





An Abstract Argumentation and Logic Programming Comparison Based on 5-Valued Labellings

Samy Sá^(✉) and João Alcântara

Universidade Federal do Ceará, Fortaleza, Brazil
samy@ufc.br, jnando@lia.ufc.br

Abstract. Abstract argumentation and logic programming are two formalisms of non-monotonic reasoning that share many similarities. Previous studies contemplating connections between the two formalisms provided back and forth translations from one to the other and found they correspond in multiple different semantics, but not all. In this work, we propose a new set of five argument labels to revisit the semantic correspondences between abstract argumentation and logic programming. By doing so, we shed light on why the two formalisms are not absolutely equivalent. Our investigation lead to the specification of the novel least-stable semantics for abstract argumentation which corresponds to the L-stable semantics of logic programming.

Keywords: Abstract argumentation · Logic programming · Argument labellings

1 Introduction

Logic Programming (LP) and Abstract Argumentation Frameworks are two different formalisms widely used for the representation of knowledge and reasoning. Abstract Argumentation (AA) was itself inspired by logic programming in its origins, which naturally led to several studies concerning connections between them [5, 6, 8, 11, 14, 16]. One of the main approaches to observe those connections is based on the comparison of the different semantics proposed for each formalism. To that end, the first questions were raised and answered in [6], the work that originally introduced abstract argumentation: a translation from a logic program into an abstract argumentation framework was proposed and used to prove that the stable models (resp. the well-founded model) of a logic program correspond to the stable extensions (resp. the grounded extension) of its corresponding abstract argumentation framework. Other advances were made when [16] observed the equivalence between the complete semantics for abstract argumentation and the p-stable semantics for logic programs. Those particular semantics generalise many others in their respective formalisms, wielding a plethora of results gathered in [5] and recently expanded in [4]. One particular equivalence formerly expected to hold, however, could not be achieved, namely the correspondence between the semi-stable semantics from abstract argumentation [3] and the L-stable semantics from logic

programming [9]. The work we are about to present is largely motivated by that non-correspondence result.

In our efforts to understand why the semi-stable and L-stable semantics do not correspond to one another, we detected the fault is always related to some arguments whose attackees coincide; in their redundancy, one or more of those arguments would be irrelevant to the evaluation of those arguments they mutually attack. We were also able to identify that sink¹ arguments play a special role in pinpointing the culprits. To achieve our goals for this work, we follow the presentation of [5] while revising some key definitions: (i) the instantiation of abstract argumentation frameworks from logic programs is revised to include special arguments we call *default arguments* (which are always sinks) and (ii) we revise the definition of complete argument labellings [2] to use a different set of labels. Our revision of complete labellings follows [1, 10] to contemplate partial labellings and different categories of undecidedness. The resulting labellings, here on called 5-valued labellings, can isolate any arguments causing semi-stable and L-stable semantics to differ and are shown to preserve the main results in [5].

Based on those results, we introduce a novel semantics for abstract argumentation called the *least-stable* (or L-stable) semantics, which we show is equivalent to the L-stable LP semantics. The new AA semantics closes the previous gap preventing the claim that AA is actually equivalent to LP. Our results add further to the literature concerning semantics of non-monotonic reasoning formalisms, potentially allowing us to import proof procedures and implementations from formal argumentation to logic programming and vice-versa. Among other implications of our work, we set precedence to the proposal of new argumentation semantics based on partial labellings. We also establish that sink arguments have a role in the semantic evaluation of abstract argumentation frameworks, which may also help us identify new interesting AA semantics. Moreover, we show that if we restrict our attention to sink arguments, the standard 3-valued labellings are enough to capture the logic programming L-stable semantics.

2 Preliminaries

2.1 Abstract Argumentation

Abstract argumentation was introduced in [7] and most of the definitions in this section are based on that work. Concerning new definitions, we will compute the set of sink arguments of an abstract argumentation framework and use them to propose argumentation semantics in the form of 5-valued labellings, a core trait of our approach.

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

We may refer to AA frameworks simply as argumentation frameworks or AF's.

Definition 2. *Given an argumentation framework $AF = (Ar, att)$, the set of its sink arguments is given by $SINKS_{AF} = \{A \in Ar \mid \forall B, (A, B) \notin att\}$.*

¹ In graph theory, a sink node is one from which no edges emerge.

The traditional approach of [7] to semantics involves the identification of sets of arguments (called extensions) based on the concept of admissibility (see [7]). An alternative formalization of AA semantics was introduced in [2] where argument labellings are used instead of extensions. The argument labellings of [2] are functions attributing to each argument a single label from $\{\text{in}, \text{out}, \text{undec}\}$. In this work, we specialise the labels out , undec into two different labels each based on the sink arguments they attack. Our labelling concept follows the lead of [1], where the authors explore alternative sets of labels and special cases of undecidedness in argumentation semantics. We borrow some of the labels discussed in their work, namely the “*I don’t care*” (idc) label (originally proposed in [10]) and the “*I don’t know*” (idk) label ([1]) and we introduce a new label of our own, here dubbed “*It doesn’t matter if it’s out*” (ido). Hence, in our work, given an argumentation framework $\text{AF} = (Ar, \text{att})$, an *argument labelling* Al will be a function $Al : Ar \rightarrow \{\text{in}, \text{out}, \text{ido}, \text{idk}, \text{idc}\}$. We will start defining complete labellings, since the other relevant semantics are special cases of them.

Definition 3. *An argument labelling is called a complete argument labelling iff for each $A \in Ar$ it holds that*

- $Al(A) = \text{in}$ iff all B that attacks A has $Al(B) \in \{\text{out}, \text{ido}\}$.
- $Al(A) = \text{out}$ iff at least one B that attacks A has $Al(B) = \text{in}$ and A attacks no sink C with $Al(C) = \text{idk}$.
- $Al(A) = \text{ido}$ iff at least one B that attacks A has $Al(B) = \text{in}$ and A attacks at least one sink C with $Al(C) = \text{idk}$
- $Al(A) = \text{idk}$ iff at least one B that attacks A has $Al(B) \in \{\text{out}, \text{ido}\}$, no B that attacks A has $Al(B) = \text{in}$, and every sink C attacked by A has $Al(C) = \text{idk}$.
- $Al(A) = \text{idc}$ iff at least one B that attacks A has $Al(B) \in \{\text{out}, \text{ido}\}$, no B that attacks A has $Al(B) = \text{in}$, and at least one sink C attacked by A has $Al(C) \neq \text{idk}$.

As one can observe, the sink arguments influence the evaluation of arguments in our definition. For instance, the only difference between labelling an argument as out or ido is whether that argument attacks any idk -labelled sinks. As such, out and ido are merely specialisations of the out label from the standard 3-valued labellings of [2]. The same holds for idk and idc , as specialisations of undec from [2]. Intuitively, idk replaces undec in most cases; idc (resp. ido) is used when the status of a classically undec (resp. out) argument is irrelevant to the status of its attackee. In a way, the labels ido , idc allow us to sometimes evaluate one extra argument (using ido) or one less argument (using idc) as undecided. This is enough to ensure that our 5-valued complete labellings from Definition 3 are one to one related to the 3-valued complete labellings of [2] and preserve the complete semantics for abstract argumentation frameworks. The same holds for the grounded, preferred, stable and semi-stable semantics, as they are all particular cases of the complete semantics. Another straightforward result is

Proposition 1. *Let (Ar, att) be an AF and Al be one of its complete argument labellings. For every sink argument $C \in \text{SINKS}_{\text{AF}}$, it holds $Al(C) \in \{\text{in}, \text{out}, \text{idk}\}$.*

Given an argument labelling Al , we write $v(Al) = \{A \mid Al(A) = v\}$ for $v \in \{\text{in}, \text{out}, \text{ido}, \text{idk}, \text{idc}\}$. An argument labelling provides a partition of Ar , so we might as well present Al as a tuple $(\text{in}(Al), \text{out}(Al), \text{ido}(Al), \text{idk}(Al), \text{idc}(Al))$.

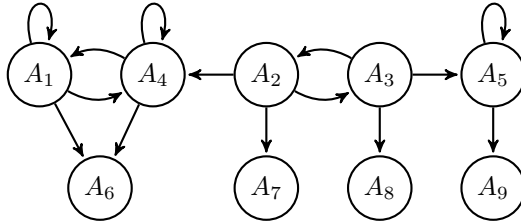
The next definition adapts from well-known results concerning the 3-valued argument labellings of [2,5]. Our adaptation maps the 3-valued label *undec* to $\text{idk} \cup \text{idc}$, the specialised labels we created from the standard *undec*.

Definition 4. Let \mathcal{AL} be the set of all complete argument labellings of AF. Then

- $Al \in \mathcal{AL}$ is grounded if $\nexists Al' \in \mathcal{AL}$ such that $\text{in}(Al') \subsetneq \text{in}(Al)$.
- $Al \in \mathcal{AL}$ is preferred if $\nexists Al' \in \mathcal{AL}$ such that $\text{in}(Al') \supsetneq \text{in}(Al)$.
- $Al \in \mathcal{AL}$ is stable if $\text{idk}(Al) \cup \text{idc}(Al) = \emptyset$.
- $Al \in \mathcal{AL}$ is semi-stable if $\nexists Al' \in \mathcal{AL}$ such that $\text{idk}(Al') \cup \text{idc}(Al') \subsetneq \text{idk}(Al) \cup \text{idc}(Al)$.

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

Example 1. The argumentation framework AF below has complete labellings $Al_1 = (\{\}, \{\}, \{\}, Ar, \{\})$, $Al_2 = (\{A_2, A_8\}, \{A_3, A_7\}, \{A_4\}, \{A_1, A_5, A_6, A_9\}, \{\})$, $Al_3 = (\{A_3, A_7, A_9\}, \{A_2, A_5, A_8\}, \{\}, \{A_1, A_4, A_6\}, \{\})$. The grounded labelling of AF is Al_1 and its preferred labellings are Al_2, Al_3 . AF has no stable labellings, but has two semi-stable labellings, namely Al_2, Al_3 . For contrast, the corresponding argument labellings in Caminada’s 3-valued approach are $Al'_1 = (\{\}, \{\}, Ar)$, $Al'_2 = (\{A_2, A_8\}, \{A_3, A_7, A_4\}, \{A_1, A_5, A_6, A_9\})$, $Al'_3 = (\{A_3, A_7, A_9\}, \{A_2, A_5, A_8\}, \{A_1, A_4, A_6\})$. We highlight the difference between Al_2 and Al'_2 due to $\text{idc}(Al_2) \neq \emptyset$.



It should be noticed that sink arguments were only attributed labels *in*, *out*, *idk*, as previously stated in Proposition 1.

2.2 Logic Programs and Semantics

In the current paper, we account for propositional normal logic programs², which we call logic programs or simply programs from now on. We will follow the presentation of logic programs and their semantics as done in [5], which, in turn, is based on [13].

Definition 5. A rule r is an expression

$$r : c \leftarrow a_1, \dots, a_n, \text{not } b_1, \dots, \text{not } b_m$$

($m, n \geq 0$) where c , each a_i ($1 \leq i \leq n$) and each b_j ($1 \leq j \leq m$) are atoms and *not* represents negation as failure. A literal is either an atom a (positive literal) or a

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

negated atom $\text{not } a$ (negative literal). Given a rule r as above, c is called the head of r , which we denote $\text{head}(r)$, and $a_1, \dots, a_n, \text{not } b_1, \dots, \text{not } b_m$ is called the body of r , denoted $\text{body}(r)$. Further, we divide $\text{body}(r)$ in two sets $\text{body}^+(r) = \{a_1, \dots, a_n\}$ and $\text{body}^-(r) = \{\text{not } b_1, \dots, \text{not } b_m\}$. A logic program (or simply a program or LP) P is then defined as a finite set of rules. If every $r \in P$ has $\text{body}^-(r) = \emptyset$, then P is positive. The Herbrand Base of P is the set HB_P of all atoms appearing in the program.

Definition 6. A 3-valued Herbrand Interpretation I of a logic program P is a pair $\langle T, F \rangle$ with $T, F \subseteq HB_P$ and $T \cap F = \emptyset$. The atoms in T are said to be true, the atoms in F are said to be false and the atoms in $HB_P \setminus (T \cup F)$ are said to be undefined.

Definition 7. A 3-valued Herbrand interpretation I of a logic program P is called a model if $I(\text{head}(r)) \geq \min(\{I(l) \mid l \in \text{body}(r)\})$ for each $r \in P$, according to the order $\text{true} > \text{undefined} > \text{false}$.

Let I be a 3-valued Herbrand Interpretation of P , the reduct of P with respect to I (written P/I) can be obtained replacing each occurrence of a naf-literal $\text{not } c$ in P “true” if $c \in F$, by “false” if $c \in T$, or by “undefined” otherwise. In this context, “true”, “false” and “undefined” are special atoms not occurring in P which are necessarily evaluated according to their naming conventions. This procedure ensures P/I has no instances of negative literals. As consequence, P/I has a unique least 3-valued model [13] hereby denoted $\Psi_P(I) = \langle T_\Psi, F_\Psi \rangle$ ³ with minimal T_Ψ and maximal F_Ψ (w.r.t. set inclusion) such that, for every $a \in HB_P$:

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

Definition 8. Let $I = \langle T, F \rangle$ be a 3-valued Herbrand Interpretation of program P .

- I is a partial stable (or P -stable) model of P iff $\Psi_P(I) = I$.
- T is a well-founded model of P iff I is a P -stable model of P where T is minimal (w.r.t. set inclusion) among all P -stable models of P .
- T is a regular model of P iff I is a P -stable model of P where T is maximal (w.r.t. set inclusion) among all P -stable models of P .
- T is a stable model of P iff I is a P -stable model of P where $T \cup F = HB_P$.
- T is an L -stable model of P iff I is a P -stable model of P where $T \cup F$ is maximal (w.r.t. set inclusion) among all P -stable models of P .

While some of the above definitions are not standard in logic programming literature, their equivalence is proved in [5]. This approach is favored in our work due to the structural similarities to Definition 4, simplifying the proof of some results.

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

Example 2. Consider the following logic program P :

$$\begin{array}{l} r_1 : c \leftarrow \text{not } c \quad r_4 : c \leftarrow \text{not } c, \text{not } a \\ r_2 : a \leftarrow \text{not } b \quad r_5 : g \leftarrow \text{not } g, \text{not } b \\ r_3 : b \leftarrow \text{not } a \end{array}$$

³ The above definition consists of a least fix-point of the immediate consequence operator Ψ defined in [13], which is guaranteed to exist and be unique for positive programs.

This program has three P-stable models: $I_1 = \langle \{ \}, \{ \} \rangle$, $I_2 = \langle \{a\}, \{b\} \rangle$, $I_3 = \langle \{b\}, \{a, g\} \rangle$. P has $\{ \}$ for its well-founded model (always unique), $\{a\}$, $\{b\}$ for its regular models, no stable models and only $\{b\}$ as an L-stable model.

3 From Logic Programs to Argumentation Frameworks

In this section, we review the procedure used to instantiate argumentation frameworks based on logic programs from [5] with slight updates to each step. Our updates are required to include special arguments we call *default arguments*, which are necessarily sinks, in the result. By design, they will be the only sinks in the output of our procedure.

3.1 Step 1: AF Instantiation

Given a normal logic program P , we recursively instantiate the following *arguments*:

Definition 9. *Let P be a logic program.*

- If $r : c \leftarrow \text{not } b_1, \dots, \text{not } b_m$ is a rule in P then r is also an argument (say A) with $\text{Conc}(A) = c$, $\text{Rules}(A) = \{r\}$, $\text{VuL}(A) = \{b_1, \dots, b_m\}$ and $\text{Sub}(A) = \{A\}$.
- If $r : c \leftarrow a_1, \dots, a_n, \text{not } b_1, \dots, \text{not } b_m$ is a rule in P and for each $a_i \in \text{body}^+(r)$ there is an argument A_i with $\text{Conc}(A_i) = a_i$ and $r \notin \text{Rules}(A_i)$ then

$$c \leftarrow (A_1), \dots, (A_n), \text{not } b_1, \dots, \text{not } b_m$$

is an argument (say A) with $\text{Conc}(A) = c$, $\text{Rules}(A) = \bigcup \text{Rules}(A_i) \cup \{r\}$, $\text{VuL}(A) = \bigcup \text{VuL}(A_i) \cup \{b_1, \dots, b_m\}$ and $\text{Sub}(A) = \bigcup \text{Sub}(A_i) \cup \{A\}$.

- For each different $\text{head}(r)$ where $r \in P$, there is a default argument (say $\text{not } A$) such that $\text{Conc}(\text{not } A) = \text{not } \text{head}(r)$, $\text{Rules}(\text{not } A) = \emptyset$, $\text{VuL}(\text{not } A) = \{\text{head}(r)\}$ and $\text{Sub}(\text{not } A) = \{\text{not } A\}$.

Above, $\text{Conc}(A)$, $\text{Rules}(A)$, $\text{VuL}(A)$ and $\text{Sub}(A)$ respectively refer to what we call the conclusion, rules, vulnerabilities and subarguments of A . Our definition is very similar to the one in [5], which consists of only the first two bullet points above. The difference is the third bullet point, which is required to instantiate the default arguments.

Example 3. Based on the logic program P from Example 2 we can instantiate: $A_i = r_i$, $1 \leq i \leq 5$, based on the first bullet point of Definition 9; $A_6 = \text{not } c$, $A_7 = \text{not } a$, $A_8 = \text{not } b$, $A_9 = \text{not } g$, based on the third bullet point of Definition 9. No arguments are instantiated following the second bullet point because all $r \in P$ have $\text{body}^+(r) = \emptyset$. To exemplify the components of some of those arguments, A_1 has $\text{Conc}(A_1) = c$, $\text{Rules}(A_1) = \{r_1\}$, $\text{VuL}(A_1) = \{c\}$ and $\text{Sub}(A_1) = \{A_1\}$. A_6 has $\text{Conc}(A_6) = \text{not } c$, $\text{Rules}(A_6) = \emptyset$, $\text{VuL}(A_6) = \{c\}$ and $\text{Sub}(A_6) = \{A_6\}$.

The next step is to determine the attack relation of the argumentation framework: an argument attacks another iff its conclusion is a vulnerability of the latter.

Definition 10. *Let A and B be arguments⁴. We say A attacks B iff $\text{Conc}(A) \in \text{VuL}(B)$.*

⁴ According to Definition 9.

Example 4. The argument A_1 from Example 3 consists of $r : c \leftarrow \text{not } c$ and has $\text{Conc}(A_1) = c$ and $\text{VuL}(A_1) = \{c\}$; since $\text{Conc}(A_1) \in \text{VuL}(A_1)$, A_1 attacks itself. A_6 , in comparison, attacks no arguments, but is also attacked by A_1 .

Definition 11. *The argumentation framework AF_P associated with a program P is $\text{AF}_P = (Ar_P, att_P)$, where Ar_P is the set of all arguments from P (Definition 9) and $att_P = \{(A, B) \in Ar \times Ar \mid \text{Conc}(A) \in \text{VuL}(B)\}$ (Definition 10).*

Given AF_P , the computation of $\text{SINKS}_{\text{AF}_P}$ retrieves its set of default arguments.

Example 5. Following the names of arguments used in Example 3, the AF_P corresponding to P from Example 2 is precisely the framework depicted in Example 1.

3.2 Step 2: Applying Argumentation Semantics

Once we have AF_P , we will be interested in its semantics from the standpoint of abstract argumentation, which are given by Definition 3.

3.3 Step 3: Computing Conclusion Labellings

The goal of this step is to translate the 5-valued argument labellings obtained in the previous step to sets of conclusions comparable to the models of the original program, which are in turn 3-valued. For this reason, conclusions will be labeled exclusively within the elements of $\{\text{in}, \text{out}, \text{undec}\}$. In what follows, we will refer to the necessary sets of labels as $V_5 = \{\text{in}, \text{out}, \text{ido}, \text{idk}, \text{idc}\}$ and $V_3 = \{\text{in}, \text{out}, \text{undec}\}$.

Definition 12. *Let P be a program. A conclusion labelling is a map $Cl : HB_P \rightarrow V_3$.*

As in [5], we follow the approach of [15]: for each conclusion, we map the labels of arguments that yield it (taken from V_5) to a label in V_3 . The label attributed to each $a \in HB_P$ is then the best one⁵ according to the total order $\text{in} > \text{undec} > \text{out}$.

Definition 13. *The mapping $\sigma : V_5 \rightarrow V_3$ is such that $\sigma(\text{in}) = \text{in}$, $\sigma(\text{out}) = \sigma(\text{ido}) = \text{out}$, $\sigma(\text{idk}) = \sigma(\text{idc}) = \text{undec}$.*

Definition 14. *Let $\text{AF}_P = (Ar_P, att_P)$ be the AF associated with a program P and Al be an argument labelling of AF_P . We say that Cl is the associated conclusion labelling of Al iff Cl is a conclusion labelling such that for each $c \in HB_P$ it holds that $Cl(c) = \max(\{\sigma(Al(A)) \mid \text{Conc}(A) = c\} \cup \{\text{out}\})$ where $\text{in} > \text{undec} > \text{out}$.*

Given a conclusion labelling Cl , we write $\text{in}(Cl)$ to denote $\{c \mid Cl(c) = \text{in}\}$, $\text{out}(Cl)$ for $\{c \mid Cl(c) = \text{out}\}$ and $\text{undec}(Cl)$ for $\{c \mid Cl(c) = \text{undec}\}$. Because a conclusion labelling defines a partition of HB_P into sets of in/out/undec-labelled conclusions, we may sometimes write Cl as the triple $(\text{in}(Cl), \text{out}(Cl), \text{undec}(Cl))$.

⁵ In case there is no argument for a particular conclusion, it will be simply labelled out.

Example 6. Recall the complete argument labellings of AF in Example 1 are: $Al_1 = (\{\}, \{\}, \{\}, Ar, \{\})$, $Al_2 = (\{A_2, A_8\}, \{A_3, A_7\}, \{A_4\}, \{A_1, A_5, A_6, A_9\}, \{\})$, $Al_3 = (\{A_3, A_7, A_9\}, \{A_2, A_5, A_8\}, \{\}, \{A_1, A_4, A_6\}, \{\})$. Furthermore, remember $\text{Conc}(A_1) = \text{Conc}(A_4) = c$ (Example 3). Observe that $Al_2(A_4) = \text{ido}$ and $Al_2(A_1) = \text{idk}$. Given that $\sigma(\text{ido}) = \text{out}$ and $\sigma(\text{idk}) = \text{undec}$, we will find $Cl_2(c) = \text{undec}$ in the corresponding conclusion labelling Cl_2 . Following Definition 14, we produce the conclusion labellings $Cl_1 = (\{\}, \{\}, \{a, b, c, g\})$, $Cl_2 = (\{a\}, \{b\}, \{c, g\})$, and $Cl_3 = (\{b\}, \{a, g\}, \{c\})$ respectively associated to Al_1, Al_2, Al_3 . Please observe that $(\text{in}(Cl_i), \text{out}(Cl_i))$ ($1 \leq i \leq 3$) are precisely the p -stable models of P (Example 2).

4 Semantic Correspondences

In this section, we overview results concerning semantic relations between abstract argumentation and logic programming. We show that our approach based on 5-valued labellings preserves the original results and provides us a new correspondence.

First, we must formally define functions to map argument labellings to conclusion labellings and vice-versa. Our intention is to later prove they are each other's inverse. The function AL2CL below simply follows Definition 14, while CL2AL is introduced anew. In the latter, we will find that an argument is out or ido in the resulting argument labelling if some of its vulnerabilities are in in the original conclusion labelling. Deciding if that argument is out or ido will depend solely on whether it attacks an undec-labelled sink. The case for idk and idc arguments is similarly based on undec and the status of the sink arguments they attack.

Definition 15. *Let P be a program and AF_P be its associated argumentation framework. Let \mathcal{AL} be the set of all argument labellings of AF_P and let \mathcal{CL} be the set of all conclusion labellings of AF_P .*

- We define a function AL2CL: $\mathcal{AL} \rightarrow \mathcal{CL}$ such that for each $Al \in \mathcal{AL}$, $\text{AL2CL}(Al)$ is the associated conclusion labelling of Al (Definition 14).
- We define a function CL2AL : $\mathcal{CL} \rightarrow \mathcal{AL}$ such that for each $Cl \in \mathcal{CL}$ and each $A \in \text{Ar}_P$ it holds that:
 - $\text{CL2AL}(Cl)(A) = \text{in}$ iff $\text{Vul}(A) \subseteq \text{out}(Cl)$
 - $\text{CL2AL}(Cl)(A) = \text{out}$ iff $\text{Vul}(A) \cap \text{in}(Cl) \neq \emptyset$ and $\{\text{Conc}(B) \mid B \in \text{SINKS}_{\text{AF}_P} \text{ and } (A, B) \in \text{att}\} \cap \text{undec}(Cl) = \emptyset$
 - $\text{CL2AL}(Cl)(A) = \text{ido}$ iff $\text{Vul}(A) \cap \text{in}(Cl) \neq \emptyset$ and $\{\text{Conc}(B) \mid B \in \text{SINKS}_{\text{AF}_P} \text{ and } (A, B) \in \text{att}\} \cap \text{undec}(Cl) \neq \emptyset$
 - $\text{CL2AL}(Cl)(A) = \text{idk}$ iff $\text{Vul}(A) \cap \text{in}(Cl) = \emptyset$, while $\text{Vul}(A) \setminus \text{out}(Cl) \neq \emptyset$ and $\text{Conc}(B) \mid B \in \text{SINKS}_{\text{AF}_P} \text{ and } (A, B) \in \text{att} \subseteq \text{undec}(Cl)$
 - $\text{CL2AL}(Cl)(A) = \text{idc}$ iff $\text{Vul}(A) \cap \text{in}(Cl) = \emptyset$, while $\text{Vul}(A) \setminus \text{out}(Cl) \neq \emptyset$ and $\{\text{Conc}(B) \mid B \in \text{SINKS}_{\text{AF}_P} \text{ and } (A, B) \in \text{att}\} \setminus \text{undec}(Cl) \neq \emptyset$

Theorem 1. *When restricted to complete argument labellings and complete conclusion labellings, the functions AL2CL and CL2AL are bijections and each other's inverse.*

We are now ready to present some key results of our work. In what follows, let P be a logic program and AF_P be its associated argumentation framework. Further, assume $Cl = (\text{in}(Cl), \text{out}(Cl), \text{undec}(Cl))$ is an arbitrary conclusion labelling of AF_P . The results below are based on our σ -mapping from 5-valued labellings to the standard in/out/undec-labellings from [2] and theorems from [5] connecting the standard labellings to logic programming models. In summary, the following theorems assure our 5-valued labellings preserve all the results from [5].

Theorem 2. *Let Al be a complete argument labelling of AF_P . Then $\text{AL2CL}(Al) = Cl$ iff $\langle \text{in}(Cl), \text{out}(Cl) \rangle$ is a p -stable model of P .*

As our definition of complete argument labelling corresponds to the definition of [2] and the characterization of in is equivalent in both settings, we can borrow the following result from [5]:

Lemma 1. *Let P be a program and $\text{AF}_P = (Ar, att)$ be its associated argumentation framework. Let Al_1 and Al_2 be complete argument labellings of AF_P , and $Cl_1 = \text{AL2CL}(Al_1)$ and $Cl_2 = \text{AL2CL}(Al_2)$. It holds that*

1. $\text{in}(Al_1) \subseteq \text{in}(Al_2)$ iff $\text{in}(Cl_1) \subseteq \text{in}(Cl_2)$,
2. $\text{in}(Al_1) = \text{in}(Al_2)$ iff $\text{in}(Cl_1) = \text{in}(Cl_2)$, and
3. $\text{in}(Al_1) \subsetneq \text{in}(Al_2)$ iff $\text{in}(Cl_1) \subsetneq \text{in}(Cl_2)$.

From Theorem 2 and Lemma 1, we obtain:

Theorem 3. *Let Al be the grounded argument labelling of AF_P . Then $\text{AL2CL}(Al) = Cl$ iff $\text{in}(Cl)$ is the well-founded model of P .*

Theorem 4. *Let Al be a preferred argument labelling of AF_P . Then $\text{AL2CL}(Al) = Cl$ iff $\text{in}(Cl)$ is a regular model of P .*

In addition, we know $\text{idk}, \text{idc} = \emptyset$ in a complete argument labelling Al iff $\text{undec} = \emptyset$ in $\text{AL2CL}(Al) = Cl$ (Definition 14). Thus,

Theorem 5. *Let Al be a stable argument labelling of AF_P . Then $\text{AL2CL}(Al) = Cl$ iff $\text{in}(Cl)$ is a stable model of P .*

Following the queue of the theorems above, we can as well consider the conclusion labellings where $\text{in}(Cl)$ is an L-stable model of P . The natural candidate argument labellings to correspond to those conclusion labellings would be the semi-stable argument labellings, since both semi-stable and L-stable semantics respectively minimize undecided arguments and undecided atoms. Unfortunately, in the same way as in [5], we find there are cases where the semi-stable semantics does not exactly correspond to the L-stable semantics. Our running example is enough to show that: as one can observe from the computed labellings in Example 6, the framework from Example 1 has two semi-stable argument labellings (hence two semi-stable conclusion labellings), namely Al_2, Al_3 , but Cl_3 is the sole complete conclusion labelling for which $\text{in}(Cl)$ is an L-stable model of P . In other words, P has a single L-stable model, but AF_P has two semi-stable argument labellings. Differently from previous works, however, our labellings isolated a possible culprit to that difference in Example 6, for we have $Al_2(A_4) = \text{idc}$. But why may it be a culprit? Because if A_4 was instead labelled idk in Al_2 , only Al_3 would be semi-stable, fixing the expected correspondence.

5 The L-Stable Semantics for Abstract Argumentation Frameworks

The discussion about the semi-stable and L-stable semantics by the end of Sect. 4 led us to the suggestion that changing the labels of some particular arguments could lead those semantics to coincide. Unfortunately, such changes are far from trivial, as changing the label of an argument is likely to affect the labels of others. Fortunately, there are other parts of the three-step procedure from Sect. 3 that can be easily adapted to achieve a similar result: the conception of a new abstract argumentation semantics that corresponds perfectly to the L-stable semantics from logic programming.

Definition 16. *Let \mathcal{AL} be the set of all complete argument labellings of AF . Then Al is a least-stable (or L-stable) argument labelling of AF if $Al \in \mathcal{AL}$ and $\nexists Al' \in \mathcal{AL}$ such that $\text{idk}(Al') \cup \text{ido}(Al') \subsetneq \text{idk}(Al) \cup \text{ido}(Al)$.*

Lemma 2. *Let Al be a complete labelling of an argumentation framework and $Cl = \text{AL2CL}(Al)$. Then $\exists A \in \text{idk}(Al) \cup \text{ido}(Al)$ with $\text{Conc}(A) = c$ iff $Cl(c) = \text{undec}$.*

Theorem 6. *Let Al be a least-stable argument labelling of AF_P . Then $\text{AL2CL}(Al) = Cl$ iff $\text{in}(Cl)$ is an L-stable model of P .*

This correspondence ensures the L-stable semantic for argumentation frameworks has similar properties to the semi-stable semantics.

Corollary 1. *Every AF has at least one L-stable labelling.*

Corollary 2. *If Al is a stable labelling of AF_P , then Al is an L-stable labelling of AF_P .*

Corollary 3. *If AF_P has at least one stable labelling, then Al is an L-stable labelling of AF_P iff Al is a semi-stable labelling of AF_P iff Al is a stable labelling of AF_P .*

However, differently than semi-stable, the L-stable AA semantic captures the L-stable semantics for LP.

6 On the Role of Sink Arguments

We have affirmed sink arguments together with 5-valued labellings are able to isolate the possible culprits for the difference between the semi-stable and L-stable semantics. Now we resume this discussion to provide a more detailed account on the role played by sink arguments in argumentation semantics. In particular, we will emphasize the following two points:

- When considering only the sink arguments obtained in our translation (Definition 9), 3-valued labellings suffice to capture the equivalence between logic programming semantics and abstract argumentation semantics (including L-stable).
- The sink arguments together with the 5-valued labellings offer a more fine-grained view of the notion of undecidedness when compared with 3-valued labellings.

The sink arguments can be employed to obtain the conclusion labellings associated with complete argument labellings:

Theorem 7. *Let P be a logic program, $\text{AF}_P = (Ar_P, att_P)$ be its associated argumentation framework and Al be a complete 5-valued argument labelling of AF_P . We define $Cl : HB_P \rightarrow V_3$:*

$$Cl(c) = \begin{cases} \overline{\sigma(Al(A))} & \text{if } \exists A \in Ar_P \text{ such that } \text{Conc}(A) = \text{not } c \\ \text{out} & \text{otherwise} \end{cases}$$

in which $\overline{\text{in}} = \text{out}$, $\overline{\text{out}} = \text{in}$ and $\overline{\text{undec}} = \text{undec}$. Then $\text{AL2CL}(Al) = Cl$ and $(\text{in}(Cl), \text{out}(Cl))$ is a p -stable model of P .

From Definition 9, there is at most one argument A in AF_P such that $\text{Conc}(A) = \text{not } c$ for each $c \in HB_P$. Therefore, the function Cl above is well-defined.

Theorem 8. *Let P be a program, $\text{AF}_P = (Ar_P, att_P)$ be its associated argumentation framework, Al be a 5-valued argument labelling of AF_P and $\text{AL3}(Al) : \text{SINKS}_{\text{AF}_P} \rightarrow \{\text{in}, \text{out}, \text{undec}\}$, in which for every $A \in \text{SINKS}_{\text{AF}_P}$, we have $\text{AL3}(Al)(A) = \sigma(Al(A))$. Let also \mathcal{AL} be the set of all complete argument labellings of AF_P . Then*

- *Al is the grounded argument labelling of AF_P iff $Al \in \mathcal{AL}$ and $\nexists Al' \in \mathcal{AL}$ such that $\text{out}(\text{AL3}(Al')) \subsetneq \text{out}(\text{AL3}(Al))$.*
- *Al is a preferred argument labelling of AF_P iff $Al \in \mathcal{AL}$ and $\nexists Al' \in \mathcal{AL}$ such that $\text{out}(\text{AL3}(Al')) \supsetneq \text{out}(\text{AL3}(Al))$.*
- *Al is a stable argument labelling of AF_P iff $Al \in \mathcal{AL}$ and $\text{undec}(\text{AL3}(Al)) = \emptyset$.*
- *Al is an L-stable argument labelling of AF_P iff $Al \in \mathcal{AL}$ and $\nexists Al' \in \mathcal{AL}$ such that $\text{undec}(\text{AL3}(Al')) \subsetneq \text{undec}(\text{AL3}(Al))$.*

Theorem 8 highlights a distinction between the semi-stable and L-stable semantics over an argumentation framework AF_P obtained from a logic program P : the semi-stable argument labellings of AF_P are those complete argument labellings with minimal set of undec-labelled arguments, whereas the L-stable argument labellings of AF_P are those complete argument labellings with minimal set of undec-labelled sinks. Therefore, the labels assigned to the sink arguments of AF_P suffice to know if a labelling is grounded, preferred, stable and L-stable, but they may not suffice to know if it is semi-stable. Revisiting Example 6, we have both Al_2 and Al_3 are complete labellings with minimal undec arguments (semi-stable, see Example 1); in contrast, Al_3 is the only complete labelling with minimal undec sinks (L-stable, see Example 2).

One may then wonder if the 5-valued labelling semantics we introduced in this paper are of some interest on their own. First, observe that the 5 labels are a necessary condition to define the L-stable argumentation semantics (complete labellings with minimal $(\text{id}_o \cup \text{id}_k)$ -valued arguments). Further, although one can mimic the L-stable logic programming semantics with 3-valued labellings, it will work as expected just for argumentation frameworks obtained from logic programs through Definition 11; in an arbitrary argumentation framework (one that does not account for default arguments), unlike our definition of L-stable semantics based on 5 labels, this adaptation of the 3-valued approach to capture L-stable will not produce meaningful results.

Besides, we emphasize the role played by sink arguments in our 5-valued labellings: in an argumentation framework AF without sink arguments, its complete 3-valued labellings coincide precisely with its 5-valued counterpart by simply replacing `undec` for `idk` in the respective labellings. This means that the semi-stable and L-stable argumentation semantics collapse into each other for those frameworks! What is more, owing to the 5-valued labellings, we can define new semantics with interesting properties: for instance, consider a new argumentation semantics (let us call it “I-stable”) defined by the complete labellings with minimal `idk` arguments. In an argumentation framework without sink arguments, it will coincide with both semi-stable and L-stable argumentation semantics and yet it will possibly produce distinct results for some argumentation frameworks. As such, these semantics offer three different views of undecidedness in which the difference between them is positively determined by the sink arguments.

7 Conclusion

This work was largely motivated by the curious result of [5] stating the semi-stable semantics from abstract argumentation is not equivalent to the L-stable semantics from logic programming. Both semantics are based on the minimization of undecided arguments and atoms in their respective domains, share very similar properties, and yet are different. Our efforts to understand the differences between the semantics led us to consider the instantiation of special arguments in logic programming we called default arguments and the specialization of the standard 3-valued argument labellings from [2] into more expressive 5-valued partial labellings (as previously considered in [1]). To accomplish our goals, we considered the labels `in`, `out`, `ido`, `idk`, `idc` and introduced a corresponding definition of complete labellings revising the meanings of `out`, `idk`, `idc` and introducing the new label `ido`, here dubbed “It doesn’t matter if it’s out”. Following the revised concepts, we showed our approach preserves well-known results in the semantic comparison of abstract argumentation and logic programming. While we did not try to resolve the semi-stable-L-stable equivalence problem, we shed light into it, as our 5-valued labellings can isolate the possible culprit arguments causing the differences based on the labels `ido`, `idc`. Instead, we introduced a new semantics for abstract argumentation frameworks: the least-stable (or L-stable) semantics. What is more, we proved our newly established semantics to be equivalent to the logic programming L-stable semantics for normal programs and their corresponding instantiated argumentation frameworks. The non-equivalence result proved in [5] was based on a particular procedure for the instantiation of argumentation frameworks from logic programs. The 5-valued labellings, the sink arguments and our revision of their procedure are the key changes to obtain an argumentation-based semantics corresponding to L-stable. Indeed, we show that if we restrict our attention to sink arguments, the standard 3-valued labelling is enough to characterize the logic programming semantics, including L-stable.

It is worth noticing that partial labellings were also employed by [12] in an algorithm for enumerating preferred extensions of an argumentation framework. As we do, they also consider five labels, namely `in`, `out`, `undec`, `blank`, and `must-out` labels. In

their algorithm, all arguments of the framework are initially blank and must-out is used at run time to flag arguments that are candidate to be labelled out.

Natural ramifications of this work include an in-depth investigation of properties of the least-stable argumentation semantics. Our newly defined 5-valued complete labellings may also lead to the proposal of other interesting semantics as we try to maximize or minimize different combinations of the labels at our disposal. We also intend to investigate whether sink arguments play other relevant roles in the characterization or computation of argumentation semantics.

References

1. Baroni, P., Giacomin, M., Liao, B.: I don't care, i don't know ... i know too much! on incompleteness and undecidedness in abstract argumentation. In: Eiter, T., Strass, H., Truszczynski, M., Woltran, S. (eds.) *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation*. LNCS (LNAI), vol. 9060, pp. 265–280. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-14726-0_18
2. Caminada, M.: On the issue of reinstatement in argumentation. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) *JELIA 2006*. LNCS (LNAI), vol. 4160, pp. 111–123. Springer, Heidelberg (2006). https://doi.org/10.1007/11853886_11
3. Caminada, M.: Semi-stable semantics. In: Dunne, P.E., Bench-Capon, T.J.M. (eds.) *Computational Models of Argument: Proceedings of COMMA 2006, September 11–12, 2006, Liverpool, UK*. *Frontiers in Artificial Intelligence and Applications*, vol. 144, pp. 121–130. IOS Press (2006). <http://www.booksonline.iospress.nl/Content/View.aspx?piid=1932>
4. Caminada, M., Harikrishnan, S., Sá, S.: Comparing logic programming and formal argumentation; the case of ideal and eager semantics. In: *Argument and Computation Pre-press*, pp. 1–28 (2021). <https://doi.org/10.3233/AAC-200528>
5. Caminada, M., Sá, S., Alcântara, J., Dvořák, W.: On the equivalence between logic programming semantics and argumentation semantics. *Int. J. Approx. Reason.* **58**, 87–111 (2015)
6. Dung, P.: An argumentation procedure for disjunctive logic programs. *J. Logic Program.* **24**, 151–177 (1995)
7. Dung, P.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n -person games. *Artif. Intell.* **77**, 321–357 (1995)
8. Dvořák, W., Gaggl, S.A., Wallner, J.P., Woltran, S.: Making use of advances in answer-set programming for abstract argumentation systems. In: Tompits, H., Abreu, S., Oetsch, J., Pührer, J., Seipel, D., Umeda, M., Wolf, A. (eds.) *INAP/WLP -2011*. LNCS (LNAI), vol. 7773, pp. 114–133. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41524-1_7
9. Eiter, T., Leone, N., Saccá, D.: On the partial semantics for disjunctive deductive databases. *Ann. Math. Artif. Intell.* **19**(1–2), 59–96 (1997)
10. Jakobovits, H., Vermeir, D.: Robust semantics for argumentation frameworks. *J. Logic Comput.* **9**(2), 215–261 (1999)
11. Nieves, J.C., Cortés, U., Osorio, M.: Preferred extensions as stable models. *Theory Pract. Logic Program.* **8**(4), 527–543 (2008)
12. Nofal, S., Atkinson, K., Dunne, P.E.: Algorithms for decision problems in argument systems under preferred semantics. *Artif. Intell.* **207**, 23–51 (2014)
13. Przymusiński, T.: The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae* **13**(4), 445–463 (1990)

14. Toni, F., Sergot, M.: Argumentation and answer set programming. In: Balduccini, M., Son, T.C. (eds.) *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*. LNCS (LNAI), vol. 6565, pp. 164–180. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20832-4_11
15. Wu, Y., Caminada, M.: A labelling-based justification status of arguments. *Stud. Logic* **3**(4), 12–29 (2010)
16. Wu, Y., Caminada, M., Gabbay, D.M.: Complete extensions in argumentation coincide with 3-valued stable models in logic programming. *Studia Logica* **93**(1–2), 383–403 (2009)