and it is not possible to determine which solution is the original hidden framework. Learning all frameworks allows us to not arbitrarily chose one solution and accidentally discard the original framework. So, if we obtain further labelings or other information later, we are still able to reconstruct exactly the hidden argumentation framework.

How many argumentation frameworks can still be constructed after learning all input labelings?
To answer this question we consider the number of argumentation frameworks n_{afs} that can be constructed after learning all input labelings. For n=4 we have a median of 34 argumentation frameworks that can be constructed after learning. For n=8 we already have a median of 262,144 constructible argumentation frameworks. In addition to that, there was no instance for n=8, where $n_{afs} = 1$ and in the worst case there where over 245 $billion$ argumentation frameworks that produce the input labelings. For the instances with 12 or 16 arguments it was not possible to reliably measure the number of argumentation frameworks with the standard Java functionality.

Still, from the results for 4 and 8 arguments we can already see that the number of constructible argumentation frameworks increases heavily. For n arguments we have 2^{n^2} possible argumentation frameworks, i.e. the number of possible argumentation frameworks increases exponentially with the number of arguments. Based on this, we can hypothesize that the number of argumentation frameworks that can be constructed increases in a similar order of magnitude with the number of arguments.

How much time does is take to learn all input labelings?
We consider the total time t_{learn} that is used for learning. This includes the time for assembling and combining the acceptance conditions for each input labeling. Figure 17 shows four boxplots, one for each configuration $n = 4, 8, 12, 16$. The whiskers represent the 5% and 95% quantiles. In addition to that, for each configuration the yellow dots depict the time for each of the 100 individual instances with n arguments. Note the logarithmic scale of the vertical axis.

One thing we can observe is, the range of values increases significantly with the number of arguments. While the interquartile range (IQR) is 0.18 ms for n=4 it increases to 0.94 ms, 19.8 ms and 274.8 ms for n=$8, 12, 16$ respectively. This can also be noticed by observing the dots for the individual instances. It gets significantly more straight as the number of arguments increases.

We can also conclude from this, that the learning time for an instance can vary greatly based on other factors than the number arguments. Potential factors include the number of input labelings or the complexity of the individual instance.

Overall, we can hypothesize an exponential increase of the total learning time with the number of arguments. This can be explained by an increase in the time per labeling, but it is also due to a significant increase in the number of admissible labelings with the number of arugments of an argumentation frameworks.

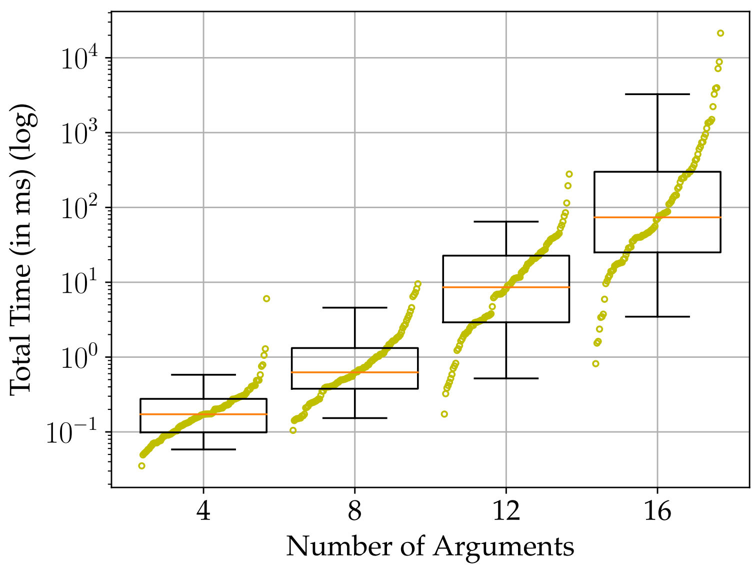


Figure 17: The total learning time t_{learn} in relation to the number of arguments.

How does the time to learn a single input labeling scale with the number of arguments? Just like in the experiment before, we consider the learning time per labeling t_{TPL}. With the number of arguments fixed to $n = 4, 8, 12, 16$ we will get a clearer view of the scaling of the learning time. The median learning time per labeling is shown in Figure 18.

For the instances with 4 and 8 arguments the median time per labeling is 23 μs and 21 μs respectively. For 12 arguments we have a median time per labeling of already 89.6 μs, while for the 100 random instances with 16 arguments the median time per labeling was 275.3 μs.

The decrease of the time per labeling from n=4 to n=8 suggests that there is some computational overhead in the implementation, that we may be able to optimize. However, in general the time per labeling increases with the number of arguments as one might expect. Considering the results in Figure 18 and the fact that the complexity of computing the acceptance conditions for a labeling is in $O(n^2)$, we may expect the time per labeling to scale in that order of magnitude. As noted before, the process of learning a labeling can be accelerated by parallelization in the future.

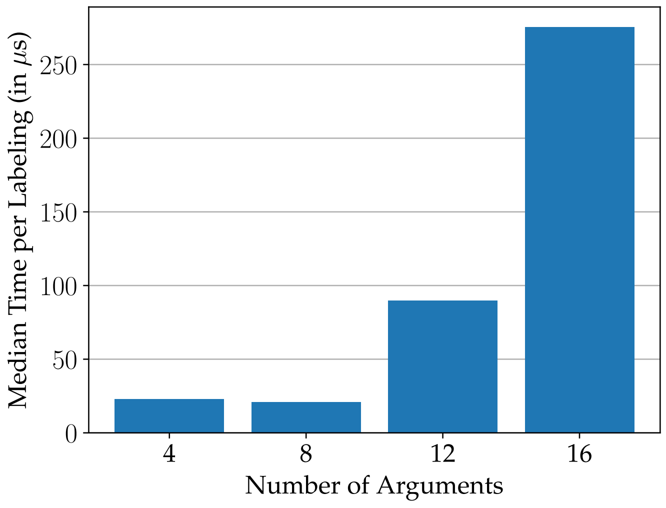


Figure 18: Median Learning Time per Labeling in relation to the number of arguments.

7.4. Experiment 3

In this experiment we want to investigate whether we are consistently able to reconstruct the hidden argumentation framework used to generate the input labelings under the assumption that we have access to all conflict-free, admissible, complete and stable labelings. If it is possible to reconstruct the hidden argumentation framework, we measure how many labelings are needed for that and how long it takes to do so.

Scenario. For the experiment, we again randomly generate 100 hidden argumentation frameworks $F = (\mathsf{Arg}, \mathsf{R})$ with $n = |\mathsf{Arg}| = 4, 8, 12, 16$ arguments. The set of attacks R is generated as follows: for each pair of arguments a, b the attack (a, b) has a probability of $p_{att} = \frac{1}{n}$ to be included in R. For each argumentation framework we compute the input labelings ℓ_σ with $\sigma \in \{cf, ad, co, st\}$. Then, we randomly learn input labelings until there is only argumentation framework left that produces the already processed input labelings, i.e. $|\mathbb{F}| = 1$ where $\mathbb{F}$ is the set of argumentation frameworks that are consistent with the internal acceptance conditions of the algorithm. This argumentation framework is then also exactly the hidden argumentation framework used for computing the input labelings. If we have $|\mathbb{F}| > 1$, then we continue learning until no input labeling is left.

Metrics. In the following we define some questions that we are going to answer with this experiment. Each question has a corresponding metric that will be investigated in order to answer the question:

1. *Is it always possible to reconstruct exactly the hidden argumentation framework?*
   In the case of extensions we do not always have a unique solution when learning all cf, ad, co and st extensions [16]. For labelings however, it is not clear yet if that is also the case. For that reason, we consider the number of argumentation frameworks
   $$n_{afs} = |\mathbb{F}|$$
   that are constructed by the algorithm after learning all input labelings and examine whether there is a unique solution or not, i.e. we check if $n_{afs} = 1$.

2. *How many labelings need to be learned to reconstruct the hidden argumentation framework?*
   In cases where we are able to reconstruct the hidden argumentation framework we consider the number of labelings $|L|$ that have been learned until that point. Based on this we can also investigate how this number scales with the number of arguments.

3. *How much time does it take to find the hidden argumentation framework?*
   For this question we will look at the time t_{learn} it takes to process the input labelings and the time t_{constr} it takes to actually construct the final argumentation framework. The metric we will consider for this question is then
   $$t_{total} = t_{learn} + t_{constr}.$$

7.4.1. Results

In the following we consider the results of the third experiment by examining the previously defined questions and their corresponding metric.

Is it always possible to reconstruct exactly the hidden argumentation framework?
In this experiment we learned conflict-free, admissible, complete and stable labelings at random until only one argumentation framework was left that produces them, i.e. until $n_{afs} = 1$. While the existence of a unique solution has nothing to do with the algorithm itself, it is still interesting to investigate in order to gain a better understanding of the scenario of learning labelings itself.

For all sizes of the argumentation framework $n = 4, 8, 12, 16$ we were able to reconstruct exactly the hidden argumentation framework for every instance. So, compared to the previous experiment, where we did not learn any conflict-free labelings, we were able to consistently reconstruct the hidden argumentation framework. That means, the conflict-free labelings play an important role in this. Of course, in order to verify that it holds for every argumentation framework that it is uniquely identified by its conflict-free, admissible, complete and stable labelings, a formal proof is needed. This will be left for future work.

How many labelings need to be learned to reconstruct the hidden argumentation framework? After establishing that we are consistently able to reconstruct the hidden argumentation framework in this scenario, we now consider the number of labelings that were necessary to determine the exact argumentation framework that produced them. The median number of labelings needed for that in relation to the number of arguments is shown in Figure 19.

For the instances with 4 arguments we needed to learn a median of 7 random labelings until we found the hidden argumentation framework. These instances had a median of 14 labelings in total, so for n=4 we had to learn 50% of the total labelings to reconstruct the hidden argumentation framework. For n=8 we already had to learn a median of 22 labelings. However, in total these instances had a median of 87 labelings thus we only had to learn about 25% of the labelings. In the case of n=12 we have 39 learned labelings of 732 in total, i.e. about 5.3%. Finally, for n=16 we learned a median of 44.5 out of 4578 labelings, which is about 0.98%.

What we can observe here, is that the total number of labelings with respect to the considered semantics increases significantly more than the number of labelings we need to learn to reach $n_{afs} = 1$. Looking at Figure 19 we can hypothesize a logarithmic growth of the number of learned labelings with an increasing number of arguments.

Another thing to note is that we learned labelings at random. That means, we are potentially able to reduce the number of learned labelings even further by optimizing how we chose the labelings that we learn. Some ideas in this regard will be touched on in Section 8.

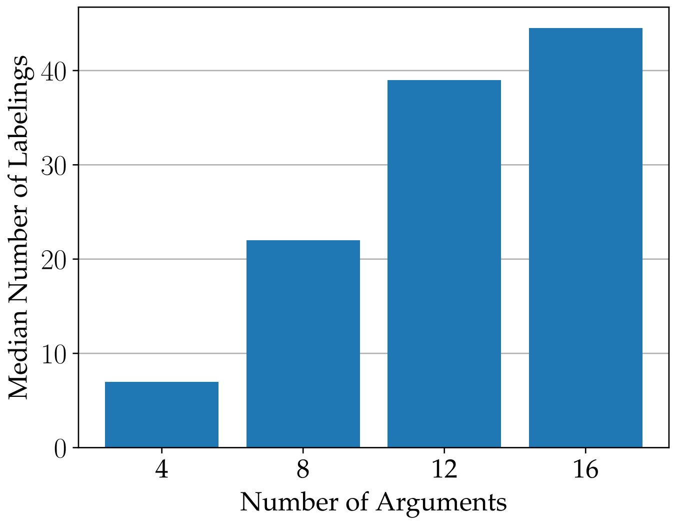


Figure 19: The median number of learned labelings in relation to the number of arguments of the hidden argumentation framework.

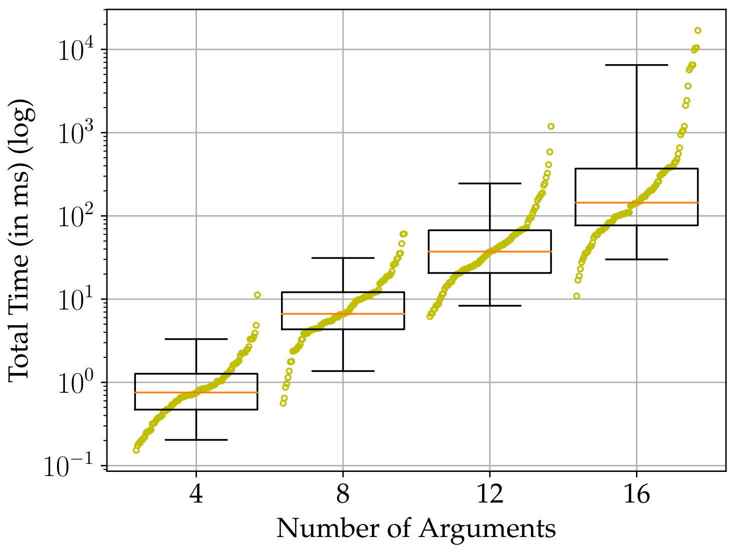


Figure 20: The total time t_{total} in relation to the number of arguments.

How much time does it take to find the hidden argumentation framework?
We consider the time it takes to learn all necessary input labelings and to construct the resulting argumentation framework. The results are shown in Figure 20. Similarly to earlier, we have four boxplots, one for each configuration $n = 4, 8, 12, 16$. The whiskers represent the 5% and 95% quantiles. In addition to that, for each configuration the yellow dots depict the time for each of the 100 individual instances with n arguments. Note the logarithmic scale of the vertical axis.

We can observe, that the range of values increases significantly with the number of arguments. For instance, for n=4 we have total times ranging from 0.15 ms to 10 ms, while we have a range from 10 ms to 20 s for n=16. Similarly, the median total time also increases polynomially with the number of arguments. This increase is due to an increase in the time per labeling and the increased number of processed labelings as investigated in the previous paragraph.

On average, the construction part makes up between 0.06% and 2.8% of the total time. So, in contrast to the first experiment, the construction of the argumentation framework only makes up a very small amount of the total time. The reason for that is as follows: In this experiment, after each input labeling we compute the number of argumentation frameworks that are consistent with the acceptance conditions (as described in Section 6.1). That means, we have to compute the models of each acceptance condition after each labeling. For that, we transform the conditions into DNF when combining acceptance conditions, which increases the learning time. As a side effect, the model computation in the construction step will be significantly easier and thus takes very little time.

8. Future Work

In the following we will take a look at some open questions that may be answered in the future as well as some ideas on how to extend the proposed approach. Furthermore, we take a closer look at an idea on how to adapt the approach to an elicitation scenario, similar to the one described in [15].

8.1. Open Questions

There are a couple of open questions regarding the approach presented in this thesis, that might be worth it to investigate in the future. We take a closer look at two of these questions.

How can we efficiently compute an argumentation framework $F \in \mathbb{F}_{min}$?
In the proposed algorithm we compute the set of all argumentation frameworks $\mathbb{F}$ that produce a set of input labelings L. In Section 5.4.4 we introduced the notion of minimal constructed argumentation frameworks $\mathbb{F}_{min} \subseteq \mathbb{F} = \{F = (\mathsf{Arg}, \mathsf{R}) \in \mathbb{F} \mid \forall G = (\mathsf{Arg}, \mathsf{R}') \in \mathbb{F} : \mathsf{R} \subseteq \mathsf{R}'\}$, i.e. the argumentation frameworks that satisfy the acceptance conditions with a minimal number of attacks. These argumentation frameworks are particularily interesting, because every attack has a specific purpose and is absolutely necessary to produce the input labelings. If we know the set of all argumentation frameworks $\mathbb{F}$ that produce the input labelings, it is rather trivial to filter out those that are minimal among them. However, it might be possible to compute some or maybe even all minimal argumentation framework in a more efficient way.

Is is always possible to reconstruct the exact argumentation framework from the set of all cf, ad, co and st input labelings?
In the third experiment of the evaluation we learned random conflict-free, admissible, complete and stable labelings until there was only argumentation framework left that produces them. In the experiment, we were always able to reach the point of having only one argumentation framework left. On the other hand, in the second experiment we learned all admissible, complete and stable labelings, but there was not always an unambiguous solution. That raises the question under which circumstances it is possible to exactly reconstruct the argumentation framework for a set of labelings. To verify that, a formal proof is required.

For extension-based semantics there have been multiple studies [16, 18] and this question has been answered exhaustively, but for labeling-based semantics no such investigation exists as of yet.

8.2. Extending the Approach

There are also various possibilities to extend or adapt the introduced approach in the future. Some of those ideas will be discussed in the following.

Extensions instead of Labelings. In its current form the proposed algorithm works with labelings as input. It is in fact necessary to have labelings, since we need the distinction between out- and undec-labeled arguments to compute the acceptance conditions. This distinction is not made in extension-based semantics. However, it might be possible to redefine the acceptance conditions in such a way that we only need extensions as input.

Negative Examples. The algorithm by Niskanen et al. [17] works with positive and negative examples in the form of extensions as input. Niskanen et al. simply use a negated condition to express that an example is negative. In our algorithm it is not that simple. The fact that we use local acceptance conditions for each argument, means we cannot just negate them if a labeling is considered as a negative example. In order to fully capture a negative example we would need to find another way to do so.

Other Semantics. In Section 6 it was already mentioned that it is not possible to encode all information of labelings with respect to semantics like grounded, preferred or any naive-based semantics. What these semantics have in common is that they all rely on cardinality constraints. We can consider the acceptance conditions of the algorithm to be *local*, i.e. from the perspective of individual arguments. However, in order to encode the cardinality constraints this local level is not enough. One possibility to address this problem is introducing some form of *global* conditions. While the global conditions give us more options to encode information, they would also increase the complexity of computing the models that correspond to the argumentation frameworks.

Other Semantic Information. The concept of acceptance conditions as used in this thesis is quite versatile. Currently, the acceptance conditions are only used to encode the information from input labelings. However, they can easily be adapted to be able to encode other semantic information about arguments. For instance, we could encode self-attack freeness of the hidden argumentation framework by adding the literal $\neg r_{aa}$ to the acceptance condition for each argument $a \in \mathsf{Arg}$. We could also encode that an argument a must be unattacked by setting its acceptance conditions to $C_a = \bigwedge_{b \in \mathsf{Arg}} \neg r_{ba}$.

It might even be possible to encode more complex semantic information, like credulous or sceptical acceptance of an argument a with respect to a semantics σ. An argument a is considered credulously accepted with respect to σ semantics, if it holds that a occurs in every σ-extension of an argumentation framework. Sceptical acceptance means, the argument occurs in any σ-extension of the argumentation framework. Encoding this information depends on the semantics and it might be interesting to come up with some possibilities to achieve this. It might also be necessary to utilize some form of global conditions for this.

Algorithmic Optimizations. The implementation of the approach used for the evaluation can be optimized further to increase the performance significantly. The concept of local acceptance conditions is very helpful in this regard. The acceptance conditions are defined in such a way that they are mutually independent. That means, there is no overlap in atoms between them. Subsequently, the computation of the models of the acceptance conditions is also independent of each other and can thus be done in parallel. Furthermore, even the computation of the acceptance conditions for an input labeling can be parallelized.

The acceptance conditions also have two very distinctive forms: the condition is either a conjunction of negative literals, or are disjunction of positive literals. It might be possible to exploit this during the model computation and speed up this process.

8.3. Eliciting Argumentation Frameworks From Labelings

In this section we will take a closer look at an idea on how to adapt the proposed approach for learning argumentation frameworks from labelings to incorporate the concept behind elicitation [15]. The argumentation framework elicitation problem describes a scenario where we have an entity with a hidden argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$. The attack relation of this argumentation framework is unknown to us. However, we are allowed to ask the entity different questions, e.g. *"Is $E \subseteq \mathsf{Arg}$ a σ-extension of F?"*. The entity answers truthfully and we may then use the answer and elicit the hidden argumentation framework.

In the third experiment of the evaluation we randomly learned labelings until only one argumentation framework that could produce them was left. We measured how many labelings were necessary to reach this goal. However, since the labelings to learn were picked at random this raises the question whether we are able to minimize the number of learned labelings to reach the hidden argumentation framework.

In order to answer this question we define a scenario which combines the idea of eliciting argumentation frameworks with the algorithm proposed in this thesis as follows:

There is an entity which knows a hidden argumentation framework $F = (\mathsf{Arg}, \mathsf{R})$. We want to elicit the hidden argumentation framework. For that, we use the iterative algorithm described in this thesis and we are able to ask the entity questions. That means, we begin by asking a question. As an answer we obtain some labeling of the hidden argumentation framework We can then learn this labeling and update the acceptance conditions. Then, we repeat these steps until we reach the point where there is only one argumentation framework left that is consistent with the acceptance conditions. This would then be the hidden argumentation framework which we have successfully elicited.

There are many possible questions or queries that we can send to the entity, for example:

- Is ℓ_σ a labeling of F with respect to σ semantics?
- Give me a labeling ℓ_σ of F with respect to σ semantics.
- Give me a labeling ℓ_σ of F where the argument a is labeled in.

In order to do better than just learning random labelings we have to ask the right questions. So, in each iteration we have to decide which question is the best, i.e. which question gives us the most information. For that, we can define some metric that helps us decide which exact query is the most helpful in finding the hidden argumentation framework based on the current status of the acceptance conditions.

In the following lets assume we only have the query *Give me a labeling ℓ_σ of F where the argument a is labeled* in. That means we have to decide for which argument $a \in \mathsf{Arg}$ we want to make this query. For that, we could consider the metric *number of occurrences in optional attacks O_a* as described already in Section 6.1. This metric simply counts the occurrences of an argument in the atoms which are not present in the acceptance conditions. We can then say the argument for which this metric is highest is the most uncertain, i.e. there are the least restrictions on the attacks related to this argument by the acceptance conditions. That means, by getting a labeling where this argument is labeled in we have a good chance of ruling out or confirming many of these optional attacks. For that argument we then ask the above question and learn the labeling that we obtain from the entity.

Algorithm 5 General idea on how an elicitation algorithm for the above scenario could be structured.

```
1: **init** set of conditions $C = \{C_a = \top \mid a \in \mathsf{Arg}\}$, set of arguments Arg.
2: $\mathbb{F} \leftarrow constructFrameworks(C)$
3: **while** $|\mathbb{F}| > 1$ **do**
4:     $a \leftarrow findMostUncertainArgument(\mathsf{Arg}, C)$
5:     $\ell_\sigma \leftarrow$ Give me a labeling $\ell_\sigma$ of $F$ where the argument $a$ is labeled in.
6:     $C \leftarrow learnLabeling(C, \ell_\sigma)$
7:     $\mathbb{F} \leftarrow constructFrameworks(C)$
8: **end while**
9: $F \leftarrow constructFramework(C)$
10: **return** $F$.
```

Algorithm 5 shows how this procedure could look like. We begin with a set of arguments Arg and a set of empty acceptance conditions C. Then we learn labelings until only one argumentation framework can be constructed for the conditions C. We begin each iteration by determining which argument is the most uncertain, i.e. we compute for each argument a chosen metric, e.g. the above mentioned number of occurrences in optional attack, and the argument a with the highest score is selected. For that argument we query the entity for a labeling where a is labeled in. We then learn this labeling and check the size of the set of consistent argumentation frameworks.

If we reach the point $|\mathbb{F}| = 1$ we stop and return that argumentation framework. This is exactly the hidden argumentation framework of the entity.

Example 8.1. *Lets for instance consider the set of arguments* $\mathsf{Arg} = \{a, b, c, d, e\}$ *and the set of acceptance conditions* C *after already learning two labelings:*

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

Our goal is to elicit the hidden argumentation framework of the entity. Using the methods described in section 6.1 we can determine that currently there are 576 argumentation frameworks consistent with these conditions.

We start in the while loop of the above defined abstract algorithm. That means we compute for each argument, how often it occurs in the optional attacks. In this example the arguments a, b *and* c *have one optional attack they occur in, i.e.* (d, a), (d, b) *and* (d, c) *respectively. The argument* e *has the two optional attacks* (e, d) *and* (d, e) *and the argument* d *has 6 optional attacks related to it. Using the number of related optional attacks as a metric, we would then decide that* d *is the argument with the most uncertainty.*

We can then ask the entity to give us a labeling where d *is labeled* in. *The entity might then for instance return the labeling* $\ell_{cf} = \{\mathsf{in} = \{d\}, \mathsf{out} = \{e\}, \mathsf{undec} = \{a, b, c\}\}$. *From this labeling we could then learn that* d *does not attack itself or* a, b *and* c. *We would also learn that* d *must attack* e. *This is also captured in the acceptance conditions after learning this labeling and combining it with the prior acceptance conditions:*

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

After learning this labeling, there are still 18 argumentation frameworks in $\mathbb{F}$ *that are consistent with the acceptance conditions and we would continue by finding the argument with the most uncertainty again. If we had instead learned some random labeling, there is a good chance that we would have made little or no progress towards reducing the size of* $\mathbb{F}$.

As we have seen, by being able to make specific queries to an entity we can try to minimize the number of learned labelings. In the future, one could come up with other interesting questions and investigate what kind of metric is most useful to decide how to ask them.

9. Conclusion

In this thesis we looked at the scenario of learning argumentation frameworks from labelings. More specifically, we considered a scenario where we learn a set or stream of input labelings with respect to different semantics with the goal of constructing the set of all argumentation framework that produce these input labelings.

In the course of this thesis we developed a novel approach to learn argumentation frameworks from labelings. The approach uses mutually independent acceptance conditions for each argument, that represent the information about acceptability obtained from the input labelings. The acceptance conditions consist of atoms representing attacks in an argumentation framework. That means, the models of the acceptance conditions can be used to construct corresponding argumentation frameworks that satisfy the conditions and thus produce the input labelings. The approach allows us to learn labelings in an iterative way and at any point we can consider the acceptance conditions to obtain an intermediate result. Furthermore, the acceptance conditions are very versatile and could also be used to encode other semantic information like credulous acceptance of arguments or self-attack freeness of the argumentation framework.

The proposed approach was implemented and evaluated with different data sets. The evaluation shows that the algorithm is able to learn up to 200 labelings and construct an argumentation framework with hundreds of arguments that produces them in less than 3 seconds. Moreover, the performance of the algorithm scales only quadratically with the number of arguments and could easily be improved even further by utilizing parallelization.

A. Proofs

A.1. Conflict-free Semantics

Theorem 5.1. *Let ℓ be a conflict-free input labeling and $C = \{C_a = AccCond_{cf}(a, \ell) \mid a \in \mathsf{Arg}\}$ is the set of acceptance conditions for the input labeling ℓ. With $\mathbb{F}$ we denote the set of argumentation frameworks consistent with C.*

The acceptance conditions are sound *for conflict-free input labelings, i.e.*

$$\forall F \in \mathbb{F} : \ell \text{ is conflict-free} \implies F \text{ produces } \ell$$

The acceptance conditions are complete *for conflict-free input labelings, i.e.*

$$\forall G = (\mathsf{Arg}, \mathsf{R}) : G \text{ produces } \ell \implies G \in \mathbb{F}$$

Proof of Theorem 5.1 (Soundness). Let ℓ be a conflict-free input labeling and Arg denotes the set of arguments. $C = \{C_a = AccCond_{cf}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of acceptance conditions computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C.

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

Assume there exists an argumentation framework $F = (\mathsf{Arg}, \mathsf{R}') \in \mathbb{F}$ which does not produce 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 acceptance conditions. We consider any argument $a \in \mathsf{in}(\ell)$. That means, there must be a model $\mathcal{A}$ of the acceptance condition C_a such that $\mathcal{A}(r_{ba}) = true$. However, according to Definition 5.3 the acceptance condition 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 acceptance condition C_a such that $\mathcal{A}(r_{ba}) = false$ for all arguments $b \in \mathsf{in}(\ell)$. However, according to Definition 5.3 the acceptance condition 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 acceptance condition C_a such that $\mathcal{A}(r_{ba}) = true$ for any arguments $b \in \mathsf{in}(\ell)$. However, according to Definition 5.3 the acceptance condition 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 5.1 (Completeness). Let ℓ be a conflict-free input labeling and Arg denotes the set of arguments. $C = \{C_a = AccCond_{cf}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of acceptance conditions computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C.

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 produces 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 2.10 the following three conditions hold for all arguments $a \in \mathsf{Arg}$ in F:

1. If $a \in \mathsf{in}(\ell)$, then $\forall b \in \mathsf{Arg} : (b, a) \in \mathsf{R}' \rightarrow b \notin \mathsf{in}(\ell)$
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 acceptance condition C_a for the in-labeled argument a is a conjunction and contains the literal $\neg r_{ba}$. This means that $\mathcal{A}(r_{ba}) = false$ for all models $\mathcal{A}$ of C_a. Similarly, the acceptance condition for b enforces that $\mathcal{A}(r_{ab}) = false$ for all models $\mathcal{A}$. Thus, since F satisfies the first condition from above, it must also satisfy the acceptance condition 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 acceptance condition $C_a = \bigvee\limits_{b \in \mathsf{in}(\ell)} r_{ba}$. It follows that F always satisfies the acceptance conditions 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 acceptance condition for undec-labeled argument in conflict-free labelings $C_a = \bigwedge\limits_{b \in \mathsf{in}(\ell)} \neg r_{ba}$. Thus every undec-labeled argument and its incoming attacks in F always satisfy all related acceptance conditions.

It follows that, if the argumentation framework F produces the conflict-free labeling ℓ it also satisfies the acceptance conditions for all arguments as defined in Definition 5.3. Thus, it holds for all conflict-free labelings that, if F produces ℓ, then $F \in \mathbb{F}$.

□

A.2. Admissible Semantics

Theorem 5.2. *Let ℓ be an admissible input labeling and $C = \{C_a = AccCond_{ad}(a, \ell) \mid a \in \mathsf{Arg}\}$ is the set of acceptance conditions for the input labeling ℓ. With $\mathbb{F}$ we denote the set of argumentation frameworks consistent with C.*

The acceptance conditions are sound *for admissible input labelings, i.e.*

$$\forall F \in \mathbb{F} : \ell \textit{ is admissible} \implies F \textit{ produces } \ell$$

The acceptance conditions are complete *for admissible input labelings, i.e.*

$$\forall G = (\mathsf{Arg}, \mathsf{R}) : G \textit{ produces } \ell \implies G \in \mathbb{F}$$

Proof of Theorem 5.2 (Soundness). Let ℓ be an admissible input labeling and Arg denotes the set of arguments. $C = \{C_a = AccCond_{ad}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of acceptance conditions computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C.

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 $F \in \mathbb{F}$ produces the labeling ℓ.

Assume there exists an argumentation framework $F = (\mathsf{Arg}, \mathsf{R}')\in \mathbb{F}$ which does not produce 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 $(b, a) \in \rightarrow'$ 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 acceptance condition C_a such that $\mathcal{A}(r_{ba}) = true$. However, according to Definition 5.4 the acceptance condition of a is defined as $C_a = \bigwedge_{b \in \mathsf{Arg} \setminus \mathsf{out}(\ell)} \neg r_{ba}$. Thus, it holds that $\mathcal{A}(r_{ba} = false$ 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 produce the input labeling ℓ and thus the algorithm is *sound* for admissible input labelings.

□

Proof of Theorem 5.2 (Completeness). Let ℓ be an admissible input labeling and Arg denotes the set of arguments. $C = \{C_a = AccCond_{ad}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of acceptance conditions computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C.

We proof by contradiction that the algorithm is complete for admissible labelings, i.e. for every argumentation framework F it holds that, if F produces 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 2.11 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 acceptance condition 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 $\mathcal{A}(r_{ba}) = false$ for all models $\mathcal{A}$ of C_a.

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

It follows that, if the argumentation framework F produces the admissible labeling ℓ it also satisfies the acceptance conditions for all arguments as defined in Definition 5.4. Thus, it holds for all admissible labelings that, if F produces ℓ, then $F \in \mathbb{F}$.

$\square$

A.3. Complete Semantics

Theorem 5.3. *Let ℓ be a complete input labeling and $C = \{C_a = AccCond_{co}(a, \ell) \mid a \in \mathsf{Arg}\}$ is the set of acceptance conditions for the input labeling ℓ. With $\mathbb{F}$ we denote the set of argumentation frameworks consistent with C.*

The acceptance conditions are sound *for complete input labelings, i.e.*

$$\forall F \in \mathbb{F} : \ell \textit{ is complete} \implies F \textit{ produces } \ell$$

The acceptance conditions are complete *for complete input labelings, i.e.*

$$\forall G = (\mathsf{Arg}, \mathsf{R}) : G \textit{ produces } \ell \implies G \in \mathbb{F}$$

Proof of Theorem 5.3 (Soundness). Let ℓ be a complete input labeling and Arg denotes the set of arguments. $C = \{C_a = AccCond_{co}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of acceptance conditions computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C.

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 acceptance condition 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 $\mathcal{A}(r_{ca}) = true$. However, from the acceptance condition C_c of c it would then follow that $\mathcal{A}(r_{dc}) = false$ 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 produce the input labeling ℓ, i.e. the algorithm is *sound* for complete input labelings.

□

Proof of Theorem 5.3 (Completeness). Let ℓ be a complete input labeling and Arg denotes the set of arguments. $C = \{C_a = AccCond_{co}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of acceptance conditions computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C.

We proof that the algorithm is complete for complete input labelings, i.e. for every argumentation framework F it holds that, if F produces 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 2.12 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 acceptance conditions. We now show that from the third and fourth condition it follows that F also satisfies the acceptance condition 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 acceptance condition $C_{a,1} = \bigwedge_{b \in \mathsf{in}(\ell)} \neg r_{ba}$ for any undec-labeled argument. This also equals the first part of the acceptance condition 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 acceptance condition for undec-labeled arguments in complete labelings as defined in Definition 5.5.

It follows that, if the argumentation framework F produces the complete labeling ℓ it also satisfies the acceptance conditions for all arguments. Thus, it holds for all complete labelings that, if F produces ℓ, then $F \in \mathbb{F}$.

□

A.4. Stable Semantics

Theorem 5.4. *Let ℓ be a stable input labeling and $C = \{C_a = AccCond_{st}(a, \ell) \mid a \in \mathsf{Arg}\}$ is the set of acceptance conditions for the input labeling ℓ. With $\mathbb{F}$ we denote the set of argumentation frameworks consistent with C.*

The acceptance conditions are sound *for stable input labelings, i.e.*

$$\forall F \in \mathbb{F} : \ell \textit{ is stable} \implies F \textit{ produces } \ell$$

The acceptance conditions are complete *for stable input labelings, i.e.*

$$\forall G = (\mathsf{Arg}, \mathsf{R}) : G \textit{ produces } \ell \implies G \in \mathbb{F}$$

Proof of Theorem 5.4 (Soundness). Let ℓ be a stable input labeling and Arg denotes the set of arguments. $C = \{C_a = AccCond_{st}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of acceptance conditions computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C.

We proof by contradiction that the acceptance conditions 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 does not produce 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 acceptance conditions in C. Every acceptance condition $C_a \in C$ is defined as $C_a = AccCond_{st}(a, \ell)$. Per Definition 5.6 the acceptance condition 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 $\mathcal{A}(r_{ba}) = true$ 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 acceptance condition C'_a it holds for every model $\mathcal{A}$ that there exists some argument $b \in \mathsf{in}(\ell)$ such that $\mathcal{A}(r_{ba}) = true$. 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 acceptance conditions C computed for a stable labeling ℓ. Thus, it follows that every $F \in \mathbb{F}$ must produce the input labeling ℓ and the algorithm is *sound* for stable input labelings.

$\square$

Proof of Theorem 5.4 (Completeness). Let ℓ be a stable input labeling and Arg denotes the set of arguments. $C = \{C_a = AccCond_{st}(a, \ell)\}_{a \in \mathsf{Arg}}$ is the set of acceptance conditions computed for ℓ and $\mathbb{F}$ is the set of argumentation frameworks consistent with C.

We proof that the algorithm is complete for stable input labelings ℓ, i.e. for every argumentation framework F it holds that, if F produces 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 2.15 the following two conditions hold for all arguments $a \in \mathsf{Arg}$ in F:

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

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 acceptance conditions 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 produces the stable labeling ℓ it also satisfies the acceptance conditions for all arguments as defined in Definition 5.6. Thus, it holds for all stable labelings that, if F produces ℓ, then $F \in \mathbb{F}$.

□

A.5. Monotonicity

Theorem 5.6. *Let* Arg *be a set of arguments and L_1, L_2 are sets of labelings. $\mathbb{F}_1$, $\mathbb{F}_2$ denote the sets of argumentation frameworks constructed by the algorithm defined in Section 5.4 for the input labelings L_1 and L_2 respectively.*

The algorithm for constructing argumentation frameworks from labelings is monotonous.

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

Proof of Theorem 5.6 (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 acceptance conditions 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 5.4 is monotonous. Consider any argument a. Per definition the acceptance condition for a with respect to L_2 is then $C_{a,2} = \bigwedge_{\ell_\sigma \in L_2} AccCond_\sigma(a, \ell)$. 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.

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

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

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

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

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

We can now split up this formula and write it as

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

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 acceptance condition 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 acceptance condition, i.e. $\mathcal{M}_{a,1} \subseteq \mathcal{M}_{a,2}$. Since the models of an acceptance condition 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 monotonous. □

B. Contents of the attached CD

- PDF-version of the masters thesis
- Evaluation data (see Section 7)
- Implementation of the algorithm (see Algorithm 4)

References

[1] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. „Abstract argumentation frameworks and their semantics“. In: *Handbook of formal argumentation* 1 (2018), pp. 157–234.

[2] Pietro Baroni and Massimiliano 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.

[3] Stefano Bistarelli et al. „A First Overview of ICCMA'19.“ In: *AI^3@ AI* IA*. 2020, pp. 90–102.

[4] Gerhard Brewka and Stefan Woltran. „Abstract dialectical frameworks“. In: *Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*. 2010.

[5] Gerhard Brewka et al. „Abstract dialectical frameworks revisited“. In: *Twenty-Third International Joint Conference on Artificial Intelligence*. 2013.

[6] Martin Caminada. „Semi-stable semantics“. In: *COMMA* 144 (2006), pp. 121–130.

[7] Martin WA Caminada and Dov M Gabbay. „A logical account of formal argumentation“. In: *Studia Logica* 93.2 (2009), pp. 109–145.

[8] Andrea Cohen et al. „A survey of different approaches to support in argumentation systems“. In: *The Knowledge Engineering Review* 29.5 (2014), p. 513.

[9] Phan Minh Dung. „On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games“. In: *Artificial intelligence* 77.2 (1995), pp. 321–357.

[10] Paul E Dunne et al. „Weighted argument systems: Basic definitions, algorithms, and complexity results“. In: *Artificial Intelligence* 175.2 (2011), pp. 457–486.

[11] Paul E Dunne et al. „Characteristics of multiple viewpoints in abstract argumentation“. In: *Artificial Intelligence* 228 (2015), pp. 153–178.

[12] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

[13] Hadassa Jakobovits and Dirk Vermeir. „Robust semantics for argumentation frameworks“. In: *Journal of logic and computation*. Vol. 9. OUP, 1999, pp. 215–261.

[14] Hiroyuki Kido and Beishui Liao. „A Bayesian approach to direct and inverse abstract argumentation problems“. In: *arXiv preprint arXiv:1909.04319* (2019).

[15] Isabelle Kuhlmann. „Towards Eliciting Attacks in Abstract Argumentation Frameworks“. In: *Online Handbook of Argumentation for AI* (2021), p. 27.

[16] Isabelle Kuhlmann et al. „Distinguishability in Abstract Argumentation“. In: (2021).

[17] Andreas Niskanen, Johannes Wallner, and Matti Järvisalo. „Synthesizing argumentation frameworks from examples“. In: *Journal of Artificial Intelligence Research* 66 (2019), pp. 503–554.

[18] Emilia Oikarinen and Stefan Woltran. „Characterizing strong equivalence for argumentation frameworks“. In: *Artificial intelligence* 175.14-15 (2011), pp. 1985–2009.

[19] John L Pollock. *Cognitive carpentry: A blueprint for how to build a person*. Mit Press, 1995.

[20] Régis Riveret and Guido Governatori. „On learning attacks in probabilistic abstract argumentation“. In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. 2016, pp. 653–661.

[21] Matthias 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)*. July 2014.