SYNTACTIC AND SEMANTIC CONNECTIONS BETWEEN LOGIC PROGRAMMING AND ARGUMENTATION SYSTEMS

SAMY SÁ
Universidade Federal do Ceará, Brazil
samy@ufc.br

WOLFGANG DVOŘÁK
TU Wien, Institute of Logic and Computation, Austria
dvorak@dbai.tuwien.ac.at

MARTIN CAMINADA
Cardiff University, United Kingdom
CaminadaM@cardiff.ac.uk

Abstract

Logic programming was one of the first formalisms to incorporate non-monotonic reasoning and, as such, is the origin of many semantics for this type of reasoning. Many of the core argumentation systems, including Abstract Argumentation, Assumption-Based Argumentation and Abstract Dialectical Frameworks even find their historical roots in the logic programming literature, borrowing terminology, procedures, notation and semantics from this niche. In this chapter, we provide an overview of the connections between logic programming and a series of argumentation systems, focusing on the semantic perspective to find their relative expressive power. The systems we examine in detail include the ones we already mentioned, as well as Argumentation Frameworks with Sets of Attacking Arguments. In each case, we consider translations and find whether they preserve the semantics of their respective source and target formalism, under some of the most common semantics. For some of the cases where equivalence does not hold, we consider how to restore it. Apart from that, we also offer an overview of how some of these argumentation systems can be implemented using Answer-Set Programming and their specialized solvers.

1 Introduction

In the current book chapter, we will examine the connections between logic programming and a series of formalisms for non-monotonic reasoning which we gather under the umbrella of *argumentation systems*. As one would expect, the first connections are historical: the investigations leading to the proposal of Abstract Argumentation Frameworks (AFs) [Dung, 1995b], often taken as the seminal work in computational argumentation theory, happened amidst the development of semantics for logic programming with negation as failure [Dung, 1991; Kakas *et al.*, 1994; Dung, 1995a]. Just alike, several core argumentation systems find their roots in the literature of logic programming, including the approaches of Defeasible Logic Programming (DeLP) [Simari and Loui, 1992; García and Simari, 2004; García and Simari, 2018], Assumption-Based Argumentation (ABA) [Bondarenko *et al.*, 1997; Dung *et al.*, 2009; Čyras *et al.*, 2018] and Abstract Dialectical Frameworks (ADFs) [Brewka and Woltran, 2010; Brewka *et al.*, 2013]. Immediately, it was argued that some of these systems could model the semantic entailment relations from logic programming using translations.

One well-known example can be found in [Dung, 1995b], where the author showed how to translate a normal logic program (NLP) into an AF. Based on this translation, Dung proved that the *stable models* (resp. the *well-founded model*) of an NLP correspond to the *stable extensions* (resp. the *grounded extension*) of its corresponding AF. These results led to several studies concerning connections between them [Dung, 1995b; Nieves *et al.*, 2008; Caminada and Gabbay, 2009; Egly *et al.*, 2010; Toni and Sergot, 2011; Caminada *et al.*, 2015b; Sá and Alcântara, 2021a; Caminada *et al.*, 2022] . Notably, in [Caminada and Gabbay, 2009], they proved that the *three-valued stable models* of a NLP correspond to the *complete extensions* of its corresponding AF. Later, [Caminada *et al.*, 2015b] investigated whether the same results would hold for the particular cases of the three-valued models and complete extensions semantics, showing there are some exceptions to the expected correspondences.

Just like in these works, here we will primarily be concerned with semantics, investigating whether each argumentation system we examine is able (or not) to model the entailment of logic programming (and vice-versa). On that matter, we will provide a detailed overview of the connections between logic programming with AFs, ABA, ADFs and Argumentation Frameworks with

Sets of Attacking Arguments (SETAFs) [Nielsen and Parsons, 2006; Bikakis *et al.*, 2021]. Other notable systems will be mentioned in connection to logic programs only briefly, as required of each case.

Besides motivation and semantics, some connections naturally arise in the implementation of argumentation systems using logic programming solvers. More specifically, given the non-monotonic nature and the fact that typical reasoning tasks in argumentation are NP/coNP-hard, answer-set programming [Gelfond and Lifschitz, 1991] is an immediate choice for the implementation of argumentation systems. This connection has been explored and developed in multiple works, such as [Nieves *et al.*, 2008; Wakaki and Nitta, 2008; Egly *et al.*, 2008; Egly *et al.*, 2010; Dvořák *et al.*, 2011; Dvořák *et al.*, 2015; Ellmauthaler and Strass, 2014; Dvořák and Wallner, 2020; Sakama and Rienstra, 2017; Cyras and Toni, 2016; Lehtonen *et al.*, 2017; Lehtonen *et al.*, 2021b].

The current chapter is structured as follows. First, in Section 2, we introduce the necessary concepts and notation we require to discuss logic programs. Then, in Section 3, we examine how Abstract Argumentation can be used to model the entailment of Logic Programming and vice versa. The next couple of sections follow the same approach to compare logic programming to other notable argumentation systems: in Section 4, we examine how Assumption-Based Argumentation can be used to model the entailment of Logic Programming (and vice-versa) and in Section 5 we do the same for Abstract Dialectical Frameworks and for Frameworks with Sets of Attacking Arguments. In Section 6, we discuss possibilities for the implementation of argumentation systems based on Answer-Set Programming (ASP) solvers. We round off with a discussion in Section 7, where we mention a few other systems worth notice and their connection to logic programming.

2 Logic Programs: Syntax and Semantics

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

Definition 1. *A* logic programming rule *(or simply* a rule*, for short) is an*

expression

$$x \leftarrow y_1, \ldots, y_n, \texttt{not } z_1, \ldots, \texttt{not } z_m \quad (n \geq 0,\ m \geq 0)$$

where x, each y_i ($1 \leq i \leq n$) and each z_j ($1 \leq j \leq m$) is an atom, and not *represents negation as failure (NAF). A* logic program *(or simply* a program*) P consists of a finite set of rules.*

Intuitively, a rule r such as $x \leftarrow y_1, \ldots, y_n, \texttt{not } z_1, \ldots, \texttt{not } z_m$ expresses there is a proof for x if each of $y_1, \ldots, y_n$ can be proven while each of $z_1, \ldots, z_n$ cannot. Moving forward, we may refer to x as the *head* or *consequent* of the rule (writing $head(r) = x$) and to $y_1, \ldots, y_n, \texttt{not } z_1, \ldots, \texttt{not } z_m$ as its *body* (writing $body(r) = \{y_1, \ldots, y_n, \texttt{not } z_1, \ldots, \texttt{not } z_m\}$). Moreover, we say that $body^+(r) = \{y_1, \ldots, y_n\}$ is the strong part of the body and that $body^-(r) = \{\texttt{not } z_1, \ldots, \texttt{not } z_m\}$ is the weak part of the body. Each expression $\texttt{not } w$, where w is an atom, is called a *NAF literal*. Then, a rule is NAF-free iff it does not contain NAF literals (i.e., iff $m = 0$). Similarly, a program is NAF-free iff all of its rules are NAF-free. Finally, the Herbrand Base of a logic program P (written as HB_P) is the set of all atoms in P.

In the following, we recall the definitions of logic programming semantics found in [Przymusinski, 1990; Caminada *et al.*, 2015b; Caminada and Schulz, 2017], but in slightly different fashion for the sake of uniformity. We will comment on what is different as we advance through the concepts.

Definition 2. *A* 3-valued interpretation *of a logic program P with respect to a set of atoms $Atms \supseteq HB_P$ is a triple $I = (T, F, U)$* [1] *such that T, F and U are pairwise disjoint and $T \cup F \cup U = Atms$.*

Intuitively, a *3-valued interpretation* (or simply *an interpretation*) of P w.r.t. $Atms \supseteq HB_P$ evaluates the atoms in $Atms$ according to the truth values *true*, *false* and *undecided*. Then, given an interpretation $I = (T, F, U)$, the atoms in T are said to be *true*, the atoms in F are said to be *false* and those in U are said to be *undecided*. Further, each interpretation I can be characterized as a function $I : Atms \rightarrow \{true, false, undecided\}$.

Given an interpretation, a program may be transformed into a corresponding NAF-free program through the notion of *reduct*:

[1] Traditionally, program interpretations are presented as a pair (T, F), where the result of $U = HB_P \setminus (T \cup F)$ is left implicit. Making U explicit helps with uniformity in our text.

Definition 3. *The* reduct *of a logic program P w.r.t. an interpretation $I = (T, F, U)$, written as P^I, is obtained by replacing in all rules of P the occurrences of each NAF literal* `not` x *by* $\mathbf{t}$ *if* $x \in F$*, by* $\mathbf{f}$ *if* $x \in T$*, and by* $\mathbf{u}$ *otherwise.*

In the context of a program reduct, $\mathbf{t}$, $\mathbf{f}$ and $\mathbf{u}$ are auxiliary terms interpreted as positive literals (atoms) not in HB_P. As such, the reduct of any program P is necessarily a NAF-free program. Semantically, each interpretation $I = (T, F, U)$ of P w.r.t. HB_P is extended into a corresponding interpretation $\check{I} = (\check{T}, \check{F}, \check{U})$ of P w.r.t. $HB_P \cup \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ such that $\check{T} = T \cup \{\mathbf{t}\}$, $\check{F} = F \cup \{\mathbf{f}\}$, and $\check{U} = U \cup \{\mathbf{u}\}$.

Concerning NAF-free programs, the notion of 3-valued model easily follows:

Definition 4. *Given a NAF-free program P, an interpretation $I = (T, F, U)$ of P w.r.t. $Atms \supseteq HB_P$ is a* 3-valued model *(or simply a model) of P if, for each rule $x \leftarrow y_1, \ldots, y_n$ in P, it holds that*

$$I(x) \geq \mathtt{min}(\{\check{I}(y_i) \mid i \in \{1, \ldots, n\}\})$$

following the truth order true $>$ undecided $>$ false.

Intuitively, in a model of P, the head of each rule is at least as true as the least true literal in its body.

When P is a NAF-free logic program (possibly containing $\mathbf{t}$, $\mathbf{f}$ or $\mathbf{u}$), the existence of a unique *minimal* 3-valued model $\Phi(P) = (T, F, U)$ with minimal T and maximal F (w.r.t. $\subseteq$) among all 3-valued models of P is ensured [Przymusinski, 1990]. Then, since any program reduct P^I is NAF-free, it necessarily has a unique minimal 3-valued model $\Phi(P^I)$. This leads to the core semantics we will discuss for logic programs in this work:

Definition 5. *([Przymusinski, 1990]) Let* `Mod` *be a model of P. We say that*

`Mod` *is a* 3-valued stable model *of P iff* $\Phi(P^{\mathtt{Mod}}) = \mathtt{Mod}$.

We now recall various logic programming semantics which are based on 3-valued interpretations. The presentation below was originally provided in [Caminada *et al.*, 2015b; Caminada *et al.*, 2022], where correspondences to other equivalent concepts for the same semantics were discussed. It is heavily

based on Przymusinski's three-valued stable semantics [Przymusinski, 1990] and tailored to ease the comparison to argumentation semantics[2].

Definition 6. *([Caminada* et al.*, 2015b; Caminada* et al.*, 2022]) Let P be a logic program and* $\texttt{Mod} = (T, F, U)$ *be a 3-valued interpretation of P w.r.t. HB_P. We say that* Mod *is:*

1. *the* well-founded model *of P iff* Mod *is the 3-valued stable model where T is $\subseteq$-minimal among all 3-valued stable models of P*
2. *a* regular model *of P iff* Mod *is a 3-valued stable model where T is $\subseteq$-maximal among all 3-valued stable models of P*
3. *a* 2-valued stable model *of P iff* Mod *is a 3-valued stable model where $T \cup F = HB_P$*
4. *an* L-stable model *of P iff* Mod *is a 3-valued stable model where U is $\subseteq$-minimal among all 3-valued stable models of P*
5. *a* pre-ideal model *of P iff* Mod *is a 3-valued stable model where $T \subseteq T_{reg}$ for each regular model $\texttt{Mod}_{reg} = (T_{reg}, F_{reg}, U_{reg})$ of P*
6. *the* ideal model *of P iff* Mod *is the pre-ideal model where T is $\subseteq$-maximal among all pre-ideal models of P*
7. *a* pre-eager model *of P iff* Mod *is a 3-valued stable model where $T \subseteq T_{ls}$ for each L-stable model $\texttt{Mod}_{ls} = (T_{ls}, F_{ls}, U_{ls})$ of P*
8. *the* eager model *of P iff* Mod *is the pre-eager model where T is $\subseteq$-maximal among all pre-eager models of P*

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

[2] A similar characterization of these semantics (except well-founded) as special cases of the three-valued stable semantics (there called P-stable models) can be found in [Saccà and Zaniolo, 1997]

3 On the Connection between Abstract Argumentation and Logic Programming

In the current section, we will show how abstract argumentation and logic programming are related. Each system has its own syntax and a particular variety of semantics, which relate to the evaluation of a particular set of sentences. For the logic programs we are interested in, the sentences are propositional atoms, whereas in argumentation frameworks, the sentences are called arguments. To proceed, we must consider back and forth translations: one will show how an argumentation framework can be encoded as a logic program (which we will call `AA2LP`), the other will show how a logic program can be encoded by an argumentation framework (which we will call `LP2AA`). In each case, we will show whether some semantics for the destiny system captures some semantics from the origin system. The discussion we conduct in this section summarizes results obtained in [Dung, 1995b; Caminada and Gabbay, 2009; Caminada *et al.*, 2015b; Sá and Alcântara, 2021a; Caminada *et al.*, 2022].

3.1 Abstract Argumentation: Syntax and Semantics

Intuitively, an argumentation framework portrays a set of arguments and the conflicts among them. The conflicts are modelled as a binary relation over the arguments, leading to the instantiation of an argument graph. For current purposes, we restrict ourselves to finite argumentation frameworks.

Definition 7. *([Dung, 1995b]) An* argumentation framework *is a pair* $\mathrm{AF} = (Ar, att)$ *where* Ar *is a finite set of arguments and* $att \subseteq Ar \times Ar$.

Argumentation semantics are commonly presented in the form of *argument extensions* [Dung, 1995b] or *argument labellings*, which, as explained in [Caminada, 2006; Caminada and Gabbay, 2009; Baroni *et al.*, 2018], coincide with their respective extension-based variants. The core semantics for argumentation frameworks is commonly considered to be the *complete semantics*, especially because a variety of other argumentation semantics can be obtained as particular cases of that semantics.

Definition 8. *Let* $AF = (Ar, att)$ *be an argumentation framework. An* argument labelling *is a function* $\texttt{ArgLab} : Ar \rightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$.

Given an argument labelling $\texttt{ArgLab}$, we write $\texttt{val}(\texttt{ArgLab})$ to refer to the set of arguments labelled as $\texttt{val} \in \{\texttt{in}, \texttt{out}, \texttt{undec}\}$ in $\texttt{ArgLab}$. For convenience, we may as well refer to $\texttt{ArgLab}$ as the 3-tuple $(\texttt{in}(\texttt{ArgLab}), \texttt{out}(\texttt{ArgLab}), \texttt{undec}(\texttt{ArgLab}))$. We invite the reader to notice how argument labellings are inherently similar to the interpretations found in logic programming (see Definition 2).

Definition 9. *Let $AF = (Ar, att)$ be an argumentation framework. An argument labelling* $\texttt{ArgLab}$ *is called a* complete argument labelling *iff for each $A \in Ar$ it holds that:*

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

It was shown in [Caminada and Gabbay, 2009] that the complete semantics for abstract argumentation corresponds to the 3-valued stable model semantics of logic programming. The result is obtained through suitable translations from abstract argumentation frameworks to logic programs (and back) and mapping the complete labellings of the framework to the 3-valued stable models of the corresponding program (and vice-versa). From there, the same translations can be used to obtain a series of additional results [Caminada *et al.*, 2015b; Caminada *et al.*, 2022] regarding the correspondence of different argumentation and logic programming semantics.

Before we introduce the translations and properly show how those results are obtained, we must define the other semantics we require for argumentation frameworks. On that matter, we invite the reader to observe the similarities between Definition 6 and Definition 10.

Definition 10. *Let* $\texttt{ArgLab}$ *be an argument labelling of argumentation framework $AF = (Ar, att)$.* $\texttt{ArgLab}$ *is called:*

- *the* grounded argument labelling *iff* $\texttt{ArgLab}$ *is a complete argument labelling where* $\texttt{in}(\texttt{ArgLab})$ *is $\subseteq$-minimal among all complete argument labellings of AF*

- *a* preferred argument labelling *iff* $\texttt{ArgLab}$ *is a complete argument labelling where* $\texttt{in}(\texttt{ArgLab})$ *is* $\subseteq$*-maximal among all complete argument labellings of* AF
- *a* stable argument labelling *iff* $\texttt{ArgLab}$ *is a complete argument labelling where* $\texttt{undec}(\texttt{ArgLab}) = \emptyset$
- *a* semi-stable argument labelling *iff* $\texttt{ArgLab}$ *is a complete argument labelling where* $\texttt{undec}(\texttt{ArgLab})$ *is* $\subseteq$*-minimal among all complete argument labellings of* AF
- *a* pre-ideal argument labelling[3] *iff* $\texttt{ArgLab}$ *is a complete argument labelling where* $\texttt{in}(\texttt{ArgLab}) \subseteq \texttt{in}(\texttt{ArgLab}_{pr})$ *for each preferred argument labelling* $\texttt{ArgLab}_{pr}$ *of* AF
- *the* ideal argument labelling *iff* $\texttt{ArgLab}$ *is a pre-ideal argument labelling and* $\texttt{in}(\texttt{ArgLab})$ *is* $\subseteq$*-maximal among all pre-ideal argument labellings of* AF
- *a* pre-eager argument labelling *iff* $\texttt{ArgLab}$ *is a complete argument labelling where* $\texttt{in}(\texttt{ArgLab}) \subseteq \texttt{in}(\texttt{ArgLab}_{sem})$ *for each semi-stable argument labelling* $\texttt{ArgLab}_{sem}$ *of* AF
- *the* eager argument labelling *iff* $\texttt{ArgLab}$ *is a pre-eager argument labelling where* $\texttt{in}(\texttt{ArgLab})$ *is* $\subseteq$*-maximal among all pre-eager argument labellings of* AF

Each argumentation framework has one or more complete labellings, a unique grounded labelling, one or more preferred labellings, zero or more stable labellings, one or more semi-stable labellings[4], one or more pre-ideal labellings, a unique ideal labelling, one or more pre-eager labellings and a unique eager labelling.

[3] A similar concept is also present in [Dung *et al.*, 2007] where the ideal extension is defined in terms of *ideal sets*. We opted for a slightly narrower concept based on the complete semantics (instead of the admissible semantics) following [Caminada *et al.*, 2022]. This is merely a matter of choice, since the ideal extension of an argumentation framework is necessarily complete [Dung *et al.*, 2007].

[4] This is because we only consider finite argumentation frameworks. An infinite argumentation framework can have zero or more semi-stable labellings [Caminada and Verheij, 2010; Weydert, 2011; Baumann and Spanring, 2015; Baumann, 2018].

3.2 From Abstract Argumentation to Logic Programming

Now we will turn our attention to how argumentation semantics can be modelled by logic programming semantics via a suitable translation.

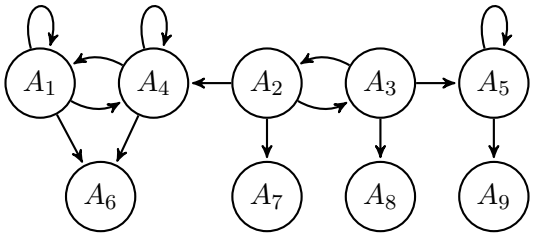
Intuitively, given an argumentation framework, an argument can be *accepted* only if all of its attackers are *rejected* (i.e., *not accepted*). That comprehension leads to a straightforward translation from argumentation frameworks to logic programs which can be found in [Wu *et al.*, 2009; Caminada *et al.*, 2015b]:

Definition 11. *([Wu* et al.*, 2009]) Let $AF = (Ar, att)$ be an argumentation framework, the logic program associated to AF is*

$$\texttt{AA2LP}(AF) = P_{AF} = \{A \leftarrow \texttt{not } B_1, \ldots, \texttt{not } B_m \mid A \in Ar \text{ and } \{B_i \mid (B_i, A) \in att\} = \{B_1, \ldots, B_m\}\}.$$

Given AF, the program P_{AF} lists one rule for each argument a in AF expressing there is a proof for a (semantically, that it is *true*) if each of its attackers cannot be proven (semantically, if they are *false*).A special case is an argument (say A) without any attackers, which translates to a rule "$A \leftarrow$" with an empty body. Because there is only one rule for each argument, the exclusive set of conditions allows the rules to be read as "if and only if".

Example 1. *Consider the argumentation framework $AF = (\{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9\}, \{(A_1, A_1), (A_1, A_4), (A_1, A_6), (A_2, A_3), (A_2, A_4), (A_2, A_7), (A_3, A_2), (A_3, A_5), (A_3, A_8), (A_4, A_1), (A_4, A_4), (A_4, A_6), (A_5, A_5), (A_5, A_9)\})$, depicted below:*



Its associated logic program P_{AF} is[5]*:*

[5]Please notice that we use the names of the arguments as atoms in the associated logic

$r_1: \ A_1 \leftarrow \texttt{not}\ A_1, \texttt{not}\ A_4$ $\quad$ $r_2: \ A_2 \leftarrow \texttt{not}\ A_3$
$r_3: \ A_3 \leftarrow \texttt{not}\ A_2$ $\quad$ $r_4: \ A_4 \leftarrow \texttt{not}\ A_1, \texttt{not}\ A_2, \texttt{not}\ A_4$
$r_5: \ A_5 \leftarrow \texttt{not}\ A_3, \texttt{not}\ A_5$ $\quad$ $r_6: \ A_6 \leftarrow \texttt{not}\ A_1, \texttt{not}\ A_4$
$r_7: \ A_7 \leftarrow \texttt{not}\ A_2$ $\quad$ $r_8: \ A_8 \leftarrow \texttt{not}\ A_3$
$r_9: \ A_9 \leftarrow \texttt{not}\ A_5$

Notice how the rules in P_{AF} immediately describe the conditions for each argument in AF to be accepted based on the attack relation. For instance, the rule $r_2: \ A_2 \leftarrow \texttt{not}\ A_3$ expresses that A_2 should be proven (i.e. accepted) if and only if A_3 is not. Indeed A_3 is the only attacker of A_2 in AF. Furthermore, if AF had an argument B with no attackers, a corresponding rule "$r_B: \ B \leftarrow$" with empty body would occur in P_{AF}.

Programs such as P_{AF} pertain to the class of *AF-Programs* ([Caminada *et al.*, 2015b]), which includes all logic programs corresponding to the description of an argumentation framework.[6]

Definition 12. *(AF-Program [Caminada* et al.*, 2015b]) A logic program P is an* AF-Program *if for each $c \in HB_P$ there is at most one rule with conclusion c.*

Albeit simple, the translation function `AA2LP` was shown to preserve the semantics of any input argumentation frameworks [Caminada *et al.*, 2015b; Caminada *et al.*, 2022] for all the *complete semantics* and its particular cases listed in Definition 10. In fact, `AA2LP` preserves the complete labelling semantics from the input argumentation framework without change[7]:

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

[6] Please notice that although each logic program that is the result of translating an argumentation framework is an AF-Program, it is *not* the case that every AF-Program can be the result of translating an argumentation framework. A counter example would be an AF-Program with a strong literal in the body of one of its rules.

[7] The original results require a translation between argumentation labellings and program models, but only because they defined program interpretations as a pair (T, F) (leaving implicit the set of undecided atoms) instead of a tuple (T, F, U). Given an argumentation labelling $(\texttt{in}, \texttt{out}, \texttt{undec})$, their translation involved only omitting `undec` to provide $(\texttt{in}, \texttt{out})$ as the resulting interpretation.

Theorem 1. *([Caminada and Gabbay, 2009]) Let $AF = (Ar, att)$ be an argumentation framework and* $\texttt{ArgLab} = (\texttt{in}, \texttt{out}, \texttt{undec})$ *be a complete labelling of AF. Then* ArgLab *is a 3-valued stable model of P_{AF}.*

This result rules in favor of logic programming subsuming abstract argumentation, since it portrays the *coincidence* of semantics, not merely a way to map labellings to models. A range of results similar to that of Theorem 1, immediately follows.

Corollary 2. *Let $AF = (Ar, att)$ be an argumentation framework,* $\texttt{ArgLab} = (\texttt{in}, \texttt{out}, \texttt{undec})$ *be a complete argument labelling of AF and* $\texttt{AA2LP}(AF) = P_{\mathrm{AF}}$. *Then:*

- *If* ArgLab *is* grounded*, then* ArgLab *is the well-founded model of P_{AF}.*
- *If* ArgLab *is* preferred*, then* ArgLab *is a regular model of P_{AF}.*
- *If* ArgLab *is* stable*, then* ArgLab *is a 2-valued stable model of P_{AF}.*
- *If* ArgLab *is* semi-stable*, then* ArgLab *is a L-stable model of P_{AF}.*
- *If* ArgLab *is* pre-ideal*, then* ArgLab *is a pre-ideal model of P_{AF}.*
- *If* ArgLab *is* ideal*, then* ArgLab *is the ideal model of P_{AF}.*
- *If* ArgLab *is* pre-eager*, then* ArgLab *is a pre-eager model of P_{AF}.*
- *If* ArgLab *is* eager*, then* ArgLab *is the eager model of P_{AF}.*

3.3 From Logic Programming to Abstract Argumentation

Moving from normal logic programming to abstract argumentation requires more steps and intricate machinery.

We start with describing how the rules of a Logic Program can be used to construct arguments. For this, we revisit the approach of [Caminada *et al.*, 2015b] with a slight change to include *default arguments*, which were introduced in [Sá and Alcântara, 2021a].

Definition 13. *Let P be a logic program, we define the arguments and default arguments induced by P as follows:*

- *If c is an atom in HB_P and there is at least one $r \in P$ for which $head(r) = c$, then* not c *is a* default *argument (say D_c) with*
  - $\texttt{Conc}(D_c) = \texttt{not}\ c$,
  - $\texttt{Rules}(D_c) = \emptyset$
  - $\texttt{Vul}(D_c) = \{c\}$, *and*
  - $\texttt{Sub}(D_c) = \{D_c\}$.
- *If $c \leftarrow \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ is a rule in P, then it is also an argument (say A) with*
  - $\texttt{Conc}(A) = c$,
  - $\texttt{Rules}(A) = \{c \leftarrow \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m\}$,
  - $\texttt{Vul}(A) = \{b_1, \ldots, b_m\}$, *and*
  - $\texttt{Sub}(A) = \{A\}$.
- *If $c \leftarrow a_1, \ldots, a_n, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ is a rule in P and for each a_i ($1 \leq i \leq n$) there exists an argument A_i with $\texttt{Conc}(A_i) = a_i$ and such that $c \leftarrow a_1, \ldots, a_n, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ is not contained in $\texttt{Rules}(A_i)$, then $c \leftarrow (A_1), \ldots, (A_n), \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m$ is an argument (say A) with*
  - $\texttt{Conc}(A) = c$,
  - $\texttt{Rules}(A) = \texttt{Rules}(A_1) \cup \ldots \cup \texttt{Rules}(A_n) \cup \{c \leftarrow a_1, \ldots, a_n, \texttt{not}\ b_1, \ldots, \texttt{not}\ b_m\}$
  - $\texttt{Vul}(A) = \texttt{Vul}(A_1) \cup \ldots \cup \texttt{Vul}(A_n) \cup \{b_1, \ldots, b_m\}$, *and*
  - $\texttt{Sub}(A) = \{A\} \cup \texttt{Sub}(A_1) \cup \ldots \cup \texttt{Sub}(A_n)$.

In essence, an argument can be seen as a tree-like structure of rules (the only difference with a real tree is that a rule can occur at more than one place in the argument) corresponding to a possible proof for some atom in the language of the program. Following that idea, default arguments concern possible proofs that can be drawn from the program using no rules. Default arguments model the fact that NAF-literals are true by default in every semantics of a logic program, hence their name.

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

The next step in constructing the argumentation framework is to determine the attack relation: an argument attacks another iff its conclusion is among the vulnerabilities of the attacked argument. With that we mind, we can propose:

Definition 14. *Let P be a logic program. The argumentation framework associated with P is*

$$\texttt{LP2AA}(P) = AF_P = (Ar_P, att_P)$$

where Ar_P is the set of arguments from P (Definition 13) and

$$att_P = \{(A, B) \mid \texttt{Conc}(A) \in \texttt{Vul}(B)\}$$

.

Concerning AF_P, please notice that:

- Each argument that is not a default argument attacks one and only one default argument.
- Each default argument attacks zero arguments[8].

Therefore, in AF_P, one can differentiate whether two arguments have the same conclusion or not [Sá and Alcântara, 2021a]: it suffices to check if they attack the same default argument, which in turn are the only arguments in AF_P that do not attack any arguments. This is an advantage over the original definition of [Caminada *et al.*, 2015b].

We can can now apply argumentation semantics to the resulting argumentation framework and, based on the resulting argument labelling(s), obtain their associated conclusion labelling(s) using the approach of [Caminada *et al.*, 2015b; Caminada *et al.*, 2022].

Definition 15. *([Caminada* et al.*, 2015b; Caminada* et al.*, 2022]) Let P be a logic program. A conclusion labelling of P is a function* $\texttt{ConcLab} : HB_P \rightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$. *Let* $\texttt{ArgLab}$ *be an argument labelling of AF_P. We say that*

[8]This is because the conclusion of a default argument is a NAF-literal, which are never among the vulnerabilities of arguments.

`ConcLab` *is the* associated conclusion labelling *of* `ArgLab` *iff* `ConcLab` *is a conclusion labelling such that for each* $c \in HB_P$ *it holds that*

$$\texttt{ConcLab}(c) = max(\{\texttt{ArgLab}(A) \mid \texttt{Conc}(A) = c\} \cup \{\texttt{out}\})$$

where $\texttt{in} > \texttt{undec} > \texttt{out}$.

We say that a conclusion labelling is complete *iff it is the associated conclusion labelling of a complete argument labelling.*

Further, we define the function `ArgLab2ConcLab` *such that for any complete argument labelling* `ArgLab`*, we have that* $\texttt{ArgLab2ConcLab}(\texttt{ArgLab})$ *is the conclusion labelling associated with* `ArgLab`*. Finally, we define the functions* `ConcLab2ArgLab` *as the inverse of* `ArgLab2ConcLab`[9].

Fundamentally, conclusion labellings and program interpretations are the same, the only difference being the naming convention for the truth values in use[10]. It has been shown in [Wu *et al.*, 2009; Caminada *et al.*, 2015b] that complete conclusion labellings coincide with 3-valued stable models[11].

Theorem 3. *([Wu* et al.*, 2009; Caminada* et al.*, 2015b]) Let* P *be a logic program and* $AF_P = (Ar_P, att_P)$ *be its associated argumentation framework. It holds that:*

1. *if* `Mod` *is a 3-valued stable model of* P *then* `Mod` *is a complete conclusion labelling of* P
2. *if* `ConcLab` *is a complete conclusion labelling of* P *then* `ConcLab` *is a 3-valued stable model of* P

Once again, we obtain a coincidence result, but one should notice that conclusion labellings are not considered argumentation semantics, since they

[9] It has been shown in [Caminada *et al.*, 2015b] that when restricted to complete argument labellings and complete conclusion labellings, `ArgLab2ConcLab` and `ConcLab2ArgLab` are both bijective and each others inverse.

[10] In previous works, such as in [Wu *et al.*, 2009; Caminada *et al.*, 2015b], the authors considered special functions `ConcLab2Mod` and `Mod2ConcLab` to convert between conclusion labellings and logic programming models, but they are not required here due to our choice of notation for program interpretations.

[11] The results reported were obtained using the translation from [Wu *et al.*, 2009; Caminada *et al.*, 2015b]. The only difference is that we add default arguments to the set of arguments obtained from a program in Definition 13. It was shown in [Sá and Alcântara, 2021a] that the introduction of the extra arguments preserves all the results from previous works.

do not evaluate arguments. For this reason, we have a higher interest in the corresponding result regarding complete argument labellings:

Theorem 4. *([Wu* et al.*, 2009; Caminada* et al.*, 2015b]) Let P be a logic program and $AF_P = (Ar_P, att_P)$ be its associated argumentation framework. Then* `ArgLab` *is a complete argument labelling of AF_P if and only if* `ArgLab2ConcLab(ArgLab)` *is a 3-valued stable model of P.*

Differently from the previous result, this one does not involve an immediate *coincidence*, only a way to map corresponding models and labellings. This difference is quite significant[12]: given a program P,

- if one retrieves the complete *conclusion* labellings of AF_P having minimal/maximal `in`/`out`/`undec`, they *will coincide* with the 3-valued stable models having minimal/maximal *true*/*false*/*undec* (due to Theorem 3);
- but if one retrieves the complete *argument* labellings of AF_P having minimal/maximal `in`/`out`/`undec`, they *may or may not correspond* to the 3-valued stable models having minimal/maximal *true*/*false*/*undec* [Caminada *et al.*, 2015b].

The following example[13] illustrates a scenario where minimizing undecidness at the argument level is not the same as minimizing undecidedness at the conclusion level.

Example 2. *Consider the following logic program P:*

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

One can then build the arguments from P:

[12] These results are discussed at length in [Caminada *et al.*, 2015b] and complemented in [Caminada *et al.*, 2022].

[13] The program example and its corresponding AF are adapted from [Caminada *et al.*, 2015b; Caminada *et al.*, 2022] to include default arguments. One of the original arguments was also removed for compactness.

- $A_1 = r_1$, *with* $\texttt{Conc}(A_1) = c$ *and* $\texttt{Vul}(A_1) = \{c\}$
- $A_2 = r_2$, *with* $\texttt{Conc}(A_2) = a$ *and* $\texttt{Vul}(A_2) = \{b\}$
- $A_3 = r_3$, *with* $\texttt{Conc}(A_3) = b$ *and* $\texttt{Vul}(A_3) = \{a\}$
- $A_4 = r_4$, *with* $\texttt{Conc}(A_4) = c$ *and* $\texttt{Vul}(A_4) = \{c, a\}$
- $A_5 = r_5$, *with* $\texttt{Conc}(A_5) = g$ *and* $\texttt{Vul}(A_5) = \{g, b\}$
- $A_6 = \texttt{not}\ c$, *with* $\texttt{Conc}(A_6) = \texttt{not}\ c$ *and* $\texttt{Vul}(A_6) = \{c\}$
- $A_7 = \texttt{not}\ a$, *with* $\texttt{Conc}(A_7) = \texttt{not}\ a$ *and* $\texttt{Vul}(A_7) = \{a\}$
- $A_8 = \texttt{not}\ b$, *with* $\texttt{Conc}(A_8) = \texttt{not}\ b$ *and* $\texttt{Vul}(A_8) = \{b\}$
- $A_9 = \texttt{not}\ g$, *with* $\texttt{Conc}(A_9) = \texttt{not}\ g$ *and* $\texttt{Vul}(A_9) = \{g\}$

The associated argumentation framework AF_P of P is shown in Figure 1.

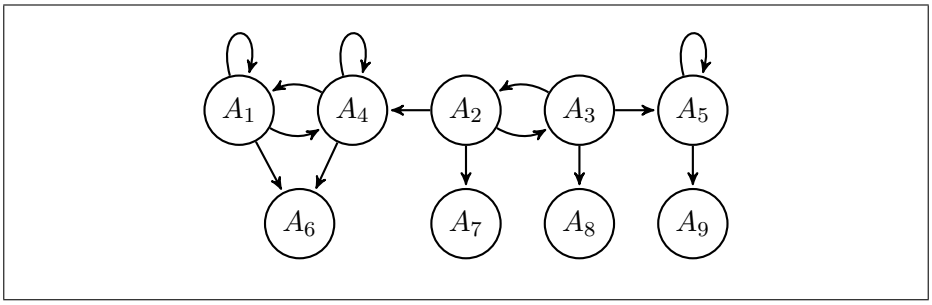


Figure 1: The argumentation framework AF_P associated with P.

The complete argument labellings of AF_P are

- $\texttt{ArgLab}_1 = (\emptyset, \emptyset, \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9\})$
- $\texttt{ArgLab}_2 = (\{A_2, A_8\}, \{A_3, A_4, A_7\}, \{A_1, A_5, A_6, A_9\})$
- $\texttt{ArgLab}_3 = (\{A_3, A_7, A_9\}, \{A_2, A_5, A_8\}, \{A_1, A_4, A_6\})$

The associated complete conclusion labellings are

- $\texttt{ConcLab}_1 = (\emptyset, \emptyset, \{a, b, c, g\})$,

- $\texttt{ConcLab}_2 = (\{a\}, \{b\}, \{c, g\})$, *and*
- $\texttt{ConcLab}_3 = (\{b\}, \{a, g\}, \{c\})$.

$\texttt{ArgLab}_2$ *and* $\texttt{ArgLab}_3$ *are semi-stable argument labellings, that is, complete argument labellings where* undec *is* $\subseteq$*-minimal. Hence, the associated conclusion labellings* $\texttt{ConcLab}_2$ *and* $\texttt{ConcLab}_3$ *are semi-stable conclusion labellings. However, because* $\texttt{undec}(\texttt{ConcLab}_2) \supset \texttt{undec}(\texttt{ConcLab}_3)$*, we find that* $\texttt{ConcLab}_2$ *is not an L-stable model of* P*. So here we have an example of a logic program where the semi-stable and L-stable conclusion labellings do not correspond.*

Further, because $\texttt{ArgLab}_2$ *and* $\texttt{ArgLab}_3$ *are both semi-stable argument labellings,* $\texttt{ArgLab}_1$ *is the only pre-eager argument labelling and the eager argument labelling of* AF_P*. On the other hand, since only* $\texttt{ConcLab}_3$ *is an L-stable model of* P*, both* $\texttt{ConcLab}_1$ *and* $\texttt{ConcLab}_3$ *are pre-eager models and* $\texttt{ConcLab}_3$ *is the eager model of* P*. Therefore, the pre-eager and the eager argument semantics may fail to capture the pre-eager and eager semantics for logic programs.*

Overall, the results found in [Caminada *et al.*, 2015b; Caminada *et al.*, 2022] can be summarized as follows:

Theorem 5. *([Caminada* et al.*, 2015b; Caminada* et al.*, 2022]) Let* P *be a logic program,* AF_P *be its associated argumentation framework and* ArgLab *be a complete argument labelling of* AF_P*. Then:*

1. ArgLab *is grounded if and only if* $\texttt{ArgLab2ConcLab}(\texttt{ArgLab})$ *is well-founded*
2. ArgLab *is preferred if and only if* $\texttt{ArgLab2ConcLab}(\texttt{ArgLab})$ *is regular*
3. ArgLab *is stable if and only if* $\texttt{ArgLab2ConcLab}(\texttt{ArgLab})$ *is stable*
4. ArgLab *is pre-ideal if and only if* $\texttt{ArgLab2ConcLab}(\texttt{ArgLab})$ *is pre-ideal*
5. ArgLab *is ideal if and only if* $\texttt{ArgLab2ConcLab}(\texttt{ArgLab})$ *is ideal*
6. *If* ArgLab *is semi-stable,* $\texttt{ArgLab2ConcLab}(\texttt{ArgLab})$ *may not be L-stable (and vice-versa).*

7. *If* `ArgLab` *is pre-eager,* `ArgLab2ConcLab(ArgLab)` *may not be pre-eager (and vice-versa).*

8. *If* `ArgLab` *is eager,* `ArgLab2ConcLab(ArgLab)` *may not be eager (and vice-versa).*

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

3.4 The Conundrum of Minimizing Undecided Arguments vs Undecided Conclusions

The non-correspondence result concerning the semi-stable argument semantics and L-stable logic programming semantics (from [Caminada *et al.*, 2015b]) motivated further investigation in attempts to understand their differences and whether it is possible to devise an argumentation semantics that captures the L-stable semantics. Among these efforts, Sá and Alcântara observed that the difference is always related to some arguments whose attackees coincide: in their redundancy, one or more of those arguments would become irrelevant to the evaluation of those arguments they mutually attack [Sá and Alcântara,

[14] It has to be mentioned that formalisms such as ASPIC+ [Modgil and Prakken, 2013; Modgil and Prakken, 2014; Modgil and Prakken, 2018], ABA [Bondarenko *et al.*, 1997; Toni, 2014; Čyras *et al.*, 2018] and logic-based argumentation [Gorogiannis and Hunter, 2011; Besnard and Hunter, 2014] have been stated in terms of extensions instead of in terms of labellings. However, as extensions and labellings coincide [Caminada, 2006; Caminada and Gabbay, 2009; Baroni *et al.*, 2018] they could be viewed in terms of labellings as well.

2021a]. They also identified that sink[15] arguments played a major role in pinpointing the culprit arguments. As we consider the instantiation of default arguments in Definition 13, they correspond precisely to the sinks in the argumentation framework associated with a given program. Further, Definition 13 ensures there is precisely one default argument for each atom that can be proven in a program, intuitively allowing the definition of new argumentation semantics that maximize/minimize the labels of default arguments. This led to the proposal of the L-stable argumentation semantics.

Definition 16. *Let* ArgLab *be an argument labelling of* AF*. Given*

$$\texttt{SINKS}_{\text{AF}} = \{A \in Ar \mid \forall B \in Ar, (A, B) \notin att\},$$

we say that ArgLab *is an* L-stable argument labelling *iff* ArgLab *is a complete argument labelling where* $\texttt{undec}(\texttt{ArgLab}) \cap \texttt{SINKS}_{\text{AF}}$ *is* $\subseteq$*-minimal among the complete argument labellings of* AF*.* [16]

The L-stable argument labellings share similar properties to those of the semi-stable argument labellings [Sá and Alcântara, 2021a]:

- Every AF has at least one L-stable labelling.
- If AF has at least one stable labelling, then ArgLab is an L-stable argument labelling of AF if and only if ArgLab is a semi-stable argument labelling of AF if and only if ArgLab is a stable argument labelling of AF.

Example 3. *Looking back at Example 2, we had that both*

$$\texttt{ArgLab}_2 = (\{A_2, A_8\}, \{A_3, A_4, A_7\}, \{A_1, A_5, A_6, A_9\}) \textit{ and}$$

$$\texttt{ArgLab}_3 = (\{A_3, A_7, A_9\}, \{A_2, A_5, A_8\}, \{A_1, A_4, A_6\})$$

are semi-stable argument labellings, but only $\texttt{ArgLab2ConcLab}(\texttt{ArgLab}_3)$ *is an L-stable model of* P*. Given that* $\texttt{SINKS}_{\text{AF}_P} = \{A_6, A_7, A_8, A_9\}$*, we have that only* $\texttt{ArgLab}_3$ *is an L-stable argument labelling. Hence, the L-stable argument labellings of* AF_P *correspond to the L-stable models of* P *for this example.*

[15] In graph theory terminology, a sink is a node from which no edges originate.

[16] The definition we provide differs from the original one in [Sá and Alcântara, 2021a], but they proved that the property we use in our definition is exclusively satisfied by L-stable argument labellings.

It was proved in [Sá and Alcântara, 2021a] that the L-stable argument semantics indeed captures the L-stable program semantics.

Theorem 6. *([Sá and Alcântara, 2021a]) Let P be a logic program, AF_P be its associated argumentation framework and* `ArgLab` *be a complete argument labelling of AF_P. Then* `ArgLab` *is an L-stable argument labelling of AF_P if and only if* `ArgLab2ConcLab(ArgLab)` *is an L-stable model of P.*

Furthermore, we can devise new semantics based on the L-stable labellings in a similar spirit to the pre-eager and eager semantics:

Definition 17. *Let* `ArgLab` *be an argument labelling of AF.* `ArgLab` *is called:*

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

Now, given that the L-stable argument semantics captures the L-stable program semantics, we obtain as a corollary of Theorem 6 that

Corollary 7. *Let P be a logic program, AF_P be its associated argumentation framework and* `ArgLab` *be a complete argument labelling of AF_P. Then*

1. `ArgLab` *is an L-pre-eager argument labelling of AF_P if and only if* `ArgLab2ConcLab(ArgLab)` *is a pre-eager model of P.*
2. `ArgLab` *is the L-eager argument labelling of AF_P if and only if* `ArgLab2ConcLab(ArgLab)` *is the eager model of P.*

These results ensure the existence of argument semantics able to model every logic programming semantics for which [Caminada *et al.*, 2015b; Caminada *et al.*, 2022] could not find correspondence results. However, they bring new questions: what logic programming semantics could model the L-stable, L-pre-eager and L-eager argument semantics?

From the discussion of translation `AA2LP` in Section 3.2, we can ensure that such logic programming semantics can definitely be obtained, however

it is possible they have not been defined in the logic programming literature and will seem counter-intuitive. The reason is that we have one-to-one correspondence between arguments in a given argumentation framework AF and program rules in $\texttt{AA2LP}(\mathrm{AF})$. This means that only a subset of the program rules corresponds to default arguments, therefore minimizing undecided default arguments in AF corresponds to minimizing undecided conclusions for only a subset of the atoms (or rules) in the program.

Defining such semantics anew could prove to be a complicated task, since the concept of sink nodes (or sink atoms) is not as obvious in the context of logic programs. Fortunately, given an argumentation framework AF and its corresponding program $\texttt{AA2LP}(\mathrm{AF})$, it is rather simple to retrieve what atoms in $\texttt{AA2LP}(\mathrm{AF})$ correspond to the sinks in AF: since all arguments that are not sinks by definition attack one or more arguments, the sinks correspond to those atoms in $\texttt{AA2LP}(\mathrm{AF})$ for which their respective NAF-literals do not occur in the rules of $\texttt{AA2LP}(\mathrm{AF})$.[17] [18]

Definition 18. *Given AF, let* $P = \texttt{AA2LP}(\mathrm{AF})$ *and* $\texttt{Mod} = (T, F, U)$ *be a 3-valued interpretation of* P *w.r.t.* HB_P*. Further, given*

$$\texttt{SINKS}_P = \{c \in HB_P \mid \forall r \in P, \texttt{not}\ c \notin body(r)\},$$

we say that Mod *is*

- *an* L*-stable model *of* P *iff* Mod *is a 3-valued stable model where* $U \cap \texttt{SINKS}_P$ *is minimal (w.r.t.* $\subseteq$*) among all 3-valued stable models of* P*.*
- *an* L-pre-eager model *of* P *iff* Mod *is a 3-valued stable model where* $T \subseteq T_{l^*}$ *for each L***-stable model* $\texttt{Mod}_{l^*} = (T_{l^*}, F_{l^*}, U_{l^*})$ *of* P
- *the* L-eager model *of* P *iff* Mod *is the pre-eager model where* T *is* $\subseteq$*-maximal among all pre-eager models of* P

[17] As an abuse of notation, we allow ourselves to reuse the function name SINKS both to retrieve from an AF its sink arguments and to retrieve from a program its atoms corresponding to sink arguments of a corresponding AF. The subscript text should be enough to indicate what is the case.

[18] As an example, notice how $\texttt{not}\ A_6, \texttt{not}\ A_7, \texttt{not}\ A_8$ and $\texttt{not}\ A_9$ do not occur in the rules of P_{AF} in Example 1.

Now, since the 3-valued stable models of $\texttt{AA2LP}(\mathrm{AF})$ are precisely the complete argument labellings of AF (Theorem 1), we obtain the following results, which complement Corollary 2:

Corollary 8. *Let $AF = (Ar, att)$ be an argumentation framework, $\texttt{ArgLab} = (\texttt{in}, \texttt{out}, \texttt{undec})$ be a complete argument labelling of AF, and $\texttt{AA2LP}(\mathrm{AF}) = P_{\mathrm{AF}}$. Then:*

- *If* ArgLab *is* L-stable*, then* ArgLab *is an L^*-stable model of P_{AF}.*
- *If* ArgLab *is* L-pre-eager*, then* ArgLab *is a L-pre-eager model of P_{AF}.*
- *If* ArgLab *is* L-eager*, then* ArgLab *is the L-eager model of P_{AF}.*

At this point, once again, we must seek different argumentation semantics capable of expressing the new program semantics. This game may stabilize around a few semantics based on minimal undecided literals and arguments, but it might just as well go on indefinitely. We should also mind how the translation functions have a fundamental role in the results obtained.[19] One can say that AA2LP is a definitive translation, given the coincidental results it provides starting from argumentation semantics, but since LP2AA can only provide correspondence results, it leaves open the possibility that it may somehow be improved for the sake of capturing logic programming semantics.

4 On the Connection between Assumption-Based Argumentation and Logic Programming

Assumption-Based Argumentation (ABA) [Bondarenko *et al.*, 1997; Dung *et al.*, 2009] is a rule-based argumentation formalism where some special sentences called *assumptions*, which are true by default (that is, unless their contrary can be proved) and have a central role in semantics. Just like in abstract

[19] For instance, a different translation including singleton arguments for *undefined* atoms in a program (those that are not in the heads of any rules) was proposed in [Cramer and Saldanha, 2020]. Their translation also includes default arguments for negative literals (just like ours), causing the undefined atoms to be labelled as undec by the grounded AF semantics, whereas in our approach, they will be labelled out. As an effect, they find that the grounded AF semantics captures the *weak completion semantics* [Hölldobler and Kencana Ramli, 2009] for logic programs.

argumentation, theories in ABA are proposed as *frameworks* whose semantics are primarily retrieved in terms of *extensions* and *labellings*, but, at the same time, ABA frameworks share similar syntax to logic programs, allowing their semantics to be understood in terms of *interpretations* and *models* [Sá and Alcântara, 2019; Sá and Alcântara, 2021b]. The similarities may be illustrated with matching examples even before we formally introduce ABA frameworks.

Example 4. *Take into consideration the logic program* P[20] *as follows:*

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

Following [Bondarenko et al.*, 1997],* P *would be translated into the ABA framework* $ABA(P) = \mathcal{F}$[21] *below.*

$$\mathcal{F}: \quad \begin{array}{llll} a \leftarrow \beta & c \leftarrow \gamma & \overline{\alpha} = a & \overline{\beta} = b \\ b \leftarrow \alpha & d \leftarrow b, c & \overline{\gamma} = c & \overline{\delta} = d \end{array}$$

Notice how the rules in $\mathcal{F}$ *(depicted in the first two columns) mirror the rules of* P*, while the operator* $^-$ *captures the semantics of* `not` *. The language of* $\mathcal{F}$ *is a bit different from the language of* P*, but not by much. It consists of eight sentences, namely* $a, b, c, d, \alpha, \beta, \gamma, \delta$*, where* $\alpha, \beta, \gamma, \delta$ *respectively model* $\texttt{not}\ a, \texttt{not}\ b, \texttt{not}\ c, \texttt{not}\ d$ *as native elements in* $\mathcal{F}$*'s language. The sentences* $\alpha, \beta, \gamma, \delta$ *are called* assumptions *and each one has a unique contrary in our example:* a *is the contrary of* α *(which corresponds to* $\texttt{not}\ a$*),* b *is the contrary of* β *(which corresponds to* $\texttt{not}\ b$*), and so on.*

Moving on, if one follows [Caminada and Schulz, 2017] to translate $\mathcal{F}$ *into a logic program, one would obtain* P *as a result. However, this would also be the program obtained if the input is* $\mathcal{F}'$ *below, which corresponds to* $\mathcal{F}$ *except that* δ *is not in its language.*

$$\mathcal{F}': \quad \begin{array}{llll} a \leftarrow \beta & c \leftarrow \gamma & \overline{\alpha} = a & \overline{\beta} = b \\ b \leftarrow \alpha & d \leftarrow b, c & \overline{\gamma} = c & \end{array}$$

While the difference between $\mathcal{F}, \mathcal{F}'$ *may seem small, the semantics of these frameworks are significantly different just because* $\mathcal{F}$ *has one assumption more than* $\mathcal{F}'$*. And yet, the approach of [Caminada and Schulz, 2017] translates both* $\mathcal{F}$ *and* $\mathcal{F}'$ *to* P*.*

[20] This program is extracted from [Caminada and Schulz, 2017].

[21] At this time, to avoid the necessity of formal concepts, only the core syntactic elements of $\mathcal{F}$ are shown.

The ABA frameworks above illustrate the main challenge when translating from ABA to logic programming: the semantics of logic programs evaluate *all*[22] sentences in the language of the program (the atoms) while the mainstream semantics of ABA evaluate only *some* sentences in the language of each framework (the assumptions). Understanding this is the only difference explains how ABA can easily capture inference and semantics from logic programming and highlights the only obstacle for logic programming to capture inference and semantics from ABA. Fortunately, the difference will only be observed over semantics minimizing undecided sentences over ABA frameworks where the assumptions are not one-to-one related to non-assumption sentences as their contraries [Caminada and Schulz, 2017] and even then it can be fixed by an operation called *semantic projection* [Sá and Alcântara, 2021b].

4.1 Assumption-Based Argumentation: Syntax and Semantics

We briefly restate the core concepts of ABA frameworks [Bondarenko *et al.*, 1997; Dung *et al.*, 2007; Dung *et al.*, 2009] before we can proceed to discuss the translations and technical results regarding connections between Logic Programming and ABA.

Definition 19. *([Dung* et al.*, 2009]) An ABA framework is a tuple* $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, {}^{-}\rangle$ *where:*

- $\langle \mathcal{L}, \mathcal{R}\rangle$ *is a deductive system where* $\mathcal{L}$ *is a logical language and* $\mathcal{R}$ *is a set of inference rules on that language*
- $\mathcal{A} \subseteq \mathcal{L}$ *is a (non-empty) set, whose elements are referred to as assumptions*
- ${}^{-}$ *is a total mapping from* $\mathcal{A}$ *into* $\mathcal{L} \setminus \mathcal{A}$,[23] *where* $\overline{\alpha}$ *is called the contrary of* α.

[22] If one accounts of NAF-literals as sentences in the language of a program, the semantics would evaluate necessarily *half* the sentences, but traditionally the language of the program is defined in terms of its Herbrand Base. In either case, ABA frameworks are flexible regarding how many sentences in the language of a framework are assumptions.

[23] In the ABA literature it is common that the contrary relation ${}^{-}$ allows an assumption to be the contrary of another assumption or even for an assumption to have multiple contraries. In each case, ABA frameworks can be rewritten into an equivalent ABA framework where each assumption has a single contrary that is not an assumption [Caminada and Schulz, 2017].

For current purposes, we restrict ourselves to ABA frameworks that are *flat* [Bondarenko *et al.*, 1997], meaning that no assumption appears in the head of an inference rule. Furthermore, we follow [Dung *et al.*, 2009] in that each assumption has a unique contrary. This choice makes it easier to define some of the concepts we need.

Definition 20. *([Dung* et al.*, 2009]) Given a deductive system $\langle \mathcal{L}, \mathcal{R} \rangle$, and a set of assumptions $\mathcal{A} \subseteq \mathcal{L}$, an* argument *for $c \in \mathcal{L}$ (the* conclusion *or* claim*) supported by $S \subseteq \mathcal{A}$ is a tree with nodes labelled by formulas in $\mathcal{L}$ or by the special symbol $\top$ such that:*

- *the root is labelled c*
- *for every node N*
  - *if N is a leaf then N is labelled either by an assumption or by $\top$*
  - *if N is not a leaf and b is the label of N, then there exists an inference rule $b \leftarrow b_1, \ldots, b_m$ ($m \geq 0$) and either $m = 0$ and the child of N is labelled by $\top$, or $m > 0$ and N has m children, labelled by $b_1, \ldots, b_m$ respectively*
- *S is the set of all assumptions labelling the leaves.*

We say that a set of assumptions $\mathcal{A}sms \subseteq \mathcal{A}$ enables the construction of an argument A if A is supported by a subset of $\mathcal{A}sms$. A set of assumptions $\mathcal{A}sms_1$ is said to *attack* an assumption α iff $\mathcal{A}sms_1$ enables the construction of an argument for $\overline{\alpha}$. A set of assumptions $\mathcal{A}sms_1$ is said to attack a set of assumptions $\mathcal{A}sms_2$ iff $\mathcal{A}sms_1$ attacks some assumption $\alpha \in \mathcal{A}sms_2$.

The next step is to describe the various ABA semantics, which can be conveyed in the forms of *assumption extensions* [Bondarenko *et al.*, 1997], *assumption labellings* [Schultz and Toni, 2014; Schulz and Toni, 2017] or *interpretations and models* [Sá and Alcântara, 2019; Sá and Alcântara, 2021b] with corresponding translations between them. For the sake of uniformity in our presentation, we opt to prioritize the discussion of ABA semantics in the form of *assumption labellings*.

Definition 21. *([Schultz and Toni, 2014; Schulz and Toni, 2017]) An assumption labelling of $\mathcal{F}$ is a total function $\mathcal{L} : \mathcal{A} \rightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$.*

The same conventions we applied before to interpretations and argument labellings apply to assumption labellings, since we can perceive all of those concepts as the same function applied over different domains.

Definition 22. *([Schultz and Toni, 2014; Schulz and Toni, 2017]) An assumption labelling $\mathcal{L} = (\texttt{in}(\mathcal{L}), \texttt{out}(\mathcal{L}), \texttt{undec}(\mathcal{L}))$ of $\mathcal{F}$ is* complete *iff for each $\alpha \in \mathcal{A}$ it holds that:*

- *if $\alpha \in \texttt{in}(\mathcal{L})$, then each $\mathcal{S} \subseteq \mathcal{A}$ attacking α has some $\beta \in \mathcal{S}$ such that $\beta \in \texttt{out}(\mathcal{L})$;*
- *if $\alpha \in \texttt{out}(\mathcal{L})$, then there exists some $\mathcal{S} \subseteq \mathcal{A}$ attacking α such that $\mathcal{S} \subseteq \texttt{in}(\mathcal{L})$;*
- *if $\alpha \in \texttt{undec}(\mathcal{L})$, then (i) each $\mathcal{S} \subseteq \mathcal{A}$ attacking α is such that $\mathcal{S} \setminus \texttt{in}(\mathcal{L}) \neq \emptyset$ and (ii) there is at least one $\mathcal{S} \subseteq \mathcal{A}$ attacking α such that $\mathcal{S} \cap \texttt{out}(\mathcal{L}) = \emptyset$.*

Notice how Definition 22 closely resembles the definition of complete argument labellings (Definition 9).

Example 5. *The ABAF $\mathcal{F}$ (Example 4) has three complete assumption labellings: $\mathcal{L}_1 = (\{\ \}, \{\ \}, \{\alpha, \beta, \gamma, \delta\})$, $\mathcal{L}_2 = (\{\beta, \delta\}, \{\alpha\}, \{\gamma\})$, and $\mathcal{L}_3 = (\{\alpha\}, \{\beta\}, \{\gamma, \delta\})$. Just alike, $\mathcal{F}'$ (Example 4) has three complete assumption labellings: $\mathcal{L}'_1 = (\{\ \}, \{\ \}, \{\alpha, \beta, \gamma\})$, $\mathcal{L}'_2 = (\{\beta\}, \{\alpha\}, \{\gamma\})$, and $\mathcal{L}'_3 = (\{\alpha\}, \{\beta\}, \{\gamma\})$.*

The other labelling-based ABA semantics are defined as usual, as particular cases of the complete semantics:

Definition 23. *Let $\mathcal{L}$ be an assumption labelling of ABA framework $\mathcal{F} = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, {}^- \rangle$. $\mathcal{L}$ is called*[24]*:*

- *the* grounded assumption labelling *iff $\mathcal{L}$ is a complete assumption labelling where $\texttt{in}(\mathcal{L})$ is $\subseteq$-minimal among all complete assumption labellings of $\mathcal{F}$*

[24]The semi-stable and the eager semantics for ABA were originally defined in [Caminada *et al.*, 2015a] as extensions. We adapted the presentation of these semantics to favour uniformity with our previous definitions.

- *a* preferred assumption labelling *iff* $\mathcal{L}$ *is a complete assumption labelling where* $\mathtt{in}(\mathcal{L})$ *is* $\subseteq$*-maximal among all complete assumption labellings of* $\mathcal{F}$
- *a* stable assumption labelling *iff* $\mathcal{L}$ *is a complete assumption labelling where* $\mathtt{undec}(\mathcal{L}) = \emptyset$
- *a* semi-stable assumption labelling *iff* $\mathcal{L}$ *is a complete assumption labelling where* $\mathtt{undec}(\mathcal{L})$ *is* $\subseteq$*-minimal among all complete assumption labellings of* $\mathcal{F}$
- *a* pre-ideal assumption labelling *iff* $\mathcal{L}$ *is a complete assumption labelling where* $\mathtt{in}(\mathcal{L}) \subseteq \mathtt{in}(\mathcal{L}_{pr})$ *for each preferred assumption labelling of* $\mathcal{F}$
- *the* ideal assumption labelling *iff* $\mathcal{L}$ *is the complete assumption labelling where* $\mathtt{in}(\mathcal{L})$ *is* $\subseteq$*-maximal among all pre-ideal assumption labellings of* $\mathcal{F}$
- *a* pre-eager assumption labelling *iff* $\mathcal{L}$ *is a complete assumption labelling where* $\mathtt{in}(\mathcal{L}) \subseteq \mathtt{in}(\mathcal{L}_{sem})$ *for each semi-stable assumption labelling of* $\mathcal{F}$
- *the* eager assumption labelling *iff* $\mathcal{L}$ *is the complete assumption labelling where* $\mathtt{in}(\mathcal{L})$ *is* $\subseteq$*-maximal among all pre-eager assumption labellings of* $\mathcal{F}$

Example 6. *Among the complete assumption labellings of* $\mathcal{F}$ *(see Example 4 and Example 5), one can observe that:* $\mathcal{L}_1$ *is the grounded assumption labelling;* $\mathcal{L}_2, \mathcal{L}_3$ *are preferred assumption labellings; there are no stable assumption labellings;* $\mathcal{L}_2$ *is the only semi-stable assumption labelling;* $\mathcal{L}_1$ *is the only pre-ideal assumption labelling and therefore it is also ideal;* $\mathcal{L}_1, \mathcal{L}_2$ *are pre-eager and so* $\mathcal{L}_2$ *is the eager assumption labelling.*

4.2 From Logic Programming to Assumption-Based Argumentation

First, we will consider how logic programming semantics can be captured by ABA using a suitable translation. Intuitively, assumptions are sentences in the language of an ABA framework that are true unless their contrary can be

proved. This describes precisely the behaviour of NAF-literals in the setting of logic programs. This intuition explains the translation proposed by [Bondarenko *et al.*, 1997], which we introduce below.

Definition 24. *([Bondarenko* et al.*, 1997]) Let P be a program. The ABA framework associated to P is*

$$\texttt{LP2ABA}(P) = \mathcal{F}_P = \langle \mathcal{L}_P, \mathcal{R}_P, \mathcal{A}_P, {}^- \rangle$$

where[25]*:*

- $\mathcal{L}_P = HB_P \cup \{\texttt{not}\ a \mid a \in HB_P\}$*;*
- $\mathcal{R}_P = P$*;*
- $\mathcal{A}_P = \{\texttt{not}\ a \mid a \in HB_P\}$*;*
- $\overline{\texttt{not}\ a} = a$ *for each* $\texttt{not}\ a \in \mathcal{A}$*.*

We draw special attention to the fact that the NAF-literals of P become assumptions while the contrary relation $^-$ implements the NAF operator not . Also, the set of rules $\mathcal{R}_P$ coincides with P, hence the framework keeps all rules from the original program unchanged.

Example 7. *Take the LP P from Example 4, then* $\texttt{LP2ABA}(P) = \mathcal{F}_P = \langle \mathcal{L}_P, \mathcal{R}_P, \mathcal{A}_P, {}^- \rangle$ *with* $\mathcal{L}_P = \{a, b, c, d, \texttt{not}\ a, \texttt{not}\ b, \texttt{not}\ c, \texttt{not}\ d\}$*,* $\mathcal{R}_P = P$*,* $\mathcal{A}_P = \{\texttt{not}\ a, \texttt{not}\ b, \texttt{not}\ c, \texttt{not}\ d\}$ *and* $^-$ *such that* $\overline{\texttt{not}\ a} = a$*,* $\overline{\texttt{not}\ b} = b$*,* $\overline{\texttt{not}\ c} = c$ *and* $\overline{\texttt{not}\ d} = d$*. We adopt the same visual queue used in the introduction of this section for easier reference:*

$$\mathcal{F}_P: \quad \begin{array}{llll} a \leftarrow \texttt{not}\ b & c \leftarrow \texttt{not}\ c & \overline{\texttt{not}\ a} = a & \overline{\texttt{not}\ b} = b \\ b \leftarrow \texttt{not}\ a & d \leftarrow b, c & \overline{\texttt{not}\ c} = c & \overline{\texttt{not}\ d} = d \end{array}$$

Notice how $\mathcal{F}$ from Example 4 is the same as $\mathcal{F}_P$, only the assumptions were renamed as $\alpha, \beta, \gamma, \delta$.

ABA frameworks such as $\mathcal{F}_P$ are called *assumption spanning* in [Caminada and Schulz, 2017].

[25] Remember we treat programs as sets of rules and that HB_P is the set of all atoms appearing in P.

Definition 25. *([Caminada and Schulz, 2017]) Let $\mathcal{F} = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, ^- \rangle$ be an ABA framework. We say that $\mathcal{F}$ is* assumption-spanning *iff for each $x \in \mathcal{L} \setminus \mathcal{A}$ there exists some $\chi \in \mathcal{A}$ such that $\overline{\chi} = x$.*

The translation function LP2ABA was shown to model the semantics of any input logic programs for the *3-valued stable semantics* and all its particular cases listed in Definition 6 [Bondarenko *et al.*, 1997; Caminada and Schulz, 2017] using an auxiliary translation between assumption labellings and program interpretations:

Definition 26. *Let $\mathcal{L} = (\mathtt{in}, \mathtt{out}, \mathtt{undec})$ be an assumption labelling of $\mathcal{F}_P$. The program interpretation of P corresponding to $\mathcal{L}$ is $\mathtt{L2I}(\mathcal{L}) = (T, F, U)$ with $T = \{\overline{\alpha} \mid \alpha \in \mathtt{out}(\mathcal{L})\}$, $F = \{\overline{\alpha} \mid \alpha \in \mathtt{in}(\mathcal{L})\}$, and $U = HB_P \setminus (T \cup F)$.*

We are now ready to list the relevant results from [Bondarenko *et al.*, 1997; Caminada and Schulz, 2017]:

Theorem 9. *Let P be a logic program. Then $\mathcal{L}$ is a complete assumption labelling of $\mathcal{F}_P$ if and only if $\mathtt{L2I}(\mathcal{L})$ is a 3-valued stable model of P.*

While this result is not coincidental, the function L2I becomes bijective when used to map assumption labellings of $\mathcal{F}_P$ to corresponding program interpretations of P.[26] In this setting, $\mathtt{L2I}^{-1}$ is also a function, therefore the result in Theorem 9 is equivalent to a coincidence result. The same reasoning holds for the results below.

Theorem 10. *Let P be a logic program and $\mathcal{L}$ be a complete assumption labelling of $\mathcal{F}_P$. Then:*[27]

- *If $\mathcal{L}$ is* grounded*, then $\mathtt{L2I}(\mathcal{L})$ is the well-founded model of P.*
- *If $\mathcal{L}$ is* preferred*, then $\mathtt{L2I}(\mathcal{L})$ is a regular model of P.*
- *If $\mathcal{L}$ is* stable*, then $\mathtt{L2I}(\mathcal{L})$ is a stable model of P.*
- *If $\mathcal{L}$ is* semi-stable*, then $\mathtt{L2I}(\mathcal{L})$ is a L-stable model of P.*

[26] This is ensured by the fact that the contrary relation ^-_P obtained in $\mathcal{F}_P$ is necessarily bijective.

[27] The results concerning semi-stable, pre-ideal, pre-eager and eager ABA semantics are new, but their proofs may be dismissed because they follow directly from Theorem 9.

- *If* $\mathcal{L}$ *is* pre-ideal*, then* $\texttt{L2I}(\mathcal{L})$ *is a pre-ideal model of* P*.*
- *If* $\mathcal{L}$ *is* ideal*, then* $\texttt{L2I}(\mathcal{L})$ *is the ideal model of* P*.*
- *If* $\mathcal{L}$ *is* pre-eager*, then* $\texttt{L2I}(\mathcal{L})$ *is a pre-eager model of* P*.*
- *If* $\mathcal{L}$ *is* eager*, then* $\texttt{L2I}(\mathcal{L})$ *is the eager model of* P*.*

The results gathered ensure that flat ABA frameworks capture normal logic programs and their semantics, leaving it open whether logic programming also captures ABA frameworks and their semantics. We will delve into this question next.

4.3 From Assumption-Based Argumentation to Logic Programming

Moving from assumption-based argumentation to logic programming is trickier because logic programs have only one set of sentences (the atoms) and a matching number of corresponding NAF-literals, but assumptions and non-assumptions may appear in any proportion in an ABA framework. If we ignore this obstacle for a moment and only consider the semantics of assumptions, there are two straightforward ways to represent assumptions from an ABA framework in a corresponding logic program:

1. they can be mapped to the NAF-literals of the resulting program [Caminada and Schulz, 2017] or
2. they can be mapped to special atoms defined[28] by the negation of their contraries [Sá and Alcântara, 2021b].

The first option results from reversing the translation `LP2ABA` of [Bondarenko *et al.*, 1997]. This approach works perfectly for the class of *assumption-spanning* ABA frameworks [Caminada and Schulz, 2017], even so that all the results of Theorem 9 and of Theorem 10 are mirrored for them. This much ensures that

[28] If a program has only one rule for a given atom c such as $r : c \leftarrow body(r)$, that rule is understood as the *definition* of c in P and it may be read as "c if and only iff $body(r)$".

Theorem 11. Normal logic programs *are equivalent*[29] *to* assumption-spanning ABA frameworks.

When it comes to ABA frameworks in general, this approach is enough to ensure that logic programming captures most of ABA semantics, but it finds exceptions in the ABA semantics that minimize undecided assumptions (such as semi-stable, pre-eager and eager). The discrepancy arises because this translation is only injective for assumption-spanning ABA frameworks, not in general.

The second option [Sá and Alcântara, 2021b] matches the language of the input ABA framework (all sentences) to the language of the output logic program (all atoms). By doing so, the models of the corresponding program will evaluate all sentences from the input ABA framework, both assumptions and non-assumptions. This intuition led to the proposal of a model theory with interpretation and model semantics for ABA in [Sá and Alcântara, 2019]. Then, in order to retrieve assumption labellings, it is necessary to restrict the language of the models in the output program to the set of assumptions from the input ABA framework [Sá and Alcântara, 2021b]. This is achieved using an operation called *projection* in the logic programming literature. The results of [Sá and Alcântara, 2021b] ensure that

Theorem 12. *([Sá and Alcântara, 2021b])* Flat ABA frameworks *are equivalent to* normal logic programs *with* projection.

In the following, we will introduce the available translations and the results ensured by each one.

4.3.1 Mapping Assumptions to NAF-literals

We proceed to describe the approach and results obtained by [Caminada and Schulz, 2017].

Given an ABA framework $\mathcal{F} = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, ^- \rangle$ any translation from ABA to Logic Programming should be primarily based on $\mathcal{R}$. As such, the idea is to translate each rule in $\mathcal{R}$ to an associated Logic Programming rule. Further, in $\mathcal{F}$, if α is an assumption whose contrary is the sentence a, we will find that α can be labelled `in` as long as all arguments for a have at least one assumption in

[29] Here, by saying two argumentation systems are *equivalent*, we mean that the sets of problems that can be expressed and solved in both systems coincide.

their support that is out. Intuitively, this means that α can be proved based on a labelling as long as a cannot. Therefore, not must be used to implement the contrary relation $^-$. In this first translation, each occurrence of α is replaced with "not a" or, using the contrary relation, "not $\overline{\alpha}$".

Definition 27. *([Caminada and Schulz, 2017]) Let $\mathcal{F} = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, ^- \rangle$ be an ABA framework, the program corresponding to $\mathcal{F}$ is*

$$\texttt{ABA2LP}_{\texttt{CS17}}(\mathcal{F}) = P_{\mathcal{F}} = \{a \leftarrow b_1, \ldots, b_n, \texttt{not } \overline{\gamma_1}, \ldots, \texttt{not } \overline{\gamma_m} \mid a \leftarrow b_1, \ldots, b_n, \gamma_1, \ldots, \gamma_m \in \mathcal{R}\}.$$

Example 8. *Remember $\mathcal{F}, \mathcal{F}'$ and P from Example 4. Both ABA frameworks present $a \leftarrow \beta$ as one of its inference rules. In each case, this rule will be translated to $a \leftarrow$ not b, given that $\overline{\beta} = b$. As we previously mentioned, we will find that $\texttt{ABA2LP}_{\texttt{CS17}}(\mathcal{F}) = \texttt{ABA2LP}_{\texttt{CS17}}(\mathcal{F}') = P$.*

As before, we must retrieve assumption labellings from the models of the resulting program, which is done by a specialized function:

Definition 28. *([Caminada and Schulz, 2017]) Let $I = (T, F, U)$ be a model of $P_{\mathcal{F}}$, the assumption labelling of $\mathcal{F}$ corresponding to I is*

$$\texttt{I2L}(I) = (\{\alpha \in \mathcal{A} \mid \overline{\alpha} \in F\}, \{\alpha \in \mathcal{A} \mid \overline{\alpha} \in T\}, \{\alpha \in \mathcal{A} \mid \overline{\alpha} \in U\}).$$

Notice how $\mathcal{A}$ is required for the computation of the function I2L. In the next section we will build over the relevance of this parameter to motivate a different translation from ABA to LP. For now, we opt to leave this parameter implicit here (as in [Caminada and Schulz, 2017]), since it can be retrieved from $\mathcal{F}$.

Theorem 13. *[Caminada and Schulz, 2017] Let $\mathcal{F}$ be an ABA framework. Then I is a 3-valued stable model of $P_{\mathcal{F}}$ if and only if $\texttt{I2L}(I)$ is a complete assumption labelling of $\mathcal{F}$.*

Differently from before, the function I2L cannot be inverted in this setting.

Example 9. *Take $\mathcal{F}'$ from Example 4, we find that $\texttt{ABA2LP}_{\texttt{CS17}}(\mathcal{F}') = P$. Now we have that $I = (\{a\}, \{b, d\}, \{c\})$ is a 3-valued stable model of P and that $\texttt{I2L}(I) = (\{\beta\}, \{\alpha\}, \{\gamma\})$ is a complete assumption labelling of $\mathcal{F}'$. However, $\texttt{L2I}(\texttt{I2L}(I)) = (\{a\}, \{b\}, \{c, d\})$, which is not the same as I. The difference follows from the fact that there is no $\delta \in \mathcal{A}'$ such that $\overline{\delta} = d$, so L2I leaves d undecided.*

Therefore, the result in Theorem 13 ensures a one-to-one correspondence, but it is not as strong as a coincidence result. This situation is quite similar to the one we found in Section 3.3, when trying to model logic programming semantics using abstract argumentation frameworks. In fact, the correspondence and exception results we will find in this scenario mimic those we found then.

On that matter, we find a similar discrepancy when trying to minimize undecided atoms in program models versus undecided assumptions in matching assumption labellings.

Example 10. *Once again, retrieve $\mathcal{F}, \mathcal{F}'$ from Example 4 and remember that $\texttt{ABA2LP}_{\texttt{CS17}}(\mathcal{F}) = \texttt{ABA2LP}_{\texttt{CS17}}(\mathcal{F}') = P$. Continuing from Example 5 and Example 6, we have that $\mathcal{F}$ has a single semi-stable assumption labelling $\mathcal{L}_2 = (\{\beta, \delta\}, \{\alpha\}, \{\gamma\})$, whereas $\mathcal{F}'$ has two: $\mathcal{L}'_2 = (\{\beta\}, \{\alpha\}, \{\gamma\})$ and $\mathcal{L}'_3 = (\{\alpha\}, \{\beta\}, \{\gamma\})$. As expected, since $\texttt{LP2ABA}(P) = \mathcal{F}$, the semantics of P mirrors that of $\mathcal{F}$: P has a single L-stable model, namely $I_2 = (\{a\}, \{b, d\}, \{c\})$. On the other hand, the L-stable models of $\texttt{ABA2LP}(\mathcal{F}') = P$ do not correspond one-to-one to the semi-stable assumption labellings of $\mathcal{F}'$. Further, P has two pre-eager models, namely $I_1 = (\{\,\}, \{\,\}, \{a, b, c, d\})$ and I_2, so I_2 is the eager model of P, whereas the only pre-eager (and therefore eager) assumption labelling of $\mathcal{F}'$ is $\mathcal{L}'_1 = (\{\,\}, \{\,\}, \{\alpha, \beta, \gamma\})$. This means that the pre-eager and eager semantics of P do not correspond to the pre-eager and eager assumption labellings of $\mathcal{F}'$.*

Overall, the results found in [Caminada and Schulz, 2017] can be summarized (and extended[30]) as follows:

Theorem 14. *Let $\mathcal{F}$ be an ABA framework, $\texttt{ABA2LP}_{\texttt{CS17}}(\mathcal{F}) = P_{\mathcal{F}}$ be its associated logic program and I be a 3-valued stable model of $P_{\mathcal{F}}$. Then:*

1. *I is well-founded if and only if $\texttt{I2L}(I)$ is grounded*
2. *I is regular if and only if $\texttt{I2L}(I)$ is preferred*
3. *I is stable if and only if $\texttt{I2L}(I)$ is stable*
4. *I is pre-ideal if and only if $\texttt{I2L}(I)$ is pre-ideal*

[30]The pre-ideal, pre-eager and eager ABA semantics are not mentioned in [Caminada and Schulz, 2017]. The result for pre-ideal follows as corollary of their proof about preferred semantics. The results for pre-eager and eager are justified in Example 10.

5. *I is ideal if and only if* $\texttt{I2L}(I)$ *is ideal*
6. *I is L-stable,* $\texttt{I2L}(I)$ *may not be semi-stable (and vice-versa)*
7. *I is pre-eager,* $\texttt{I2L}(I)$ *may not be pre-eager (and vice-versa)*
8. *I is eager,* $\texttt{I2L}(I)$ *may not be eager (and vice-versa)*

We highlight that for *assumption-spanning* ABA frameworks, I2L becomes bijective and the missing correspondence results are restored.

Theorem 15. *Let $\mathcal{F}$ be an ABA framework, $\texttt{ABA2LP}_{\texttt{CS17}}(\mathcal{F}) = P_{\mathcal{F}}$ be its associated logic program and I be a 3-valued stable model of $P_{\mathcal{F}}$. If $\mathcal{F}$ is* assumption-spanning*, then:*

1. *I is L-stable if and only if* $\texttt{I2L}(I)$ *is semi-stable*
2. *I is pre-eager if and only if* $\texttt{I2L}(I)$ *is pre-eager*
3. *I is eager if and only if* $\texttt{I2L}(I)$ *is eager*

The results gathered ensure the conclusion in Theorem 11.

4.3.2 ABA as Logic Programming with Projection

The translation of [Caminada and Schulz, 2017] has a couple of drawbacks: (i) it is not injective over ABA frameworks in general and (ii) if the input ABA framework has more non-assumptions than assumptions, the output program may have more NAF-literals than the number of assumptions in the input. Further, the function I2L requires knowledge of what the set of assumptions $\mathcal{A}$ is in the input ABA framework. Ideally, if we want logic programs to model ABA frameworks, the step where one retrieves assumption labellings from models of its corresponding program should *not* depend on knowledge of the input. This led [Sá and Alcântara, 2021b] to propose a different translation, mapping assumptions to positive literals instead of NAF-literals, thus avoiding those issues.

Given $\mathcal{F} = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, ^- \rangle$, the translation of [Sá and Alcântara, 2021b] produces a program P where:

1. $HB_P = \mathcal{L}$, which means that the assumptions of $\mathcal{F}$ now correspond to a subset of HB_P. This allows assumptions to be modelled in P regardless of the proportion between $\mathcal{A}$ and $\mathcal{L}$.

2. $P \supset \mathcal{R}$, so the inference rules from $\mathcal{F}$ are kept unaltered in the resulting program. Since `not` is alien to the native syntax of ABA, we can ensure that `not` does not appear in $\mathcal{R}$. A complimentary set of program rules using `not` is added to implement the contrary relation $^-$, ensuring that the translation is injective over ABA frameworks in general and that the set of assumptions from $\mathcal{F}$ can be retrieved syntactically from the complimentary rules.

Definition 29. *([Sá and Alcântara, 2021b]) Let $\mathcal{F} = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, ^- \rangle$ be an ABA framework, the program corresponding to $\mathcal{F}$ is*

$$\texttt{ABA2LP}_{\texttt{SA21}}(\mathcal{F}) = P'_{\mathcal{F}} = \mathcal{R} \cup \{\alpha \leftarrow \texttt{not}\ \overline{\alpha} \mid \alpha \in \mathcal{A}\}.$$

Example 11. *Recover $\mathcal{F}, \mathcal{F}'$ from Example 4. We have that $\texttt{ABA2LP}_{\texttt{SA21}}(\mathcal{F}) = P'_{\mathcal{F}} = \mathcal{R} \cup \{\alpha \leftarrow \texttt{not}\ a,\ \beta \leftarrow \texttt{not}\ b,\ \gamma \leftarrow \texttt{not}\ c,\ \delta \leftarrow \texttt{not}\ d\}$ and $\texttt{ABA2LP}_{\texttt{SA21}}(\mathcal{F}') = P'_{\mathcal{F}'} = P'_{\mathcal{F}} \setminus \{\delta \leftarrow \texttt{not}\ d\}$.*

Given an $\mathcal{F}$ and its corresponding $P'_{\mathcal{F}}$, $\mathcal{A}$ can be retrieved from $P'_{\mathcal{F}}$: each $\alpha \in \mathcal{A}$ appears as the head of a single rule r_α in $P'_{\mathcal{F}}$ for which $body^-(r_\alpha) \neq \emptyset$, a condition only met by rules in $P'_{\mathcal{F}} \setminus \mathcal{R}$.

The results based on $\texttt{ABA2LP}_{\texttt{SA21}}$ include mappings regarding assumption labellings as well as *model semantics* for ABA frameworks, which were introduced in [Sá and Alcântara, 2019]. The general idea of their work consists of treating ABA frameworks just like logic programs, which includes the computation of semantics through a division operator and steps similar to what we introduced in Section 2. As such, interpretations and models evaluate all sentences in $\mathcal{L}$, not only the assumptions.

For the sake of uniformity and because ABA model semantics are not mainstream, we will only discuss in depth the results involving assumption labellings. Concerning ABA model semantics, to discuss it briefly, [Sá and Alcântara, 2021b] proved that the 3-valued stable models of $\texttt{ABA2LP}_{\texttt{SA21}}(\mathcal{F})$ *coincide* with the complete models of $\mathcal{F}$, ensuring that:

Theorem 16. *Flat-ABA frameworks under model semantics are equivalent to ABA-programs*[31].

[31] The fragment of ABA-programs describes all the programs that can be obtained as output of $\texttt{ABA2LP}_{\texttt{SA21}}$. See [Sá and Alcântara, 2021b] for their definition and properties.

The discussion of model semantics for ABA in [Sá and Alcântara, 2019] included comparative results to the assumption labellings of [Schulz and Toni, 2017]. The mapping from ABA models to corresponding ABA assumption labellings is performed by an operation they called tuple projection (or simply *projection*), which can be applied to interpretations, models and all labellings alike.

Definition 30. *Let S be a set and $T = (S_1, S_2, \ldots, S_k)$ be a tuple of sets. The projection of S on T is*

$$\sigma_S(T) = (S_1 \cap S, S_2 \cap S, \ldots, S_k \cap S).$$

Definition 31. *([Sá and Alcântara, 2021b]) Let $I = (T, F, U)$ be a model of $P'_{\mathcal{F}}$, the assumption labelling of $\mathcal{F} = \langle \mathcal{L}, \mathcal{R}, \mathcal{A}, {}^{-} \rangle$ corresponding to I is obtained by projecting $\mathcal{A}$ on I,[32] i.e., $\texttt{I2L}_{\texttt{SA21}}(I) = \sigma_{\mathcal{A}}(I)$.*

We are now ready to list the main results from [Sá and Alcântara, 2021b]:

Theorem 17. *([Sá and Alcântara, 2021b]) Let $\mathcal{F}$ be an ABA framework. Then I is a 3-valued stable model of $P_{\mathcal{F}}$ if and only if $\texttt{I2L}_{\texttt{SA21}}(I) = \sigma_{\mathcal{A}}(I)$ is a complete assumption labelling of $\mathcal{F}$.*

Example 12. *Recover $\mathcal{F}, \mathcal{F}'$ from Example 4, for which $\texttt{ABA2LP}_{\texttt{SA21}}(\mathcal{F}) = P'_{\mathcal{F}} = \mathcal{R} \cup \{\alpha \leftarrow \texttt{not } a,\ \beta \leftarrow \texttt{not } b,\ \gamma \leftarrow \texttt{not } c,\ \delta \leftarrow \texttt{not } d\}$ and $\texttt{ABA2LP}_{\texttt{SA21}}(\mathcal{F}) = P'_{\mathcal{F}'} = P'_{\mathcal{F}} \setminus \{\delta \leftarrow \texttt{not } d\}$ (Example 11).*

- *The 3-valued stable models of $P'_{\mathcal{F}}$ are $I_1 = (\{\ \}, \{\ \}, \{a, b, c, d, \alpha, \beta, \gamma, \delta\})$, $I_2 = (\{a, \beta, \delta\}, \{b, d, \alpha\}, \{c, \gamma\})$ and $I_3 = (\{b, \alpha\}, \{a, \beta\}, \{c, d, \gamma, \delta\})$. Now we can obtain $\texttt{I2L}_{\texttt{SA21}}(I_1) = \sigma_{\mathcal{A}}(I_1) = (\{\ \}, \{\ \}, \{\alpha, \beta, \gamma, \delta\})$, $\texttt{I2L}_{\texttt{SA21}}(I_2) = \sigma_{\mathcal{A}}(I_2) = (\{\beta, \delta\}, \{\alpha\}, \{\gamma\})$, and $\texttt{I2L}_{\texttt{SA21}}(I_3) = \sigma_{\mathcal{A}}(I_3) = (\{\alpha\}, \{\beta\}, \{\gamma, \delta\})$, which correspond precisely to the complete assumption labellings of $\mathcal{F}$, as seen in Example 5.*
- *The 3-valued stable models of $P'_{\mathcal{F}'}$ are $I'_1 = (\{\}, \{\}, \{a, b, c, d, \alpha, \beta, \gamma\})$, $I'_2 = (\{a, \beta\}, \{b, d, \alpha\}, \{c, \gamma\})$ and $I'_3 = (\{b, \alpha\}, \{a, \beta\}, \{c, d, \gamma\})$. Now, we can obtain $\texttt{I2L}_{\texttt{SA21}}(I'_1) = \sigma_{\mathcal{A}}(I'_1) = (\{\ \}, \{\ \}, \{\alpha, \beta, \gamma\})$, $\texttt{I2L}_{\texttt{SA21}}(I'_2) = \sigma_{\mathcal{A}}(I'_2) = (\{\beta\}, \{\alpha\}, \{\gamma\})$, $\texttt{I2L}_{\texttt{SA21}}(I'_3) = \sigma_{\mathcal{A}}(I'_3) = (\{\alpha\}, \{\beta\}, \{\gamma\})$, which correspond precisely to the complete assumption labellings of $\mathcal{F}'$, as seen in Example 5.*

[32] Remember that $\mathcal{A}$ can be retrieved directly from $P'_{\mathcal{F}}$ as $\mathcal{A} = \{head(r) \in P'_{\mathcal{F}} \mid body^{-}(r) \neq \emptyset\}$.

When restricted to complete assumption labellings and 3-valued stable models of corresponding pair of $\mathcal{F}$ and $P'_{\mathcal{F}}$, $\texttt{I2L}_{\texttt{SA21}}$ becomes bijective and admits an inverse $\texttt{I2L}^{-1}_{\texttt{SA21}}$. For this reason, similarly to what we observed in the discussion of Theorem 9 in Section 4.2, the result in Theorem 17 is equivalent to a coincidence result. The same reasoning holds for the results below.

Theorem 18. *Let $\mathcal{F}$ be an ABA framework, $\texttt{ABA2LP}_{\texttt{SA21}}(\mathcal{F}) = P'_{\mathcal{F}}$ be its associated logic program and I be a 3-valued stable model of $P'_{\mathcal{F}}$. Then:*

1. *I is well-founded if and only if $\texttt{I2L}(I)$ is grounded*
2. *I is regular if and only if $\texttt{I2L}(I)$ is preferred*
3. *I is stable if and only if $\texttt{I2L}(I)$ is stable*
4. *I is pre-ideal if and only if $\texttt{I2L}(I)$ is pre-ideal*
5. *I is ideal if and only if $\texttt{I2L}(I)$ is ideal*
6. *I is L-stable if and only if $\texttt{I2L}(I)$ is semi-stable*
7. *I is pre-eager if and only if $\texttt{I2L}(I)$ is pre-eager*
8. *I is eager if and only if $\texttt{I2L}(I)$ is eager*

The results gathered ensure that flat ABA framework and their semantics are captured by ABA-programs with the projection of assumptions. Combined with Theorem 16 ([Sá and Alcântara, 2021b]) and the results in Section 4.2, we obtain Theorem 11.

5 Equivalence for enhanced frameworks

In this section, we will briefly discuss some notable argumentation systems of which the connection to logic programming have been studied, namely Abstract Dialectical Frameworks (ADF) [Brewka and Woltran, 2010; Brewka *et al.*, 2018] and Argumentation Frameworks with Sets of Attacking Arguments (SETAF) [Nielsen and Parsons, 2006; Bikakis *et al.*, 2021]. We consider these extensions of Dung's AFs to be representative of the diverse enhanced frameworks proposed in the literature and have chosen them because of their relations to each other and to Logic Programming have been well studied. Other

systems, for which not as much research has been conducted, will be discussed in the next section.

Differently from what we did in the previous sections, we will not fully introduce the definitions for systems we discuss here. Instead, we will focus only on their syntax to introduce appropriate translations and list the results concerning the preservation of semantics in each case.

5.1 Abstract Dialectical Frameworks

Abstract Dialectical Frameworks (ADFs) [Brewka and Woltran, 2010; Brewka *et al.*, 2013] were proposed to treat arguments (called *statements* there) as abstract and atomic entities. The connections between ADFs and logic programming start from the original definitions in [Brewka and Woltran, 2010], where the authors adopted the standard terminology and even the notion of *reduction* from logic programming (Definition 3) to obtain model semantics for ADFs. Their connection to logic programming was studied in depth by [Strass, 2013] and [Alcântara *et al.*, 2019].

Similarly to Dung's AFs, an ADF can be perceived as a directed graph of which the nodes represent statements which can get *accepted* or not. But differently from Dung's AFs, where the edges represent conflicts between arguments, here the links represent the more general notion of *dependencies*: the status (accepted/not accepted) of a node s depends only on the status of its parents ($par(s)$), i.e., the nodes with a direct link to s. For simplicity, we will restrict ourselves to finite ADFs:

Definition 32. *([Brewka and Woltran, 2010]) An abstract dialectical framework is a tuple $D = (S, L, C)$ where*

- *S is a finite set of statements (positions, nodes);*
- *$L \subseteq S \times S$ is a set of links, and $\forall s \in S$, $par(s) = \{t \in S \mid (t, s) \in L\}$;*
- *$C = \{C_s \mid s \in S\}$ is a set of total functions $C_s : 2^{par(s)} \to \{\mathbf{t}, \mathbf{f}\}$, one for each statement s. C_s is called the acceptance condition of s.*

The function C_s is intended to determine the acceptance status of a statement s, which only depends on the status of its parent nodes $par(s)$. Intuitively, s will be accepted if there exists $R \subseteq par(s)$ for which $C_s(R) = \mathbf{t}$,

which means that every statement in R is accepted while each statement in $par(s) - R$ is not accepted.

The semantics of ADFs are primarily given by interpretations and models over the set of sentences. The definitions introduced by [Brewka and Woltran, 2010; Brewka *et al.*, 2013] to obtain the *complete models* of an ADF [Brewka *et al.*, 2013] closely resemble the ones we presented in Section 2 to obtain the 3-valued stable models of a program. As such, the complete models of an ADF D are obtained as the least fixed points of a reduction operator Γ_D [Brewka *et al.*, 2013] which was adapted from logic programming to work with ADFs and was shown to always have a least model [Brewka and Woltran, 2010]. We opt not to introduce this operator here for the sake of brevity and simplicity in our presentation of ADFs.

Definition 33. *Let $D = (S, L, C^{\varphi})$ be an ADF and v be a* 3-valued interpretation *over S.*[33] *Then v is a* complete model *of D iff $v = \Gamma_D(v)$.*

As usual, some of the mainstream semantics for an ADF are obtained as special cases of the complete semantics. Most of the semantics below were originally proposed in [Brewka *et al.*, 2013], except for the L-stable ADF semantics, which was proposed by [Alcântara *et al.*, 2019]. The text of this definition is adapted to our needs, to make its presentation uniform with our previous definitions of semantics.

Definition 34. *Let $D = (S, L, C)$ be an ADF, and v a model of D. Then*

- *v is the* grounded model *of D iff v is the complete model of D for which $\mathbf{t}(v) = \{s \in S \mid v(s) = \mathbf{t}\}$ is $\subseteq$-minimal among complete models of D.*
- *v is a* preferred model *of D iff v is a complete model of D for which $\mathbf{t}(v) = \{s \in S \mid v(s) = \mathbf{t}\}$ is $\subseteq$-maximal among complete models of D.*
- *v is a* stable model *of D iff v is a 2-valued complete model of D.*
- *v is a* L-stable model *of D iff v is a complete model of D for which $\mathbf{u}(v) = \{s \in S \mid v(s) = \mathbf{u}\}$ is $\subseteq$-minimal among complete models of D.*

[33] Similarly to interpretations of logic programs, given an ADF $D = (S, L, C)$, a 3-valued interpretation (or simply interpretation) over S is a mapping $v : S \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ that assigns one of the truth values true ($\mathbf{t}$), false ($\mathbf{f}$) or unknown ($\mathbf{u}$), to each statement in S.

These ADF semantics have been shown to capture corresponding logic programming semantics (resp. 3-valued stable models, well-founded, regular, stable and L-stable) in *Attacking Abstract Dialectical Frameworks* (ADF$^+$s) [Alcântara *et al.*, 2019], a fragment of ADFs in which the unique relation involving statements is the attack relation.[34]

Definition 35. *An Attacking Abstract Dialectical Framework (ADF$^+$), is an ADF (S, L, C) such that every $(r, s) \in L$ is an* attacking link *[Brewka and Woltran, 2010], i.e., there is no $R \subseteq par(s)$ for which $C_s(R) = \mathbf{f}$ and $C_s(R \cup \{r\}) = \mathbf{t}$. This means that for every $s \in S$ and every $M \subseteq par(s)$, if $C_s(M) = \mathbf{t}$, then for every $M' \subseteq M$, we have $C_s(M') = \mathbf{t}$.*

[Alcântara *et al.*, 2019] introduced a translation $\texttt{LP2ADF}^+$ inspired by the work of [Caminada *et al.*, 2015b], on the basis of which they proved coincidental correspondence results between normal logic programs and ADF$^+$ for all ADF semantics in Definition 34. Here, we will resort to Definition 13, which is used (indirectly) as part of the translation of [Alcântara *et al.*, 2019].[35]

Definition 36. *Let P be a program, for each $a \in HB_P$, let*

$$\texttt{Sup}_P(a) = \{\texttt{Vul}(a) \mid A \textit{ is an argument with } \texttt{Conc}(A) = a\}.$$

Then, the ADF$^+$ associated to P is

$$\texttt{LP2ADF}^+(P) = D_P = (HB_P, L_P, C_P)$$

where:

- $L_P = \{(b, a) \mid b \in B \textit{ for some } B \in \texttt{Sup}_P(a)\}$*;*
- *For each $a \in HB_P$, $C_a = \left\{ B' \subseteq par(a) \setminus B \;\middle|\; B \in \texttt{Sup}_P(a) \right\}$.*

[34] The class of ADF$^+$, sometimes also referred to as support-free ADFs [Dvořák *et al.*, 2023b], is also a subclass of Bipolar Abstract Dialectical Frameworks [Brewka and Woltran, 2010].

[35] Definition 25 of [Alcântara *et al.*, 2019] speaks of *substatements*, which correspond to proofs in P and also to the arguments of our Definition 13. The *support* of a substatement A in their work is defined exactly as $\texttt{Vul}(A)$ from Definition 13. Finally, the notion of support is extended to each atom a as the set of supports from substatements with $\texttt{Conc}(A) = a$, which is the criterion we use here.

The intuition for C_a in the definition above is that if an interpretation of D_P accepts all $b \in B = \texttt{Vul}(A)$ for each argument A from P where $\texttt{Conc}(A) = a$, then in order to be a model of D_P, it must not accept a. Hence, the acceptance condition C_a for each $a \in HB_P$ requires that for each set of vulnerabilities $\texttt{Vul}(A) \in \texttt{Sup}_P(A)$, at least one $b \in par(a)$ is not accepted .

Based on the translation $\texttt{LP2ADF}^+$, [Alcântara *et al.*, 2019] proved that the 3-valued stable model semantics for normal logic programs is equivalent to the complete model semantics of ADF for the class of ADF$^+$.

Theorem 19. *([Alcântara* et al.*, 2019]) Let P be a program and D_P be its corresponding ADF$^+$. Then v is a 3-valued stable model of P iff v is a complete model of D_P.*

Until the study of [Alcântara and Sá, 2019; Alcântara *et al.*, 2019], it was unclear if any ADF semantics could capture the 3-valued semantics for normal logic programs. Theorem 20 ensures that the translation from logic programs to ADF in Definition 36 guarantees the equivalence between any semantics based on 3-valued stable models (at the logic program side) with any semantics based on complete models (at the ADF side). We highlight that, according to Theorem 19, the models of P and $\texttt{LP2ADF}^+(P)$ *coincide*, so the following results are immediate:

Corollary 20. *Let P be a program and $D_P = (A, L, C)$ be its corresponding ADF$^+$. We have*

- *v is a well-founded model of P iff v is a grounded model of D_P.*
- *v is a regular model of P iff v is a preferred model of D_P.*
- *v is a stable model of P iff v is a stable model of D_P.*
- *v is an L-stable model of P iff v is an L-stable model of D_P.*

Results such as the above can be extended to appropriate definitions of pre-ideal, ideal, pre-eager and eager semantics for ADF. Further, because $\texttt{LP2ADF}^+$ is injective and the models of P and D_P coincide for all programs P, $\texttt{LP2ADF}^+$ must be bijective and, therefore, it admits inversion. These results lead to the conclusion that

Theorem 21. Normal logic programs *are equivalent to* attacking abstract dialectical frameworks.

Given that ADF$^+$ is a fragment of ADFs in general, this means that the more general family of ADFs semantically subsume normal logic programs. On the other hand, previously, [Strass, 2013] showed a direct translation from ADFs to normal logic programs for which the program $\texttt{ADF}^+\texttt{2LP}(D)$, corresponding to an ADF D, would model the semantics of Definition 33 and Definition 34 (except for L-stable, which had not been defined) following the same correspondences we listed in Theorem 19 and Corollary 20. This means that normal logic programs semantically subsume ADFs (in general). Combining the results, we can gather that the two systems, NLPs and ADFs (in general), are inherently equivalent.

5.2 Argumentation Frameworks with Sets of Attacking Arguments

A *framework with sets of attacking arguments* (SETAF) [Nielsen and Parsons, 2006; Bikakis *et al.*, 2021] is an extension of Dung's AFs (in the context of finite AFs) to allow joint attacks on arguments. Intuitively, the need for joint attacks arises from situations where an argument may not be enough to defeat another on its own, but two or more arguments, together, might suffice.

Definition 37. *([Nielsen and Parsons, 2006]) A Framework with Sets of Attacking Arguments (*SETAF *for short) is a pair $\mathfrak{A} = (Ar, att)$, in which Ar is a finite set of arguments and $att \subseteq (2^{Ar} - \{\emptyset\}) \times Ar$.*

The attack relation att is such that if $(\mathcal{B}, a) \in att$, there is no $\mathcal{B}' \subset \mathcal{B}$ such that $(\mathcal{B}', a) \in att$, i.e., $\mathcal{B}$ is a minimal set (w.r.t. $\subseteq$) attacking a. We write $att(a) = \{\mathcal{B} \subseteq Ar \mid (\mathcal{B}, a) \in att\}$ to retrieve the attackers of a.

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

The semantics for SETAFs are generalisations of the corresponding semantics for AFs [Nielsen and Parsons, 2006] and can be defined equivalently in terms of extensions or labellings [Flouris and Bikakis, 2019; Caminada *et al.*, 2024]. For our convenience, we will adhere to the presentation of labelling-based semantics as proposed in [Flouris and Bikakis, 2019].

Definition 38. *Let* $\mathfrak{A} = (Ar, att)$ *be a* SETAF. *A labelling is a function* $\mathcal{L} : Ar \to \{\texttt{in}, \texttt{out}, \texttt{undec}\}$. *A labelling is* complete *iff for each* $a \in Ar$,

- *If* $\mathcal{L}(a) = \texttt{in}$, *then for each* $\mathcal{B} \in att(a)$, *there is* $b \in \mathcal{B}$ *s.t.* $\mathcal{L}(b) = \texttt{out}$
- *If* $\mathcal{L}(a) = \texttt{out}$, *then there is a* $\mathcal{B} \in att(a)$ *s.t.* $\mathcal{L}(b) = \texttt{in}$ *for all* $b \in \mathcal{B}$
- *If* $\mathcal{L}(a) = \texttt{undec}$, *then there is a* $\mathcal{B} \in att(a)$ *s.t.* $\mathcal{L}(b) \neq \texttt{out}$ *for each* $b \in \mathcal{B}$, *and for each* $\mathcal{B} \in att(a)$, *it holds* $\mathcal{L}(b) \neq \texttt{in}$ *for some* $b \in \mathcal{B}$.

As usual, we may use as shorthand $\texttt{in}(\mathcal{L}) = \{a \in Ar \mid \mathcal{L}(a) = \texttt{in}\}$, $\texttt{out}(\mathcal{L}) = \{a \in Ar \mid \mathcal{L}(a) = \texttt{out}\}$, $\texttt{undec}(\mathcal{L}) = \{a \in Ar \mid \mathcal{L}(a) = \texttt{undec}\}$. Also as before, a labelling defines a partition of the set of arguments, so $\mathcal{L}$ can be written as a triple $(\texttt{in}(\mathcal{L}), \texttt{out}(\mathcal{L}), \texttt{undec}(\mathcal{L}))$. Intuitively, an argument labelled `in` is explicitly accepted; an argument labelled `out` is explicitly rejected; and one labelled `undec` is left undecided, i.e., it is neither accepted nor rejected. We can now describe the remaining SETAF semantics studied in [Alcântara *et al.*, 2023]:

Definition 39. *([Flouris and Bikakis, 2019]) Let* $\mathfrak{A} = (Ar, att)$ *be a* SETAF. *A complete labelling* $\mathcal{L}$ *is called*

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

Let us consider the following example:

Example 13. *Consider the* SETAF $\mathfrak{A} = (Ar, att)$ *below:*
Concerning the semantics of $\mathfrak{A}$, *we have*

- *Complete labellings:* $\mathcal{L}_1 = (\emptyset, \emptyset, \{a, b, c, d, e\})$, $\mathcal{L}_2 = (\{a\}, \{b\}, \{c, d, e\})$ *and* $\mathcal{L}_3 = (\{b\}, \{a, e\}, \{c, d\})$;
- *Grounded labellings:* $\mathcal{L}_1 = (\emptyset, \emptyset, \{a, b, c, d, e\})$;

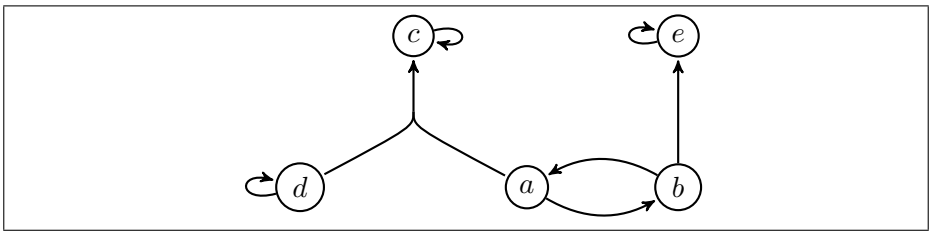


Figure 2: A SETAF $\mathfrak{A}$. Joint attacks are drawn as arrows with two or more origin nodes as, for instance, $\{d, a\}$ jointly attack argument c.

- *Preferred labellings:* $\mathcal{L}_2 = (\{a\}, \{b\}, \{c, d, e\})$ *and* $\mathcal{L}_3 = (\{b\}, \{a, e\}, \{c, d\})$*;*
- *Stable labellings: none;*
- *Semi-stable labellings:* $\mathcal{L}_3 = (\{b\}, \{a, e\}, \{c, d\})$.

The semantics of SETAF were studied in connection with Dung's AFs in [Flouris and Bikakis, 2019], with ADFs in [Polberg, 2016], [Dvořák *et al.*, 2023b] and [Alcântara and Sá, 2021] (ADF$^+$'s[36]), with ABA and CAF in [König *et al.*, 2022] and in connection with logic programming in [König *et al.*, 2022] and [Alcântara *et al.*, 2023]. For an overview of SETAFs and their properties, we refer to [Bikakis *et al.*, 2021; Caminada *et al.*, 2024]. In what follows, we will focus on the works of [Alcântara *et al.*, 2023][37], which concerns the relation between SETAF and logic programming. The authors of both works devised the same translation functions between SETAF and LP. We will start with their translation from NLP to SETAF, which was shown in [Alcântara *et al.*, 2023] to guarantee the equivalence between various kinds of NLPs models and SETAFs labellings, including the complete labellings, well-founded models and grounded labellings, regular models and preferred labellings, stable models and stable labellings, L-stable models and semi-stable labellings. Following [Alcântara *et al.*, 2023], this translation is built upon the translation from NLP to AF of [Caminada *et al.*, 2015b].

[36]Before that, [Alcântara *et al.*, 2019] showed the translation from SETAF to ADF proposed by [Polberg, 2016] necessarily returns an ADF$^+$.

[37]The translations appearing in [Alcântara *et al.*, 2023] were also proposed by [König *et al.*, 2022], but they did not explore the connections between semantics of SETAF and logic programs beyond that. In contrast, [Alcântara *et al.*, 2023] also offer translations between models and labellings and prove numerous semantic correspondence results.

Definition 40. *Let P be a program and $\mathfrak{S}_P$ be the set of all non-default arguments constructed from P following Definition 13, the SETAF corresponding to P is* $\texttt{LP2SETAF}(P) = \mathfrak{A}_P = (Ar_P, att_P)$ *with*

- $Ar_P = \{\texttt{Conc}(s) \mid s \in \mathfrak{S}_P\}$
- $att_P = \{(\mathcal{B}, a) \mid \mathcal{B}$ *is a* $\subseteq$*-minimal set s.t. for each* $A \in \mathfrak{S}_P$ *there is some* $b \in \mathcal{B} \cap \texttt{Vul}(A)\}$.

A counterpart translation from SETAFs to NLPs is also considered:

Definition 41. *Let* $\mathfrak{A} = (Ar, att)$ *be a* SETAF*, the logic program corresponding to* $\mathfrak{A}$ *is* $\texttt{SETAF2LP}(\mathfrak{A}) = P_{\mathfrak{A}}$ *with*

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

where, $\mathcal{V}_{a \in Ar} = \{V \subseteq Ar \mid V$ *is a* $\subseteq$*-minimal set s.t. for each* $\mathcal{B} \in att(a)$ *there is some* $b \in \mathcal{B} \cap V\}$.

Example 14. *Recall the* SETAF $\mathfrak{A}$ *of Example 13 (depicted in Fig 2). The program corresponding to* $\mathfrak{A}$ *is* $\texttt{SETAF2LP}(\mathfrak{A}) = P_{\mathfrak{A}}$ *comprising the rules:*

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

Further, $\texttt{LP2SETAF}(\texttt{SETAF2LP}(\mathfrak{A})) = \mathfrak{A}$. As proven by [Alcântara *et al.*, 2023], this holds for every SETAF $\mathfrak{A}$. This also suffices to show that

Theorem 22. *([Alcântara* et al.*, 2023]) Let* $\mathfrak{A}$ *be a* SETAF*. Then* $\mathcal{L}$ *is a complete argument labelling of* $\mathfrak{A}$ *if and only if* $\mathcal{L}$ *is 3-valued stable model of* $P_{\mathfrak{A}}$.

Theorem 22 portrays a *coincidence* between the labellings of $\mathfrak{A}$ and the models of its corresponding program $P_{\mathfrak{A}}$. As before, other results immediately follow:

Corollary 23. *Let* $\mathfrak{A}$ *be a* SETAF *and* $\mathcal{L} = (\texttt{in}, \texttt{out}, \texttt{undec})$ *be a complete argument labelling of* $\mathfrak{A}$*. Then:*

- $\mathcal{L}$ *is* grounded *if and only if* $\mathcal{L}$ *is the well-founded model of* $P_{\mathfrak{A}}$.

- $\mathcal{L}$ *is* preferred *if and only if* $\mathcal{L}$ *is a regular model of* $P_{\mathfrak{A}}$.
- $\mathcal{L}$ *is* stable *if and only if* $\mathcal{L}$ *is a stable model of* $P_{\mathfrak{A}}$.
- $\mathcal{L}$ *is* semi-stable *if and only if* $\mathcal{L}$ *is a L-stable model of* $P_{\mathfrak{A}}$.

On the other hand, we may obtain $\texttt{SETAF2LP}(\texttt{LP2SETAF}(P)) \neq P$ for some cases of a program P. This will only be observed if there is some $c \in HB_P$ for which there is no rule $r \in P$ with $head(r) = c$. In that case, there will be no argument for c in $\mathfrak{S}_P$, therefore c will not be in Ar_P of $\texttt{LP2SETAF}(P) = (Ar_P, att_P)$. Fortunately, this problem is easy to solve: if $\{r \in P \mid head(r) = c\} = \emptyset$, we observe that $I(c) = \mathbf{f}$ in every model I of P, so the extra atoms can be ignored [Alcântara *et al.*, 2023]. In what follows, allow us to use the shorthand

$$HB'_P = \Big\{c \in HB_P \;\Big|\; \{r \in P \mid head(r) = c\} \neq \emptyset\Big\}.$$

Theorem 24. *([Alcântara* et al.*, 2023]) Let P be a program. Then $I = (T, F, U)$ is 3-valued stable model of P if and only if $I' = (T, F \cap HB'_P, U)$ is a complete argument labelling of $\mathfrak{A}_P$.*

While the result in Theorem 24 is not coincidental, the models of P and labellings of $\mathfrak{A}_P$ are one-to-one related because $HB_P \setminus HB'_P$ can be retrieved from P. Results concerning the particular cases of the complete and 3-valued stable models immediately follow:

Corollary 25. *Let P be a program, $I = (T, U, F)$ be a 3-valued stable model of P and $I' = (T, F \cap HB'_P, U)$. Then:*

- $I = (T, F, U)$ *is* well-founded *if and only if I' is the grounded labelling of $\mathfrak{A}_P$.*
- $I = (T, F, U)$ *is* regular *if and only if I' is a preferred labelling of $\mathfrak{A}_P$.*
- $I = (T, F, U)$ *is* stable *if and only if I' is a stable labelling of $\mathfrak{A}_P$.*
- $I = (T, F, U)$ *is* L-stable *if and only if I' is a L-stable labelling of $\mathfrak{A}_P$.*

The results in Corollary 25 and Corollary 23 could be extended to appropriate definitions of pre-ideal, ideal, pre-eager and eager semantics for SETAF. Together, the results we presented lead to the conclusion that

Theorem 26. Normal logic programs *are equivalent to* frameworks with sets of attacking arguments.[38]

6 Implementing Argumentation with Answer-Set Programming

In this section we discuss how logic programming can be used to implement solvers for argumentation formalisms. That is, we consider reduction-based implementations based on answer-set programming. Answer-set programming is based on the stable model semantics of logic programming, but, compared to normal logic programs considered so far, also allows for variables to denote collections of rules, for disjunction in rule heads, and several other extensions that deal with, for instance, constraints, aggregates, and minimization/maximization. Due to the availability of efficient solvers, answer-set programming is nowadays a successful declarative programming approach for NP-hard problems. We will distinguish two kinds of ASP-implementation:

1. In what one may call compiler-style implementations one uses a program that given a specific semantics transforms an argumentation framework into an equivalent ground, i.e., without any variables, logic program. That is, each argumentation framework has to be compiled into a logic program which can be solved by ASP-solvers in order to compute the extensions or labellings. This is the same schema as in typical SAT-based approaches.

2. In what we call query-based implementations or (in the context of argumentation) ASPARTIX-style implementations one encodes the argumentation framework as an input database, i.e., as facts of the LP, which is independent of the semantics and reasoning task. This input database can then be combined with fixed encodings of semantics and reasoning tasks in order to solve specific reasoning tasks.

Compiler-style implementations date back to [Nieves *et al.*, 2008] and can also be found in a more recent paper by Sakama and Rienstra (2017). The

[38] The only exceptions to this equivalence result are programs whose syntax include irrelevant atoms, which are those $c \in HB_P \setminus HB'_P$. However, all occurrences of `not` c and each $r \in P$ for which $c \in body(r)$ can be removed from P without prejudice to the semantics of the remaining atoms.

query-based approach has been used for Dung style abstract argumentation by [Wakaki and Nitta, 2008] and the ASPARTIX system [Egly *et al.*, 2010], and similar approaches have also been used for richer abstract argumentation formalisms [Dvořák *et al.*, 2015; Ellmauthaler and Strass, 2014; Dvořák *et al.*, 2018] and structured argumentation [Lehtonen *et al.*, 2021a; Lehtonen *et al.*, 2021b]. Toni and Sergot (2011) provide a survey on the earlier works on ASP for abstract argumentation while later surveys on implementations techniques for argumentation by Charwat *et al.* (2015) and Cerutti *et al.* (2018) focus on ASPARTIX-style implementations.

In the remainder of this section we start with a brief introduction to answer-set programming, then discuss compiler style implementations for abstract argumentation frameworks, and finally discuss ASPARTIX-style approaches for different kinds of argumentation formalisms.

6.1 Answer-Set Programming

We give an overview of the syntax and semantics of disjunctive logic programs under the answer-set semantics [Gelfond and Lifschitz, 1991], generalizing or Definitions from Section 2. We fix a countable set $\mathcal{U}$ of *(domain) elements*, also called *constants*. An *atom* is an expression $p(t_1, \ldots, t_n)$, where p is a *predicate* of arity $n \geq 0$ and each t_i is either a variable or an element from $\mathcal{U}$. An atom is *ground* if it is free of variables. $B_\mathcal{U}$ denotes the set of all ground atoms over $\mathcal{U}$. A *(disjunctive) rule* r is of the form

$$x_1 \vee \cdots \vee x_k \leftarrow y_1, \ldots, y_n, \texttt{not } z_1, \ldots, \texttt{not } z_m \tag{1}$$

with $k \geq 0$, $n \geq 0$, $m \geq 0$, and in each rule at least one of k, n, m is non zero. $x_i, \ldots, x_k, y_1, \ldots, y_n, z_1, \ldots, z_m$ are atoms, and "`not` " stands for *negation as failure*. The *head* of r is the set $head(r) = \{x_1, \ldots, x_k\}$ and the *body* of r is $body(r) = \{y_1, \ldots, y_n, \texttt{not } z_1, \ldots, \texttt{not } z_m\}$. Furthermore, $body^+(r)r = \{b_1, \ldots, b_k\}$ and $body^-(r) = \{b_{k+1}, \ldots, b_m\}$. A rule r is *normal* if $k \leq 1$ and a *constraint* if $k = 0$. A rule r is *safe* if each variable in r occurs in $body^+(r)$. A rule r is *ground* if no variable occurs in r. A *fact* is a ground rule without disjunction and empty body. An *(input) database* is a set of facts. A program is a finite set of disjunctive rules. If each rule in a program is normal (resp. ground), we call the program normal (resp. ground).

For any program π, let the *Herbrand Literal Base* U_π be the set of all constants appearing in π (if no constant appears in π, we add an arbitrary

constant to U_π). Moreover, $Gr(\pi)$ is the set of rules obtained by applying, to each rule $r \in \pi$, all possible substitutions σ from the variables in r to elements of U_π. We call $Ground(\pi, U_P)$ the grounding of π, and write $Gr(\pi)$ as a shorthand for $Ground(\pi, U_P)$. The semantics of a (non-ground) program π is defined via its grounding $Gr(\pi)$.

An *interpretation* $I \subseteq B_\mathcal{U}$ *satisfies* a ground rule r iff $head(r) \cap I \neq \emptyset$ whenever $body^+(r) \subseteq I$ and $body^-(r) \cap I = \emptyset$. I satisfies a ground program π, if each $r \in \pi$ is satisfied by I. A non-ground rule r (resp., a program π) is satisfied by an interpretation I iff I satisfies all groundings of r (resp., $Gr(\pi)$). $I \subseteq B_\mathcal{U}$ is an *answer-set* of π iff it is a subset-minimal set satisfying the *Gelfond-Lifschitz reduct* $\pi^I = \{head(r) \leftarrow body^+(r) \mid I \cap body^-(r) = \emptyset, r \in Gr(\pi)\}$. For a program π, we denote the set of its answer-sets by $\mathrm{AS}(\pi)$.

Modern ASP-solvers offer additional language features. Among them we make use of the *conditional literal* [Gebser *et al.*, 2015]. In the head of a disjunctive rule literals may have conditions, e.g. consider the head of rule "$\mathbf{p}(X) : \mathbf{q}(X) \leftarrow$". Intuitively, this represents a head of disjunctions of atoms $\mathbf{p}(a)$ where also $\mathbf{q}(a)$ is true. As well rules might have conditions in their body, e.g. consider the body of rule "$\leftarrow \mathbf{p}(X) : \mathbf{q}(X)$", which intuitively represents a conjunction of atoms $\mathbf{p}(a)$ where also $\mathbf{q}(a)$ is true. Notice, that when using conditions in the head of a rule we have a disjunction of atoms while when using conditions in the body of a rule we have a conjunction of atoms.

6.2 Compiler-Style ASP Encodings

In this section we follow [Sakama and Rienstra, 2017] in order to illustrate the compiler style approach towards abstract argumentation. This approach is based on the labelling characterisation of complete semantics and thus works with three predicates $\mathbf{in}(x)$, $\mathbf{out}(x)$, $\mathbf{undec}(x)$ representing that an argument has the corresponding label.

Example 15. *As our running example for this section we will use the argumentation framework* $AF_{run} = (\{a, b, c\}, \{(a, b), (b, a), (b, c), (c, c)\})$ *as depicted in Figure 3. We have the three admissible sets* $\emptyset$, $\{a\}$, $\{b\}$, *with* $\{a\}$, $\{b\}$ *being the preferred extensions and* $\{b\}$ *being the only stable extension.*

We start with the basic encoding of the `in` and `out` labels that applies to all semantics. That is, given an argumentation framework $AF = (Ar, att)$ we

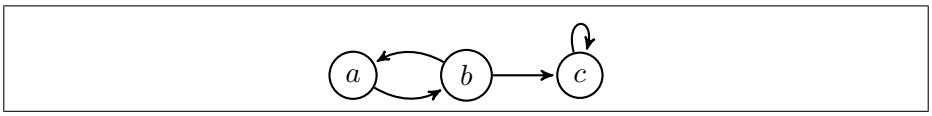


Figure 3: Illustration of our running example AF_{run} for Section 6.2.

define $\pi_{basic}(AF)$ as follows:

$$\pi_{basic}(AF) = \{\mathbf{out}(x) \leftarrow \mathbf{in}(y) \mid (y,x) \in att\} \cup$$
$$\{\mathbf{in}(x) \leftarrow \mathbf{out}(y_1), \ldots, \mathbf{out}(y_k) \mid x \in Ar, x^- = \{y_1, \ldots, y_n\}\} \cup$$
$$\{\leftarrow \mathbf{in}(x), \texttt{not } \mathbf{out}(y) \mid (y,x) \in att\} \cup$$
$$\{\leftarrow \mathbf{out}(x), \texttt{not } \mathbf{in}(y_1), \ldots, \texttt{not } \mathbf{in}(y_k) \mid x \in Ar, x^- = \{y_1, \ldots, y_n\}\}$$

$\pi_{basic}(AF)$ ensures the basic properties of labellings that if an argument is labelled in the all its neighbours are labelled out and that if an argument is labelled out it has an attacker that is labelled in. Notice that, so far, there is no restriction on the number of different labels an argument can have and no requirement to be labelled at all.

Example 16. *When considering our running example AF_{run} and apply π_{basic} we obtain the following ground logic program $\pi_{basic}(AF_{run})$:*

$$\begin{array}{ll}
\mathbf{out}(b) \leftarrow \mathbf{in}(a). & \leftarrow \mathbf{in}(b), \texttt{not } \mathbf{out}(a). \\
\mathbf{out}(a) \leftarrow \mathbf{in}(b). & \leftarrow \mathbf{in}(a), \texttt{not } \mathbf{out}(b). \\
\mathbf{out}(c) \leftarrow \mathbf{in}(b). & \leftarrow \mathbf{in}(c), \texttt{not } \mathbf{out}(b). \\
\mathbf{out}(c) \leftarrow \mathbf{in}(c). & \leftarrow \mathbf{in}(c), \texttt{not } \mathbf{out}(c). \\
\mathbf{in}(a) \leftarrow \mathbf{out}(b). & \leftarrow \mathbf{out}(a), \texttt{not } \mathbf{in}(b). \\
\mathbf{in}(b) \leftarrow \mathbf{out}(a). & \leftarrow \mathbf{out}(b), \texttt{not } \mathbf{in}(a). \\
\mathbf{in}(c) \leftarrow \mathbf{out}(b), \mathbf{out}(c). & \leftarrow \mathbf{out}(c), \texttt{not } \mathbf{in}(b), \texttt{not } \mathbf{in}(c).
\end{array}$$

Stable Semantics. Let us now consider stable semantics. For stable semantics we only need in and out labels. In the following we extend the basic encoding by two constraints: (i) each argument must be labelled in or out and (ii) no argument can be labelled both in and out.

$$\begin{aligned}\pi_{st}(AF) =&\{\mathbf{in}(x) \vee \mathbf{out}(x) \leftarrow \mid x \in Ar\} \cup \\ &\{\leftarrow \mathbf{in}(x), \mathbf{out}(x) \mid x \in Ar\}\end{aligned}$$

Example 17. *When considering our running example AF_{run} we obtain the following ground logic program $\pi_{st}(AF_{run})$:*

$$\begin{array}{ll}\mathbf{in}(a) \vee \mathbf{out}(a) \leftarrow . & \leftarrow \mathbf{in}(a), \mathbf{out}(a). \\ \mathbf{in}(b) \vee \mathbf{out}(b) \leftarrow . & \leftarrow \mathbf{in}(b), \mathbf{out}(b). \\ \mathbf{in}(c) \vee \mathbf{out}(c) \leftarrow . & \leftarrow \mathbf{in}(c), \mathbf{out}(c).\end{array}$$

Theorem 27. *([Sakama and Rienstra, 2017]) For every argumentation framework $AF = (Ar, att)$, the stable labellings of AF are in one-to-one correspondence with the stable models of the logic program $\pi_{basic}(AF) \cup \pi_{st}(AF)$.*

The attentive reader may have noticed that $\pi_{basic}(AF) \cup \pi_{stb}(AF)$ uses disjunction in some rule heads and is thus not a normal logic program. However, often normal logic programs are preferable over disjunctive ones (e.g., due to limitations of solvers or computational advantages) and thus one might ask whether we can avoid disjunctive rules here. Indeed the addition for stable semantics can be reformulated as follows in order to obtain normal logic programs.

$$\begin{aligned}\pi_{stb'}(AF) =&\{\mathbf{in}(x) \leftarrow \texttt{not } \mathbf{out}(x) \mid x \in Ar\} \cup \\ &\{\mathbf{out}(x) \leftarrow \texttt{not } \mathbf{in}(x) \mid x \in Ar\}\end{aligned}$$

Example 18. *When considering our running example AF_{run} we obtain the following ground logic program $\pi_{st'}(AF_{run})$:*

$$\begin{array}{ll}\mathbf{in}(a) \leftarrow \texttt{not } \mathbf{out}(a). & \mathbf{out}(a) \leftarrow \texttt{not } \mathbf{in}(a). \\ \mathbf{in}(b) \leftarrow \texttt{not } \mathbf{out}(b). & \mathbf{out}(b) \leftarrow \texttt{not } \mathbf{in}(b). \\ \mathbf{in}(c) \leftarrow \texttt{not } \mathbf{out}(c). & \mathbf{out}(c) \leftarrow \texttt{not } \mathbf{in}(c).\end{array}$$

Theorem 28. *([Sakama and Rienstra, 2017]) For every argumentation framework $AF = (Ar, att)$, the stable labellings of AF are in one-to-one correspondence with the stable models of the normal logic program $\pi_{basic}(AF) \cup \pi_{stb'}(AF)$.*

Complete Semantics. Now we turn our attention to complete semantics. Here we have to deal with all three labels. Again we have two types of constraints: (i) each argument must be labelled `in`, `out`, or `undec`. (ii) no argument can be labelled with two of the labels.

$$\begin{aligned}\pi_{co}(AF) =& \{\mathbf{in}(x) \vee \mathbf{out}(x) \vee \mathbf{undec}(x) \leftarrow \mid x \in Ar\} \cup \\ & \{\leftarrow \mathbf{in}(x), \mathbf{out}(x) \mid x \in Ar\} \cup \\ & \{\leftarrow \mathbf{in}(x), \mathbf{undec}(x) \mid x \in Ar\} \cup \\ & \{\leftarrow \mathbf{out}(x), \mathbf{undec}(x) \mid x \in Ar\}\end{aligned}$$

Example 19. *When considering our running example AF_{run} we obtain the following ground logic program $\pi_{st}(AF_{run})$:*

$$\begin{array}{ll}\mathbf{in}(a) \vee \mathbf{out}(a) \vee \mathbf{undec}(a) \leftarrow . & \leftarrow \mathbf{in}(a), \mathbf{undec}(a). \\ \mathbf{in}(b) \vee \mathbf{out}(b) \vee \mathbf{undec}(b) \leftarrow . & \leftarrow \mathbf{in}(b), \mathbf{undec}(b). \\ \mathbf{in}(c) \vee \mathbf{out}(c) \vee \mathbf{undec}(c) \leftarrow . & \leftarrow \mathbf{in}(c), \mathbf{undec}(c). \\ \leftarrow \mathbf{in}(a), \mathbf{out}(a). & \leftarrow \mathbf{out}(a), \mathbf{undec}(a). \\ \leftarrow \mathbf{in}(b), \mathbf{out}(b). & \leftarrow \mathbf{out}(b), \mathbf{undec}(b). \\ \leftarrow \mathbf{in}(c), \mathbf{out}(c). & \leftarrow \mathbf{out}(c), \mathbf{undec}(c).\end{array}$$

Theorem 29. *([Sakama and Rienstra, 2017]) For every argumentation framework $AF = (Ar, att)$, the complete labellings of AF are in one-to-one correspondence with the stable models of the logic program $\pi_{basic}(AF) \cup \pi_{co}(AF)$.*

Again we can modify $\pi_{com}(AF)$ in order to obtain a normal logic program.

$$\begin{aligned}\pi_{co'}(AF) =& \{\mathbf{in}(x) \leftarrow \texttt{not } \mathbf{out}(x), \texttt{not } \mathbf{undec}(x) \mid x \in Ar\} \cup \\ & \{\mathbf{out}(x) \leftarrow \texttt{not } \mathbf{in}(x), \texttt{not } \mathbf{undec}(x) \mid x \in Ar\} \cup \\ & \{\mathbf{undec}(x) \leftarrow \texttt{not } \mathbf{in}(x), \texttt{not } \mathbf{out}(x) \mid x \in Ar\}\end{aligned}$$

Example 20. *When considering our running example AF_{run} we obtain the following ground logic program $\pi_{co'}(AF_{run})$:*

$$\begin{aligned}
\mathbf{in}(a) &\leftarrow \texttt{not}\ \mathbf{out}(a), \texttt{not}\ \mathbf{undec}(a). & \mathbf{out}(c) &\leftarrow \texttt{not}\ \mathbf{in}(c), \texttt{not}\ \mathbf{undec}(c).\\
\mathbf{in}(b) &\leftarrow \texttt{not}\ \mathbf{out}(b), \texttt{not}\ \mathbf{undec}(b). & \mathbf{undec}(a) &\leftarrow \texttt{not}\ \mathbf{in}(a), \texttt{not}\ \mathbf{out}(a).\\
\mathbf{in}(c) &\leftarrow \texttt{not}\ \mathbf{out}(c), \texttt{not}\ \mathbf{undec}(c). & \mathbf{undec}(b) &\leftarrow \texttt{not}\ \mathbf{in}(b), \texttt{not}\ \mathbf{out}(b).\\
\mathbf{out}(a) &\leftarrow \texttt{not}\ \mathbf{in}(a), \texttt{not}\ \mathbf{undec}(a). & \mathbf{undec}(c) &\leftarrow \texttt{not}\ \mathbf{in}(c), \texttt{not}\ \mathbf{out}(c).\\
\mathbf{out}(b) &\leftarrow \texttt{not}\ \mathbf{in}(b), \texttt{not}\ \mathbf{undec}(b).
\end{aligned}$$

Theorem 30. *([Sakama and Rienstra, 2017]) For every argumentation framework $AF = (Ar, att)$, the complete labellings of AF are in one-to-one correspondence with the stable models of the normal logic program $\pi_{basic}(AF) \cup \pi_{co'}(AF)$.*

Preferred Semantics. Finally, for preferred semantics we introduce three new predicates $\mathbf{IN}(x), \mathbf{OUT}(x), \mathbf{UNDEC}(x)$ that will correspond to the actual labels of the preferred labelling. The main idea is that in the program we allow that an argument can satisfy both $\mathbf{in}(x)$ and $\mathbf{out}(x)$ and then map the resulting stable model to an argument labelling as follows. An argument a is: (i) labelled `in` if $\mathbf{in}(a)$ holds and $\mathbf{out}(x)$ does not hold; (ii) labelled `out` if $\mathbf{out}(a)$ holds and $\mathbf{in}(x)$ does not hold, (iii) labelled `undec` if $\mathbf{in}(a)$ and $\mathbf{out}(x)$ hold. We use the new predicates $\mathbf{IN}(x), \mathbf{OUT}(x), \mathbf{UNDEC}(x)$ to compute the argument labels.

$$\begin{aligned}
\pi_{pr}(AF) = &\{\mathbf{in}(x) \vee \mathbf{out}(x) \leftarrow \mid x \in Ar\} \cup\\
&\{\mathbf{IN}(x) \leftarrow \mathbf{in}(x), \texttt{not}\ \mathbf{out}(x) \mid x \in Ar\} \cup\\
&\{\mathbf{OUT}(x) \leftarrow \texttt{not}\ \mathbf{in}(x), \mathbf{out}(x) \mid x \in Ar\} \cup\\
&\{\mathbf{UNDEC}(x) \leftarrow \mathbf{in}(x), \mathbf{out}(x) \mid x \in Ar\}
\end{aligned}$$

Example 21. *When considering our running example AF_{run} we obtain the following ground logic program $\pi_{pr}(AF_{run})$:*

$$\begin{aligned}
\mathbf{in}(a) \vee \mathbf{out}(a) \leftarrow .& \qquad \mathbf{OUT}(a) \leftarrow \texttt{not } \mathbf{in}(a), \mathbf{out}(a).\\
\mathbf{in}(b) \vee \mathbf{out}(b) \leftarrow .& \qquad \mathbf{OUT}(b) \leftarrow \texttt{not } \mathbf{in}(b), \mathbf{out}(b).\\
\mathbf{in}(c) \vee \mathbf{out}(c) \leftarrow .& \qquad \mathbf{OUT}(c) \leftarrow \texttt{not } \mathbf{in}(c), \mathbf{out}(c).\\
\mathbf{IN}(a) \leftarrow \mathbf{in}(a), \texttt{not } \mathbf{out}(a).& \qquad \mathbf{UNDEC}(a) \leftarrow \mathbf{in}(a), \mathbf{out}(a).\\
\mathbf{IN}(b) \leftarrow \mathbf{in}(b), \texttt{not } \mathbf{out}(b).& \qquad \mathbf{UNDEC}(b) \leftarrow \mathbf{in}(b), \mathbf{out}(b).\\
\mathbf{IN}(c) \leftarrow \mathbf{in}(c), \texttt{not } \mathbf{out}(c).& \qquad \mathbf{UNDEC}(c) \leftarrow \mathbf{in}(c), \mathbf{out}(c).
\end{aligned}$$

Theorem 31. *([Sakama and Rienstra, 2017]) For every argumentation framework $AF = (Ar, att)$, the preferred labellings of AF are in one-to-one correspondence with the stable models of the logic program $\pi_{basic}(AF) \cup \pi_{pr}(AF)$.*

Again we have a disjunctive rule in our logic program. This time, in contrast with the previous encodings, we cannot replace these rules with normal rules (this is due to complexity results for preferred semantics [Sakama and Rienstra, 2017] [39]).

6.3 ASPARTIX-style ASP-Encodings

We will now discuss ASPARTIX-style implementations of argumentation formalisms. That is, we follow a query based approach where we (a) have queries (that do not depend on the actual framework) encoding semantics and reasoning tasks, which are combined with (b) an input database that encodes the actual argumentation framework. First, we define an input format that encodes argumentation frameworks of the considered argumentation formalism as input database, an ASP program consisting solely of facts. This input format is independent of the actual semantics and reasoning task one aims to solve. Then for each semantics of interest one provides an ASP encoding, an ASP query consisting of non-ground rules that, when combined with an input database, results answer-sets that are in one-to-one correspondence with extensions (or labellings) of the argumentation framework represented by the input database. Moreover, in order to implement specific reasoning tasks one can add modules encoding theses reasoning tasks. In order to solve a reasoning task (for

[39] This based on the fact that complexity of reasoning with preferred semantics is located on the second level of the polynomial hierarchy while reasoning with admissible, complete and stable semantics is located on the first level of the polynomial hierarchy [Dvořák and Dunne, 2018].

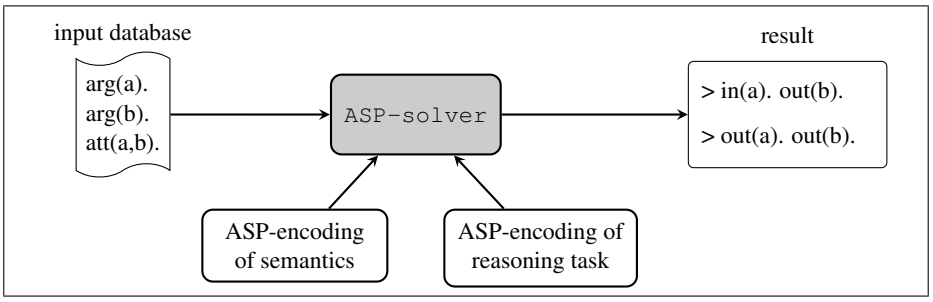


Figure 4: Basic workflow of ASPARTIX like implementations

a given framework under a given semantics) one then combines the input encoding of the framework, the encoding of the semantics, and the encoding of the reasoning task and runs a state of the art ASP-solver on that to obtain the corresponding answer-sets. This answer-sets can then be easily interpreted in order to answer the reasoning task. This standard workflow is illustrated in Figure 4. What we described so far is the standard workflow when using one-shot solving via a single ASP encoding to solve reasoning problems. However, in particular in systems optimized towards performance, also multi-shot methods and incremental approaches have been exploited for argumentation systems [Dvořák *et al.*, 2020b; Lehtonen *et al.*, 2021b]. That is, instead of solving a reasoning task with a single call of an ASP-solver such methods might use several calls to an ASP-solver in order to solve a single argumentation reasoning task. However, as these approaches are often tailored to specific reasoning tasks one loses a bit of the flexibility and modularity of the standard workflow.

6.3.1 Abstract Argumentation

Here we start with encoding a given argumentation framework $AF = (Ar, att)$ as facts [Egly *et al.*, 2010]. We use a unary predicate $\mathbf{arg}(x)$ to encode the arguments and a binary predicate $\mathbf{att}(x, y)$ to encode the attacks.

$$\begin{aligned}\pi_{input}(AF) = &\{\mathbf{arg}(x) \mid x \in Ar\}\cup \\ &\{\mathbf{att}(x, y) \mid (x, y) \in att\}\end{aligned}$$

Notice that $\pi_{input}(AF)$ only consists of facts and will act as input database for our queries. That is, $\pi_{input}(AF)$ is the only part that actually depends on

the given argumentation framework.

Example 22. *When considering our running example AF_{run} we obtain the following input database $\pi_{input}(AF_{run})$:*

$$\begin{array}{ll} \mathbf{arg}(a). & \mathbf{att}(a,b). \\ \mathbf{arg}(b). & \mathbf{att}(b,a). \\ \mathbf{arg}(c). & \mathbf{att}(b,c). \\ & \mathbf{att}(c,c). \end{array}$$

In the following we introduce encodings of the central argumentation semantics. We start with conflict-free sets. This encoding will also act as the basis for most of the other semantics. We use a predicate $\mathbf{in}$ to encode that an argument is in the set and a predicate $\mathbf{out}$ to encode that an argument is not in the set. We first encode that each argument a is either in the set ($\mathbf{in}(a)$) or it is not in the set ($\mathbf{out}(a)$). That is, we simply generate subsets of the arguments. We then add a constraint to ensure that we do not select two arguments that appear in the same attack.

$$\begin{aligned} \pi_{cf} = \{ & \mathbf{in}(X) \leftarrow \texttt{not}\ \mathbf{out}(X). \\ & \mathbf{out}(X) \leftarrow \texttt{not}\ \mathbf{in}(X). \\ & \leftarrow \mathbf{in}(X), \mathbf{in}(Y), \mathbf{att}(X,Y).\} \end{aligned}$$

If we now compute the answer-sets of $\pi_{AF}(AF) \cup \pi_{cf}$ we obtain the conflict-free sets AF from the answer-sets by, for each answer-set, forming a set with the arguments a that satisfy $\mathbf{in}(a)$.

Example 23. *The three answer-sets of $\pi_{input}(AF_{run}) \cup \pi_{cf}$ are (we neglect the input predicates $\mathbf{arg}$, $\mathbf{att}$):*

$$\begin{array}{l} \{\mathbf{out}(a), \mathbf{out}(b), \mathbf{out}(c)\} \\ \{\mathbf{in}(a), \mathbf{out}(b), \mathbf{out}(c)\} \\ \{\mathbf{out}(a), \mathbf{in}(b), \mathbf{out}(c)\} \end{array}$$

These answer-sets correspond to the three conflict-free sets $\emptyset$, $\{a\}$, $\{b\}$ of AF_{run}.

Proposition 32. *([Egly* et al.*, 2010]) For every argumentation framework $AF = (Ar, att)$, the conflict-free sets of AF are in one-to-one correspondence with the stable models of the normal logic program $\pi_{input}(AF) \cup \pi_{cf}$.*

Now starting from the characterisation of conflict-free sets we can encode more evolved semantics by adding the additional constraints these semantics have for their extensions. For admissible semantics we define a unary predicate **defeated** that encodes that an argument is attacked by the selected extensions, i.e., it includes the arguments that would be labeled `out` in a labelling based characterisation. We then add a constraint stating that there is no argument that attacks the extensions but is not attacked by the extension which ensures that the extension defends all its arguments.

$$\begin{aligned}\pi_{ad} = \pi_{cf} \cup \{&\mathbf{defeated}(X) \leftarrow \mathbf{in}(Y), \mathbf{att}(Y,X).\\ &\leftarrow \mathbf{in}(X), \mathbf{att}(Y,X), \texttt{not } \mathbf{defeated}(Y).\}\end{aligned}$$

Example 24. *The three answer-sets of $\pi_{input}(AF_{run}) \cup \pi_{ad}$ are (we neglect the input predicates* **arg** *and* **att***):*

$$\{\mathbf{out}(a), \mathbf{out}(b), \mathbf{out}(c)\}$$
$$\{\mathbf{in}(a), \mathbf{out}(b), \mathbf{out}(c), \mathbf{defeated}(b)\}$$
$$\{\mathbf{out}(a), \mathbf{in}(b), \mathbf{out}(c), \mathbf{defeated}(a), \mathbf{defeated}(c)\}$$

These answer-sets correspond to the three admissible sets $\emptyset$, $\{a\}$, $\{b\}$ of AF_{run}.

Again we obtain a one-to-one correspondence between admissible sets and the stable models of $\pi_{input}(AF) \cup \pi_{adm}$.

Proposition 33. *([Egly* et al.*, 2010]) For every argumentation framework $AF = (Ar, att)$, the admissible sets of AF are in one-to-one correspondence with the stable models of the normal logic program $\pi_{input}(AF) \cup \pi_{ad}$.*

Next we extend the encoding of admissible semantics by (a) a predicate **undefended** that contains all arguments that are not defended by the extension and (b) a constraint that states that there is not argument outside the extension the is defended.

$$\begin{aligned}\pi_{com} = \pi_{adm} \cup \{&\mathbf{undefended}(X) \leftarrow \mathbf{att}(Y,X), \texttt{not } \mathbf{defeated}(Y).\\ &\leftarrow \mathbf{out}(X), \texttt{not } \mathbf{undefended}(X).\}\end{aligned}$$

Example 25. *The three answer-sets of $\pi_{input}(AF_{run}) \cup \pi_{com}$ are (we neglect the input predicates* $\mathbf{arg}$ *and* $\mathbf{att}$*):*

$\{\mathbf{out}(a), \mathbf{out}(b), \mathbf{out}(c), \mathbf{undefended}(a), \mathbf{undefended}(b), \mathbf{undefended}(c)\}$
$\{\mathbf{in}(a), \mathbf{out}(b), \mathbf{out}(c), \mathbf{defeated}(b), \mathbf{undefended}(b), \mathbf{undefended}(c)\}$
$\{\mathbf{out}(a), \mathbf{in}(b), \mathbf{out}(c), \mathbf{defeated}(a), \mathbf{defeated}(c), \mathbf{undefended}(a),$
$\mathbf{undefended}(c)\}$

These answer-sets correspond to the three complete extensions $\emptyset$, $\{a\}$, $\{b\}$ of AF_{run}.

Proposition 34. *([Egly* et al.*, 2010]) For every argumentation framework $AF = (Ar, att)$, the complete extensions of AF are in one-to-one correspondence with the stable models of the normal logic program $\pi_{input}(AF) \cup \pi_{com}$.*

Along the same lines we one can extend the conflict-free encoding for stable semantics by again defining a predicate $\mathbf{defeated}$ and adding a constraint that rules out arguments that are neither in the extension nor attacked.

$$\pi_{st} = \pi_{cf} \cup \{\mathbf{defeated}(X) \leftarrow \mathbf{in}(Y), \mathbf{att}(Y, X).$$
$$\leftarrow \mathbf{out}(X), \texttt{not } \mathbf{defeated}(X).\}$$

Example 26. *The only answer-set of $\pi_{input}(AF_{run}) \cup \pi_{st}$ is (we neglect the input predicates* $\mathbf{arg}$ *and* $\mathbf{att}$*):*

$$\{\mathbf{out}(a), \mathbf{in}(b), \mathbf{out}(c), \mathbf{defeated}(a), \mathbf{defeated}(c)\}$$

The unique answer-set correspond to the only stable extension $\{b\}$ of AF_{run}.

As before, this encoding explicitly encodes all the requirements of stable extensions as rules of the LP. That is, it does not use the close connection between stable semantics for AFs and stable model semantics for LPs. However, there is also a more direct approach using conditional literals [Dvořák *et al.*, 2020b], that follows the correspondence result from Section 3.2.

$$\pi_{st'}(AF) = \{\mathbf{in}(Y) \leftarrow \mathbf{arg}(Y), \texttt{not } \mathbf{in}(X) : \mathbf{att}(X, Y).\}$$

This encoding typically results in smaller grounding of the program in the ASP-solving process and has shown better performance on benchmark instances. Of course, both encodings provide the one-to-one correspondence between stable extensions and answer-sets.

Example 27. *The only answer-set of $\pi_{input}(AF_{run}) \cup \pi_{st'}$ is (we neglect the input predicates* $\mathbf{arg}$ *and* $\mathbf{att}$*) is $\{\mathbf{in}(b)\}$. This answer-set corresponds to the only stable extension $\{b\}$ of AF_{run}.*

Proposition 35. *([Egly* et al.*, 2010; Dvořák* et al.*, 2020b]) For every argumentation framework $AF = (Ar, att)$, we have that the stable extensions of AF are in one-to-one correspondence with the stable models of the normal logic programs $\pi_{input}(AF) \cup \pi_{st}$ and $\pi_{input}(AF) \cup \pi_{st'}$ respectively.*

Finally, let us consider preferred semantics. Due to complexity results, we know that we need disjunctive rules for preferred semantics. As preferred extensions are $\subseteq$-maximal admissible set we start from the encoding for admissible semantics (or alternatively from the encoding complete semantics) and can then apply ASP-techniques for $\subseteq$-maximization. Here we follow [Gaggl *et al.*, 2015] and present an encoding based on saturation that uses conditional disjunction in the rule heads. The use of conditional disjunction allows for a compact encoding compared to earlier versions,

$$
\begin{aligned}
\pi_{pr} = \pi_{adm} \cup \\
\{\ & \mathbf{notTrivial} \leftarrow \mathbf{out}(X). \\
& \mathbf{witness}(X) : \mathbf{out}(X) \leftarrow \mathbf{notTrivial}. \\
& \mathbf{spoil} \vee \mathbf{witness}(Z) : \mathbf{att}(Z,Y) \leftarrow \mathbf{witness}(X), \mathbf{att}(Y,X). \\
& \mathbf{spoil} \leftarrow \mathbf{witness}(X), \mathbf{witness}(Y), \mathbf{att}(X,Y). \\
& \mathbf{spoil} \leftarrow \mathbf{in}(X), \mathbf{witness}(Y), \mathbf{att}(X,Y). \\
& \mathbf{witness}(X) \leftarrow \mathbf{spoil}, \mathbf{arg}(X). \\
& \leftarrow \texttt{not}\ \mathbf{spoil}, \mathbf{notTrivial}.\}
\end{aligned}
$$

The admissible part generates admissible extensions and the additional rules deal with the $\subseteq$-maximality. The encoding first tests whether we are in the trivial case where all arguments are in the extension and thus the extension is clearly preferred. If not, we aim to construct a larger extension by adding some of the arguments which are not in the admissible set (at least one). These new arguments are stored in the predicate $\mathbf{witness}$. We then have another rule that adds further arguments if those are required to defend the arguments already included as witness. If that is not possible, we obtain the constant $\mathbf{spoil}$ which indicates that the set we constructed is not admissible. Moreover we have two rules that check whether the constructed set is conflict-free and if

not yield the constant **spoil**. The two final rules are then due to the saturation technique. First, whenever we obtain spoil we have to include all arguments as witnesses. This ensures that we only get one answer-set for each preferred extension and also makes sure that we rule out such potential answer-sets when we find an model for the same admissible set that does not lead **spoil**. Now assume we find a model M that satisfies all rules but the last one and does not include spoil. Of course, the last rule ensures that M is not an answer-set, but it also rules out all answer-sets M' that coincide on the admissible part but yield **spoil** as M is always a smaller model than M' on the reduct of M'.

Example 28. *The two answer-sets of $\pi_{input}(AF_{run}) \cup \pi_{pr}$ are (we neglect the input predicates* **arg** *and* **att***):*

$$\{\mathbf{in}(a), \mathbf{out}(b), \mathbf{out}(c), \mathbf{defeated}(b), \mathbf{spoil}, \mathbf{notTrivial}, \mathbf{witness}(a), \mathbf{witness}(b), \mathbf{witness}(c)\}$$
$$\{\mathbf{out}(a), \mathbf{in}(b), \mathbf{out}(c), \mathbf{defeated}(a), \mathbf{defeated}(c), \mathbf{spoil}, \mathbf{notTrivial}, \mathbf{witness}(a), \mathbf{witness}(b), \mathbf{witness}(c)\}$$

These two answer-sets correspond to the preferred extensions $\{a\}$, $\{b\}$ of AF_{run}.

Now consider the answer-set $S = \{\mathbf{out}(a), \mathbf{out}(b), \mathbf{out}(c)\}$ for the admissible encoding (which corresponds to the empty set). If we extend S by the atoms $\mathbf{witness}(a)$ *and* **notTrivial***, i.e., we consider $S' = \{\mathbf{out}(a), \mathbf{out}(b), \mathbf{out}(c), \mathbf{witness}(a), \mathbf{notTrivial}\}$, we satisfy all but the last rule of π_{pr}. Moreover, the last rule is clearly violated as* **spoil** *is not included in S'. That is, S' is itself not an answer-set of the program but also excludes all other models extending S and containing* **spoil** *from being an answer-set, as this would be violating the minimal model property on the reduct.*

Proposition 36. *([Gaggl* et al.*, 2015]) For every argumentation framework $AF = (Ar, att)$, we have that the preferred extensions of AF are in one-to-one correspondence with the stable models of the logic program $\pi_{input}(AF) \cup \pi_{pr}$.*

An alternative approach towards encodings of preferred semantics [Dvořák *et al.*, 2020b] is to use the encoding for admissible semantics and exploit clingo domain heuristics [Gebser *et al.*, 2013] to perform the $\subseteq$-maximization on the **in** predicate.

In this section we presented prototypical ASP encodings for the selected semantics. Indeed, there are ASP encodings for most of the argumentation semantics and those are integrated in the ASPARTIX system [Dvořák *et al.*, 2020a]. For the interested reader we next provide the relevant pointers to the literature. Encodings for semi-stable and stage semantics are discussed in [Egly *et al.*, 2010; Gaggl *et al.*, 2015], ideal semantics are discussed in [Faber and Woltran, 2009; Dvořák *et al.*, 2020b], encodings for cf2 semantics are presented in [Osorio *et al.*, 2010; Gaggl and Woltran, 2013], for encodings of resolution-based grounded semantics see [Dvořák *et al.*, 2011], and strongly admissible semantics and their minimization are discussed in [Dvořák and Wallner, 2020].

ASP queries to decide acceptance problems. Given the encoding of an argumentation semantics we can use an ASP-solver to compute an extension or even all extensions of an argumentation framework. However, sometimes we are even more interested in the acceptance status of an argument. Classic acceptance problems are the credulous/skeptical acceptance of an argument a (or a set of arguments $S = \{a_1, \ldots a_n\}$). That is, deciding whether an argument a (or a set S) is contained in one of the extensions, in all of the extensions respectively. We can simply address these problems by adding an additional rule to the encoding. For credulous acceptance we require that $\mathbf{in}(a)$ holds while for skeptical acceptance we are looking for a counter example by requiring that $\mathbf{in}(a)$ does not hold.

$$\begin{aligned}
\pi_{cred}(a) &= \{\leftarrow \texttt{not } \mathbf{in}(a)\} \\
\pi_{cred}(S) &= \{\leftarrow \texttt{not } \mathbf{in}(a) \mid a \in S\} \\
\pi_{skept}(a) &= \{\leftarrow \mathbf{in}(a)\} \\
\pi_{skept}(S) &= \{\leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_n) \mid S = \{a_1, \ldots a_n\}\}
\end{aligned}$$

If we now solve $\pi_{input}(AF) \cup \pi_\sigma \cup \pi_{cred}(a)$ the solver either provides an answer-set that corresponds to a σ-extension that contains the argument a or returns that the program is unsatisfiable. In the later case we know that no such σ-extension exists and thus can answer the query negatively. Similarly, if we now solve $\pi_{input}(AF) \cup \pi_\sigma \cup \pi_{skept}(a)$ the solver either provides an answer-set that corresponds to a σ-extension that does not include the argument a, i.e., argument a is not skeptically accepted, or returns that the program is unsat-

isfiable. In the later case we know that no σ-extension that acts as counter example exists and thus can answer the query positively.

ASP for generalizations of abstract argumentation frameworks. ASP encodings have been provided for a number of generalizations of AFs. Some of the extensions are directly based on the encoding for AFs. For instance when dealing with preferences or support the encodings [Egly *et al.*, 2010] construct a new attack relation that corresponds to standard AF and then use the encodings of semantics for AFs. Similarly, for argumentation frameworks with recursive attacks (AFRAs) the encoding of the ASPARTIX system [40] constructs an equivalent AF and exploits the existing encodings. However, for other generalizations careful adaptations or even new encodings are required. Popular examples are the encodings for extended argumentation frameworks (EAF) [Dvořák *et al.*, 2015], SETAFs [Dvořák *et al.*, 2018] and Abstract Dialectical Frameworks (ADF) [Ellmauthaler and Wallner, 2012; Ellmauthaler and Strass, 2014]. While a full discussion of these encodings is beyond the scope of this chapter we want to give a first impression how these generalizations can be approached via ASP by illustrating the input encodings of the formalisms in Figure 5.

6.3.2 Assumption-Based Argumentation

In this section we discuss the ASPARTIX-style approach towards assumption-based argumentation by Lehtonen *et al.* (2021a, 2021b). That is, we present an approach that encodes assumption-based argumentation frameworks (ABAFs) as input database and then provide fixed encodings of semantics that do not depend on the actual framework.

In the following let $\mathcal{F} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ \ })$ be an ABAF such that $\mathcal{R} = \{r_1, r_2, ...r_k\}$. We use the following set of facts $\pi_{ABA}(\mathcal{F})$ to represent the

[40] `https://www.dbai.tuwien.ac.at/research/argumentation/aspartix/afra.html`

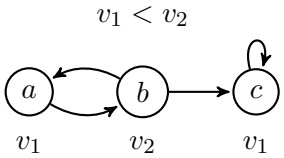
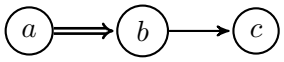
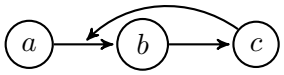
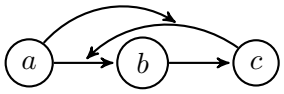
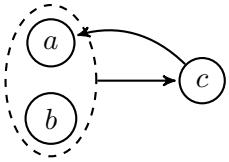
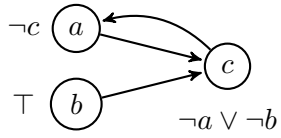
ARGUMENTATION FRAMEWORK	INPUT ENCODING
Value-based AF	
	$\mathbf{arg}(a)$. $\mathbf{arg}(b)$. $\mathbf{arg}(c)$. $\mathbf{att}(a,b)$. $\mathbf{att}(b,a)$. $\mathbf{att}(b,c)$.$\mathbf{att}(c,c)$. $\mathbf{val}(a,v_1)$. $\mathbf{val}(b,v_2)$. $\mathbf{val}(a,v_1)$. $\mathbf{valpref}(v_1,v_2)$
Bipolar AF	
	$\mathbf{arg}(a)$. $\mathbf{arg}(b)$. $\mathbf{arg}(c)$. $\mathbf{att}(b,c)$. $\mathbf{support}(a,b)$.
Extended AF	
	$\mathbf{arg}(a)$. $\mathbf{arg}(b)$. $\mathbf{arg}(c)$. $\mathbf{att}(a,b)$. $\mathbf{att}(b,c)$. $\mathbf{d}(c,a,b)$.
AF with recursive attacks	
	$\mathbf{afraA}(a)$. $\mathbf{afraA}(b)$. $\mathbf{afraA}(c)$. $\mathbf{afraR}(att1,a,b)$. $\mathbf{afraR}(att2,b,c)$. $\mathbf{afraR}(att3,c,att1)$. $\mathbf{afraR}(att4,a,att3)$.
AFs with collective attacks (SETAF)	
	$\mathbf{arg}(a)$. $\mathbf{arg}(b)$. $\mathbf{arg}(c)$. $\mathbf{att}(att1,c)$. $\mathbf{mem}(att1,a)$. $\mathbf{mem}(att1,b)$. $\mathbf{att}(att2,a)$. $\mathbf{mem}(att2,c)$.
Abstract Dialectical Frameworks (ADF)	
	$\mathbf{statement}(a)$. $\mathbf{statement}(b)$. $\mathbf{statement}(c)$. $\mathbf{ac}(a,neg(c))$. $\mathbf{ac}(b,c(v))$. $\mathbf{ac}(c,or(neg(a),neg(b)))$.

Figure 5: Input encodings for different generalizations of AFs.

ABAF $\mathcal{F}$ for our further investigations:

$$\begin{aligned}\pi_{ABA}(\mathcal{F}) =& \{\mathbf{assumption}(a) \mid a \in \mathcal{A}\} \cup \\ & \{\mathbf{head}(i,h) \mid r_i \in \mathcal{R}, h \in head(r_i)\} \cup \\ & \{\mathbf{body}(i,b) \mid r_i \in \mathcal{R}, b \in body(r_i)\} \cup \\ & \{\mathbf{contrary}(a,\overline{a}) \mid a \in \mathcal{A}\}.\end{aligned}$$

Notice that $\pi_{ABA}(\mathcal{F})$ introduces a unique identifiers for each rules and then uses the binary predicate $\mathbf{head}$ to encode the head of a rule and a predicate $\mathbf{body}$ to encode the body atoms of the rule. Moreover, we can easily extend the input encoding to also deal with ABAFs that have a preference relation $\leq$ (commonly referred to as ABA$^+$ frameworks). That is we introduce a binary predicate $\mathbf{preferred}$ and add a fact $\mathbf{preferred}(a,b)$ to the input database whenever $a \leq b$, i.e., we obtain $\pi_{ABA^+}(\mathcal{F},\leq) = \pi_{ABAF}(\mathcal{F}) \cup \{\mathbf{preferred}(x,y) \mid y \leq x\}$. However, in this chapter we will focus on encodings of ABAFs without preferences and moreover restrict ourselves to flat ABAFs. The interested reader is referred to [Lehtonen *et al.*, 2021a; Lehtonen *et al.*, 2021b] for ASP encodings of ABA$^+$.

We start with encoding conflict-free assumption sets. Again this encoding will be the basis for the encodings of the other semantics. To this end we use two unary predicates $\mathbf{in}$ and $\mathbf{out}$ to encode that an assumption is in respectively outside the considered assumption set. We then introduce the unary predicate $\mathbf{supported}$ which encodes that a statement can be derived from the selected assumptions. In order to derive $\mathbf{supported}$ we use a unary predicate $\mathbf{triggered}$ to encode all rules whose body is satisfied by the selected assumptions and the already derived statements. Moreover, we have the unary $\mathbf{defeated}$ predicate that computes the assumptions which are attacked by the selected assumption set and is based on $\mathbf{supported}$ and the contrary function. Finally, to ensure conflict-freeness, we have a constraint that states that none

of the selected assumptions is defeated.

$$\begin{aligned}\pi_{cf} = \{&\mathbf{in}(X) \leftarrow \mathbf{assumption}(X), \texttt{not}\ \mathbf{out}(X).\\ &\mathbf{out}(X) \leftarrow \mathbf{assumption}(X), \texttt{not}\ \mathbf{in}(X).\\ &\mathbf{supported}(X) \leftarrow \mathbf{assumption}(X), \mathbf{in}(X).\\ &\mathbf{supported}(X) \leftarrow \mathbf{head}(R, X), \mathbf{triggered}(R).\\ &\mathbf{triggered}(R) \leftarrow \mathbf{head}(R, \_), \mathbf{supported}(X) : \mathbf{body}(R, X).\\ &\mathbf{defeated}(X) \leftarrow \mathbf{supported}(Y), \mathbf{contrary}(X, Y).\\ &\leftarrow \mathbf{in}(X), \mathbf{defeated}(X).\}\end{aligned}$$

The conflict-free sets of the ABAF correspond to the answer-set of the LP $\pi_{ABA}(D) \cup \pi_{cf}$, i.e., given an answer-set we can compute the corresponding assumption set by inspecting the **in** predicate. However, for ABAFs we are not only interested in the accepted assumptions but also in the statements that can be derived from them. These statements are accessible via the **supported** predicate, i.e., for each answer-set a statement s is a consequence of the selected assumptions if and only if $\mathbf{supported}(s)$ holds in that answer-set.

Proposition 37. *([Lehtonen* et al.*, 2021a]) For every ABAF $\mathcal{F} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ })$, the conflict-free assumption sets of $\mathcal{F}$ are in one-to-one correspondence with the answer-sets of $\pi_{ABA}(\mathcal{F}) \cup \pi_{cf}$. Moreover, a statement s can be derived from a conflict-free assumption set iff $\mathbf{supported}(s)$ is in the corresponding answer-set.*

Next we consider admissible semantics. The idea here is to consider assumptions that are not defeated by the set of selected assumptions and test whether they attack the selected assumptions, which of course would violate admissibility. To this end we introduce the predicate **derivedFromUndef** that contains all undefeated assumptions and all statements that can be derived from them. Furthermore, we again use a predicate **triggeredByUndef** which collects the rules we use to derive these statements. Finally, with the predicate **attackedByUndef** we derive the assumptions attacked by the undefeated assumptions and add a constraint that none of the assumptions in our set is attacked by the undefeated assumptions.

$\pi_{ad} = \pi_{cf} \cup \{$
$\mathbf{derivedFromUndef}(X) \leftarrow \mathbf{assumption}(X), \texttt{not } \mathbf{defeated}(X).$
$\mathbf{derivedFromUndef}(X) \leftarrow \mathbf{head}(R, X), \mathbf{triggeredByUndef}(R).$
$\mathbf{triggeredByUndef}(R) \leftarrow \mathbf{head}(R, \_),$
$\mathbf{derivedFromUndef}(X) : \mathbf{body}(R, X).$
$\mathbf{attackedByUndef}(X) \leftarrow \mathbf{contrary}(X, Y), \mathbf{derivedFromUndef}(Y).$
$\leftarrow \mathbf{in}(X), \mathbf{attackedByUndef}(X).\}$

Proposition 38. *([Lehtonen* et al.*, 2021a]) For every ABAF $\mathcal{F} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ })$, the admissible assumption sets of $\mathcal{F}$ are in one-to-one correspondence with the answer-sets of $\pi_{ABA}(\mathcal{F}) \cup \pi_{ad}$. Moreover, a statement s can be derived from an admissible assumption set iff $\mathbf{supported}(s)$ is in the corresponding answer-set.*

Now the step from admissible to complete semantics is rather easy. We just add a constraint that there are no assumptions that are neither in the selected set nor attacked by the undefeated assumptions.

$$\pi_{co} = \pi_{adm} \cup \{\leftarrow \mathbf{out}(X), \texttt{not } \mathbf{attackedByUndef}(X).\}$$

Proposition 39. *([Lehtonen* et al.*, 2021a]) For every ABAF $\mathcal{F} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{\ })$, the complete assumption sets of $\mathcal{F}$ are in one-to-one correspondence with the answer-sets of $\pi_{ABA}(\mathcal{F}) \cup \pi_{co}$. Moreover, a statement s can be derived from a complete assumption set iff $\mathbf{supported}(s)$ is in the corresponding answer-set.*

Given the encoding for admissible and complete semantics one can use standard techniques for $\subseteq$-maximization to deal with preferred semantics. For enumeration Lehtonen *et al.* [2021a] propose to use preferential optimization statements while for skeptical reasoning algorithms iterative calls to the ASP-solver are used [Lehtonen *et al.*, 2021b].

Now for stable semantics, we only rely on the conflict-free encoding and add a constraint that each assumption that is not selected must be defeated by the selected assumptions.

$$\pi_{st} = \pi_{cf} \cup \{\leftarrow \mathbf{out}(X), \texttt{not } \mathbf{defeated}(X).\}$$

Proposition 40. *([Lehtonen* et al.*, 2021a]) For every ABAF $\mathcal{F} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, \overline{})$, the stable assumption sets of $\mathcal{F}$ are in one-to-one correspondence with the answer-sets of $\pi_{ABA}(\mathcal{F}) \cup \pi_{st}$. Moreover, a statement s can be derived from a stable assumption set iff $\mathbf{supported}(s)$ is in the corresponding answer-set.*

We next provide an alternative encoding of stable semantics following the correspondence result between ABA stable semantics and stable model semantics of LPs in Section 4.3.2 (cf. Theorem 18). That is, we restate ABA rules as LP rules using conditional literals and then add rules stating that an assumption is supported if its contrary is not. In order to be compatible with the other encodings we then introduce rules that fills the $\mathbf{in}$ predicate with the supported assumptions.

$$\begin{aligned}\pi_{st'} = \{&\mathbf{supported}(X) \leftarrow \mathbf{head}(R, X), \mathbf{supported}(Y) : \mathbf{body}(R, Y).\\ &\mathbf{supported}(X) \leftarrow \mathbf{contrary}(X, Z), \texttt{not } \mathbf{supported}(Z).\\ &\mathbf{in}(X) \leftarrow \mathbf{assumption}(X), \mathbf{supported}(X).\}\end{aligned}$$

ASP queries to decide acceptance problems. Given the characterisations above we can use ASP-solver to compute an assumption set or even all assumption sets of an argumentation framework (under a given semantics). However, sometimes we are even more interested in the acceptance status of an assumption or a statement. Classic acceptance problems are the credulous/skeptical acceptance of a statement s, i.e., deciding whether there is an assumption set that implies s or deciding whether s is implied by all assumption sets (under a given semantics). Again we can address these problems by adding the corresponding rule to the encoding. For credulous acceptance we require that $\mathbf{supported}(s)$ holds while for skeptical acceptance we are looking for a counter example by requiring that $\mathbf{supported}(s)$ does not hold.

$$\begin{aligned}\pi_{cred}(s) &= \{\leftarrow \texttt{not } \mathbf{supported}(s).\}\\ \pi_{skept}(s) &= \{\leftarrow \mathbf{supported}(s).\}\end{aligned}$$

If we now solve $\pi_{ABA}(\mathcal{F}) \cup \pi_\sigma \cup \pi_{cred}(s)$ the solver either provides an answer-set that corresponds to a σ-assumption set that implies the statement s or returns that the program is unsatisfiable. In the later case we know that there is no such σ-assumption set and thus can answer the query negatively. Similarly, if we now solve $\pi_{ABA}(\mathcal{F}) \cup \pi_\sigma \cup \pi_{skept}(s)$ the solver either provides an answer-set that corresponds to a σ-assumption set that does not imply the statement s, i.e., statement s is not skeptically accepted, or returns that the program is unsatisfiable. In the later case we know that no σ-assumption set that acts as counter-example exists and thus can answer the query positively.

7 Discussion

In this chapter, we have discussed several argumentation systems (Dung's AFs, ABA, ADFs, SETAFs) focusing on their connections to logic programming. In each case, whenever it applies, we highlighted similarities concerning syntax, terminology and the representation and computation of semantic models, implementation in ASP as well as translations from theories in each system to and from normal logic programs. The highest interest when comparing logical systems usually concerns their relative expressive power, so we focused on the relation between the various systems based on semantics: for each argumentation system, we considered whether they can model inference from logic programming and vice-versa. The results gathered ensure that:

1. normal logic programs semantically subsume Dung's AFs (Section 3) [41]

2. normal logic programs are semantically subsumed by flat-ABA, but they become equivalent if the programs are equipped with semantic projection (Section 4) [42]

3. normal logic programs are semantically subsumed by ADFs[43], but they remain equivalent to the fragment of ADF$^+$s (Section 5.1) [44]

[41] The results were gathered primarily from [Caminada *et al.*, 2015b; Sá and Alcântara, 2021a; Caminada *et al.*, 2022].

[42] The results were gathered primarily from [Bondarenko *et al.*, 1997; Caminada and Schulz, 2017; Sá and Alcântara, 2019; Sá and Alcântara, 2021b].

[43] If ADFs are equipped with three-valued acceptance conditions, which were proposed in [Alcântara and Sá, 2019].

[44] The results were gathered primarily from [Strass, 2013; Alcântara *et al.*, 2019].

4. normal logic programs are equivalent to SETAF, with their semantics coinciding for all programs rid of irrelevant atoms (Section 5.2) [45]

Other systems worth mentioning regarding their relation to logic programming are Defeasible Logic Programming (DeLP) [Simari and Loui, 1992; García and Simari, 2004] and Claim-Augmented AFs (CAFs) [Dvořák and Woltran, 2020; König *et al.*, 2022].

DeLP [Simari and Loui, 1992] is a rule-based argumentation system introduced in [García and Simari, 2004] as a formalism that combines results of logic programming and defeasible argumentation. Overall, the presentation of concepts in DeLP follow conventions from Logic Programming, but one important difference is that their semantics is inherently based on the possible *derivations* of claims, which intuitively amounts to the arguments constructed from a logic program following Definition 13. Further, the syntax of DeLP includes strong negation and the possibility of a priority relation between rules, which is extended to a set of preferences over arguments. These features pose as obstacles to a direct comparison between DeLP and normal logic programs or even AFs.

Intuitively, Claim-Augmented AFs (CAFs) [Dvořák and Woltran, 2020] are the same as AFs, but each argument is accompanied by a *claim* (or conclusion, as we consider in Definition 13). CAFs admit the argument extension semantics for AFs (including complete, grounded, preferred, etc.) and also claim-based variants of extensions where the claims of arguments in a σ-extension S of a CAF are extracted into corresponding σ-claim-extensions. The conversion between argument-based and claim-based extensions is identical to that observed in [Caminada *et al.*, 2015b] to convert the argument labellings of $\texttt{LP2AA}(P)$ (see Section 3.3) into *conclusion labellings*,which can be compared to the models of the program P. In fact, whatever P is, $\texttt{LP2AA}(P)$ can be immediately perceived as a CAF, since each argument A built using Definition 13 has a single atom (a claim) assigned as $\texttt{Conc}(A)$. From this connection, a series of results concerning the relationship between CAFs and normal logic programs are implied from the results of [Caminada *et al.*, 2015b] and have been considered in [Dvořák *et al.*, 2023a; König *et al.*, 2022]: it means CAFs are subsumed by logic programs in the same way AFs are; but then translating from LPs to CAFs, we will find some differences between

[45] The results were gathered primarily from [König *et al.*, 2022; Alcântara *et al.*, 2023].

the L-stable program semantics and the CAF semi-stable claim-based semantics. Based on this observation, [Dvořák *et al.*, 2023c] introduces alternative versions of semantics for CAFs based on the additional information about the claims of the argument. Prominently it provides a variant of semi-stable semantics that naturally maps to L-stable program semantics. Computational properties of these semantics have then been investigated in [Dvořák *et al.*, 2023a]. [König *et al.*, 2022] studied the connection between CAFs and logic programs, but they only prove correspondence for the stable semantics.[46]

It is also worth mentioning the investigation of [Caminada *et al.*, 2015a] about the relationship between AFs and ABA. Their translations between AF and ABA could be combined with the translations from [Caminada *et al.*, 2015b] between NLP and AF (see Section 3) to obtain results about the relationship between ABA and LP. We conjecture that the potential results obtained through the combined translations would prove themselves redundant to the results found in [Caminada and Schulz, 2017] and [Sá and Alcântara, 2021b] (Section 4).

We have also provided a detailed overview of the implementation of argumentation systems and the retrieval of their semantics using answer-set programs (ASP), an expressive class of logic programs based on the stable model semantics. Our discussion in Section 6 includes mapping theories of systems such as AFs and ABA to corresponding answer set programs, allowing the computation of their semantics using ASP solvers. On that matter, several authors contributed to this line of research. Nieves *et al.* [2008] provided a compiler-style approach to implement abstract argumentation in ASP, where the logic program is computed from the considered argumentation framework. On the other hand, Wakaki and Nitta [2008] and Egly *et al.* [2008] provided the first query-based implementations for abstract argumentation, where the argumentation framework is provided as input database. While Wakaki and Nitta followed the labelling-based characterisations of the semantics, Egly *et al.* followed the extension-based characterisation in their encodings. For a comparison of these early works the interested reader is referred to [Toni and Sergot, 2011]. The ASPARTIX system [47] of [Egly *et al.*, 2008] was later extended to

[46] [König *et al.*, 2022] discusses back-and-forth translations between multiple systems, including CAFs, NLPs, ABA frameworks, ADFs and SETAFs, but the focus is primarily on the syntax of theories in each system.

[47] `https://www.dbai.tuwien.ac.at/research/argumentation/aspartix/`

deal with several generalizations of abstract argumentation and new semantics that were introduced in the literature (see, e.g., [Egly *et al.*, 2010; Dvořák *et al.*, 2015; Ellmauthaler and Strass, 2014; Dvořák and Wallner, 2020]). Alternative encodings in a compiler-style approach were introduced by Sakama and Rienstra [2017]. Moreover, ASP techniques have also been applied to structured argumentation formalisms, prominently assumption-based argumentation. The first approaches first build a corresponding abstract argumentation framework and then use ASP Encodings on these frameworks to deal with the argumentation semantics [Cyras and Toni, 2016; Lehtonen *et al.*, 2017]. However, more recent systems directly approach the assumption-based argumentation semantics without constructing abstract frameworks [Lehtonen *et al.*, 2021a; Lehtonen *et al.*, 2021b].

References

[Alcântara and Sá, 2019] João Alcântara and Samy Sá. On three-valued acceptance conditions of abstract dialectical frameworks. *Electronic Notes in Theoretical Compututer Science*, 344(C):3–23, aug 2019.

[Alcântara and Sá, 2021] João Alcântara and Samy Sá. Equivalence results between SETAF and attacking abstract dialectical frameworks. In Leila Amgoud and Richard Booth, editors, *Proceedings of the 19th International Workshop on Nonmonotonic Reasoning (NMR 2021)*, pages 139–148, 2021.

[Alcântara *et al.*, 2019] João Alcântara, Samy Sá, and Juan Acosta-Guadarrama. On the equivalence between abstract dialectical frameworks and logic programs. *Theory and Practice of Logic Programming*, 19(5-6):941–956, 2019.

[Alcântara *et al.*, 2023] João Alcântara, Renan Cordeiro, and Samy Sá. On the equivalence between logic programming and SETAF. *(submitted, under review)*, 2023.

[Baroni *et al.*, 2018] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. Abstract argumentation frameworks and their semantics. In *Handbook of Formal Argumentation*, volume 1. College Publications, 2018.

[Baumann and Spanring, 2015] Ringo Baumann and Christof Spanring. Infinite argumentation frameworks — on the existence and uniqueness of extensions. In Th. Eiter, H. Strass, M. Truszczyński, and S. Woltran, editors, *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation — Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, page 281–295. Springer, 2015.

[Baumann, 2018] Ringo Baumann. On the nature of argumentation semantics: Existence and uniqueness, expressibility, and replaceability. In *Handbook of Formal*

Argumentation, volume 1. College Publications, 2018.

[Besnard and Hunter, 2014] Philippe Besnard and Anthony Hunter. Constructing argument graphs with deductive arguments: A tutorial. *Argument & Computation*, 5:5–30, 2014. Special Issue: Tutorials on Structured Argumentation.

[Bikakis *et al.*, 2021] Antonis Bikakis, Andrea Cohen, Wolfgang Dvořák, Giorgos Flouris, and Simon Parsons. Joint attacks and accrual in argumentation frameworks. In D. Gabbay, M. Giacomin, G. Simari, and M. Thimm, editors, *Handbook of Formal Argumentation*, volume 2, chapter 2. College Publications, 2021.

[Bondarenko *et al.*, 1997] Andrei Bondarenko, Phan Minh Dung, Bob Kowalski, and Francesca Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93:63–101, 1997.

[Brewka and Woltran, 2010] Gerhard Brewka and Stefan Woltran. Abstract dialectical frameworks. In *Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 102–111. AAAI Press, 2010.

[Brewka *et al.*, 2013] Gerhard Brewka, Stefan Ellmauthaler, Hannes Strass, Johannes Wallner, and Stefan Woltran. Abstract dialectical frameworks revisited. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 803–809. AAAI Press, 2013.

[Brewka *et al.*, 2018] Gerhard Brewka, Stefan Ellmauthaler, Hannes Strass, Johannes P. Wallner, and Stefan Woltran. Abstract dialectical frameworks. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, volume 1, chapter 5, pages 237–285. College Publications, 2018.

[Caminada and Gabbay, 2009] Martin Caminada and Dov Gabbay. A logical account of formal argumentation. *Studia Logica*, 93(2-3):109–145, 2009. Special issue: new ideas in argumentation theory.

[Caminada and Schulz, 2017] Martin Caminada and Claudia Schulz. On the equivalence between assumption-based argumentation and logic programming. *Journal of Artificial Intelligence Research*, 60:779–825, 2017.

[Caminada and Verheij, 2010] Martin Caminada and Bart Verheij. On the existence of semi-stable extensions. In G. Danoy, M. Seredynski, R. Booth, B. Gateau, I. Jars, and D. Khadraoui, editors, *Proceedings of the 22nd Benelux Conference on Artificial Intelligence*, 2010.

[Caminada *et al.*, 2015a] Martin Caminada, Samy Sá, João Alcântara, and Wolfgang Dvorák. On the difference between assumption-based argumentation and abstract argumentation. *IfCoLog Journal of Logics and their Applications*, 2:15–34, 2015.

[Caminada *et al.*, 2015b] Martin Caminada, Samy Sá, João Alcântara, and Wolfgang Dvořák. On the equivalence between logic programming semantics and argumentation semantics. *International Journal of Approximate Reasoning*, 58:87–111,

2015.

[Caminada *et al.*, 2022] Martin Caminada, Sri Harikrishnan, and Samy Sá. Comparing logic programming and formal argumentation; the case of ideal and eager semantics. *Argument & Computation*, 13:91–120, 2022.

[Caminada *et al.*, 2024] Martin Caminada, Matthias König, Anna Rapberger, and Markus Ulbricht. Attack semantics and collective attacks revisited. *Argument & Computation*, 2024. in print.

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

[Cerutti *et al.*, 2018] Federico Cerutti, Sarah Gaggl, Matthias Thimm, and Johannes Wallner. Foundations of implementations for formal argumentation. In P. Baroni, D. Gabbay, M. Giacomin, and L. van der Torre, editors, *Handbook of Formal Argumentation*, chapter 15, pages 688–767. College Publications, 2018. also appears in IfCoLog Journal of Logics and their Applications 4(8):2623–2706.

[Charwat *et al.*, 2015] Günther Charwat, Wolfgang Dvořák, Sarah Alice Gaggl, Johannes Peter Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation - A survey. *Artificial Intelligence*, 220:28–63, 2015.

[Cramer and Saldanha, 2020] Marcos Cramer and Emmanuelle-Anna Dietz Saldanha. Logic programming, argumentation and human reasoning. In Mehdi Dastani, Huimin Dong, and Leon van der Torre, editors, *Logic and Argumentation*, pages 58–79, Cham, 2020. Springer International Publishing.

[Cyras and Toni, 2016] Kristijonas Cyras and Francesca Toni. ABA+: assumption-based argumentation with preferences. In Ch. Baral, J. Delgrande, and F. Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*, pages 553–556. AAAI Press, 2016.

[Čyras *et al.*, 2018] Kristijonas Čyras, Xiuyi Fan, Claudia Schulz, and Francesca Toni. Assumption-based argumentation: Disputes, explanations, preferences. In *Handbook of Formal Argumentation*, volume 1. College Publications, 2018.

[Dung *et al.*, 2007] Phan Minh Dung, Paolo Mancarella, and Francesca Toni. Computing ideal sceptical argumentation. *Artificial Intelligence*, 171(10-15):642–674, 2007.

[Dung *et al.*, 2009] Phan Minh Dung, Bob Kowalski, and Francesca Toni. Assumption-based argumentation. In G. Simari and I. Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 199–218. Springer US, 2009.

[Dung, 1991] Phan Minh Dung. Negations as hypotheses: An abductive foundation for logic programming. In Koichi Furukawa, editor, *Logic Programming, Proceed-*

ings of the Eigth International Conference, Paris, France, June 24-28, 1991, pages 3–17. MIT Press, 1991.

[Dung, 1995a] Phan Minh Dung. An argumentation-theoretic foundation for logic programming. *The Journal of Logic Programming*, 22(2):151–177, 1995.

[Dung, 1995b] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.

[Dvořák and Dunne, 2018] Wolfgang Dvořák and Paul Dunne. Computational problems in formal argumentation and their complexity. In P. Baroni, D. Gabbay, M. Giacomin, and L. van der Torre, editors, *Handbook of Formal Argumentation*, chapter 14, pages 631–687. College Publications, 2018. also appears in IfCoLog Journal of Logics and their Applications 4(8):2557–2622.

[Dvořák and Wallner, 2020] Wolfgang Dvořák and Johannes Wallner. Computing strongly admissible sets. In H. Prakken, S. Bistarelli, F. Santini, and C. Taticchi, editors, *Computational Models of Argument - Proceedings of COMMA 2020, Perugia, Italy, September 4-11, 2020*, volume 326 of *Frontiers in Artificial Intelligence and Applications*, pages 179–190. IOS Press, 2020.

[Dvořák *et al.*, 2011] Wolfgang Dvořák, Sarah Gaggl, Johannes Wallner, and Stefan Woltran. Making use of advances in answer-set programming for abstract argumentation systems. In H. Tompits, S. Abreu, J. Oetsch, J. Pührer, D. Seipel, M. Umeda, and A. Wolf, editors, *Applications of Declarative Programming and Knowledge Management - 19th International Conference, INAP 2011, and 25th Workshop on Logic Programming, WLP 2011, Vienna, Austria, September 28-30, 2011, Revised Selected Papers*, volume 7773 of *Lecture Notes in Computer Science*, pages 114–133. Springer, 2011.

[Dvořák *et al.*, 2015] Wolfgang Dvořák, Sarah Gaggl, Thomas Linsbichler, and Johannes Wallner. Reduction-based approaches to implement Modgil's extended argumentation frameworks. In Th. Eiter, H. Strass, M. Truszczynski, and S. Woltran, editors, *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, volume 9060 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2015.

[Dvořák *et al.*, 2018] Wolfgang Dvořák, Alexander Greßler, and Stefan Woltran. Evaluating SETAFs via Answer-Set Programming. In *Proceedings of the Second International Workshop on Systems and Algorithms for Formal Argumentation*, pages 10–21. CEUR-WS.org, 2018.

[Dvořák *et al.*, 2020a] Wolfgang Dvořák, Sarah Gaggl, Anna Rapberger, Johannes Wallner, and Stefan Woltran. The ASPARTIX system suite. In H. Prakken, S. Bistarelli, F. Santini, and C. Taticchi, editors, *Computational Models of Argument - Proceedings of COMMA 2020, Perugia, Italy, September 4-11, 2020*, vol-

ume 326 of *Frontiers in Artificial Intelligence and Applications*, pages 461–462. IOS Press, 2020.

[Dvořák *et al.*, 2020b] Wolfgang Dvořák, Anna Rapberger, Johannes Wallner, and Stefan Woltran. ASPARTIX-V19 - an answer-set programming based system for abstract argumentation. In A. Herzig and J. Kontinen, editors, *Foundations of Information and Knowledge Systems - 11th International Symposium, FoIKS 2020, Dortmund, Germany, February 17-21, 2020, Proceedings*, volume 12012 of *Lecture Notes in Computer Science*, pages 79–89. Springer, 2020.

[Dvořák *et al.*, 2023a] Wolfgang Dvořák, Alexander Greßler, Anna Rapberger, and Stefan Woltran. The complexity landscape of claim-augmented argumentation frameworks. *Artif. Intell.*, 317:103873, 2023.

[Dvořák *et al.*, 2023b] Wolfgang Dvořák, Atefeh Keshavarzi Zafarghandi, and Stefan Woltran. Expressiveness of setafs and support-free adfs under 3-valued semantics. *J. Appl. Non Class. Logics*, 33(3-4):298–327, 2023.

[Dvořák *et al.*, 2023c] Wolfgang Dvořák, Anna Rapberger, and Stefan Woltran. A claim-centric perspective on abstract argumentation semantics: Claim-defeat, principles, and expressiveness. *Artif. Intell.*, 324:104011, 2023.

[Dvořák and Woltran, 2020] Wolfgang Dvořák and Stefan Woltran. Complexity of abstract argumentation under a claim-centric view. *Artificial Intelligence*, 285:103290, 2020.

[Egly *et al.*, 2008] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. ASPARTIX: implementing argumentation frameworks using answer-set programming. In M. de la Banda and E. Pontelli, editors, *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, volume 5366 of *Lecture Notes in Computer Science*, pages 734–738. Springer, 2008.

[Egly *et al.*, 2010] Uwe Egly, Sarah Gaggl, and Stefan Woltran. Answer-set programming encodings for argumentation frameworks. *Argument & Computation*, 1(2):147–177, 2010.

[Ellmauthaler and Strass, 2014] Stefan Ellmauthaler and Hannes Strass. The DIAMOND system for computing with abstract dialectical frameworks. In S. Parsons, N. Oren, Ch. Reed, and F. Cerutti, editors, *Computational Models of Argument - Proceedings of COMMA 2014, Atholl Palace Hotel, Scottish Highlands, UK, September 9-12, 2014*, volume 266 of *Frontiers in Artificial Intelligence and Applications*, pages 233–240. IOS Press, 2014.

[Ellmauthaler and Wallner, 2012] Stefan Ellmauthaler and Johannes Wallner. Evaluating abstract dialectical frameworks with ASP. In B. Verheij, S. Szeider, and S. Woltran, editors, *Computational Models of Argument - Proceedings of COMMA 2012, Vienna, Austria, September 10-12, 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 505–506. IOS Press, 2012.

[Faber and Woltran, 2009] Wolfgang Faber and Stefan Woltran. Manifold answer-

set programs for meta-reasoning. In E. Erdem, F. Lin, and T. Schaub, editors, *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings*, volume 5753 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2009.

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

[Gaggl and Woltran, 2013] Sarah Alice Gaggl and Stefan Woltran. The cf2 argumentation semantics revisited. *Journal of Logic and Computation*, 23(5):925–949, 2013.

[Gaggl *et al.*, 2015] Sarah Gaggl, Norbert Manthey, Alessandro Ronca, Johannes Wallner, and Stefan Woltran. Improved answer-set programming encodings for abstract argumentation. *Theory and Practice of Logic Programming*, 15(4-5):434–448, 2015.

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

[García and Simari, 2018] Alejandro Javier García and Guillermo Ricardo Simari. Argumentation based on logic programming. In *Handbook of formal argumentation*, pages 409–436. College Publications, 2018.

[Gebser *et al.*, 2013] Martin Gebser, Benjamin Kaufmann, Javier Romero, Ramón Otero, Torsten Schaub, and Philipp Wanko. Domain-specific heuristics in answer set programming. In M. desJardins and M. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*, pages 350–356. AAAI Press, 2013.

[Gebser *et al.*, 2015] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Marius Lindauer, Max Ostrowski, Javier Romero, Torsten Schaub, and Sven Thiele. *Potassco User Guide*. University of Potsdam, second edition edition, 2015.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.

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

[Hölldobler and Kencana Ramli, 2009] Steffen Hölldobler and Carroline Dewi Puspa Kencana Ramli. Logic programs under three-valued łukasiewicz semantics. In *Logic Programming: 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings 25*, pages 464–478. Springer, 2009.

[Kakas *et al.*, 1994] A. C. Kakas, P. Mancarella, and Phan Minh Dung. The accept-

ability semantics for logic programs. In *Proceedings of the Eleventh International Conference on Logic Programming*, page 504–519, Cambridge, MA, USA, 1994. MIT Press.

[König *et al.*, 2022] Matthias König, Anna Rapberger, and Markus Ulbricht. Just a matter of perspective. In F. Toni, S. Polberg, R. Booth, M. Caminada, and H. Kido, editors, *Computational Models of Argument - Proceedings of COMMA 2022, Cardiff, Wales, UK, 14-16 September 2022*, volume 353 of *Frontiers in Artificial Intelligence and Applications*, pages 212–223. IOS Press, 2022.

[Lehtonen *et al.*, 2017] Tuomo Lehtonen, Johannes Wallner, and Matti Järvisalo. From structured to abstract argumentation: Assumption-based acceptance via AF reasoning. In A. Antonucci, L. Cholvy, and O. Papini, editors, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty - 14th European Conference, ECSQARU 2017, Lugano, Switzerland, July 10-14, 2017, Proceedings*, volume 10369 of *Lecture Notes in Computer Science*, pages 57–68. Springer, 2017.

[Lehtonen *et al.*, 2021a] Tuomo Lehtonen, Johannes Wallner, and Matti Järvisalo. Declarative algorithms and complexity results for assumption-based argumentation. *Journal of Artificial Intelligence Research*, 71:265–318, 2021.

[Lehtonen *et al.*, 2021b] Tuomo Lehtonen, Johannes Wallner, and Matti Järvisalo. Harnessing incremental answer set solving for reasoning in assumption-based argumentation. *Theory and Practice of Logic Programming*, 21(6):717–734, 2021.

[Modgil and Prakken, 2013] Sanjay Modgil and Henry Prakken. A general account of argumentation with preferences. *Artificial Intellligence*, 195:361–397, 2013.

[Modgil and Prakken, 2014] Sanjay Modgil and Henry Prakken. The ASPIC+ framework for structured argumentation: a tutorial. *Argument & Computation*, 5:31–62, 2014. Special Issue: Tutorials on Structured Argumentation.

[Modgil and Prakken, 2018] Sanjay Modgil and Henry Prakken. Abstract rule-based argumentation. In *Handbook of Formal Argumentation*, volume 1. College Publications, 2018.

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

[Nieves *et al.*, 2008] Juan Carlos Nieves, Mauricio Osorio, and Ulises Cortés. Preferred extensions as stable models. *Theory and Practice of Logic Programming*, 8(4):527–543, 2008.

[Osorio *et al.*, 2010] Mauricio Osorio, Juan Carlos Nieves, and Ignasi Gómez-Sebastià. CF2-extensions as answer-set models. In P. Baroni, F. Cerutti, M. Giacomin, and G. Simari, editors, *Computational Models of Argument: Proceedings of COMMA 2010, Desenzano del Garda, Italy, September 8-10, 2010*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 391–402. IOS Press,

2010.

[Polberg, 2016] Sylwia Polberg. Understanding the abstract dialectical framework. In *European Conference on Logics in Artificial Intelligence*, pages 430–446. Springer, 2016.

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

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

[Sá and Alcântara, 2021a] Samy Sá and João Alcântara. 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, Prague, Czech Republic, September 21–24, 2021, Proceedings 16*, pages 159–172. Springer, 2021.

[Sá and Alcântara, 2021b] Samy Sá and João Alcântara. Assumption-based argumentation is logic programming with projection. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty: 16th European Conference, ECSQARU 2021, Prague, Czech Republic, September 21–24, 2021, Proceedings 16*, pages 173–186. Springer, 2021.

[Saccà and Zaniolo, 1997] Domenico Saccà and Carlo Zaniolo. Deterministic and non-deterministic stable models. *Journal of Logic and Computation*, 7(5):555–579, 1997.

[Sakama and Rienstra, 2017] Chiaki Sakama and Tjitze Rienstra. Representing argumentation frameworks in answer set programming. *Fundamenta Informaticae*, 155(3):261–292, 2017.

[Schultz and Toni, 2014] Claudia Schultz and Francesca Toni. Complete assumption labellings. In *Proceedings of COMMA 2014*, pages 405–412, 2014.

[Schulz and Toni, 2017] Claudia Schulz and Francesca Toni. Labellings for assumption-based and abstract argumentation. *International Journal of Approximate Reasoning*, 84:110–149, 2017.

[Simari and Loui, 1992] Guillermo Simari and Ronald Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53:125–157, 1992.

[Strass, 2013] Hannes Strass. Approximating operators and semantics for abstract dialectical frameworks. *Artificial Intelligence*, 205:39–70, 2013.

[Toni and Sergot, 2011] Francesca Toni and Marek Sergot. Argumentation and answer set programming. In Marcello Balduccini and Tran Cao Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays*

Dedicated to Michael Gelfond on the Occasion of His 65th Birthday, pages 164–180. Springer, 2011.

[Toni, 2014] Francesca Toni. A tutorial on assumption-based argumentation. *Argument & Computation*, 5:89–117, 2014. Special Issue: Tutorials on Structured Argumentation.

[Wakaki and Nitta, 2008] Toshiko Wakaki and Katsumi Nitta. Computing argumentation semantics in answer set programming. In H. Hattori, T. Kawamura, T. Idé, M. Yokoo, and Y. Murakami, editors, *New Frontiers in Artificial Intelligence, JSAI 2008 Conference and Workshops, Asahikawa, Japan, June 11-13, 2008, Revised Selected Papers*, volume 5447 of *Lecture Notes in Computer Science*, pages 254–269. Springer, 2008.

[Weydert, 2011] Emil Weydert. Semi-stable extensions for infinite frameworks. In P. de Causmaecker, J. Maervoet, T. Messelis, K. Verbeeck, and T. Vermeulen, editors, *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)*, pages 336–343, 2011.

[Wu *et al.*, 2009] Yining Wu, Martin Caminada, and Dov Gabbay. Complete extensions in argumentation coincide with 3-valued stable models in logic programming. *Studia Logica*, 93(1-2):383–403, 2009. Special issue: new ideas in argumentation theory.