Argument & Computation 13 (2022) 93–120
DOI 10.3233/AAC-200528
IOS Press

Comparing logic programming and formal argumentation; the case of ideal and eager semantics

Martin Caminada [a,*], Sri Harikrishnan [a] and Samy Sá [b]

[a] *Cardiff University, United Kingdom*
[b] *Universidade Federal do Ceará, Brazil*

Abstract. The connection between logic programming and formal argumentation has been studied starting from the landmark 1995 paper of Dung. Subsequent work has identified a standard translation from logic programs to (instantiated) argumentation frameworks, under which pairwise correspondences hold between various logic programming semantics and various formal argumentation semantics. This includes the correspondence between 3-valued stable and complete semantics, between well-founded and grounded semantics and between 2-valued stable (LP) and stable (argumentation) semantics. In the current paper, we show that the existing translation is able to yield the additional correspondence between ideal semantics for logic programming and ideal semantics for formal argumentation. We also show that correspondence does *not* hold between eager semantics for logic programming and eager semantics for formal argumentation, at least when translating from logic programming to formal argumentation. Overall, the current work should be seen as completing the analysis of correspondences between mainstream admissibility-based argumentation semantics and their logic programming counterparts.

Keywords: Formal argumentation, logic programming, ideal semantics, eager semantics

1. Introduction

The connection between logic programming and formal argumentation has been studied starting from the landmark paper of Dung [24]. Subsequent work of Wu *et al.* [47] defined a translation from logic programs to (instantiated) argumentation frameworks where each argument consists of a derivation for a particular conclusion, using the rules of the logic program; weak negation is used to determine the attack relation between those arguments. The idea is that a derivation (argument) for conclusion p attacks any derivation (argument) that has a rule containing `not` p. Based on this translation, Wu *et al.* [47] were able to show that the 3-valued stable model semantics for logic programming corresponds to the complete semantics for formal argumentation. More specifically, the conclusions associated with the complete argument labellings of the resulting argumentation framework coincide with the 3-valued stable models of the logic program one started with.

The translation from logic programs to argumentation frameworks of Wu *et al.* [47] has subsequently been used to identify additional semantic correspondences between logic programs and their associated

*Corresponding author. E-mail: CaminadaM@cardiff.ac.uk.

argumentation frameworks. For instance, Caminada *et al.* [16] used this translation to show the additional correspondence between well-founded semantics for logic programming and grounded semantics for formal argumentation, between 2-valued stable semantics for logic programming and stable semantics for formal argumentation, and between regular semantics for logic programming and preferred semantics for formal argumentation.[1]

In the current paper, we identify an additional correspondence between logic programming semantics and formal argumentation semantics: the correspondence between ideal semantics for logic programming and ideal semantics for formal argumentation. To do so, we will make use of a recent reformulation of ideal semantics for logic programming [17], which brings it closer to ideal semantics for formal argumentation.

The reformulation of ideal semantics for logic programming of [17] allows eager semantics for logic programming to be defined in a similar way. We then examine whether this coincides with eager semantics for formal argumentation. We will show that the answer is negative, and our analysis allows us to pinpoint where precisely the difference is.

The current work is part of a line of research that examines the expressiveness of formal argumentation. This is done by translating a particular formalism for non-monotonic reasoning to formal argumentation, and examining whether the conclusions yielded by the thus obtained argumentation formalism coincide with what is entailed by the non-monotonic reasoning formalism one started with. In his original 1995 AIJ paper, Dung did this for logic programming under the (2-valued) stable and well-founded semantics, as well as for Default Logic [24]. Follow-up research did this for Nute's Defeasible Logic [27], Pollock's OSCAR system [28] and for logic programming under 3-valued stable [47] and regular [16] semantics. It has to be mentioned that not all forms of non-monotonic reasoning can be captured by formal argumentation, as evidenced by the impossibility result of Caminada *et al.* regarding logic programming under the L-stable semantics [16]. Hence, by pursuing this line of research, one gains insight in what forms of non-monotonic reasoning can and cannot be represented by formal argumentation. Apart from this, a successful translation of a particular form of non-monotonic reasoning to formal argumentation makes it possible to apply argumentation techniques for explaining its entailment, for instance by using dialectical explanation techniques [12,18,25,30,43].

The remaining part of the current paper is structured as follows. First, in Section 2, we review some of the basic theory regarding logic programming and formal argumentation, including the existing translation (from [47]) from a logic program to an (instantiated) argumentation framework. We explain that whereas argumentation semantics are defined by minimizing and maximizing labels on the argument level, logic programming semantics are defined by minimizing and maximizing labels on the conclusion level. In Section 3, we present one of the main results of the current paper: the correspondence between the ideal model of a logic program and the ideal labelling of its associated argumentation framework. In Section 4, we perform a similar analysis regarding the eager model of a logic program and the eager labelling of is associated argumentation framework, and observe that here correspondence does not hold. In Section 5, we examine the correspondence between argumentation and logic programming (under ideal and eager semantics) when translating from an argumentation framework to its associated logic program. We round off with a discussion of the obtained results and an overview of the overall correspondences and non-correspondences between argumentation semantics and logic programming semantics in Section 6.

[1] The first two correspondences are also observed in [24], though in a different way.

2. Formal preliminaries

We start with a brief overview of some basic concepts in formal argumentation. For current purposes, we restrict ourselves to finite argumentation frameworks.

Definition 1 ([24])**.** An *argumentation framework* is a pair (Ar, att) where Ar is a finite set of arguments and $att \subseteq Ar \times Ar$.

For current purposes, we apply the labelling versions of argumentation semantics. As is explained in [10,13,14], these coincide with their respective extension-based variants.

Definition 2. Let $AF = (Ar, att)$ be an argumentation framework. An *argument labelling* is a function $\texttt{ArgLab} : Ar \rightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$. An argument labelling is called a *complete argument labelling* iff for each $A \in Ar$ it holds that:

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

Given an argument labelling ArgLab, we write val(ArgLab) to refer to the set of arguments labelled as $\texttt{val} \in \{\texttt{in}, \texttt{out}, \texttt{undec}\}$ in ArgLab. For convenience, we may as well refer to ArgLab as the tuple $(\texttt{in}(\texttt{ArgLab}), \texttt{out}(\texttt{ArgLab}), \texttt{undec}(\texttt{ArgLab}))$.

Definition 3. Let ArgLab be an argument labelling of argumentation framework $AF = (Ar, att)$. ArgLab is called:

- the *grounded argument labelling* iff ArgLab is a complete argument labelling where in(ArgLab) is minimal among all complete argument labellings of AF
- a *preferred argument labelling* iff ArgLab is a complete argument labelling where in(ArgLab) is maximal among all complete argument labellings of AF
- a *stable argument labelling* iff ArgLab is a complete argument labelling where undec(ArgLab) $= \emptyset$
- a *semi-stable argument labelling* iff ArgLab is a complete argument labelling where undec(ArgLab) is minimal among all complete argument labellings of AF

Each argumentation framework has one or more complete labellings, a unique grounded labelling, one or more preferred labellings, zero or more stable labellings and one or more semi-stable labellings.[2]

We now shift our attention to logic programming. We start with formally introducing the notion of a logic program. For current purposes, we restrict ourselves to normal logic programs, which are logic programs without strong negation where the head of each rule consists of a single atom.

Definition 4. A *logic programming rule* is an expression $x \leftarrow y_1, \ldots, y_n, \texttt{not}\ z_1, \ldots, \texttt{not}\ z_m$ ($n \geqslant 0$, $m \geqslant 0$) where x, each y_i ($1 \leqslant i \leqslant n$) and each z_j ($1 \leqslant j \leqslant m$) is an atom, and not represents negation as failure (NAF). A *logic program* P consists of a finite set of logic programming rules.

[2]This is because we only consider finite argumentation frameworks. An infinite argumentation framework can have zero or more semi-stable labellings [6,7,19,46].

Given a logic programming rule $x \leftarrow y_1, \ldots, y_n, \texttt{not}\ z_1, \ldots, \texttt{not}\ z_m$ (or simply *a rule*, for short), we say that x is the *head* of the rule and $y_1, \ldots, y_n, \texttt{not}\ z_1, \ldots, \texttt{not}\ z_m$ is the *body* of the rule. Moreover, we say that $y_1, \ldots, y_n$ is the strong part of the body and that $\texttt{not}\ z_1, \ldots, \texttt{not}\ z_m$ is the weak part of the body. Each expression $\texttt{not}\ w$, where w is an atom, is called a *NAF literal*. We say that a rule is NAF-free iff it does not contain any NAF literals (that is, iff $m = 0$).

Concerning logic programs (or simply *programs*, for short), we assume the availability of three special atoms TRUE, FALSE and UNDEFINED, which can only occur in the strong part of the body of a rule. The Herbrand Base of a logic program P (written as HB_P) is the set of all atoms in P (excluding the special atoms TRUE, FALSE and UNDEFINED). Finally, we say that a logic program is NAF-free iff each of its rules is NAF-free.

In the following, we recall the definitions of logic programming semantics as described in [16,17,35].

Definition 5. A *3-valued interpretation* of a logic program P with respect to a set of atoms $Atms \supseteq HB_P$ is a pair $\langle T, F \rangle$ where $T, F \subseteq Atms$ and $T \cap F = \emptyset$.

Definition 6. A 3-valued interpretation $\langle T, F \rangle$ of a NAF-free logic program P w.r.t. $Atms \supseteq HB_P$ is a *3-valued model* of P w.r.t. $Atms$ if for all logic programming rules $x \leftarrow y_1, \ldots, y_n$ in P it holds that

- $x \in T$ or
- $x \in F$ and $\exists i \in \{1, \ldots, n\} : (y_i \in F \vee y_i = \texttt{FALSE})$ or
- $x \in Atms \setminus (T \cup F)$ and $\exists i \in \{1, \ldots, n\} : (y_i \in F \vee y_i \in Atms \setminus (T \cup F) \vee y_i = \texttt{FALSE} \vee y_i = \texttt{UNDEFINED})$

When P is a NAF-free logic program (possibly containing TRUE, FALSE or UNDEFINED), we write $\Phi_{Atms}(P)$ for its unique *minimal* 3-valued model $\langle T, F \rangle$ (w.r.t. $Atms$), i.e. $\langle T, F \rangle$ has minimal T and maximal F (w.r.t. $\subseteq$) among all 3-valued models of P w.r.t. $Atms$ [35].

Definition 7. The *reduct* of a logic program P w.r.t. a 3-valued interpretation $\texttt{Mod} = \langle T, F \rangle$, written as $P^{\texttt{Mod}}$, is obtained by replacing in every rule each NAF literal $\texttt{not}\ x$ by TRUE if $x \in F$, by FALSE if $x \in T$, and by UNDEFINED otherwise.

Since $P^{\texttt{Mod}}$ is a NAF-free program, it has a unique minimal 3-valued model, written as $\Phi_{HB_P}(P^{\texttt{Mod}})$.

We now recall various logic programming semantics which are based on 3-valued interpretations. The presentation below was originally provided in [16], where correspondences to other equivalent concepts for the same semantics were also discussed. It is heavily based on Przymusinski's three-valued stable semantics [35] and tailored to ease the comparison to argumentation semantics.[3]

Definition 8 ([16])**.** Let P be a logic program and $\texttt{Mod} = \langle T, F \rangle$ be a 3-valued interpretation of P w.r.t. HB_P. We say that Mod is:

(1) a *3-valued stable model* iff $\Phi_{HB_P}(P^{\texttt{Mod}}) = \texttt{Mod}$
(2) the *well-founded model* iff Mod is a 3-valued stable model where T is minimal (w.r.t. $\subseteq$) among all 3-valued stable models of P
(3) a *regular model* iff Mod is a 3-valued stable model where T is maximal (w.r.t. $\subseteq$) among all 3-valued stable models of P
(4) a *2-valued stable model* iff Mod is a 3-valued stable model where $T \cup F = HB_P$

[3] A similar characterization of these semantics (except well-founded) as special cases of the three-valued stable semantics (there called P-stable models) can also be found in [39].

(5) an *L-stable model* iff Mod is a 3-valued stable model where $T \cup F$ is maximal (w.r.t. $\subseteq$) among all 3-valued stable models of P

Each logic program has one or more 3-valued stable models, a unique well-founded model, one or more regular models, zero or more 2-valued stable models and one or more L-stable models.

Next, we recall how logic programming semantics can be expressed in terms of formal argumentation. For this, we first need to translate a logic program to an argumentation framework, for which we apply the approach of [16].

Definition 9. Let P be a logic program.

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

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). If A 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.

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 10. Let A and B be arguments in the sense of Definition 9. We say that A *attacks* B iff $\texttt{Conc}(A) \in \texttt{Vul}(B)$.

Definition 11. 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 9 and att_P is the attack relation in the sense of Definition 10.

Based on the thus constructed argumentation framework, we can apply argumentation semantics and, based on the resulting argument labelling(s) determine the associated conclusion labelling(s), using the approach of [16].

Definition 12 ([16])**.** Let P be a logic program. A conclusion labelling of P is a function $\texttt{ConcLab} : HB_P \rightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$. Let $AF_P = (Ar_P, att_P)$ be the associated argumentation framework of P 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 $\texttt{ConcLab}(c) = \max(\{\texttt{ArgLab}(A) \mid \texttt{Conc}(A) = c\} \cup \{\texttt{out}\})$ where $\texttt{in} > \texttt{undec} > \texttt{out}$. We say that a conclusion labelling is *complete* iff it is the associated conclusion labelling of a complete argument labelling. We define the function ArgLab2ConcLab such that for any complete argument labelling ArgLab, ArgLab2ConcLab(ArgLab) is its associated conclusion labelling. We define ConcLab2ArgLab as the inverse function of ArgLab2ConcLab.

Conclusion labellings are very close to logic programming models. One of the differences is that logic programming models have two explicit values (indicated by membership of the sets T and F) whereas conclusion labellings have three values (the labels in, out and undec). Converting between conclusion labellings and logic programming models can be done using the two functions ConcLab2Mod and Mod2ConcLab [16,47]. The former function converts a conclusion labelling to a logic programming model and is defined as $\texttt{ConcLab2Mod}(\texttt{ConcLab}) = (\texttt{in}(\texttt{ConcLab}), \texttt{out}(\texttt{ConcLab}))$. The latter function converts a logic programming model to a conclusion labelling and is defined as $\texttt{Mod2ConcLab}((T, F)) = (T, F, HB_P \setminus (T \cup F))$. As mentioned in [16], ConcLab2Mod and Mod2ConcLab are bijective functions that are each other's inverse, making conclusion labellings and logic programming models one-to-one related.

It has been shown in [16,47] that complete conclusion labellings coincide with 3-valued stable models.

Theorem 1 ([16,47]). *Let P be a logic program and let $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. It holds that:*

(1) *if* Mod *is a 3-valued stable model of P then* Mod2ConcLab(Mod) *is a complete conclusion labelling of P*
(2) *if* ConcLab *is a complete conclusion labelling of P then* ConcLab2Mod(ConcLab) *is a 3-valued stable model of P*

Now that we observed the correspondence between 3-valued stable models and complete conclusion labellings, we turn to the question of what are the conclusion labellings that correspond to well-founded, regular, 2-valued stable, L-stable, pre-ideal, ideal, pre-eager and eager models.

Definition 13. Let P be a logic program and let $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. Let ConcLab be a conclusion labelling of P. ConcLab is called:

(1) the *conc-grounded conclusion labelling* iff ConcLab is a complete conclusion labelling of P where in(ConcLab) is minimal (w.r.t. $\subseteq$) among all complete conclusion labellings of P
(2) a *conc-preferred conclusion labelling* iff ConcLab is a complete conclusion labelling of P where in(ConcLab) is maximal (w.r.t. $\subseteq$) among all complete conclusion labellings of P
(3) a *conc-stable conclusion labelling* iff ConcLab is a complete conclusion labelling of P where $\texttt{undec}(\texttt{ConcLab}) = \emptyset$
(4) a *conc-semi-stable conclusion labelling* iff ConcLab is a complete conclusion labelling of P where undec(ConcLab) is minimal (w.r.t. $\subseteq$) among all complete conclusion labellings of P

It is not difficult to see that the conclusion labellings of Definition 13 are one-to-one related to the logic programming models of Definition 8.[4]

[4]This has been observed informally in [16, Section 7.1]. However, for current purposes we would like to make this explicit in the form of a theorem with associated proof.

Theorem 2. *Let P be a logic program and let $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. It holds that:*

(1) *if* `Mod` *is the well-founded model of P then* `Mod2ConcLab(Mod)` *is the conc-grounded conclusion labelling of P*
(2) *if* `ConcLab` *is the conc-grounded conclusion labelling of P then* `ConcLab2Mod(ConcLab)` *is the well-founded model of P*
(3) *if* `Mod` *is a regular model of P then* `Mod2ConcLab(Mod)` *is a conc-preferred conclusion labelling of P*
(4) *if* `ConcLab` *is a conc-preferred conclusion labelling of P then* `ConcLab2Mod(ConcLab)` *is a regular model of P*
(5) *if* `Mod` *is a 2-valued stable model of P then* `Mod2ConcLab(Mod)` *is a conc-stable conclusion labelling of P*
(6) *if* `ConcLab` *is a conc-stable conclusion labelling of P then* `ConcLab2Mod(ConcLab)` *is a 2-valued stable model of P*
(7) *if* `Mod` *is an L-stable model of P then* `Mod2ConcLab(Mod)` *is a conc-semi-stable conclusion labelling of P*
(8) *if* `ConcLab` *is a conc-semi-stable conclusion labelling of P then* `ConcLab2Mod(ConcLab)` *is an L-stable model of P*

Proof. (1) Let `Mod` be the well-founded model of P. Therefore, using point 2 of Definition 8, `Mod` is a 3-valued stable model where T is minimal among all 3-valued stable models of P. Let `ConcLab` = `Mod2ConcLab(Mod)`. It was proved in Theorem 1 that the 3-valued stable models of P correspond to the complete conclusion labellings of P. Therefore, `ConcLab` is a complete conclusion labelling of P, where `in(ConcLab)` is minimal among all complete conclusion labellings of P. Therefore, using point 1 of Definition 13, it follows that `ConcLab` is the conc-grounded conclusion labelling of P. That is, `Mod2ConcLab(Mod)` is the conc-grounded conclusion labelling of P.

(2) Let `ConcLab` be the conc-grounded conclusion labelling of P. Therefore, using point 1 of Definition 13 `ConcLab` is a complete conclusion labelling of P where `in(ConcLab)` is minimal among all complete conclusion labelling of P. Let `Mod` = `ConcLab2Mod(ConcLab)`. By Theorem 1, `Mod` is a 3-valued stable model where T is minimal among all 3-valued stable models of P. Therefore, using point 2 of Definition 8, it follows that `Mod` is the well founded model of P. That is, `ConcLab2Mod(ConcLab)` is the well founded model of P.

The proofs for the remaining parts of Theorem 2 are each analogous to one of the above. □

From Thereom 1 and Theorem 2 it follows that each logic program has one or more complete conclusion labellings, a unique conc-grounded conclusion labelling, one or more conc-preferred conclusion labellings, zero or more conc-stable conclusion labellings and one or more conc-semi-stable conclusion labellings.[5]

The way of defining conclusion labellings as stated in Definition 13 is fundamentally different from what is done by most argumentation formalisms (e.g. ASPIC+ [31–33], ABA [9,20,44] and logic-based argumentation [8,26]. The following definition is meant to reflect the process by which these formalisms derive their conclusions.

[5]Please notice that conc-grounded, conc-preferred, conc-stable and conc-semi-stable conclusion labellings were respectively called well-founded, regular, stable and L-stable conclusion labellings in [16].

Definition 14. Let P be a logic program and let $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. Let ConcLab be a conclusion labelling of P. ConcLab is called:

(1) the *arg-grounded conclusion labelling* iff ConcLab = ArgLab2ConcLab(ArgLab) where ArgLab is the grounded argument labelling of AF_P
(2) an *arg-preferred conclusion labelling* iff ConcLab = ArgLab2ConcLab(ArgLab) where ArgLab is a preferred argument labelling of AF_P
(3) an *arg-stable conclusion labelling* iff ConcLab = ArgLab2ConcLab(ArgLab) where ArgLab is a stable argument labelling of AF_P
(4) an *arg-semi-stable conclusion labelling* iff ConcLab = ArgLab2ConcLab(ArgLab) where ArgLab is a semi-stable argument labelling of AF_P

Definition 13 and Definition 14 allow us to observe the fundamental difference between logic programming and (instantiated) argumentation. Logic programming, in essence, does the maximization and the minimization on the *conclusion* level. That is, it (conceptually) takes all complete argument labellings, converts these to conclusion labellings, and then selects the maximal/minimal among these. Instantiated argumentation, on the other hand, does the maximization and minimization on the *argument* level. That is, it (conceptually) takes all complete argument labellings, selects the maximal/minimal among these, and then converts these to conclusion labellings. So whereas logic programming does the maximization/minimization *after* converting argument labellings to conclusion labellings, instantiated argumentation does the maximization/minimization *before* converting argument labellings to conclusion labellings.[6]

In order to assess whether logic programming semantics coincide with argumentation semantics, one therefore has to assess 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. Several such correspondences have been observed in [16]. An overview is provided by Theorem 3.

Theorem 3 ([16]). *Let P be a logic program and let $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. Let* ConcLab *be a conclusion labelling of P. It holds that:*

(1) ConcLab *is the conc-grounded conclusion labelling iff* ConcLab *is the arg-grounded conclusion labelling*
(2) ConcLab *is a conc-preferred conclusion labelling iff* ConcLab *is an arg-preferred conclusion labelling*
(3) ConcLab *is a conc-stable conclusion labelling iff* ConcLab *is an arg-stable conclusion labelling*

These results imply that:

- Well-founded semantics for logic programming coincides with grounded semantics for formal argumentation. That is:
  - if Mod is the well-founded model of P then Mod2ConcLab(Mod) is the arg-grounded conclusion labelling of P
  - if ConcLab is the arg-grounded conclusion labelling of P then ConcLab2Mod(ConcLab) is the well-founded model of P

[6] It has to be mentioned that formalisms such as ASPIC+ [31–33], ABA [9,20,44] and logic-based argumentation [8,26] have been stated in terms of extensions instead of in terms of labellings. However, as extensions and labellings coincide [10,13,14] they could be viewed in terms of labellings as well.

- Regular semantics for logic programming coincides with preferred semantics for formal argumentation. That is:
  - ∗ if Mod is a regular model of P then Mod2ConcLab(Mod) is an arg-preferred conclusion labelling of P
  - ∗ if ConcLab is an arg-preferred conclusion labelling of P then ConcLab2Mod(ConcLab) is a regular model of P
- 2-valued stable semantics for logic programming coincides with stable semantics for formal argumentation. That is:
  - ∗ if Mod is a 2-valued stable model of P then Mod2ConcLab(Mod) is an arg-stable conclusion labelling of P
  - ∗ if ConcLab is an arg-stable conclusion labelling of P then ConcLab2Mod(ConcLab) is a 2-valued stable model of P

Please notice that conc-semi-stable conclusion labellings do *not* coincide with arg-semi-stable conclusion labellings. That is, minimizing undec on the conclusion level yields different results than minimizing undec on the argument level. A counter example (taken from [16]) is provided in Section 4. As a result of this, L-stable semantics for logic programming is *not* the same as semi-stable semantics for formal argumentation. We refer to [16] for details.

In general, proving that for a particular semantics *sem* the conc-*sem* conclusion labellings coincide with the arg-*sem* conclusion labellings is not a trivial affair. Moreover (as is for instance the case when *sem* is semi-stable) the correspondence can turn out not to hold.

3. The equivalence between logic programming and formal argumentation under ideal semantics

In the current paper, we ask ourselves whether the correspondence between logic programming and (instantiated) argumentation also holds under ideal semantics. That is, does the ideal model of a logic program P coincide with the associated conclusion labelling of the ideal argument labelling of AF_P? Before we can answer to this question, we first need to define ideal semantics in the context of formal argumentation and logic programming.

Definition 15. Let ArgLab be an argument labelling of argumentation framework $AF = (Ar, att)$. ArgLab is called:

- a *pre-ideal argument labelling*[7] iff ArgLab is a complete argument labelling where in(ArgLab)$\subseteq$ in(ArgLab_{pr}) for each preferred argument labelling ArgLab_{pr} of AF
- the *ideal argument labelling* iff ArgLab is a pre-ideal argument labelling where in(ArgLab) is maximal (w.r.t. $\subseteq$) among all pre-ideal argument labellings of AF

Each argumentation framework has one or more pre-ideal argument labellings and a unique ideal argument labelling [15]. The pre-ideal argument labelling where in(ArgLab) is minimal (w.r.t. $\subseteq$) among all pre-ideal argument labellings of AF is the grounded labelling. For the purposes of the current

[7] A similar concept is also present in Dung et. al. [25] where the ideal extension is defined in terms of *ideal sets*. We opted for a slightly narrower concept based on the complete semantics (instead of the admissible semantics) for our convenience. This is merely a matter of choice, since the ideal extension of an argumentation framework is necessarily complete [25].

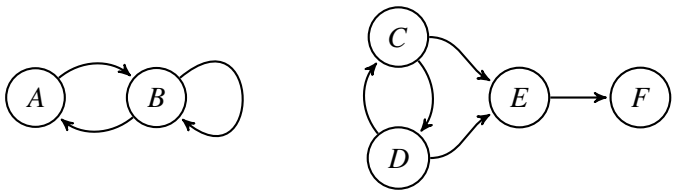

Fig. 1. Argumentation framework AF_1.

paper, what matters is the ideal argument labelling; the notion of a pre-ideal argument labelling merely serves as an auxiliary concept, to support the construction of some of the proofs. Our definition of the ideal argument labelling is based on [13,15] where it is observed that it coincides with the extension-based definition of ideal semantics of [25].

The example bellow should help the reader differentiate the relevant semantics.

Example 1 (Based on [25])**.** The argumentation framework $AF_1 = (\{A, B, C, D, E, F\}, \{(A, B), (B, A), (B, B), (C, D), (C, E), (D, C), (D, E), (E, F)\})$ (Fig. 1) has the following six complete labellings:

$$\texttt{ArgLab}_1 = \big(\{A\}, \{B\}, \{C, D, E, F\}\big)$$
$$\texttt{ArgLab}_2 = \big(\{A, C, F\}, \{B, D, E\}, \emptyset\big)$$
$$\texttt{ArgLab}_3 = \big(\{A, D, F\}, \{B, C, E\}, \emptyset\big)$$
$$\texttt{ArgLab}_4 = \big(\emptyset, \emptyset, \{A, B, C, D, E, F\}\big)$$
$$\texttt{ArgLab}_5 = \big(\{C, F\}, \{D, E\}, \{A, B\}\big)$$
$$\texttt{ArgLab}_6 = \big(\{D, F\}, \{C, E\}, \{A, B\}\big)$$

Of these, only $\texttt{ArgLab}_2$ and $\texttt{ArgLab}_3$ are preferred. This means that only $\texttt{ArgLab}_1$ and $\texttt{ArgLab}_4$ are pre-ideal and that $\texttt{ArgLab}_1$ is the ideal argument labelling of AF_1. Notice also that here $\texttt{in}(\texttt{ArgLab}_1) \subsetneq \texttt{in}(\texttt{ArgLab}_2) \cap \texttt{in}(\texttt{ArgLab}_3)$ and that $\texttt{in}(\texttt{ArgLab}_1) \supsetneq \texttt{in}(\texttt{ArgLab}_4)$, where $\texttt{ArgLab}_4$ is the grounded labelling of AF_1.

Given the definition of the pre-ideal and ideal argument labellings, defining the associated arg-pre-ideal and arg-ideal conclusion labellings is straightforward.

Definition 16. Let P be a logic program and let $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. Let ConcLab be a conclusion labelling of P. ConcLab is called:

(1) an *arg-pre-ideal conclusion labelling* iff ConcLab = ArgLab2ConcLab(ArgLab) where ArgLab is a pre-ideal argument labelling of AF_P
(2) the *arg-ideal conclusion labelling* iff ConcLab = ArgLab2ConcLab(ArgLab) where ArgLab is the ideal argument labelling of AF_P

The concept of ideal semantics for logic programming was originally introduced by Alferes *et al.* in terms of *scenaria* [2]. The approach of [2] has been reformulated (in an equivalent way) in terms of models in [17, Appendix B]. This reformulation is used for our current characterisation of ideal semantics for logic programming.

Definition 17. Let P be a logic program and $\texttt{Mod} = \langle T, F \rangle$ be a 3-valued interpretation of P w.r.t. HB_P. We say that Mod is:

(1) a *pre-ideal model* iff Mod is a 3-valued stable model where $T \subseteq T_{reg}$ for each regular model $\texttt{Mod}_{reg} = \langle T_{reg}, F_{reg} \rangle$ of P
(2) the *ideal model*[8] iff Mod is a pre-ideal model where T is maximal (w.r.t. $\subseteq$) among all pre-ideal models of P

Each logic program has one or more pre-ideal models and a unique ideal model. The pre-ideal models correspond to the Saccà and Zaniolo's deterministic models [38], so the above characterization of the ideal model for logic programs corresponds to the MD-stable model of [38]. Furthermore, the pre-ideal model where T is minimal corresponds to the well-founded model [38]. For the purposes of the current paper, what matters is the ideal model; the notion of a pre-ideal model merely serves as an auxiliary concept, to support the construction of some of the proofs.

The next concepts to be defined are those of a conc-pre-ideal conclusion labelling and that of the conc-ideal conclusion labelling.

Definition 18. Let P be a logic program and let $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. Let ConcLab be a conclusion labelling of P. ConcLab is called:

(1) a *conc-pre-ideal conclusion labelling* iff ConcLab is a complete conclusion labelling of P where $\texttt{in(ConcLab)} \subseteq \texttt{in}(\texttt{ConcLab}_{cpref})$ for each conc-preferred conclusion labelling $\texttt{ConcLab}_{cpref}$ of P
(2) the *conc-ideal conclusion labelling* iff ConcLab is a conc-pre-ideal conclusion labelling of P where in(ConcLab) is maximal (w.r.t. $\subseteq$) among all conc-pre-ideal conclusion labellings of P

It turns out that the conc-pre-ideal conclusion labellings (resp. the ideal conclusion labelling) and the pre-ideal models (resp. the ideal model) are one-to-one related.

Theorem 4. *Let P be a logic program and let $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. It holds that:*

(1) *if* Mod *is a pre-ideal model of P then* Mod2ConcLab(Mod) *is a conc-pre-ideal conclusion labelling of P*
(2) *if* ConcLab *is a conc-pre-ideal conclusion labelling of P then* ConcLab2Mod(ConcLab) *is a pre-ideal model of P*
(3) *if* Mod *is the ideal model of P then* Mod2ConcLab(Mod) *is the conc-ideal conclusion labelling of P*
(4) *if* ConcLab *is the conc-ideal conclusion labelling of P then* ConcLab2Mod(ConcLab) *is the ideal model of P*

[8] In logic programming literature, this concept corresponds to Saccà and Zaniolo's MD-stable (maximally-deterministic stable) model [38], which are defined in terms of their strongly-founded models. In their terminology, the unique MD-stable model of a logic program is the maximal strongly-founded model that is contained in every partial stable model (concerning only T). The correspondence of the MD-stable semantics for logic programs to the ideal semantics for abstract argumentation frameworks is straightforward because the strongly-founded models of [38] correspond to Przymusinski's three-valued stable models and the partial stable models of [38] correspond to the regular models of a program [29].

Proof. (1) Let Mod be a pre-ideal model of P. Therefore, using point 1 of Definition 17, Mod is a 3-valued stable model where $T \subseteq T_{reg}$ for each regular model of P. Let ConcLab = Mod2ConcLab(Mod). It has been proven in Theorem 2 that the regular models of P correspond to the conc-preferred conclusion labellings of P. This follows that ConcLab is a complete conclusion labelling of P, where in(ConcLab) $\subseteq$ in(ConcLab_{cpref}) for each conc-preferred conclusion labelling of P. Therefore, using point 1 of Definition 18, it follows that ConcLab is the conc-pre-ideal conclusion labelling of P. That is, Mod2ConcLab(Mod) is the conc-pre-ideal conclusion labelling of P.

(2) Let ConcLab be a conc-pre-ideal conclusion labelling of P. Therefore, using point 1 of Definition 18, ConcLab is a complete conclusion labelling of P where in(ConcLab) $\subseteq$ in(ConcLab_{cpref}) for each conc-preferred conclusion labelling of P. Further, Let Mod = ConcLab2Mod(ConcLab). It has been proven in Theorem 2 that the regular models of P correspond to conc-preferred conclusion labellings of P. Therefore, Mod is a 3-valued stable model where $T \subseteq T_{reg}$ for each regular model of P. Therefore, using point 1 of Definition 17, it follows that Mod is the pre-ideal model of P. That is, ConcLab2Mod(ConcLab) is the pre-ideal model of P.

(3) Let Mod be the ideal model of P. Therefore, using point 2 of Definition 17, Mod is a pre-ideal model where T is maximal among all pre-ideal models of P. Let ConcLab = Mod2ConcLab(Mod). Using points 1 and 2 of this theorem, it has been proven that the pre-ideal models of P correspond to the conc-pre-ideal conclusion labellings of P. It follows that ConcLab is a conc-pre-ideal conclusion labelling of P where in(ConcLab) is maximal among all conc-pre-ideal conclusion labellings of P. Therefore, using point 2 of Definition 18, it follows that ConcLab is the conc-ideal conclusion labelling of P. That is, Mod2ConcLab(Mod) is the conc-ideal conclusion labelling of P.

(4) Let ConcLab be the conc-ideal conclusion labelling of P. Therefore, using point 2 of Definition 18, ConcLab is a conc-pre-ideal conclusion labelling of P where in(ConcLab) is maximal among all conc-pre-ideal conclusion labelling of P. Let Mod = ConcLab2Mod(ConcLab). Using points 1 and 2 of this Theorem, it has been proven that the pre-ideal model of P correspond to the conc-pre-ideal conclusion labelling of P. Therefore, Mod is a pre-ideal model where T is maximal among all pre-ideal models of P. Therefore, using point 1 of Definition 17, it follows that Mod is the ideal model of P. That is ConcLab2Mod(ConcLab) is the ideal model of P. □

From the fact that there exists one or more pre-ideal models and a unique ideal model, it follows from Theorem 4 that there exist one or more conc-pre-ideal conclusion labellings and a unique conc-ideal conclusion labelling.

The next step will be to show the equivalence between the conc-pre-ideal (resp. conc-ideal) conclusion labellings and the arg-pre-ideal (resp. arg-ideal) conclusion labellings. In order to do so, we need to state the following lemma from [16].

Lemma 5 ([16]). *Let P be a logic program and AF_P be its associated argumentation framework. Let* ArgLab_1 *and* ArgLab_2 *be complete argument labellings of AF_P. Let* $\text{ConcLab}_1 = \text{ArgLab2ConcLab}(\text{ArgLab}_1)$ *and* $\text{ArgLab}_2 = \text{ArgLab2ConcLab}(\text{ArgLab}_2)$. *It holds that:*

(1) $\text{in}(\text{ArgLab}_1) \subseteq \text{in}(\text{ArgLab}_2)$ *iff* $\text{in}(\text{ConcLab}_1) \subseteq \text{in}(\text{ConcLab}_2)$
(2) $\text{in}(\text{ArgLab}_1) = \text{in}(\text{ArgLab}_2)$ *iff* $\text{in}(\text{ConcLab}_1) = \text{in}(\text{ConcLab}_2)$

We are now ready to state and prove one of the main theorems of this paper.

Theorem 6. *Let* ConcLab *be a conclusion labelling of logic program* P*. It holds that:*

(1) ConcLab *is an arg-pre-ideal conclusion labelling of* P *iff* ConcLab *is a conc-pre-ideal conclusion labelling of* P
(2) ConcLab *is the arg-ideal conclusion labelling of* P *iff* ConcLab *is the conc-ideal conclusion labelling of* P

Proof.

(1) "$\Longrightarrow$":
Let ConcLab be an arg-pre-ideal conclusion labelling of P. This means there exists a complete argument labelling ArgLab, with ConcLab = ArgLab2ConcLab(ArgLab), such that in(ArgLab) $\subseteq$ in(ArgLab_{pref}) for each preferred argument labelling ArgLab_{pref} of AF_P. From Lemma 5 (point 1) it follows that in(ConcLab) $\subseteq$ in($\text{ConcLab}_{argpref}$) for each arg-preferred conclusion labelling $\text{ConcLab}_{argpref}$ of P. As each arg-preferred conclusion labelling is also a conc-preferred conclusion labelling and vice versa (point 2 of Theorem 3) it then follows that in(ConcLab) $\subseteq$ in($\text{ConcLab}_{concpref}$) for each conc-preferred labelling $\text{ConcLab}_{concpref}$ of P. Hence, ConcLab is a conc-pre-ideal conclusion labelling of P.
"$\Longleftarrow$":
Let ConcLab be a conc-pre-ideal conclusion labelling of P. This means that ConcLab is a complete conclusion labelling s.t. in(ConcLab) $\subseteq$ in($\text{ConcLab}_{concpref}$) for each conc-preferred conclusion labelling $\text{ConcLab}_{concpref}$ of P. As each conc-preferred conclusion labelling is also an arg-preferred conclusion labelling and vice versa (point 2 of Theorem 3), it follows that in(ConcLab) $\subseteq$ in($\text{ConcLab}_{argpref}$) for each arg-preferred conclusion labelling $\text{ConcLab}_{argpref}$ of P. Let ArgLab be such that ConcLab = ArgLab2ConcLab(ArgLab). From Lemma 5 (point 1), it follows that in(ArgLab) $\subseteq$ in(ArgLab_{pref}) for each preferred argument labelling of AF_P. This means that ArgLab is a pre-ideal argument labelling of AF_P. Hence, ConcLab is an arg-pre-ideal conclusion labelling of P.

(2) "$\Longrightarrow$":
Let ConcLab be the arg-ideal conclusion labelling of P. This means that there is an argument labelling ArgLab with ConcLab = ArgLab2ConcLab(ArgLab) such that ArgLab is the ideal argument labelling of AF_P. That is, ArgLab is a pre-ideal argument labelling of AF_P of which in(ArgLab) is maximal among all pre-ideal argument labellings of AF_P. So for each pre-ideal argument labelling $\text{ArgLab}_{preideal}$ of AF_P, it holds that if in($\text{ArgLab}_{preideal}$) $\supseteq$ in(ArgLab) then $\text{ArgLab}_{preideal}$ = ArgLab. From Lemma 5 it follows that for each arg-pre-ideal conclusion labelling $\text{ConcLab}_{argpreideal}$ of P it holds that if in($\text{ConcLab}_{argpreideal}$) $\supseteq$ in(ConcLab) then $\text{ConcLab}_{argpreideal}$ = ConcLab. As each arg-pre-ideal conclusion labelling is also a conc-pre-ideal conclusion labelling and vice versa (point (1) of the current theorem) it follows that ConcLab is a conc-pre-ideal conclusion labelling (as ConcLab is an arg-pre-ideal labelling, since ArgLab is a pre-ideal labelling) such that for each conc-pre-ideal labelling $\text{ConcLab}_{concpreideal}$ it holds that if in($\text{ConcLab}_{concpreideal}$) $\supseteq$ in(ConcLab) then $\text{ConcLab}_{concpreideal}$ = ConcLab. That is, ConcLab is a conc-pre-ideal conclusion labelling where in(ConcLab) is maximal (w.r.t. $\subseteq$) among all conc-pre-ideal conclusion labellings. Hence, ConcLab is the conc-ideal conclusion labelling of P.

"$\Longleftarrow$":
Let ConcLab be the conc-ideal conclusion labelling of P. This means that ConcLab is a conc-pre-ideal conclusion labelling where in(ConcLab) is maximal (w.r.t. $\subseteq$) among all conc-pre-ideal conclusion labellings. That is, ConcLab is a conc-pre-ideal conclusion labelling such that for each conc-pre-ideal conclusion labelling $\texttt{ConcLab}_{concpreideal}$ it holds that if $\texttt{in}(\texttt{ConcLab}_{concpreideal}) \supseteq \texttt{in}(\texttt{ConcLab})$ then $\texttt{ConcLab}_{concpreideal} = \texttt{ConcLab}$. From point (1) of the current theorem, it follows that ConcLab is an arg-pre-ideal conclusion labelling such that for each arg-pre-ideal conclusion labelling $\texttt{ConcLab}_{argpreideal}$ it holds that if $\texttt{in}(\textit{ConcLab}_{argpreideal}) \supseteq \texttt{in}(\texttt{ConcLab})$ then $\texttt{ConcLab}_{argpreideal} = \texttt{ConcLab}$. Let ArgLab be such that $\texttt{ConcLab} = \texttt{ArgLab2ConcLab}(\texttt{ArgLab})$. From Lemma 5 it follows that for each pre-ideal argument labelling $\texttt{ArgLab}_{preideal}$ it holds that if $\texttt{in}(\texttt{ArgLab}_{preideal}) \supseteq \texttt{in}(\texttt{ArgLab})$ then $\texttt{ArgLab}_{preideal} = \texttt{ArgLab}$. That is, ArgLab is a pre-ideal argument labelling where in(ArgLab) is maximal among all pre-ideal argument labellings. That is, ArgLab is the ideal argument labelling. Hence, ConcLab is the arg-ideal conclusion labelling of P. □

We are now ready to formally state the correspondence between ideal semantics for formal argumentation and ideal semantics for logic programming.

Theorem 7. *Let* Mod *be the ideal model of logic program* P *and* ArgLab *be the ideal argument labelling of* AF_P. *It holds that*

(1) $\texttt{Mod} = \texttt{ConcLab2Mod}(\texttt{ArgLab2ConcLab}(\texttt{ArgLab}))$
(2) $\texttt{ArgLab} = \texttt{ConcLab2ArgLab}(\texttt{Mod2ConcLab}(\texttt{Mod}))$

Proof. (1) Let $\texttt{ConcLab} = \texttt{ArgLab2ConcLab}(\texttt{ArgLab})$. It then holds (Definition 16) that ConcLab is the arg-ideal conclusion labelling. From point (2) of Theorem 6 it follows that ConcLab is also the conc-ideal conclusion labelling. From point (4) of Theorem 4 it follows that ConcLab2Mod(ConcLab) is the ideal model of P. As P has a unique ideal model, $\texttt{Mod} = \texttt{ConcLab2Mod}(\texttt{ConcLab})$. That is, $\texttt{Mod} = \texttt{ConcLab2Mod}(\texttt{ArgLab2ConcLab}(\texttt{ArgLab}))$.
(2) Let $\texttt{ConcLab} = \texttt{Mod2ConcLab}(\texttt{Mod})$. From point (3) of Theorem 4 it follows that ConcLab is the conc-ideal conclusion labelling of P. Hence, by Definition 12 and point (2) of Definition 16, ConcLab2ArgLab(ConcLab) is the ideal argument labelling of AF_P. As the ideal argument labelling is unique, it follows that $\texttt{ArgLab} = \texttt{ConcLab2ArgLab}(\texttt{ConcLab})$. That is, $\texttt{ArgLab} = \texttt{ConcLab2ArgLab}(\texttt{Mod2ConcLab}(\texttt{Mod}))$. □

4. The difference between logic programming and formal argumentation under eager semantics

The eager semantics for formal argumentation was first introduced in [11], where it was defined in terms of extensions. For current purposes, we apply the equivalent notion of eager semantics from [13, 15], where it is defined in terms of argument labellings. Roughly, the idea is that instead of selecting the maximal complete labelling that is part of every preferred labelling (as is the case for ideal semantics) one selects the maximal complete labelling that is part of every semi-stable labelling.

Definition 19. Let ArgLab be an argument labelling of argumentation framework $AF = (Ar, att)$. ArgLab is called:

- a *pre-eager argument labelling* iff ArgLab is a complete argument labelling where in(ArgLab) $\subseteq$ in($\texttt{ArgLab}_{sem}$) for each semi-stable argument labelling $\texttt{ArgLab}_{sem}$ of AF
- the *eager argument labelling* iff ArgLab is a pre-eager argument labelling where in(ArgLab) is maximal (w.r.t. $\subseteq$) among all pre-eager argument labellings of AF

Each argumentation framework has one or more pre-eager argument labellings and a unique eager argument labelling [13]. Every pre-ideal argument labelling is also a pre-eager argument labelling, but the opposite does not hold. Finally, the pre-eager argument labelling where in(ArgLab) is minimal (w.r.t. $\subseteq$) among all pre-eager argument labellings of AF is the grounded labelling. For the purposes of the current paper, what matters is the eager argument labelling; the notion of a pre-eager argument labelling merely serves as an auxiliary concept to support the construction of some of the proofs.

Given an argumentation framework AF, its ideal argument labelling $\texttt{ArgLab}_{id}$ and its eager argument labelling $\texttt{ArgLab}_{ea}$, it can be observed that $\texttt{in}(\texttt{ArgLab}_{id}) \subseteq \texttt{in}(\texttt{ArgLab}_{ea})$ (as well as $\texttt{out}(\texttt{ArgLab}_{id}) \subseteq \texttt{out}(\texttt{ArgLab}_{ea})$). For formal argumentation, eager semantics can therefore be seen as more credulous than ideal semantics.

The example below should help the reader differentiate eager from the other relevant semantics.

Example 2. The argumentation framework $AF_2 = (\{A, B, C, D, E\}, \{(A, B), (B, A), (B, C), (C, D), (D, E), (E, C)\})$ (depicted in Fig. 2) has the following three complete labellings:

$$\texttt{ArgLab}_1 = \big(\emptyset, \emptyset, \{A, B, C, D, E\}\big)$$
$$\texttt{ArgLab}_2 = \big(\{A\}, \{B\}, \{C, D, E\}\big)$$
$$\texttt{ArgLab}_3 = \big(\{B, D\}, \{A, C, E\}, \emptyset\big)$$

Of these, $\texttt{ArgLab}_1$ is the grounded labelling, while $\texttt{ArgLab}_2$ and $\texttt{ArgLab}_3$ are both preferred, and only $\texttt{ArgLab}_3$ is semi-stable. It follows that only $\texttt{ArgLab}_1$ is pre-ideal, while both $\texttt{ArgLab}_1$ and $\texttt{ArgLab}_3$ are pre-eager. Therefore, $\texttt{ArgLab}_1$ is the ideal argument labelling of AF_2 and $\texttt{ArgLab}_3$ is the eager argument labelling of AF_2.

Given the definition of the pre-eager and eager argument labellings, defining the associated arg-pre-eager and arg-eager conclusion labellings is straightforward.

Definition 20. Let P be a logic program and let $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. Let ConcLab be a conclusion labelling of P. ConcLab is called:

(1) an *arg-pre-eager conclusion labelling* iff ConcLab = ArgLab2ConcLab(ArgLab) where ArgLab is a pre-eager argument labelling of AF_P

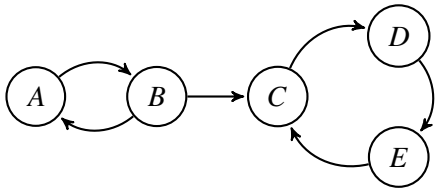

Fig. 2. Argumentation framework AF_2.

(2) the *arg-eager conclusion labelling* iff ConcLab = ArgLab2ConcLab(ArgLab) where ArgLab is the eager argument labelling of AF_P

In the context of logic programming, a concept similar to eager semantics has, as far as we are aware, not been defined before. In the current paper, we define it in a similar way as ideal semantics for logic programming, with the difference that instead of being based on regular models (as for the ideal model) it is based on L-stable models.

Definition 21. Let P be a logic program and $\mathtt{Mod} = \langle T, F\rangle$ be a 3-valued interpretation of P w.r.t. HB_P. We say that Mod is:

(1) a *pre-eager model* iff Mod is a 3-valued stable model where $T \subseteq T_{lstab}$ for each L-stable model $\mathtt{Mod}_{lstab} = \langle T_{lstab}, F_{lstab}\rangle$ of P
(2) the *eager model* iff Mod is a pre-eager model where T is maximal (w.r.t. $\subseteq$) among all pre-eager models of P

Each logic program has one or more pre-eager models and a unique eager model. The pre-eager model where T is minimal corresponds to the well-founded model. For the purposes of the current paper, what matters is the eager model; the notion of a pre-eager model merely serves as an auxiliary concept.

Given a logic program P, its ideal model $\mathtt{Mod}_{id} = \langle T_{id}, F_{id}\rangle$ and its eager model $\mathtt{Mod}_{ea} = \langle T_{ea}, F_{ea}\rangle$, it can be observed that $T_{id} \subseteq T_{ea}$ (as well as $F_{id} \subseteq F_{ea}$). For logic programming, eager semantics can therefore be seen as more credulous than ideal semantics.

The next concepts to be defined are those of a conc-pre-eager conclusion labelling and that of the conc-eager conclusion labelling.

Definition 22. Let P be a logic program and let $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. Let ConcLab be a conclusion labelling of P. ConcLab is called:

(1) a *conc-pre-eager conclusion labelling* iff ConcLab is a complete conclusion labelling of P where in(ConcLab) $\subseteq$ in($\mathtt{ConcLab}_{csem}$) for each conc-semi-stable conclusion labelling $\mathtt{ConcLab}_{csem}$ of P
(2) the *conc-eager conclusion labelling* iff ConcLab is a conc-pre-eager conclusion labelling of P where in(ConcLab) is maximal (w.r.t. $\subseteq$) among all conc-pre-eager conclusion labellings of P

It turns out that the conc-pre-eager conclusion labellings (resp. the eager conclusion labelling) and the pre-eager models (resp. the eager model) are one-to-one related.

Theorem 8. *Let P be a logic program and let $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. It holds that:*

(1) *if* Mod *is a pre-eager model of P then* Mod2ConcLab(Mod) *is a conc-pre-eager conclusion labelling of P*
(2) *if* ConcLab *is a conc-pre-eager conclusion labelling of P then* ConcLab2Mod(ConcLab) *is a pre-eager model of P*
(3) *if* Mod *is the eager model of P then* Mod2ConcLab(Mod) *is the conc-eager conclusion labelling of P*
(4) *if* ConcLab *is the conc-eager conclusion labelling of P then* ConcLab2Mod(ConcLab) *is the eager model of P*

Table 1
The arguments constructed from P

	A_1	A_2	A_3	A_4	A_5	A_6
Conclusion	a	b	c	d	c	e
Vulnerabilities	$\{b\}$	$\{a\}$	$\{c, a\}$	$\{d, b\}$	$\{c, e\}$	$\{e\}$

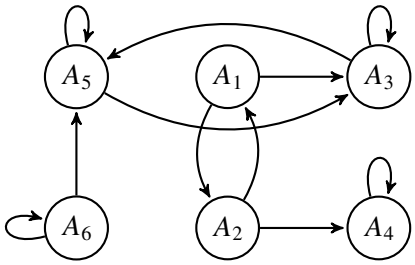

Fig. 3. Argumentation framework AF_P.

Proof. The proof is analogous to that of Theorem 4. □

From the fact that there exists one or more pre-eager models and a unique eager model, it follows from Theorem 8 that there exist one or more conc-pre-eager conclusion labellings and a unique conc-eager conclusion labelling.

Although logic programming and formal argumentation coincide for ideal semantics, they do not coincide for eager semantics. A counter example is the following.[9]

Example 3. Consider the following logic program P:

$$\mathtt{r}_1 : a \leftarrow \mathtt{not}\ b \qquad \mathtt{r}_4 : d \leftarrow \mathtt{not}\ d, \mathtt{not}\ b$$
$$\mathtt{r}_2 : b \leftarrow \mathtt{not}\ a \qquad \mathtt{r}_5 : c \leftarrow \mathtt{not}\ c, \mathtt{not}\ e$$
$$\mathtt{r}_3 : c \leftarrow \mathtt{not}\ c, \mathtt{not}\ a \qquad \mathtt{r}_6 : e \leftarrow \mathtt{not}\ e$$

Each of these rules corresponds to an argument (A_1 to A_6). See Table 1 for details. The associated argumentation framework AF_P is shown in Fig. 3.

Given logic program P and argumentation framework AF_P, three complete argument labellings with associated conclusion labellings and logic programming models can be distinguished. These are shown in Table 2.

From Table 2 it can be observed that only $\mathtt{ArgLab}_1$ and $\mathtt{ArgLab}_2$ are semi-stable argument labellings. This means that only $\mathtt{ArgLab}_3$ is a pre-eager argument labelling, and therefore $\mathtt{ArgLab}_3$ is the eager argument labelling. This means that $\mathtt{ConcLab}_3$ is the arg-eager conclusion labelling.

From Table 2 it can also be observed that only $\mathtt{ConcLab}_2$ is a conc-semi-stable conclusion labelling. Therefore, $\mathtt{ConcLab}_2$ and $\mathtt{ConcLab}_3$ are both conc-pre-eager conclusion labellings, which means that $\mathtt{ConcLab}_2$ is the conc-eager conclusion labelling.

In a similar way, it can be observed that only $\mathtt{Mod}_2$ is an L-stable model, $\mathtt{Mod}_2$ and $\mathtt{Mod}_3$ are pre-eager models, and (more importantly) that $\mathtt{Mod}_2$ is the eager model.

[9]This counter example is taken from [16] where it was used in the context of semi-stable and L-stable semantics. The example has been slightly adapted to increase readability.

Table 2
Labellings and models of P and AF_P

Complete argument labellings	Complete conclusion labellings	3-valued stable models
$\texttt{ArgLab}_1$:	$\texttt{ConcLab}_1$:	$\texttt{Mod}_1$:
$(\{A_1\}, \{A_2, A_3\}, \{A_4, A_5, A_6\})$	$(\{a\}, \{b\}, \{c, d, e\})$	$\langle\{a\}, \{b\}\rangle$
$\texttt{ArgLab}_2$:	$\texttt{ConcLab}_2$:	$\texttt{Mod}_2$:
$(\{A_2\}, \{A_1, A_4\}, \{A_3, A_5, A_6\})$	$(\{b\}, \{a, d\}, \{c, e\})$	$\langle\{b\}, \{a, d\}\rangle$
$\texttt{ArgLab}_3$:	$\texttt{ConcLab}_3$:	$\texttt{Mod}_3$:
$(\emptyset, \emptyset, \{A_1, A_2, A_3, A_4, A_5, A_6\})$	$(\emptyset, \emptyset, \{a, b, c, d, e\})$	$\langle\emptyset, \emptyset\rangle$

As such, we have observed that the eager model coincides with the conc-eager conclusion labelling.[10] However, these do *not* coincide with the arg-eager conclusion labelling. Hence, eager semantics for argumentation does not coincide with eager semantics for logic programming.

Although in general, eager semantics for logic programming does not coincide with eager semantics for formal argumentation (as shown by Example 3) there are classes of logic programs for which these semantics do coincide. This critically depends on the logic program having its arg-semi-stable conclusion labellings coincide with its conc-semi-stable conclusion labellings.

Definition 23 ([16][11])**.** A logic program is called *semi-stable-compatible* iff:

- each arg-semi-stable conclusion labelling is a conc-semi-stable conclusion labelling
- each conc-semi-stable conclusion labelling is an arg-semi-stable conclusion labelling

Example classes of logic programs that are semi-stable-compatible are AF-programs, stable programs and stratified programs [16]. We now proceed to prove that for each semi-stable-compatible program, its eager model coincides with the eager argument labelling of its associated logic program. For this, we first need the following intermediate result.

Theorem 9. *Let* ConcLab *be a conclusion labelling of a semi-stable-compatible logic program P. It holds that:*

(1) ConcLab *is an arg-pre-eager conclusion labelling iff* ConcLab *is a conc-pre-eager conclusion labelling*
(2) ConcLab *is the arg-eager conclusion labelling iff* ConcLab *is the conc-eager conclusion labelling*

Proof.

(1) "$\Longrightarrow$":
Let ConcLab be an arg-pre-eager conclusion labelling of P. This means there exists a complete argument labelling ArgLab, with ConcLab = ArgLab2ConcLab(ArgLab), such that in(ArgLab) $\subseteq$ in($\texttt{ArgLab}_{sem}$) for each semi-stable argument labelling $\texttt{ArgLab}_{sem}$ of AF_P. From Lemma 5 (point 1) it follows that in(ConcLab) $\subseteq$ in($\texttt{ConcLab}_{argsem}$) for each arg-semi-stable conclusion labelling $\texttt{ConcLab}_{argsem}$ of P. The fact that P is a semi-stable-compatible

[10] This should not come as a surprise, as it follows from Theorem 2.
[11] These programs are called semi-stable-L-stable compatible in [16]. We opted to slightly change the terminology in order for it to become more in line with the rest of the current paper.

logic program means that each arg-semi-stable conclusion labelling is also a conc-semi-stable conclusion labelling and vice versa (Definition 23). We therefore obtain that $\texttt{in(ConcLab)} \subseteq \texttt{in(ConcLab}_{concsem}\texttt{)}$ for each conc-semi-stable labelling $\texttt{ConcLab}_{concsem}$ of P. Hence, ConcLab is a conc-pre-eager conclusion labelling of P.

"$\Longleftarrow$":

Let ConcLab be a conc-pre-eager conclusion labelling of P. This means that ConcLab is a complete conclusion labelling s.t. $\texttt{in(ConcLab)} \subseteq \texttt{in(ConcLab}_{concsem}\texttt{)}$ for each conc-semi-stable conclusion labelling $\texttt{ConcLab}_{concsem}$ of P. The fact that P is a semi-stable-compatible logic program means that each conc-semi-stable conclusion labelling is also an arg-semi-stable conclusion labelling and vice versa (Definition 23). We therefore obtain that $\texttt{in(ConcLab)} \subseteq \texttt{in(ConcLab}_{argsem}\texttt{)}$ for each arg-semi-stable conclusion labelling $\texttt{ConcLab}_{argsem}$ of P. Let ArgLab be such that ConcLab = ArgLab2ConcLab(ArgLab). From Lemma 5 (point 1) it follows that $\texttt{in(ArgLab)} \subseteq \texttt{in(ArgLab}_{sem}\texttt{)}$ for each semi-stable argument labelling of AF_P. This means that ArgLab is a pre-eager argument labelling of AF_P. Hence, ConcLab is an arg-pre-eager conclusion labelling of P.

(2) "$\Longrightarrow$":

Let ConcLab be the arg-eager conclusion labelling of P. This means that there is an argument labelling ArgLab with ConcLab = ArgLab2ConcLab(ArgLab) such that ArgLab is the eager argument labelling of AF_P. That is, ArgLab is a pre-eager argument labelling of AF_P of which in(ArgLab) is maximal among all pre-eager argument labellings of AF_P. So for each pre-eager argument labelling $\texttt{ArgLab}_{preeager}$ it holds that if $\texttt{in(ArgLab}_{preeager}\texttt{)} \supseteq \texttt{in(ArgLab)}$ then $\texttt{ArgLab}_{preeager} = \texttt{ArgLab}$. From Lemma 5 it follows that for each arg-pre-eager conclusion labelling $\texttt{ConcLab}_{argpreeager}$ it holds that if $\texttt{in(ConcLab}_{argpreeager}\texttt{)} \supseteq \texttt{in(ConcLab)}$ then $\texttt{ConcLab}_{argpreeager} = \texttt{ConcLab}$. As each arg-pre-eager conclusion labelling of P is also a conc-pre-eager conclusion labelling of P and vice versa (point (1) of the current theorem) it follows that ConcLab is a conc-pre-eager conclusion labelling (as ConcLab is an arg-pre-eager labelling, since ArgLab is a pre-eager labelling) such that for each conc-pre-eager labelling $\texttt{ConcLab}_{concpreeager}$ it holds that if $\texttt{in(ConcLab}_{concpreeager}\texttt{)} \supseteq \texttt{in(ConcLab)}$ then $\texttt{ConcLab}_{concpreeager} = \texttt{ConcLab}$. That is, ConcLab is a conc-pre-eager conclusion labelling where in(ConcLab) is maximal (w.r.t. $\subseteq$) among all conc-pre-eager conclusion labellings. Hence, ConcLab is the conc-eager conclusion labelling.

"$\Longleftarrow$":

Let ConcLab be the conc-eager conclusion labelling of P. This means that ConcLab is a conc-pre-eager conclusion labelling where in(ConcLab) is maximal (w.r.t. $\subseteq$) among all conc-pre-eager conclusion labellings. That is, ConcLab is a conc-pre-eager conclusion labelling such that for each conc-pre-eager conclusion labelling $\texttt{ConcLab}_{concpreeager}$ it holds that if $\texttt{in(ConcLab}_{concpreeager}\texttt{)} \supseteq \texttt{in(ConcLab)}$ then $\texttt{ConcLab}_{concpreeager} = \texttt{ConcLab}$. From point (1) of the current theorem, it follows that ConcLab is an arg-pre-eager conclusion labelling such that for each arg-pre-eager conclusion labelling $\texttt{ConcLab}_{argpreeager}$ it holds that if $\texttt{in(}\mathit{ConcLab}_{argpreeager}\texttt{)} \supseteq \texttt{in(ConcLab)}$ then $\texttt{ConcLab}_{argpreeager} = \texttt{ConcLab}$. Let ArgLab be such that ConcLab = ArgLab2ConcLab(ArgLab). From Lemma 5 it follows that for each pre-eager argument labelling $\texttt{ArgLab}_{preeager}$ it holds that if $\texttt{in(ArgLab}_{preeager}\texttt{)} \supseteq \texttt{in(ArgLab)}$ then $\texttt{ArgLab}_{preeager} = \texttt{ArgLab}$. That is, ArgLab is a pre-eager argument labelling where in(ArgLab) is maximal among all pre-eager argument labellings. That is, ArgLab is the eager argument labelling. Hence, ConcLab is the arg-eager conclusion labelling. □

Theorem 10. *Let P be a semi-stable-compatible logic program, let* `Mod` *be the eager model of P and let* `ArgLab` *be the eager argument labelling of AF_P. It holds that*

(1) `Mod` = `ConcLab2Mod(ArgLab2ConcLab(ArgLab))`
(2) `ArgLab` = `ConcLab2ArgLab(Mod2ConcLab(Mod))`

Proof. (1) Let `ConcLab` = `ArgLab2ConcLab(ArgLab)`. It then holds (Definition 20) that `ConcLab` is the arg-eager conclusion labelling. From point (2) of Theorem 9 it follows that `ConcLab` is also the conc-eager conclusion labelling. From point (4) of Theorem 8 it follows that `ConcLab2Mod(ConcLab)` is the eager model of P. Since P has a unique eager model, `Mod` = `ConcLab2Mod(ConcLab)`. That is, `Mod` = `ConcLab2Mod(ArgLab2ConcLab(ArgLab))`.
(2) Let `ConcLab` = `Mod2ConcLab(Mod)`. From point (3) of Theorem 8 it follows that `ConcLab` is the conc-eager conclusion labelling of P. Hence, by Definition 12 and point (2) of Definition 20, `ConcLab2ArgLab(ConcLab)` is the eager argument labelling of AF_P. As the eager argument labelling is unique, it follows that `ArgLab` = `ConcLab2ArgLab(ConcLab)`. That is, `ArgLab` = `ConcLab2ArgLab(Mod2ConcLab(Mod))`. □

5. Translating argumentation frameworks to logic programs

So far, we have studied the connection between logic programming semantics and argumentation semantics using a translation *from* a logic program *to* an argumentation framework. It is also possible to go the other way around, using a translation *from* an argumentation framework *to* a logic program. For this, we use a definition from [16,47]. Basically, the idea is that a rule is created for each argument in the argumentation framework. The argument itself goes in the head of the rule and its attackers go in the body of the rule (as weakly negated statements).

Definition 24. Let $AF = (Ar, att)$ be an argumentation framework. Then the logic program associated with AF is $P_{AF} = \{A \leftarrow \texttt{not}\ B_1, \ldots, \texttt{not}\ B_m \mid \{B_i \mid (B_i, A) \in att\} = \{B_1, \ldots, B_m\}\}$.

As an example, the argumentation framework of Fig. 3 (page 109) has the following associated logic program:

$A_1 \leftarrow \texttt{not}\ A_2$
$A_2 \leftarrow \texttt{not}\ A_1$
$A_3 \leftarrow \texttt{not}\ A_1, \texttt{not}\ A_3, \texttt{not}\ A_5$
$A_4 \leftarrow \texttt{not}\ A_2, \texttt{not}\ A_4$
$A_5 \leftarrow \texttt{not}\ A_3, \texttt{not}\ A_5, \texttt{not}\ A_6$
$A_6 \leftarrow \texttt{not}\ A_6$

We first observe that this logic program is not the same as logic program P of Example 3 that generated the argumentation framework of Fig. 3. This illustrates that in general, if one translates a logic program to an argumentation framework (using Definition 11) and then translates the resulting argumentation framework back to a logic program (using Definition 24), the logic program one ends up with is not necessarily the same as what one started with.

One could also examine what happens when starting not with a logic program, but with an argumentation framework. That is, what happens when one translates an argumentation framework to a logic program (using Definition 24) and then back to an argumentation framework (using Definition 11)? It turns out that the argumentation framework one ends up with is isomorphic with the argumentation framework one started with [16, Theorem 33].

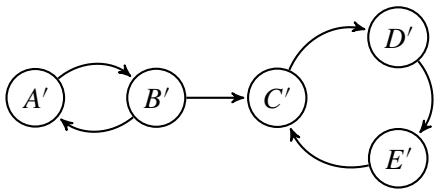

Fig. 4. Argumentation framework $AF_{P_{AF_2}}$.

To understand the nature of this isomorphism, we observe that when translating an argumentation framework AF to a logic program (Definition 11) each argument A_i will yield a corresponding rule r_i with A_i as its head and the attackers of A_i as weakly negated statements in its body. When one uses the thus obtained logic program P_{AF} to construct the associated argumentation framework $AF_{P_{AF}}$ (Definition 24) each rule r_i will form an argument of its own.

As an example, consider again the argumentation framework AF_2 that was depicted in Fig. 2 (page 107) and was previously discussed in Example 2. This argumentation framework has the following associated logic program P_{AF_2}:

$$A \leftarrow \texttt{not}\ B \qquad D \leftarrow \texttt{not}\ C$$
$$B \leftarrow \texttt{not}\ A \qquad E \leftarrow \texttt{not}\ D$$
$$C \leftarrow \texttt{not}\ B, \texttt{not}\ E$$

The logic program P_{AF_2} has the associated argumentation framework $AF_{P_{AF_2}}$ shown in Fig. 4 with:

$$A' = \text{“}A \leftarrow \texttt{not}\ B'' \qquad D' = \text{“}D \leftarrow \texttt{not}\ C''$$
$$B' = \text{“}B \leftarrow \texttt{not}\ A'' \qquad E' = \text{“}E \leftarrow \texttt{not}\ D''$$
$$C' = \text{“}C \leftarrow \texttt{not}\ B, \texttt{not}\ E''$$

It is not difficult to see that AF_2 and $AF_{P_{AF_2}}$ are isomorphic. Moreover, we observe that the complete argument labellings of AF_2 (which are $(\emptyset, \emptyset, \{A, B, C, D, E\})$, $(\{A\}, \{B\}, \{C, D, E\})$ and $(\{B, D\}, \{A, C, E\}, \emptyset)$) correspond also to the complete conclusion labellings of $AF_{P_{AF_2}}$.

Given two isomorphic argumentation frameworks, we can translate an argument labelling of one to an argument labelling of the other. This is formally defined as follows.

Definition 25. Let $AF = (Ar, att)$ and $AF' = (Ar', att')$ be isomorphic argumentation frameworks and let I be the function that defines this isomorphism. That is, $I : Ar \rightarrow Ar'$ is a function such that $(A, B) \in att$ iff $(I(A), I(B)) \in att'$. Let $\texttt{ArgLab}$ be an argument labelling of AF. We define the associated argument labelling $\texttt{ArgLab}'$ of AF' as $(\{I(A) \mid \texttt{ArgLab}(A) = \texttt{in}\}, (\{I(A) \mid \texttt{ArgLab}(A) = \texttt{out}\}, (\{I(A) \mid \texttt{ArgLab}(A) = \texttt{undec}\})$.

Definition 25 allows us to prove the following lemma.

Lemma 11. *Let $AF = (Ar, att)$ be an argumentation framework and let P_{AF} be its associated logic program (Definition 24). Let* $\texttt{ArgLab}$ *be the ideal argument labelling of AF and let* $\texttt{ConcLab}$ *be the arg-ideal conclusion labelling of P_{AF}. It holds that* $\texttt{ArgLab} = \texttt{ConcLab}$.

Proof. Let $AF_{P_{AF}}$ be the argumentation framework associated with P_{AF}. It holds that AF and $AF_{P_{AF}}$ are isomorphic [16, Theorem 33]. Let $\texttt{ArgLab}'$ be argument labelling of $AF_{P_{AF}}$ that is associated with $\texttt{ArgLab}$ under the isomorphism (Definition 25). As ideal semantics is language independent [4,5] it follows that $\texttt{ArgLab}'$ is the ideal argument labelling of $AF_{P_{AF}}$. It then follows that $\texttt{ConcLab} = \texttt{ArgLab2ConcLab}(\texttt{ArgLab}')$. In order to prove that $\texttt{ArgLab} = \texttt{ConcLab}$ we need to show the following:

- $\texttt{in}(\texttt{ArgLab}) \subseteq \texttt{in}(\texttt{ConcLab})$
  Let $A \in \texttt{in}(\texttt{ArgLab})$. Then "$A \leftarrow \texttt{not}\ B_1, \ldots, \texttt{not}\ B_m$" $\in \texttt{in}(\texttt{ArgLab}')$.
  Therefore, $A \in \texttt{in}(\texttt{ConcLab})$.
- $\texttt{out}(\texttt{ArgLab}) \subseteq \texttt{out}(\texttt{ConcLab})$
  Let $A \in \texttt{out}(\texttt{ArgLab})$. Then "$A \leftarrow \texttt{not}\ B_1, \ldots, \texttt{not}\ B_m$" $\in \texttt{out}(\texttt{ArgLab}')$.
  Therefore, $A \in \texttt{out}(\texttt{ConcLab})$.
- $\texttt{undec}(\texttt{ArgLab}) \subseteq \texttt{undec}(\texttt{ConcLab})$
  Let $A \in \texttt{undec}(\texttt{ArgLab})$. Then "$A \leftarrow \texttt{not}\ B_1, \ldots, \texttt{not}\ B_m$" $\in \texttt{undec}(\texttt{ArgLab}')$.
  Therefore, $A \in \texttt{undec}(\texttt{ConcLab})$. □

Theorem 12. *Let $AF = (Ar, att)$ be an argumentation framework and let P_{AF} be its associated logic program (in line with Definition 24). Let* $\texttt{ArgLab}$ *be the ideal argument labelling of AF and let* $\texttt{Mod} = \langle T, F \rangle$ *be the ideal model of P. It holds that:*

(1) $\texttt{Mod} = \texttt{ConcLab2Mod}(\texttt{ArgLab})$
(2) $\texttt{ArgLab} = \texttt{Mod2ConcLab}(\texttt{Mod})$

Proof. (1) As $\texttt{ArgLab}$ is the ideal argument labelling of AF, it follows from Lemma 11 that $\texttt{ArgLab}$ is also the arg-ideal conclusion labelling of P_{AF}. From point (2) of Theorem 6 it follows that $\texttt{ArgLab}$ is also a conc-ideal conclusion labelling of P_{AF}. From Theorem 4 (point 4) it follows that $\texttt{ConcLab2Mod}(\texttt{ArgLab})$ is the ideal model of P_{AF}. That is, $\texttt{ConcLab2Mod}(\texttt{ArgLab}) = \texttt{Mod}$.
(2) As $\texttt{Mod}$ is the ideal model of P_{AF} it follows that $\texttt{Mod2ConcLab}(\texttt{Mod})$ is the conc-ideal conclusion labelling of P_{AF} (Theorem 4, point 3). From Theorem 6 (point 2) it follows that $\texttt{Mod2ConcLab}(\texttt{Mod})$ is also the arg-ideal conclusion labelling of P_{AF}. From Lemma 11 it then follows that $\texttt{Mod2ConcLab}(\texttt{Mod})$ is also the ideal argument labelling of AF. That is, $\texttt{Mod2ConcLab}(\texttt{Mod}) = \texttt{ArgLab}$. □

The proof of Theorem 12 can be generalised for any argumentation semantics σ_{arg} and associated logic programming semantics σ_{lp}. As long as the σ_{lp} models of a logic program P coincide with the σ_{arg} argument labellings of AF_P,[12] it follows that the σ_{arg} argument labellings of any argumentation framework AF also coincide with the σ_{lp} models of the associated logic program P_{AF} [16].

We next focus our attention on eager semantics. As we have seen in Example 3, the eager model of a logic program does *not* necessarily coincide with the eager argument labelling of the associated argumentation framework.

At the same time, Example 3 cannot be seen as a counter example against the equivalence of eager semantics for formal argumentation and eager semantics for logic programming, when translating from formal argumentation to logic programming. That is, Example 3 does *not* show that there exists an

[12] which is the case when $\sigma_{lp} =$ *ideal* and $\sigma_{\text{arg}} =$ *ideal*, when $\sigma_{lp} =$ *3-valued stable* and $\sigma_{\text{arg}} =$ *complete*, when $\sigma_{lp} =$ *regular* and $\sigma_{\text{arg}} =$ *preferred* and when $\sigma_{lp} =$ *2-valued stable* and $\sigma_{\text{arg}} =$ *stable*

argumentation framework AF whose eager argument labelling does not coincide with the eager model of its associated logic program P_{AF}. In fact, it turns out that for each argumentation framework AF, its eager argument labelling *does* coincide with the eager model of the associated logic program P_{AF}. In order to prove so, we first observe that an argumentation framework translates into a very specific class of logic programs, called *AF-programs* [16].

Definition 26 ([16])**.** A logic program P is called an AF-program iff there is at most one rule r with $head(r) = c$ for each $c \in HB_P$.

It turns out that each AF-program is semi-stable-compatible [16, Theorem 29].

Theorem 13 ([16])**.** *Let P be an AF-program. It holds that P is semi-stable-compatible.*

Lemma 14. *Let $AF = (Ar, att)$ be an argumentation framework and let P_{AF} be its associated logic program (Definition 24). Let* `ArgLab` *be the eager argument labelling of AF and let* `ConcLab` *be the arg-eager conclusion labelling of P_{AF}. It holds that* `ArgLab` = `ConcLab`.

Proof. Similar to the proof of Lemma 11. □

Theorem 15. *Let $AF = (Ar, att)$ be an argumentation framework and let P_{AF} be its associated logic program (in line with Definition 24). Let* `ArgLab` *be the eager argument labelling of AF and let* `Mod` $= \langle T, F \rangle$ *be the eager model of P. It holds that:*

(1) `Mod` = `ConcLab2Mod(ArgLab)`
(2) `ArgLab` = `Mod2ConcLab(Mod)`

Proof. (1) As `ArgLab` is the eager argument labelling of AF, it follows from Lemma 14 that `ArgLab` is also the arg-eager conclusion labelling of P_{AF}. From point (2) of Theorem 9 it follows that `ArgLab` is also a conc-eager conclusion labelling of P_{AF}. From Theorem 8 (point 4) it follows that `ConcLab2Mod(ArgLab)` is the eager model of P_{AF}. That is, `ConcLab2Mod(ArgLab)` = `Mod`.
(2) As `Mod` is the eager model of P_{AF} it follows that `Mod2ConcLab(Mod)` is the conc-eager conclusion labelling of P_{AF} (Theorem 8, point 3). From Theorem 9 (point 2) it follows that `Mod2ConcLab(Mod)` is also the arg-eager conclusion labelling of P_{AF}. From Lemma 14 it then follows that `Mod2ConcLab(Mod)` is also the eager argument labelling of AF, That is, `Mod2ConcLab(Mod)` = `ArgLab`. □

6. Discussion

In the current paper, we extended the work of [16,47] by examining whether additional semantical correspondences hold between formal argumentation and logic programming, in particular in the context of ideal and eager semantics. For this, we made use of a recent reformulation of ideal semantics for logic programming (from [17]) that is roughly equivalent to the characterization of MD-stable models for logic programming [38], but offers a simpler presentation. This reformulation of ideal semantics for logic programming made it possible to define eager semantics for logic programming in a similar way.

Overall, we observe that when translating a logic program to an argumentation framework, the ideal model of the logic program coincides with the ideal argument labelling of the argumentation framework.

Table 3
Semantical correspondences and non-correspondences when translating from LP to argumentation

Logic programming model	Coincides	Argument labelling
3-valued stable	Yes [47]	Complete
Well-founded	Yes [16]	Grounded
Regular	Yes [16]	Preferred
L-stable	No [16]	Semi-stable
2-valued stable	Yes [16]	Stable
Ideal	Yes (Theorem 7)	Ideal
Eager	No (Example 3)	Eager

Table 4
Semantical correspondences and non-correspondences when translating from argumentation to LP

Argument labelling	Coincides	Logic programming model
Complete	Yes [47]	3-valued-stable
Grounded	Yes [16]	Well-founded
Preferred	Yes [16]	Regular
Semi-stable	Yes [16]	L-stable
Stable	Yes [16]	2-valued stable
Ideal	Yes (Theorem 12)	Ideal
Eager	Yes (Theorem 15)	Eager

However, the eager model of the logic program does not necessarily coincide with the eager argument labelling of the argumentation framework. These two findings, together with the findings of [16,47], are summarised in Table 3.

When translating an argumentation framework to a logic program, the situation is slightly different. Not only does the ideal argument labelling of the argumentation framework coincide with the ideal model of the logic program, the eager argument labelling of the argumentation framework also coincides with the eager model of the logic program. These two findings, together with the findings of [16,47], are summarised in Table 4.

Table 3 and Table 4 round off our analysis of how mainstream admissibility-based semantics (at least those mentioned in [13]) relate to their logic programming counterparts. Notice that we did not analyse any correspondences in the context of non-admissibility-based semantics, such as naive, stage, CF2 and stage2. This is because as far as we are aware, comparable semantics have not been defined in the context of logic programming, and it is not immediately clear how such could be done.

The results of the current paper (and of [16,47]) go beyond the correspondences mentioned in Table 3 and Table 4, as we essentially compared two fundamental ways in which formal argumentation theory can be used to infer conclusions. Both are based on complete semantics (which in logic programming context is called 3-valued stable semantics). One of them (the instantiated argumentation way) works by making a selection on the argument level (such as selecting those argument labellings where a particular label is minimal or maximal) and then translating to the conclusion level. The other one (the logic programming way) works (at least conceptually) by translating all argument labellings to conclusion labellings and then doing the selection on the conclusion level (such as selecting those conclusion labellings where a particular label is minimal or maximal). What Table 3 and Table 4 in essence show is in which case doing a particular selection on the argument level yields the same result as doing a comparable selection on the conclusion level. Our findings should also benefit some similar investigations

exploring semantic correspondences between logic programming semantics and other argumentation formalisms, such as [1,36,37,42].

It is worth noticing that Nieves and Osorio [34] found that the ideal semantics for argumentation frameworks could be captured by the WFS+ semantics[13] [23] for normal logic programs, but their result is based on a rather different translation from ours. Intuitively, their translation describes an argumentation framework in terms of conditions for each argument to be defeated (an argument is defeated if one or more of its attackers is not defeated; an argument is defeated if all of its potential defenders are defeated) and accepted (an argument is accepted if it is not defeated). As an example, consider the very simple argumentation framework $AF_3 = (\{A\}, \{(A, A)\})$. Using the translation of Nieves and Osorio [34], we obtain the program $\Pi_{AF_3} = \{r_1 : def(A) \leftarrow \texttt{not}\ def(A),\ r_2 : def(A) \leftarrow def(A),\ r_3 : acc(A) \leftarrow \texttt{not}\ def(A)\}$. This translation favours the use of more classical semantics obtained by replacing `not` with the classic negation. In our example, Π_{AF_3} has a single 2-valued model $\{def(A)\}$, which will also be its WFS+ model.[14] From there, one can obtain the ideal extension of AF_3 by taking the arguments that are accepted in the WFS+ model of Π_{AF_3}. In this case, there are no accepted arguments, so the ideal extension of AF_3 is empty, which is correct. In contrast, the same result cannot be replicated using our translation; we would find $P_{AF_3} = \{A \leftarrow \texttt{not}\ A\}$, for which the WFS+ model is $\{A\}$. Notice that $\{A\}$ is not even admissible for AF_3, so immediately the WFS+ semantics will differ from all admissible argumentation semantics using our translation. We believe the above advocates in favour of our approach being more straightforward, especially concerning translations. It should also be noticed that Nieves and Osorio do not consider translating logic programs into argumentation frameworks (they only translate one way, we do both) and that their approach does not properly differentiate between defeated and undecided arguments for some argumentation semantics in the resulting program. On that matter, it suffices to see that A is undecided in the ideal argument labelling of AF_3, but the WFS+ semantics for Π_{AF_3} concludes $def(A)$, which should intuitively mean that A is defeated. Instead, in their attempt to capture the ideal semantics of AF_3, $def(A)$ means only that A is not accepted, therefore finding no conclusion to whether A is defeated or undecided. In contrast, our approach is natively 3-valued, allowing us to properly state the acceptance status of all arguments in an argumentation framework in each of the semantics we mentioned.

One particular topic that is still open is how formal argumentation relates to logic programming in the context of preferences. In formal argumentation, preferences are often accounted for by removing [21,33] or even adding [21] particular attacks from and to the argumentation framework. In logic programming, preferences are often accounted for by selecting among the various logic programming models those that are most in line with the preferences (see [22] for an overview). Hence, whereas preferences in formal argumentation are accounted for *before* computing semantics, possibly changing the argumentation framework itself, in logic programming the preferences are accounted for *during* seman-

[13] Given a program P, the WFS+ semantics provides a unique model which complements the well-founded model WFS(P) of P with atoms that are classically entailed by P. Dix [23] writes that $P \models a$ iff a is true in every 2-valued model of P. Here, models are expressed simply as sets of true atoms. The WFS+ model of a logic program P is then obtained as the least fix-point of the sequence $M_{k+1} = SEM(P \cup M_k)$ of subsets of HB_P, where $SEM(P) = \text{WFS}(P) \cup \{a \in HB_P \mid P \models a\}$ and $M_0 = \emptyset$.

[14] For short, let us make $\Pi_{AF_3} = \Pi_3^0$. Starting from $M_0 = \emptyset$, we obtain $M_1 = SEM(\Pi_3^0 \cup M_0) = SEM(\Pi_3^0) = \text{WFS}(\Pi_3^0) \cup \{a \in HB_{\Pi_3^0} \mid \Pi_3^0 \models a\}$. Given that $\text{WFS}(\Pi_3^0) = \emptyset$, this means $M_1 = \emptyset \cup \{def(A)\} = \{def(A)\}$. Next, we must compute $M_2 = SEM(\Pi_3^0 \cup M_1) = SEM(\Pi_3^0 \cup \{def(A)\})$. For short, let us make $\Pi_3^0 \cup \{def(A)\} = \Pi_3^1$. We obtain $M_2 = SEM(\Pi_3^0 \cup M_1) = SEM(\Pi_3^1) = \text{WFS}(\Pi_3^1) \cup \{a \in HB_{\Pi_3^1} \mid \Pi_3^1 \models a\}$. Now we have $\text{WFS}(\Pi_3^1) = \{def(A)\}$, so $M_2 = \{def(A)\} \cup \{def(A)\} = \{def(A)\} = M_1$. This means $\text{WFS+}(\Pi_{AF_3}) = \{def(A)\}$.

tics computation. To identify whether there are conditions under which the first approach corresponds to the second approach is a promising topic for further research.

So far, the existing approaches were compared almost exclusively within their native systems. One exception would be Wakaki's approach to preference-based argumentation [45], which adapts the approach of Sakama and Inoue's Prioritized Logic Programming (PLP) [40] to formal argumentation. Wakaki [45] proposes preference-based argumentation semantics and shows some correspondences to preferential semantics for logic programs hold, and we believe several more correspondences are also likely to hold. This idea is supported by recent findings of Silva et. al. [41], who complemented the set of semantics of Amgoud and Vesic [3] for preferential AFs to show they preserve the same hierarchy as their counterpart semantics in Dung's AFs. It turns out that the logic programming semantics we studied (together with [16,47]) hold the same hierarchy as their counterpart AF semantics too.

Therefore, a particularly promising path of research enabled by our work is the potential adaptation of approaches using static preferences over literals from logic programming (such as [40], but see also [22]) to develop new approaches for preferences in abstract argumentation. This much is enabled by Definition 24 (taken from [16,47]), which ensures an AF can be written as a normal logic program that fully preserves its admissibility-based semantics, at least up from complete semantics. In short, the semantics of any approaches to logic programs with preferences over literals can be adapted for AFs using the mapping in Definition 24.

References

[1] J. Alcântara, S. Sá and J. Acosta-Guadarrama, On the equivalence between abstract dialectical frameworks and logic programs, *Theory and Practice of Logic Programming* **19**(5–6) (2019), 941–956. doi:10.1017/S1471068419000280.

[2] J.J. Alferes, P.M. Dung and L. Pereira, Scenario semantics of extended logic programs, in: *Proc. 2nd International Workshop on Logic Programming and Non-monotonic Reasoning*, A. Nerode and L. Pereira, eds, MIT Press, 1993, pp. 334–348.

[3] L. Amgoud and S. Vesic, A new approach for preference-based argumentation frameworks, *Annals of Mathematics and Artificial Intelligence* **63**(2) (2011), 149–183. doi:10.1007/s10472-011-9271-9.

[4] P. Baroni, M.W.A. Caminada and M. Giacomin, An introduction to argumentation semantics, *Knowledge Engineering Review* **26**(4) (2011), 365–410. doi:10.1017/S0269888911000166.

[5] P. Baroni and M. Giacomin, On principle-based evaluation of extension-based argumentation semantics, *Artificial Intelligence* **171**(10–15) (2007), 675–700. doi:10.1016/j.artint.2007.04.004.

[6] R. Baumann, On the nature of argumentation semantics: Existence and uniqueness, expressibility, and replaceability, in: *Handbook of Formal Argumentation*, Vol. 1, College Publications, 2018.

[7] R. Baumann and C. Spanring, Infinite argumentation frameworks – on the existence and uniqueness of extensions, in: *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation – Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, T. Eiter, H. Strass, M. Truszczyński and S. Woltran, eds, Springer, 2015, pp. 281–295. doi:10.1007/978-3-319-14726-0_19.

[8] P. Besnard and A. Hunter, Constructing argument graphs with deductive arguments: A tutorial, *Argument & Computation* **5** (2014), 5–30, Special Issue: Tutorials on Structured Argumentation. doi:10.1080/19462166.2013.869765.

[9] A. Bondarenko, P.M. Dung, R.A. Kowalski and F. Toni, An abstract, argumentation-theoretic approach to default reasoning, *Artificial Intelligence* **93** (1997), 63–101. doi:10.1016/S0004-3702(97)00015-5.

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

[11] M.W.A. Caminada, Comparing two unique extension semantics for formal argumentation: Ideal and eager, in: *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2007)*, M.M. Dastani and E. de Jong, eds, 2007, pp. 81–87.

[12] M.W.A. Caminada, Argumentation semantics as formal discussion, in: *Handbook of Formal Argumentation*, Vol. 1, College Publications, 2018, pp. 487–518.

[13] M.W.A. Caminada, P. Baroni and M. Giacomin, Abstract argumentation frameworks and their semantics, in: *Handbook of Formal Argumentation*, Vol. 1, College Publications, 2018.
[14] M.W.A. Caminada and D.M. Gabbay, A logical account of formal argumentation, *Studia Logica* **93**(2–3) (2009), 109–145, Special issue: new ideas in argumentation theory. doi:10.1007/s11225-009-9218-x.
[15] M.W.A. Caminada and G. Pigozzi, On judgment aggregation in abstract argumentation, *Autonomous Agents and Multi-Agent Systems* **22**(1) (2011), 64–102. doi:10.1007/s10458-009-9116-7.
[16] M.W.A. Caminada, S. Sá, J. Alcântara and W. Dvořák, On the equivalence between logic programming semantics and argumentation semantics, *International Journal of Approximate Reasoning* **58** (2015), 87–111. doi:10.1016/j.ijar.2014.12.004.
[17] M.W.A. Caminada and C. Schulz, On the equivalence between assumption-based argumentation and logic programming, *Journal of Artificial Intelligence Research* **60** (2017), 779–825. doi:10.1613/jair.5581.
[18] M.W.A. Caminada and S. Uebis, An implementation of argument-based discussion using aspic, in: *Computational Models of Argument. Proceedings of COMMA 2020*, H. Prakken, S. Bistarelli, F. Santini and C. Taticchi, eds, IOS Press, 2020, pp. 455–456.
[19] M.W.A. Caminada and B. Verheij, On the existence of semi-stable extensions, in: *Proceedings of the 22nd Benelux Conference on Artificial Intelligence*, G. Danoy, M. Seredynski, R. Booth, B. Gateau, I. Jars and D. Khadraoui, eds, 2010.
[20] K. Čyras, X. Fan, C. Schulz and F. Toni, Assumption-based argumentation: Disputes, explanations, preferences, in: *Handbook of Formal Argumentation*, Vol. 1, College Publications, 2018.
[21] K. Čyras and F. Toni, Aba+: Assumption-based argumentation with preferences, in: *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2016)*, 2016, pp. 553–556.
[22] J. Delgrande, T. Schaub, H. Tompits and K. Wang, A classification and survey of preference handling approaches in nonmonotonic reasoning, *Computational Intelligence* **20**(2) (2004), 308–334. doi:10.1111/j.0824-7935.2004.00240.x.
[23] J. Dix, A classification theory of semantics of normal logic programs: I. strong properties, *Fundamenta Informaticae* **22**(3) (1995), 227–255. doi:10.3233/FI-1995-2233.
[24] P.M. 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. doi:10.1016/0004-3702(94)00041-X.
[25] P.M. Dung, P. Mancarella and F. Toni, Computing ideal sceptical argumentation, *Artificial Intelligence* **171**(10–15) (2007), 642–674. doi:10.1016/j.artint.2007.05.003.
[26] N. Gorogiannis and A. Hunter, Instantiating abstract argumentation with classical logic arguments: Postulates and properties, *Artificial Intelligence* **175**(9–10) (2011), 1479–1497. doi:10.1016/j.artint.2010.12.003.
[27] G. Governatori, M.J. Maher, G. Antoniou and D. Billington, Argumentation semantics for defeasible logic, *Journal of Logic and Computation* **14**(5) (2004), 675–702. doi:10.1093/logcom/14.5.675.
[28] H. Jakobovits and D. Vermeir, Robust semantics for argumentation frameworks, *Journal of logic and computation* **9**(2) (1999), 215–261. doi:10.1093/logcom/9.2.215.
[29] A.C. Kakas and P. Mancarella, Short note preferred extensions are partial stable models, *The Journal of Logic Programming* **14**(3–4) (1992), 341–348. doi:10.1016/0743-1066(92)90015-U.
[30] B. Marshall, R. Booth and M.W.A. Caminada, Disco: A web-based implementation of discussion games for grounded and preferred semantics, in: *Computational Models of Argument. Proceedings of COMMA 2018*, J. Lawrence, S. Modgil and K. Budzynska, eds, 2018, pp. 453–454.
[31] S. Modgil and H. Prakken, A general account of argumentation with preferences, *Artificial Intellligence* **195** (2013), 361–397. doi:10.1016/j.artint.2012.10.008.
[32] S. Modgil and H. Prakken, The ASPIC+ framework for structured argumentation: A tutorial, *Argument & Computation* **5** (2014), 31–62, Special Issue: Tutorials on Structured Argumentation. doi:10.1080/19462166.2013.869766.
[33] S. Modgil and H. Prakken, Abstract rule-based argumentation, in: *Handbook of Formal Argumentation*, Vol. 1, College Publications, 2018. doi:10.1007/978-3-319-75553-3.
[34] J.C. Nieves and M. Osorio, Ideal extensions as logical programming models, *Journal of Logic and Computation* **26**(5) (2016), 1361–1393. doi:10.1093/logcom/exu014.
[35] T.C. Przymusinski, The well-founded semantics coincides with the three-valued stable semantics, *Fundamenta Informaticae* **13**(4) (1990), 445–463.
[36] S. Sá, On the expressive power of argumentation formalisms, in: *Online Handbook of Argumentation for AI*, 2020, p. 33.
[37] S. Sá and J. Alcântara, Interpretations and models for assumption-based argumentation, in: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC '19*, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1139–1146.
[38] D. Saccà and C. Zaniolo, Stable models and non-determinism in logic programs with negation, in: *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Nashville, Tennessee, USA, April 2–4, 1990, D.J. Rosenkrantz and Y. Sagiv, eds, ACM Press, 1990, pp. 205–217.
[39] D. Saccà and C. Zaniolo, Deterministic and non-deterministic stable models, *Journal of Logic and Computation* **7**(5) (1997), 555–579. doi:10.1093/logcom/7.5.555.

[40] C. Sakama and K. Inoue, Prioritized logic programming and its application to commonsense reasoning, *Artificial Intelligence* **123**(1–2) (2000), 185–222. doi:10.1016/S0004-3702(00)00054-0.
[41] R. Silva, S. Sá and J. Alcântara, Semantics hierarchy in preference-based argumentation frameworks, in: *Computational Models of Argument – Proceedings of COMMA 2020*, Perugia, Italy, September 4–11, 2020, H. Prakken, S. Bistarelli, F. Santini and C. Taticchi, eds, Frontiers in Artificial Intelligence and Applications, Vol. 326, IOS Press, 2020, pp. 339–346.
[42] H. Strass, Approximating operators and semantics for abstract dialectical frameworks, *Artificial Intelligence* **205** (2013), 39–70. doi:10.1016/j.artint.2013.09.004.
[43] P.M. Thang, P.M. Dung and N.D. Hung, Towards a common framework for dialectical proof procedures in abstract argumentation, *Journal of Logic and Computation* **19**(6) (2009), 1071–1109. doi:10.1093/logcom/exp032.
[44] F. Toni, A tutorial on assumption-based argumentation, *Argument & Computation* **5** (2014), 89–117, Special Issue: Tutorials on Structured Argumentation. doi:10.1080/19462166.2013.869878.
[45] T. Wakaki, Preference-based argumentation built from prioritized logic programming, *Journal of Logic and Computation* **25**(2) (2015), 251–301. doi:10.1093/logcom/exs066.
[46] E. Weydert, Semi-stable extensions for infinite frameworks, in: *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)*, P. de Causmaecker, J. Maervoet, T. Messelis, K. Verbeeck and T. Vermeulen, eds, 2011, pp. 336–343.
[47] Y. Wu, M.W.A. Caminada and D.M. 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. doi:10.1007/s11225-009-9210-5.