Labellings and their algorithms for Abstract Argumentation Framework Semantics

D.J.S. Diehl *
Bachelor Thesis (7.5 EC)
Artificial Intelligence
Student number: 6779271
Supervisor: W. van Woerkom
Second reader: dr. R.W.F. Nouwen

1st July 2022

Abstract

This thesis provides an overview of the most notable semantics for Dung's argumentation frameworks. In addition to this, we will combine the works of (Modgil and Caminada, 2009), (Caminada, 2007a) and (Caminada, 2010). to provide labelling algorithms for each semantic. The main research goal of this thesis was to write concrete implementations of these algorithms. We start with the original semantics introduced by (Dung, 1995). These are the grounded, preferred and stable semantics. Then we continue with the semi-stable semantic of (Caminada, 2006), the stage semantic of (Verheij, 1996), and end with the ideal semantic of (Dung et al., 2006) and the eager semantic of (Caminada, 2007a). We will extend the research of Caminada et al. by providing a procedure for sceptical judgement aggregation, which is used for the ideal and eager labelling algorithms.

Keywords: Argumentation frameworks, Non-monotonic reasoning, Labelling algorithms, Extensions, Judgement aggregation, Grounded, Preferred, Stable, Semi-stable, Stage, Ideal, Eager

*Special Thanks to W. van Woerkom for supervising this thesis and for providing feedback on earlier versions.

Contents

1 Introduction

A fascinating way in which humans reason is by making defeasible inferences. This means that we can make inferences that can be retracted later when more information is acquired. For example, suppose there is a bird named Tweety. On the basis that Tweety is a bird and that we know that birds usually fly, we infer that Tweety can fly. However, we have good reason to retract this statement later when we learn that Tweety is a penguin.

Defeasible reasoning is something that we do a lot in our everyday lives, which we could demonstrate with a few examples of (Strasser and Antonelli, 2001). Imagine we see the streets are wet, from this we may infer that it has been raining recently. However, when we later see that the rooftops are dry and recall that this day the streets are usually cleaned, we will retract this inference. This type of reasoning is called abductive reasoning, which could be understood as inferring to the best explanation and is one example of defeasible reasoning. Another example would be when we assume something is true (or false) because it has not been stated. When making a puzzle about crossing a river, we automatically assume that there are no bridges or other means of transportation available than stated (McCarthy, 1981; McCarthy and Hayes, 1981). Similarly, when we store data in databases, not stating a flight in a flight schedule, simply means there is no flight at that time. We also find defeasible inferences in scientific reasoning, where we create hypotheses based on current evidence, which could later be retracted or altered based on newly acquired evidence (Strasser and Antonelli, 2001).

A vast area in AI research is trying to bridge the gap between human reasoning and automated inference. Inference plays an important part in arriving at justifiable conclusions. Many logics have been defined that aim to capture human deductive reasoning. Like deductive reasoning, defeasible reasoning can follow complex patterns. Unfortunately, those patterns are beyond reach for the logics we use for deductive reasoning, like classical logic and intuitionistic logic, since by their very nature they do not allow for a retraction of inferences. This is caused by the fact that these logics are monotonic, which states that consequences are robust under the addition of information. To capture and represent defeasible inferences a family of formal frameworks was devised, collectively called non-monotonic logic (NML). NML's, like the name suggests, violate the monotony property (Strasser and Antonelli, 2001).

A notable example of those NML's are argumentation frameworks, introduced in (Dung, 1995). An argumentation framework is an abstract representation of an argument, it consists of a set of arguments and an attack relation upon them, which means that arguments can attack each other, just like in a normal argumentation. To determine the acceptability of a group of arguments Dung introduces semantics and extensions. A semantic is a collection of criteria and an extension is a set of the arguments that meet the requirements posed by the semantic. Such an extension is a subset of the original set of arguments. To find extensions, the labelling approach is often used, originally proposed in (Pollock, 1995). Simply put, a labelling is an elaboration of an extension, which not only points out which arguments are acceptable, but also which are unacceptable or uncertain.

Concerning the semantics of argumentation frameworks questions could be posed, bringing light to the characteristics of each semantic. (Modgil and Caminada, 2009) pose the following questions that naturally arise from Dung's definition of argumentation frameworks and their semantics.

1. Global questions
    (a) Does an extension exist?
    (b) Give an extension (it does not matter which, just give one).
    (c) Give all extensions.
2. Local questions
    (a) Is A contained in an extension? (Credulous membership question).
    (b) Is A contained in all extensions? (Sceptical membership question).
    (c) Is A attacked by an extension?
    (d) Is A attacked by all extensions?
    (e) Give an extension containing A.
    (f) Give all extensions containing A.
    (g) Give an extension that attacks A.
    (h) Give all extensions that attack A.

In the global questions, we look from the perspective of the entire argumentation framework, while in local questions we look from the perspective of a single argument. Modgil and Caminada answer the global and local questions by describing the procedures and giving pseudocode for algorithms that find the labellings for several semantics and prove that their procedures are correct using argument games and thus answer their respective question. They do this for the grounded, preferred, stable and semi-stable semantics. (Caminada, 2010) also provides this for the stage semantics, and (Caminada and Pigozzi, 2011) describe a procedure to find the ideal and eager labellings.

The research goals of this thesis are as follows. We will start with an overview of argumentation frameworks and their most notable semantics. For each semantic, I will explain the procedures that find their respective labelling(s). The semantics that will be discussed are the following. The grounded, preferred, stable, semi-stable semantics as described in (Modgil and Caminada, 2009), the stage semantics from (Caminada, 2010), the ideal semantics from (Dung et al., 2006), and the eager semantics from (Caminada, 2007b). We will then build further upon the work of (Modgil and Caminada, 2009), and provide procedures that compute labellings for each of these semantics. We will use judgement aggregation for the procedures of the ideal and eager labellings, as described in (Caminada and Pigozzi, 2011). These procedures currently have no pseudocode provided by Caminada et al., therefore I will expand on their previous research and provide pseudocode at the end of those sections. The proof of correctness for all the procedures is included in the papers above. The final step of this thesis is to use this information to write concrete implementations of these procedures, which is submitted along with this thesis. The concrete implementation of my algorithms will be written in the programming language C#. The global question that each implementation should answer is as follows.

Give all labellings that satisfy semantic S.

The rest of this thesis will be structured as follows. We begin in Section 2 with an overview of Dung's argumentation framework and introduce the necessary theory for the semantics, extensions and labellings. Then in Section 3, we will look at each semantics individually. For each semantic, we start by introducing the definition, then look at how this relates to their respective extension and labelling and follow this up by explaining the implementation for the labelling algorithm, accompanied by pseudocode. Finally in section 4, we will end with a reflection on all the semantics and implemented algorithms.

2 Abstract argumentation frameworks

Before we are able to implement and understand the algorithms that compute labellings in argumentation frameworks we need to have an understanding of the underlying theory and essential definitions. We will start by looking at Dung's argumentation frameworks and their semantics.

The main idea of formal argumentation is that non-monotonic reasoning can be achieved by constructing and evaluating arguments. An argument expresses one or more reasons that lead to a proposition, which allows them to be defeasible, namely in the way that the validity of their conclusions can be disputed by other arguments. Whether a claim can be accepted or rejected, not only depends on the existence of arguments that support this claim but also depends on the existence of possible counterarguments, which themselves can also be attacked by other counterarguments and so on (Baroni et al., 2011).

Nowadays, the abstract argumentation theory of (Dung, 1995) holds a significant role in formal argumentation. The central concept in this theory are argumentation frameworks, which can be regarded as a directed graph where nodes represent arguments and arrows represent an attack relation on the arguments. In contrast to other approaches in formal argumentation, Dung's Argumentation frameworks only focusses on the topology of the arguments. Meaning his approach does not distinguish between the various ways in which an argument can attack another. Such as undermining, undercutting or rebutting (Koons, 2005). However, argumentation frameworks can be extended to have several different attack relations. This is for example the case in ASPIC+ (Prakken, 2010), which extends Dung's argumentation frameworks and makes use of all three forms of attack.

(Dung, 1995) defines an argumentation framework as follows.

Definition 1 *An* argumentation framework *is a pair $AF = (AR, attacks)$. Where AR is a set of arguments and attacks is a binary relation on AR, i.e., $attacks \subseteq AR \times AR$.*

For two arguments $a, b \in AR$, $attacks(a, b)$ means that argument a is attacking argument b, and that argument b is attacked by a. Before continuing lets look at a few examples of argumentation frameworks.

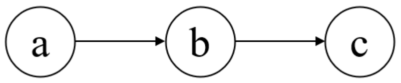

Figure 1: Example of an argumentation framework. Its set of arguments AR consists of three arguments *a*, *b*, and *c*. In this AF argument *a* attacks *b* and argument *b* attacks *c*.

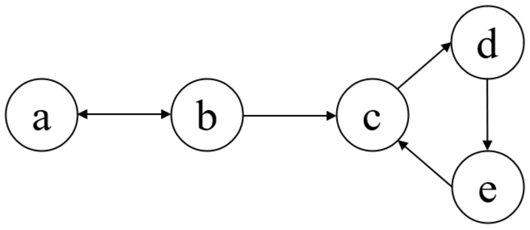

Figure 2: An Argumentation framework with five arguments. Arguments *a* and *b* mutually attack each other and argument *b* attacks *c*. The arguments *c*, *d* and *e* form a cycle of attacks, where *c* attacks *d*, *d* attacks *e* and *e* attacks *c* (Caminada, 2007b).

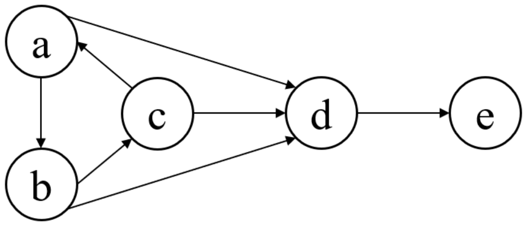

Figure 3: Another example of an argumentation framework, with five arguments. Here argument *a*, *b* and *c* form a cycle. Arguments *a*, *b* and *c* attack *d*, and argument *d* attacks *e* (Caminada, 2010).

2.1 Semantics for argumentation frameworks

Now we know what argumentation frameworks are, a simple question arises. How do we determine which arguments we should accept and which arguments we should reject? To determine the acceptability of a group of arguments Dung introduces semantics and extensions. A semantic is a collection of criteria and an extension is a set of the arguments that meet the requirements posed by the semantic. Such an extension is a subset of the original set of arguments (Elaroussi et al., 2022).

To determine the acceptability of arguments in the entire framework, we first need to consider the acceptability of arguments relative to other arguments. To be able to do this (Dung, 1995) introduces the notion of acceptable arguments relative to a set of arguments.

Definition 2 *An argument $a \in AR$ is* acceptable *with respect to a set S of arguments, (also called defended by S) iff for each argument $b \in AR$: if b attacks a, then b is attacked by S.*

A way to look at this is that an argument is acceptable to a set of arguments S if it is defended by this set S itself, i.e. for every attacker of this argument there is an argument in S that attacks this attacker. In addition to this, it also makes sense that the arguments we accept should not attack each other. This property is captured by Dung's principle of conflict-freeness.

Definition 3 *A set S of arguments is said to be* conflict-free *if there are no arguments a and b in S such that a attacks b.*

Central to Dung's approach is the idea of an admissible set of arguments. This is achieved by combining Definitions 2 and 3.

Definition 4 *A conflict-free set of arguments S is* admissible *iff each argument in S is acceptable with respect to S.*

An admissible set of arguments can be regarded as a valid set of arguments, where each argument is defended by the set itself. There are two main approaches for the semantics of argumentation frameworks described in the literature: the extension-based approach and the labelling-approach.

2.1.1 Labelling-based approach

In this section, we give a brief overview of the labelling-based approach, initially introduced in (Pollock, 1995). A labelling assigns each argument exactly one label. Most commonly used are the labels: IN, OUT and $UNDEC$. Here the label IN means that the argument is accepted (meaning it is justified), the label OUT means the argument is rejected because it is overruled by another argument, and the label $UNDEC$ means that we abstain from forming an opinion on whether the argument is accepted or rejected. (Modgil and Caminada, 2009) define a labelling as stated below.

Definition 5 *Let $AF = (A, R)$ be an argumentation framework.*

- *A* labelling *is a total function $L : A \rightarrow \{IN, OUT, UNDEC\}$*
- *We define:*
  - $in(L) = \{x \mid L(x) = IN\}$;
  - $out(L) = \{x \mid L(x) = OUT\}$;
  - $undec(L) = \{x \mid L(x) = UNDEC\}$;

From here on we will represent a labelling as a triple ($in(\mathcal{L})$, $out(\mathcal{L})$, $undec(\mathcal{L})$). In Figure 1, a reasonable labelling would be to assign argument a the label IN, as it is not attacked by any argument. Then we would assign argument b the label OUT, as it is attacked by argument a which is accepted. Then we can assign argument c the label IN, as it is only attacked by b which is rejected. The resulting labelling would look as follows

$\mathcal{L} = (\{a, c\}, \{b\}, \emptyset)$. Another labelling would be to assign all arguments the label IN, this labelling seems less reasonable however, as the accepted set has arguments attacking other accepted arguments, meaning it is not conflict-free. On the other hand, we could assign all arguments the label $UNDEC$, which seems excessively cautious as argument a is unattacked. To be able to select reasonable labellings, out of all possible labellings, we use labelling-based argumentation semantics (Baroni et al., 2011).

We will now define what it means for an argument to be assigned a legal labelling. This definition for legal labellings is meant for admissible labellings, which can be used for most semantics. By contrast, the definition for stage semantics (3.5) is built upon conflict-freeness and not admissibility. There we will define the definition for legal labellings for conflict-free semantics. (Modgil and Caminada, 2009) handle the following definition for legal labellings in admissibility-based semantics.

Definition 6 *Let $\mathcal{L}$ be a labelling for $AF = (A, R)$ and $x \in A$, then:*

1. *x is* legally IN *iff x is labelled IN and every y that attacks x (yRx) is labelled OUT;*
2. *x is* legally OUT *iff x is labelled OUT and there is at least one y that attacks x and y is labelled IN;*
3. *x is* legally UNDEC *iff x is labelled $UNDEC$, there is no y that attacks x such that y is labelled IN, and it is not the case that: for all y, y attacks x implies y is labelled OUT.*

The rules above defining legal labellings, aim to capture one's intuitive understanding of when arguments should be accepted or rejected. Here it is the case that an argument is IN, only if all its attackers are OUT. An argument is OUT if it has at least one attacker that is IN. And an argument is $UNDEC$ if it cannot be IN or OUT. Since arguments can be legally labelled, they can also be illegally labelled (Modgil and Caminada, 2009).

Definition 7 *For $l \in \{IN, OUT, UNDEC\}$ an argument x is said to be* illegally *l iff x is labelled l, and it is not legally l.*

1. *An* admissible labelling *$\mathcal{L}$ is a labelling without arguments that are illegally IN and without arguments that are illegally OUT.*
2. *A* complete labelling *$\mathcal{L}$ is an admissible labelling without arguments that are illegally $UNDEC$.*

Observe that a labelling where all arguments are labelled $UNDEC$ is admissible according to this definition. Admissible labellings require good reasons before an argument can be labelled IN or OUT, but always leave room to abstain from an argument. Complete labellings however, are stricter as to when an argument should be $UNDEC$. Complete labellings are forced to label arguments IN or OUT when they can be and only allow $UNDEC$ arguments when there are insufficient grounds for accepting or rejecting an argument (Baroni et al., 2011).

2.1.2 Extension-based approach

In the extension-based approach, our aim is to identify sets of arguments, called extensions, which can survive conflict from outside attackers altogether. Collectively they embody a reasonable position of an autonomous reasoner. As we will focus on labellings in this paper, we can regard an extension simply as the IN-set of a labelling. For a proof that there is a one-to-one correspondence between labellings and extensions see (Caminada, 2007a; Caminada, 2011; Caminada and Pigozzi, 2011). Remember however, that just like labellings there exists specific extension-based argumentation semantics, which allows us to select "reasonable" sets of arguments among all possible ones (Baroni et al., 2011).

For the sake of completeness, I will still provide the definition of the extension for every semantics. Subsequently follow a few more definitions, used and necessary for the extension-based approach. Extension-based argumentation semantics often use the terms A^+, A^-, to denote sets of arguments attacked or defended by a single argument, and use $Args^+$ and $Args^-$ which do the same for sets. These are useful as the acceptability of arguments depends on the interaction with other arguments, and we have no OUT- or $UNDEC$-set to place restrictions on. The following terms are defined as in (Baroni et al., 2011).

Definition 8 *Let $AF = (Ar, att)$, for $A \in Ar$ and $Args \subseteq Ar$, we write:*

1. A^- *for* $\{B \mid (B, A) \in att\}$;
2. A^+ *for* $\{B \mid (A, B) \in att\}$;
3. $Args^-$ *for* $\bigcup_{A \in Args} A^-$;
4. $Args^+$ *for* $\bigcup_{A \in Args} A^+$.

Definitions of extensions often use the expressions maximal and minimal with respect to set inclusiveness. A *maximal* set w.r.t set inclusion among a collection of sets, is a set that is not a subset of some other set in this collection. A *minimal* set w.r.t set inclusion among a collection of sets, is a set in the collection that is not a superset of any other set in the collection. In addition to maximal, a set can also be the greatest set. A set is the *greatest* w.r.t set inclusion if it is bigger than everything it can be compared to.

Each labelling can be turned into an extension, which means we have admissible extensions which are just admissible sets. The rules for admissible labellings and extensions together are part of admissible semantics, which is the foundation for a lot of the semantics we will examine next.

3 Semantics and their labelling algorithms

Now that we have had all the preliminary theory we can look at some of the most notable semantics of argumentation frameworks. This section will give an overview of the grounded, preferred, stable, semi-stable, stage, ideal and eager semantics.

3.1 Grounded semantics

The first semantic we will examine is the grounded semantic, introduced in (Dung, 1995). The basic idea is to only accept the arguments that one cannot avoid to accept and reject the arguments that one cannot avoid to reject, and refrain from passing any other judgement upon the rest of the arguments. This principle is also called taking the most "grounded" position, hence the name. By refraining as much as possible we get the most sceptical or least committed semantics based on complete extensions (Baroni et al., 2011). For any argumentation framework, there exists exactly one grounded extension.

Grounded semantics can be characterized by a fixed point of the characterization function of (Dung, 1995), which is defined as follows.

Definition 9 *Let $AF = (Ar, att)$ be an argumentation framework and $Args \subseteq Ar$. The* characteristic function *of AF, denoted by F_{AF} is defined as follows: $F : 2^{Ar} \rightarrow 2^{Ar}$, $F_{AF}(S) = \{A \mid A$ is acceptable with respect to $S\}$.*

So, the characteristic function looks for all arguments that are acceptable (or defended) by set S. Using the characteristic function, we can formulate the definition for the complete extension of Dung, which is as follows.

Definition 10 *A conflict-free set of arguments E is a* complete extension *iff $E = F_{AF}(E)$.*

The grounded extension is then simply the minimal (w.r.t. to set inclusion) complete extension. This means it is the least fixed point of the characterization function. Thus, we get the following definition for the grounded extension (Dung, 1995).

Definition 11 *The* grounded extension *of an argumentation framework AF, denoted by GE_{AF} is the least fixed point of F_{AF}.*

This definition signifies that we can build the grounded extension from the ground up, starting with an empty set, and iteratively apply the characterization function on this set, until the result set of the characteristic function is the same as the input.

We can take this definition and easily turn it into a definition for a labelling. As the extension is based on a complete extension, the labelling should be based on a complete labelling. Since we only want to accept arguments that we can in no way avoid accepting, we only need to minimize the IN-set of the labelling (which is the extension). Because the labelling must be complete, placing the restriction of minimizing the IN-set has the result that the OUT-set is minimal as well, as an argument can only be OUT if it is attacked by an argument that is IN. This results in the following definition for the grounded labelling (Modgil and Caminada, 2009).

Definition 12 *A labelling $\mathcal{L}$ is a* grounded labelling *iff there does not exist a complete labelling $\mathcal{L}'$ such that $in(\mathcal{L}') \subset in(\mathcal{L})$.*

Note that we still get the grounded extension if we minimize the OUT-set of the labelling instead of the IN-set, or if we maximize the $UNDEC$-set.

3.1.1 Procedure for the grounded labelling

Because the grounded semantics is defined as the least fixed point, the procedure to find the grounded labelling is fairly simple. We just keep applying the characterization function to find arguments to label IN and then update the OUT-set accordingly.

In the first iteration, we start by looking for arguments that are not attacked, as there are no arguments labelled OUT. We label all these unattacked arguments IN. Then we can automatically label all arguments attacked by an argument we just labelled IN OUT, as they are now attacked by an argument in S and therefore no longer acceptable in the grounded extension.

Then we go to the next iteration, where we repeat this process. We now look for all arguments that are only attacked by arguments that are labelled OUT and label those arguments IN. Then with the newly IN-labelled arguments, we look for new arguments that are attacked by those IN-labelled arguments and again label those OUT. We repeat this process until executing an iteration yields the same result as before the iteration. We complete the labelling by giving all unlabelled arguments the labelling $UNDEC$. This procedure is captured in the following algorithm.

Algorithm 1 Grounded Labelling (Modgil and Caminada, 2009)

```
1:  function FindGroundedLabelling
2:      $\mathcal{L}_0 \leftarrow (\emptyset, \emptyset, \emptyset)$
3:
4:      repeat
5:          $in(\mathcal{L}_{i+1}) \leftarrow in(\mathcal{L}_i) \cup \{x \mid x$ is not labelled in $\mathcal{L}_i$, and $\forall y :$ if $yRx$ then $y \in out(\mathcal{L}_i)\}$
6:          $out(\mathcal{L}_{i+1}) \leftarrow out(\mathcal{L}_i) \cup \{x \mid x$ is not labelled in $\mathcal{L}_i$, and $\exists y : yRx$ and $y \in in(\mathcal{L}_{i+1})\}$
7:      until $\mathcal{L}_i \neq \mathcal{L}_{i+1}$
8:
9:      $\mathcal{L}_g \leftarrow (in(\mathcal{L}_i), out(\mathcal{L}_i), (A) - (in(\mathcal{L}_i) \cup out(\mathcal{L}_i))$
10:     return $\mathcal{L}_g$
```

If we apply the algorithm above to the examples in Figures 1, 2 and 3, we get the following grounded labellings.

- Figure 1: $\mathcal{L}_G = (\{a, c\}, \{b\}, \emptyset)$. Argument a is not under attack meaning we can label it IN, this makes b OUT. Then in the second iteration c gets labelled IN, because of b.
- Figure 2: $\mathcal{L}_G = (\emptyset, \emptyset, \{a, b\})$. The algorithm ends after one iteration as there are no arguments that are not under attack.
- Figure 3: $\mathcal{L}_G = (\emptyset, \emptyset, \{a, b, c, d, e\})$. Again, there are no unattacked arguments.

3.2 Preferred semantics

While the standpoint of the grounded semantics is sceptical, we could also consider the alternative view of accepting as many arguments as possible, we call this a credulous point of view. For example, when two arguments have a mutual attack relation between them, like Figure 2, looking from a sceptical point of view, we would decide we can make both arguments neither IN nor OUT. Hence the grounded labelling would have both arguments labelled $UNDEC$. But we could consider accepting one of the mutually exclusive alternatives, this results in a labelling where we label one of the arguments IN and the other OUT. This idea of maximizing the arguments that are accepted is embodied by the preferred semantics (Baroni et al., 2011). The preferred extension of (Dung, 1995) is defined as follows.

Definition 13 *A* preferred extension *of an argumentation framework AF is a maximal (with respect to set inclusion) admissible set of AF.*

Every argumentation framework has at least one preferred extension, and there can be more than one. (Modgil and Caminada, 2009) define its corresponding labelling in the following manner.

Definition 14 *$\mathcal{L}$ is a* preferred labelling *iff $\mathcal{L}$ is an admissible labelling such that for no admissible labelling $\mathcal{L}'$ it is the case that $in(\mathcal{L}') \supset in(\mathcal{L})$.*

When compared with the definition of the grounded semantics, we now consider admissible labellings, rather than complete, and change the restriction to maximizing the IN-set, instead of minimizing it. We also get preferred labellings when we maximize the OUT-set instead of the IN-set. Because maximal admissible sets (w.r.t. set inclusion) are equivalent to maximal complete extensions and as the grounded extension is in any complete extension, we get that the grounded extension is contained in every preferred extension (Baroni et al., 2011).

To find all the preferred labellings we need to traverse a search tree of labellings. To find all admissible labellings efficiently, we can use the definitions of Legally IN and OUT to define a transition step (Modgil and Caminada, 2009).

Definition 15 *Let $\mathcal{L}$ be a labelling for $Af = (A, R)$ and x an argument that is illegally IN in $\mathcal{L}$. A* transition step *on x in $\mathcal{L}$ consists of the following:*

1. *the label of x is changed from IN to OUT*
2. *for every $y \in x \cup \{z \mid xRz\}$, if y is illegally OUT, then the label of y is changed from OUT to UNDEC (i.e., any argument made illegally OUT by 1 is changed to UNDEC in 2)*

We can turn this definition into a function transition step, which takes as input $\mathcal{L}$ and x and applies the operations above to return a labelling $\mathcal{L}'$. We can then iterate the function over a labelling to form a transition sequence, which (Modgil and Caminada, 2009) define as follows.

Definition 16 *A* transition sequence *is a list* $[\mathcal{L}_0, x_1, \mathcal{L}_1, x_2, ..., x_n, \mathcal{L}_n]$ *with* $(n \geq 0)$*, where for* $i = 1...n$*,* x_i *is illegally IN in* L_{i-1}*, and* $L_i = \textit{transition step}(L_{i-1}, x_i)$*.*

- *A transition sequence is said to be* terminated *iff* L_n *does not contain any argument that is illegally IN.*
- *For any terminated transition sequence* $[\mathcal{L}_0, x_1, \mathcal{L}_1, x_2, ..., x_n, \mathcal{L}_n]$*, it holds that* $\mathcal{L}_n$ *is an admissible labelling.*
- *For any preferred labelling L, it holds that there exists a terminated transition sequence* $[\mathcal{L}_0, x_1, \mathcal{L}_1, x_2, ..., x_n, \mathcal{L}_n]$*, where* $\mathcal{L}_n = \mathcal{L}$*.*

When we combine the properties above with the definition for preferred labellings, we get that the terminated transition sequences whose final labellings maximize the arguments labelled *IN* are exactly the preferred labellings. If we only want to find complete labellings we can guide the choice of arguments on which we apply the transition step. We do this by only choosing an argument that is super-illegally *IN*, when it is available (Modgil and Caminada, 2009).

Definition 17 *An argument x in* $\mathcal{L}$ *that is illegally IN, is also* super-illegally *IN iff it is attacked by a y that is legally IN in* $\mathcal{L}$*, or UNDEC in* $\mathcal{L}$*.*

(Caminada, 2007a) shows that guiding the choice of arguments in this manner still results in admissible labellings at the end of every transition sequence and that every preferred labelling has a terminated transition sequence. Meaning we can use super-illegal arguments to more efficiently traverse the search tree of admissible labellings.

3.2.1 Procedure for preferred labellings

From now on all procedures will be called in the following manner, unless specified otherwise. We start by setting the candidate labellings to the empty set and call the labelling procedure of the semantic we want, supplying it the all-in labelling, where all arguments are assigned the label *IN*. The procedure then alters the candidate labellings and when finished it will contain all the labellings of the respective semantic.

Algorithm 2 Procedure call (Modgil and Caminada, 2009)

```
1: function FIND_LABELLINGS
2:     candidate_Labellings ← $\emptyset$
3:     CALL_LABELLING_PROCEDURE(all-in)
4:     return candidate_Labellings
```

We can use the transition step to traverse the search tree of admissible labellings in search of preferred labellings. To find all preferred labellings we have the following algorithm.

Algorithm 3 Preferred Labellings (Modgil & Caminada, 2009)

```
1: function FIND_PREFERRED_LABELLINGS($\mathcal{L}$)
2:     if $\exists\mathcal{L}' \in$ candidate_Labellings : $in(\mathcal{L}) \subset in(\mathcal{L}')$ then return
3:
4:     if $\mathcal{L}$ does not contain an argument that is illegally IN then
5:         for each $\mathcal{L}' \in$ candidate_Labellings do
6:             if $in(\mathcal{L}') \subset in(\mathcal{L})$ then
7:                 candidate_Labellings $\leftarrow$ candidate_Labellings $- \{\mathcal{L}'\}$
8:         candidate_Labellings $\leftarrow$ candidate_Labellings $\cup \{\mathcal{L}\}$
9:         return
10:    else
11:        if $\mathcal{L}$ has an argument that is super-illegally IN then
12:            $x \leftarrow$ some argument $\in \mathcal{L}$ that is super-illegally IN
13:            FIND_PREFERRED_LABELLINGS(TRANSITION_STEP($\mathcal{L}, x$))
14:        else
15:            for each $x \in \mathcal{L}$, that is illegally IN do
16:                FIND_PREFERRED_LABELLINGS(TRANSITION_STEP($\mathcal{L}, x$))
```

At every node, we enter the procedure FIND_PREFERRED_LABELLINGS with the labelling $\mathcal{L}$ that we are currently at. In line 2 we check if there already exists a labelling $\mathcal{L}'$ better than $\mathcal{L}$. In this case, we prune the search tree and backtrack to select another argument in the search tree, with the statement return.

In line 4 we check if the transition sequence has ended by checking if there are no arguments that are illegally IN. When that is the case we remove all candidates that become obsolete by this labelling, as their *in*-set is no longer maximized (lines 5-7). We then add this argument to the candidate labellings in line 8. Note that this argument can never be worse than any existing candidate due to line 2. If the transition sequence has not terminated yet, we apply all possible transition steps in lines 14-16. But if there exists an argument that is super-illegally IN, then we only need to check this branch, as this still yields all admissible labellings (lines 11-13).

If we apply the algorithm above to the examples in Figures 1, 2 and 3, we get the following preferred labellings.

- Figure 1: $\mathcal{L}_{pref} = (\{a, c\}, \{b\}, \emptyset)$. The same as the grounded labelling.
- Figure 2: $\mathcal{L}_{pref1} = (\{b, d\}, \{a, c, e\}, \emptyset)$, $\mathcal{L}_{pref2} = (\{a\}, \{b\}, \{c, d, e\})$. Here we have two options as there is a mutual attack between *a* and *b*. When we accept argument *b*, we can rule out arguments *a*, *c* and *e*, which leaves argument *d* which we accept. When we accept argument *a*, we can rule out *b*. This leaves the cycle which we entirely label undecided, as we cannot accept *c*, because this would make d OUT, which then makes e IN, but doing this would make c OUT. For the same reasons, we cannot accept *d* and *e*.

- Figure 3: $\mathcal{L}_{pref} = (\emptyset, \emptyset, \{a, b, c, d, e\})$. This time we have an empty preferred extension. This is caused by the fact that the only admissible labelling for AF is totally $UNDEC$.

3.3 Stable semantics

In the semantics discussed so far, there is the option to abstain from accepting or rejecting an argument by assigning them the label undecided. However, it is possible to prefer a more committed labelling, with no room for indecisiveness or neutrality. This is the case for the stable semantic, where there are no undecided labelled arguments. This results in an interesting property of the stable semantics, namely that it is xenophobic. This means that every argument outside of the stable extension is attacked by an argument in the stable extension (Baroni et al., 2011). The stable extension of (Dung, 1995) is defined below.

Definition 18 *A conflict-free set of arguments S is called a* stable extension *iff S attacks each argument which does not belong to S.*

(Modgil and Caminada, 2009) define stable labellings in the following manner.

Definition 19 *A complete labelling $\mathcal{L}$ is a* stable labelling *iff $undec(\mathcal{L}) \neq \emptyset$.*

At first glance, these definitions seem quite different, but because every argument in the stable extension attacks everything outside it, we get a labelling where all these attacked arguments are labelled OUT. it is then easy to see that this labelling and extension is admissible. They are also complete as the $UNDEC$-set of the labelling is empty. Note that Definition 19 is stated in terms of complete labellings, which automatically means that every stable labelling is also an admissible, conflict-free and preferred labelling. The existence of a stable extension (and labelling) is not guaranteed, as it is possible that all complete extensions contain undecided arguments (Modgil and Caminada, 2009), see Figure 3.

3.3.1 Procedure for stable labellings

Because all stable labellings are also preferred labellings, we can easily adapt the previous algorithm from Section 3.2.1 to only yield labellings where no arguments are labelled $UNDEC$. We do this by altering line 2. In addition to this, we no longer need to compare the IN-set with other labellings, as there is no restriction $in(\mathcal{L}') \subseteq in(\mathcal{L})$. Meaning we can remove lines 5-7, which results in the following algorithm (alterations marked in yellow).

Algorithm 4 Stable Labellings (Modgil and Caminada, 2009)

```
1:  function FIND_STABLE_LABELLINGS($\mathcal{L}$)
2:      if $undec(\mathcal{L}) \neq \emptyset$ then return
3:
4:      if $\mathcal{L}$ does not contain an argument that is illegally IN then
5:          candidate_Labellings ← candidate_Labellings ∪ {$\mathcal{L}$}
6:          return
7:      else
8:          if $\mathcal{L}$ has an argument that is super-illegally IN then
9:              $x$ ← some argument ∈ $\mathcal{L}$ that is super-illegally IN
10:             FIND_STABLE_LABELLINGS(TRANSITION_STEP($\mathcal{L}, x$))
11:         else
12:             for each $x \in \mathcal{L}$, that is illegally IN do
13:                 FIND_STABLE_LABELLINGS(TRANSITION_STEP($\mathcal{L}, x$))
```

If we apply the algorithm above to the examples in Figures 1, 2 and 3, we get the following stable labellings. As every stable labelling is a preferred labelling, all labellings we find here are also found in 3.2.1.

- Figure 1: $\mathcal{L}_{stable} = (\{a, c\}, \{b\}, \emptyset)$. As the $UNDEC$-set was empty of this preferred labelling, we get the same stable labelling.
- Figure 2: $\mathcal{L}_{stable} = (\{b, d\}, \{a, c, e\}, \emptyset)$. In this case, one of the preferred labellings gets eliminated, as the cycle was labelled undecided. Leaving only one stable extension.
- Figure 3: As the only preferred labelling was totally $UNDEC$ there are no stable labellings.

3.4 Semi-stable semantics

As illustrated in the previous section of stable semantics, the restriction of not allowing undecided arguments causes no labellings in some cases. However, in those cases, we still would like to have a way to express some form of opinion on the arguments. A sophisticated idea to achieve this is by passing judgement on the largest possible set of arguments and leaving as few arguments as possible undecided (Baroni et al., 2011). Semi-stable semantics do this, proposed in (Caminada, 2006). The definition of (Caminada, Verheij et al., 2010) for the semi-stable extension is as follows.

Definition 20 *$Args$ is a* semi-stable extension *iff $Args$ is an admissible set where $Args \cup Args^+$ is maximal (w.r.t.set-inclusion) among all admissible sets.*

Semi-stable semantics can be placed between preferred and stable semantics, when we take into account that every stable extension is also a semi-stable extension, and every semi-stable extension is also a preferred extension (Caminada, 2007a). When put into a labelling, we get the following definition from (Modgil and Caminada, 2009).

Definition 21 *$\mathcal{L}$ is a* semi-stable labelling *iff $\mathcal{L}$ is an admissible labelling such that for no admissible labelling $\mathcal{L}$' is it the case that undec($\mathcal{L}'$) $\subset$ undec($\mathcal{L}$).*

3.4.1 Procedure for semi-stable labellings

As all semi-stable labellings are also preferred labellings, we can take the procedure of Section 3.2.1 and make some adjustments. We only need to alter lines 2 and 5 to minimize the $UNDEC$-set, instead of maximizing the IN-set. The paper of (Modgil and Caminada, 2009) contains a typo where it says that line 5 should be *undec*($\mathcal{L}'$) $\subset$ *in*($\mathcal{L}$). This, however, will result in wrong labellings. The algorithm that computes semi-stable labellings is as follows.

Algorithm 5 Semi-Stable Labellings (Modgil and Caminada, 2009)

```
 1: function FIND_SEMI-STABLE_LABELLINGS($\mathcal{L}$)
 2:     if $\exists \mathcal{L}' \in$ candidate_Labellings : $undec(\mathcal{L}') \subset undec(\mathcal{L})$ then return

 3:     if $\mathcal{L}$ does not contain an argument that is illegally IN then
 4:         for each $\mathcal{L}' \in$ candidate_Labellings do
 5:             if $undec(\mathcal{L}) \subset undec(\mathcal{L}')$ then
 6:                 candidate_Labellings $\leftarrow$ candidate_Labellings $- \{\mathcal{L}'\}$
 7:         candidate_Labellings $\leftarrow$ candidate_Labellings $\cup \{\mathcal{L}\}$
 8:         return
 9:     else
10:         if $\mathcal{L}$ has an argument that is super-illegally IN then
11:             $x \leftarrow$ some argument $\in \mathcal{L}$ that is super-illegally IN
12:             FIND_SEMI-STABLE_LABELLINGS(TRANSITION_STEP($\mathcal{L}, x$))
13:         else
14:             for each $x \in \mathcal{L}$, that is illegally IN do
15:                 FIND_SEMI-STABLE_LABELLINGS(TRANSITION_STEP($\mathcal{L}, x$))
```

If we apply the algorithm above to the examples in Figures 1, 2 and 3, we get the following semi-stable labellings.

- Figure 1: $\mathcal{L}_{semi\_stable} = (\{a, c\}, \{b\}, \emptyset)$. As there was a stable labelling, we get the same semi-stable labelling.
- Figure 2: $\mathcal{L}_{semi\_stable} = (\{b, d\}, \{a, c, e\}, \emptyset)$. Same as the previous example.
- Figure 3: $\mathcal{L}_{semi\_stable} = (\emptyset, \emptyset, \{a, b, c, d, e\})$. In contrast to stable labellings, we now have a semi-stable labelling for this AF. Which is the same as the preferred labelling.

3.5 Stage semantics

Introduced in (Verheij, 1996) and further developed in (Verheij, 2003), stage semantics is one of the oldest semantics for argumentation frameworks. Instead of the usual extension-based approach, stage semantics were originally stated in the form of pairs (J, D). Here, J is a set of justified arguments and D is a set of defeated arguments. Although stage semantics have remained relatively unknown in the literature, we have good reasons to accept it as one of the mainstream semantics for abstract argumentation, as it implements a fundamentally different intuition compared to the traditional admissibility-based semantics discussed in the preceding sections (Caminada, 2010). Despite this alternative notation, it is still possible to translate it to the extension- and labelling-based approach. This allows us to provide an algorithm for computing all stage labellings, as shown by (Caminada, 2010). The definition of Caminada for the stage extensions is as follows.

Definition 22 *Let $AF = (Ar, att)$ be an argumentation framework. A* stage extension *is a conflict-free set $Args \subseteq Ar$ where $Args \cup Args^+$ is maximal (w.r.t. set inclusion) set among all conflict-free sets.*

When we compare the definitions of semi-stable and stage extensions, we see that instead of being admissibility based, stage extensions are based on conflict-free sets. Because every admissible set is conflict-free, we get that every stable extension is also a stage extension. If there exists at least one stable extension, then it also holds that every stage extension is a stable extension (Baroni et al., 2011). When we regard stage semantics in the labelling-based approach, we get the following definition of (Caminada, 2010).

Definition 23 *Let $Af = (Ar, att)$ be an argumentation framework. $\mathcal{L}$ is a* stage labelling *iff $\mathcal{L}$ is a conflict-free labelling where $UNDEC$ is minimal (w.r.t. set inclusion) among all conflict-free labellings.*

Where maximizing $Args \cup Args^+$ for stage extension $Args$ is the same as minimizing the $UNDEC$ set, since $UNDEC = Ar - (Args \cup Args^+)$. Because of the shift from admissibility to conflict-freeness, we need to redefine what it means for arguments to be IN or OUT in a conflict-free labelling. This is done as follows in (Caminada, 2010).

Definition 24 *Let $AF = (Ar, att)$ be an argumentation framework. A* conflict-free labelling *is a labelling $\mathcal{L}$ such that for every $A \in Ar$ it holds that:*

- *if $Lab(A) = IN$, then $\forall B \in Ar : (B \; att \; A \supset Lab(B) \neq IN)$.*
- *if $Lab(A) = OUT$, then $\exists B \in Ar : (B \; att \; A \wedge Lab(B) = IN)$.*

The only difference is that for an admissible labelling the first clause is stronger, as it does not allow arguments labelled IN, to be attacked by an argument labelled $UNDEC$. The definition for illegally IN and OUT then also changes. We can redefine them in the following manner.

Definition 25 *Let $\mathcal{L}$ be a labelling of argumentation framework $AF = (Ar, att)$.*

1. *An IN-labelled argument A is called* illegally *IN, iff*
   $\exists B \in Ar : (B \; att \; A \wedge \mathcal{L}(B) = IN) \vee \exists B \in Ar : (A \; att \; B \wedge \mathcal{L}(B) = IN)$.

2. *An OUT-labelled argument A is called* illegally *OUT, iff* $\neg\exists B \in Ar : (B\ att\ A \wedge Lab(B) = IN)$.

This redefinition also alters the working of the transition step of Definition 15, although it stays the same on the surface, we use legally IN and OUT to determine which arguments need to change their label. Caminada then proceeds to prove that a terminated transition sequence with the new definitions results in a conflict-free labelling and that every stage labelling is found at the end of some terminated transition sequence.

3.5.1 Procedure for stage labellings

As stage labellings are very similar to semi-stable labellings, we can alter the code of Section 3.4.1 to find stage labellings. The only difference between the two being the definition of (il)legally IN and OUT. As this definition has changed, we can no longer make use of arguments that are super-illegally IN, meaning we can remove lines 10-13. This yields the following algorithm.

Algorithm 6 Stage Labellings (Caminada, 2010)

```
 1: function FIND_STAGE_LABELLINGS($\mathcal{L}$)
 2:     if $\exists\mathcal{L}' \in$ candidate_Labellings : $undec(\mathcal{L}') \subset undec(\mathcal{L})$ then return
 3:
 4:     if $\mathcal{L}$ does not contain an argument that is illegally IN then
 5:         for each $\mathcal{L}' \in$ candidate_Labellings do
 6:             if $undec(\mathcal{L}) \subset undec(\mathcal{L}')$ then
 7:                 candidate_Labellings ← candidate_Labellings − {$\mathcal{L}'$}
 8:         candidate_Labellings ← candidate_Labellings ∪ {$\mathcal{L}$}
 9:         return
10:     else
11:         for each $x \in \mathcal{L}$, that is illegally IN do
12:             FIND_SEMI-STABLE_LABELLINGS(TRANSITION_STEP($\mathcal{L}$, $x$))
```

Note that the definitions of illegally IN on lines 4 and 11 have changed. Also remember that the transition step takes into account the definition of legally IN and OUT, hence the transition step also needs to be altered.

If we apply the algorithm above to the examples in figure 1, 2 and 3, we get the following stage labellings.

- Figure 1: $\mathcal{L}_{stage} = (\{a, c\}, \{b\}, \emptyset)$. In this case the change from admissible to conflict-free, causes no difference.
- Figure 2: $\mathcal{L}_{stage} = (\{b, d\}, \{a, c, e\}, \emptyset)$. Same for this example. Here the cycle is influenced by argument b, only allowing d to be labelled IN.

- Figure 3: $\mathcal{L}_{stage1} = (\{c, e\}, \{a, d\}, \{b\})$, $\mathcal{L}_{stage2} = (\{b, e\}, \{c, d\}, \{a\})$, $\mathcal{L}_{stage3} = (\{a, e\}, \{b, d\}, \{c\})$. This AF suddenly has three stage labellings, as opposed to one semi-stable labelling. This is the case, because in the cycle we can label either of the three arguments *IN*, which was not the case in admissible labellings.

3.6 Judgement aggregation

Till now all semantics considered were from the perspective of a single agent, but what if a group of agents had different views and needed to arrive at a consensus. Assume we have a group of people that all have their own opinion on the arguments of an argumentation framework. The group tries to form an opinion together on which arguments to accept and they try to accept as much as possible. They start by examining what they all agree and disagree on and check whether this position is still defensible. If this is not the case, they water it down by abstaining from accepting and rejecting some arguments. They keep abstaining until their position is defensible (Baroni et al., 2011).

There are two types of judgement aggregation. The first is sceptical judgement aggregation, which is explained above. The other is credulous judgement aggregation, but here the idea is that it is not a problem to accept or reject arguments that are not accepted or rejected by each member of the group, as long as the private opinion of each group member is not directly against the group outcome (Caminada and Pigozzi, 2011). In this thesis, we will only look at the procedure for sceptical judgement aggregation.

To define a procedure for sceptical judgement aggregation we need to know what it means for a labelling to be less, equally or more committed than another labelling. This is defined in (Caminada and Pigozzi, 2011) as follows.

Definition 26 *For two Labellings $\mathcal{L}_1$ and $\mathcal{L}_2$ of argumentation framework $AF = (Ar, Atts)$, we say that:*

1. *$\mathcal{L}_1$ is* more or equally committed *than $\mathcal{L}_2$, $\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$, iff $in(\mathcal{L}_1) \subseteq in(\mathcal{L}_2)$ and $out(\mathcal{L}_1) \subseteq out(\mathcal{L}_2)$*
2. *$\mathcal{L}1$ is* less or equally committed *than $\mathcal{L}_2$, $\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$, iff $in(\mathcal{L}_1) \subseteq in(\mathcal{L}_2)$ and $out(\mathcal{L}_1) \subseteq out(\mathcal{L}_2)$*

We call a labelling *bigger* if it is more committed than another labelling, *smaller* if it is less committed than another labelling, and the *biggest* if it is the most committed labelling.

3.6.1 procedure for sceptical judgement aggregation

First, we need a procedure for the initial step of the group stage, where we initially accept unanimous accepted arguments and reject unanimous rejected arguments. We call this procedure the sceptical initial aggregation operator (Sio_{AF}). This procedure looks as follows.

Algorithm 7 Sceptical Initial Aggregation Operator (SIO$_{AF}$)

```
function SIO_AF($\mathcal{L}abs$)
    $\mathcal{L}_{init} \leftarrow (\emptyset, \emptyset, \emptyset)$

    for each argument $x \in \mathcal{A}$ do
        if argument $x \in in(\mathcal{L}), \forall \mathcal{L} \in \mathcal{L}abs$ then
            $in(\mathcal{L}_{init}) \leftarrow x \cup in(\mathcal{L}_{init})$
        else if argument $x \in out(\mathcal{L}), \forall \mathcal{L} \in \mathcal{L}abs$ then
            $out(\mathcal{L}_{init}) \leftarrow x \cup out(\mathcal{L}_{init})$
        else
            $undec(\mathcal{L}_{init}) \leftarrow x \cup undec(\mathcal{L}_{init})$

    return $\mathcal{L}_{init}$
```

The next step in the procedure is to water the initial labelling down until it is admissible. This is done by the contraction function. The contraction function relabels arguments from IN or OUT to $UNDEC$. (Caminada and Pigozzi, 2011) define it as follows.

Definition 27 *Let Labellings be the set of all possible labellings of argumentation framework $AF = (Ar, att)$. The* contraction function *is a function C_{AF} : $Labellings \times Ar \rightarrow Labellings$ such that $C_{AF}(L, A) = (L - (A, IN), (A, OUT)) \cup (A, UNDEC)$.*

Turning this function in an algorithm, yields the following.

Algorithm 8 Contraction Function: (C$_{AF}$)

```
function C_AF($\mathcal{L}, x$)
    $\mathcal{L}_{i+1} \leftarrow (\emptyset, \emptyset, \emptyset)$
    $in(\mathcal{L}_{i+1}) \leftarrow in(\mathcal{L}_i) - \{x\}$
    $out(\mathcal{L}_{i+1}) \leftarrow out(\mathcal{L}_i) - \{x\}$
    $undec(\mathcal{L}_{i+1}) \leftarrow \{x\} \cup undec(\mathcal{L}_i)$
    return $\mathcal{L}_{i+1}$
```

We now need to iterate over this function until the result is an admissible labelling. We call this chain of functions a contraction sequence, which is defined in (Caminada and Pigozzi, 2011) as follows.

Definition 28 *Let L be a labelling of argumentation framework $AF = (Ar, att)$.*
A contraction sequence *from $\mathcal{L}$ is a list of labellings $[\mathcal{L}_1, \ldots, \mathcal{L}_m]$ such that:*

1. *$\mathcal{L}_1 = \mathcal{L}$,*
2. *for each $j \in \{1, \ldots, m\}$: $L_{j+1} = C_{AF}(\mathcal{L}_j, A)$, where A is an argument that is illegally IN or illegally OUT in $\mathcal{L}_j$, and*

3. *$\mathcal{L}_m$ is a labelling without any illegal IN or illegal OUT.*

See (Caminada and Pigozzi, 2011) for a proof that the end of the contraction sequence results in the biggest admissible labelling. Finally, all that is left to do is turn this into an algorithm that performs the contraction sequence, we call this procedure the sceptical aggregation operator (So_{AF}).

Algorithm 9 Sceptical Aggregation Operator: (So_{AF})

```
1: function $\text{So}_{AF}(\mathcal{L}abs)$
2:     $\mathcal{L}_0 \leftarrow \text{Sio}_{AF}(\mathcal{L}abs)$
3:     $\mathcal{L}_m \leftarrow$ ITERATE_CONTRACTION_SEQUENCE($\mathcal{L}_0$)
4:     return $\mathcal{L}_m$

5: function ITERATE_CONTRACTION_SEQUENCE($\mathcal{L}$)
6:     if $\mathcal{L}$ does not contain any argument that is illegally IN or illegally OUT then
7:         return $\mathcal{L}$
8:     else
9:         x ← some argument that is Illegally IN or Illegally Out in $\mathcal{L}$
10:        $\mathcal{L}_n \leftarrow \text{C}_{AF}(\mathcal{L}, x)$
11:        return ITERATE_CONTRACTION_SEQUENCE($\mathcal{L}_n$)
```

Now we can use the procedure So_{AF} to compute the ideal and eager labellings in the following sections.

3.7 Ideal semantics

Ideal semantics were originally proposed in (Dung et al., 2006). The idea for the ideal semantics is to be slightly more sceptical than only taking the intersection of all preferred extensions. The reason for this being, that the intersection of all preferred extensions might not be an admissible set itself. This is demonstrated in (Caminada, 2007b). The definition for the ideal extension by Caminada, 2007b is as follows.

Definition 29 *The* ideal extension *is the greatest (w.r.t. set-inclusion) admissible set that is a subset of each preferred extension.*

The definition for ideal labellings as described in (Baroni et al., 2011) is as follows.

Definition 30 *Let $AF = (Ar, att)$ be an argumentation framework. The* ideal labelling *of AF is the biggest admissible labelling that is smaller or equal to each preferred labelling.*

3.7.1 Procedure for ideal labellings

As the ideal labellings are acquired using judgement aggregation, we need to have a new procedure call for this algorithm.

Algorithm 10 Ideal Labelling

```
1: function FIND_IDEAL_LABELLING
2:     candidate_Labellings ← $\emptyset$
3:     FIND_PREFERRED_LABELLINGS(all-in)
4:     preferred_Labellings ← candidate_Labellings
5:     ideal_Labelling ← $\text{So}_{AF}$(preferred_Labellings)
6:     return ideal_Labelling
```

If we apply the algorithm above to the examples in Figures 1, 2 and 3, we get the following ideal labellings.

- Figure 1: $\mathcal{L}_{ideal} = (\{a, c\}, \{b\}, \emptyset)$. This example has only one preferred labelling, hence we get the same ideal labelling.
- Figure 2: $\mathcal{L}_{ideal} = (\emptyset, \emptyset, \{a, b, c, d\})$. Here the ideal labelling is different from the preferred labellings. As the intersection was already empty, we get a totally *UNDEC* labelling.
- Figure 3: $\mathcal{L}_{ideal} = (\emptyset, \emptyset, \{a, b, c, d, e\})$. Same as in the first example.

3.8 Eager semantics

An alternative approach that is very close to ideal semantics is that of eager semantics, introduced in (Caminada, 2007b). Remember that the ideal extension is the (unique) biggest admissible (and complete) subset of each preferred extension. The eager extension is the (unique) biggest admissible (and complete) subset of each semi-stable extension. This makes eager semantics the most credulous unique status semantics in the literature (Baroni et al., 2011).

The definition for the eager extension is as follows Caminada, 2007b.

Definition 31 *The* eager extension *is the greatest (w.r.t. set-inclusion) admissible set that is a subset of each semi-stable extension.*

Again, we can take this definition and translate it into a definition for the labelling-approach.

Definition 32 *Let $AF = (Ar, att)$ be an argumentation framework. The* eager labelling *of AF is the biggest admissible labelling that is smaller or equal to each semi-stable labelling*

Just like the ideal extension, there exists exactly one eager extension for every argumentation framework.

3.8.1 Procedure for eager labellings

We only need to slightly alter the procedure of 3.7.1 to apply the sceptical aggregation operator on the semi-stable labellings, instead of the preferred labellings.

Algorithm 11 Eager Labelling

```
1: function FIND_EAGER_LABELLING
2:     candidate_Labellings ← $\emptyset$
3:     FIND_SEMI-STABLE_LABELLINGS(all-in)
4:     semi-stable_Labellings ← candidate_Labellings
5:     eager_Labelling ← $So_{AF}$(semi-stable_Labellings)
6:     return eager_Labelling
```

When we apply the algorithm above to the examples in Figures 1, 2 and 3, we get the following eager labellings.

- Figure 1: $\mathcal{L}_{eager} = (\{a, c\}, \{b\}, \emptyset)$. This example yields the same result as for semi-stable labellings, since there was only one.
- Figure 2: $\mathcal{L}_{eager} = (\{b, d\}, \{a, c, e\}, \emptyset)$. Here we also get exactly the semi-stable labelling.
- Figure 3: $\mathcal{L}_{eager} = (\emptyset, \emptyset, \{a, b, c, d, e\})$. The same here.

4 Conclusions

Since the introduction of Dung's theory of formal argumentation, various semantics have been devised for abstract argumentation. With this thesis, I aim to provide a basic overview of the most notable semantics and give a complete overview of how to implement the algorithms that find the respective labellings for each semantic. We saw that a lot of the semantics are closely related, often based on the principles of completeness, admissibility and conflict-freeness. The main difference between a lot of the semantics is the restriction placed upon complete or admissible labellings. The table below demonstrates how admissibility-based semantics can be expressed by placing different restrictions on complete labellings.

restriction on complete labelling	resulting semantics
no restrictions	complete semantics
empty `undec`	stable semantics
maximal `in`	preferred semantics
maximal `out`	preferred semantics
maximal `undec`	grounded semantics
minimal `in`	grounded semantics
minimal `out`	grounded semantics
minimal `undec`	semi-stable semantics
maximal w.r.t. $\sqsubseteq$ while compatible with each complete labelling	ideal semantics

Table 1: Describing admissibility based semantics in terms of complete labellings (Baroni et al., 2011, p. 23).

Another interesting way to see the relation between the semantics is by examining the ontology of argumentation semantics. We observed for example that every stable labelling is also a semi-stable labelling and that every semi-stable labelling is also a preferred labelling. Figure 4 depicts a graphical overview of an ontology of argumentation semantics. Note that the same relations hold in extension-based semantics. The figure does not include eager labellings, but those could be put at exactly the same place as ideal labellings, meaning that every eager labelling is also a complete labelling.

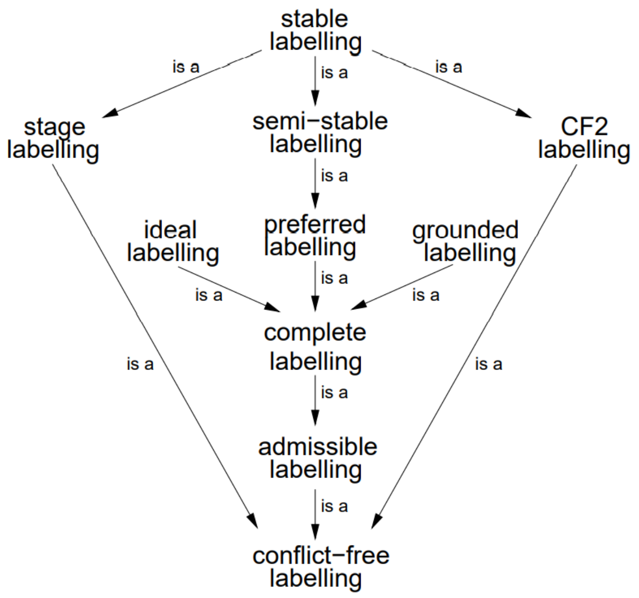

Figure 4: An ontology of labelling semantics (Baroni et al., 2011, p. 24).

The semantics in this paper are not the only semantics for argumentation frameworks. This overview could be extended by adding for example the robust, stage2 and CF2 semantics. Another possibility is examining how the semantics featured in this thesis behave in infinite argumentation frameworks, or one could define procedures that calculate extensions in extended argumentation frameworks with several attack relations. There is also the possibility for some new semantics to be devised.

The final step of this thesis was to turn the procedures into concrete and running algorithms. I am happy with the results of the algorithms I wrote and learned a lot while researching and describing these semantics. Explaining them is one thing but the joy you get when the algorithm finally works as intended after debugging it is indescribable. I hope the reader also learned a lot about argumentation frameworks and gets just as excited as I was to start implementing them.

References

Baroni, P., Caminada, M., & Giacomin, M. (2011). An introduction to argumentation semantics. *The knowledge engineering review*, *26*(4), 365–410.

Caminada, M. (2006). On the issue of reinstatement in argumentation. *European Workshop on Logics in Artificial Intelligence*, 111–123.

Caminada, M. (2007a). An algorithm for computing semi-stable semantics. *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, 222–234.

Caminada, M. (2007b). Comparing two unique extension semantics for formal argumentation: Ideal and eager. *Proceedings of the 19th Belgian-Dutch conference on artificial intelligence (BNAIC 2007)*, 81–87.

Caminada, M. (2010). An algorithm for stage semantics. In *Computational models of argument* (pp. 147–158). IOS Press.

Caminada, M. (2011). A labelling approach for ideal and stage semantics. *Argument & Computation*, *2*(1), 1–21.

Caminada, M., & Pigozzi, G. (2011). On judgment aggregation in abstract argumentation. *Autonomous Agents and Multi-Agent Systems*, *22*(1), 64–102.

Caminada, M., Verheij, B., Danoy, G., Seredynski, M., Booth, R., Gateau, B., Jars, I., & Khadraoui, D. (2010). On the existence of semi-stable extensions. *argumentation*, *3*, 4.

Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, *77*(2), 321–357.

Dung, P. M., Mancarella, P., & Toni, F. (2006). A dialectic procedure for sceptical, assumption-based argumentation. *Comma*, *144*, 145–156.

Elaroussi, M., Nourine, L., Radjef, M. S., & Vilmin, S. (2022). On the preferred extensions of argumentation frameworks: Bijections with naive extensions. *arXiv preprint arXiv:2202.05506*.

Koons, R. (2005). Defeasible reasoning. *Stanford Encyclopedia of Philosophy*. https://plato.stanford.edu/entries/reasoning-defeasible/.

McCarthy, J. (1981). Epistemological problems of artificial intelligence. In *Readings in artificial intelligence* (pp. 459–465). Elsevier.

McCarthy, J., & Hayes, P. J. (1981). Some philosophical problems from the standpoint of artificial intelligence. In *Readings in artificial intelligence* (pp. 431–450). Elsevier.

Modgil, S., & Caminada, M. (2009). Chapter 6: Proof theories and algorithms for abstract argumentation frameworks. *Argumentation in AI*, 105–129.

Pollock, J. L. (1995). *Cognitive carpentry: A blueprint for how to build a person*. Mit Press.

Prakken, H. (2010). An abstract framework for argumentation with structured arguments. *Argument & Computation*, *1*(2), 93–124.

Strasser, C., & Antonelli, G. A. (2001). Non-monotonic logic. *Stanford Encyclopedia of Philosophy*. https://plato.stanford.edu/entries/logic-nonmonotonic/.

Verheij, B. (1996). Two approaches to dialectical argumentation: Admissible sets and argumentation stages. *Proc. NAIC*, *96*, 357–368.

Verheij, B. (2003). Deflog: On the logical interpretation of prima facie justified assumptions. *Journal of Logic and Computation*, *13*(3), 319–346.