On the Equivalence between Logic Programming Semantics and Argumentation Semantics

Martin Caminada[a], Samy Sá[b], João Alcântara[b], Wolfgang Dvořák[c]

[a] *University of Aberdeen*
[b] *Universidade Federal do Ceará*
[c] *University of Vienna, Faculty of Computer Science, Austria*

Abstract

In the current paper, we re-examine the connection between formal argumentation and logic programming from the perspective of semantics. We observe that one particular translation from logic programs to instantiated argumentation (the one described by Wu, Caminada and Gabbay) is able to serve as a basis for describing various equivalences between logic programming semantics and argumentation semantics. In particular, we are able to show equivalence between regular semantics for logic programming and preferred semantics for formal argumentation. We also show that there exist logic programming semantics (L-stable semantics) that cannot be captured by any abstract argumentation semantics.

Keywords: Abstract Argumentation Semantics, Logic Programming Semantics

1. Introduction

The connection between logic programming and formal argumentation goes back to logic programming inspired formalisms like the work of Prakken and Sartor [1] or the work of Simari and Loui [2], as well as to the seminal work of Dung [3] in which various connections were pointed out. To some extent, the work of Dung [3] can be seen as an attempt to provide an abstraction of certain aspects of logic programming. The connection between logic

Email addresses: martin.caminada@abdn.ac.uk (Martin Caminada), samy@ufc.br (Samy Sá), jnando@lia.ufc.br (João Alcântara), wolfgang.dvorak@univie.ac.at (Wolfgang Dvořák)

programming and argumentation is especially clear when it comes to comparing the different semantics that have been defined for logic programming with the different semantics that have been defined for formal argumentation. In the current paper, we continue this line of research. We do this by pointing out that one particular translation from logic programming to formal argumentation (the one of Wu et al. [4]) is able to account for a whole range of equivalences between logic programming semantics and formal argumentation semantics. This includes both existing results like the equivalence between stable model semantics (LP) and stable semantics (argumentation) [3], between well-founded semantics (LP) and grounded semantics (argumentation) [3], and between partial stable model semantics (LP) and complete semantics [4], as well as a newly proved equivalence between regular model semantics (LP) and preferred semantics (argumentation).

Our work is based on the fact that argumentation semantics are defined on the argument level, whereas logic programming semantics are defined on the conclusion level (with an argument being a defeasible derivation for a particular conclusion). Moreover, it holds that some of the most common argumentation semantics (grounded, preferred, semi-stable and stable) are based on selecting the complete labellings [5, 6] where a particular label is maximal or minimal whereas some of the most common logic programming semantics (well-founded, regular, L-stable and stable) are based on selecting the P-stable models where a particular truth value is maximal or minimal. This, together with the previously observed equivalence between complete labellings and P-stable models [4] allows us to examine any additional equivalences in a systematic way: does maximizing or minimizing a particular label on the argument level coincide with maximizing or minimizing a particular truth value on the conclusion level?

The results of the current paper, however, are relevant for more than just the connection between logic programming and formal argumentation. They also shed light on specific aspects of instantiated argumentation theory in general (e.g. [7, 8, 9]). In particular, we show the connection between argument-labellings at the abstract level and conclusion-labellings at the instantiated level. With one notable exception, we are able to show that maximizing (or minimizing) a particular label (`in`, `out` or `undec`) at the argument level coincides with maximizing (or minimizing) the same label at the conclusion level. These results are relevant as they indicate the possibilities (and limitations) of applying argument-based abstractions to formalisms for non-monotonic reasoning.

This paper is structured as follows. First, in Section 2, we introduce the main concepts to be applied in the current paper, such as the various semantics of abstract argumentation and logic programming to be examined. Then, in Section 3 we provide an overview of the three step process of instantiated argumentation and how it is applied in the particular context of logic programming based argumentation. In Section 4 we examine some existing work on the minimization and maximization of argument labellings. Similarly, in Section 5 we examine the issue of minimization and maximization of conclusion labellings. The connection between argument labellings and conclusion labellings is then studied in Section 6. We use this connection to study the equivalence between argumentation semantics and logic programming semantics in Section 7. For this, we point out that argument labellings coincide with argument extensions and conclusion labellings coincide with logic programming models. One notable exception on the equivalence between argumentation semantics and logic programming semantics is studied in Section 8, where we examine possible ways in which this equivalence can be restored. A reverse translation from argumentation frameworks to logic programs is then specified in Section 9, and it is observed that for this translation the equivalence of argumentation semantics and logic programming semantics is even stronger than for the translation of (unrestricted) logic programs to argumentation frameworks. Finally, we round off with a discussion of the obtained results in Section 10.

2. Preliminaries

In this section, we introduce the main definitions used throughout the paper as well as the first connections between formal argumentation and logic programming. We start with the definitions of abstract argumentation frameworks and their various semantics and then move on to logic programs and their various semantics. In order to highlight similarities between these concepts, we provide definitions of each formalism in a similar fashion.

2.1. Abstract Argumentation Frameworks and Semantics

In the current paper, we follow the approach of Dung [3]. To simplify things, we restrict ourselves to finite argumentation frameworks.

Definition 1 ([3])**.** *An* argumentation framework *is a pair* (Ar, att) *where* Ar *is a finite set of arguments and* $att \subseteq Ar \times Ar$*.*

Arguments are related to others by the attack relation att, in the sense that an argument A attacks the argument B iff $(A, B) \in att$. An argumentation framework can be depicted as a directed graph where the arguments are nodes and each attack is an arrow.

Definition 2 ([3]). *(defense/conflict-free). Let (Ar, att) be an argumentation framework, $A \in Ar$ and $\mathcal{A}rgs \subseteq Ar$.*

- *$\mathcal{A}rgs$ is said to be* conflict-free *iff there exists no arguments $A, B \in \mathcal{A}rgs$ such that $(A, B) \in att$.*
- *$\mathcal{A}rgs$ is said to* defend *an argument A iff every argument that attacks A is attacked by some argument in $\mathcal{A}rgs$.*
- *The* characteristic function *$F : 2^{Ar} \rightarrow 2^{Ar}$ is defined as $F(\mathcal{A}rgs) = \{A | A \text{ is defended by } \mathcal{A}rgs\}$.*
- *A conflict-free set $\mathcal{A}rgs$ is said to be* admissible *iff $\mathcal{A}rgs \subseteq F(\mathcal{A}rgs)$, that is the arguments in the set can defend themselves against any attackers in the framework.*
- *We write $\mathcal{A}rgs^+ = \{A | A \text{ is attacked by an argument in } \mathcal{A}rgs\}$ to refer to the set of arguments attacked by $\mathcal{A}rgs$.*

The traditional approaches to argumentation semantics are based on sets (commonly referred to as "extensions") of arguments. Some of the mainstream approaches are summarized in the following definition.[1]

Definition 3. *(extension-based argumentation semantics). Given an argumentation framework $AF = (Ar, att)$ and $S \subseteq Ar$:*

- *S is a complete extension of AF iff S is a conflict-free fixpoint of F (that is, if S is conflict-free and $S = F(S)$).*
- *S is a grounded extension of AF iff S is the minimal (w.r.t. set inclusion) conflict-free fixpoint of F.*

[1]The characterization of the extension-based semantics in Definition 3 is slightly different than the way these were originally defined, by Dung [3], but equivalence is proved by Caminada and Gabbay [6].

- *S is a preferred extension of AF iff S is a maximal (w.r.t. set inclusion) conflict-free fixpoint of F.*
- *S is a stable extension of AF iff S is a conflict-free fixpoint of F such that $S \cup S^+ = Ar$.*
- *S is a semi-stable extension of AF iff S is a conflict-free fixpoint of F with maximal $S \cup S^+$ (w.r.t. set inclusion).*

From Definition 3, it directly follows that every grounded, preferred, stable or semi-stable extension of a given argumentation framework is also a complete extension of that argumentation framework.

Example 1. *Let $AF = (Ar, att)$ be an abstract argumentation framework such that $Ar = \{A_1, \ldots, A_6\}$ and $att = \{(A_6, A_4), (A_4, A_6), (A_4, A_5), (A_5, A_3), (A_3, A_3)\}$. We depict AF as a directed graph, in Figure 1.*

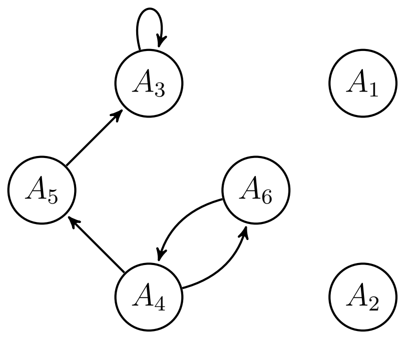


Figure 1: An abstract argumentation framework with 6 arguments.

Concerning semantics, AF has:

- Complete extensions: $\{A_1, A_2\}$, $\{A_1, A_2, A_5, A_6\}$, and $\{A_1, A_2, A_4\}$.
- Grounded extension: $\{A_1, A_2\}$.
- Preferred extensions: $\{A_1, A_2, A_5, A_6\}$, and $\{A_1, A_2, A_4\}$.
- Stable extensions: $\{A_1, A_2, A_5, A_6\}$.
- Semi-stable extensions: $\{A_1, A_2, A_5, A_6\}$.

It is worth to note that semi-stable semantics coincides with the stable semantics whenever the framework has at least one stable extension. The reason is straightforward: A stable extension of (Ar, att) is characterized by having $S \cup S^+ = Ar$, which implies having maximal $S \cup S^+$. As a consequence, every stable extension is also a semi-stable extension. Furthermore, the existence of at least one stable extensions is sufficient for every semi-stable extension also to be a stable extension.

2.2. Logic Programs and Semantics

In the current paper, we account for propositional normal logic programs[2], which we call logic programs or simply programs from now on.

Definition 4. *A* rule *r is an expression $r : c \leftarrow a_1, \ldots, a_n, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ ($n \geq 0$, $m \geq 0$) where c, each a_i ($1 \leq i \leq n$) and each b_j ($1 \leq j \leq m$) are atoms and* `not` *represents negation as failure. c is called the* head *of the rule, and $a_1, \ldots, a_n, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ is called the* body *of the rule. Furthermore, $a_1, \ldots, a_n$ is called the* strong *part of the body and $\texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ is called the* weak *part of the body. Let r be a rule, we write $head(r)$ to denote its head (the atom c), $body^+(r)$ to denote the set $\{a_1, \ldots, a_n\}$ and $body^-(r)$ to denote the set $\{\texttt{not}\ b_1, \ldots, \texttt{not}\ b_m\}$. A logic program P is then defined as a finite set of rules. If every rule $r \in P$ has $body^-(r) = \emptyset$, we say that P is a positive program. The* Herbrand Base *of a program P is the set HB_P of all atoms appearing in the program.*

A wide range of logic programs semantics can be defined based on the 3-valued interpretations of programs [11], defined as follows.

Definition 5. *A 3-valued Herbrand Interpretation I of a logic program P is a pair $\langle T, F \rangle$ with $T, F \subseteq HB_P$ and $T \cap F = \emptyset$. The atoms in T are said to be* true*, the atoms in F are said to be* false *and the atoms in $HB_P \setminus (T \cup F)$ are said to be* undefined.

Let I be a 3-valued Herbrand Interpretation of the logic program P. The reduct of P with respect to I (written as P/I) is the logic program constructed using the following steps.

[2]These are logic programs whose rules may contain weak but not strong negation, and where the head of each rule is a single atom [10].

1. Starting from P, remove each rule r from P that has $\texttt{not}\ b_i \in body^-(r)$ for some $b_i \in T$;
2. From the result of step 1, for each rule, for every $b_i \in F$, remove $\texttt{not}\ b_i$ from the body of the rule.
3. From the result of step 2, replace any remaining occurrences of $\texttt{not}\ b_i$ by $\mathbf{u}$.

In the above procedure, $\mathbf{u}$ is an atom not in HB_P which is undefined in all interpretations of P (a constant). It can be observed that P/I is a positive program, since all instances of weak negation have been removed. As a consequence, P/I has a unique least 3-valued model [11] denoted as $\Psi_P(I) = \langle T_\Psi, F_\Psi \rangle$[3] with minimal T_Ψ and maximal F_Ψ (w.r.t. set inclusion) such that, for every $a \in HB_P$:

- $a \in T_\Psi$ if there is a rule $r' \in P/I$ with $head(r') = a$ and $body^+(r') \subseteq T_\Psi$;
- $a \in F_\Psi$ if every rule $r' \in P/I$ with $head(r') = a$ has $body^+(r') \cap F_\Psi \neq \emptyset$;

Given the above preliminaries, we are now ready to describe the logic programming semantics studied in this paper.

Definition 6. *(logic programming semantics). Let $I = \langle T, F \rangle$ be a 3-valued Herbrand Interpretation of logic program P.*

- *I is a partial stable*[4] *(or P-stable) model of P iff $\Psi_P(I) = I$.*
- *T is a well-founded model of P iff I is a P-stable model of P where T is minimal (w.r.t. set inclusion) among all P-stable models of P.*
- *T is a regular model of P iff I is a P-stable model of P where T is maximal (w.r.t. set inclusion) among all P-stable models of P.*

[3]The above definition consists of a least fix-point of the immediate consequence operator Ψ defined by Przymusinski [11], which is guaranteed to exist and be unique for positive programs.

[4]We use "partial models" in the sense of the 1991 paper of Przymusinski [12], where partial models are the same as the 3-valued models. The name "partial models" was previously used in a 1990 paper of Saccà and Zaniolo [13] to refer to M-stable models (equivalent to regular models as shown by You et al. [14]) instead of P-stable models (equivalent to 3-valued models as shown by Eiter et al. [15]), but this nomenclature is no longer used.

- *T is a (2-valued) stable model of P iff I is a P-stable model of P where $T \cup F = HB_P$.*
- *T is an L-stable model of P iff I is a P-stable model of P where $T \cup F$ is maximal (w.r.t. set inclusion) among all P-stable models of P.*

Some of the definitions above are not standard in the logic programming literature, but can be found via equivalence results in different papers. The definition of a partial stable model is compatible with that in the work of Przymusinski [11, 12] (where it is also called a *3-valued stable model*). The above definition of a stable model and the well-founded model also goes back to the work of Przymusinski [11]. The other definitions, namely of regular and L-stable models are based on the work of Eiter et al. [15], where authors discuss partial stable models (P-stable models) and variations such as maximal (M-stable) models and least undefined (L-stable) models. In their paper [15] it is shown that for normal logic programs, P-stable models coincide with Przymusinski's original notion of 3-valued stable models [11], even though they only consider the T part of the 3-valued interpretations. The M-stable models are then defined as the maximal P-stable models and shown to be equivalent to the regular models. Our definition of L-stable models is then the same as in the work of Eiter et al. [15], where it is based on Przymusinski's notion of 3-valued stable models [11] instead of being based on the equivalent notion of P-stable models.

The advantage of our slightly different (though equivalent) way of describing logic programming semantics (Definition 6) is that well-founded, regular, stable and L-stable models are all specified as special cases of P-stable models. This is similar to what happens in Defintion 3, where grounded, preferred, stable and semi-stable extensions are specified as special forms of complete extensions. In fact, the similarity between Definition 6 and Definition 3 will turn out to be quite useful for examining the differences and similarities between argumentation and logic programming (see sections 6 and 7).

The following example should help the reader to further understand the above concepts.

Example 2. *Consider a normal logic program P with rules $\{r_1, ..., r_6\}$:*

$$\begin{array}{llll}
r_1: & b \leftarrow c, \texttt{not}\ a & r_2: & a \leftarrow \texttt{not}\ b \\
r_3: & p \leftarrow c, d, \texttt{not}\ p & r_4: & p \leftarrow \texttt{not}\ a \\
r_5: & c \leftarrow d & r_6: & d \leftarrow
\end{array}$$

The logic program from Example 2 has:

- P-stable models: $\langle \{d, c\}, \{\ \}\rangle$, $\langle \{d, c, p, b\}, \{a\}\rangle$, $\langle \{d, c, a\}, \{b\}\rangle$.
- Well-founded model: $\{d, c\}$.
- Regular models: $\{d, c, p, b\}$, $\{d, c, a\}$.
- Stable models: $\{d, c, p, b\}$.
- L-stable models: $\{d, c, p, b\}$.

It is worth to note that L-stable semantics coincides with the stable semantics whenever the program has one or more stable models. The reason is straightforward: Any stable model of P is a P-stable model with $T \cup F = HB_P$, which means maximal $T \cup F$. As a consequence, every stable model is also an L-stable model. Furthermore, the existence of at least one stable model is sufficient to assure that each L-stable model is also a (2-valued) stable model. We also note that each logic program has a unique well-founded model, and that there exist logic programs that do not have any (2-valued) stable models.

3. Logic Programming as Argumentation; a three-step process

In the current section, we examine how argumentation theory can be applied in the context of logic programming. In essence, our treatment is based on the approach of Wu et al. [4].[5] The idea is to apply the standard three-step process of instantiated argumentation, similar to what is done in, for instance, ASPIC [7, 8] and logic-based argumentation [18]. One starts with a particular knowledge base and constructs the associated argumentation framework (step 1), then applies abstract argumentation semantics (step 2) and subsequently looks at what the results of the argumentation semantics imply at the level of conclusions (step 3). We now specify what this process looks like in the specific context of logic programming.

[5]One particular difference is that in our current approach, arguments are recursive structures, whereas in the approach of Wu at al. [4], they are trees of rules. The disadvantage of the latter approach is that if one identifies the nodes of a tree with rules, one is not able to apply the same rule at different positions in the argument. The approach in the current paper, which is based on ASPIC [8, 16] and the ideas of Vreeswijk [17], avoids this problem.

3.1. Step 1: Argumentation Framework Construction

The approach of instantiated argumentation starts with a particular given knowledge base. In the context of logic programming, the knowledge base consists of a logic program. For current purposes, we consider this logic program to be normal, as we defined previously. Based on a particular logic program P, one can then start to construct *arguments*, which is done in the following recursive way:

Definition 7. *Let P be a logic program.*

- *If $c \leftarrow \texttt{not}\ b_1, \dots, \texttt{not}\ b_m$ is a rule in P then it is also an argument (say A) with*
  - $\texttt{Conc}(A) = c$,
  - $\texttt{Rules}(A) = \{c \leftarrow \texttt{not}\ b_1, \dots, \texttt{not}\ b_m\}$,
  - $\texttt{Vul}(A) = \{b_1, \dots, b_m\}$, *and*
  - $\texttt{Sub}(A) = \{A\}$.
- *If $c \leftarrow a_1, \dots, a_n, \texttt{not}\ b_1, \dots, \texttt{not}\ b_m$ is a rule in P and for each a_i ($1 \leq i \leq n$) there exists an argument A_i with $\texttt{Conc}(A_i) = a_i$ and $c \leftarrow a_1, \dots, a_n, \texttt{not}\ b_1, \dots, \texttt{not}\ b_m$ is not contained in $\texttt{Rules}(A_i)$ then $c \leftarrow (A_1), \dots, (A_n), \texttt{not}\ b_1, \dots, \texttt{not}\ b_m$ is an argument (say A) with*
  - $\texttt{Conc}(A) = c$,
  - $\texttt{Rules}(A) = \texttt{Rules}(A_1) \cup \dots \cup \texttt{Rules}(A_n) \cup \{c \leftarrow a_1, \dots, a_n, \texttt{not}\ b_1, \dots, \texttt{not}\ b_m\}$
  - $\texttt{Vul}(A) = \texttt{Vul}(A_1) \cup \dots \cup \texttt{Vul}(A_n) \cup \{b_1, \dots, b_m\}$, *and*
  - $\texttt{Sub}(A) = \{A\} \cup \texttt{Sub}(A_1) \cup \dots \cup \texttt{Sub}(A_n)$.

In essence, an argument can be seen as a tree-like structure of rules (the only difference with a real tree is that a rule can occur at more than one place in the argument)[6] and in examples we often represent it as such. If A

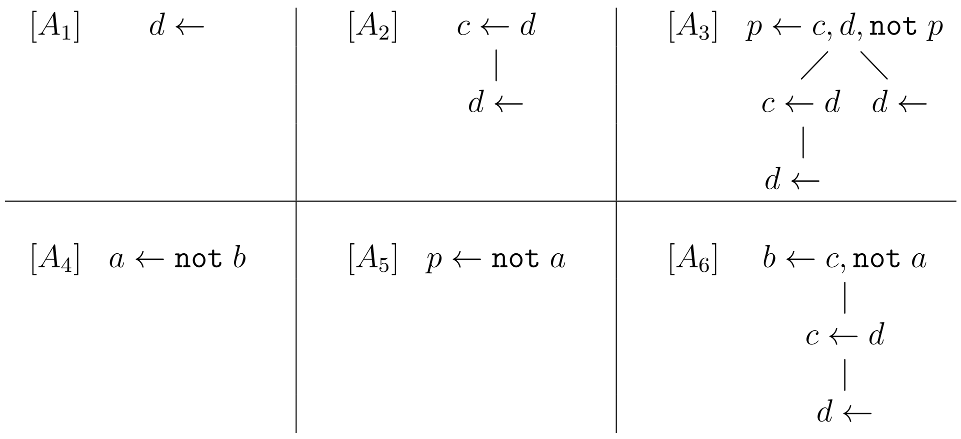
[6] Notice that Definition 7 allows the same rule to occur multiple times if it is in different branches, but not if it is in the same branch. This implies that, for instance, for logic program $P = \{a \leftarrow b, c;\ b \leftarrow d;\ c \leftarrow d;\ d \leftarrow;\ e \leftarrow f;\ f \leftarrow e;\ f \leftarrow\}$, $a \leftarrow (b \leftarrow (d \leftarrow)), (c \leftarrow (d \leftarrow))$ is a well-formed argument, whereas $e \leftarrow (f \leftarrow (e \leftarrow (f \leftarrow)))$ is not. This is to prevent a finite logic program from generating an infinite number of arguments, which would be particularly troublesome in the context of semi-stable semantics (see [19, 20]).

is an argument then $\texttt{Conc}(A)$ is referred to as the *conclusion* of A, $\texttt{Rules}(A)$ is referred to as the *rules* of A, $\texttt{Vul}(A)$ is referred to as the *vulnerabilities* of A and $\texttt{Sub}(A)$ is referred to as the *subarguments* of A.

Example 3. *Given the logic program P from Example 2, one can construct the following arguments:*

$$\begin{array}{llll}
A_1: & d \leftarrow & A_2: & c \leftarrow (A_1) \\
A_3: & p \leftarrow (A_2), (A_1), \texttt{not}\ p & A_4: & a \leftarrow \texttt{not}\ b \\
A_5: & p \leftarrow \texttt{not}\ a & A_6: & b \leftarrow (A_2), \texttt{not}\ a
\end{array}$$

In our example, it holds that $\texttt{Conc}(A_1) = d$, $\texttt{Conc}(A_2) = c$, $\texttt{Conc}(A_3) = p$, $\texttt{Conc}(A_4) = a$, $\texttt{Conc}(A_5) = p$ and $\texttt{Conc}(A_6) = b$. Furthermore, $\texttt{Vul}(A_1) = \texttt{Vul}(A_2) = \emptyset$, $\texttt{Vul}(A_3) = \{p\}$, $\texttt{Vul}(A_4) = \{b\}$, and $\texttt{Vul}(A_5) = \texttt{Vul}(A_6) = \{a\}$. The arguments are graphically depicted in Figure 2.



$[A_1]$ $d \leftarrow$
$[A_2]$ $c \leftarrow d$
$d \leftarrow$
$[A_3]$ $p \leftarrow c, d, \texttt{not}\ p$
$c \leftarrow d$
$d \leftarrow$
$d \leftarrow$
$[A_4]$ $a \leftarrow \texttt{not}\ b$
$[A_5]$ $p \leftarrow \texttt{not}\ a$
$[A_6]$ $b \leftarrow c, \texttt{not}\ a$
$c \leftarrow d$
$d \leftarrow$

Figure 2: Arguments constructed from P.

We draw attention of the reader to the relations amongst these arguments. Observe, for instance, that A_1 is a subargument of A_2. Furthermore, A_2 is a subargument of A_3 and A_6 and, as consequence, A_1 is a subargument of A_3 and A_6.

The next step in constructing the argumentation framework is to determine the attack relation: An argument attacks another iff its conclusion is one of the vulnerabilities of the attacked argument.

Definition 8. *Let A and B be arguments in the sense of Definition 7. We say that A* attacks *B iff* $\texttt{Conc}(A) \in \texttt{Vul}(B)$*.*

For the arguments of Figure 2, it holds that A_6 attacks A_4, A_4 attacks A_6 (mutual attacks), A_4 attacks A_5, A_5 attacks A_3, and A_3 attacks itself. The resulting argumentation framework (depicted in Figure 3) is essentially the same argumentation framework as in Example 1 (more precisely, it is isomorphic).

The notion of attack has a clear conceptual meaning. The fact that $b \in \texttt{Vul}(A)$ means that A is constructed using at least one rule containing `not` b in its body. In essence, A is a defeasible derivation that depends on b not being derivable. An argument B that provides a (possibly defeasible) derivation of b (that is, $\texttt{Conc}(B) = b$) can therefore be seen as *attacking* A.

Using the thus defined concepts of arguments and attacks, one is then able to define the argumentation framework that is associated to a particular logic program.

Definition 9. *Let P be a logic program. We define its associated argumentation framework as $AF_P = (Ar_P, att_P)$ where Ar_P is the set of arguments in the sense of Definition 7 and att_P is the attack relation in the sense of Definition 8.*

As an example, the argumentation framework associated with the logic program of Example 2 is depicted in Figure 3.

3.2. Step 2: Applying Argumentation Semantics

Once the argumentation framework has been constructed, the next question becomes which arguments should be accepted and which arguments should be rejected. As we have seen in Section 2, several approaches have been stated in the literature for determining this. For current purposes, we focus on the concept of complete semantics [3], which can be defined using the concept of a complete labelling [5, 6].

Definition 10. *Let $AF = (Ar, att)$ be an argumentation framework. An* argument labelling *is a function $ArgLab : Ar \to \{\texttt{in}, \texttt{out}, \texttt{undec}\}$. An argument labelling is called a* complete argument labelling *iff for each $A \in Ar$ it holds that:*

- *if $ArgLab(A) = \texttt{in}$ then for every $B \in Ar$ that attacks A it holds that $ArgLab(B) = \texttt{out}$*

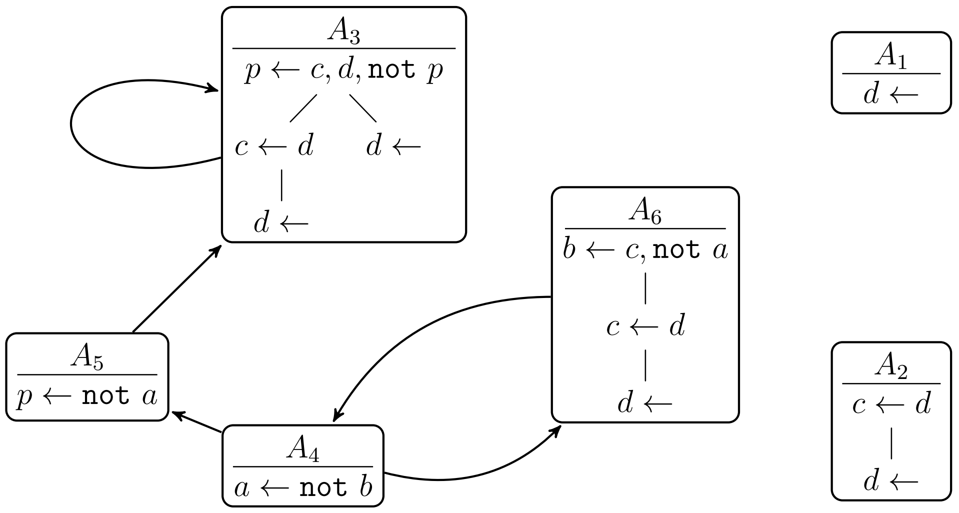


Figure 3: The abstract framework built using arguments instantiated from P

- *if* $ArgLab(A) = \texttt{out}$ *then there exists a* $B \in Ar$ *that attacks* A *such that* $ArgLab(B) = \texttt{in}$
- *if* $ArgLab(A) = \texttt{undec}$ *then (i) not every* $B \in Ar$ *that attacks* A *has* $ArgLab(B) = \texttt{out}$ *and (ii) no* $B \in Ar$ *that attacks* A *has* $ArgLab(B) = \texttt{in}$

With an argument labelling, one can express any arbitrary position on which arguments to accept (labelled `in`), which arguments to reject (labelled `out`) and which arguments to abstain from having an explicit opinion about (labelled `undec`). However, some of these positions are more reasonable than others. The idea of a complete labelling is that a position is reasonable iff one has sufficient reasons for each argument one accepts (all its attackers are rejected), for each argument one rejects (it has an attacker that is accepted) and for each argument one abstains (there are insufficient grounds to accept it and insufficient grounds to reject it).

When *ArgLab* is an argument labelling, we write $\texttt{in}(ArgLab)$ to denote the set of $\{A \mid ArgLab(A) = \texttt{in}\}$, $\texttt{out}(ArgLab)$ for $\{A \mid ArgLab(A) = \texttt{out}\}$ and $\texttt{undec}(ArgLab)$ for $\{A \mid ArgLab(A) = \texttt{undec}\}$. Since an argument labelling essentially defines a partition among the arguments (into a set of `in`-labelled arguments, a set of `out`-labelled arguments and a set of `undec`-labelled arguments), we sometimes write *ArgLab* as a triple $(\mathcal{A}rgs_1, \mathcal{A}rgs_2, \mathcal{A}rgs_3)$ where $\mathcal{A}rgs_1 = \texttt{in}(ArgLab)$, $\mathcal{A}rgs_2 = \texttt{out}(ArgLab)$ and $\mathcal{A}rgs_3 = \texttt{undec}(ArgLab)$.

In the argumentation framework of Figure 3, there are three possible complete labellings: $ArgLab_1 = (\{A_1, A_2\}, \{\ \}, \{A_3, A_4, A_5, A_6\})$, $ArgLab_2 = (\{A_1, A_2, A_5, A_6\}, \{A_3, A_4\}, \{\ \})$ and $ArgLab_3 = (\{A_1, A_2, A_4\}, \{A_5, A_6\}, \{A_3\})$.

3.3. Step 3: converting argument labellings to conclusion labellings

When it comes to practical questions like what to believe or what to do, in the end what is important are not so much the arguments themselves but their conclusions. In the argumentation process, this means that for each position on which *arguments* to accept, reject or abstain we need to determine the associated position on which *conclusions* to accept, reject or abstain.

For current purposes, we follow the approach described by Wu and Caminada [21]. Here, the idea is for each conclusion to identify the "best" argument that yields it. We assume a strict total order between the different individual labels such that $\texttt{in} > \texttt{undec} > \texttt{out}$. The best argument for a particular conclusion is then the argument with the highest label. In case there is no argument at all for a particular conclusion, the conclusion is simply labelled `out`.

Definition 11 ([21])**.** *Let P be a logic program. A conclusion labelling is a function $ConcLab : HB_P \rightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$ where HB_P is the set of all atoms occurring in P.*
Let $AF_P = (Ar_P, att_P)$ be an argumentation framework and $ArgLab$ be an argument labelling of AF_P. We say that $ConcLab$ is the associated conclusion labelling *of $ArgLab$ iff $ConcLab$ is a conclusion labelling such that for each $c \in HB_P$ it holds that $ConcLab(c) = max(\{ArgLab(A) \mid \texttt{Conc}(A) = c\} \cup \{\texttt{out}\})$ where $\texttt{in} > \texttt{undec} > \texttt{out}$. We say that a conclusion labelling is* complete *iff it is associated with a complete argument labelling.*

When $ConcLab$ is a conclusion labelling, we write $\texttt{in}(ConcLab)$ to denote the set $\{c \mid ConcLab(c) = \texttt{in}\}$, $\texttt{out}(ConcLab)$ for $\{c \mid ConcLab(c) = \texttt{out}\}$ and $\texttt{undec}(ConcLab)$ for $\{c \mid ConcLab(c) = \texttt{undec}\}$. Because a conclusion labelling essentially defines a partition among HB_P into three sets of `in`-labelled, `out`-labelled and `undec`-labelled conclusions respectively), we write $ConcLab$ as a triple $(\mathcal{C}oncs_1, \mathcal{C}oncs_2, \mathcal{C}oncs_3)$ where $\mathcal{C}oncs_1 = \texttt{in}(ConcLab)$, $\mathcal{C}oncs_2 = \texttt{out}(ConcLab)$ and $\mathcal{C}oncs_3 = \texttt{undec}(ConcLab)$.

Recall that the complete argument labellings of argumentation framework in Figure 3 are: $ArgLab_1 = (\{A_1, A_2\}, \{\ \}, \{A_3, A_4, A_5, A_6\})$, $ArgLab_2 =$

$(\{A_1, A_2, A_5, A_6\}, \{A_3, A_4\}, \{\ \})$, and $\mathit{ArgLab}_3 = (\{A_1, A_2, A_4\}, \{A_5, A_6\}, \{A_3\})$. In the same example, observe that two of the involved arguments yield conclusion p, namely $A_3 :\ p \leftarrow (A_5), (A_6), \texttt{not}\ p$ and $A_5 :\ p \leftarrow \texttt{not}\ a$. Observe that A_3 is labelled `out`, while A_5 is labelled `in` by ArgLab_2. As a consequence, the label of p in the conclusion labelling associated with ArgLab_2 is `in`. Using a similar reasoning, we obtain that the associated conclusion labelling $\mathit{ConcLab}_1$ of ArgLab_1 is $(\{c, d\}, \{\ \}, \{p, a, b\})$, the associated conclusion labelling $\mathit{ConcLab}_2$ of argument labelling ArgLab_2 is $(\{c, d, p, b\}, \{a\}, \{\ \})$, and the associated conclusion labelling $\mathit{ConcLab}_2$ of argument labelling ArgLab_2 is $(\{c, d, a\}, \{b\}, \{p\})$.

4. On the Minimization and Maximization of Argument Labellings

Now that the general overview of the three-step process has been provided, we subsequently zoom in on some of its steps, starting with the argument labellings level (step 2). In particular, we provide a brief overview (based on work originally published by Caminada et al. [5, 6]) of how the different argument labellings relate to each other, especially when it comes to maximizing or minimizing a particular label.

Theorem 12 ([5, 6])**.** *Let ArgLab be a complete argument labelling of argumentation framework $AF = (Ar, att)$. It holds that*

- `in`(ArgLab) *is maximal (w.r.t. set-inclusion) among all complete argument labellings of AF iff* `out`(ArgLab) *is maximal (w.r.t. set-inclusion) among all complete argument labellings of AF.*
- `in`(ArgLab) *is minimal (w.r.t. set-inclusion) among all complete argument labellings of AF iff* `out`(ArgLab) *is minimal (w.r.t. set-inclusion) among all complete argument labellings of AF.*

Theorem 13. *Let $AF = (Ar, att)$ be an argumentation framework. The complete argument labelling ArgLab where* `in`(ArgLab) *is minimal (w.r.t. set inclusion) among all complete argument labellings is unique.*

Theorem 14. *Let $AF = (Ar, att)$ be an argumentation framework and ArgLab be one of its complete argument labellings. Then* `undec`(ArgLab) *is maximal (w.r.t. set-inclusion) among all complete argument labellings iff* `in`(ArgLab) *is minimal (w.r.t. set-inclusion) among all complete argument labellings.*

To summarize, the complete argument labellings where `in` is maximal are the same as the complete argument labellings where `out` is maximal (Theorem 12). These argument labellings are referred to as *preferred argument labellings*. Furthermore, the unique complete argument labelling where `in` is minimal (Theorem 13) is the same as the unique complete argument labelling where `out` is minimal (Theorem 12) and the same as the unique complete argument labelling where `undec` is maximal (Theorem 14). This argument labelling is referred to as the *grounded argument labelling*. The complete argument labellings where `undec` is minimal are referred to as *semi-stable argument labellings*. The complete argument labellings where `undec` is empty are referred to as *argstable argument labellings*. In fact, from our results, it follows that (*i*) every argstable argument labelling is also semi-stable and (*ii*) every semi-stable argument labelling is also preferred. These kinds of argument labellings are further summarized in Table 1.

Condition	Resulting Argument Labelling
NONE	Complete
MAX In	Preferred
MAX Out	Preferred
MAX Undec	Grounded
MIN In	Grounded
MIN Out	Grounded
MIN Undec	Semi-stable
NO Undec	Argstable

Table 1: Kinds of argument labellings

It is relatively straightforward to define associated classes of conclusion labellings. For instance, a conclusion labelling is said to be a *preferred conclusion labelling* iff it is the associated conclusion labelling of a preferred argument labelling. Similarly, a conclusion labelling is said to be the *grounded conclusion labelling* iff it is the associated conclusion labelling of the grounded argument labelling, and a conclusion labelling is said to be a *semi-stable conclusion labelling* iff it is the associated conclusion labelling of a semi-stable argument labelling. Furthermore, a conclusion labelling is said to be an *argstable conclusion labelling* iff it is the associated conclusion labelling of an argstable argument labelling.

5. On the Minimization & Maximization of Conclusion Labellings

The concepts of preferred, grounded, semi-stable, and argstable conclusion labellings, as defined at the end of the previous section, are based on the common idea of performing the maximization/minimization at the level of argument labellings and then identifying the associated conclusion labellings. An alternative procedure would be simply to identify *all* complete conclusion labellings and then to perform the maximization/minimization right at the level of the conclusion labellings. In the current section, we analyze this alternative procedure. We observe that the thus derived conclusion labellings relate to each other in a way that is very similar to how the different types of argument labellings of the previous section relate to each other.

Theorem 15. *Let ConcLab be a complete conclusion labelling of logic program P and the associated argumentation framework $AF_P = (Ar_P, att_P)$. It holds that*

- `in`(*ConcLab*) *is maximal (w.r.t. set-inclusion) among all complete conclusion labellings of AF iff* `out`(*ConcLab*) *is maximal (w.r.t. set-inclusion) among all complete conclusion labellings of AF.*
- `in`(*ConcLab*) *is minimal (w.r.t. set-inclusion) among all complete conclusion labellings of AF iff* `out`(*ConcLab*) *is also minimal (w.r.t. set-inclusion) among all complete conclusion labellings of AF.*

Theorem 16. *Let P be a logic program and $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. The complete conclusion labelling ConcLab of AF_P where* `in`(*ConcLab*) *is minimal (w.r.t. set inclusion) among all complete conclusion labellings of AF_P is unique.*

Theorem 17. *Let P be a logic program, $AF_P = (Ar_P, att_P)$ be the associated argumentation framework of P and ConcLab be one of the complete conclusion labellings of AF_P. It holds that* `undec`(*ConcLab*) *is maximal (w.r.t. set-inclusion) among all complete conclusion labellings of AF_P iff* `in`(*ConcLab*) *is minimal (w.r.t. set-inclusion) among all complete conclusion labellings of AF_P.*

To summarize, the complete conclusion labellings where `in` is maximal are the same as the complete conclusion labellings where `out` is maximal

(Theorem 15). These argument labellings are referred to as *regular conclusion labellings*. Furthermore, the unique complete conclusion labelling where `in` is minimal (Theorem 16) is the same as the unique complete conclusion labelling where `out` is minimal (Theorem 15) and the same as the unique complete conclusion labelling where `undec` is maximal (Theorem 17). This conclusion labelling is referred to as the *well-founded conclusion labelling*. The complete conclusion labellings where `undec` is minimal are referred to as *L-stable conclusion labellings*. Furthermore, the complete conclusion labellings where `undec` is empty are referred to as *concstable conclusion labellings*. In fact, from our results, it follows that (*i*) every concstable conclusion labelling is also L-stable and (*ii*) every L-stable conclusion labelling is also a regular conclusion labelling. These kinds of conclusion labellings are further summarized in Table 2.

Condition	Resulting Conclusion Labelling
NONE	Complete
MAX In	Regular
MAX Out	Regular
MAX Undec	Well-founded
MIN In	Well-founded
MIN Out	Well-founded
MIN Undec	L-stable
NO Undec	Concstable

Table 2: Kinds of conclusion labellings derived from programs

6. Maximizing and Minimizing Argument Labellings vs. Maximizing and Minimizing Conclusion Labellings

So far, we have described two ways of selecting particular subsets of the complete conclusion labellings:

1. Perform minimization (resp. maximization) of a particular label at the level of complete argument labellings, then determine the associated conclusion labellings. This is the approach sketched in Section 4.
2. Take all complete conclusion labellings (these are the associated labellings of *all* complete argument labellings) and then perform the minimization (resp. maximization) of a particular label at the level

of complete conclusion labellings. This is the approach sketched in Section 5.

An interesting question is whether the outcome of the two procedures is actually the same. That is, does minimizing (resp. maximizing) a particular label at the level of complete argument-labellings yield the same result as minimizing (resp. maximizing) the label at the level of complete conclusion labellings? We will see that, in general, the answer is "yes", with one notable exception.

We first formally define two functions between argument labellings and conclusion labellings.

Definition 18. *Let P be a logic program and AF_P be its associated argumentation framework. Let $ArgLabs$ be the set of all argument labellings of AF_P and let $ConcLabs$ be the set of all conclusion labellings of P and AF_P.*

- *We define a function* `ArgLab2ConcLab`: *$ArgLabs \rightarrow ConcLabs$ such that for each $ArgLab \in ArgLabs$, it holds that* `ArgLab2ConcLab`$(ArgLab)$ *is the associated conclusion labelling of $ArgLab$.*

- *We define a function* `ConcLab2ArgLab` : *$ConcLabs \rightarrow ArgLabs$ such that for each $ConcLab \in ConcLabs$ and each $A \in Ar_P$ it holds that:*

  - `ConcLab2ArgLab`$(ConcLab)(A)$ = `in` *iff for each $v \in$* `Vul`(A) *it holds that $ConcLab(v)$ =* `out`
  - `ConcLab2ArgLab`$(ConcLab)(A)$ = `out` *iff there exists a $v \in$* `Vul`(A) *such that $ConcLab(v)$ =* `in`
  - `ConcLab2ArgLab`$(ConcLab)(A)$ = `undec` *iff not for all $v \in$* `Vul`(A) *it holds that $ConcLab(v)$ =* `out` *and there is no $v \in$* `Vul`(A) *such that $ConcLab(v)$ =* `in`

Theorem 19. *When restricted to complete argument labellings and complete conclusion labellings, the functions* `ArgLab2ConcLab` *and* `ConcLab2ArgLab` *are bijections and each other's inverse.*

We are now ready to provide some of the key results of the current paper.

Theorem 20. *Let $ConcLab$ be a conclusion labelling of logic program P and associated argumentation framework $AF_P = (Ar, att)$. It holds that $ConcLab$ is a preferred conclusion labelling iff it is a regular conclusion labelling.*

Theorem 21. *Let ConcLab be a conclusion labelling of logic program P and associated argumentation framework $AF_P = (Ar, att)$. It holds that ConcLab is the grounded conclusion labelling iff it is the well-founded conclusion labelling.*

Theorem 22. *Let ConcLab be a conclusion labelling of logic program P and associated argumentation framework $AF_P = (Ar, att)$. It holds that ConcLab is an argstable conclusion labelling iff it is a concstable conclusion labelling.*

Theorems 20 and 21 rely on lemmas 5 and 6 in the appendix. These state that increasing the occurrences of the `in`-label (resp. `out`-label) on the argument level coincides with increasing the occurrences of the `in`-label (resp. the `out`-label) on the conclusion level. Theorem 22 is proved by observing that the absence of the `undec`-label on the argument level coincides with the absence of the `undec`-label on the conclusion level. We refer to the appendix for details.

Now that the correspondances between preferred and regular, between grounded and well-founded and between argstable and concstable have been observed, we ask ourselves whether a similar correspondance also exists between semi-stable and L-stable. That is, are semi-stable conclusion labellings the same as L-stable conclusion labellings? The answer turns out to be negative. As a counter example, consider the following program.

Example 4. *Consider the following logic program P:*

$$\begin{array}{llll} r_0: & x \leftarrow \texttt{not}\ x & r_1: & c \leftarrow \texttt{not}\ c, \texttt{not}\ x \\ r_2: & a \leftarrow \texttt{not}\ b & r_3: & b \leftarrow \texttt{not}\ a \\ r_4: & c \leftarrow \texttt{not}\ c, \texttt{not}\ a & r_5: & g \leftarrow \texttt{not}\ g, \texttt{not}\ b \end{array}$$

When building the arguments from P we have that each rule r_i corresponds to an argument. That is we have the following arguments, $A_0 = r_0, A_1 = r_1, A_2 = r_2, A_3 = r_3, A_4 = r_4,$ and $A_5 = r_5$. In the table below we give the conclusions and vulnerabilities of the arguments.

	A_0	A_1	A_2	A_3	A_4	A_5
`Conc`(.)	x	c	a	b	c	g
`Vul`(.)	$\{x\}$	$\{c, x\}$	$\{b\}$	$\{a\}$	$\{c, a\}$	$\{g, b\}$

The associated argumentation framework AF_P of P is shown in Figure 4.

The complete argument labellings and the associated complete conclusion labellings of AF_P are given side by side in the following table.

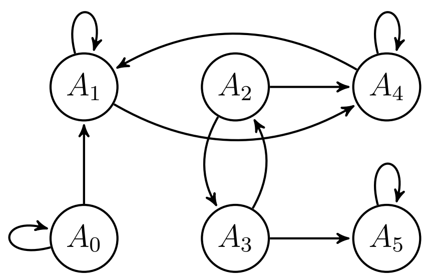


Figure 4: The argumentation framework AF_P associated with P.

complete argument labellings	*complete conclusion labellings*
$ArgLab_1\colon (\emptyset, \emptyset, \{A_1, A_2, A_3, A_4, A_5\})$	$ConcLab_1\colon (\emptyset, \emptyset, \{a, b, c, g\})$
$ArgLab_2\colon (\{A_2\}, \{A_3, A_4\}, \{A_1, A_5\})$	$ConcLab_2\colon (\{a\}, \{b\}, \{c, g\})$
$ArgLab_3\colon (\{A_3\}, \{A_2, A_5\}, \{A_1, A_4\})$	$ConcLab_3\colon (\{b\}, \{a, g\}, \{c\})$

$ArgLab_2$ and $ArgLab_3$ are semi-stable argument labellings. Hence, the associated conclusion labellings $ConcLab_2$ and $ConcLab_3$ are semi-stable conclusion labellings. However, because $\texttt{undec}(ConcLab_2)$ *is not minimal (because* $\texttt{undec}(ConcLab_3) \subsetneq \texttt{undec}(ConcLab_2)$*), we find that $ConcLab_2$ is not L-stable. So here we have an example of a logic program where the semi-stable and L-stable conclusion labellings do not coincide.*

An attentive reader can blame the literal $\texttt{not}\ a$ in the body of $r_4 \in P$ for disrupting the equivalence between these two semantics. Indeed, considering L-stable semantics, one can argue that this $\texttt{not}\ a$ is superfluous as it can be eliminated from r_4 without changing any L-stable conclusion labelling of P. Moreover, in the resulting program, semi-stable and L-stable conclusion labellings coincide with each other. The question is if we can always obtain such an equivalence after eliminating these superfluous literals. The answer is negative, as we show with an example:

Example 5. *Consider the following logic program P_2:*

$$\begin{array}{llll}
r_1: & c \leftarrow \texttt{not}\ c, \texttt{not}\ d & r_2: & a \leftarrow \texttt{not}\ b \\
r_3: & b \leftarrow \texttt{not}\ a & r_4: & c \leftarrow \texttt{not}\ c, \texttt{not}\ a \\
r_5: & g \leftarrow \texttt{not}\ g, \texttt{not}\ b & r_6: & d \leftarrow \texttt{not}\ e \\
r_7: & e \leftarrow \texttt{not}\ d & &
\end{array}$$

Note that from the L-stable semantics point of view, there is no superfluous literal in P_2 as the elimination of any literal from any body of any rule in P_2 will change at least one of its L-stable conclusion labellings.

As before, one can then build the arguments $A_1, \dots, A_7$ corresponding to the rules in P_2. Below we give the conclusions and vulnerabilities.

	A_1	A_2	A_3	A_4	A_5	A_6	A_7
$\texttt{Conc}(.)$	c	a	b	c	g	d	e
$\texttt{Vul}(.)$	$\{c,d\}$	$\{b\}$	$\{a\}$	$\{a,c\}$	$\{b,g\}$	$\{e\}$	$\{d\}$

The associated argumentation framework AF_{P_2} of P_2 is shown in Figure 5.

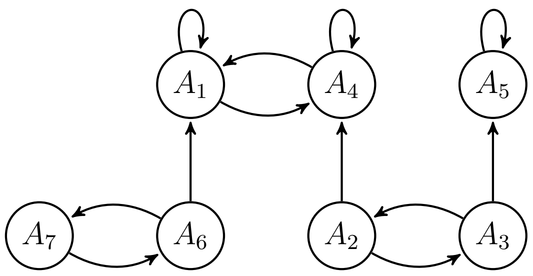


Figure 5: The argumentation framework AF_{P_2} associated with P_2.

One can check that the complete argument labellings and the associated complete conclusion labellings of AF_{P_2} are:

complete argument labellings	*complete conclusion labellings*
$ArgLab_1\colon (\emptyset, \emptyset, \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\})$	$ConcLab_1\colon (\emptyset, \emptyset, \{a, b, c, d, e, g\})$
$ArgLab_2\colon (\{A_2\}, \{A_3, A_4\}, \{A_1, A_5, A_6, A_7\})$	$ConcLab_2\colon (\{a\}, \{b\}, \{c, d, e, g\})$
$ArgLab_3\colon (\{A_3\}, \{A_2, A_5\}, \{A_1, A_4, A_6, A_7\})$	$ConcLab_3\colon (\{b\}, \{a, g\}, \{c, d, e\})$
$ArgLab_4\colon (\{A_6\}, \{A_1, A_7\}, \{A_2, A_3, A_4, A_5\})$	$ConcLab_4\colon (\{d\}, \{e\}, \{a, b, c, g\})$
$ArgLab_5\colon (\{A_7\}, \{A_6\}, \{A_1, A_2, A_3, A_4, A_5\})$	$ConcLab_5\colon (\{e\}, \{d\}, \{a, b, c, g\})$
$ArgLab_6\colon (\{A_2, A_6\}, \{A_1, A_3, A_4, A_7\}, \{A_5\})$	$ConcLab_6\colon (\{a, d\}, \{b, c, e\}, \{g\})$
$ArgLab_7\colon (\{A_2, A_7\}, \{A_3, A_4, A_6\}, \{A_1, A_5\})$	$ConcLab_7\colon (\{a, e\}, \{b, d\}, \{c, g\})$
$ArgLab_8\colon (\{A_3, A_6\}, \{A_1, A_2, A_5, A_7\}, \{A_4\})$	$ConcLab_8\colon (\{b, d\}, \{a, e, g\}, \{c\})$
$ArgLab_9\colon (\{A_3, A_7\}, \{A_2, A_5, A_6\}, \{A_1, A_4\})$	$ConcLab_9\colon (\{b, e\}, \{a, d, g\}, \{c\})$

$ArgLab_6$ and $ArgLab_8$ are semi-stable argument labellings. Also, the associated conclusion labellings $ConcLab_6$ and $ConcLab_8$ are semi-stable con-

clusion labellings. In addition, ConcLab$_9$ *is also L-stable, but it is not semi-stable as* `undec`$(ArgLab_9)$ *is not minimal. Consequently, the absence of superfluous literals in a logic program is not a sufficient condition to guarantee the equivalence between semi-stable and L-stable conclusion labellings.*

We summarize our results concerning the comparison of argument-based conclusion labellings and logic programming-based conclusion labellings in Table 3.

Argument-Based Conclusion Labelling	Relation	Logic Programming-Based Conclusion Labelling
Preferred	$\equiv$	Regular
Grounded	$\equiv$	Well-Founded
Semi-stable	$\not\equiv$	L-stable
Argstable	$\equiv$	Concstable

Table 3: Comparison of argument-based and LP-based conclusion labellings of a program

7. On the Connection between Argument Extensions and Logic Programming Models

So far, we have examined the general question of how argument labellings are related to conclusion labellings. We found that:

- maximizing `in` at the argument level yields the same result as maximizing `in` at the conclusion level
- minimizing `in` at the argument level yields the same result as minimizing `in` at the conclusion level
- maximizing `out` at the argument level yields the same result as maximizing `out` at the conclusion level
- minimizing `out` at the argument level yields the same result as minimizing `out` at the conclusion level
- maximizing `undec` at the argument level yields the same result as maximizing `undec` at the conclusion level
- minimizing `undec` at the argument level does *not* yield the same result as minimizing `undec` at the conclusion level

Cond. on Comp. Arg. Lab.	Name of Ass. Conc. Lab.	Result	Name of Conc. Lab.	Cond. on Comp. Conc. Lab.
None	Complete	[4]	Complete	None
Max. IN	Preferred (Th12)	Th20	Regular (Th15)	Max. IN
Max. OUT	Preferred (Th12)	Th20	Regular (Th15)	Max. OUT
Max. UNDEC	Grd. (Ths 12,14)	Th21	WF (Ths 15,17)	Max. UNDEC
Min. IN	Grd. (Ths 12,14)	Th21	WF (Ths 15,17)	Min. IN
Min. OUT	Grd. (Ths 12,14)	Th21	WF (Ths 15,17)	Min. OUT
Min. UNDEC	Semi-stable	Ex4	L-stable	Min. UNDEC
Empty UNDEC	Argstable	Th22	Concstable	Empty UNDEC

Table 4: Equivalence and Non-equivalence results on logic programming semantics and argumentation semantics

These results are summarized in Table 4, that should be read as follows. The left-hand side of the table is related to the process of maximizing/minimizing a particular labelling at the argument-level, and then generating the associated labellings at the conclusion level (as was outlined in Section 4). Here, we have that selecting the complete argument labellings with maximal `in` is the same as selecting the complete argument labellings with maximal `out` (Th12). These selected argument labellings are called the *preferred argument labellings*, just like their associated conclusion labellings are called the *preferred conclusion labellings.* We also have that selecting the complete argument labellings with maximal `undec` is the same as selecting the complete argument labellings with minimal `in`, and the same as selecting the complete argument labellings with minimal `out` (Ths 12, 14). This unique (Th13) selected argument labelling is called the *grounded argument labelling*, just like its associated conclusion labelling is called the *grounded conclusion labelling* (abbreviated "Grd." in the table).

The right-hand side of the table is related to the process of first generating *all* complete argument labellings and associated complete conclusion labellings, and then to maximize/minimize a particular label at the conclusion level (as was outlined in Section 5). Here, we have that selecting the complete conclusion labellings with maximal `in` is the same as selecting the complete conclusion labellings with maximal `out` (Th20). These selected conclusion labellings are called the *regular conclusion labellings.* We also have that selecting the complete conclusion labellings with maximal `undec` is the same as selecting the complete conclusion labellings with minimal `in`, and the same as selecting the complete conclusion labellings with minimal `out`. This unique (Th16) selected labelling is called the *well-founded conclusion* *labelling* (abbreviated as “WF” in the table).

In the middle column of the table, the connection between these two approaches is indicated. From Th20 it follows that the preferred conclusion labellings are precisely the same as the regular conclusion labellings. From Th21 it follows that the grounded conclusion labelling is precisely the same as the well-founded conclusion labelling. From Th22 it follows that the argstable conclusion labellings are precisely the same as the concstable conclusion labellings. Example 4, however, makes clear that in general, the semi-stable conclusion labellings are *not* the same as the L-stable conclusion labellings. Minimizing `undec` at the argument-level yields fundamentally different results as minimizing `undec` at the conclusion level.

In the current paper, we have proved these results for an instantiation based on logic programs. However, similar results can be obtained also for other forms of instantiated argumentation. In order to apply the proofs that were specified in sections 4, 5 and 6, all that matters is that attack is defined on the conclusion of the attacking argument and the set of vulnerabilities (`Vul`) of the attacked argument. This makes our results directly applicable also to formalisms such as ABA [9] and ASPIC [8, 22].[7]

7.1. Relating Conclusion Labellings to Models of Programs

In the current paper, we have chosen an instantiation based on logic programming partly because of its relative simplicity, but also because it allows us to study an additional research question: how are (traditional) approaches to argumentation semantics related to (traditional) approaches to logic programming semantics?

Caminada and Gabbay [6] specify two functions to convert an argument extension to an argument labelling and vice versa. The function `Lab2Ext` converts an argument labelling to an argument extension (in essence, a set of arguments) and is defined as $\texttt{Lab2Ext}(\mathit{ArgLab}) = \texttt{in}(\mathit{ArgLab})$. The function `Ext2Lab` converts a conflict-free set of arguments (for instance, an argument extension) to an argument labelling and is defined as $\texttt{Ext2Lab}(\mathcal{A}rgs) = (\mathcal{A}rgs, \mathcal{A}rgs^{+}, Ar \setminus (\mathcal{A}rgs \cup \mathcal{A}rgs^{+}))$.[8] When `Ext2Lab` and `Lab2Ext` are restricted to operate on complete extensions and complete labellings, they become bijective functions that are each other’s inverse [6]. This means that

[7]In ASPIC, the set of vulnerabilities of an argument consists of the statements on which it can be rebutted or undercut.

[8]We recall that (Definition 2) $\mathcal{A}rgs^{+} = \{A | A$ is attacked by an argument in $\mathcal{A}rgs\}$.

complete extensions and complete labellings are one-to-one related, just like preferred extensions and preferred labellings, the grounded extension and the grounded labelling, semi-stable extensions and semi-stable labellings, and stable extensions and stable labellings.

Now that we have observed that the traditional approaches to argumentation semantics (argument extensions) coincide with the approach of argument labellings, the next step is to show that the traditional approaches to logic programming semantics (models based on fixpoints of a reduced program) coincide with the approach of conclusion labellings.

We now introduce two functions `ConcLab2Mod` and `Mod2ConcLab` to convert a conclusion labelling to a model and vice versa. The first function `ConcLab2Mod` converts a conclusion labelling to a model and is defined as $\texttt{ConcLab2Mod}(\mathit{ConcLab}) = (\texttt{in}(\mathit{ConcLab}), \texttt{out}(\mathit{ConcLab}))$. The second function `Mod2ConcLab` converts a model to a conclusion labelling and is defined as $\texttt{Mod2ConcLab}((T, F)) = (T, F, \mathit{HB}_P \setminus (T \cup F))$. It is not difficult to see that `ConcLab2Mod` and `Mod2ConcLab` are bijective functions that are each other's inverse, making conclusion labellings and models one-to-one related.

The next step is to show that P-stable models coincide with complete conclusion labellings. This actually follows from the foundational work of Wu et al. [4]. Here it is proved that if one applies complete semantics at step 2 of the argumentation process, one obtains the P-stable models of the original logic program. From this, together with the results in the current paper, it directly follows that complete conclusion labellings are one-to-one related to P-stable models.

From the correspondence between complete conclusion labellings and P-stable models, the other correspondences between conclusion labellings and models follow. Since a regular model is a P-stable model with maximal T, and a regular conclusion labelling is a complete conclusion labelling with maximal `in`, it follows that they correspond to each other (through the functions `ConcLab2Mod` and `Mod2ConcLab`). Similar correspondences can be observed between the well-founded model and the well-founded conclusion labelling, between L-stable models and the L-stable conclusion labellings and between stable models and concstable conclusion labellings. That is, the various types of logic programming models are actually different types of conclusion labellings.

7.2. Abstract Argumentation & Logic Programming Semantics Equivalences

We have now arrived at the main point of the current paper: the connection between (traditional) approaches to argumentation semantics and (traditional) approaches to logic programming semantics. Let us again look at the 3-step process of Section 3. Assume that step 1 (AF construction) and step 3 (converting argument labellings to conclusion labellings) are fixed, and that the only degree of freedom is which semantics to apply at step 2. From the results in the current paper, it follows that

- if one applies complete semantics at step 2, the overall outcome is equivalent to calculating the P-stable models of the original logic program (See [4])
- if one applies preferred semantics at step 2, the overall outcome is equivalent to applying regular semantics to the original logic program (Theorem 20)
- if one applies grounded semantics at step 2, the overall outcome is equivalent to applying well-founded semantics to the original logic program (Theorem 21)
- if one applies stable semantics at step 2, the overall outcome is equivalent to applying stable model semantics to the original logic program (Theorem 22)

That is, differences in logic programming semantics can be reduced purely to differences in what happens at the abstract argumentation level (step 2). In essence, partial stable model semantics coincides with complete semantics, preferred semantics coincides with regular semantics, grounded semantics coincides with well-founded semantics and stable semantics coincides with stable model semantics.

Moreover, we are also able to explain *why* these semantics coincide. Recall that the various argumentation semantics that are studied in the current paper are based on minimization or maximization (of a particular label) at the *argument* level, whereas the various logic programming semantics turn out to be based on minimization and maximization (of a particular label) at the *conclusion* level. The fact that argumentation semantics coincide with logic programming semantics is due to the fact that what happens at the argument level tends to coincide with what happens at the conclusion level.

The fact that preferred semantics coincides with regular semantics is *because* maximizing `in` at the argument level is the same as maximizing `in` at the conclusion level. Similarly, the fact that grounded semantics coincides with well-founded semantics is because minimizing `in` at the argument level is the same as minimizing `in` at the conclusion level. Also, the fact that stable semantics coincides with stable model semantics is because ruling out `undec` at the argument level is the same as ruling out `undec` at the conclusion level. Finally, the fact that semi-stable semantics does *not* coincide with L-stable model semantics is because minimizing `undec` at the argument level is something really different from minimizing `undec` at the conclusion level.

8. Semi-Stable Semantics versus L-Stable Semantics Revisited

As we have seen, since minimizing `undec` at the argument-level does not yield the same result as minimizing `undec` at the conclusion level, semi-stable semantics does not coincide with L-stable semantics. The underlying reason is that although an increased occurrence of the `in`-label on the argument level coincides with an increased occurrence of the `in`-label on the conclusion level (Lemma 5 in the appendix) and an increased occurrence of the `out`-label on the argument level coincides with an increased occurrence of the `out`-label on the conclusion level (Lemma 6 in the appendix), a similar result does not hold for the `undec`-label. That is, it is *not* the case that an increased occurrence of the `undec`-label on the argument level coincides with an increased occurrence of the `undec`-label on the conclusion level (as illustrated by Example 4 and Example 5).

In the current section, we study the discrepancy between semi-stable and L-stable in more detail. In particular, we are interested in the following two questions:

1. is there any abstract argumentation semantics at all that can generate results that are equivalent to L-stable conclusion labellings?
2. is there a class of restricted logic programs for which minimizing `undec` at the argument level yields the same result as minimizing `undec` at the conclusion level?

As for the first question, what we are looking for is an abstract argumentation semantics that can be applied at step 2 of the argumentation process. We assume steps 1 and 3 to remain the same. That is, we need an abstract argumentation semantics that is able to generate the L-stable conclusion labellings, just like preferred semantics is able to generate the regular conclusion labellings and grounded semantics is able to generate the well-founded conclusion labelling. Therefore, the selection of the arguments in such a semantics should be based purely on the *structure* of the graph, and not on the particular contents of the arguments. This can be warranted by requiring that the semantics satisfies the *language independence* principle [23].

Definition 23. *We say that an abstract argumentation semantics X is L-stable generating iff it is a function such that*

1. *For any logic program P, X takes as input AF_P and yields as output a set of argument labellings $ArgLabs$*
2. *X satisfies language independence, that is, if for any pair of argumentation frameworks AF_1, AF_2, if AF_1 is isomorphic to AF_2 by a mapping M of their arguments (the nodes in the graphs), then each labelling of AF_1 can be mapped to a (different) labelling of AF_2 by that same mapping M.*
3. *It holds that $\{\texttt{ArgLab2ConcLab}(ArgLab) \mid ArgLab \in ArgLabs\}$ is precisely the same as the set of all L-stable conclusion labellings of AF_P.*

Theorem 24. *There exists no abstract argumentation semantics that is L-stable generating.*

Proof. Consider the following two logic programs P with rules $r_0, ..., r_4$ and P' with rules $r'_0, ..., r'_4$:

$$\begin{array}{ll|ll}
r_0: & x \leftarrow \texttt{not}\ x & r'_0: & x \leftarrow \texttt{not}\ x \\
r_1: & c \leftarrow \texttt{not}\ x & r'_1: & d \leftarrow \texttt{not}\ x \\
r_2: & a \leftarrow \texttt{not}\ b & r'_2: & a \leftarrow \texttt{not}\ b \\
r_3: & b \leftarrow \texttt{not}\ a & r'_3: & b \leftarrow \texttt{not}\ a \\
r_4: & c \leftarrow \texttt{not}\ a, \texttt{not}\ c & r'_4: & c \leftarrow \texttt{not}\ a, \texttt{not}\ c, \texttt{not}\ d
\end{array}$$

For the above programs, please observe that:

- P has three P-stable models: $S_1 = \langle\{\ \}, \{\ \}\rangle$, $S_2 = \langle\{a\}, \{b\}\rangle$, $S_3 = \langle\{b\}, \{a\}\rangle$, where S_2, S_3 are L-stable models.

- P' has three P-stable models: $S_1 = \langle\{\ \}, \{\ \}\rangle$, $S_2 = \langle\{a\}, \{b, c\}\rangle$, $S_3 = \langle\{b\}, \{a\}\rangle$, where S_2 is the single L-stable model.

The arguments $A_1, ..., A_4$ built from P and $A'_1, ..., A'_4$ built from P' are listed below.

A_0 :	$x \leftarrow \texttt{not}\ x$	$A_{0'}$:	$x \leftarrow \texttt{not}\ x$
A_1 :	$c \leftarrow \texttt{not}\ x$	$A_{1'}$:	$d \leftarrow \texttt{not}\ x$
A_2 :	$a \leftarrow \texttt{not}\ b$	$A_{2'}$:	$a \leftarrow \texttt{not}\ b$
A_3 :	$b \leftarrow \texttt{not}\ a$	$A_{3'}$:	$b \leftarrow \texttt{not}\ a$
A_4 :	$c \leftarrow \texttt{not}\ a, \texttt{not}\ c$	$A_{4'}$:	$c \leftarrow \texttt{not}\ a, \texttt{not}\ c, \texttt{not}\ d$

The argumentation frameworks of P and P' are depicted in Figure 6.

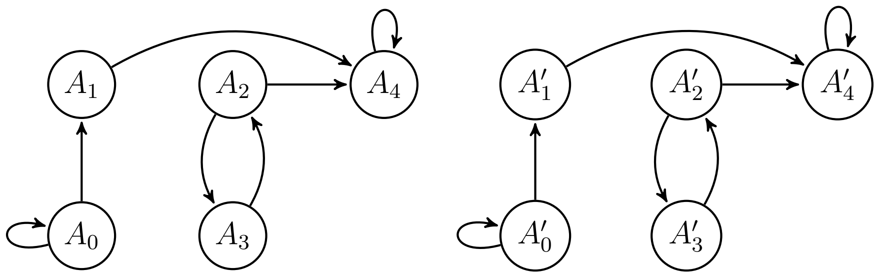


Figure 6: The argumentation frameworks associated with P and P'.

Since P has two L-stable models while P' has only one, the L-stable semantics is sensitive to the difference between them. However, given that these programs have isomorphic associated graphs, they are indiscernible in the perspective of abstract argumentation semantics: By the language independence principle, both argumentation frameworks have the same extensions under the L-stable generating semantics. But this is incompatible with the requirement that the argumentation frameworks should yield *different* L-stable models. Contradiction. As a consequence, we conclude that no semantics of abstract argumentation can coincide with the L-stable semantics for every program. $\square$

Now that we have observed that in general no abstract argumentation semantics is able to coincide with L-stable model semantics, the next question is whether one can define a restricted class of logic programs for which semi-stable semantics does coincide with L-stable semantics.

Definition 25. *Let P be a logic program. We say that P is* semi-stable-L-stable compatible *iff it holds that* $\{\texttt{ArgLab2ConcLab}(\mathit{ArgLab}) \mid \mathit{ArgLab}$ *is a semi-stable argument labelling of* $AF_P\} = \{\mathit{ConcLab} \mid \mathit{ConcLab}$ *is a L-stable conclusion labelling of P and* $AF_P\}$.

We proceed to provide three classes of logic programs that are semi-stable-L-stable compatible: Stable programs, AF-programs, and stratified programs.

Definition 26. *(Stable program) A logic program P is a stable program if it has at least one stable model.*

Theorem 27. *Every stable program is semi-stable-L-stable compatible.*

Definition 28. *(AF-program)*

A logic program P is an AF-program *if there is at most one rule with $head(r) = c$, for each $c \in HB_P$.*

Theorem 29. *Every AF-program is semi-stable-L-stable compatible.*

Definition 30. *(stratified program) A logic program P is* stratified *if it is possible to attribute a positive integer $l(p)$ to each atom $p \in HB_P$ in a way that, for each rule r in P with $head(r) = p$, it holds: (i) for each $q \in body^-(r)$, $l(q) < l(p)$; and (ii) for each $q \in body^+(r)$, $l(q) \leq l(p)$.*

Theorem 31. *Every stratified program is semi-stable-L-stable compatible.*

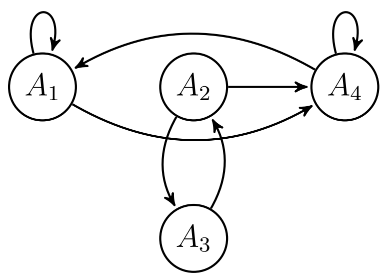
9. Translating Argumentation Frameworks to Logic Programs

So far, when making the connection between argumentation semantics and logic programming semantics, we have used a translation *from* logic programs *to* argumentation frameworks. In the current section, we go the other way around. That is, we still examine the connection between argumentation semantics and logic programming semantics, but this time using a translation *from* argumentation frameworks *to* logic programs.

In essence, each argument generates an associated logic programming rule, with the name of the argument in its head, and the name of its attackers in the weak part of the body.

Definition 32 ([4])**.** *Let $AF = (Ar, att)$ be an argumentation framework. Then $P_{AF} = \{A \leftarrow \texttt{not}\ B_1, \ldots, \texttt{not}\ B_m \mid A, B_1, \ldots, B_m \in Ar (m \geq 0)$ and $\{B_i \mid (B_i, A) \in att\} = \{B_1, \ldots, B_m\}\}$ is the logic program associated to AF.*

Example 6. *Consider AF, the argumentation framework below:*



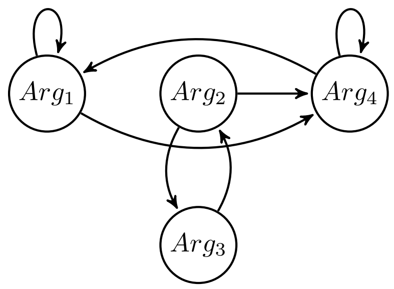
This is the argumentation framework on the left hand side of Figure 6, copied here for easier reference. Its associated logic program P_{AF} is:[9]

$$\begin{array}{llll} r_1: & A_1 \leftarrow \texttt{not}\ A_1, \texttt{not}\ A_4 & r_2: & A_2 \leftarrow \texttt{not}\ A_3 \\ r_3: & A_3 \leftarrow \texttt{not}\ A_2 & r_4: & A_4 \leftarrow \texttt{not}\ A_4, \texttt{not}\ A_1, \texttt{not}\ A_2 \end{array}$$

The first thing to be observed is that for any argumentation framework AF, the associated logic program P_{AF} is an AF-program (Definition 28).

Now let us examine what happens if we translate P_{AF} back to argumentation theory. That is, what does $AF_{P_{AF}}$ look like? Since each rule in P_{AF} has only a weak part (no strong atoms in the body), each argument of $AF_{P_{AF}}$ consists of precisely one rule (Definition 7) and the attack relation of $AF_{P_{AF}}$ coincides with the original attack relation of AF.

Example 7. *(Example 6 continued) The argumentation framework $AF_{P_{AF}}$ associated with P_{AF} from Example 6 is:*



[9]Please notice that we use the names of the arguments as atoms in the associated logic program. Doing so brings no prejudice to our original definitions on logic programs. Instead of using the names of arguments, we could use atoms such as $a_1, a_2, ..., a_4$ or a, b, c, d in order to build an alike program. In that setting, each atom would be represent one of the arguments.

Here, each argument Arg_i consists of only the rule r_i from P_{AF}.

Theorem 33. *Let AF be an argumentation framework, P_{AF} be the associated logic program (Definition 32), and $AF_{P_{AF}}$ be the argumentation framework that is associated with this logic program (Definition 9). It holds that AF and $AF_{P_{AF}}$ are isomorphic.*

It has been observed that complete semantics, as well as the preferred semantics, grounded semantics, semi-stable semantics and stable semantics all satisfy the language independence principle [23]. This implies that the complete, preferred, grounded, semi-stable and stable labellings of $AF_{P_{AF}}$ are essentially the same (modulo isomorphism) as the complete, preferred, grounded, semi-stable and stable labellings of AF.

What does this mean for the connection between argumentation semantics and logic programming semantics, from the perspective of translating argumentation frameworks to logic programs? First of all, we observe that for the translation of P_{AF} to $AF_{P_{AF}}$ the earlier observed results hold. That is, applying complete semantics to $AF_{P_{AF}}$ yields the same results as applying partial stable model semantics to P_{AF}, applying preferred semantics to $AF_{P_{AF}}$ yields the same results as applying regular semantics P_{AF}, applying grounded semantics to $AF_{P_{AF}}$ yields the same results as applying well-founded semantics to P_{AF}, applying stable semantics to $AF_{P_{AF}}$ yields the same results as applying stable model semantics to P_{AF}, and (due to the fact that P_{AF} is an AF-program) applying semi-stable semantics to $AF_{P_{AF}}$ yields the same results as applying L-stable semantics to P_{AF}.

From the facts that all of the above mentioned argumentation semantics satisfy the language independence principle, and that AF is isomorphic with $AF_{P_{AF}}$, it then follows that applying complete semantics to AF yields the same results as applying the P-stable models of P_{AF}, applying preferred semantics to AF yields the same results as applying regular semantics to P_{AF}, applying grounded semantics to AF yields the same results as applying well-founded semantics to P_{AF}, applying semi-stable semantics to AF yields the same results as applying L-stable model semantics to P_{AF} and applying stable semantics to AF yields the same results as applying stable model semantics to P_{AF}. These results have been summarized in Table 5.

Hence, we see that, when translating argumentation frameworks to logic programs, the connection between argumentation semantics and logic programming semantics is stronger than when translating (unrestricted) logic

Abstract argumentation semantics on AF	Relation	Logic Programming semantics on P_{AF}
Complete	$\equiv$	P-stable
Preferred	$\equiv$	Regular
Grounded	$\equiv$	Well-Founded
Semi-stable	$\equiv$	L-stable
Stable	$\equiv$	Stable

Table 5: The relations between semantics over AF-programs.

programs to argumentation frameworks. Whereas for the latter translation, semi-stable and L-stable do not coincide, for the former translation they do.

Osorio et al. [24] exploit the connection between argumentation semantics and logic programming semantics by resorting to an alternative translation from argumentation frameworks to logic programs. Indeed, they have characterized the complete extensions of an AF in terms of Clark's completion models [25] of the corresponding logic program. They also argue that their translation only requires classical logic where very efficient software exists. When compared to the proposal presented in this section, we would say that our translation is simpler and fairly standard in the literature and that has for instance been applied by Wu et al. [4]. Overall, both translations can be employed to establish the connections displayed in Table 5.

10. Discussion

In this paper we studied various connections amongst abstract argumentation semantics and logic programming semantics. We started by providing definitions of these in a way that already suggests their connections. After these definitions, we then started to look deeper into the reasons why the semantics coincidence. We re-examined how the standard three-step process of instantiated argumentation can be applied in the context of logic programming. We observed that the various abstract argumentation semantics (grounded, preferred, stable and semi-stable) are based on complete semantics, and essentially select among the complete labellings those where the set of arguments with a particular label (`in`, `out` or `undec`) is maximal or minimal. We also observed that the various logic programming semantics (well-founded, regular, stable and L-stable) are based on partial stable model semantics, and essentially select among the partial stable models those where a particular truth value (T, F or U) is maximal or minimal.

Using the already established connection between complete semantics in argumentation and partial stable model semantics in logic programming [4], we were then able to explain one of the essential differences between argumentation and logic programming: where the various argumentation semantics essentially maximize and minimize a particular label at the *argument level*, the various logic programming semantics essentially maximize and minimize a particular label at the *conclusion level*.

In order to establish correspondences between the various argumentation semantics and logic programming semantics, we then needed to examine whether maximizing or minimizing a particular label on the argument level has the same effect as maximizing or minimizing the label on the conclusion level. We found that, with one notable exception, this indeed turns out to be the case, thus obtaining correspondence between grounded and well-founded semantics, between preferred and regular semantics and between argstable and concstable semantics.

We also found that in the general case, the semi-stable semantics for argumentation does not coincide with L-stable semantics for logic programming (although three special classes of logic programs were identified for which correspondence does hold). This is caused by the fact that, in general, minimizing `undec` on the argument level does not yield the same results as minimizing `undec` on the conclusion level. We also observed that there exists no abstract argumentation semantics that is able to yield L-stable model semantics.[10]

We observed that when going the other way around (translating argumentation frameworks to logic programs) the correspondence between logic programming and argumentation is even stronger. This is because the translation from argumentation to logic programming yields a special class of logic programs (AF-programs) for which correspondence between semi-stable and L-stable does hold, in addition to the other argumentation and LP correspondences.

The obtained correspondences between logic programming and argumentation open up opportunities for using techniques from one field to the other

[10]At least when applying the standard way of using a logic program to construct an argumentation framework (step 1) and applying the standard way of mapping an argument labelling to a conclusion labelling (step 3).

field. For instance, it becomes possible to apply the dialectical proof procedures for formal argumentation (like the Standard Grounded Game [26], the Grounded Persuasion Game [27, 28] and the Admissibility Game [29]) in the context of logic programming, to determine the status of a single atom without having to construct an entire logic programming model. Similarly, one could use recent developments for computing the preferred labellings of an argumentation framework [30] to determine the regular models of a normal logic program. Notice that we can do so because we are able to translate (unrestricted) normal logic programs to instantiated argumentation. In this way, we go beyond the work of for instance Osorio et al. [24], whose translation only goes the other way around (from argumentation to logic programming) whereas our approach is able to go both ways.

Logic programming has long served as an inspiration for argumentation theory. The approach of Prakken and Sartor [1] for instance, applies extended logic programming clauses to construct arguments, which are then evaluated using grounded semantics. Defeasible Logic Programming (DeLP) [31] is also based on logic programming clauses, but unlike the approach of Prakken and Sartor, it does not come with a declarative semantics. Instead, entailment is defined in terms of dialectical proof procedures. Moreover, DeLP does not seem to support the notion of abstraction when it comes to determining the accepted arguments. Whereas formalisms like ASPIC [7, 8, 22], ABA [9] and logic-based argumentation [18] are able to construct a graph and subsequently select the set(s) of accepted arguments (that is, to apply argumentation semantics) without needing to look at the internal structure of the arguments, DeLP *does* need to have access to the internal structure of the arguments when applying the dialectical proof procedures that determine which arguments to accept. This makes it difficult to compare DeLP with Dung-style abstract argumentation, because in DeLP there is no such thing as a purely abstract level.

Although the current paper studied the connection between existing logic programming semantics and existing argumentation semantics, the results are also relevant for other formalisms. Using the recently observed connection between logic programming and Assumption-Based Argumentation (ABA) [32] one can directly apply our results to ABA.[11] It then directly follows

[11] Although the work of Schulz and Toni [32] translates logic programs to ABA theories, it is fairly straightforward also to go the other way around, that is, to translate ABA theories

that the original assumption-based ABA semantics [35] are in fact based on maximizing and minimizing particular labels on the conclusion level, whereas the ABA-AA interpretation (ABA as an instantiation of Dung-style abstract argumentation, [9]) is based on maximizing and minimizing particular labels on the argument level. As we have seen, these are not always the same. Thus, our theory helps to explain the previously observed difference between ABA assumption-based semi-stable semantics and ABA-AA semi-stable semantics [33, 34], as the former is equivalent to L-stable semantics through the translation between ABA and logic programming.

Apart from ABA, our findings are also relevant for the ASPIC formalism [8, 22], as it is possible to translate an ASPIC theory to a logic program. ASPIC-style undercutting could be modelled by including rule-specific weakly negated "*ab*" atoms (similarly to what one often observes in Circumscription). Strong negation and ASPIC-style rebutting could be modelled by introducing additional atoms and generating "semi-normal" logic programming rules (similar to semi-normal defaults [36]) where the head of the rule also occurs strongly negated in the weak part of the body. As an example of how such a translation would work, consider the following ASPIC theory: $R_s = \{\rightarrow a;\ \rightarrow b;\ \rightarrow c\}$ and $R_d = \{r_1 : a \Rightarrow d;\ r_2 : b \Rightarrow \neg d;\ r_3 : c \Rightarrow \neg r_1\}$ (with $\neg r_1$ expressing that the rule r_1 is undercut). This would generate the following logic program: $P = \{a \leftarrow;\ b \leftarrow;\ c \leftarrow;\ a \leftarrow a, \texttt{not}\ ab_1, \texttt{not}\ nd;\ nd \leftarrow b, \texttt{not}\ ab_2, \texttt{not}\ d;\ ab_1 \leftarrow c, \texttt{not}\ ab_3\}$ (notice that nd stands for the negation of d). Hence, we see that although ASPIC is conceptually richer than logic programming, it is still perfectly possible to model concepts like strong negation, rebutting and undercutting by means of logic programming, and hence to translate an ASPIC theory to a logic program. Moreover, the translation is such that the arguments that can be constructed using the original ASPIC theory coincide with the arguments that can be constructed using the resulting logic program, regarding both their structure and attack relation. Hence, our results on the equivalence between minimization and maximization on the argument level and minimization and maximization on the conclusion level are also relevant in the

to logic programs. This does require, however, that the ABA theory one starts with is "assumption complete" in the sense that every non-assumption atom is the contrary of at least one assumption. This is needed to preserve equivalence between the ABA semantics and the logic programming semantics, in particular between ABA semi-stable [33, 34] and logic programming L-stable.

context of formalisms like ASPIC.

More generally, although the current paper focuses mainly on instantiated argumentation based on logic programming, its main findings are in fact relevant for a wide range of instantiated argumentation formalisms (like [8, 37, 18, 38, 9]) as it specifies the possibilities and impossibilities of using the argumentation approach to specify nonmonotonic entailment, or to model existing nonmonotonic formalisms. If the aim is, for instance, to model a formalism that maximizes `in` or `out` at the conclusion level (like for instance is the case in Pollock's 1995 OSCAR system [39]) the argumentation approach will do fine (as evidenced by Jakobovits and Vermeir [40]). However, if the aim is to model a formalism that minimizes `undec` at the conclusion level, the argumentation approach will be of limited help (Theorem 24). Hence, the current paper has shed some light on the strengths and limitations of using the argumentation approach for specifying nonmonotonic entailment.

Acknowledgements

This work has been supported by the National Research Fund, Luxembourg (LAAMI project), by the Engineering and Physical Sciences Research Council (EPSRC, UK), grant ref. EP/J012084/1 (SAsSy project), by CNPq (Universal 2012 - Proc. 473110/2012-1), and by CNPq/CAPES (Casadinho/PROCAD 2011).

A. Proofs of Theorems

A.1. Theorems and Proofs from Section 4:

Proposition 1. *Let $AF = (Ar, att)$ be an argumentation framework. An argument labelling ArgLab is a complete argument labelling iff for each $A \in Ar$ it holds that:*

- *if for every $B \in Ar$ that attacks A it holds that $ArgLab(B) =$ `out`, then $ArgLab(A) =$ `in`*
- *if there exists a $B \in Ar$ that attacks A such that $ArgLab(B) =$ `in`, then $ArgLab(A) =$ `out`*
- *if not for every $B \in Ar$ that attacks A it holds that $ArgLab(B) =$ `out` and there does not exist a $B \in Ar$ that attacks A such that $ArgLab(B) =$ `in`, then $ArgLab(A) =$ `undec`*

Lemma 1 ([5, 6])**.** *Let $ArgLab_1$ and $ArgLab_2$ be complete argument labellings of argumentation framework $AF = (Ar, att)$. It holds that*

1. $\texttt{in}(ArgLab_1) \subseteq \texttt{in}(ArgLab_2)$ *iff* $\texttt{out}(ArgLab_1) \subseteq \texttt{out}(ArgLab_2)$
2. $\texttt{in}(ArgLab_1) \subsetneq \texttt{in}(ArgLab_2)$ *iff* $\texttt{out}(ArgLab_1) \subsetneq \texttt{out}(ArgLab_2)$
3. $\texttt{in}(ArgLab_1) = \texttt{in}(ArgLab_2)$ *iff* $\texttt{out}(ArgLab_1) = \texttt{out}(ArgLab_2)$

From Lemma 1 it immediately follows that $\texttt{in}(ArgLab_1) = \texttt{in}(ArgLab_2)$ iff $ArgLab_1 = ArgLab_2$, and that $\texttt{out}(ArgLab_1) = \texttt{out}(ArgLab_2)$ iff $ArgLab_1 = ArgLab_2$. Moreover, from Lemma 1, one can then obtain the following result.

Theorem 12 ([5, 6])**.** *Let ArgLab be a complete argument labelling of argumentation framework $AF = (Ar, att)$. It holds that*

- $\texttt{in}(ArgLab)$ *is maximal (w.r.t. set-inclusion) among all complete argument labellings of AF iff* $\texttt{out}(ArgLab)$ *is maximal (w.r.t. set-inclusion) among all complete argument labellings of AF.*

- $\texttt{in}(ArgLab)$ *is minimal (w.r.t. set-inclusion) among all complete argument labellings of AF iff* $\texttt{out}(ArgLab)$ *is minimal (w.r.t. set-inclusion) among all complete argument labellings of AF.*

Lemma 2 ([5, 6])**.** *Let ArgLab and $ArgLab'$ be complete argument labellings of argumentation framework $AF = (Ar, att)$. It holds that*

1. *if* $\texttt{in}(ArgLab) \subseteq \texttt{in}(ArgLab')$ *then* $\texttt{undec}(ArgLab) \supseteq \texttt{undec}(ArgLab')$
2. *if* $\texttt{in}(ArgLab) \subsetneq \texttt{in}(ArgLab')$ *then* $\texttt{undec}(ArgLab) \supsetneq \texttt{undec}(ArgLab')$
3. *if* $\texttt{out}(ArgLab) \subseteq \texttt{out}(ArgLab')$ *then* $\texttt{undec}(ArgLab) \supseteq \texttt{undec}(ArgLab')$
4. *if* $\texttt{out}(ArgLab) \subsetneq \texttt{out}(ArgLab')$ *then* $\texttt{undec}(ArgLab) \supsetneq \texttt{undec}(ArgLab')$

From Lemma 2, the following result follows.

Theorem 34. *Let ArgLab be a complete argument labelling of argumentation framework $AF = (Ar, att)$. It holds that*

1. *if* $\texttt{undec}(ArgLab)$ *is minimal (w.r.t. set-inclusion) among all complete argument labellings of AF then* $\texttt{in}(ArgLab)$ *and* $\texttt{out}(ArgLab)$ *are maximal (w.r.t. set-inclusion) among all complete argument labellings of AF, and*

2. *if* $\texttt{undec}(ArgLab)$ *is maximal (w.r.t. set-inclusion) among all complete argument labellings of* AF *then* $\texttt{in}(ArgLab)$ *and* $\texttt{out}(ArgLab)$ *are minimal (w.r.t. set-inclusion) among all complete argument labellings of* AF*.*

Proof. We only state the proof of statement (1), the proof of (2) is very similar. Let $ArgLab$ be a complete argument labelling and suppose that $\texttt{undec}(ArgLab)$ is minimal. That is, for each complete argument labelling $ArgLab'$ of AF, it holds that if $\texttt{undec}(ArgLab') \subseteq \texttt{undec}(ArgLab)$ then also $\texttt{undec}(ArgLab) \subseteq \texttt{undec}(ArgLab')$. To show that $\texttt{in}(ArgLab)$ is maximal, we need to prove that for each complete argument labelling $ArgLab'$, if $\texttt{in}(ArgLab) \subseteq \texttt{in}(ArgLab')$ then $\texttt{in}(ArgLab') \subseteq \texttt{in}(ArgLab)$. Let $ArgLab'$ be a complete argument labelling such that $\texttt{in}(ArgLab) \subseteq \texttt{in}(ArgLab')$. From Lemma 2 it then follows that $\texttt{undec}(ArgLab) \supseteq \texttt{undec}(ArgLab')$. This, together with the assumption that $\texttt{undec}(ArgLab)$ is minimal, gives us that $\texttt{undec}(ArgLab) = \texttt{undec}(ArgLab')$. Therefore, it cannot be the case that $\texttt{in}(ArgLab) \subsetneq \texttt{in}(ArgLab')$ (otherwise $\texttt{undec}(ArgLab) \supsetneq \texttt{undec}(ArgLab')$ would follow from Lemma 2, which would contradict that $\texttt{undec}(ArgLab) = \texttt{undec}(ArgLab')$). As we assumed $\texttt{in}(ArgLab) \subseteq \texttt{in}(ArgLab')$ we can conclude that $\texttt{in}(ArgLab) = \texttt{in}(ArgLab')$ and thus $\texttt{in}(ArgLab') \subseteq \texttt{in}(ArgLab)$. The case of maximality of $\texttt{out}(ArgLab)$ now follows from Theorem 12. □

Although strictly spoken, theorems 13 and 14 (but not Theorem 34) have been proven by Caminada et al. [5, 6], this was done by making the conncetion with extension-based argumentation semantics. Our alternative proofs show that the labelling theory is able to "stand on its own feet", in the sense that fundamental properties (like the uniqueness of the grounded labelling) can be proved without applying any results from extension-based argumentation theory.

Theorem 13. *Let* $AF = (Ar, att)$ *be an argumentation framework. The complete argument labelling* $ArgLab$ *where* $\texttt{in}(ArgLab)$ *is minimal (w.r.t. set inclusion) among all complete argument labellings is unique.*

Proof. Let $ArgLab_1$ and $ArgLab_2$ be two complete argument labellings where both $\texttt{in}(ArgLab_1)$ and $\texttt{in}(ArgLab_2)$ are minimal. We will now prove that $ArgLab_1 = ArgLab_2$. First we observe that from Theorem 12, it follows that also $\texttt{out}(ArgLab_1)$ and $\texttt{out}(ArgLab_2)$ are minimal. Let us now define $ArgLab$ to be the outcome of the skeptical judgment aggregation operator of [41, Def. 18]. It holds that *ArgLab* is a complete argument labelling [41, Th. 8] such that $\texttt{in}(\mathit{ArgLab}) \subseteq \texttt{in}(\mathit{ArgLab}_1)$, $\texttt{out}(\mathit{ArgLab}) \subseteq \texttt{out}(\mathit{ArgLab}_1)$, $\texttt{in}(\mathit{ArgLab}) \subseteq \texttt{in}(\mathit{ArgLab}_2)$ and $\texttt{out}(\mathit{ArgLab}) \subseteq \texttt{out}(\mathit{ArgLab}_2)$ ([41, Th. 7]). The fact that ArgLab_1 and ArgLab_2 are complete argument labellings with minimal `in` and minimal `out` then implies that $\texttt{in}(\mathit{ArgLab}) = \texttt{in}(\mathit{ArgLab}_1)$, $\texttt{out}(\mathit{ArgLab}) = \texttt{out}(\mathit{ArgLab}_1)$, $\texttt{in}(\mathit{ArgLab}) = \texttt{in}(\mathit{ArgLab}_2)$ and $\texttt{out}(\mathit{ArgLab}) = \texttt{out}(\mathit{ArgLab}_2)$, so $\mathit{ArgLab} = \mathit{ArgLab}_1$ and $\mathit{ArgLab} = \mathit{ArgLab}_2$, so $\mathit{ArgLab}_1 = \mathit{ArgLab}_2$. □

Theorem 14. *Let $AF = (Ar, att)$ be an argumentation framework and ArgLab be one of its complete argument labellings. Then* $\texttt{undec}(\mathit{ArgLab})$ *is maximal (w.r.t. set-inclusion) among all complete argument labellings iff* $\texttt{in}(\mathit{ArgLab})$ *is minimal (w.r.t. set-inclusion) among all complete argument labellings.*

Proof. "$\Rightarrow$": This follows from Theorem 34.
"$\Leftarrow$": Let *ArgLab* be the unique (Theorem 13) complete argument labelling where $\texttt{in}(\mathit{ArgLab})$ is minimal. That is, for any complete argument labelling ArgLab' it holds that $\texttt{in}(\mathit{ArgLab}) \subseteq \texttt{in}(\mathit{ArgLab}')$, so we also have (Lemma 2) $\texttt{undec}(\mathit{ArgLab}) \supseteq \texttt{undec}(\mathit{ArgLab}')$. Therefore, *ArgLab* is also a complete argument labelling where $\texttt{undec}(\mathit{ArgLab})$ is maximal. □

A.2. Theorems and Proofs from Section 5:

Lemma 3. *Let $\mathit{ConcLab}_1$ and $\mathit{ConcLab}_2$ be complete conclusion labellings of logic program P and $AF_P = (Ar_P, att_P)$ be the associated argumentation framework, then it holds that*

1. $\texttt{in}(\mathit{ConcLab}_1) \subseteq \texttt{in}(\mathit{ConcLab}_2)$ *iff* $\texttt{out}(\mathit{ConcLab}_1) \subseteq \texttt{out}(\mathit{ConcLab}_2)$
2. $\texttt{in}(\mathit{ConcLab}_1) = \texttt{in}(\mathit{ConcLab}_2)$ *iff* $\texttt{out}(\mathit{ConcLab}_1) = \texttt{out}(\mathit{ConcLab}_2)$
3. $\texttt{in}(\mathit{ConcLab}_1) \subsetneq \texttt{in}(\mathit{ConcLab}_2)$ *iff* $\texttt{out}(\mathit{ConcLab}_1) \subsetneq \texttt{out}(\mathit{ConcLab}_2)$

Proof. Let ArgLab_1 be a complete argument labelling of which $\mathit{ConcLab}_1$ is the associated conclusion labelling, and let ArgLab_2 be a complete argument labelling of which $\mathit{ConcLab}_2$ is the associated conclusion labelling.

1. "$\Rightarrow$": Suppose $\texttt{in}(\mathit{ConcLab}_1) \subseteq \texttt{in}(\mathit{ConcLab}_2)$. Let $c \in \texttt{out}(\mathit{ConcLab}_1)$. We will now prove that $c \in \texttt{out}(\mathit{ConcLab}_2)$. First, we observe that the fact that $c \in \texttt{out}(\mathit{ConcLab}_1)$ implies (Definition 11) that for each argument A such that $\texttt{Conc}(A) = c$ it holds that $\mathit{ArgLab}_1(A) = \texttt{out}$. This implies (Definition 10) that each such A has an attacker (say

B) such that $ArgLab_1(B) = \mathtt{in}$, which implies that (Definition 11) that $ConcLab_1(\mathtt{Conc}(B)) = \mathtt{in}$. Given the assumption that $\mathtt{in}(ConcLab_1) \subseteq \mathtt{in}(ConcLab_2)$, it then follows that $ConcLab_2(\mathtt{Conc}(B)) = \mathtt{in}$, which (Definition 11) implies that there exists an argument $C \in Ar_P$ with $\mathtt{Conc}(C) = \mathtt{Conc}(B)$ and $ArgLab_2(C) = \mathtt{in}$. Since the notion of attack is based on the conclusion of the attacking argument (Definition 8), it follows that C attacks A. Since C is labelled $\mathtt{in}$ by $ArgLab_2$, it follows (Definition 10) that A is labelled $\mathtt{out}$ by $ArgLab_2$. Since this holds for any argument A with conclusion c, it follows (Definition 11) that $ConcLab_2(c) = \mathtt{out}$. That is, $c \in \mathtt{out}(ConcLab_2)$.

"$\Leftarrow$": Suppose $\mathtt{out}(ConcLab_1) \subseteq \mathtt{out}(ConcLab_2)$. Further, assume $c \in \mathtt{in}(ConcLab_1)$. First, we observe that the fact that $c \in \mathtt{in}(ConcLab_1)$ implies (Definition 11) that there is an argument (say A) such that $\mathtt{Conc}(A) = c$ and $ArgLab_1(A) = \mathtt{in}$. This implies (Definition 10) that for each attacker B of A it holds that $ArgLab_1(B) = \mathtt{out}$. This means that (Definition 8) for each argument B with $\mathtt{Conc}(B) \in \mathtt{Vul}(A)$ it holds that $ArgLab_1(B) = \mathtt{out}$, which then implies (Definition 11) that for each $b \in \mathtt{Vul}(A)$ it holds that $ConcLab_1(b) = \mathtt{out}$. From our initial assumption that $\mathtt{out}(ConcLab_1) \subseteq \mathtt{out}(ConcLab_2)$, it then follows that $ConcLab_2(b) = \mathtt{out}$. So for every $B \in Ar_P$ with $\mathtt{Conc}(B) = b$, it holds that $ArgLab_2(B) = \mathtt{out}$ (Definition 11), which implies that all attackers of A are labelled $\mathtt{out}$ by $ArgLab_2$ (Definition 8). Hence, it follows that A is labelled $\mathtt{in}$ by $ArgLab_2$ (Definition 1). Since $\mathtt{Conc}(A) = c$, it follows (Definition 11) that $ConcLab_2(c) = \mathtt{in}$. That is, $c \in \mathtt{in}(ConcLab_2)$.

2. This follows from point 1.
3. This follows from point 1 and point 2.

$\square$

From Lemma 3 it immediately follows that $\mathtt{in}(ConcLab_1) = \mathtt{in}(ConcLab_2)$ iff $ConcLab_1 = ConcLab_2$, and that $\mathtt{out}(ConcLab_1) = \mathtt{out}(ConcLab_2)$ iff $ConcLab_1 = ConcLab_2$.

Theorem 15. *Let ConcLab be a complete conclusion labelling of logic program P and the associated argumentation framework $AF_P = (Ar_P, att_P)$. It holds that*

- $\mathtt{in}(ConcLab)$ *is maximal (w.r.t. set-inclusion) among all complete conclusion labellings of AF iff* $\mathtt{out}(ConcLab)$ *is maximal (w.r.t. set-inclusion) among all complete conclusion labellings of AF.*

- $\texttt{in}(\mathit{ConcLab})$ *is minimal (w.r.t. set-inclusion) among all complete conclusion labellings of AF iff* $\texttt{out}(\mathit{ConcLab})$ *is also minimal (w.r.t. set-inclusion) among all complete conclusion labellings of AF.*

Proof. This follows directly from Lemma 3. □

Lemma 4. *Let* $\mathit{ConcLab}_1$ *and* $\mathit{ConcLab}_2$ *be complete conclusion labellings of logic program* P *and the associated argumentation framework* $AF_P = (Ar_P, att_P)$*. It holds that*

1. if $\texttt{in}(\mathit{ConcLab}_1) \subseteq \texttt{in}(\mathit{ConcLab}_2)$ *then* $\texttt{undec}(\mathit{ConcLab}_1) \supseteq \texttt{undec}(\mathit{ConcLab}_2)$
2. if $\texttt{out}(\mathit{ConcLab}_1) \subseteq \texttt{out}(\mathit{ConcLab}_2)$ *then* $\texttt{undec}(\mathit{ConcLab}_1) \supseteq \texttt{undec}(\mathit{ConcLab}_2)$
3. if $\texttt{in}(\mathit{ConcLab}_1) \subsetneq \texttt{in}(\mathit{ConcLab}_2)$ *then* $\texttt{undec}(\mathit{ConcLab}_1) \supsetneq \texttt{undec}(\mathit{ConcLab}_2)$
4. if $\texttt{out}(\mathit{ConcLab}_1) \subsetneq \texttt{out}(\mathit{ConcLab}_2)$ *then* $\texttt{undec}(\mathit{ConcLab}_1) \supsetneq \texttt{undec}(\mathit{ConcLab}_2)$

Proof. We prove the four statements separately.

1. Suppose that $\texttt{in}(\mathit{ConcLab}_1) \subseteq \texttt{in}(\mathit{ConcLab}_2)$. Then, by Lemma 3, it follows that $\texttt{out}(\mathit{ConcLab}_1) \subseteq \texttt{out}(\mathit{ConcLab}_2)$. By definition, we have that $(\texttt{in}(\mathit{ConcLab}_1), \texttt{out}(\mathit{ConcLab}_1), \texttt{undec}(\mathit{ConcLab}_1))$ as well as $(\texttt{in}(\mathit{ConcLab}_2), \texttt{out}(\mathit{ConcLab}_2), \texttt{undec}(\mathit{ConcLab}_2))$ are partitions. Then, given that $\texttt{in}(\mathit{ConcLab}_1) \subseteq \texttt{in}(\mathit{ConcLab}_2)$ and $\texttt{out}(\mathit{ConcLab}_1) \subseteq \texttt{out}(\mathit{ConcLab}_2)$, we conclude $\texttt{undec}(\mathit{ConcLab}_1) \supseteq \texttt{undec}(\mathit{ConcLab}_2)$.
2. Similar to the first point.
3. Suppose that $\texttt{in}(\mathit{ConcLab}_1) \subsetneq \texttt{in}(\mathit{ConcLab}_2)$, by Lemma 3 it follows that also $\texttt{out}(\mathit{ConcLab}_1) \subsetneq \texttt{out}(\mathit{ConcLab}_2)$. Consider $(\texttt{in}(\mathit{ConcLab}_1), \texttt{out}(\mathit{ConcLab}_1), \texttt{undec}(\mathit{ConcLab}_1))$ and $(\texttt{in}(\mathit{ConcLab}_2), \texttt{out}(\mathit{ConcLab}_2), \texttt{undec}(\mathit{ConcLab}_2))$ are partitions. Combining this with the facts that $\texttt{in}(\mathit{ConcLab}_1) \subsetneq \texttt{in}(\mathit{ConcLab}_2)$ and $\texttt{out}(\mathit{ConcLab}_1) \subsetneq \texttt{out}(\mathit{ConcLab}_2)$, we get $\texttt{undec}(\mathit{ConcLab}_1) \supsetneq \texttt{undec}(\mathit{ConcLab}_2)$.
4. Similar to the third point.

□

Theorem 35. *Let ConcLab be a complete conclusion labelling of logic program* P *and associated argumentation framework* $AF_P = (Ar_P, att_P)$*. It holds that*

1. *if* $\texttt{undec}(\mathit{ConcLab})$ *is minimal (w.r.t. set-inclusion) among all complete conclusion labellings of AF then* $\texttt{in}(\mathit{ConcLab})$ *and* $\texttt{out}(\mathit{ConcLab})$ *are maximal (w.r.t. set-inclusion) among all complete conclusion labellings of AF, and*

2. *if* $\texttt{undec}(\mathit{ConcLab})$ *is maximal (w.r.t. set-inclusion) among all complete conclusion labellings of AF then* $\texttt{in}(\mathit{ConcLab})$ *and* $\texttt{out}(\mathit{ConcLab})$ *are minimal (w.r.t. set-inclusion) among all complete conclusion labellings of AF.*

Proof. We only present the proof for statement (1), statement (2) can be shown similarly.

Suppose *ConcLab* is a complete conclusion labelling such that the set $\texttt{undec}(\mathit{ConcLab})$ is minimal. Then, for each complete conclusion labelling $\mathit{ConcLab}'$, if $\texttt{undec}(\mathit{ConcLab}') \subseteq \texttt{undec}(\mathit{ConcLab})$ then $\texttt{undec}(\mathit{ConcLab}) \subseteq \texttt{undec}(\mathit{ConcLab}')$. In order to prove that $\texttt{in}(\mathit{ConcLab})$ is maximal, we need to prove that for each complete conclusion labelling $\mathit{ConcLab}'$, if $\texttt{in}(\mathit{ConcLab}) \subseteq \texttt{in}(\mathit{ConcLab}')$ then $\texttt{in}(\mathit{ConcLab}') \subseteq \texttt{in}(\mathit{ConcLab})$. Assume $\texttt{in}(\mathit{ConcLab}) \subseteq \texttt{in}(\mathit{ConcLab}')$ for some complete conclusion labelling $\mathit{ConcLab}'$. It follows (Lemma 4) that $\texttt{undec}(\mathit{ConcLab}) \supseteq \texttt{undec}(\mathit{ConcLab}')$. From our initial assumption, one can conclude that $\texttt{undec}(\mathit{ConcLab}) \subseteq \texttt{undec}(\mathit{ConcLab}')$. As a consequence we have $\texttt{undec}(\mathit{ConcLab}) = \texttt{undec}(\mathit{ConcLab}')$. This means it cannot be the case that $\texttt{in}(\mathit{ConcLab}) \subsetneq \texttt{in}(\mathit{ConcLab}')$ (otherwise Lemma 4 would imply that $\texttt{undec}(\mathit{ConcLab}) \supsetneq \texttt{undec}(\mathit{ConcLab}')$) so $\texttt{in}(\mathit{ConcLab}) = \texttt{in}(\mathit{ConcLab}')$, so $\texttt{in}(\mathit{ConcLab}') \subseteq \texttt{in}(\mathit{ConcLab})$. From the thus obtained fact that *ConcLab* has maximal `in`, it follows (Theorem 15) that *ConcLab* also has maximal `out`. □

Theorem 16. *Let P be a logic program and $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. The complete conclusion labelling ConcLab of AF_P where* $\texttt{in}(\mathit{ConcLab})$ *is minimal (w.r.t. set inclusion) among all complete conclusion labellings of AF_P is unique.*

Proof. It suffices to show that the associated conclusion labelling of the grounded argument labelling has a set of `in`-labelled conclusions that is a subset of the set of `in`-labelled conclusions of any arbitrary complete conclusion labelling. Let ArgLab_{gr} be the grounded argument labelling of AF_P and $\mathit{ConcLab}_{gr}$ be its associated conclusion labelling (that is, $\mathit{ConcLab}_{gr}$ is the grounded conclusion labelling). We need to prove that for any complete conclusion labelling *ConcLab* it holds that $\texttt{in}(\mathit{ConcLab}_{gr}) \subseteq \texttt{in}(\mathit{ConcLab})$. Let *ConcLab* be an arbitrary complete conclusion labelling of P and AF_P. Assume it is the associated conclusion labelling of complete argument labelling *ArgLab*. From the fact that ArgLab_{gr} is the grounded argument labelling, it follows (Theorem 13) that $\texttt{in}(\mathit{ArgLab}_{gr}) \subseteq \texttt{in}(\mathit{ArgLab})$. Let

$c \in \texttt{in}(\mathit{ConcLab}_{gr})$. Then (Definition 11) there exists an argument $A \in Ar_P$ with $\texttt{Conc}(A) = c$ and $\mathit{ArgLab}_{gr}(A) = \texttt{in}$. From the fact that $\texttt{in}(\mathit{ArgLab}_{gr}) \subseteq \texttt{in}(\mathit{ArgLab})$ it follows that $\mathit{ArgLab}(A) = \texttt{in}$, so (Definition 11) $\mathit{ConcLab}(c) = \texttt{in}$. That is, $c \in \texttt{in}(\mathit{ConcLab})$. So $\texttt{in}(\mathit{ConcLab}_{gr}) \subseteq \texttt{in}(\mathit{ConcLab})$. □

Theorem 17. *Let P be a logic program, $AF_P = (Ar_P, att_P)$ be the associated argumentation framework of P and $\mathit{ConcLab}$ be one of the complete conclusion labellings of AF_P. It holds that $\texttt{undec}(\mathit{ConcLab})$ is maximal (w.r.t. set-inclusion) among all complete conclusion labellings of AF_P iff $\texttt{in}(\mathit{ConcLab})$ is minimal (w.r.t. set-inclusion) among all complete conclusion labellings of AF_P.*

Proof. "⇒": This follows from Theorem 35.
"⇐": Let $\mathit{ConcLab}$ be the unique (Theorem 16) complete conclusion labelling with minimal $\texttt{in}(\mathit{ConcLab})$. That is, for any complete conclusion labelling $\mathit{ConcLab}'$ it holds that $\texttt{in}(\mathit{ConcLab}) \subseteq \texttt{in}(\mathit{ConcLab}')$, so (by Lemma 4) $\texttt{undec}(\mathit{ConcLab}) \supseteq \texttt{undec}(\mathit{ConcLab}')$. Therefore, $\mathit{ConcLab}$ is also a complete conclusion labelling where $\texttt{undec}(\mathit{ConcLab})$ is maximal. □

A.3. Theorems and Proofs from Section 6:

Theorem 19. *When restricted to complete argument labellings and complete conclusion labellings, the functions* `ArgLab2ConcLab` *and* `ConcLab2ArgLab` *are bijections and each other's inverse.*

Proof. It suffices to prove two things:

"$\texttt{ConcLab2ArgLab}(\texttt{ArgLab2ConcLab}(\mathit{ArgLab})) = \mathit{ArgLab}$"

Let ArgLab be a complete argument labelling, and let A be an argument. We distinguish three cases.

1. $\mathit{ArgLab}(A) = \texttt{in}$. From the fact that ArgLab is a complete argument labelling, it follows that $\mathit{ArgLab}(B) = \texttt{out}$ for every attacker B of A. From the definition of attack (Definition 8) it follows that for each $b \in \texttt{Vul}(A)$, for each argument B with $\texttt{Conc}(B) = b$, $\mathit{ArgLab}(B) = \texttt{out}$. This then implies that for each $b \in \texttt{Vul}(A)$ it holds that $\texttt{ArgLab2ConcLab}(\mathit{ArgLab})(b) = \texttt{out}$. By the definition of `ConcLab2ArgLab`, we finally obtain that $\texttt{ConcLab2ArgLab}(\texttt{ArgLab2ConcLab}(\mathit{ArgLab}))(A) = \texttt{in}$.

2. $ArgLab(A) = \texttt{out}$. From the fact that $ArgLab$ is a complete argument labelling, it follows that there exists an attacker B of A such that $ArgLab(B) = \texttt{in}$. Let $b = \texttt{Conc}(B)$. From the definition of attack, it then follows that $b \in \texttt{Vul}(A)$. From the definition of $\texttt{ArgLab2ConcLab}$, it follows that $\texttt{ArgLab2ConcLab}(ArgLab)(b) = \texttt{in}$. From the definition of $\texttt{ConcLab2ArgLab}$, it then follows that $\texttt{ConcLab2ArgLab}(\texttt{ArgLab2ConcLab}(ArgLab))(A) = \texttt{out}$.
3. $ArgLab(A) = \texttt{undec}$. From the fact that $ArgLab$ is a complete argument labelling, it follows that not each attacker C of A has $ArgLab(C) = \texttt{out}$ (i), and there is no attacker D if A that has $ArgLab(D) = \texttt{in}$ (ii). From (i) together with (ii), it follows that there exists an attacker B of A with $ArgLab(B) = \texttt{undec}$. Let $b = \texttt{Conc}(B)$. From (ii) together with the definition of attack, it follows that there is no argument B' with $\texttt{Conc}(B') = b$ such that $ArgLab(B') = \texttt{in}$. Hence, $\texttt{ArgLab2ConcLab}(ArgLab)(b) = \texttt{undec}$ (iii). Furthermore, from (ii) together with the definition of attack, it follows that for each argument D with $\texttt{Conc}(D) \in \texttt{Vul}(A)$ it holds that $ArgLab(D) \neq \texttt{in}$. Hence, for each $d \in \texttt{Vul}(A)$, $\texttt{ArgLab2ConcLab}(ArgLab)(d) \neq \texttt{in}$ (iv). From (iii) and (iv), together with the definition of $\texttt{ConcLab2ArgLab}$, it follows that $\texttt{ConcLab2ArgLab}(\texttt{ArgLab2ConcLab}(ArgLab))(A) = \texttt{undec}$.

"$\texttt{ArgLab2ConcLab}(\texttt{ConcLab2ArgLab}(ConcLab)) = ConcLab$"

Let $ConcLab$ be a complete conclusion labelling. This by definition implies that there exists a complete argument labelling $ArgLab$ with $\texttt{ArgLab2ConcLab}(ArgLab) = ConcLab$. As we have observed in the previously it holds that $\texttt{ConcLab2ArgLab}(\texttt{ArgLab2ConcLab}(ArgLab)) = ArgLab$. It then follows that $\texttt{ConcLab2ArgLab}(ConcLab) = ArgLab$. This then implies that $\texttt{ArgLab2ConcLab}(\texttt{ConcLab2ArgLab}(ConcLab)) = \texttt{ArgLab2ConcLab}(ArgLab)$. Combing these observations we finally obtain $\texttt{ArgLab2ConcLab}(\texttt{ConcLab2ArgLab}(ConcLab) = ConcLab$.

□

Lemma 5. *Let P be a logic program, $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. Let $ArgLab_1$ and $ArgLab_2$ be complete argument labellings of AF_P, and $ConcLab_1$ and $ConcLab_2$ be their respective associated conclusion labellings. It holds that*

1. $\texttt{in}(ArgLab_1) \subseteq \texttt{in}(ArgLab_2)$ *iff* $\texttt{in}(ConcLab_1) \subseteq \texttt{in}(ConcLab_2)$,

2. $\mathtt{in}(\mathit{ArgLab}_1) = \mathtt{in}(\mathit{ArgLab}_2)$ *iff* $\mathtt{in}(\mathit{ConcLab}_1) = \mathtt{in}(\mathit{ConcLab}_2)$, *and*
3. $\mathtt{in}(\mathit{ArgLab}_1) \subsetneq \mathtt{in}(\mathit{ArgLab}_2)$ *iff* $\mathtt{in}(\mathit{ConcLab}_1) \subsetneq \mathtt{in}(\mathit{ConcLab}_2)$.

Proof.

1. "$\Rightarrow$": Suppose $\mathtt{in}(\mathit{ArgLab}_1) \subseteq \mathtt{in}(\mathit{ArgLab}_2)$. Further, assume that $c \in \mathtt{in}(\mathit{ConcLab}_1)$. Then, by definition of $\mathtt{ArgLab2ConcLab}$, there exists an argument $A \in Ar_P$ with $\mathtt{Conc}(A) = c$ and $\mathit{ArgLab}_1(A) = \mathtt{in}$. From our initial assumption, it follows that $\mathit{ArgLab}_2(A) = \mathtt{in}$. So, by definition of $\mathtt{ArgLab2ConcLab}$, $c \in \mathtt{in}(\mathit{ConcLab}_2)$.
   "$\Leftarrow$": Suppose $\mathtt{in}(\mathit{ConcLab}_1) \subseteq \mathtt{in}(\mathit{ConcLab}_2)$. Let $A \in \mathtt{in}(\mathit{ArgLab}_1)$. Then it follows (Definition 10) that each attacker B of A is labelled $\mathtt{out}$ by ArgLab_1. That is (Definition 8) for each $B \in Ar_P$ with $\mathtt{Conc}(B) \in \mathtt{Vul}(A)$ it holds that $\mathit{ArgLab}_1(B) = \mathtt{out}$. According to the definition of $\mathtt{ArgLab2ConcLab}$ it then follows that for each $v \in \mathtt{Vul}(A)$ it holds that $\mathit{ConcLab}_1(v) = \mathtt{out}$. From our initial assumption it follows (Lemma 3) that $\mathtt{out}(\mathit{ConcLab}_1) \subseteq \mathtt{out}(\mathit{ConcLab}_2)$. Therefore, $\mathit{ConcLab}_2(v) = \mathtt{out}$. This (by definition of $\mathtt{ArgLab2ConcLab}$) implies that each argument (say C) with $\mathtt{Conc}(C) \in \mathtt{Vul}(A)$ is labelled $\mathtt{out}$ by ArgLab_2. Therefore (Definition 8) each attacker of A is labelled $\mathtt{out}$ by ArgLab_2, so (Lemma 1) A is labelled $\mathtt{in}$ by ArgLab_2. That is, $A \in \mathtt{in}(\mathit{ArgLab}_2)$.
2. This follows directly from point 1.
3. This follows directly from points 1 and 2.

□

Lemma 6. *Let P be a logic program, $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. Let ArgLab_1 and ArgLab_2 be complete argument labellings of AF_P, and $\mathit{ConcLab}_1$ and $\mathit{ConcLab}_2$ be their respective associated conclusion labellings. It holds that*

1. $\mathtt{out}(\mathit{ArgLab}_1) \subseteq \mathtt{out}(\mathit{ArgLab}_2)$ *iff* $\mathtt{out}(\mathit{ConcLab}_1) \subseteq \mathtt{out}(\mathit{ConcLab}_2)$,
2. $\mathtt{out}(\mathit{ArgLab}_1) = \mathtt{out}(\mathit{ArgLab}_2)$ *iff* $\mathtt{out}(\mathit{ConcLab}_1) = \mathtt{out}(\mathit{ConcLab}_2)$,
3. $\mathtt{out}(\mathit{ArgLab}_1) \subsetneq \mathtt{out}(\mathit{ArgLab}_2)$ *iff* $\mathtt{out}(\mathit{ConcLab}_1) \subsetneq \mathtt{out}(\mathit{ConcLab}_2)$.

Proof. This follows from Lemma 5, with Lemma 1 and Lemma 3. □

Theorem 20. *Let ConcLab be a conclusion labelling of logic program P and associated argumentation framework $AF_P = (Ar, att)$. It holds that ConcLab is a preferred conclusion labelling iff it is a regular conclusion labelling.*

Proof. "$\Rightarrow$": Suppose *ConcLab* is a preferred conclusion labelling. By definition, there must exists a preferred argument labelling *ArgLab* such that `ArgLab2ConcLab`(*ArgLab*) = *ConcLab*. The fact that *ArgLab* is a preferred argument labelling means that `in`(*ArgLab*) is maximal (w.r.t. set-inclusion) among all complete argument labellings. This implies (Lemma 5) that `in`(*ConcLab*) is maximal (w.r.t. set-inclusion) among all complete conclusion labellings. That is, *ConcLab* is a regular conclusion labelling.
"$\Leftarrow$": Let *ConcLab* be a regular conclusion labelling and take an argument labeling *ArgLab* such that `ArgLab2ConcLab`(*ArgLab*) = *ConcLab*. The fact that *ConcLab* is a regular conclusion labelling means that `in`(*ConcLab*) is maximal (w.r.t. set-inclusion) among all complete conclusion labellings. This implies (by Lemma 5) that `in`(*ArgLab*) is maximal (w.r.t. set-inclusion) among all complete argument labellings. That is, *ConcLab* is a preferred conclusion labelling. □

Theorem 21. *Let ConcLab be a conclusion labelling of logic program P and associated argumentation framework $AF_P = (Ar, att)$. It holds that ConcLab is the grounded conclusion labelling iff it is the well-founded conclusion labelling.*

Proof. "$\Rightarrow$": Suppose *ConcLab* is the grounded conclusion labelling. Then, by definition, *ConcLab* must have been generated by the grounded argument labelling. That is, *ConcLab* = `ArgLab2ConcLab`(*ArgLab*) where *ArgLab* is the grounded argument labelling. The fact that *ArgLab* is the grounded argument labelling means that `in`(*ArgLab*) is minimal (w.r.t. set inclusion) among all complete argument labellings. This implies (Lemma 5) that `in`(*ConcLab*) is minimal (w.r.t. set inclusion) among all complete conclusion labellings. That is, *ConcLab* is the well-founded conclusion labelling.
"$\Leftarrow$": Let *ConcLab* be the well-founded conclusion labelling. As the well-founded conclusion labelling is a complete conclusion labelling, it must have been generated by a complete argument labelling. That is, *ConcLab* = `ArgLab2ConcLab`(*ArgLab*) for some complete argument labelling *ArgLab*. The fact that *ConcLab* is the well-founded conclusion labelling means that `in`(*ConcLab*) is minimal among all complete conclusion labellings. This implies (Lemma 5) that `in`(*ArgLab*) is minimal (w.r.t. set inclusion) among all complete argument labellings. That is, *ArgLab* is the grounded argument labelling, so by definition *ConcLab* is the grounded conclusion labelling. □

Theorem 22. *Let ConcLab be a conclusion labelling of logic program P and associated argumentation framework $AF_P = (Ar, att)$. It holds that ConcLab is an argstable conclusion labelling iff it is a concstable conclusion labelling.*

Proof. "$\Rightarrow$": Let *ConcLab* be an argstable conclusion labelling. So there is a stable argument labelling *ArgLab* such that `ArgLab2ConcLab`(*ArgLab*) = *ConcLab*. The fact that *ArgLab* is a stable argument labelling means that no argument is labelled `undec`. But then, by Definition 11, also no conclusion in *ConcLab* is labelled `undec`. Hence, *ConcLab* is a concstable conclusion labelling.
"$\Leftarrow$": Let *ConcLab* be a concstable conclusion labelling and take an argument labeling *ArgLab* such that `ConcLab2ArgLab`(*ConcLab*). The fact that *ConcLab* is a concstable labelling means that no conclusion is labelled `undec` by *ConcLab*. This, by definition of `ConcLab2ArgLab` implies that also no argument in *ArgLab* is labelled `undec`. That is, *ArgLab* is a stable argument labelling, which implies that *ConcLab* is a argstable conclusion labelling. □

A.4. Theorems and Proofs from Section 8:

Theorem 27. *Every stable program is semi-stable-L-stable compatible.*

Proof. Recall from Section 2.1 that semi-stable semantics will coincide with stable semantics whenever the framework has at least one stable extension. Furthermore, recall from Section 2.2 that L-stable semantics coincides with stable semantics whenever the program has one or more stable models. Because the argumentation stable semantics and program stable semantics are equivalent, our result holds for stable programs. □

Theorem 29. *Every AF-program is semi-stable-L-stable compatible.*

Proof. Let P be an AF-program and AF_P its associated argumentation framework. First we have to show that for each $c \in HB_P$ there is a most one argument having conclusion c. We do this by induction on the number n of rules in the program P. For $n = 1$ with just one rule r it is easy to see that we have one argument if $body^+(r) = \emptyset$ and no argument otherwise. Now let us assume the claim holds for programs with at most $n - 1$ rules. Towards a contradiction assume there is a c with two different arguments A'_0,A''_0. As there is just one rule rule with $head(r) = c$ both arguments are based on the same rule. As both arguments are different it must hold that (i) $body^+(r) \neq \emptyset$ and (ii) for at least one $a_i \in body^+(r)$ there must exist two arguments A_1,A_2

with $\texttt{Conc}(A_1) = \texttt{Conc}(A_2) = a_i$ and $r \notin \texttt{Rules}(A_1), r \notin \texttt{Rules}(A_2)$. But this implies that we can construct two arguments for a_i from the program $P \setminus \{r\}$ and thus contradicts our induction hypothesis. We obtain that for each $c \in HB_P$ there is a most one argument having conclusion c. As a consequence, minimizing undecided arguments is the same as minimizing undecided conclusions and P is semi-stable-L-stable compatible. $\square$

Theorem 31. *Every stratified program is semi-stable-L-stable compatible.*

Proof. A stratified program P has a single P-stable model S, so AF_P has a single complete conclusion labelling that coincides with S [4]. Therefore, the only L-stable extension of P coincides with the sole semi-stable conclusion labelling of AF_P. $\square$

A.5. Theorems and Proofs from Section 9:

Theorem 33. *Let AF be an argumentation framework, P_{AF} be the associated logic program (Definition 32), and $AF_{P_{AF}}$ be the argumentation framework that is associated with this logic program (Definition 9). It holds that AF and $AF_{P_{AF}}$ are isomorphic.*

Proof. Let $AF = (Ar, att)$ with $Ar = \{A_1, A_2, \ldots, A_n\}$. Further, let $P_{AF} = \{r_1, r_2, \ldots, r_n\}$ with each r_i being the rule corresponding to argument A_i, and $AF_{P_{AF}} = (Ar_{P_{AF}}, att_{P_{AF}})$ with $Ar_{P_{AF}} = \{Arg_1, Arg_2, \ldots, Arg_n\}$ with each Arg_i consisting of the single rule r_i. It is easy to see that AF is isomorphic with $AF_{P_{AF}}$ through the bijective function f such that $f(A_i) = Arg_i$. To this end, first consider two arguments A_i, A_j such that $(A_i, A_j) \in att$. Then A_i is the head of r_i and $\texttt{not}\ A_i$ is in the body of rule r_j. Thus also Arg_i attacks Arg_j, i.e., $(Arg_i, Arg_j) \in att_{P_{AF}}$. Finally consider A_i, A_j such that $(A_i, A_j) \notin att$. Then A_i is the head of r_i but $\texttt{not}\ A_i$ is not contained in the body of rule r_j. Thus Arg_i does not attack Arg_j, i.e., $(Arg_i, Arg_j) \notin att_{P_{AF}}$. Hence f is a isomorphism between AF and P_{AF}. $\square$

References

[1] H. Prakken, G. Sartor, Argument-based extended logic programming with defeasible priorities, Journal of Applied Non-Classical Logics 7 (1997) 25–75.

[2] G. Simari, R. Loui, A mathematical treatment of defeasible reasoning and its implementation, Artificial Intelligence 53 (1992) 125–157.

[3] P. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, Artificial Intelligence 77 (1995) 321–357.

[4] Y. Wu, M. Caminada, D. Gabbay, Complete extensions in argumentation coincide with 3-valued stable models in logic programming, Studia Logica 93 (1-2) (2009) 383–403, special issue: new ideas in argumentation theory.

[5] M. Caminada, On the issue of reinstatement in argumentation, in: M. Fischer, W. van der Hoek, B. Konev, A. Lisitsa (Eds.), Logics in Artificial Intelligence; 10th European Conference, JELIA 2006, Springer, 2006, pp. 111–123, LNAI 4160.

[6] M. Caminada, D. Gabbay, A logical account of formal argumentation, Studia Logica 93 (2-3) (2009) 109–145, special issue: new ideas in argumentation theory.

[7] M. Caminada, L. Amgoud, An axiomatic account of formal argumentation, in: Proceedings of the AAAI-2005, 2005, pp. 608–613.

[8] M. Caminada, L. Amgoud, On the evaluation of argumentation formalisms, Artificial Intelligence 171 (5-6) (2007) 286–310.

[9] P. Dung, R. Kowalski, F. Toni, Assumption-based argumentation, in: G. Simari, I. Rahwan (Eds.), Argumentation in Artificial Intelligence, Springer US, 2009, pp. 199–218.

[10] J. W. Lloyd, Foundations of logic programming; (2nd extended ed.), Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[11] T. Przymusinski, The well-founded semantics coincides with the three-valued stable semantics, Fundamenta Informaticae 13 (4) (1990) 445–463.

[12] T. C. Przymusinski, Stable semantics for disjunctive programs, New Generation Computing 9 (1991) 401–424.

[13] D. Saccà, C. Zaniolo, Stable models and non-determinism in logic programs with negation., in: D. J. Rosenkrantz, Y. Sagiv (Eds.), PODS, ACM Press, 1990, pp. 205–217.

URL `http://dblp.uni-trier.de/db/conf/pods/pods90.html#SaccaZ90`

[14] J.-H. You, L. Y. Yuan, On the equivalence of semantics for normal logic programs, The Journal of Logic Programming 22 (3) (1995) 211 – 222. `doi:http://dx.doi.org/10.1016/0743-1066(94)00023-Y`.
URL `http://www.sciencedirect.com/science/article/pii/074310669400023Y`

[15] T. Eiter, N. Leone, D. Saccá, On the partial semantics for disjunctive deductive databases, Ann. Math. Artif. Intell. 19 (1-2) (1997) 59–96.

[16] S. Modgil, H. Prakken, The ASPIC+ framework for structured argumentation: a tutorial, Argument & Computation 5 (2014) 31–62.

[17] G. Vreeswijk, Abstract argumentation systems, Artificial Intelligence 90 (1997) 225–279.

[18] N. Gorogiannis, A. Hunter, Instantiating abstract argumentation with classical logic arguments: Postulates and properties, Artificial Intelligence 175 (9-10) (2011) 1479–1497.

[19] M. Caminada, B. Verheij, On the existence of semi-stable extensions, in: G. Danoy, M. Seredynski, R. Booth, B. Gateau, I. Jars, D. Khadraoui (Eds.), Proceedings of the 22nd Benelux Conference on Artificial Intelligence, 2010.

[20] E. Weydert, Semi-stable extensions for infinite frameworks, in: P. de Causmaecker, J. Maervoet, T. Messelis, K. Verbeeck, T. Vermeulen (Eds.), Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011), 2011, pp. 336–343.

[21] Y. Wu, M. Caminada, A labelling-based justification status of arguments, Studies in Logic 3 (4) (2010) 12–29.

[22] Y. Wu, M. Podlaszewski, Implementing crash-resistance and non-interference in logic-based argumentation, Journal of Logic and Computation, *In print*. First published online: March 25, 2014. `doi:10.1093/logcom/exu017`.

[23] P. Baroni, M. Caminada, M. Giacomin, An introduction to argumentation semantics, Knowledge Engineering Review 26 (4) (2011) 365–410.

[24] M. Osorio, J. C. Nieves, A. Santoyo, Complete extensions as clark's completion semantics, in: ENC, IEEE, 2013, pp. 81–88.

[25] K. L. Clark, Negation as failure, in: H. Gallaire, J. Minker (Eds.), Logic and Data Bases, Plenum Press, New York, 1978, pp. 293–322.

[26] S. Modgil, M. Caminada, Proof theories and algorithms for abstract argumentation frameworks, in: I. Rahwan, G. Simari (Eds.), Argumentation in Artificial Intelligence, Springer, 2009, pp. 105–129.

[27] M. Caminada, M. Podlaszewski, Grounded semantics as persuasion dialogue, in: B. Verheij, S. Szeider, S. Woltran (Eds.), Computational Models of Argument - Proceedings of COMMA 2012, 2012, pp. 478–485.

[28] M. Caminada, M. Podlaszewski, User-computer persuasion dialogue for grounded semantics, in: J. W. Uiterwijk, N. Roos, M. H. Winands (Eds.), Proceedings of BNAIC 2012; The 24th Benelux Conference on Artificial Intelligence, 2012, pp. 343–344.

[29] M. Caminada, W. Dvořák, S. Vesic, Preferred semantics as socratic discussion, Journal of Logic and Computation, *In print.* First published online: February 12, 2014. `doi:10.1093/logcom/exu005`.

[30] S. Nofal, K. Atkinson, P. Dunne, Algorithms for decision problems in argumentation systems under preferred semantics, Artificial Intelligence 207 (2014) 23–51.

[31] A. García, G. Simari, Defeasible logic programming: an argumentative approach, Theory and Practice of Logic Programming 4 (1) (2004) 95–138.

[32] C. Schulz, F. Toni, Logic programming in assumption-based argumentation revisited — semantics and graphical representation, in: Proceedings of AAAI 2015, 2015, in print.

[33] M. Caminada, S. Sá, J. Alcântara, W. Dvořák, On the difference between assumption-based argumentation and abstract argumentation, Tech. rep., University of Aberdeen (2013).

[34] M. Caminada, S. Sá, J. Alcântara, W. Dvořák, On the difference between assumption-based argumentation and abstract argumentation, in: K. Hindriks, M. de Weerdt, B. van Riemsdijk, M. Warnier (Eds.), Proceedings BNAIC 2013, 2013, pp. 25–32.

[35] A. Bondarenko, P. Dung, R. Kowalski, F. Toni, An abstract, argumentation-theoretic approach to default reasoning, Artificial Intelligence 93 (1997) 63–101.

[36] R. Reiter, A logic for default reasoning, Artificial Intelligence 13 (1980) 81–132.

[37] H. Prakken, An abstract framework for argumentation with structured arguments, Argument and Computation 1 (2) (2010) 93–124.

[38] S. Modgil, H. Prakken, A general account of argumentation with preferences, Artificial Intellligence 195 (2013) 361–397.

[39] J. Pollock, Cognitive carpentry. A blueprint for how to build a person, MIT Press, Cambridge, MA, 1995.

[40] H. Jakobovits, D. Vermeir, Robust semantics for argumentation frameworks, Journal of logic and computation 9(2) (1999) 215–261.

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