On the Equivalence between Logic Programming Semantics and Argumentation Semantics

Martin Caminada[1,2⋆], Samy Sá[3⋆⋆], and João Alcântara[3⋆⋆]

[1] Université du Luxembourg
[2] University of Aberdeen
[3] Universidade Federal do Ceará

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

1 Introduction

The link between logic programming and formal argumentation theory goes back to the seminal work of [1] in which various connections were pointed out. To some extent, the approach of abstract argumentation is a way of providing an abstraction of some aspects of logic programming. This connection is especially clear when comparing the different semantics for logic programming with the different semantics for formal argumentation. In this paper, we continue such a line of research by pointing out that the translation of [2] from logic programming to formal argumentation can account for a whole range of equivalences between logic programming semantics and formal argumentation semantics. This includes both existing results like the equivalence between stable model semantics (LP) and stable semantics (argumentation) [1], well-founded semantics (LP) and grounded semantics (argumentation) [1], and partial stable model semantics (LP) and complete semantics [2], as well as a newly proved equivalence between regular model semantics (LP) and preferred semantics (argumentation).

In this paper, besides exploiting the connection between logic programming and formal argumentation, our results shed light on some aspects of instantiated

⋆ Supported by the National Research Fund, Luxembourg (LAAMI project) and by the Engineering and Physical Sciences Research Council (EPSRC, UK), grant ref. EP/J012084/1 (SAsSy project).

⋆⋆ Supported by CNPq (Universal 2012 - Proc. n 473110/2012-1), CAPES (PROCAD 2009), CNPq/CAPES (Casadinho/PROCAD 2011).

argumentation theory (e.g. [3–6]). In particular we show the connection between argument-labellings at the abstract level and conclusion-labellings at the instantiated level. With one notable exception, we are able to show that maximizing (or minimizing) a particular label (`in`, `out` or `undec`) at the argument level coincides with maximizing (or minimizing) the same label at the conclusion level. These results are relevant as they indicate the possibilities (and limitations) of applying argument-based abstractions to formalisms for non-monotonic reasoning.

2 Preliminaries

In the current paper, we follow the approach of Dung [1]. We will restrict ourselves to finite argumentation frameworks.

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

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

Definition 2 ([1]). *(defense/conflict-free). Let* (Ar, Att) *be an argumentation framework,* $A \in Ar$ *and* $\mathcal{A}rgs \subseteq Ar$. *We say* $Args$ *is* conflict-free *iff there exists no arguments* $A, B \in \mathcal{A}rgs$ *such that* $(A, B) \in Att$. *We say* $Args$ defends A *iff every argument attacking* A *is attacked by some argument in* $Args$. *We define a function* $F : 2^{Ar} \rightarrow 2^{Ar}$, *such that* $F(Args) = \{A | A\ is\ defended\ by\ Args\}$, *to determine the set of all arguments defended by* $Args$. *We define* $Args^+ = \{A | A\ is\ attacked\ by\ Args\}$ *to refer to the set of arguments attacked by* $Args$.

Traditional approaches to argumentation semantics are based on extensions of arguments. Some of the mainstream approaches are summarized below:[4]

Definition 3. *(extension-based argumentation semantics). Given an argumentation framework* $AF = (Ar, Att)$, *and a conflict-free set of arguments* S:

- S *is a complete extension of* AF *iff* $S = F(S)$.
- S *is a grounded extension of* AF *iff* S *is a minimal*[5] *complete extension of* AF.
- S *is a preferred extension of* AF *iff* S *is a maximal complete extension of* AF.
- S *is a stable extension of* AF *iff* S *is a complete ext. of* AF *with* $S^+ = Ar \backslash S$.
- S *is a semi-stable extension of* AF *iff* S *is a complete ext. of* AF *with maximal* $S \cup S^+$.

As for logic programming, we will focus on propositional normal logic programs, which we will call logic programs or simply programs from now on.

[4] The characterization of the extension-based semantics in Definition 3 differs slightly from that in their original version (see [1]), but equivalence is proved in [7].

[5] When referring to minimal/maximal, we assume the underlying order is set inclusion.

Definition 4. *A* logic program P *is a set of rules of the form* $c \leftarrow a_1, \ldots, a_m$, $\texttt{not}\ b_1, \ldots, \texttt{not}\ b_n$ *($m, n \in \mathbb{N}$), where c, a_i ($1 \leq i \leq m$) and b_j ($1 \leq j \leq n$) are atoms and* not *represents negation as failure. We say c is the* head *of the rule, and $a_1, \ldots, a_m, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n$ is its* body. *The* Herbrand Base *of P is the set HB_P of all atoms occurring in P.*

A wide range of logic programming semantics can be defined based on the 3-valued interpretations (for short, interpretation) of programs [8]:

Definition 5. *A* 3-valued interpretation I *of a program P is a pair $< T; F >$, where $T \cup F \subseteq HB_P$ and $T \cap F = \emptyset$. Atoms in T (resp. F) are intended to be true (resp. false) in I. Atoms in $U = HB_P \backslash (T \cup F)$ are considered as undefined in I.*

Let $I =< T; F >$ be a 3-valued interpretation of the program P, take P/I to be the program built by the execution of the following steps:

1. Remove any $c \leftarrow a_1, \ldots, a_m, \texttt{not}\, b_1, \ldots, \texttt{not}\, b_n \in P$ with $\{b_1, \ldots, b_n\} \cap T \neq \emptyset$;
2. Afterwards, remove any occurrence of $\texttt{not}\ b_i$ from P such that $b_i \in F$.
3. Then, replace any occurrence of $\texttt{not}\ b_i$ left by a special atom $\mathbf{u}$ ($\mathbf{u} \notin HB_P$).

We note $\mathbf{u}$ was tailored to be undefined in every interpretation of P. As shown in [8], P/I has a unique least 3-valued model: $\Psi(I) =< T_\Psi; F_\Psi >$ with minimal T_Ψ and maximal F_Ψ such that, for every $c \in HB_P$:

- $c \in T_\Psi$ if $c \leftarrow a_1, \ldots, a_m \in P/I$ and $\{a_1, \ldots, a_m\} \subseteq T_\Psi$;
- $c \in F_\Psi$ if for every $c \leftarrow a_1, \ldots, a_m \in P/I$, $\{a_1, \ldots, a_m\} \cap F_\Psi \neq \emptyset$;
- $c \in U_\Psi$ otherwise.

We now specify the logic programming semantics to be examined in this paper.

Definition 6. *Let P be a program and $I =< T, F >$ be an interpretation:*

- *I is a partial stable model (p.s.m.) of P iff $I = \Psi(I)$ [8].*
- *I is a well-founded model of P iff I is a p.s.m. of P with minimal T [8].*
- *I is a regular model of P iff I is a p.s.m. of P with maximal T [9].*
- *I is a stable model of P iff I is a p.s.m. of P where $F = HB_P \backslash T$, i.e., $U = \emptyset$ [8].*
- *I is an L-stable model of P iff I is a p.s.m. of P with maximal $T \cup F$ [9].*

3 Logic Programming as Argumentation; a 3-step process

The next thing to examine is how argumentation theory can be applied in the context of logic programming. Our treatment is based on [2][6]. The idea is to apply (as in [3–6]) the standard three-step process of instantiated argumentation. One starts with a knowledge base and builds the associated argumentation framework (step 1), then applies abstract argumentation semantics (step 2) and then looks at what the results of the argumentation semantics imply at the level of conclusions (step 3).

[6] One difference is that in our approach, arguments are recursive, whereas in [2], they are trees of rules. However, if one identifies the nodes of a tree with rules, one cannot apply the same rule at different positions in the argument. Our approach, which is based on [3, 4], avoids this problem.

3.1 Step 1: Argumentation Framework Construction

The approach of instantiated argumentation starts with a particular knowledge base; in our case, it will be a normal logic program. From this program, one can start to construct *arguments* recursively as follows:

Definition 7. *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 $\texttt{Conc}(A) = c$, $\texttt{Rules}(A) = \{c \leftarrow \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m\}$, and $\texttt{Vul}(A) = \{b_1, \ldots, b_m\}$.*
- *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 ($1 \leq i \leq n$) there exists an argument A_i with $\texttt{Conc}(A_i) = a_i$ and $c \leftarrow a_1, \ldots, a_n, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m \notin \texttt{Rules}(A_i)$ then $c \leftarrow (A_1), \ldots, (A_n), \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ is an argument (say A) with $\texttt{Conc}(A) = c$, $\texttt{Rules}(A) = \texttt{Rules}(A_1) \cup \ldots \cup \texttt{Rules}(A_n) \cup \{c \leftarrow a_1, \ldots, a_n, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m\}$, and $\texttt{Vul}(A) = \texttt{Vul}(A_1) \cup \ldots \cup \texttt{Vul}(A_n) \cup \{b_1, \ldots, b_m\}$.*

An argument A 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 A). We refer to $\texttt{Conc}(A)$ as the *conclusion* of A and $\texttt{Vul}(A)$ as the *vulnerabilities* of A.

The next step is to determine the attack relation: an argument attacks another iff its conclusion is one of the vulnerabilities of the attacked argument.

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

The notion of attack has a clear meaning: if $b \in \texttt{Vul}(A)$, then A is built using at least one rule with $\texttt{not}\ b$ in its body. Hence, A is a defeasible derivation that depends on b not being derivable. An argument B providing a (possibly defeasible) derivation of b (i.e., $\texttt{Conc}(B) = b$) can thus be seen as *attacking* A.

Now one can define the argumentation framework associated to a program:

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

3.2 Step 2: Applying Argumentation Semantics

Once the argumentation framework has been built, the next question is which arguments should be accepted and which should be rejected. As shown in Section 2, several approaches have been stated for determining this. Here we will focus on complete semantics [1], which can be defined via complete labellings [10, 7].

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

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

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

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

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

3.3 Step 3: converting argument labellings to conclusion labellings

For many practical purposes, what matters are not so much the arguments themselves, but the conclusions they support. Hence, for each position on which *arguments* to accept, reject or abstain we need to specify the associated position on which *conclusions* to accept, reject or abstain. For current purposes, we follow the approach described in [11]. Here, the idea is for each conclusion to identify the "best" argument that yields it. We assume a strict total order between different individual labels such that $\texttt{in} > \texttt{undec} > \texttt{out}$. The best argument for a conclusion is the one with the highest label. If there is no argument at all for a particular conclusion, it will be labelled `out`.

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

When *ConcLab* is a conclusion labelling, we write $\texttt{in}(ConcLab)$ to denote the set of $\{c \mid ConcLab(c) = \texttt{in}\}$, $\texttt{out}(ConcLab)$ for $\{c \mid ConcLab(c) = \texttt{out}\}$ and $\texttt{undec}(ConcLab)$ for $\{c \mid ConcLab(c) = \texttt{undec}\}$. Sometimes we will write a conclusion labelling as $(\mathcal{C}oncs_1, \mathcal{C}oncs_2, \mathcal{C}oncs_3)$ where $\mathcal{C}oncs_1 = \texttt{in}(ConcLab)$, $\mathcal{C}oncs_2 = \texttt{out}(ConcLab)$ and $\mathcal{C}oncs_3 = \texttt{undec}(ConcLab)$.

4 Minimization/Maximization of Argument Labellings

In [10, 7] it was observed that for each complete argument labelling *ArgLab* of a particular argumentation framework AF, it holds that:

- `in`(*ArgLab*) is maximal among all complete argument labellings of AF iff `out`(*ArgLab*) is maximal among all complete argument labellings of AF
- `in`(*ArgLab*) is minimal among all complete argument labellings of AF iff `out`(*ArgLab*) is minimal among all complete argument labellings of AF iff `undec`(*ArgLab*) is maximal among all complete argument labellings of AF

If a complete argument labelling has maximal `in` (or equivalently, maximal `out`) we call it a *preferred argument labelling*. If it has minimal `in` (or equivalently, minimal `out` or maximal `undec`), we call it a *grounded argument labelling*. Otherwise, if it has minimal `undec`, we call it a *semi-stable argument labelling*. Lastly, if it has no argument at all labelled `undec`, we call it an *argstable argument labelling*.

Argument labellings and argument extensions are one-to-one related. In fact, an extension is the `in`-labelled part of the associated labelling: if *ArgLab* is a complete (resp. preferred, grounded, semi-stable or argstable) argument labelling of argumentation framework $AF = (Ar, att)$, then `in`(*ArgLab*) is a complete (resp. preferred, grounded, semi-stable or stable) extension of AF. Furthermore, if E is a complete (resp. preferred, grounded, semi-stable or stable) extension of AF then $(E, E^+, Ar \setminus (E \cup E^+))$ is a complete (resp. preferred, grounded, semi-stable or argstable) labelling of AF (see [10, 7] for details).

Note that if *ArgLab* is a complete (or respectively, preferred, grounded, semi-stable or argstable) argument labelling, then the associated conclusion labelling (Definition 11) will be called a complete (or respectively, preferred, grounded, semi-stable or argstable) conclusion labelling.

5 Minimization/Maximization of Conclusion Labellings

Preferred, grounded, semi-stable, and argstable conclusion labellings, as defined in the previous section, are based on the common idea of performing the maximization/minimization at the level of argument labellings and then identifying the associated conclusion labellings. An alternative procedure would be simply to identify *all* complete conclusion labellings and then to perform the maximization/minimization right at the level of the conclusion labellings.

It turns out that (as for argument labellings) some of the maximizations and minimisations of the conclusion labellings are equivalent to others. In [12], it is proved that for each complete conclusion labelling *ConcLab* of structured argumentation framework AF, it holds that:

- `in`(*ConcLab*) is maximal among all complete conclusion labellings of AF iff `out`(*ConcLab*) is maximal among all complete conclusion labellings of AF
- `in`(*ConcLab*) is minimal among all complete conclusion labellings of AF iff `out`(*ConcLab*) is minimal among all complete conclusion labellings of AF iff `undec`(*ConcLab*) is maximal among all complete argument labellings of AF

If a complete conclusion labelling has maximal `in` (or equivalently, maximal `out`) we call it a *regular conclusion labelling*. If it has minimal `in` (or equivalently, minimal `out` or maximal `undec`), we call it a *well-founded conclusion*

labelling. Otherwise, if it has minimal `undec`, we call it an *L-stable conclusion labelling*. Lastly, if it has no argument at all labelled `undec`, we call it a *concstable conclusion labelling*.

Conclusion labellings and logic programming models turn out to be one-to-one related. The basis of this result is [2], where the equivalence between complete conclusion labellings and partial stable models was identified. More specifically:

- if *ConcLab* is a complete conclusion labelling of structured argumentation framework AF_P (generated by a logic program P) then $< \texttt{in}(ConcLab); \texttt{out}(ConcLab) >$ is a partial stable model of P
- if $< T; F >$ is a partial stable model of P then $(T, F, HB_P \setminus (T \cup F))$ is a complete conclusion labelling of the argumentation framework AF_P

From this result, other correspondences between conclusion labellings and logic programming models follow. As a regular model is a partial stable model with maximal T, and a regular conclusion labelling is a complete conclusion labelling with maximal `in`, it follows that they correspond to each other. Similar correspondences hold between the well-founded model and the well-founded conclusion labelling, between L-stable models and L-stable conclusion labellings, and between stable models and concstable conclusion labellings. To sum up, the various types of logic programming models are actually different forms of conclusion labellings.

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

So far, we have selected subsets of the complete conclusion labellings as follows:

1. Perform minimization (resp. maximization) of a label at the level of complete argument labellings, then obtain the associated conclusion labellings. This procedure was described in Section 4, and is in fact similar to what is done in instantiated argumentation in general [3–6].
2. Take all complete conclusion labellings (these are the associated labellings of *all* complete argument labellings) and then perform the minimization (resp. maximization) of a particular label at the level of complete conclusion labellings. This procedure was described in Section 5 and is in fact similar to what is being done by various logic programming semantics.

An interesting question is whether the outcome of the two procedures is the same. That is, does minimizing/maximizing a label at the level of argument labellings equal to minimizing/maximizing the label at the level of conclusion labellings? We will see that the answer is "yes", with one notable exception.[7]

Theorem 1. *Let ConcLab be a conclusion labelling of logic program P and associated argumentation framework $AF_P = (Ar, att)$. It holds that ConcLab is a preferred conclusion labelling iff it is a regular conclusion labelling.*

[7] Proofs that have been omitted due to space restrictions can be found in [12].

Theorem 2. *Let ConcLab be a conclusion labelling of logic program P and associated argumentation framework $AF_P = (Ar, att)$. It holds that ConcLab is the grounded conclusion labelling iff it is the well-founded conclusion labelling.*

Theorem 3. *Let ConcLab be a conclusion labelling of logic program P and associated argumentation framework $AF_P = (Ar, att)$. It holds that ConcLab is an argstable conclusion labelling iff it is a concstable conclusion labelling.*

One can also ask whether semi-stable conclusion labellings are the same as L-stable conclusion labellings. Here, however, the answer is negative:

Example 1. Let P be the program below, whose associated argumentation framework AF_P is in Fig. 1, and let $\{A_1, A_2, A_3, A_4, A_5\}$ be arguments built from P.[8]

$$r_1 : c \leftarrow \texttt{not}\ c \qquad r_2 : a \leftarrow \texttt{not}\ b$$
$$r_3 : b \leftarrow \texttt{not}\ a \qquad r_4 : c \leftarrow \texttt{not}\ c, \texttt{not}\ a$$
$$r_5 : g \leftarrow \texttt{not}\ g, \texttt{not}\ b$$

- $A_1 = r_1$, with $\texttt{Conc}(A_1) = c$ and $\texttt{Vul}(A_1) = \{c\}$
- $A_2 = r_2$, with $\texttt{Conc}(A_2) = a$ and $\texttt{Vul}(A_2) = \{b\}$
- $A_3 = r_3$, with $\texttt{Conc}(A_3) = b$ and $\texttt{Vul}(A_3) = \{a\}$
- $A_4 = r_4$, with $\texttt{Conc}(A_4) = c$ and $\texttt{Vul}(A_4) = \{c, a\}$
- $A_5 = r_5$, with $\texttt{Conc}(A_5) = g$ and $\texttt{Vul}(A_5) = \{g, b\}$

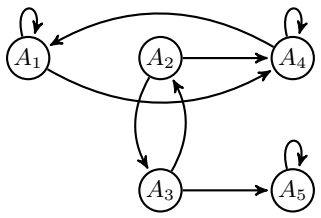

Fig. 1. The argumentation framework AF_P associated with P.

The complete argument labellings of AF_P are $ArgLab_1 = (\emptyset, \emptyset, \{A_1, A_2, A_3, A_4, A_5\})$, $ArgLab_2 = (\{A_2\}, \{A_3, A_4\}, \{A_1, A_5\})$, and $ArgLab_3 = (\{A_3\}, \{A_2, A_5\}, \{A_1, A_4\})$. The associated complete conclusion labellings are $ConcLab_1 = (\emptyset, \emptyset, \{a, b, c, g\})$, $ConcLab_2 = (\{a\}, \{b\}, \{c, g\})$, and $ConcLab_3 = (\{b\}, \{a, g\}, \{c\})$.

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

[8] We thank Wolfgang Dvořák for this example.

7 On the Connection between Argumentation Semantics and Logic Programming Semantics

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

- maximizing `in` (or, equivalently, maximizing `out`) at the argument level yields the same result as maximizing `in` (or, equivalently, maximizing `out`) at the conclusion level. Hence, preferred conclusion labellings and regular conclusion labellings coincide.
- minimizing `in` (or, equivalently, minimizing `out` or maximizing `undec`) at the argument level yields the same result as minimizing `in` (or, equivalently, minimizing `out` or maximizing `undec`) at the conclusion level. Hence, the grounded conclusion labelling and the well-founded conclusion labelling coincide.
- minimizing `undec` at the argument level does *not* yield the same result as minimizing `undec` at the conclusion level. Hence, semi-stable conclusion labellings and L-stable conclusion labellings do *not* coincide.
- ruling out `undec` at the argument level yields the same result as ruling out `undec` at the conclusion level. Hence, argstable conclusion labellings and concstable conclusion labellings coincide.

We have now arrived at the main point of this paper: the connection between (traditional) approaches to argumentation semantics and (traditional) approaches to logic programming semantics. Let us again look at the 3-step process of Section 3. Assume that steps 1 and 3 are fixed. At step 2, it follows that

- if one applies complete semantics at step 2, the overall outcome is equivalent to calculating the partial stable models of the original logic program [2]
- if one applies preferred semantics at step 2, the overall outcome is equivalent to applying regular semantics to the original logic program
- if one applies grounded semantics at step 2, the overall outcome is equivalent to applying well-founded semantics to the original logic program
- if one applies stable semantics at step 2, the overall outcome is equivalent to applying stable model semantics to the original logic program

Thus, differences in logic programming semantics can be reduced to differences in abstract argumentation semantics (see Table 1). We are also able to explain *why* these semantics coincide, as what happens at the argument level tends to affect the conclusion level. For instance, preferred semantics coincides with regular semantics *because* maximizing `in` at either argument or conclusion level yields the same results; grounded semantics coincides with well-founded semantics *because* minimizing `in` at either argument or conclusion level yields the same results; stable semantics coincides with stable model semantics *because* ruling out `undec` at either argument or conclusion level yields the same results. Finally, semi-stable semantics does *not* coincide with L-stable model *because* minimizing `undec` at the argument level does not yield the same result as doing so at the conclusion level.

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

Table 1. Connections between argumentation semantics and LP semantics

8 Semi-Stable and L-Stable Semantics Revisited

We will now focus on the previously observed discrepancy between semi-stable semantics and L-stable semantics. If semi-stable semantics is not able to generate L-stable conclusion labellings, then is there perhaps any other abstract argumentation semantics that can generate these? More precisely, we are interested in an abstract argumentation semantics to be applied at step 2 of the argumentation process, whose associated conclusion labellings (step 3) are precisely the L-stable labellings. Furthermore, this semantics should purely be defined on the structure of the graph (argumentation framework) and not rely on the actual contents of the arguments. That is, the semantics should satisfy the *language independence principle* [13].

Definition 12. *We say that an abstract argumentation semantics X is L-stable generating iff it is a function such that*

1. *For any logic program P, X takes as input AF_P and yields as output a set of argument labellings ArgLabs*
2. *X satisfies language independence [13, Definition 37], meaning that for any pair of argumentation frameworks AF_1, AF_2 that are isomorphic[9] by a mapping M of their arguments (the nodes in the graphs), each labelling of AF_1 can be mapped to a different labelling of AF_2 by the same mapping M.*
3. *It holds that $\{ConcLab \mid ConcLab$ is the associated conclusion labelling of some $ArgLab \in ArgLabs\}$ is precisely the set of all L-stable conclusion labellings of AF_P.*

Theorem 4. *No abstract argumentation semantics is L-stable generating.*

Proof. Consider the programs P with rules $r_1, ..., r_4$ and P' with rules $r'_1, ..., r'_4$:

$r_1 : c \leftarrow \texttt{not}\ c$	$r'_1 : d \leftarrow \texttt{not}\ c, \texttt{not}\ d$
$r_2 : a \leftarrow \texttt{not}\ b$	$r'_2 : a \leftarrow \texttt{not}\ b$
$r_3 : b \leftarrow \texttt{not}\ a$	$r'_3 : b \leftarrow \texttt{not}\ a$
$r_4 : c \leftarrow \texttt{not}\ c, \texttt{not}\ a$	$r'_4 : c \leftarrow \texttt{not}\ c, \texttt{not}\ a, \texttt{not}\ d$

The argumentation frameworks of P and P' are depicted in Fig. 2. Note that:

[9] Two argumentation frameworks AF_1, AF_2 are isomorphic (as in graph isomorphism) if there is an edge-preserving bijection from the arguments (the nodes) of AF_1 to those of AF_2, when these argumentation frameworks are perceived as graphs.

– P has three partial stable models: $S_1 =< \emptyset; \emptyset >$, $S_2 =< \{a\}; \{b\} >$ and $S_3 =< \{b\}; \{a\} >$, where S_2 and S_3 are L-stable models.
– P' has three partial stable models: $S_1 =< \emptyset; \emptyset >$, $S_2 =< \{a\}; \{b, c\} >$ and $S_3 =< \{b\}; \{a\} >$, where S_2 is the single L-stable model.

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

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

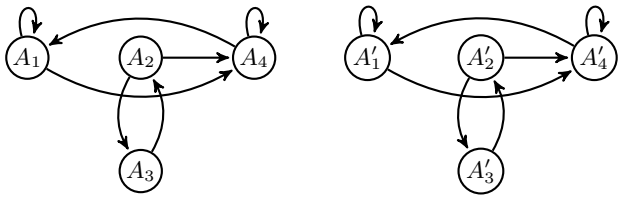

Fig. 2. The argumentation frameworks associated with P and P'.

Though P has two L-stable models and P' has only one, they are indiscernible in abstract argumentation semantics. Thus, no semantics of abstract argumentation can coincide with the L-stable semantics for each and every program.

9 Discussion

In this paper, we have studied several connections between abstract argumentation semantics and logic programming semantics. We observed that various argumentation semantics are based on maximizations and minimizations (of a particular label) at the *argument level* whereas various logic programming semantics are based on maximizations and minimizations (of a particular label) at the *conclusion level*. Where performing the maximizations/minimizations at the argument level yields the same results as performing the maximizations/minimizations at the conclusion level, the associated argumentation semantics and logic programming semantics coincide. Where performing the maximizations/minimizations at the argument level does *not* yield the same results as performing the maximizations/minimizations at the conclusion level, the corresponding argumentation semantics and logic programming semantics (semistable / L-stable) do not coincide.

Although the current paper focuses mainly on instantiated argumentation based on logic programming, its main findings are in fact relevant for instantiated argumentation in general (like [3, 4, 6, 5]) as it specifies the possibilities

and impossibilities of using the argumentation approach to specify nonmonotonic entailment, or to model existing nonmonotonic formalisms. If the aim is, for instance, to model a formalism that maximizes `in` or `out` at the conclusion level (like [14]) the argumentation approach will do fine (as evidenced by [15]). However, if the aim is to model a formalism that minimizes `undec` at the conclusion level, the argumentation approach will not be able to provide any help (Theorem 4). Hence, the current paper has shed some light on the strengths and limitations of using the argumentation approach for specifying nonmonotonic entailment.

References

1. Dung, P.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence **77** (1995) 321–357
2. Wu, Y., Caminada, M., Gabbay, D.: 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.
3. Caminada, M., Amgoud, L.: On the evaluation of argumentation formalisms. Artificial Intelligence **171**(5-6) (2007) 286–310
4. Prakken, H.: An abstract framework for argumentation with structured arguments. Argument and Computation **1**(2) (2010) 93–124
5. Modhil, S., Prakken, H.: A general account of argumentation with preferences. Artificial Intellligence (2013) in press.
6. Gorogiannis, N., Hunter, A.: Instantiating abstract argumentation with classical logic arguments: Postulates and properties. Artificial Intelligence **175**(9-10) (2011) 1479–1497
7. Caminada, M., Gabbay, D.: A logical account of formal argumentation. Studia Logica **93**(2-3) (2009) 109–145 Special issue: new ideas in argumentation theory.
8. Przymusinski, T.: The well-founded semantics coincides with the three-valued stable semantics. Fundamenta Informaticae **13**(4) (1990) 445–463
9. Eiter, T., Leone, N., Saccá, D.: On the partial semantics for disjunctive deductive databases. Ann. Math. Artif. Intell. **19**(1-2) (1997) 59–96
10. Caminada, M.: On the issue of reinstatement in argumentation. In Fischer, M., van der Hoek, W., Konev, B., Lisitsa, A., eds.: Logics in Artificial Intelligence; 10th European Conference, JELIA 2006, Springer (2006) 111–123 LNAI 4160.
11. Wu, Y., Caminada, M.: A labelling-based justification status of arguments. Studies in Logic **3**(4) (2010) 12–29
12. Caminada, M., Sá, S., Alcântara, J.: On the equivalence between logic programming semantics and argumentation semantics. Technical Report ABDN–CS–13–01, University of Aberdeen (2013)
13. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. Knowledge Engineering Review **26**(4) (2011) 365–410
14. Pollock, J.: Cognitive Carpentry. A Blueprint for How to Build a Person. MIT Press, Cambridge, MA (1995)
15. Jakobovits, H., Vermeir, D.: Robust semantics for argumentation frameworks. Journal of logic and computation **9(2)** (1999) 215–261