TPLP **24** (6): 1208–1236, 2024.

doi:10.1017/S1471068424000188 First published online 6 November 2024

On the Equivalence between Logic Programming and SETAF

JOÃO ALCÂNTARA, RENAN CORDEIRO and SAMY SÁ
Federal University of Ceará, Fortaleza, Brazil
(*e-mails:* jnando@dc.ufc.br, renandcsc@alu.ufc.br, samy@ufc.br)

submitted 30 September 2023; revised 18 July 2024; accepted 21 July 2024

Abstract

A framework with sets of attacking arguments (*SETAF*) is an extension of the well-known Dung's Abstract Argumentation Frameworks (*AAF*s) that allows joint attacks on arguments. In this paper, we provide a translation from Normal Logic Programs (*NLP*s) to *SETAF*s and vice versa, from *SETAF*s to *NLP*s. We show that there is pairwise equivalence between their semantics, including the equivalence between *L*-stable and semi-stable semantics. Furthermore, for a class of *NLP*s called Redundancy-Free Atomic Logic Programs (*RFALP*s), there is also a structural equivalence as these back-and-forth translations are each other's inverse. Then, we show that *RFALP*s are as expressive as *NLP*s by transforming any *NLP* into an equivalent *RFALP* through a series of program transformations already known in the literature. We also show that these program transformations are confluent, meaning that every *NLP* will be transformed into a unique *RFALP*. The results presented in this paper enhance our understanding that *NLP*s and *SETAF*s are essentially the same formalism.

KEYWORDS: argumentation, logic programming semantics, program transformations

1 Introduction

Argumentation and logic programming are two of the most successful paradigms in artificial intelligence and knowledge representation. Argumentation revolves around the idea of constructing and evaluating arguments to determine the acceptability of a claim. It models complex reasoning by considering various pieces of evidence and their interrelationships, making it a powerful tool for handling uncertainty and conflicting information. On the other hand, logic programming provides a formalism for expressing knowledge and defining computational processes through a set of logical rules.

In this scenario, the Abstract Argumentation Frameworks (*AAF*s) proposed by Dung (1995b) in his seminal paper have exerted a dominant influence over the development of formal argumentation. We can depict such frameworks simply as a directed graph whose nodes represent arguments and edges represent the attack relation between them. Indeed, in *AAF*s, the content of these arguments is not considered, and the attack relation stands as the unique relation. The simplicity and elegance of *AAF*s have made them an appealing formalism for computational applications.

In Dung's proposal, the semantics for *AAF*s are given in terms of extensions, which are sets of arguments satisfying certain criteria of acceptability. Naturally, different criteria of acceptability will lead to different extension-based semantics, including Dung's original concepts of complete, stable, preferred, and grounded semantics (Dung 1995b) and semi-stable semantics (Verheij 1996; Caminada 2006). A richer characterization based on labelings was proposed by Caminada and Gabbay (2009) to describe these semantics. Differently from extensions, which explicitly regard solely the accepted arguments, the labeling-based approach permits a more fine-grained setting, where each argument is assigned a label `in`, `out`, or `undec`. Intuitively, we accept an argument labeled as `in`, reject one labeled as `out`, and consider one labeled as `undec` as undecided, meaning it is neither accepted nor rejected.

Despite providing distinct perspectives on reasoning and decision-making, argumentation and logic programming have clear connections. Indeed, we can see in Dung's (1995b) work how to translate a Normal Logic Program (*NLP*) into an *AAF*. Besides, the author proved that stable models (resp., the well-founded model) of an *NLP* correspond to stable extensions (resp., the grounded extension) of the associated *AAF*. These results led to several studies concerning connections between argumentation and logic programming (Dung 1995a; Nieves *et al.* 2008; Wu *et al.* 2009; Toni and Sergot 2011; Dvořák *et al.* 2013; Caminada *et al.* 2015b, 2022). In particular, Wu *et al.* (2009) established the equivalence between complete semantics and partial stable semantics. These semantics generalize a series of other relevant semantics for each system, as extensively documented by Caminada *et al.* (2015b). However, one equivalence formerly expected to hold remained elusive: the correspondence between the semi-stable semantics (Caminada 2006) in *AAF* and the *L*-stable semantics in *NLP* (Eiter *et al.* 1997) could not be attained. Caminada *et al.* (2015b) even showed that with their proposed translation from *NLP*s to *AAF*s, there cannot be a semantics for *AAF*s equivalent to *L*-stable semantics.

Caminada and Schulz (2017) demonstrated how to translate Assumption-Based Argumentation (*ABA*) (Bondarenko *et al.* 1997; Dung *et al.* 2009; Toni 2014) to *NLP*s and how this translation can be reapplied for a reverse translation from *NLP*s to *ABA*. Curiously, the problematic direction here is from *ABA* to *NLP*. Caminada and Schulz (2017) proved that with their translation, there cannot be a semantics for *NLP*s equivalent to the semi-stable semantics (Schulz and Toni 2015; Caminada *et al.* 2015a) for *ABA*.

Since then, a great effort has been made to identify paradigms where semi-stable and *L*-stable semantics are equivalent. The strategy employed by Alcântara *et al.* (2019) was to look for more expressive argumentation frameworks than *AAF*s: Attacking Dialectical Frameworks, a support-free fragment of Abstract Dialectical Frameworks (Brewka and Woltran 2010; Brewka *et al.* 2013), a generalization of *AAF*s designed to express arbitrary relationships among arguments. A translation from *NLP* to ADF^+ was proved by Alcântara *et al.* (2019) to account for various equivalences between their semantics, including the definition of a semantics for ADF^+ corresponding to the *L*-stable semantics for *NLP*s.

In a similar vein, other relevant proposals explored the equivalence between L-stable and semi-stable semantics for Claim-augmented Argumentation Frameworks (CAFs) (Dvořák *et al.* 2023; Rapberger 2020; Rocha and Cozman 2022b), which are a generalization of AAFs where each argument is explicitly associated with a claim, and for Bipolar Argumentation Frameworks (BAFs) with conclusions (Rocha and Cozman 2022a), a generalization of CAFs with the inclusion of an explicit notion of support between arguments. In both frameworks, the equivalence with NLPs does not just involve their semantics; it is also structural as there is a one-to-one mapping from them to NLPs.

Instead of looking for more expressive argumentation frameworks, the idea proposed by Sá and Alcântara (2021a) was to introduce more fine-grained semantics to deal with AAFs. Then a five-valued setting was employed rather than the usual three-valued one. As in the previous cases, this approach also captures the correspondence between the semantics for AAFs and NLPs. Specifically, it captures the correspondence involving L-stable semantics.

The connections between ABA and logic programming were later revisited by Sá and Alcântara (2019, 2021b), where they proposed a new translation from ABA frameworks to NLPs. The correspondence between their semantics (including L-stable) is obtained by selecting specific atoms in the characterization of the NLP semantics.

In summary, in the connections between NLP and argumentation semantics, the Achilles' heel is the relationship between L-stable and semi-stable semantics.

In this paper, we focus on the relationship between logic programming and $SETAF$ (Nielsen and Parsons 2006), an extension of Dung's AAFs to allow joint attacks on arguments. Following the strategy adopted by Caminada *et al.* (2015b) and Alcântara *et al.* (2019), we resort to the characterization of the $SETAF$ semantics in terms of labelings (Flouris and Bikakis 2019). As a starting point, we provide a mapping from NLPs to $SETAF$s (and vice versa) and show that NLPs and $SETAF$s are pairwise equivalent under various semantics, including the equivalence between L-stable and semi-stable. These results were inspired directly by two of our previous works: the equivalence between NLPs and ADF^+s (Alcântara *et al.* 2019), and the equivalence between ADF^+s and $SETAF$s (Alcântara and Sá 2021).

Furthermore, we investigate a class of NLPs called Redundancy-Free Atomic Logic Programs ($RFALP$s) (König *et al.* 2022). In $RFALP$s, the translations from NLPs to $SETAF$s and vice versa preserve the structure of each other's theories. In essence, these translations become inverses of each other. Consequently, the equivalence results concerning NLPs and $SETAF$s have deeper implications than the correspondence results between NLPs and AAFs: they encompass equivalence in both semantics as well as structure.

Some of these results are not new as recently they have already been obtained independently by König *et al.* (2022). In fact, their translation from NLPs to $SETAF$s and vice versa coincide with ours, and the structural equivalence between $RFALP$s and $SETAF$s has also been identified there. However, their focus differs from ours. While their work establishes the equivalence between stable models and stable extensions, it does not explore equivalences involving labeling-based semantics or address the controversy relating semi-stable semantics and L-stable semantics, which is a key motivation for this work. In comparison with König *et al.*'s work, the novelty of our proposal lies essentially in the aspects below:

- Our proofs of these results follow a significantly distinct path as they are based on properties of argument labelings and are deeply rooted in the works of Caminada *et al.* (2015b); Alcântara *et al.* (2019); Alcântara and Sá (2021).
- We prove the equivalence between partial stable, well-founded, regular, stable, and semi-stable model semantics for *NLP*s and, respectively, complete, grounded, preferred, stable, and semi-stable labelings for *SETAF*s. In particular, for the first time, an equivalence between *L*-stable model semantics for *NLP*s and semi-stable labelings for *SETAF*s is established.
- We provide a more in-depth analysis of the relationship between *NLP*s and *SETAF*s. Going beyond just proving semantic equivalence, we define functions that map labelings to interpretations and interpretations to labelings. These functions allow us to see interpretations and labelings as equivalent entities, further strengthening the connections between *NLP*s and *SETAF*s. In substance, we demonstrate that the equivalence also holds at the level of interpretations/labelings.

The strong connection we establish between interpretations and labelings opens doors for future exploration. This extends the applicability of our equivalence results to novel semantics beyond those investigated here, potentially even encompassing multivalued settings. This holds particular significance for the logic programming community. Well-established concepts from argumentation, such as argument strength (Beirlaen *et al.* 2018), can now be translated and investigated within the context of logic programming. This underscores the value of our decision to employ labelings instead of extensions as a more suitable approach to bridge the gap between *NLP*s and *SETAF*s.

Our research offers another key contribution, particularly relevant to the logic programming community: it explores the expressiveness of *RFALP*s. We demonstrate that a specific combination (denoted by $\mapsto_{UTPM}$) of program transformations can transform any *NLP* into an *RFALP* with exactly the same semantics. In simpler terms, *RFALP*s possess the same level of expressiveness as *NLP*s. Although each program transformation in $\mapsto_{UTPM}$ was proposed by Brass and Dix (1994, 1997, 1999), the combination of these program transformations (to our knowledge) has not been investigated yet. Then we establish several properties of $\mapsto_{UTPM}$. Among other original contributions of our work related to $\mapsto_{UTPM}$, we highlight the following results:

- Given an *NLP*, if repeatedly applying $\mapsto_{UTPM}$ leads to a program where no further transformations are applicable (irreducible program), then the resulting program is guaranteed to be an *RFALP*.
- We show that $\mapsto_{UTPM}$ is confluent; that is, given an *NLP*, it does not matter the order by which we apply repeatedly these program transformations; whenever we arrive at an irreducible program, they will always result in the same *RFALP* (and in the same corresponding *SETAF*). Hence, besides *NLP*s and *RFALP*s being equally expressive, each *NLP* is associated with a unique *RFALP*.
- The *SETAF* corresponding to an *NLP* is invariant with respect to $\mapsto_{UTPM}$; that is, if P_2 is obtained from P_1 via $\mapsto_{UTPM}$ (denoted by $P_1 \mapsto_{UTPM} P_2$), both P_1 and P_2 will be translated into the same *SETAF*.
- We show that $\mapsto_{UTPM}$ preserves the semantics for *NLP*s studied in this paper: if $P_1 \mapsto_{UTPM} P_2$, then P_1 and P_2 have the same partial stable models, well-founded models, regular models, stable models, and *L*-stable models.

The structure of the paper unfolds as follows: in Section 2, we establish the fundamental definitions related to *SETAF*s and *NLP*s. In Section 3, we adapt the procedure from Caminada *et al.* (2015b) and Alcântara *et al.* (2019) to translate *NLP*s into *SETAF*s, and subsequently, in the following section, we perform the reverse translation from *SETAF*s to *NLP*s. In both directions, we demonstrate that our labeling-based approach effectively preserves semantic correspondences, including the challenging case involving the equivalence between semi-stable semantics (on the *SETAF*s side) and L-stable semantics (on the *NLP*s side). In Section 5, we focus on *RFALP*s and reveal that, when restricted to them, the translation processes between *NLP*s and *SETAF*s are each other's inverse. Then, in Section 6, we guarantee that *RFALP*s are as expressive as *NLP*s. We conclude the paper with a discussion of our findings and outline potential avenues for future research endeavors.

The proofs for all novel results are provided in the Supplementary Material.

2 Preliminaries

2.1 SETAF

Nielsen and Parsons (2006) proposed an extension of Dung's (1995b) Abstract Argumentation Frameworks (AAFs) to allow joint attacks on arguments. The resulting framework, called $SETAF$, is defined next:

Definition 1 (SETAF (Nielsen and Parsons 2006)).
A framework with sets of attacking arguments ($SETAF$ for short) is a pair $\mathfrak{A} = (\mathcal{A}, Att)$, in which $\mathcal{A}$ is a finite set of arguments and $Att \subseteq (2^{\mathcal{A}} \setminus \{\emptyset\}) \times \mathcal{A}$ is an attack relation such that if $(\mathcal{B}, a) \in Att$, there is no $\mathcal{B}' \subset \mathcal{B}$ such that $(\mathcal{B}', a) \in Att$; that is, $\mathcal{B}$ is a minimal set (w.r.t. $\subseteq$) attacking a[1]. By $Att(a) = \{\mathcal{B} \subseteq \mathcal{A} \mid (\mathcal{B}, a) \in Att\}$, we mean the set of attackers of a.

In AAFs, only individual arguments can attack arguments. In $SETAF$s, the novelty is that sets of two or more arguments can also attack arguments. This means that $SETAF$s $(\mathcal{A}, Att)$ with $|\mathcal{B}| = 1$ for each $(\mathcal{B}, a) \in Att$ amount to (standard Dung) AAFs.

The semantics for $SETAF$s are generalizations of the corresponding semantics for AAFs (Nielsen and Parsons 2006) and can be defined equivalently in terms of extensions or labelings (Flouris and Bikakis 2019). Our focus here will be on their labeling-based semantics.

Definition 2 (Labelings (Flouris and Bikakis 2019)).
Let $\mathfrak{A} = (\mathcal{A}, Att)$ be a $SETAF$. A labeling is a function $\mathcal{L} : \mathcal{A} \to \{\texttt{in}, \texttt{out}, \texttt{undec}\}$. It is admissible iff for each $a \in \mathcal{A}$,

- If $\mathcal{L}(a) = \texttt{in}$, then for each $\mathcal{B} \in Att(a)$, it holds $\mathcal{L}(b) = \texttt{out}$ for some $b \in \mathcal{B}$.
- If $\mathcal{L}(a) = \texttt{out}$, then there exists $\mathcal{B} \in Att(a)$ such that $\mathcal{L}(b) = \texttt{in}$ for each $b \in \mathcal{B}$.

A labeling $\mathcal{L}$ is called complete iff it is admissible and for each $a \in \mathcal{A}$,

[1] In the original definition of $SETAF$s by Nielsen and Parsons (2006), attacks are not necessarily subset-minimal.

https://doi.org/10.1017/S1471068424000188 Published online by Cambridge University Press

- If $\mathcal{L}(a) = \texttt{undec}$, then there exists $\mathcal{B} \in Att(a)$ such that $\mathcal{L}(b) \neq \texttt{out}$ for each $b \in \mathcal{B}$, and for each $\mathcal{B} \in Att(a)$, it holds $\mathcal{L}(b) \neq \texttt{in}$ for some $b \in \mathcal{B}$.

We write $\texttt{in}(\mathcal{L})$ for $\{a \in \mathcal{A} \mid \mathcal{L}(a) = \texttt{in}\}$, $\texttt{out}(\mathcal{L})$ for $\{a \in \mathcal{A} \mid \mathcal{L}(a) = \texttt{out}\}$, and $\texttt{undec}(\mathcal{L})$ for $\{a \in \mathcal{A} \mid \mathcal{L}(a) = \texttt{undec}\}$. As a labeling essentially defines a partition among the arguments, we sometimes write $\mathcal{L}$ as a triple $(\texttt{in}(\mathcal{L}), \texttt{out}(\mathcal{L}), \texttt{undec}(\mathcal{L}))$. Intuitively, an argument labeled in represents explicit acceptance; an argument labeled out indicates rejection; and one labeled undec is undecided; that is, it is neither accepted nor rejected. We can now describe the *SETAF* semantics studied in this paper:

Definition 3 (Semantics (Flouris and Bikakis 2019)).
Let $\mathfrak{A} = (\mathcal{A}, Att)$ be a *SETAF*. A complete labeling $\mathcal{L}$ is called

- grounded iff $\texttt{in}(\mathcal{L})$ is minimal (w.r.t. $\subseteq$) among all complete labelings of $\mathfrak{A}$.
- preferred iff $\texttt{in}(\mathcal{L})$ is maximal (w.r.t. $\subseteq$) among all complete labelings of $\mathfrak{A}$.
- stable iff $\texttt{undec}(\mathcal{L}) = \emptyset$.
- semi-stable iff $\texttt{undec}(\mathcal{L})$ is minimal (w.r.t. $\subseteq$) among all complete labelings of $\mathfrak{A}$.

Let us consider the following example:

Example 1.
Consider the *SETAF* $\mathfrak{A} = (\mathcal{A}, Att)$ below:

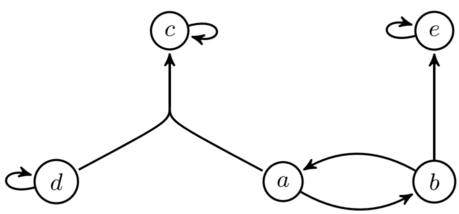


Fig. 1. A *SETAF* $\mathfrak{A}$.

Concerning the semantics of $\mathfrak{A}$, we have

- Complete labelings: $\mathcal{L}_1 = (\emptyset, \emptyset, \{a, b, c, d, e\})$, $\mathcal{L}_2 = (\{a\}, \{b\}, \{c, d, e\})$ and $\mathcal{L}_3 = (\{b\}, \{a, e\}, \{c, d\})$;
- Grounded labelings: $\mathcal{L}_1 = (\emptyset, \emptyset, \{a, b, c, d, e\})$;
- Preferred labelings: $\mathcal{L}_2 = (\{a\}, \{b\}, \{c, d, e\})$ and $\mathcal{L}_3 = (\{b\}, \{a, e\}, \{c, d\})$;
- Stable labelings: none;
- Semi-stable labelings: $\mathcal{L}_3 = (\{b\}, \{a, e\}, \{c, d\})$.

2.2 Logic programs and semantics

Now, we take a look at propositional Normal Logic Programs. To delve into their definition and semantics, we will follow the presentation outlined by Caminada *et al.* (2015b), which draws from the foundation laid out by Przymusinski (1990).

https://doi.org/10.1017/S1471068424000188 Published online by Cambridge University Press

Definition 4 (Caminada et al. (2015b).)
A rule r is an expression

$$r : c \leftarrow a_1, \ldots, a_m, \texttt{not } b_1, \ldots, \texttt{not } b_n \tag{1}$$

where $(m, n \geq 0)$; c, each a_i $(1 \leq i \leq m)$ and each b_j $(1 \leq j \leq n)$ are atoms, and `not` represents negation as failure. A literal is either an atom a (positive literal) or a negated atom $\texttt{not } a$ (negative literal). Given a rule r as above, c is called the head of r, which we denote as $head(r)$, and $body(r) = \{a_1, \ldots, a_m, \texttt{not } b_1, \ldots, \texttt{not } b_n\}$ is called the body of r. Further, we divide $body(r)$ into two sets $body^+(r) = \{a_1, \ldots, a_m\}$ and $body^-(r) = \{\texttt{not } b_1, \ldots, \texttt{not } b_n\}$. A fact is a rule where $m = n = 0$. A Normal Logic Program (*NLP*) or simply a program P is a finite set of rules. If every $r \in P$ has $body^-(r) = \emptyset$, P is a positive program. The Herbrand Base of P is the set HB_P of all atoms appearing in P.

A wide range of *NLP* semantics are based on the three-valued interpretations of programs (Przymusinski 1990):

Definition 5 (Three-Valued Herbrand Interpretation (Przymusinski 1990)).
A three-valued Herbrand Interpretation $\mathcal{I}$ (or simply interpretation) of an NLP P is a pair $\langle T, F\rangle$ with $T, F \subseteq HB_P$ and $T \cap F = \emptyset$. The atoms in T are true in $\mathcal{I}$, the atoms in F are false in $\mathcal{I}$, and the atoms in $HB_P \setminus (T \cup F)$ are undefined in $\mathcal{I}$. For convenience, when the NLP P is clear from the context, we will refer to the set of undefined atoms in $HB_P \setminus (T \cup F)$ simply as $\overline{T \cup F}$.

Now we will consider the main semantics for *NLP*s. Let $\mathcal{I} = \langle T, F\rangle$ be a three-valued Herbrand interpretation of an *NLP* P; the reduct of P with respect to $\mathcal{I}$ (written as $P/\mathcal{I}$) is the *NLP* constructed using the following steps:

1. Remove any $a \leftarrow a_1, \ldots, a_m,\ \texttt{not } b_1, \ldots, \texttt{not } b_n \in P$ such that $b_j \in T$ for some j $(1 \leq j \leq n)$;
2. Afterward, remove any occurrence of $\texttt{not } b_j$ from P such that $b_j \in F$;
3. Then, replace any occurrence of $\texttt{not } b_j$ left by a special atom $\mathbf{u}$ $(\mathbf{u} \notin HB_P)$.

In the above procedure, $\mathbf{u}$ is assumed to be an atom not in HB_P which is undefined in all interpretations of P (a constant). Note that $P/\mathcal{I}$ is a positive program since all negative literals have been removed. As a consequence, $P/\mathcal{I}$ has a unique least three-valued model (Przymusinski 1990), obtained by the Ψ operator:

Definition 6 (Ψ Operator (Przymusinski 1990)).
Let P be a positive program and $\mathcal{J} = \langle T, F\rangle$ be an interpretation. Define $\Psi_P(\mathcal{J}) = \langle T', F'\rangle$, where

- $c \in T'$ iff $c \in HB_P$ and there exists $c \leftarrow a_1, \ldots, a_m \in P$ such that for all i, $1 \leq i \leq m$, $a_i \in T$;
- $c \in F'$ iff $c \in HB_P$ and for every $c \leftarrow a_1, \ldots, a_m \in P$, there exists i, $1 \leq i \leq m$, such that $a_i \in F$.

The least three-valued model of P is given by $\Psi_P^{\uparrow \omega}$ (Przymusinski 1990), the least fixed point of Ψ_P iteratively obtained as follows:

https://doi.org/10.1017/S1471068424000188 Published online by Cambridge University Press

$$\Psi_P^{\uparrow 0} = \langle \emptyset, HB_P \rangle$$

$$\Psi_P^{\uparrow i+1} = \Psi_P(\Psi_P^{\uparrow i})$$

$$\Psi_P^{\uparrow \omega} = \left\langle \bigcup_{i<\omega} \left\{ T_i \mid \Psi_P^{\uparrow i} = \langle T_i, F_i \rangle \right\}, \bigcap_{i<\omega} \left\{ F_i \mid \Psi_P^{\uparrow i} = \langle T_i, F_i \rangle \right\} \right\rangle$$

where ω denotes the first infinite ordinal.

We can now describe the logic programming semantics studied in this paper:

Definition 7.
Let P be an NLP and $\mathcal{I} = \langle T, F \rangle$ be an interpretation; by $\Omega_P(\mathcal{I}) = \Psi_{\frac{P}{\mathcal{I}}}^{\uparrow \omega}$, we mean the least three-valued model of $\frac{P}{\mathcal{I}}$. We say that

- $\mathcal{I}$ is a partial stable model of P iff $\Omega_P(\mathcal{I}) = \mathcal{I}$ (Przymusinski 1990).
- $\mathcal{I}$ is a well-founded model of P iff $\mathcal{I}$ is a partial stable model of P where there is no partial stable model $\mathcal{I}' = \langle T', F' \rangle$ of P such that $T' \subset T$; that is, T is minimal (w.r.t. set inclusion) among all partial stable models of P (Przymusinski 1990).
- $\mathcal{I}$ is a regular model of P iff $\mathcal{I}$ is a partial stable model of P where there is no partial stable model $\mathcal{I}' = \langle T', F' \rangle$ of P such that $T \subset T'$; that is, T is maximal (w.r.t. set inclusion) among all partial stable models of P (Eiter et al. 1997).
- $\mathcal{I}$ is a (two-valued) stable model of P iff $\mathcal{I}$ is a partial stable model of P where $T \cup F = HB_P$ (Przymusinski 1990).
- $\mathcal{I}$ is an L-stable model of P iff $\mathcal{I}$ is a partial stable model of P where there is no partial stable model $\mathcal{I}' = \langle T', F' \rangle$ of P such that $T \cup F \subset T' \cup F'$; that is, $T \cup F$ is maximal (w.r.t. set inclusion) among all partial stable models of P (Eiter et al. 1997).

Although some of these definitions are not standard in logic programming literature, their equivalence was proved by Caminada *et al.* (2015b). This format helps us to relate *NLP* and *SETAF* semantics due to the structural similarities between Definition 7 and Definitions 2 and 3. We illustrate these semantics in the following example:

Example 2.
Consider the following logic program P:

$$r_1 : a \leftarrow \texttt{not}\, b \quad r_2 : b \leftarrow \texttt{not}\, a \quad r_3 : c \leftarrow \texttt{not}\, a, \texttt{not}\, c$$

$$r_4 : c \leftarrow \texttt{not}\, c, \texttt{not}\, d \quad r_5 : d \leftarrow \texttt{not}\, d \quad r_6 : e \leftarrow \texttt{not}\, b, \texttt{not}\, e$$

This program has

- Partial Stable Models: $\mathcal{M}_1 = \langle \emptyset, \emptyset \rangle$, $\mathcal{M}_2 = \langle \{a\}, \{b\} \rangle$ and $\mathcal{M}_3 = \langle \{b\}, \{a, e\} \rangle$;
- Well-founded model: $\mathcal{M}_1 = \langle \emptyset, \emptyset \rangle$;
- Regular models: $\mathcal{M}_2 = \langle \{a\}, \{b\} \rangle$ and $\mathcal{M}_3 = \langle \{b\}, \{a, e\} \rangle$;
- Stable models: none;
- L-stable model: $\mathcal{M}_3 = \langle \{b\}, \{a, e\} \rangle$.

https://doi.org/10.1017/S1471068424000188 Published online by Cambridge University Press

3. From *NLP* to *SETAF*

In this section, we revisit the three-step process of argumentation framework instantiation as employed by Caminada *et al.* (2015b) for translating an *NLP* into an *AAF*. This method is based on the approach introduced by Wu *et al.* (2009) and shares similarities with the procedures used in ASPIC (Caminada and Amgoud 2005, 2007) and logic-based argumentation (Gorogiannis and Hunter 2011). Its first step involves taking an *NLP* and constructing its associated *AAF*. Then, we apply *AAF* semantics in the second step, followed by an analysis of the implications of these semantics at the level of conclusions (step 3). In our case, starting with an *NLP* P, we derive the associated *SETAF* $(\mathcal{A}_P, Att_P)$. Unlike the construction described by Caminada *et al.* (2015b), rules with identical conclusions in P will result in a single argument in $\mathcal{A}_P$. This distinction is capital for establishing the equivalence results between *NLP*s and *SETAF*s. Additionally, it simplifies steps 2 and 3, making them more straightforward to follow. We now detail this process.

3.1 SETAF construction

We will devise one translation from *NLP* to *SETAF* that is sufficiently robust to guarantee the equivalence between various kinds of *NLP*s models and *SETAF*s labelings. Specifically, our approach will establish the correspondence between partial stable models and complete labelings, well-founded models and grounded labelings, regular models and preferred labelings, stable models and stable labelings, L-stable models and semi-stable labelings. Our method is built upon a translation from *NLP* to *AAF* proposed by Caminada *et al.* (2015b), where *NLP* rules are directly translated into arguments. We will adapt this approach for *SETAF* by employing the translation method outlined by Caminada *et al.* (2015b) to construct statements, and then statements corresponding to rules with the same head will be grouped to form a single argument. Taking an *NLP* P, we can start to construct statements recursively as follows:

Definition 8 (Statements and Arguments).
Let P be an NLP.

- If $c \leftarrow \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n$ is a rule in P, then it is also a statement (say s) with
  - $\texttt{Conc}(s) = c$,
  - $\texttt{Rules}(s) = \{c \leftarrow \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n\}$,
  - $\texttt{Vul}(s) = \{b_1, \ldots, b_n\}$, and
  - $\texttt{Sub}(s) = \{s\}$.
- If $c \leftarrow a_1, \ldots, a_m, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n$ is a rule in P and for each a_i $(1 \leq i \leq m)$ there exists a statement s_i with $\texttt{Conc}(s_i) = a_i$ and $c \leftarrow a_1, \ldots, a_m, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n$ is not contained in $\texttt{Rules}(s_i)$, then $c \leftarrow (s_1), \ldots, (s_m)$, $\texttt{not}\ b_1, \ldots, \texttt{not}\ b_n$ is a statement (say s) with

https://doi.org/10.1017/S1471068424000188 Published online by Cambridge University Press

— $\texttt{Conc}(s) = c$,
— $\texttt{Rules}(s) = \texttt{Rules}(s_1) \cup \ldots \cup \texttt{Rules}(s_m) \cup \{c \leftarrow a_1, \ldots, a_m, \texttt{not } b_1, \ldots, \texttt{not } b_n\}$
— $\texttt{Vul}(s) = \texttt{Vul}(s_1) \cup \ldots \cup \texttt{Vul}(s_m) \cup \{b_1, \ldots, b_n\}$, and
— $\texttt{Sub}(s) = \{s\} \cup \texttt{Sub}(s_1) \cup \ldots \cup \texttt{Sub}(s_m)$.

By $\mathfrak{S}_P$, we mean the set of all statements we can construct from P as above. Then we define $\mathcal{A}_P = \{\texttt{Conc}(s) \mid s \in \mathfrak{S}_P\}$ as the set of all arguments we can construct from P. For an argument c from P ($c \in \mathcal{A}_P$), we have that

- $\texttt{Conc}(c) = c$, and
- $\texttt{Vul}_P(c) = \{\texttt{Vul}(s) \mid s \in \mathfrak{S}_P \textit{ and } \texttt{Conc}(s) = c\}$.

If c is an argument, then $\texttt{Conc}(c)$ is referred to as the *conclusion* of c, and $\texttt{Vul}_P(c)$ is referred to as the *vulnerabilities* of c in P. When the context is clear, we will write simply $\texttt{Vul}(c)$ instead of $\texttt{Vul}_P(c)$.

Now we will clarify the connection between the existence of statements and the existence of a derivation in a reduct.

Lemma 1.
Let P be an NLP, $\mathcal{I} = \langle T, F\rangle$ an interpretation and $\Omega_P(\mathcal{I}) = \langle T', F'\rangle$ the least three-valued model of $\frac{P}{\mathcal{I}}$. It holds

(i) $c \in T'$ iff there exists a statement s constructed from P such that $\texttt{Conc}(s) = c$ and $\texttt{Vul}(s) \subseteq F$.
(ii) $c \in F'$ iff for every statement s constructed from P such that $\texttt{Conc}(s) = c$, we have $\texttt{Vul}(s) \cap T \neq \emptyset$

We can prove both results in Lemma 1 by induction. Assuming that $\Psi^{\uparrow i}_{\frac{P}{\mathcal{I}}} = \langle T_i, F_i\rangle$ for each $i \in \mathbb{N}$, we can prove the right-hand side of item 1 and the left-hand side of item 2 by induction on the value of i after guaranteeing the following results:

- If $c \in T_i$, then there exists a statement s constructed from P such that $\texttt{Conc}(s) = c$ and $\texttt{Vul}(s) \subseteq F$.
- If $c \notin F_i$, then there exists a statement s constructed from P such that $\texttt{Conc}(s) = c$ and $\texttt{Vul}(s) \cap T = \emptyset$.

The remaining cases of Lemma 1 can be proved by structural induction on the construction of a statement s (see a detailed account of the proof of Lemma 1 in Supplementary Material).

Lemma 1 ensures that statements are closely related to derivations in a reduct. An atom c is true in the least three-valued model of $\frac{P}{\mathcal{I}}$ iff we can construct a statement with conclusion c and whose vulnerabilities are false according to $\mathcal{I}$; otherwise, c is false in the least three-valued model of $\frac{P}{\mathcal{I}}$ iff for every statement whose conclusion is c, at least one of its vulnerabilities is true in $\mathcal{I}$. The next result is a direct consequence of Lemma 1:

Corollary 2.
Let P be an NLP.

https://doi.org/10.1017/S1471068424000188 Published online by Cambridge University Press

- *Assume* $\mathcal{I} = \langle \emptyset, HB_P \rangle$ *and* $\Omega_P(\mathcal{I}) = \langle T', F' \rangle$. *It holds that* $c \in T'$ *iff there exists a statement* s *constructed from* P *such that* $\texttt{Conc}(s) = c$.
- *There is no statement* s *constructed from* P *such that* $\texttt{Conc}(s) = c$ *iff* $c \in F'$ *for every interpretation* $\mathcal{I}$ *with* $\Omega_P(\mathcal{I}) = \langle T', F' \rangle$.

The reduct of P with respect to $\langle \emptyset, HB_P \rangle$ gives us all the possible derivations of P, and from these derivations, we can construct all the statements associated with P. On the other hand, the atoms that are lost in the translation, that is, the atoms not associated with statements, are simply those that are false in the least three-valued model of every possible reduct of P. Besides establishing connections between statements and derivations in a reduct, Lemma 1 also plays a central role in the proof of Theorems 4 and 5.

Apart from that, intuitively, we can see a statement as a tree-like structure representing a possible derivation of an atom from the rules of a program. In contrast, an argument for c in P is associated with the (derivable) atom c itself and can be obtained by collecting all the statements with the same conclusion c (i.e., all the possible ways of deriving c in P).

Example 3.
Consider the *NLP* P below with rules $\{r_1, \ldots, r_8\}$:

$$r_1 : a \qquad r_2 : b \leftarrow a \qquad r_3 : c \leftarrow \texttt{not}\ c$$
$$r_4 : d \leftarrow b, \texttt{not}\ a, \texttt{not}\ d \qquad r_5 : d \leftarrow \texttt{not}\ c, \texttt{not}\ d \qquad r_6 : e \leftarrow b, c, \texttt{not}\ e$$
$$r_7 : c \leftarrow f, \texttt{not}\ g \qquad r_8 : f \leftarrow c, g$$

According to Definition 8, we can construct the following statements from P:

$$s_1 : a \qquad s_2 : b \leftarrow (s_1) \qquad s_3 : c \leftarrow \texttt{not}\ c$$
$$s_4 : d \leftarrow (s_2), \texttt{not}\ a, \texttt{not}\ d \qquad s_5 : d \leftarrow \texttt{not}\ c, \texttt{not}\ d \qquad s_6 : e \leftarrow (s_2), (s_3), \texttt{not}\ e$$

In the next table, we give the conclusions and vulnerabilities of each statement:

	s_1	s_2	s_3	s_4	s_5	s_6
$\texttt{Conc}(.)$	a	b	c	d	d	e
$\texttt{Vul}(.)$	$\emptyset$	$\emptyset$	$\{c\}$	$\{a, d\}$	$\{c, d\}$	$\{c, e\}$

Alternatively, we can depict statements as possible derivations as in Figure 2:

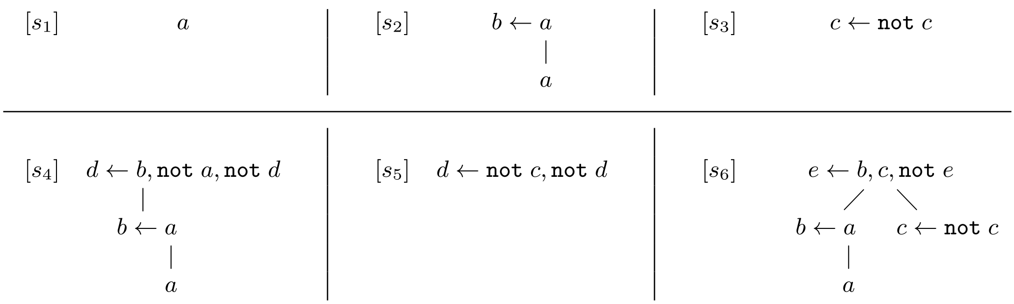


Fig. 2. Statements constructed from P.

https://doi.org/10.1017/S1471068424000188 Published online by Cambridge University Press

The vulnerabilities of a statement s are associated with the negative literals found in the derivation of s. If $\texttt{not}\, a$ is one of them, we know that a is one of its vulnerabilities. This means that if a is derived, then $\texttt{Conc}(s)$ cannot be obtained via this derivation represented by s. However, it can still be obtained via other derivations/statements. For instance, in the program P of Example 3, the derivation of a suffices to prevent the derivation of d via statement s_4 (for that reason, $a \in \texttt{Vul}(s_4)$), but we still can derive d via s_5. Notice also that there are no statements with conclusions f and g. From Corollary 2, we know that it is not possible to derive them in P as they are false in the least three-valued model of each reduct of P. In addition, to determine the vulnerabilities of an atom (and not only of a specific derivation leading to this atom), we collect these data about the statements with the same conclusions to give the conclusions and vulnerabilities of each argument. In our example, we obtain the following results:

	a	b	c	d	e
$\texttt{Conc}(.)$	a	b	c	d	e
$\texttt{Vul}(.)$	$\{\emptyset\}$	$\{\emptyset\}$	$\{c\}$	$\{\{a,d\},\{c,d\}\}$	$\{\{c,e\}\}$

As the vulnerabilities of an atom/argument a are a collection of the vulnerabilities of the statements whose conclusion is a, any set containing at least one atom in each of these statements suffices to prevent the derivation of a in P. In our example, there are two statements with the same conclusion d and $\texttt{Vul}(d) = \{\{a,d\},\{c,d\}\}$. Thus any set of atoms containing $\{d\}$ or $\{a,c\}$ prevents the conclusion of d in P. We will resort to these minimal sets to determine the attack relation:

Definition 9.
Let P be an NLP and let $\mathcal{B}$ and a be, respectively, a set of arguments and an argument in the sense of Definition 8. We say that $(\mathcal{B}, a) \in Att_P$ iff $\mathcal{B}$ is a minimal set (w.r.t. set inclusion) such that for each $V \in \texttt{Vul}_P(a)$, there exists $b \in \mathcal{B} \cap V$.

For the arguments of Example 3, it holds that both a and b are not attacked, c attacks itself, c attacks e, e attacks itself, d attacks itself, a and c (collectively) attack d. This strategy of extracting statements from *NLP*s rules and then gathering those with identical conclusions into arguments is not novel; Alcântara *et al.* (2019) proposed a translation from *NLP*s into Abstract Dialectical Frameworks (Brewka and Woltran 2010; Brewka *et al.* 2013) by following a similar path. Using the thus-defined notions of arguments and attacks, we define the *SETAF* associated with an *NLP*.

Definition 10.
Let P be an NLP. We define its associated $SETAF$ as $\mathfrak{A}_P = (\mathcal{A}_P, Att_P)$, where $\mathcal{A}_P$ is the set of arguments in the sense of Definition 8 and Att_P is the attack relation in the sense of Definition 9.

As an example, the *SETAF* $\mathfrak{A}_P = (\mathcal{A}_P, Att_P)$ associated with the *NLP* of Example 3 is depicted in Figure 3.

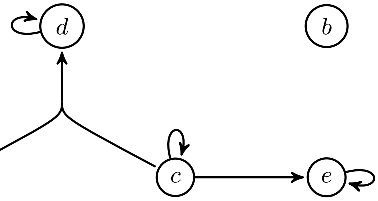


Fig. 3. A *SETAF* $\mathfrak{A}_P = (\mathcal{A}_P, Att_P)$.

3.2 Equivalence results

Once the *SETAF* has been constructed, we show the equivalence between the semantics for an *NLP* P and their counterpart for the associated *SETAF* $\mathfrak{A}_P$. One distinguishing characteristic of our approach in comparison with König *et al.*'s (2022) proposal is that it is more organic. We prove the equivalence results by identifying connections between fundamental notions used in the definition of the semantics for *NLP*s and *SETAF*s. With this purpose, we introduce two functions: $\mathcal{L}2\mathcal{I}_P$ associates an interpretation to each labeling while $\mathcal{I}2\mathcal{L}_P$ associates a labeling to each interpretation. We then investigate the conditions under which they are each other's inverse and employ these results to prove the equivalence between the semantics. These functions essentially permit us to treat interpretations and labelings interchangeably.

Definition 11 ($\mathcal{L}2\mathcal{I}_P$ and $\mathcal{I}2\mathcal{L}_P$ Functions).
Let P be an NLP, $\mathfrak{A}_P = (\mathcal{A}_P, Att_P)$ be its associated $SETAF$, $\mathcal{I}nt$ be the set of all the three-valued interpretations of P and $\mathcal{L}ab$ be the set of all labelings of $\mathfrak{A}_p$. We introduce a function $\mathcal{L}2\mathcal{I}_P : \mathcal{L}ab \to \mathcal{I}nt$ such that $\mathcal{L}2\mathcal{I}_P(\mathcal{L}) = \langle T, F \rangle$, in which

- $T = \{c \in HB_P \mid c \in \mathcal{A}_P \text{ and } \mathcal{L}(c) = \texttt{in}\}$;
- $F = \{c \in HB_P \mid c \notin \mathcal{A}_P \text{ or } c \in \mathcal{A}_P \text{ and } \mathcal{L}(c) = \texttt{out}\}$;
- $\overline{T \cup F} = \{c \in HB_P \mid c \in \mathcal{A}_P \text{ and } \mathcal{L}(c) = \texttt{undec}\}$.

We introduce a function $\mathcal{I}2\mathcal{L}_P : \mathcal{I}nt \to \mathcal{L}ab$ such that for any $\mathcal{I} = \langle T, F \rangle \in \mathcal{I}nt$ and any $c \in \mathcal{A}_P$,

- $\mathcal{I}2\mathcal{L}_P(\mathcal{I})(c) = \texttt{in}$ if $c \in T$;
- $\mathcal{I}2\mathcal{L}_P(\mathcal{I})(c) = \texttt{out}$ if $c \in F$;
- $\mathcal{I}2\mathcal{L}_P(\mathcal{I})(c) = \texttt{undec}$ if $c \notin T \cup F$.

$\mathcal{I}2\mathcal{L}_P(\mathcal{I})(c)$ is not defined if $c \notin \mathcal{A}_P$.

The correspondence between labelings and interpretations is clear for those atoms $c \in HB_P$ in which $c \in \mathcal{A}_P$. In this case, we have that c is interpreted as true iff c is labeled as `in`; c is interpreted as false iff c is labeled as `out`. In contradistinction, those atoms $c \in HB_P$ not associated with arguments ($c \notin \mathcal{A}_P$) are simply interpreted as false. This will suffice to guarantee our results; next theorem assures us that $\mathcal{I}2\mathcal{L}_P(\mathcal{L}2\mathcal{I}_P(\mathcal{L})) = \mathcal{L}$:

https://doi.org/10.1017/S1471068424000188 Published online by Cambridge University Press

Theorem 3.
Let P be an NLP and $\mathfrak{A}_P = (\mathcal{A}_P, Att_P)$ be the associated SETAF. For any labeling $\mathcal{L}$ of $\mathfrak{A}_P$, it holds $\mathcal{I}2\mathcal{L}_P(\mathcal{L}2\mathcal{I}_P(\mathcal{L})) = \mathcal{L}$.

In general, $\mathcal{L}2\mathcal{I}_P(\mathcal{I}2\mathcal{L}_P(\mathcal{I}))$ is not equal to $\mathcal{I}$, because of those atoms c occurring in an *NLP* P, but not in $\mathcal{A}_P$. However, when $\mathcal{M}$ is a partial stable model, $\mathcal{L}2\mathcal{I}_P(\mathcal{I}2\mathcal{L}_P(\mathcal{M})) = \mathcal{M}$:

Theorem 4.
Let P be an NLP, $\mathfrak{A}_P = (\mathcal{A}_P, Att_P)$ be the associated SETAF and $\mathcal{M} = \langle T, F\rangle$ be a partial stable model of P. It holds that $\mathcal{L}2\mathcal{I}_P(\mathcal{I}2\mathcal{L}_P(\mathcal{M})) = \mathcal{M}$.

This means that when restricted to partial stable models and complete labelings, $\mathcal{L}2\mathcal{I}_P$ and $\mathcal{I}2\mathcal{L}_P$ are each other's inverse. From Lemma 1, and Theorems 3 and 4, we can obtain the following result:

Theorem 5.
Let P be an NLP and $\mathfrak{A}_P = (\mathcal{A}_P, Att_P)$ be the associated SETAF. It holds

- *$\mathcal{L}$ is a complete labeling of $\mathfrak{A}_P$ iff $\mathcal{L}2\mathcal{I}_P(\mathcal{L})$ is a partial stable model of P.*
- *$\mathcal{M}$ is a partial stable model of P iff $\mathcal{I}2\mathcal{L}_P(\mathcal{M})$ is a complete labeling of $\mathfrak{A}_P$.*

Theorem 5 is one of the main results of this paper. It plays a central role in ensuring the equivalence between the semantics for *NLP* and their counterpart for *SETAF*:

Theorem 6.
Let P be an NLP and $\mathfrak{A}_P = (\mathcal{A}_P, Att_P)$ be the associated SETAF. It holds

1. *$\mathcal{L}$ is a grounded labeling of $\mathfrak{A}_P$ iff $\mathcal{L}2\mathcal{I}_P(\mathcal{L})$ is a well-founded model of P.*
2. *$\mathcal{L}$ is a preferred labeling of $\mathfrak{A}_P$ iff $\mathcal{L}2\mathcal{I}_P(\mathcal{L})$ is a regular model of P.*
3. *$\mathcal{L}$ is a stable labeling of $\mathfrak{A}_P$ iff $\mathcal{L}2\mathcal{I}_P(\mathcal{L})$ is a stable model of P.*
4. *$\mathcal{L}$ is a semi-stable labeling of $\mathfrak{A}_P$ iff $\mathcal{L}2\mathcal{I}_P(\mathcal{L})$ is an L-stable model of P.*

The following result is a direct consequence of Theorems 4 and 6:

Corollary 7.
Let P be an NLP and $\mathfrak{A}_P = (\mathcal{A}_P, Att_P)$ be the associated SETAF. It holds

1. *$\mathcal{M}$ is a well-founded model of P iff $\mathcal{I}2\mathcal{L}_P(\mathcal{M})$ is a grounded labeling of $\mathfrak{A}_P$.*
2. *$\mathcal{M}$ is a regular model of P iff $\mathcal{I}2\mathcal{L}_P(\mathcal{M})$ is a preferred labeling of $\mathfrak{A}_P$.*
3. *$\mathcal{M}$ is a stable model of P iff $\mathcal{I}2\mathcal{L}_P(\mathcal{M})$ is a stable labeling of $\mathfrak{A}_P$.*
4. *$\mathcal{M}$ is an L-stable model of P iff $\mathcal{I}2\mathcal{L}_P(\mathcal{M})$ is a semi-stable labeling of $\mathfrak{A}_P$.*

Next, we consider the *NLP* exploited by Caminada *et al.* (2015b) as a counterexample to show that in general, L-stable models and semi-stable labelings do not coincide with each other in their translation from *NLP*s to *AAF*s:

Example 4.
Let P be the *NLP* and $\mathfrak{A}_P$ be the associated *SETAF* depicted in Figure 4:

(a)

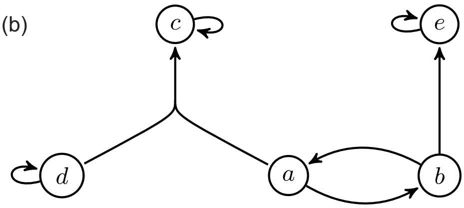
$d \leftarrow \texttt{not}\ d$ $\quad$ $c \leftarrow \texttt{not}\ c, \texttt{not}\ d$

$a \leftarrow \texttt{not}\ b$ $\quad$ $b \leftarrow \texttt{not}\ a$

$c \leftarrow \texttt{not}\ c, \texttt{not}\ a$ $\quad$ $e \leftarrow \texttt{not}\ e, \texttt{not}\ b$

P

(b)



$\mathfrak{A}_P$

Fig. 4. *NLP* P and its associated *SETAF* $\mathfrak{A}_P$.

As expected from Theorems 5 and 6, we obtain in Table 1 the equivalence between partial stable models and complete labelings, well-founded models and grounded labelings, regular models and preferred labelings, stable models and stable labelings, L-stable models and semi-stable labelings. We emphasize the coincidence between L-stable models and semi-stable labelings in Table 1 as it does not occur in the approach of Caminada *et al.* (2015b). In their work, the associated *AAF* possesses two semi-stable labelings in contrast with the unique L-stable model $\mathcal{M}_3$ of P. In the next two sections, we will show that this relation between *NLP*s and *SETAF*s has even deeper implications.

4 From *SETAF* to *NLP*

Now we will provide a translation in the other direction, that is, from *SETAF*s to *NLP*s. As in the previous section, this translation guarantees the equivalence between the semantics for *NLP*s and their counterpart for *SETAF*s.

Definition 12.
Let $\mathfrak{A} = (\mathcal{A}, Att)$ be a *SETAF*. For any argument $a \in \mathcal{A}$, we will assume $\mathcal{V}_a = \{V \mid V$ *is a minimal set (w.r.t. set inclusion) such that for each* $\mathcal{B} \in Att(a)$, *there exists* $b \in \mathcal{B} \cap V\}$. We define the associated *NLP* $P_{\mathfrak{A}}$ as follows:

$$P_{\mathfrak{A}} = \{a \leftarrow \texttt{not}\ b_1, \ldots \texttt{not}\ b_n \mid a \in \mathcal{A}, V \in \mathcal{V}_a \ \textit{and}\ V = \{b_1, \ldots, b_n\}\}.$$

Example 5.
Recall the *SETAF* $\mathfrak{A}$ of Example 1 (it is the same as that in Figure 4b). The associated *NLP* $P_{\mathfrak{A}}$ is

$$\begin{array}{ll} d \leftarrow \texttt{not}\ d & c \leftarrow \texttt{not}\ c, \texttt{not}\ d \\ a \leftarrow \texttt{not}\ b & b \leftarrow \texttt{not}\ a \\ c \leftarrow \texttt{not}\ c, \texttt{not}\ a & e \leftarrow \texttt{not}\ e, \texttt{not}\ b \end{array}$$

Notice that $P_{\mathfrak{A}}$ and the *NLP* P of Example 4 are the same. As it will be clear in the next section, this is not merely a coincidence. Besides, from Definition 12, it is clear that $HB_{P_{\mathfrak{A}}} = \mathcal{A}$. Consequently, when considering a *SETAF* $\mathfrak{A}$ and its associated *NLP* $P_{\mathfrak{A}}$, the definition of the function $\mathcal{L}2\mathcal{I}_{\mathfrak{A}}$ (resp., $\mathcal{I}2\mathcal{L}_{\mathfrak{A}}$), which associates labelings with interpretations (resp., interpretations with labelings), will be simpler than the definition of $\mathcal{L}2\mathcal{I}_P$ (resp., $\mathcal{I}2\mathcal{L}_P$) presented in the previous section.

https://doi.org/10.1017/S1471068424000188 Published online by Cambridge University Press

Table 1. *Semantics for P and $\mathfrak{A}_P$*

Partial stable models	$\mathcal{M}_1 = \langle \emptyset, \emptyset \rangle$ $\mathcal{M}_2 = \langle \{a\}, \{b\} \rangle$ $\mathcal{M}_3 = \langle \{b\}, \{a, e\} \rangle$	Complete labelings	$\mathcal{L}_1 = (\emptyset, \emptyset, \{a, b, c, d, e\})$ $\mathcal{L}_2 = (\{a\}, \{b\}, \{c, d, e\})$ $\mathcal{L}_3 = (\{b\}, \{a, e\}, \{c, d\})$
Well-founded models	$\mathcal{M}_1 = \langle \emptyset, \emptyset \rangle$	Grounded labelings	$\mathcal{L}_1 = (\emptyset, \emptyset, \{a, b, c, d, e\})$
Regular models	$\mathcal{M}_2 = \langle \{a\}, \{b\} \rangle$ $\mathcal{M}_3 = \langle \{b\}, \{a, e\} \rangle$	Preferred labelings	$\mathcal{L}_2 = (\{a\}, \{b\}, \{c, d, e\})$ $\mathcal{L}_3 = (\{b\}, \{a, e\}, \{c, d\})$
Stable models	None	Stable labelings	None
L-stable models	$\mathcal{M}_3 = \langle \{b\}, \{a, e\} \rangle$	Semi-stable Labelings	$\mathcal{L}_3 = (\{b\}, \{a, e\}, \{c, d\})$

Definition 13 ($\mathcal{L}2\mathcal{I}_{\mathfrak{A}}$ and $\mathcal{I}2\mathcal{L}_{\mathfrak{A}}$ Functions).
Let $\mathfrak{A}$ be a *SETAF* and P be its associated *NLP*, $\mathcal{L}ab$ be the set of all labelings of $\mathfrak{A}$ and $\mathcal{I}nt$ be the set of all the three-valued interpretations of $P_{\mathfrak{A}}$. We introduce the functions

- $\mathcal{L}2\mathcal{I}_{\mathfrak{A}} : \mathcal{L}ab \to \mathcal{I}nt$, in which

$$\mathcal{L}2\mathcal{I}_{\mathfrak{A}}(\mathcal{L}) = \langle \mathtt{in}(\mathcal{L}), \mathtt{out}(\mathcal{L}) \rangle .$$

  Obviously $\overline{\mathtt{in}(\mathcal{L}) \cup \mathtt{out}(\mathcal{L})} = \mathtt{undec}(\mathcal{L})$.
- $\mathcal{I}2\mathcal{L}_{\mathfrak{A}} : \mathcal{I}nt \to \mathcal{L}ab$, in which for $\mathcal{M} = \langle T, F \rangle \in \mathcal{I}nt$,

$$\mathcal{I}2\mathcal{L}_{\mathfrak{A}}(\mathcal{M}) = (T, F, \overline{T \cup F}).$$

In contrast with $\mathcal{L}2\mathcal{I}_P$ and $\mathcal{I}2\mathcal{L}_P$, the functions $\mathcal{L}2\mathcal{I}_{\mathfrak{A}}$ and $\mathcal{I}2\mathcal{L}_{\mathfrak{A}}$ are each other's inverse in the general case:

Theorem 8.
Let $\mathfrak{A} = (\mathcal{A}, Att)$ be a SETAF and $P_{\mathfrak{A}}$ its associated NLP.

- *For any labeling $\mathcal{L}$ of $\mathfrak{A}$, it holds $\mathcal{I}2\mathcal{L}_{\mathfrak{A}}(\mathcal{L}2\mathcal{I}_{\mathfrak{A}}(\mathcal{L})) = \mathcal{L}$.*
- *For any interpretation $\mathcal{I}$ of $P_{\mathfrak{A}}$, it holds $\mathcal{L}2\mathcal{I}_{\mathfrak{A}}(\mathcal{I}2\mathcal{L}_{\mathfrak{A}}(\mathcal{I})) = \mathcal{I}$.*

A similar result to Theorem 5 also holds here:

Theorem 9.
Let $\mathfrak{A}$ be a SETAF and $P_{\mathfrak{A}}$ be its associated NLP. It holds

- *$\mathcal{L}$ is a complete labeling of $\mathfrak{A}$ iff $\mathcal{L}2\mathcal{I}_{\mathfrak{A}}(\mathcal{L})$ is a partial stable model of $P_{\mathfrak{A}}$.*
- *$\mathcal{M}$ is a partial stable model of $P_{\mathfrak{A}}$ iff $\mathcal{I}2\mathcal{L}_{\mathfrak{A}}(\mathcal{M})$ is a complete labeling of $\mathfrak{A}$.*

From Theorem 9, we can ensure the equivalence between the semantics for *NLP* and their counterpart for *SETAF*:

Theorem 10.
Let $\mathfrak{A}$ be a SETAF and $P_{\mathfrak{A}}$ its associated NLP. It holds

1. *$\mathcal{L}$ is a grounded labeling of $\mathfrak{A}$ iff $\mathcal{L}2\mathcal{I}_{\mathfrak{A}}(\mathcal{L})$ is a well-founded model of $P_{\mathfrak{A}}$.*
2. *$\mathcal{L}$ is a preferred labeling of $\mathfrak{A}$ iff $\mathcal{L}2\mathcal{I}_{\mathfrak{A}}(\mathcal{L})$ is a regular model of $P_{\mathfrak{A}}$.*

https://doi.org/10.1017/S1471068424000188 Published online by Cambridge University Press

3. $\mathcal{L}$ *is a stable labeling of* $\mathfrak{A}$ *iff* $\mathcal{L}2\mathcal{I}_{\mathfrak{A}}(\mathcal{L})$ *is a stable model of* $P_{\mathfrak{A}}$.
4. $\mathcal{L}$ *is a semi-stable labeling of* $\mathfrak{A}$ *iff* $\mathcal{L}2\mathcal{I}_{\mathfrak{A}}(\mathcal{L})$ *is an L-stable model of* $P_{\mathfrak{A}}$.

The following result is a direct consequence of Theorems 8 and 10:

Corollary 11.
Let $\mathfrak{A}$ *be a SETAF and* $P_{\mathfrak{A}}$ *its associated NLP. It holds*

1. $\mathcal{M}$ *is a well-founded model of* $P_{\mathfrak{A}}$ *iff* $\mathcal{I}2\mathcal{L}_{\mathfrak{A}}(\mathcal{M})$ *is a grounded labeling of* $\mathfrak{A}$.
2. $\mathcal{M}$ *is a regular model of* $P_{\mathfrak{A}}$ *iff* $\mathcal{I}2\mathcal{L}_{\mathfrak{A}}(\mathcal{M})$ *is a preferred labeling of* $\mathfrak{A}$.
3. $\mathcal{M}$ *is a stable model of* $P_{\mathfrak{A}}$ *iff* $\mathcal{I}2\mathcal{L}_{\mathfrak{A}}(\mathcal{M})$ *is a stable labeling of* $\mathfrak{A}$.
4. $\mathcal{M}$ *is an L-stable model of* $P_{\mathfrak{A}}$ *iff* $\mathcal{I}2\mathcal{L}_{\mathfrak{A}}(\mathcal{M})$ *is a semi-stable labeling of* $\mathfrak{A}$.

Recalling the *SETAF* $\mathfrak{A}$ and its associated $P_{\mathfrak{A}}$ of Example 5, we obtain the expected equivalence results related to their semantics (see Table 1). In the next section, we will identify a class of *NLP*s in which the translation from a *SETAF* to an *NLP* (Definition 12) behaves as the inverse of the translation from an *NLP* to a *SETAF* (Definition 10).

5 On the relation between *RFALP*s and *SETAF*s

We will recall a particular kind of *NLP*s, called Redundancy-Free Atomic Logic Programs (*RFALPs*). From an *RFALP* P, we obtain its associated *SETAF* $\mathfrak{A}_P$ via Definition 10; from $\mathfrak{A}_P$, we obtain its associated *NLP* $P_{\mathfrak{A}_P}$ via Definition 12. By following the other direction, from a *SETAF* $\mathfrak{A}$, we obtain its associated *NLP* $P_{\mathfrak{A}}$, and from $P_{\mathfrak{A}}$, its associated *SETAF* $\mathfrak{A}_{P_{\mathfrak{A}}}$. An important result mentioned in this section is that $P = P_{\mathfrak{A}_P}$ and $\mathfrak{A} = \mathfrak{A}_{P_{\mathfrak{A}}}$; that is, the translation from an *NLP* to a *SETAF* and the translation from a *SETAF* to an *NLP* are each other's inverse. Next, we define *RFALP*s:

Definition 14 (RFALP (König et al. 2022)).
We define a Redundancy-Free Atomic Logic Program (*RFALP*) P as an *NLP* such that

1. P is redundancy-free; that is, $HB_P = \{head(r) \mid r \in P\}$, and if $c \leftarrow \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n \in P$, there is no rule $c \leftarrow \texttt{not}\ c_1, \ldots, \texttt{not}\ c_{n'} \in P$ such that $\{c_1, \ldots, c_{n'}\} \subset \{b_1, \ldots, b_n\}$.
2. P is atomic; that is, each rule has the form $c \leftarrow \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n$ $(n \geq 0)$.

First, Proposition 12 sustains that for any *SETAF* $\mathfrak{A}$, its associated *NLP* $P_{\mathfrak{A}}$ will always be an *RFALP*:

Proposition 12.
Let $\mathfrak{A} = (\mathcal{A}, Att)$ be a *SETAF* and $P_{\mathfrak{A}}$ its associated *NLP*. It holds $P_{\mathfrak{A}}$ is an *RFALP*.

The following results guarantee that $\mathfrak{A} = \mathfrak{A}_{P_{\mathfrak{A}}}$ (Theorem 13) and $P = P_{\mathfrak{A}_P}$ (Theorem 14):

Theorem 13.
Let $\mathfrak{A} = (\mathcal{A}, Att)$ *be a SETAF,* $P_{\mathfrak{A}}$ *its associated NLP, and* $\mathfrak{A}_{P_{\mathfrak{A}}}$ *the associated SETAF of* $P_{\mathfrak{A}}$. *It holds that* $\mathfrak{A} = \mathfrak{A}_{P_{\mathfrak{A}}}$.

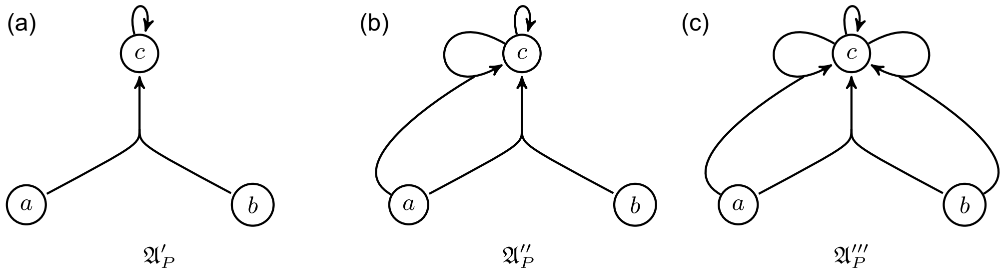


Fig. 5. Possible *SETAF*s associated with P.

Theorem 14.
Let P be an RFALP, $\mathfrak{A}_P$ its associated SETAF, and $P_{\mathfrak{A}_P}$ the associated NLP of $\mathfrak{A}_P$. It holds that $P = P_{\mathfrak{A}_P}$.

Remark 1.
Minimality is crucial to ensure that the translation from an *NLP* to a *SETAF* and the translation from a *SETAF* to an *NLP* are each other's inverse. If the minimality requirement in Definition 1 (and consequently in Definition 9) were dropped, any *SETAF* (among other combinations) in Figure 5 could be a possible candidate to be the associated *SETAF* $\mathfrak{A}_P$ of the *RFALP* P

$$c \leftarrow \texttt{not } a, \texttt{not } c \quad c \leftarrow \texttt{not } b, \texttt{not } c$$

$$a \qquad\qquad\qquad b$$

As a result, Theorem 13 would no longer hold, and these translations would not be each other's inverse. Notice also that the *SETAF*s in Figure 5 have the same complete labelings as non-minimal attacks are irrelevant and can be ignored when determining semantics based on complete labelings.

Theorems 13 and 14 reveal that *SETAF*s and *RFALP*s are essentially the same formalism. The equivalence between them involves their semantics and is also structural: two distinct *SETAF*s will always be translated into two distinct *RFALP*s and vice versa. In contradistinction, Theorem 14 would not hold if we had replaced our translation from *NLP* to *SETAF* (Definition 10) with that from *NLP* to *AAF* presented by Caminada *et al.* (2015b). Thus, the connection between *NLP*s and *SETAF*s is more robust than that between *NLP*s and *AAF*s. In the forthcoming section, we will explore how expressive *RFALP*s can be; we will ensure they are as expressive as *NLP*s.

6 On the expressiveness of *RFALP*s

Dvořák *et al.* (2019) comprehensively characterized the expressiveness of *SETAF*s. Now we compare the expressiveness of *NLP*s with that of *RFALP*s. In the previous section, we established that *SETAF*s and *RFALP*s are essentially the same formalism. We demonstrated that from the *SETAF* $\mathfrak{A}_P$ associated with an *NLP* P, we can obtain P; and conversely, from the *NLP* $P_{\mathfrak{A}}$ associated with a *SETAF* $\mathfrak{A}$, we can obtain $\mathfrak{A}$. Here,

https://doi.org/10.1017/S1471068424000188 Published online by Cambridge University Press

we reveal that this connection between *SETAF*s and *RFALP*s is even more substantial: *RFALP*s are as expressive as *NLP*s when considering the semantics for *NLP*s we have exploited in this paper. With this aim in mind, we transform any *NLP* P into an *RFALP* P^* by resorting to a specific combination (denoted by $\mapsto_{UTPM}$) of some program transformations proposed by Brass and Dix (1994, 1997, 1999). Although each program transformation in $\mapsto_{UTPM}$ was proposed by Brass and Dix (1994, 1997, 1999), the combination of these program transformations (as far as we know) has not been investigated yet. Then, we show that P and P^* share the same partial stable models. Since well-founded models, regular models, stable models, and L-stable models are all settled on partial stable models, it follows that both P and P^* also coincide under these semantics. Based on Dunne *et al.*'s (2015) work, where they define the notion of expressiveness of the semantics for *AAF*s, we define formally expressiveness in terms of the signatures of the semantics for *NLP*s:

Definition 15 (Expressiveness).
Let $\mathcal{P}$ be a class of *NLP*s. The signature $\Sigma^{\mathcal{P}}_{PSM}$ of the partial stable models associated with $\mathcal{P}$ is defined as

$$\Sigma^{\mathcal{P}}_{PSM} = \{\sigma(P) \mid P \in \mathcal{P}\},$$

where $\sigma(P) = \{\mathcal{I} \mid \mathcal{I} \textit{ is a partial stable model of } P\}$ is the set of all partial stable models of P.

Given two classes $\mathcal{P}_1$ and $\mathcal{P}_2$ of *NLP*s, we say that $\mathcal{P}_1$ and $\mathcal{P}_2$ have the same expressiveness for the partial stable models semantics if $\Sigma^{\mathcal{P}_1}_{PSM} = \Sigma^{\mathcal{P}_2}_{PSM}$

In other words, $\mathcal{P}_1$ and $\mathcal{P}_2$ have the same expressiveness if

- For every $P_1 \in \mathcal{P}_1$, there exists $P_2 \in \mathcal{P}_2$ such that P_1 and P_2 have the same set of partial stable models.
- For every $P_2 \in \mathcal{P}_2$, there exists $P_1 \in \mathcal{P}_1$ such that P_1 and P_2 have the same set of partial stable models.

Similarly, we can define when $\mathcal{P}_1$ and $\mathcal{P}_2$ have the same expressiveness for the well-founded, regular, stable, and L-stable semantics.

As the class of *RFALP*s is contained in the class of all *NLP*s, to show that these classes have the same expressiveness for these semantics, it suffices to prove that for every *NLP*, there exists an *RFALP* with the same set of partial stable models. We will obtain this result by resorting to a combination of program transformations:

Definition 16 (Program Transformation (Brass and Dix, 1994, 1997, 1999)).
A program transformation is any binary relation $\mapsto$ between *NLP*s. By $\mapsto^*$ we mean the reflexive and transitive closure of $\mapsto$.

Thus, $P \mapsto^* P'$ means that there is a finite sequence $P = P_1 \mapsto \cdots \mapsto P_n = P'$. We are particularly interested in program transformations preserving partial stable models:

Definition 17 (Equivalence Transformation (Brass and Dix, 1994, 1997, 1999)).
We say a program transformation $\mapsto$ is a partial stable model equivalence transformation if for any *NLP*s P_1 and P_2 with $P_1 \mapsto P_2$, it holds $\mathcal{M}$ is a partial stable model of P_1 iff $\mathcal{M}$ is a partial stable model of P_2.

From Definitions 18 to 21, we focus on the following program transformations introduced by Brass and Dix (1994, 1997, 1999): *Unfolding* (*it is also known as Generalized Principle of Partial Evaluation* (*GPPE*)), *elimination of tautologies*, *positive reduction*, and *elimination of non-minimal Rules*. They are sufficient for our purposes.

Definition 18 (Unfolding (Brass and Dix, 1994, 1997, 1999)).
An *NLP* P_2 results from an *NLP* P_1 by unfolding ($P_1 \mapsto_U P_2$) iff there exists a rule $c \leftarrow a, a_1, \ldots, a_m, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n \in P_1$ such that

$$P_2 = (P_1 - \{c \leftarrow a, a_1, \ldots, a_m, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n\})$$

$$\cup \{c \leftarrow a_1{}', \ldots, a_p{}', a_1, \ldots, a_m, \texttt{not}\ b_1{}', \ldots, \texttt{not}\ b_q{}', \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n \mid$$

$$a \leftarrow a_1{}', \ldots, a_p{}', \texttt{not}\ b_1{}', \ldots, \texttt{not}\ b_q{}' \in P_1\}.$$

Definition 19 (Elimination of Tautologies (Brass and Dix, 1994, 1997, 1999)).
An *NLP* P_2 results from an *NLP* P_1 by elimination of tautologies ($P_1 \mapsto_T P_2$) iff there exists a rule $r \in P_1$ such that $head(r) \in body^+(r)$ and $P_2 = P_1 - \{r\}$.

Definition 20 (Positive Reduction (Brass and Dix, 1994, 1999)).
An *NLP* P_2 results from an *NLP* P_1 by positive reduction ($P_1 \mapsto_P P_2$) iff there exists a rule $c \leftarrow a_1, \ldots, a_m, \texttt{not}\ b, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n \in P_1$ such that $b \notin \{head(r) \mid r \in P_1\}$ and

$$P_2 = (P_1 - \{c \leftarrow a_1, \ldots, a_m, \texttt{not}\ b, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n\})$$

$$\cup \{c \leftarrow a_1, \ldots, a_m, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_n\}\,.$$

Definition 21 (Elimination of Non-Minimal Rules (Brass and Dix, 1994, 1999)).
An *NLP* P_2 results from an *NLP* P_1 by elimination of non-minimal rules ($P_1 \mapsto_M P_2$) iff there are two distinct rules r and r' in P_1 such that $head(r) = head(r')$, $body^+(r') \subseteq body^+(r)$, $body^-(r') \subseteq body^-(r)$ and $P_2 = P_1 - \{r\}$.

Now we combine these program transformations and define $\mapsto_{UTPM}$ as follows:

Definition 22 (Combined Transformation).
Let $\mapsto_{UTPM} = \mapsto_U \cup \mapsto_T \cup \mapsto_P \cup \mapsto_M$.

We call an *NLP* P *irreducible* concerning $\mapsto$ if there is no *NLP* $P' \neq P$ with $P \mapsto^* P'$. Besides, we say $\mapsto$ is *strongly terminating* iff every sequence of successive applications of $\mapsto$ eventually leads to an irreducible *NLP*. As displayed by Brass and Dix (1998), not every program transformation is strongly terminating. For instance, in the *NLP*

$$a \leftarrow b$$

$$b \leftarrow a$$

$$c \leftarrow a, \texttt{not}\ c$$

$$c$$

if we apply unfolding ($\mapsto_U$) to the third rule, this rule is replaced by $c \leftarrow b, \texttt{not}\ c$. We can now apply unfolding again to this rule and get the original program; such an oscillation can repeat indefinitely. Thus we have a sequence of program transformations that do not

https://doi.org/10.1017/S1471068424000188 Published online by Cambridge University Press

terminate. However, if we restrict ourselves to fair sequences of program transformations, the termination is guaranteed:

Definition 23 (Fair Sequences (Brass and Dix, 1998)).
A sequence of program transformations $P_1 \mapsto \cdots \mapsto P_n$ is fair with respect to $\mapsto$ if

- Every positive body atom occurring in P_1 is eventually removed in some P_i with $1 < i \leq n$ (either by removing the whole rule using a suitable program transformation or by an application of $\mapsto_U$);
- Every rule $r \in P_i$ such that $head(r) \in body^+(r)$ is eventually removed in some P_j with $i < j \leq n$ (either by applying $\mapsto_T$ or another suitable program transformation).

The sequence above of program transformations is not fair, because it does not remove the positive body atoms occurring in the program. In contrast, the sequence of program transformations given by

$$\begin{array}{l} a \leftarrow b \\ b \leftarrow a \\ c \leftarrow a, \texttt{not}\ c \\ c \end{array} \mapsto_U \begin{array}{l} a \leftarrow a \\ b \leftarrow a \\ c \leftarrow a, \texttt{not}\ c \\ c \end{array} \mapsto_T \begin{array}{l} b \leftarrow a \\ c \leftarrow a, \texttt{not}\ c \\ c \end{array} \mapsto_U \begin{array}{l} b \leftarrow a \\ c \end{array} \mapsto_U c$$

is not only fair but also terminates. The next result guarantees that it is not simply a coincidence:

Theorem 15.
The relation $\mapsto_{UTPM}$ is strongly terminating for fair sequences of program transformations; that is, such fair sequences always lead to irreducible programs.

Theorem 15 is crucial to obtain the following result:

Theorem 16.
For any NLP P, there exists an irreducible NLP P^* such that $P \mapsto^*_{UTPM} P^*$.

This means that from an *NLP* P, it is always possible to obtain an irreducible *NLP* P^* after successive applications of $\mapsto_{UTPM}$. Indeed, P^* is an *RFALP*:

Theorem 17.
Let P be an NLP and P^* be an NLP obtained after applying repeatedly the program transformation $\mapsto_{UTPM}$ until no further transformation is possible; that is, $P \mapsto^*_{UTPM} P^*$ and P^* is irreducible. Then P^* is an RFALP.

From Theorems 15 and 17, we can infer that for fair sequences, after applying repeatedly $\mapsto_{UTPM}$, we will eventually produce an *RFALP*. In fact, every *RFALP* is irreducible:

Theorem 18.
Let P be an RFALP. Then P is irreducible with respect to $\mapsto_{UTPM}$.

Theorems 16 and 17 guarantee that every *NLP* P can be transformed into an *RFALP* P^* by applying $\mapsto_{UTPM}$ a finite number of times. It remains to show that P and P^* share the same partial stable models (and consequently, the same well-founded, regular, stable, and L-stable models). Before, however, note that $\mapsto_{UTPM}$ does not introduce new atoms; instead, it can eliminate the occurrence of existing atoms in an *NLP*. For simplicity in notation, we assume throughout the rest of this section that $HB_P = HB_{P'}$ whenever $P \mapsto^*_{UTPM} P'$. Next, we recall that these program transformations preserve the least models of positive programs:

Lemma 19 (Brass and Dix, 1995, 1997).
Let P_1 and P_2 be positive programs such that $P_1 \mapsto_x P_2$, in which $x \in \{U, T, P, M\}$. It holds $\mathcal{M}$ is the least model of P_1 iff $\mathcal{M}$ is the least model of P_2.

In the sequel, we aim to extend Lemma 19 to *NLP*s. Notice, however, that we already have the result for the program transformation $\mapsto_U$:

Theorem 20 (Aravindan and Minh 1995).
Let P_1 and P_2 be NLPs such that $P_1 \mapsto_U P_2$. It holds $\mathcal{M}$ is a partial stable model of P_1 iff $\mathcal{M}$ is a partial stable model of P_2.

It remains to guarantee the result for the program transformation $\mapsto_T$, $\mapsto_P$ and $\mapsto_M$:

Theorem 21.
Let P_1 and P_2 be NLPs such that $P_1 \mapsto_T P_2$. It holds $\mathcal{M}$ is a partial stable model of P_1 iff $\mathcal{M}$ is a partial stable model of P_2.

Theorem 22.
Let P_1 and P_2 be NLPs such that $P_1 \mapsto_P P_2$. It holds $\mathcal{M}$ is a partial stable model of P_1 iff $\mathcal{M}$ is a partial stable model of P_2.

Theorem 23.
Let P_1 and P_2 be NLPs such that $P_1 \mapsto_M P_2$. It holds $\mathcal{M}$ is a partial stable model of P_1 iff $\mathcal{M}$ is a partial stable model of P_2.

Consequently, if $P_1 \mapsto_{UTPM} P_2$, then P_1 and P_2 share the same partial stable models. By repeatedly resorting to this result, we can even show that for any *NLP*, there exists an irreducible *NLP* with the same set of partial stable models, well-founded models, regular models, stable models, and L-stable models:

Theorem 24.
Let P be an NLP and P^* be an irreducible NLP such that $P \mapsto^*_{UTPM} P^*$. It holds $\mathcal{M}$ is a partial stable model of P iff $\mathcal{M}$ is a partial stable model of P^*.

Corollary 25.
Let P be an NLP and P^* be an irreducible NLP such that $P \mapsto^*_{UTPM} P^*$. It holds $\mathcal{M}$ is a well-founded, regular, stable, L-stable model of P iff $\mathcal{M}$ is, respectively, a well-founded, regular, stable, L-stable model of P^*.

As any irreducible *NLP* is an *RFALP* (Theorem 17), the following result is immediate:

Corollary 26.
For any NLP P, there exists an RFALP P^* such that $\mathcal{M}$ is a partial stable, well-founded, regular, stable, L-stable model of P iff $\mathcal{M}$ is, respectively, a partial stable, well-founded, regular, stable, L-stable model of P^*.

Given that each *NLP* can be associated with an *RFALP* preserving the semantics above, it follows that *NLP* and *RFALP*s have the same expressiveness for those semantics:

Theorem 27.
NLPs and RFALPs have the same expressiveness for partial stable, well-founded, regular, stable, and L-stable semantics.

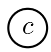
Another important result is that the *SETAF* corresponding to an *NLP* is invariant with respect to $\mapsto_{UTPM}$:

Theorem 28.
For any NLPs P_1 and P_2, if $P_1 \mapsto_{UTPM} P_2$, then $\mathfrak{A}_{P_1} = \mathfrak{A}_{P_2}$

This means that any *NLP* in a sequence of program transformations from $\mapsto_{UTPM}$ has the same corresponding *SETAF*. For instance, every *NLP* in this sequence

$$\begin{array}{l} a \leftarrow b \\ b \leftarrow a \\ c \leftarrow a, \texttt{not}\ c \\ c \end{array} \mapsto_U \begin{array}{l} a \leftarrow a \\ b \leftarrow a \\ c \leftarrow a, \texttt{not}\ c \\ c \end{array} \mapsto_T \begin{array}{l} b \leftarrow a \\ c \leftarrow a, \texttt{not}\ c \\ c \end{array} \mapsto_U \begin{array}{l} b \leftarrow a \\ c \end{array} \mapsto_U c$$

leads to the same corresponding *SETAF*, constituted by a unique (unattacked) argument:



Theorem 28 also suggests an alternative way to find the *SETAF* corresponding to an *NLP* P: instead of resorting directly to Definition 8 to construct the arguments, we can apply (starting from P) $\mapsto_{UTPM}$ successively by following a fair sequence of program transformations. By Theorems 15 and 17, we know that eventually, we will reach an *RFALP* whose corresponding *SETAF* is identical to that of the original program P (Theorem 28). Then, we apply Definition 8 to this *RFALP* to obtain the arguments and Definition 9 for the attack relation. Notably, when P is an *RFALP*, Definition 8 becomes considerably simpler, requiring only its first item to characterize the statements.

In addition, from the same *NLP*, various fair sequences of program transformations can be conceived. Recalling the *NLP*

$$\begin{array}{l} a \leftarrow b \\ b \leftarrow a \\ c \leftarrow a, \texttt{not}\ c \\ c \end{array}$$

exploited above, we can design the following alternative fair sequence

$$\begin{array}{l} a \leftarrow b \\ b \leftarrow a \\ c \leftarrow a, \texttt{not}\ c \\ c \end{array} \mapsto_M \begin{array}{l} a \leftarrow a \\ b \leftarrow a \\ c \end{array} \mapsto_U \begin{array}{l} a \leftarrow b \\ b \leftarrow b \\ c \end{array} \mapsto_T \begin{array}{l} a \leftarrow b \\ c \end{array} \mapsto_U c$$

https://doi.org/10.1017/S1471068424000188 Published online by Cambridge University Press

This sequence produced the same *RFALP* as before; it is not a coincidence. Apart from being strongly terminating for fair sequences of program transformations, the relation $\mapsto_{UTPM}$ has an appealing property; it is also confluent:

Theorem 29.
The relation $\mapsto_{UTPM}$ *is confluent; that is, for any NLPs* P, P' *and* P'', *if* $P \mapsto^*_{UTPM} P'$ *and* $P \mapsto^*_{UTPM} P''$ *and both* P' *and* P'' *are irreducible, then* $P' = P''$.

By confluent $\mapsto_{UTPM}$, we mean that it does not matter the path we take by repeatedly applying $\mapsto_{UTPM}$, if it ends, it will always lead to the same irreducible *NLP*. In addition, as any irreducible *NLP* is an *RFALP* (Theorem 17), and the translations from *SETAF* to *RFALP*s and conversely, from *RFALP*s to *SETAF* are each other's inverse (Theorems 13 and 14), we obtain that two distinct *SETAF*s will always be associated with two distinct *NLP*s. The confluence of $\mapsto_{UTPM}$ is of particular significance from the logic programming perspective as it guarantees that the ordering of the transformations in *UTPM* does not matter: we are free to choose always the "best" transformation, which maximally reduces the program. Consequently, Theorem 29 also sheds light on the search for efficient implementations in *NLP*s.

From the previous section, we know that the equivalence between *SETAF*s and *RFALP*s is not only of a semantic nature but also structural: two distinct *SETAF*s will always be translated into two distinct *RFALP*s and vice versa. Now we enhance our understanding of this result still more by establishing that

- *RFALP*s are as expressive as *NLP*s.
- The *SETAF* corresponding to an *NLP* is invariant with respect to $\mapsto_{UTPM}$; that is, if $P_1 \mapsto_{UTPM} P_2$, then $\mathfrak{A}_{P_1} = \mathfrak{A}_{P_2}$.
- Each *NLP* P leads to a unique *RFALP* P^* via the relation $\mapsto_{UTPM}$. Besides, P and P^* have the same partial stable, grounded, regular, stable, and L-stable models.

Beyond revealing the connections between *SETAF*s and *NLP*s, the results in this paper also enhance our understanding of *NLP*s themselves. To give a concrete example, let us consider the following issue: in the sequence of program transformations in $\mapsto_{UTPM}$, atoms can be removed. Are these atoms underivable and set to false in the partial stable models of the program or true/undecided atoms can be removed in this sequence? Such questions can be answered by considering some results from Section 3 and the current section. In more formal terms, let P and P^* be *NLP*s such that $P \mapsto^*_{UTPM} P^*$ and P^* is irreducible. We have

- P^* is an *RFALP* (Theorem 17), and the set of atoms occurring in P^* is $\{head(r) \mid r \in P^*\}$ (Definition 14);
- $\mathcal{A}_{P^*} = \{head(r) \mid r \in P^*\}$ is the set of all arguments we can construct from P^* (Definition 8), and $\mathfrak{A}_P = \mathfrak{A}_{P^*}$ (Theorem 28), that is, $\mathcal{A}_P = \mathcal{A}_{P^*}$;
- Thus c occurs in P, but does not occur in P^* iff there is no statement s constructed from P such that $\mathtt{Conc}(s) = c$. According to Corollary 2, $c \in F'$ for every interpretation $\mathcal{I}$ with $\Omega_P(\mathcal{I}) = \langle T', F' \rangle$.

Consequently, every atom occurring in P, but not occurring in P^* is set to false in the least three-valued model of each disjunct of P. In particular, they will be false in its partial stable models.

Supported by the findings presented in the current section, we can argue that *SETAF*s and *RFALP*s are essentially the same paradigm, and both are deeply connected with *NLP*s.

7 Conclusion and future works

This paper investigates the connections between frameworks with sets of attacking arguments (*SETAF*s) and Normal Logic Programs (*NLP*s). Building on the research of Alcântara *et al.* (2019); Alcântara and Sá (2021), we employ the characterization of the *SETAF* semantics in terms of labelings (Flouris and Bikakis 2019) to establish a mapping from *NLP*s to *SETAF*s (and vice versa). We further demonstrate the equivalence between partial stable, well-founded, regular, stable, and L-stable models semantics for *NLP*s and, respectively, complete, grounded, preferred, stable, and semi-stable labelings for *SETAF*s.

Our translation from *NLP*s to *SETAF*s offers a key advantage over the translation from *NLP*s to *AAF*s presented by Caminada *et al.* (2015b). Our approach captures the equivalence between semi-stable labelings for *SETAF*s and L-stable models for *NLP*s. In addition, their translation is unable to preserve the structure of the *NLP*s. While an *NLP* can be translated to an *AAF*, recovering the original *NLP* from the corresponding *AAF* is generally not possible. In contradistinction, we have revisited a class of *NLP*s called Redundancy-Free Atomic Logic Programs (*RFALP*s). For *RFALP*s, the translations from *NLP*s to *SETAF*s and from *SETAF*s to *NLP*s also preserve their structures as they are each other's inverse. Hence, when compared to the relationship between *NLP*s and *AAF*s, the relationship between *NLP*s and *SETAF*s is demonstrably more robust. It extends beyond semantics to encompass structural aspects.

Some of these results are not new as they have already been obtained independently by König *et al.* (2022). In fact, their translation from *NLP*s to *SETAF*s and vice versa coincide with ours, and the structural equivalence between *RFALP*s and *SETAF*s has also been identified there. Notwithstanding, our proofs of these results stem from a significantly distinct path as they are based on properties of argument labelings and are deeply rooted in the works of Caminada *et al.* (2015b); Alcântara *et al.* (2019); Alcântara and Sá (2021). For instance, our equivalence results are settled on two important aspects:

- Properties involving the maximization/minimization of labelings adapted from Caminada *et al.*'s (2015b) work to deal with labelings for *SETAF*s.
- Again inspired by Caminada *et al.* (2015b), we proposed a mapping from interpretations to labelings and a mapping from labelings to interpretations. We also showed that they are each other's inverse.

In contrast, König *et al.* (2022) demonstrated the equivalence between the semantics in terms of extensions. They also have not tackled the controversy between semi-stable and L-stable, one of our leading motivations for developing this work.

In addition to showing this structural equivalence between *RFALP*s and *SETAF*s, we have also investigated the expressiveness of *RFALP*s. To demonstrate that they are as expressive as *NLP*s, we proved that any *NLP* can be transformed into an *RFALP* with the same partial stable models through repeated applications of the program transformation $\mapsto_{UTPM}$. It is worth noticing that $\mapsto_{UTPM}$ results from the combination of the following program transformations presented by Brass and Dix (1994, 1997, 1999): unfolding, elimination of tautologies, positive reduction, and elimination of non-minimal rules. In the course of our investigations, we also have obtained relevant findings as follows:

- *RFALP*s are irreducible with respect to $\mapsto_{UTPM}$: the application of $\mapsto_{UTPM}$ to an *RFALP* will result in the same program.
- The mapping from *NLP*s to *SETAF*s is invariant with respect to the program transformation $\mapsto_{UTPM}$; that is, if an *NLP* P_2 is obtained from an *NLP* P_1 via $\mapsto_{UTPM}$, then the *SETAF* corresponding to P_1 is the same corresponding to P_2.
- The program transformation $\mapsto_{UTPM}$ is confluent: any *NLP* will lead to a unique *RFALP* after repeatedly applying $\mapsto_{UTPM}$. Consequently, two distinct *RFALP*s will always be associated with two distinct *NLP*s.

In summary, *RFALP*s (which are as expressive as *NLP*s) and *SETAF*s are essentially the same formalism. Roughly speaking, we can consider a *SETAF* as a graphical representation of an *RFALP* and an *RFALP* as a rule-based representation of a *SETAF*. Any change in one formalism is mirrored by a corresponding change in the other. Thus, *SETAF*s emerge as a natural candidate for representing argumentation frameworks corresponding to *NLP*s.

Regarding the significance and potential impact of our results, we highlight that by pursuing this line of research, one gains insight into what forms of non-monotonic reasoning can and cannot be represented by formal argumentation. In particular, by enlightening these connections between *SETAF*s and *NLP*s, many approaches, semantics, and techniques naturally developed for the former may be applied to the latter, and vice versa. While *SETAF*s serve as an inspiration for defining *RFALP*s, the representation of *NLP*s as *SETAF*s is an alternative for intuitively visualizing logic programs.

In addition, our results associated with the confluence of $\mapsto_{UTPM}$ are of particular significance from the logic programming perspective as they guarantee that the ordering of the transformations in $\mapsto_{UTPM}$ does not matter: we are free to choose always the "best" transformation, which maximally reduces the program. Consequently, our paper also sheds light on the search for efficient implementations in *NLP*s.

Natural ramifications of this work include an in-depth analysis of other program transformations beyond those studied here and their impact on *SETAF* and argumentation in general. Given the close relationship between argumentation and logic programming, a possible line of research is to investigate how Argumentation can benefit from these program transformations in the development of more efficient algorithms. The structural connection involving *RFALP*s and *SETAF*s gives rise to exploiting other extensions of Dung *AAF*s; in particular, we are interested in identifying which of them are robust enough to preserve the structure of logic programs. Along this same line of research, it is also our aim to study connections between extensions of *NLP*s (including their paraconsistent semantics) and Argumentation.

Competing interests

The authors declare none.

Supplementary material

To view supplementary material for this article, please visit http://doi.org/10.1017/S1471068424000188.

References

Alcântara, J. and Sá, S. 2021. Equivalence results between SETAF and attacking abstract dialectical frameworks. In *Proceedings NMR*, 139–148.

Alcântara, J., Sá, S. and Acosta-Guadarrama, J. 2019. On the equivalence between abstract dialectical frameworks and logic programs. *Theory and Practice of Logic Programming* 19, 5–6, 941–956.

Aravindan, C. and Minh, D. P. 1995. On the correctness of unfold/fold transformation of normal and extended logic programs. *The Journal of Logic Programming* 24, 3, 201–217.

M., Beirlaen, Heyninck, J., Pardo, P.and Strasser, C. 2018. Argument strength in formal argumentation. *FLAP* 5, 3, 629–676.

Bondarenko, A., Dung, P. M., Kowalski, R. A. and Toni, F. 1997. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence* 93, 1–2, 63–101.

Brass, S. and Dix, J. 1994. A disjunctive semantics based on unfolding and bottom-up evaluation. In *Innovationen Bei Rechen-und Kommunikationssystemen (IFIP-Congress, Workshop FG2: Disjunctive Logic Programming and Disjunctive Databases)*, Berlin Heidelberg, Springer, 83–91.

Brass, S. and Dix, J. 1995. Disjunctive semantics based upon partial and bottom-up evaluation. In *ICLP*, 199–213.

Brass, S. and Dix, J. 1997. Characterizations of the disjunctive stable semantics by partial evaluation. *The Journal of Logic Programming* 32, 3, 207–228.

Brass, S. and Dix, J. 1998. Characterizations of the disjunctive well-founded semantics: Confluent calculi and iterated gcwa. *Journal of Automated Reasoning* 20, 143–165.

Brass, S. and Dix, J. 1999. Semantics of (disjunctive) logic programs based on partial evaluation. *The Journal of Logic Programming* 40, 1, 1–46.

Brewka, G., Ellmauthaler, S., Strass, H., Wallner, J. P. and Woltran, S. 2013. Abstract dialectical frameworks revisited. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, 803–809.

Brewka, G. and Woltran, S. 2010. Abstract dialectical frameworks. In *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning*, 102–111.

Caminada, M. 2006. Semi-stable semantics. In *1st International Conference on Computational Models of Argument (COMMA)*, 144, 121–130.

Caminada, M. and Amgoud, L. 2005. An axiomatic account of formal argumentation. In *AAAI*, 6, 608–613.

Caminada, M. and Amgoud, L. 2007. On the evaluation of argumentation formalisms. *Artificial Intelligence* 171, 5–6, 286–310.

Caminada, M., Harikrishnan, S. and Sá, S. 2022. Comparing logic programming and formal argumentation; the case of ideal and eager semantics. *Argument & Computation* 13, 1, 93–120.

Caminada, M., Sá, S., Alcântara, J. and Dvořák, W. 2015a. On the difference between assumption-based argumentation and abstract argumentation. *IFCoLog Journal of Logic and its Applications* 2a, 1, 15–34.

Caminada, M., Sá, S., Alcântara, J. and Dvořák, W. 2015b. On the equivalence between logic programming semantics and argumentation semantics. *International Journal of Approximate Reasoning* 58b, 87–111.

Caminada, M. and Schulz, C. 2017. On the equivalence between assumption-based argumentation and logic programming. *Journal of Artificial Intelligence Research* 60, 779–825.

Caminada, M. W. and Gabbay, D. M. 2009. A logical account of formal argumentation. *Studia Logica* 93, 2–3, 109.

Dung, P. 1995a. An argumentation procedure for disjunctive logic programs. *Journal of Logic Programming* 24a, 151–177.

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

Dung, P. M., Kowalski, R. A. and Toni, F. 2009. Assumption-based argumentation. In *Argumentation in Artificial Intelligence*, Springer, 199–218.

Dunne, P. E., Dvořák, W., Linsbichler, T. and Woltran, S. 2015. Characteristics of multiple viewpoints in abstract argumentation. *Artificial Intelligence* 228, 153–178.

Dvořák, W., Fandinno, J. and Woltran, S. 2019. On the expressive power of collective attacks. *Argument & Computation* 10, 2, 191–230.

Dvořák, W., Gaggl, S. A., Wallner, J. P. and Woltran, S. 2013. Making use of advances in answer-set programming for abstract argumentation systems. In *Applications of Declarative Programming and Knowledge Management*, Springer, 114–133.

Dvořák, W., Rapberger, A. and Woltran, S. 2023. A claim-centric perspective on abstract argumentation semantics: Claim-defeat, principles, and expressiveness. *Artificial Intelligence* 324, 104011.

Eiter, T., Leone, N. and Saccá, D. 1997. On the partial semantics for disjunctive deductive databases. *Annals of Mathematics and Artificial Intelligence* 19, 1–2, 59–96.

Flouris, G. and Bikakis, A. 2019. A comprehensive study of argumentation frameworks with sets of attacking arguments. *International Journal of Approximate Reasoning* 109, 55–86.

Gorogiannis, N. and Hunter, A. 2011. Instantiating abstract argumentation with classical logic arguments: Postulates and properties. *Artificial Intelligence* 175, 9–10, 1479–1497.

König, M., Rapberger, A. and Ulbricht, M. 2022. Just a matter of perspective: Intertranslating expressive argumentation formalisms. In *Proceeding of the 9th International Conference on Computational Models of Argument (COMMA)*, 212–223.

Nielsen, S. H. and Parsons, S. 2006. A generalization of Dung's abstract framework for argumentation: Arguing with sets of attacking arguments. In *International Workshop on Argumentation in Multi-Agent Systems*, Springer, 54–73.

Nieves, J. C., Cortés, U. and Osorio, M. 2008. Preferred extensions as stable models. *Theory and Practice of Logic Programming* 8, 4, 527–543.

Przymusinski, T. 1990. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae* 13, 4, 445–463.

Rapberger, A. 2020. Defining argumentation semantics under a claim-centric view. In *STAIRS@ ECAI.*

Rocha, V. H. N. and Cozman, F. G. 2022b. A credal least undefined stable semantics for probabilistic logic programs and probabilistic argumentation. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 19, 309–319.

Rocha, V. H. N. and Cozman, F. G. 2022a. Bipolar argumentation frameworks with explicit conclusions: Connecting argumentation and logic programming. In *CEUR Workshop Proceedings*, CEUR-WS, 3197, 49–60.

Sá, S. and Alcântara, J. 2019. Interpretations and models for assumption-based argumentation. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 1139–1146.

Sá, S. and Alcântara, J. 2021a. An abstract argumentation and logic programming comparison based on 5-valued labellings. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty: 16th European Conference, ECSQARU 2021*, Springer, 159–172.

Sá, S. and Alcântara, J. 2021b. Assumption-based argumentation is logic programming with projection. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty: 16th European Conference, ECSQARU 2021*, Springer, 173–186.

Schulz, C. and Toni, F. 2015. Logic programming in assumption-based argumentation revisited-semantics and graphical representation. In *29th AAAI Conference on Artificial Intelligence.*

Toni, F. 2014. A tutorial on assumption-based argumentation. *Argument & Computation* 5, 1, 89–117.

Toni, F. and Sergot, M. 2011. Argumentation and answer set programming. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, Springer, 164–180.

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

Wu, Y., Caminada, M. and Gabbay, D. M. 2009. Complete extensions in argumentation coincide with 3-valued stable models in logic programming. *Studia Logica* 93, 2–3, 383.