Representing Argumentation Frameworks in Answer Set Programming

Chiaki Sakama
Department of Computer and Communication Sciences
Wakayama University, Japan
sakama@sys.wakayama-u.ac.jp

Tjitze Rienstra
Interdisciplinary Centre for Security, Reliability and Trust
University of Luxembourg, Luxembourg
tjitze@gmail.com

Abstract

This paper studies representation of argumentation frameworks (AFs) in answer set programming (ASP). Four different transformations from AFs to logic programs are provided under the complete semantics, stable semantics, grounded semantics and preferred semantics. The proposed transformations encode labelling-based argumentation semantics in a simple manner, and different semantics of AFs are uniformly characterized by stable models of transformed programs. We apply transformed programs to solving AF problems such as query-answering, enforcement of arguments, agreement or equivalence of different AFs. Logic programming encodings of AFs are also used for representing assumption-based argumentation (ABA) in ASP. The results of this paper exploit new connections between argumentation theory and logic programming, and enable one to perform various argumentation tasks using existing answer set solvers.

1 Introduction

Logic programming and argumentation theory are two different formalisms for representing knowledge, while close connections in their semantics have been revealed by several researchers [34]. Dung [13] introduces a transformation from logic programs (LPs) to (abstract) argumentation frameworks (AFs) and shows that stable models [21] (resp. the well-founded model [35]) of a logic program correspond to stable extensions (resp. the grounded extension) of a transformed argumentation framework. Further results show that similar correspondences exist between 3-valued stable models of LPs [29] and complete extensions of AFs [38], and regular models of LPs [39] and preferred extensions of AFs [8]. In addition, Dung [13] introduces a converse transformation from AFs to LPs, and shows that stable extensions (resp. the grounded

extension) of an AF correspond to stable models (resp. the well-founded model) of a transformed logic program. Similar results are known that relate preferred, complete, or other extensions of AFs to their corresponding semantics of transformed logic programs [12, 18, 19, 20, 25, 36, 38]. In his transformation from AFs to LPs, Dung uses an LP as a meta-interpreter for computing arguments. This idea is inherited by studies [18, 19, 20, 36] which use LPs as meta-interpreters for processing AFs given as input. An important difference is that Dung characterizes different semantics of AFs in terms of different semantics of LPs, while those studies [18, 19, 20, 36] characterize different semantics of AFs in terms of the answer set semantics of LPs. The latter type of characterization is important in the sense that it enables one to use existing solvers for *answer set programming* (ASP) [6] for computing different semantics of AFs. On the other hand, encoding the semantics of AFs in meta-interpretative LPs results in quite complicated programs, especially for problems under the preferred semantics that are located at the second level of the polynomial hierarchy, and the resulting programs are hardly accessible for non-experts in ASP [18].

In this paper, we introduce transformation schemes from AFs to LPs such that different semantics of AFs (i.e., complete, stable, grounded, preferred) are characterized by the answer set semantics of LPs. The proposed schemes are *simple* in the sense that encodings are easily readable and are relatively small in size, and they are also *uniform* in the sense that encodings are applied to different semantics of AFs with small modifications. Different from [18, 19, 20, 36], we do not take the meta-interpretative approach, but translate an argumentation framework into a logic program at the object level. In the object level transformation, arguments and attack relations in an AF are directly represented by rules in a transformed program, and different AFs produce different LPs in general. This viewpoint is similar to a translation given in [8, 38], where arguments and attack relations are encoded by rules of logic programs. The translation given in [8, 38], however, maps different semantics of AFs into different semantics of LPs, so that it does not address the computation of different semantics of AFs in terms of ASP. Among the meta-interpretative ASP approaches, [18, 19, 20] compute extension-based semantics of AFs, while [36] computes labelling-based semantics of AFs. It is known that there is a one-to-one correspondence between extension-based semantics and labelling-based semantics of AFs [7]. While an extension-based semantics results in sets of accepted arguments, a labelling-based semantics assigns to each argument one of three labels accepted, rejected or undecided, and thus permits the distinction between rejected and undecided arguments. In this paper, we compute labelling-based semantics of AFs in terms of ASP. We next address applications of our transformation scheme in solving several argumentation-related problems. These are (i) query-answering, (ii) enforcement of arguments, (iii) agreement among different AFs, and (iv) equivalence of AFs. We also apply our scheme for representing assumption-based argumentation (ABA) [5, 15] in ASP. We compare our encodings with existing ASP-encodings in the literature and show that our encoding is more compact than others.

The rest of this paper is organized as follows. In Section 2 we review the necessary basic notions of argumentation frameworks and answer set programming. In Section 3 we introduce transformation schemes from AFs to LPs under the complete, stable, grounded and preferred semantics. Section 4 addresses several applications of

the transformation schemes. Section 5 compares the proposed encoding with existing ASP encodings in the literature. Section 6 discusses related issues and Section 7 summarizes the paper.

2 Preliminaries

2.1 Argumentation Framework

In this section we introduce the necessary basics concerning abstract argumentation frameworks. The definitions presented here are based on [7, 13].

Definition 2.1 (argumentation framework) Let U be the universe of all possible *arguments*. An *argumentation framework* (AF) is a pair (Ar, att) where Ar is a finite subset of U and $att \subseteq Ar \times Ar$. We say that a *attacks* b iff $(a, b) \in att$. For $x \in Ar$, define $x^- = \{\, y \mid (y, x) \in att \,\}$.

An argumentation framework (Ar, att) is represented by a directed graph in which vertices are arguments in Ar and a directed arc from a to b exists whenever $(a, b) \in att$.

Definition 2.2 (labelling) A *labelling* of $AF = (Ar, att)$ is a (total) function $\mathcal{L} : Ar \to \{\, \mathtt{in}, \mathtt{out}, \mathtt{und} \,\}$.

When $\mathcal{L}(a) = \mathtt{in}$ (resp. $\mathcal{L}(a) = \mathtt{out}$ or $\mathcal{L}(a) = \mathtt{und}$) for an argument $a \in Ar$, it is written as $\mathtt{in}(a)$ (resp. $\mathtt{out}(a)$ or $\mathtt{und}(a)$). In this case, the argument a is said to be *accepted* (resp. *rejected* or *undecided*) in $\mathcal{L}$. We call $\mathtt{in}(a)$, $\mathtt{out}(a)$ and $\mathtt{und}(a)$ *labelled arguments*. A labelling $\mathcal{L}$ of $AF = (Ar, att)$ is also represented as a set of labelled arguments $S = \{\, \ell(x) \mid \mathcal{L}(x) = \ell$ for $x \in Ar$ where $\ell \in \{\mathtt{in}, \mathtt{out}, \mathtt{und}\} \,\}$.

Definition 2.3 (complete labelling) A labelling $\mathcal{L}$ of $AF = (Ar, att)$ is a *complete labelling* if for each argument $a \in Ar$, it holds that:

- $\mathcal{L}(a) = \mathtt{in}$ iff $\mathcal{L}(b) = \mathtt{out}$ for every $b \in Ar$ such that $(b, a) \in att$.
- $\mathcal{L}(a) = \mathtt{out}$ iff $\mathcal{L}(b) = \mathtt{in}$ for some $b \in Ar$ such that $(b, a) \in att$.

By definition it follows that an argument is undecided in a complete labelling if and only if none of its attackers is labelled `in`, and at least one of its attackers is labelled `und`.

Definition 2.4 (stable, grounded, preferred labelling) Let $\mathcal{L}$ be a complete labelling of $AF = (Ar, att)$. Put $\mathtt{in}(\mathcal{L}) = \{\, x \mid \mathcal{L}(x) = \mathtt{in}$ for $x \in Ar \,\}$, $\mathtt{out}(\mathcal{L}) = \{\, x \mid \mathcal{L}(x) = \mathtt{out}$ for $x \in Ar \,\}$ and $\mathtt{und}(\mathcal{L}) = \{\, x \mid \mathcal{L}(x) = \mathtt{und}$ for $x \in Ar \,\}$.

- $\mathcal{L}$ is a *stable labelling* iff $\mathtt{und}(\mathcal{L}) = \emptyset$.
- $\mathcal{L}$ is a *grounded labelling* iff $\mathtt{in}(\mathcal{L})$ is minimal with respect to set inclusion among all complete labellings of AF.

- $\mathcal{L}$ is a *preferred labelling* iff $\mathtt{in}(\mathcal{L})$ is maximal with respect to set inclusion among all complete labellings of AF.

The grounded or preferred labelling is also characterized using `out` and `und` as follows [7].

- $\mathcal{L}$ is a grounded labelling iff $\mathtt{out}(\mathcal{L})$ is minimal with respect to set inclusion among all complete labellings of AF iff $\mathtt{und}(\mathcal{L})$ is maximal with respect to set inclusion among all complete labellings of AF.
- $\mathcal{L}$ is a preferred labelling iff $\mathtt{out}(\mathcal{L})$ is maximal with respect to set inclusion among all complete labellings of AF.[1]

There is a one-to-one correspondence between the set $\mathtt{in}(\mathcal{L})$ of a complete (resp. stable, grounded, preferred) labelling $\mathcal{L}$ of AF and a *complete* (resp. *stable*, *grounded*, *preferred*) *extension* of AF [7].

2.2 Answer Set Programming

A *logic program* (LP) considered in this paper is a finite set of *rules* of the form:

$$a_1 \vee \cdots \vee a_l \ \leftarrow \ a_{l+1}, \ldots, a_m, \mathbf{not}\, a_{m+1}, \ldots, \mathbf{not}\, a_n \quad (n \geq m \geq l \geq 0) \qquad (1)$$

where each a_i is a ground atom. $\mathbf{not}$ is *negation as failure* (NAF) and $\mathbf{not}\, a$ is called an *NAF-literal*. The left-hand side of $\leftarrow$ is the *head*, and the right-hand side is the *body*. For each rule r of the above form, $head(r)$, $body^+(r)$, and $body^-(r)$ denote the sets of atoms $\{a_1, \ldots, a_l\}$, $\{a_{l+1}, \ldots, a_m\}$, and $\{a_{m+1}, \ldots, a_n\}$, respectively. A rule r is a *constraint* if $head(r) = \emptyset$; and r is a (*disjunctive*) *fact* if $body^+(r) = body^-(r) = \emptyset$. We often write a rule with variables as a shorthand of its ground instances. A logic program is simply called a *program*. A program P is called a *normal program* if $\mid head(r) \mid \leq 1$ for every rule r in P. A program P is called a *positive program* if $body^-(r) = \emptyset$ for every rule r in P. A positive program P is called a *Horn program* if $|\, head(r)\,| \leq 1$ for every rule r in P.

The semantics of a program is defined by the *stable model semantics* (or *answer set semantics*) [21, 22]. Let B be the *Herbrand base* of a program. An interpretation $I \subseteq B$ *satisfies* a rule r of the form (1) if $body^+(r) \subseteq I$ and $body^-(r) \cap I = \emptyset$ imply $head(r) \cap I \neq \emptyset$. In particular, I satisfies a constraint r such that $head(r) = \emptyset$ if $body^+(r) \setminus I \neq \emptyset$ or $body^-(r) \cap I \neq \emptyset$. An interpretation satisfying every rule in a program is a *model* of the program. Given a positive program P, a model M of P is *minimal* if there is no model N of P such that $N \subset M$. Given a program P, an interpretation I is a *stable model* of P if it coincides with a minimal model of the positive program (called a *reduct* of P with respect to I): $P^I = \{\, a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m \mid (a_1 \vee \cdots \vee a_l \ \leftarrow \ a_{l+1}, \ldots, a_m, \mathbf{not}\, a_{m+1}, \ldots, \mathbf{not}\, a_n) \in P \text{ and } \{a_{m+1}, \ldots, a_n\} \cap I = \emptyset \,\}$. Stable models coincide with minimal models in a positive program. A program may have no, one, or multiple stable models in

[1] A labelling $\mathcal{L}$ in which $\mathtt{und}(\mathcal{L})$ is minimal with respect to set inclusion among all complete labellings of AF is not a preferred labelling in general. Instead, such a labelling is a *semi-stable labelling* [7].

general. A program is *consistent* if it has at least one stable model; otherwise, the program is *inconsistent*. A stable model is also called an *answer set*. Representing knowledge by logic programs under the stable model semantics is called *answer set programming* (ASP) [6]. In this paper, we use the terms logic programming and answer set programming interchangeably.

3 Transforming AFs to LPs

In this section we present transformations from AFs to LPs under the complete, stable, grounded and preferred semantics. We first introduce some basic concepts used in these transformations. Given an argumentation framework $AF = (Ar, att)$, we define B_{Ar} to be the set of all labelled arguments in AF:

$$B_{Ar} = \{\, \mathtt{in}(x),\ \mathtt{out}(x),\ \mathtt{und}(x) \mid x \in Ar \,\}.$$

We view Ar as a set of constants and consider B_{Ar} as the *Herbrand base* on which a logic program is constructed. The following definition introduces a transformation of an argumentation framework AF into a set Γ_{AF} of rules.

Definition 3.1 (rules for AF) Given $AF = (Ar, att)$, the set Γ_{AF} of rules is defined as follows:

$$\begin{aligned}
\Gamma_{AF} \ = \ & \{\, \mathtt{in}(x) \leftarrow \mathtt{out}(y_1), \ldots, \mathtt{out}(y_k) \\
& \qquad \mid x \in Ar \text{ and } x^- = \{\, y_1, \ldots, y_k \} \, (k \geq 0) \,\} && (2) \\
& \cup \{\, \mathtt{out}(x) \leftarrow \mathtt{in}(y) \mid (y, x) \in att \,\} && (3) \\
& \cup \{\, \leftarrow \mathtt{in}(x),\ \mathbf{not}\ \mathtt{out}(y) \mid (y, x) \in att \,\} && (4) \\
& \cup \{\, \leftarrow \mathtt{out}(x),\ \mathbf{not}\ \mathtt{in}(y_1), \ldots, \mathbf{not}\ \mathtt{in}(y_k) \\
& \qquad \mid x \in Ar \text{ and } x^- = \{\, y_1, \ldots, y_k \} \, (k \geq 0) \,\}. && (5)
\end{aligned}$$

The rule (2) states that an argument x is labelled `in` if every attacker $y_1, \ldots, y_k$ of x is labelled `out`. The rule (3) states that an argument x is labelled `out` if there is an attacker y which is labelled `in`. The constraint (4) states that every `in`-labelled argument x has no attacker y which is not labelled `out`. The constraint (5) states that every `out`-labelled argument x has at least one attacker y_i $(1 \leq i \leq k)$ which is labelled `in`. (4) and (5) represent the *admissibility* condition of [7]. Note that when an argument x has no attacker ($x^- = \emptyset$), the rule (2) becomes the fact

$$\mathtt{in}(x) \leftarrow$$

which states that the argument x, which has no attackers, is always labelled `in`. Also when $x^- = \emptyset$, the rule (5) becomes the constraint

$$\leftarrow \mathtt{out}(x)$$

which states that the argument x, which has no attackers, cannot be labelled `out`.

The program Γ_{AF} serves as the core in encoding different semantics of AFs in LPs as set out in the remainder of this section.

3.1 Complete Semantics

The transformation of AFs to LPs under the complete semantics is captured by the following definition.

Definition 3.2 (AF program under the complete semantics) Given $AF = (Ar, att)$, an *AF-program under the complete semantics* Π^C_{AF} is defined as follows:

$$\Pi^C_{AF} = \Gamma_{AF} \cup \{\, \mathtt{in}(x) \vee \mathtt{out}(x) \vee \mathtt{und}(x) \leftarrow \mid x \in Ar \,\} \quad (6)$$

$$\cup \{\, \leftarrow \mathtt{in}(x), \mathtt{out}(x) \mid x \in Ar \,\}. \quad (7)$$

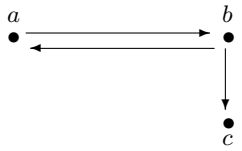
The rules of Γ_{AF} represent the necessary and sufficient condition of `in` or `out` labellings under the complete labelling of Definition 2.3. In addition, the disjunctive fact (6) states that every argument is labelled by either `in`, `out` or `und`. The constraints (7) state that each argument cannot take `in` and `out` labellings at the same time. Note that we do not need constraints such as "$\leftarrow \mathtt{in}(x), \mathtt{und}(x)$" or "$\leftarrow \mathtt{out}(x), \mathtt{und}(x)$". This is because $\mathtt{und}(x)$ does not appear in any rule except (6), so any model containing both $\mathtt{in}(a)$ and $\mathtt{und}(a)$ (or $\mathtt{out}(a)$ and $\mathtt{und}(a)$) for some $a \in Ar$ is not minimal. We will consider stable models of Π^C_{AF} that are minimal, and those non-minimal models are automatically excluded. The transformed program encodes the complete labelling of an AF.[2]

Theorem 3.1 *Let $AF = (Ar, att)$ be an argumentation framework and Π^C_{AF} an AF-program under the complete semantics. Then the complete labellings of AF coincide with the stable models of Π^C_{AF}.*

Proof: Suppose a complete labelling $\mathcal{L}$ of AF. By Definition 2.3, $\mathcal{L}(x) = \mathtt{in}$ for $x \in Ar$ if $\mathcal{L}(y) = \mathtt{out}$ for every $y \in x^-$. This sufficient condition of $\mathcal{L}(x) = \mathtt{in}$ is represented by the rule (2) of Γ_{AF}. On the other hand, the necessary condition of $\mathcal{L}(x) = \mathtt{in}$ is stated as "if $\mathcal{L}(x) = \mathtt{in}$ for $x \in Ar$ then $\mathcal{L}(y) = \mathtt{out}$ for every $y \in x^-$", which is rephrased by the statement "it is impossible to be $\mathcal{L}(x) = \mathtt{in}$ for $x \in Ar$ and $\mathcal{L}(y) \neq \mathtt{out}$ for some $y \in x^-$". The statement is represented by the constraint (4) of Γ_{AF}. Likewise, $\mathcal{L}(x) = \mathtt{out}$ for $x \in Ar$ iff $\mathcal{L}(y) = \mathtt{in}$ for some $y \in x^-$. The sufficient condition of $\mathcal{L}(x) = \mathtt{out}$ is represented by the rule (3) of Γ_{AF}, and the necessary condition of $\mathcal{L}(x) = \mathtt{out}$ is represented by the constraint (5) of Γ_{AF}. The fact that $\mathcal{L}$ is a total function from Ar to $\{\, \mathtt{in}, \mathtt{out}, \mathtt{und} \,\}$ is represented by the disjunctive facts (6) and the constraints (7). Thus, $\mathcal{L}(a) = \mathtt{in}$ (resp. $\mathcal{L}(a) = \mathtt{out}$) for $a \in Ar$ iff $\mathtt{in}(a) \in M$ (resp. $\mathtt{out}(a) \in M$) for some stable model M of Π^C_{AF}. Finally, $\mathcal{L}(x) = \mathtt{und}$ for $x \in Ar$ iff $\mathcal{L}(x) \neq \mathtt{in}$ and $\mathcal{L}(x) \neq \mathtt{out}$. If Π^C_{AF} has a stable model M such that $\mathtt{in}(a) \notin M$ and $\mathtt{out}(a) \notin M$ for some $a \in Ar$, then $\mathtt{und}(a) \in M$ by the disjunctive fact (6). On the other hand, if Π^C_{AF} has a stable model M such that $\mathtt{in}(a) \in M$ or $\mathtt{out}(a) \in M$ for some $a \in Ar$, then M satisfies the disjunctive fact (6) for $x = a$ thereby $\mathtt{und}(a) \notin M$ by the minimality of stable models. Thus, for any $a \in Ar$, $\mathtt{und}(a) \in M$ iff neither $\mathtt{in}(a) \notin M$ nor $\mathtt{out}(a) \notin M$ for any stable model M of Π^C_{AF}. Hence, the result holds. □

[2]The program Π^C_{AF} can be replaced by a normal program. We will argue the issue in Section 6.1.

Example 3.1 Suppose $AF = (\{a, b, c\}, \{(a, b), (b, a), (b, c)\})$.



Then Π^C_{AF} consists of the rules:

$$\begin{array}{l}
\texttt{in}(a) \leftarrow \texttt{out}(b), \quad \texttt{in}(b) \leftarrow \texttt{out}(a), \quad \texttt{in}(c) \leftarrow \texttt{out}(b), \\
\texttt{out}(a) \leftarrow \texttt{in}(b), \quad \texttt{out}(b) \leftarrow \texttt{in}(a), \quad \texttt{out}(c) \leftarrow \texttt{in}(b), \\
\leftarrow \texttt{in}(a), \textbf{not}\, \texttt{out}(b), \quad \leftarrow \texttt{in}(b), \textbf{not}\, \texttt{out}(a), \quad \leftarrow \texttt{in}(c), \textbf{not}\, \texttt{out}(b), \\
\leftarrow \texttt{out}(a), \textbf{not}\, \texttt{in}(b), \quad \leftarrow \texttt{out}(b), \textbf{not}\, \texttt{in}(a), \quad \leftarrow \texttt{out}(c), \textbf{not}\, \texttt{in}(b), \\
\texttt{in}(x) \vee \texttt{out}(x) \vee \texttt{und}(x) \leftarrow, \quad \leftarrow \texttt{in}(x), \texttt{out}(x) \quad \text{where } x \in \{a, b, c\}.
\end{array}$$

Π^C_{AF} has the three stable models:

$$\{\, \texttt{in}(a), \texttt{out}(b), \texttt{in}(c) \,\}, \ \{\, \texttt{out}(a), \texttt{in}(b), \texttt{out}(c) \,\}, \ \{\, \texttt{und}(a), \texttt{und}(b), \texttt{und}(c) \,\}$$

which coincide with the three complete labellings of AF.

3.2 Stable Semantics

The following definition captures the transformation of AFs to LPs under the stable semantics.

Definition 3.3 (AF program under the stable semantics) Given $AF = (Ar, att)$, an *AF-program under the stable semantics* Π^S_{AF} is defined as follows.

$$\Pi^S_{AF} \ = \ \Gamma_{AF} \quad \cup \quad \{\, \texttt{in}(x) \vee \texttt{out}(x) \leftarrow \mid \ x \in Ar \,\} \tag{8}$$

$$\cup \quad \{\, \leftarrow \texttt{in}(x), \texttt{out}(x) \ \mid \ x \in Ar \,\}. \tag{9}$$

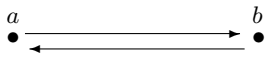
In contrast to Π^C_{AF}, the program Π^S_{AF} introduces disjunctive facts (8). This is because every argument in a stable labelling is either `in` or `out` (but not both).

Theorem 3.2 *Let $AF = (Ar, att)$ be an argumentation framework and Π^S_{AF} an AF-program under the stable semantics. Then the stable labellings of AF coincide with the stable models of Π^S_{AF}.*

Proof: A stable labelling is a complete labelling $\mathcal{L}$ such that $\texttt{und}(\mathcal{L}) = \emptyset$. Then the result follows by Theorem 3.1. □

Corollary 3.3 *$AF = (Ar, att)$ has no stable labelling iff Π^S_{AF} is inconsistent.*

Example 3.2 Suppose $AF_1 = (\{a, b\}, \{(a, b), (b, a)\})$.

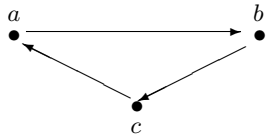


Then $\Pi^S_{AF_1}$ consists of rules:

$$\begin{array}{l}
\mathtt{in}(a) \leftarrow \mathtt{out}(b), \quad \mathtt{in}(b) \leftarrow \mathtt{out}(a), \quad \mathtt{out}(a) \leftarrow \mathtt{in}(b), \\
\mathtt{out}(b) \leftarrow \mathtt{in}(a), \quad \leftarrow \mathtt{in}(a), \mathbf{not}\ \mathtt{out}(b), \quad \leftarrow \mathtt{in}(b), \mathbf{not}\ \mathtt{out}(a), \\
\leftarrow \mathtt{out}(a), \mathbf{not}\ \mathtt{in}(b), \quad \leftarrow \mathtt{out}(b), \mathbf{not}\ \mathtt{in}(a), \\
\mathtt{in}(a) \vee \mathtt{out}(a) \leftarrow, \quad \mathtt{in}(b) \vee \mathtt{out}(b) \leftarrow, \\
\leftarrow \mathtt{in}(a), \mathtt{out}(a), \quad \leftarrow \mathtt{in}(b), \mathtt{out}(b).
\end{array}$$

$\Pi^S_{AF_1}$ has the two stable models $\{\, \mathtt{in}(a),\ \mathtt{out}(b) \,\}$ and $\{\, \mathtt{out}(a),\ \mathtt{in}(b) \,\}$ which coincide with the two stable labellings of AF.

Next suppose $AF_2 = (\{a, b, c\}, \{(a, b), (b, c), (c, a)\})$.



Then $\Pi^S_{AF_2}$ consists of rules:

$$\begin{array}{l}
\mathtt{in}(a) \leftarrow \mathtt{out}(c), \quad \mathtt{in}(b) \leftarrow \mathtt{out}(a), \quad \mathtt{in}(c) \leftarrow \mathtt{out}(b), \\
\mathtt{out}(a) \leftarrow \mathtt{in}(c), \quad \mathtt{out}(b) \leftarrow \mathtt{in}(a), \quad \mathtt{out}(c) \leftarrow \mathtt{in}(b), \\
\leftarrow \mathtt{in}(a), \mathbf{not}\ \mathtt{out}(c), \quad \leftarrow \mathtt{in}(b), \mathbf{not}\ \mathtt{out}(a), \quad \leftarrow \mathtt{in}(c), \mathbf{not}\ \mathtt{out}(b), \\
\leftarrow \mathtt{out}(a), \mathbf{not}\ \mathtt{in}(c), \quad \leftarrow \mathtt{out}(b), \mathbf{not}\ \mathtt{in}(a), \quad \leftarrow \mathtt{out}(c), \mathbf{not}\ \mathtt{in}(b), \\
\mathtt{in}(a) \vee \mathtt{out}(a) \leftarrow, \quad \mathtt{in}(b) \vee \mathtt{out}(b) \leftarrow, \quad \mathtt{in}(c) \vee \mathtt{out}(c) \leftarrow, \\
\leftarrow \mathtt{in}(a), \mathtt{out}(a), \quad \leftarrow \mathtt{in}(b), \mathtt{out}(b), \quad \leftarrow \mathtt{in}(c), \mathtt{out}(c).
\end{array}$$

$\Pi^S_{AF_2}$ is inconsistent (having no stable model) and AF_2 has no stable labelling.

3.3 Grounded Semantics

The transformation of AFs to LPs under the grounded semantics is captured by the following definition.

Definition 3.4 (AF program under the grounded semantics) Given $AF = (Ar, att)$, an *AF-program under the grounded semantics* Π^G_{AF} is defined as follows.

$$\Pi^G_{AF} \ = \ \Gamma_{AF} \ \cup \ \{\, \mathtt{und}(x) \leftarrow \mathbf{not}\ \mathtt{in}(x), \mathbf{not}\ \mathtt{out}(x) \mid \ x \in Ar \,\}. \qquad (10)$$

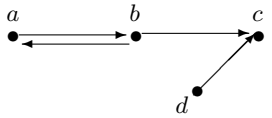
Unlike Π^C_{AF} or Π^S_{AF}, Π^G_{AF} adds no disjunctive facts or constraints to Γ_{AF}. Instead, it includes a rule (10) which states that an argument x is labelled $\mathtt{und}$ if neither $\mathtt{in}(x)$ nor $\mathtt{out}(x)$ is derived in Γ_{AF}. The grounded semantics accepts/rejects only the arguments that one cannot avoid to accept/reject, and abstains as much as possible [2]. The situation is represented by Π^G_{AF} in which $\mathtt{in}(x)$ or $\mathtt{out}(x)$ holds only if it is derived in Γ_{AF}, otherwise, $\mathtt{und}(x)$ holds.

Theorem 3.4 *Let $AF = (Ar, att)$ be an argumentation framework and Π^G_{AF} an AF-program under the grounded semantics. Then the grounded labelling of AF coincides with the stable model of Π^G_{AF}.*

Proof: Let M be the grounded labelling of AF. Since M is also a complete labelling of AF, M is a stable model of Π^C_{AF} (Theorem 3.1) thereby satisfies every rule in Γ_{AF}. Then $(\Gamma_{AF})^M$ (the reduct of Γ_{AF} with respect to M) is a Horn program and $M \setminus \{\,\mathtt{und}(x) \mid x \in Ar\}$ is the unique stable model of Γ_{AF}. Since $\mathtt{und}(x) \in M$ iff $\mathtt{in}(x) \notin M$ and $\mathtt{out}(x) \notin M$ for any $x \in Ar$, M becomes the unique stable model of Π^G_{AF}. Conversely, let M be a stable model of Π^G_{AF}. M satisfies the disjunctive facts (6) by the rule (10). Suppose that M does not satisfies the constraints (7), and $\mathtt{in}(a) \in M$ and $\mathtt{out}(a) \in M$ for some $a \in Ar$. Then $M' = M \setminus \{\,\mathtt{und}(x) \mid x \in Ar\}$ is the least model of $(\Gamma_{AF})^M$, and satisfies two rules $r_1 =$ '$\mathtt{in}(a) \leftarrow \mathtt{out}(b_1), \ldots, \mathtt{out}(b_k)$' where $a^- = \{b_1, \ldots, b_k\}$ and $r_2 =$ '$\mathtt{out}(a) \leftarrow \mathtt{in}(b_j)$' where $(b_j, a) \in att$. $\mathtt{in}(a) \in M'$ implies $\mathtt{out}(b_i) \in M'$ $(1 \leq i \leq k)$ for every $(b_i, a) \in att$, and $\mathtt{out}(a) \in M'$ implies $\mathtt{in}(b_j) \in M'$ for some $(b_j, a) \in att$ $(1 \leq j \leq k)$. Then there is b_j such that $\mathtt{in}(b_j) \in M'$ and $\mathtt{out}(b_j) \in M'$. In this case, however, $M' \setminus \{\mathtt{in}(b_j), \mathtt{out}(b_j), \mathtt{in}(a), \mathtt{out}(a)\}$ also satisfies r_1 and r_2, so M' is not the least model of $(\Gamma_{AF})^M$. Contradiction. Thus, M satisfies the constraints (7), and M is a stable model of Π^C_{AF}. Then M is a complete labelling $\mathcal{L}$ of AF (Theorem 3.1). To see that M is the grounded labelling of AF, we show that M is minimal with respect to $\mathtt{in}$-labelling among all complete labellings of AF. Since M is a stable model of Π^G_{AF}, $M' = M \setminus \{\,\mathtt{und}(x) \mid x \in Ar\}$ satisfies the constraints (4) and (5) of Γ_{AF} and is the least model of the program $(\Gamma_{AF})^M$. Suppose that M is not the grounded labelling of AF and there is the grounded labelling N of AF such that $\mathtt{in}(N) \subset \mathtt{in}(M)$ where $\mathtt{in}(M) = \{\mathtt{in}(x) \mid \mathtt{in}(x) \in M\}$. Since N is also a complete labelling of AF, N is a stable model of Π^C_{AF} and satisfies every rule in Γ_{AF}. Then $N' = N \setminus \{\,\mathtt{und}(x) \mid x \in Ar\}$ also satisfies the constraints (4) and (5) of Γ_{AF} and is the least model of the program $(\Gamma_{AF})^N$. By $\mathtt{out}(N) \subseteq \mathtt{out}(M)$ where $\mathtt{out}(M) = \{\mathtt{out}(x) \mid \mathtt{out}(x) \in M\}$, it holds that $(\Gamma_{AF})^M \subseteq (\Gamma_{AF})^N$. Since N' is also the least model of $(\Gamma_{AF})^M$, $(\Gamma_{AF})^M$ has two different least models M' and N'. Contradiction. Hence, M is minimal with respect to $\mathtt{in}$-labelling, and M is the grounded labelling of AF. □

Corollary 3.5 *Π^G_{AF} always has a single stable model.*

Example 3.3 Suppose $AF = (\{a, b, c, d\}, \{(a, b), (b, a), (b, c), (d, c)\})$.



Then Π^G_{AF} consists of rules:

$\mathtt{in}(a) \leftarrow \mathtt{out}(b), \quad \mathtt{in}(b) \leftarrow \mathtt{out}(a), \quad \mathtt{in}(c) \leftarrow \mathtt{out}(b), \mathtt{out}(d), \quad \mathtt{in}(d) \leftarrow,$
$\mathtt{out}(a) \leftarrow \mathtt{in}(b), \quad \mathtt{out}(b) \leftarrow \mathtt{in}(a), \quad \mathtt{out}(c) \leftarrow \mathtt{in}(b),$
$\mathtt{out}(c) \leftarrow \mathtt{in}(d), \quad \leftarrow \mathtt{in}(a), \mathbf{not}\ \mathtt{out}(b), \quad \leftarrow \mathtt{in}(b), \mathbf{not}\ \mathtt{out}(a),$
$\leftarrow \mathtt{in}(c), \mathbf{not}\ \mathtt{out}(b), \quad \leftarrow \mathtt{in}(c), \mathbf{not}\ \mathtt{out}(d), \quad \leftarrow \mathtt{out}(a), \mathbf{not}\ \mathtt{in}(b),$
$\leftarrow \mathtt{out}(b), \mathbf{not}\ \mathtt{in}(a), \quad \leftarrow \mathtt{out}(c), \mathbf{not}\ \mathtt{in}(b), \mathbf{not}\ \mathtt{in}(d), \quad \leftarrow \mathtt{out}(d),$
$\mathtt{und}(x) \leftarrow \mathbf{not}\ \mathtt{in}(x), \mathbf{not}\ \mathtt{out}(x)$ where $x \in \{a, b, c, d\}$.

Π^G_{AF} has the unique stable model $\{\, \mathtt{und}(a), \mathtt{und}(b), \mathtt{out}(c), \mathtt{in}(d) \,\}$ which coincides with the grounded labelling of AF.

3.4 Preferred Semantics

To transform AFs to LPs under the preferred semantics, we need to extend the Herbrand base as follows. Given an argumentation framework $AF = (Ar, att)$, let

$$B^P_{Ar} = \{\, \mathtt{in}(x), \mathtt{out}(x), \mathtt{IN}(x), \mathtt{OUT}(x), \mathtt{UND}(x) \mid x \in Ar \,\}.$$

To transform AFs to LPs under the preferred semantics, we construct a logic program over B^P_{Ar}.

Definition 3.5 (AF program under the preferred semantics) Given $AF = (Ar, att)$, an *AF-program under the preferred semantics* Π^P_{AF} is defined as follows.

$$\begin{aligned}
\Pi^P_{AF} \;=\; \Gamma_{AF} \quad &\cup \quad \{\, \mathtt{in}(x) \vee \mathtt{out}(x) \leftarrow \mid x \in Ar \,\} \\
&\cup \quad \{\, \mathtt{IN}(x) \leftarrow \mathtt{in}(x), \mathbf{not}\ \mathtt{out}(x) \mid x \in Ar \,\} \qquad (11) \\
&\cup \quad \{\, \mathtt{OUT}(x) \leftarrow \mathbf{not}\ \mathtt{in}(x), \mathtt{out}(x) \mid x \in Ar \,\} \qquad (12) \\
&\cup \quad \{\, \mathtt{UND}(x) \leftarrow \mathtt{in}(x), \mathtt{out}(x) \mid x \in Ar \,\}. \qquad (13)
\end{aligned}$$

In contrast to Π^S_{AF}, constraints $\leftarrow \mathtt{in}(x), \mathtt{out}(x)$ are not included in Π^P_{AF}. The rule (11) (called IN-rule) means that an argument x has an $\mathtt{IN}$-labelling under the preferred semantics if it is labelled $\mathtt{in}$ under the stable labelling; while the rule (12) (called OUT-rule) means that an argument x has an $\mathtt{OUT}$-labelling under the preferred semantics if it is labelled $\mathtt{out}$ under the stable labelling. On the other hand, the rule (13) (called UND-rule) means that an argument x has an $\mathtt{UND}$-labelling under the preferred semantics if it does not have a consistent stable labelling. This would happen when selecting $\mathtt{in}(x)$ (resp. $\mathtt{out}(x)$) in a disjunctive fact leads to derive $\mathtt{out}(x)$ (resp. $\mathtt{in}(x)$) in Γ_{AF}. In this case, $\mathtt{UND}(x)$ is produced by (13). Π^P_{AF} introduces these IN-OUT-UND rules to Π^S_{AF} instead of constraints (9) of Definition 3.3. Given $AF = (Ar, att)$ and a set of labelled arguments $S \subseteq B_{Ar}$, we define the set $M_S \subseteq B^P_{Ar}$ as $M_S = (\, S \setminus \{\mathtt{und}(x) \mid x \in Ar\}) \cup \{\mathtt{in}(x), \mathtt{out}(x) \mid \mathtt{und}(x) \in S\} \cup \{\, \mathtt{IN}(x) \mid \mathtt{in}(x) \in S \,\} \cup \{\, \mathtt{OUT}(x) \mid \mathtt{out}(x) \in S \,\} \cup \{\, \mathtt{UND}(x) \mid \mathtt{und}(x) \in S \,\}$. We show that there is a one-to-one correspondence between the preferred labellings of AF and the stable models of Π^P_{AF}.

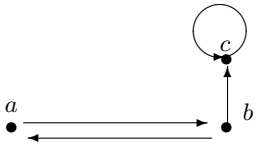
Theorem 3.6 *Let $AF = (Ar, att)$ be an argumentation framework and Π^P_{AF} an AF-program under the preferred semantics. Then S is a preferred labelling of AF iff M_S is a stable model of Π^P_{AF}.*

Proof: Suppose that S is a preferred labelling of AF. Then S is either a stable labelling or not. (i) If S is a stable labelling of AF, then S is a stable model of Π^S_{AF} (Theorem 3.2). Since S does not simultaneously contain both $\texttt{in}(x)$ and $\texttt{out}(x)$ for any $x \in Ar$, by replacing the constraint $\leftarrow \texttt{in}(x),\ \texttt{out}(x)$ of Π^S_{AF} with rules (11)–(13) in Π^P_{AF}, M_S becomes a stable model of Π^P_{AF}. (ii) Else if S is not a stable labelling of AF, S is a complete labelling such that $\texttt{und}(x) \in S$ for some $x \in Ar$. Since S is a stable model of Π^C_{AF} (Theorem 3.1), by replacing atoms $\texttt{und}(x)$ in S with $\texttt{in}(x)$ and $\texttt{out}(x)$ in M_S, M_S becomes a stable model of Π^P_{AF}.

Conversely, suppose that M_S is a stable model of Π^P_{AF}. If $\texttt{UND}(x) \notin M_S$ for any $x \in Ar$, then $\texttt{und}(x) \notin S$ and $\texttt{in}(x) \in S$ implies $\texttt{out}(x) \notin S$ for any $x \in Ar$. So S satisfies the constraints (9), and S becomes a stable model of Π^S_{AF}. By Theorem 3.2, S is a stable labelling of AF thereby a preferred labelling of AF. Else if $\texttt{UND}(x) \in M_S$ for some $x \in Ar$, S satisfies the disjunctive facts (6). Suppose that S does not satisfy the constraints (7). In this case, $\texttt{in}(a) \in S$ and $\texttt{out}(a) \in S$ for some $a \in Ar$. Then $\texttt{IN}(a) \in M_S$ and $\texttt{OUT}(a) \in M_S$ by the construction of M_S. Since M_S is a stable model of Π^P_{AF}, $\texttt{IN}(a) \in M_S$ only if $\texttt{in}(a) \in S$ and $\texttt{out}(a) \notin S$. Contradiction. Then S satisfies the constraints (7). Hence, S becomes a stable model of Π^C_{AF} thereby a complete labelling of AF (Theorem 3.1). To see that S is a preferred labelling of AF, we show that S is maximal with respect to $\texttt{in}$-labelling among all complete labellings of AF. Suppose that S is not a preferred labelling of AF and there is a preferred labelling S' of AF that is obtained from S by replacing $\texttt{und}(a)$ in S with $\texttt{in}(a)$ in S' for some $a \in Ar$. Then $M_{S'} = (M_S \setminus \{\texttt{out}(a), \texttt{UND}(a)\}) \cup \{\texttt{IN}(a)\}$ becomes a stable model of Π^P_{AF} by the first half of this proof. Then $M_{S'} \setminus \{\texttt{IN}(a)\}$ is a stable model of $\Pi^{P-}_{AF} = \Pi^P_{AF} \setminus \{\texttt{IN}(a) \leftarrow \texttt{in}(a), \textbf{not}\ \texttt{out}(a)\}$. Since $\texttt{IN}(a) \notin M_S$, M_S is also a stable model of Π^{P-}_{AF}. By $(M_{S'} \setminus \{\texttt{IN}(a)\}) \subset M_S$, however, M_S is not a minimal model hence it cannot be a stable model of Π^{P-}_{AF}. Hence, there is no such S' and S is maximal with respect to $\texttt{in}$-labelling among all complete labellings of AF. □

Corollary 3.7 *$AF = (Ar, att)$ has a stable labelling iff Π^P_{AF} has a stable model M such that $\texttt{UND}(x) \notin M$ for any $x \in Ar$.*

Example 3.4 Suppose $AF = (\{a, b, c\}, \{(a, b), (b, a), (b, c), (c, c)\})$.



Then Π^P_{AF} consists of rules:

$\texttt{in}(a) \leftarrow \texttt{out}(b), \quad \texttt{in}(b) \leftarrow \texttt{out}(a), \quad \texttt{in}(c) \leftarrow \texttt{out}(b), \texttt{out}(c),$

$\texttt{out}(a) \leftarrow \texttt{in}(b), \quad \texttt{out}(b) \leftarrow \texttt{in}(a), \quad \texttt{out}(c) \leftarrow \texttt{in}(b), \quad \texttt{out}(c) \leftarrow \texttt{in}(c),$

$\leftarrow \texttt{in}(a), \textbf{not}\ \texttt{out}(b), \quad \leftarrow \texttt{in}(b), \textbf{not}\ \texttt{out}(a), \quad \leftarrow \texttt{in}(c), \textbf{not}\ \texttt{out}(b), \quad \leftarrow \texttt{in}(c), \textbf{not}\ \texttt{out}(c),$

$\leftarrow \texttt{out}(a), \textbf{not}\ \texttt{in}(b), \quad \leftarrow \texttt{out}(b), \textbf{not}\ \texttt{in}(a), \quad \leftarrow \texttt{out}(c), \textbf{not}\ \texttt{in}(b), \textbf{not}\ \texttt{in}(c),$

$\texttt{in}(a) \vee \texttt{out}(a) \leftarrow, \quad \texttt{in}(b) \vee \texttt{out}(b) \leftarrow, \quad \texttt{in}(c) \vee \texttt{out}(c) \leftarrow,$

$\texttt{IN}(x) \leftarrow \texttt{in}(x), \textbf{not}\ \texttt{out}(x) \quad$ where $x \in \{a, b, c\}$,

$\texttt{OUT}(x) \leftarrow \textbf{not}\ \texttt{in}(x), \texttt{out}(x) \quad$ where $x \in \{a, b, c\}$,

$\texttt{UND}(x) \leftarrow \texttt{in}(x), \texttt{out}(x) \quad$ where $x \in \{a, b, c\}$.

Π^P_{AF} has the two stable models:

$$\{\, \texttt{out}(a), \texttt{in}(b), \texttt{out}(c), \texttt{OUT}(a), \texttt{IN}(b), \texttt{OUT}(c) \,\},$$
$$\{\, \texttt{in}(a), \texttt{out}(b), \texttt{in}(c), \texttt{out}(c), \texttt{IN}(a), \texttt{OUT}(b), \texttt{UND}(c) \,\}.$$

Then $\{\, \texttt{out}(a), \texttt{in}(b), \texttt{out}(c) \,\}$ and $\{\, \texttt{in}(a), \texttt{out}(b), \texttt{und}(c) \,\}$ are the two preferred labellings of AF (of which the first one is also the stable labelling).

4 Applications

In the preceding section, we presented ASP encodings of AFs. Using these encodings, various types of argumentation-based reasoning problems can be solved using existing ASP solvers. In this section, we provide some examples of such problems, and we show how our encodings can be applied to address them. While we limit ourselves in this section to the complete semantics, the methods that we present are directly applicable to the stable, grounded or preferred semantics.

4.1 Query Answering

Different algorithms exist to answer the question of whether an argument x is acceptable under a given semantics of an AF. Many algorithms are based on the notion of a *dispute tree*, which is a tree whose root is labelled with the argument x, and whose branches represent *disputes*, i.e., exchanges of arguments between an imaginary *proponent* and *opponent* [24]. Our encodings provide an alternative method to answer this question. Under the complete semantics, the question of whether an argument is acceptable under the complete semantics of an AF can be checked as follows.

Theorem 4.1 *Let $AF = (Ar, att)$ be an argumentation framework and Π^C_{AF} an AF-program under the complete semantics. For any argument $x \in Ar$,*

1. *x is labelled* in *in some complete labelling of AF iff $\Pi^C_{AF} \cup \{\, \leftarrow \textbf{not}\ \texttt{in}(x) \,\}$ has a stable model.*
2. *x is labelled* in *in every complete labelling of AF iff $\Pi^C_{AF} \cup \{\, \leftarrow \texttt{in}(x) \,\}$ has no stable model.*

The results also hold by replacing `in` *with* `out` *or* `und`.

Proof: 1. An argument x is labelled `in` in some complete labelling of AF iff Π^C_{AF} has a stable model which contains $\texttt{in}(x)$ (Theorem 3.1) iff $\Pi^C_{AF} \cup \{ \leftarrow \mathbf{not}\ \texttt{in}(x) \}$ has a stable model.
2. An argument x is labelled `in` in every complete labelling of AF iff every stable model of Π^C_{AF} contains $\texttt{in}(x)$ (Theorem 3.1) iff $\Pi^C_{AF} \cup \{ \leftarrow\ \texttt{in}(x) \}$ has no stable model. □

Theorem 4.1 uses a standard query-answering technique of ASP. This means that we can use existing ASP solvers in [6] for checking credulous or skeptical entailment of an argument in AFs.

Example 4.1 Consider the AF of Example 3.1. The program $\Pi^C_{AF} \cup \{ \leftarrow \mathbf{not}\ \texttt{in}(a) \}$ has the single stable model $\{ \texttt{in}(a), \texttt{out}(b), \texttt{in}(c) \}$. On the other hand, $\Pi^C_{AF} \cup \{ \leftarrow \texttt{in}(a) \}$ has two stable models $\{ \texttt{out}(a), \texttt{in}(b), \texttt{out}(c) \}$ and $\{ \texttt{und}(a), \texttt{und}(b), \texttt{und}(c) \}$. By these facts, the argument a is labelled `in` in some (but not every) complete labelling of AF.

4.2 Enforcement

The *enforcement problem* is the problem of determining whether an AF can be modified so that a given set of arguments becomes a subset of an extension of the AF, for example by adding new arguments that interact with existing ones [3, 23, 37]. To encode this type of problem, we introduce the notion of the *universal argumentation framework*.

Definition 4.1 (universal argumentation framework [11, 30]) The *universal argumentation framework* (UAF) is a pair (U, att_U) in which U is the set of all arguments in the language and $att_U \subseteq U \times U$ is the set of fixed attack relations over U. An argumentation framework $AF = (Ar, att)$ is called a *sub-AF* of the UAF (written $AF \sqsubseteq UAF$) if Ar is a finite subset of U and $att = att_U \cap (Ar \times Ar)$.

The universal argumentation framework serves as the universe of all possible arguments. Agents are assumed to have access only to part of the world, therefore the private argumentation framework AF of an agent is assumed to be a subgraph of the UAF. For the set U of all arguments, the set of all labelled arguments is defined as

$$B_U = \{ \texttt{in}(x), \texttt{out}(x), \texttt{und}(x) \mid x \in U \}.$$

The *enforcement problem* is defined as follows.

Definition 4.2 (enforcement) Let $AF = (Ar, att)$ be a sub-AF of $UAF = (U, att_U)$. Given an *enforcement set* $E \subset B_U$, if one can construct a new argumentation framework $AF' = (Ar', att')$ such that (i) $Ar \subseteq Ar'$ and $AF' \sqsubseteq UAF$, and (ii) AF' has a complete labelling S such that $E \subseteq S$, then AF *satisfies* the enforcement E under the complete semantics.

Note that one cannot enforce different labellings for the same argument, then E is given as a proper subset of B_U. We introduce new arguments together with attack relations involving the introduced arguments. On the other hand, we assume that attack relations between two arguments are fixed in the UAF as far as there is no change in the world. So we do not allow to introduce arbitrary attack relations that are not in att_U for satisfying an enforcement set. We introduce an AF program for the enforcement problem.

Definition 4.3 (AF program for enforcement) Let $AF = (Ar, att)$ be a sub-AF of $UAF = (U, att_U)$, and ϵ_x and $\overline{\epsilon_x}$ new propositions uniquely associated with each $x \in U$. Then an *AF-program for enforcement under the complete semantics* $\varepsilon\Pi^C_{AF}$ consists of the following rules:

1. For every $x \in U$, let $x^- \cap Ar = \{y_1, \ldots, y_k\}$ and $x^- \cap (U \setminus Ar) = \{y_{k+1}, \ldots, y_n\}$ $(0 \leq k \leq n)$.

$$\mathtt{in}(x) \leftarrow \mathtt{out}(y_1), \ldots, \mathtt{out}(y_k), [\mathtt{out}(y_{k+1})], \ldots, [\mathtt{out}(y_n)],\ [\epsilon_x] \tag{14}$$
$$\leftarrow \mathtt{out}(x), \mathbf{not}\ \mathtt{in}(y_1), \ldots, \mathbf{not}\ \mathtt{in}(y_k), [\mathbf{not}\ \mathtt{in}(y_{k+1})], \ldots, [\mathbf{not}\ \mathtt{in}(y_n)],\ [\epsilon_x] \tag{15}$$

   where

   - $[\mathtt{out}(y_j)]$ $(k+1 \leq j \leq n)$ is either a conjunction "$\mathtt{out}(y_j), \epsilon_{y_j}$" or a proposition $\overline{\epsilon_{y_j}}$
   - $[\mathbf{not}\ \mathtt{in}(y_j)]$ $(k+1 \leq j \leq n)$ is either a conjunction "$\mathbf{not}\ \mathtt{in}(y_j), \epsilon_{y_j}$" or a proposition $\overline{\epsilon_{y_j}}$
   - $[\epsilon_x] = true$ if $x \in Ar$; otherwise, $[\epsilon_x] = \epsilon_x$.

2. For every $(y, x) \in att_U$,

$$\mathtt{out}(x) \leftarrow \mathtt{in}(y), [\epsilon_x], [\epsilon_y] \tag{16}$$
$$\leftarrow \mathtt{in}(x), \mathbf{not}\ \mathtt{out}(y), [\epsilon_x], [\epsilon_y] \tag{17}$$

3. For every $x \in U$,

$$\mathtt{in}(x) \vee \mathtt{out}(x) \vee \mathtt{und}(x) \leftarrow [\epsilon_x] \tag{18}$$
$$\leftarrow \mathtt{in}(x), \mathtt{out}(x) \tag{19}$$

4. For every $x \in U \setminus Ar$,

$$\epsilon_x \vee \overline{\epsilon_x} \leftarrow, \quad \leftarrow \epsilon_x, \overline{\epsilon_x} \tag{20}$$

The rules (14) and (15) are modifications of rules (2) and (5) of Γ_{AF}. For $x \in Ar$ and $y_i \in Ar$ $(1 \leq i \leq k)$, (14) and (15) specify the same conditions as (2) and (5), respectively. For any argument $y_j \in U \setminus Ar$ $(k+1 \leq j \leq n)$, $[\mathtt{out}(y_j)]$ is either a conjunction of atoms "$\mathtt{out}(y_j), \epsilon_{y_j}$" or a proposition $\overline{\epsilon_{y_j}}$. If ϵ_{y_j} is selected by the disjunctive fact (20), then $\mathtt{out}(y_j)$ becomes "active" in the body of the rule (14). Else

if $\overline{\epsilon_{y_j}}$ is selected by (20), then $\mathtt{out}(y_j)$ is "inactive" in the body of the rule (14). The whole rule (14) is active if $x \in Ar$ ($[\epsilon_x] = true$); otherwise, it becomes active if ϵ_x is selected by (20). The rule (15) has a similar condition. The rules (16) and (17) are similar modification of rules (3) and (4) of Γ_{AF}. The rules (18) and (19) correspond to rules (6) and (7) of Π^C_{AF}. In particular, (19) does not have the condition $[\epsilon_x]$ because the constraints always hold for any x. Intuitively, enforcement may introduce new arguments in $U \setminus Ar$. If an argument $x \in U \setminus Ar$ is introduced to Ar, then it is represented by the selection of ϵ_x in (20). For those arguments newly introduced to Ar, the corresponding rules become active in $\varepsilon\Pi^C_{AF}$. For any argument $x \in U \setminus Ar$ that is not introduced to Ar, an atom $\overline{\epsilon_x}$ is selected in (20). If $\overline{\epsilon_x}$ is selected, then neither $\mathtt{in}(x)$, $\mathtt{out}(x)$ nor $\mathtt{und}(x)$ is obtained by the rule (18) for such x. The program $\varepsilon\Pi^C_{AF}$ depends on the UAF, but we assume the existence of a fixed UAF and do not explicitly write it as a parameter in $\varepsilon\Pi^C_{AF}$.

Using the program, the enforcement problem is computed in ASP as follows.

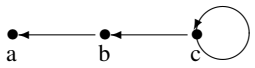
Theorem 4.2 *Let $AF = (Ar, att)$ be an argumentation framework and $\varepsilon\Pi^C_{AF}$ the program defined as above. Given an enforcement set $E \subset B_U$, AF satisfies the enforcement E under the complete semantics iff the program $\varepsilon\Pi^C_{AF} \cup \{ \leftarrow \ \mathbf{not}\, \ell(x) \mid \ell(x) \in E$ where $\ell \in \{\mathtt{in}, \mathtt{out}, \mathtt{und}\}\}$ has a stable model.*

Proof: AF satisfies the enforcement E under the complete semantics iff AF is extended to AF' in a way that AF' has a complete labelling S such that $E \subseteq S$. Suppose that $AF' = (Ar', att')$ where $Ar \subseteq Ar' \subseteq U$ and $att' = att_U \cap (Ar' \times Ar')$. In (20) of $\varepsilon\Pi^C_{AF}$, select ϵ_x for any $x \in Ar' \setminus Ar$, and select $\overline{\epsilon_x}$ for any $x \in U \setminus Ar'$. In this case, rules (14) and (15) for $x \in Ar'$ are simplified to

$$\mathtt{in}(x) \leftarrow \mathtt{out}(y_1), \ldots, \mathtt{out}(y_k), \mathtt{out}(y_{k+1}), \ldots, \mathtt{out}(y_m)$$
$$\leftarrow \mathtt{out}(x), \mathbf{not}\, \mathtt{in}(y_1), \ldots, \mathbf{not}\, \mathtt{in}(y_k), \mathbf{not}\, \mathtt{in}(y_{k+1}), \ldots, \mathbf{not}\, \mathtt{in}(y_m)$$

for some m $(k+1 \leq m \leq n)$ by unfolding (14) and (15) with atoms ϵ_x, ϵ_{y_j} and $\overline{\epsilon_{y_j}}$ selected in (20). Similar simplification is done for rules (16) and (17). Then the program $\varepsilon\Pi^C_{AF}$ contains the rules of $\Pi^C_{AF'}$. On the other hand, ϵ_x is not true for $x \in U \setminus Ar'$ in $\varepsilon\Pi^C_{AF}$. So none of $\mathtt{in}(x)$, $\mathtt{out}(x)$ and $\mathtt{und}(x)$ is derived by (18) for $x \in U \setminus Ar'$, and none of those atoms are included in a stable model of $\varepsilon\Pi^C_{AF}$. Then a program $\varepsilon\Pi^C_{AF}$ has a stable model M such that $M = N \cup \{ \epsilon_x \mid x \in Ar' \setminus Ar \} \cup \{ \overline{\epsilon_x} \mid x \in U \setminus Ar' \}$ for some stable model N of $\Pi^C_{AF'}$. Thus, AF' has a complete labelling S such that $E \subseteq S$ iff $\varepsilon\Pi^C_{AF}$ has a stable model M such that $E \subseteq M$ by Theorem 3.1. Hence the result holds by Theorem 4.1. □

Example 4.2 Let $UAF = (\{a, b, c\}, \{(b, a), (c, b), (c, c)\})$ and $AF = (\{a\}, \emptyset)$.



Then AF has the complete labelling $\{ \mathtt{in}(a) \}$. The program $\varepsilon\Pi^C_{AF}$ consists of rules:

$\mathtt{in}(a) \leftarrow \mathtt{out}(b), \epsilon_b, \quad \mathtt{in}(a) \leftarrow \overline{\epsilon_b}, \quad \mathtt{in}(b) \leftarrow \mathtt{out}(c), \epsilon_c, \epsilon_b,$
$\mathtt{in}(b) \leftarrow \overline{\epsilon_c}, \epsilon_b, \quad \mathtt{in}(c) \leftarrow \mathtt{out}(c), \epsilon_c, \quad \mathtt{in}(c) \leftarrow \overline{\epsilon_c}, \epsilon_c,$
$\mathtt{out}(a) \leftarrow \mathtt{in}(b), \epsilon_b, \quad \mathtt{out}(b) \leftarrow \mathtt{in}(c), \epsilon_b, \epsilon_c, \quad \mathtt{out}(c) \leftarrow \mathtt{in}(c), \epsilon_c,$
$\leftarrow \mathtt{in}(a), \mathbf{not}\, \mathtt{out}(b), \epsilon_b, \quad \leftarrow \mathtt{in}(b), \mathbf{not}\, \mathtt{out}(c), \epsilon_b, \epsilon_c, \quad \leftarrow \mathtt{in}(c), \mathbf{not}\, \mathtt{out}(c), \epsilon_c,$
$\leftarrow \mathtt{out}(a), \mathbf{not}\, \mathtt{in}(b), \epsilon_b, \quad \leftarrow \mathtt{out}(a), \overline{\epsilon_b}, \quad \leftarrow \mathtt{out}(b), \mathbf{not}\, \mathtt{in}(c), \epsilon_c, \epsilon_b,$
$\leftarrow \mathtt{out}(b), \overline{\epsilon_c}, \epsilon_b, \quad \leftarrow \mathtt{out}(c), \mathbf{not}\, \mathtt{in}(c), \epsilon_c, \quad \leftarrow \mathtt{out}(c), \overline{\epsilon_c}, \epsilon_c,$
$\mathtt{in}(a) \vee \mathtt{out}(a) \vee \mathtt{und}(a) \leftarrow, \quad \mathtt{in}(b) \vee \mathtt{out}(b) \vee \mathtt{und}(b) \leftarrow \epsilon_b, \quad \mathtt{in}(c) \vee \mathtt{out}(c) \vee \mathtt{und}(c) \leftarrow \epsilon_c,$
$\leftarrow \mathtt{in}(a), \mathtt{out}(a), \quad \leftarrow \mathtt{in}(b), \mathtt{out}(b), \quad \leftarrow \mathtt{in}(c), \mathtt{out}(c),$
$\epsilon_b \vee \overline{\epsilon_b} \leftarrow, \quad \epsilon_c \vee \overline{\epsilon_c} \leftarrow, \quad \leftarrow \epsilon_b, \overline{\epsilon_b}, \quad \leftarrow \epsilon_c, \overline{\epsilon_c}.$

Given the enforcement set $E = \{\mathtt{out}(a)\}$, the program $\varepsilon\Pi^C_{AF} \cup \{ \leftarrow \mathbf{not}\, \mathtt{out}(a) \}$ has the stable model $\{ \mathtt{out}(a), \mathtt{in}(b), \epsilon_b, \overline{\epsilon_c} \}$. Then AF satisfies the enforcement E. That is, to enforce $\mathtt{out}(a)$, AF is modified by introducing the new argument b and the attack relation (b, a). On the other hand, let $AF' = (\{a, b\}, \{(b, a)\})$ which has the complete labelling $\{\mathtt{out}(a), \mathtt{in}(b)\}$. The program $\varepsilon\Pi^C_{AF'}$ is obtained from $\varepsilon\Pi^C_{AF}$ by (i) removing rules "$\mathtt{in}(a) \leftarrow \overline{\epsilon_b}$" and "$\leftarrow \mathtt{out}(a), \overline{\epsilon_b}$", (ii) removing "$\epsilon_b \vee \overline{\epsilon_b} \leftarrow$" and "$\leftarrow \epsilon_b, \overline{\epsilon_b}$", and (iii) removing the proposition ϵ_b from the remaining rules. Given the enforcement set $E' = \{\mathtt{in}(a)\}$, the program $\varepsilon\Pi^C_{AF'} \cup \{ \leftarrow \mathbf{not}\, \mathtt{in}(a) \}$ has no stable model. In this case, AF' does not satisfy the enforcement E' because there is no way to make $\mathtt{in}(a)$ true by introducing a new argument to AF'. In fact, introducing c to AF' makes it identical to UAF that has the complete labelling $\{ \mathtt{und}(a), \mathtt{und}(b), \mathtt{und}(c) \}$.

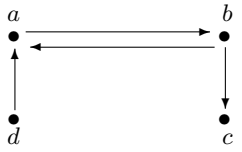
4.3 Agreement

Argumentation frameworks are used in negotiation [1] and debate [30]. In negotiation and debate, two agents having different opinions exchange their arguments to reach an agreement. In this section, we represent two agents as different AFs and formulate agreement between them.

Definition 4.4 (agreement) Let $AF_1 = (Ar_1, att_1)$ and $AF_2 = (Ar_2, att_2)$ be two sub-AFs of $UAF = (U, att_U)$. If AF_1 has a complete labelling S and AF_2 has a complete labelling T such that $S \cap T \neq \emptyset$, then AF_1 and AF_2 can reach an *agreement* under the complete semantics. In this case, we say that AF_1 and AF_2 *agree on* $S \cap T$.

By definition, two AFs can reach an agreement if they have complete labellings that agree on labellings of some arguments.

Example 4.3 Suppose $AF_1 = (\{a, b, c\}, \{(a, b), (b, a), (b, c)\})$, $AF_2 = (\{a, b, d\}, \{(a, b), (b, a), (d, a)\})$ and $UAF = (\{a, b, c, d\}, \{(a, b), (b, a), (b, c), (d, a)\})$.



Then AF_1 has the three complete labellings: $\{\, \mathtt{in}(a),\, \mathtt{out}(b),\, \mathtt{in}(c)\,\}$, $\{\, \mathtt{out}(a),\, \mathtt{in}(b),\, \mathtt{out}(c)\,\}$, $\{\, \mathtt{und}(a),\, \mathtt{und}(b),\, \mathtt{und}(c)\,\}$; while AF_2 has the single complete labelling: $\{\, \mathtt{out}(a),\, \mathtt{in}(b),\, \mathtt{in}(d)\,\}$. Then AF_1 and AF_2 agree on $\{\, \mathtt{out}(a),\, \mathtt{in}(b)\,\}$.

In negotiation or debate, agents are interested in whether they can agree on some particular arguments. Let $\gamma\Pi^C_{AF}$ be a program in which predicates $\mathtt{in}$, $\mathtt{out}$ and $\mathtt{und}$ in Π^C_{AF} are renamed by $\mathtt{in}'$, $\mathtt{out}'$ and $\mathtt{und}'$, respectively. Define

$$\begin{aligned}\Phi \;=\; & \{\, \mathtt{agree}(x) \leftarrow \mathtt{in}(x), \mathtt{in}'(x) \mid x \in U\,\} \cup \{\, \mathtt{agree}(x) \leftarrow \mathtt{out}(x), \mathtt{out}'(x) \mid x \in U\,\} \\ & \cup \{\, \mathtt{agree}(x) \leftarrow \mathtt{und}(x), \mathtt{und}'(x) \mid x \in U\,\} \cup \{\, \mathtt{ok} \leftarrow \mathtt{agree}(x) \mid x \in U\,\} \cup \{\, \leftarrow \mathbf{not}\ \mathtt{ok}\,\}\end{aligned}$$

where $\mathtt{ok}$ and $\mathtt{agree}(x)$ are new atoms. Then we have the next result.

Theorem 4.3 Let AF_1 and AF_2 be two sub-AFs of the UAF, and $\Pi^C_{AF_1}$ and $\Pi^C_{AF_2}$ their AF-programs under the complete semantics, respectively. Then AF_1 and AF_2 can reach an agreement under the complete semantics iff the program $\Pi^C_{AF_1} \cup \gamma\Pi^C_{AF_2} \cup \Phi$ has a stable model S. In this case, AF_1 and AF_2 agree on each argument x satisfying $\mathtt{agree}(x) \in S$.

Proof: AF_1 and AF_2 can reach an agreement under the complete semantics iff $\Pi^C_{AF_1}$ has a stable model M and $\Pi^C_{AF_2}$ has a stable model N such that $M \cap N \neq \emptyset$ $(*)$. $\Pi^C_{AF_1}$ has a stable model M and $\Pi^C_{AF_2}$ has a stable model N iff $\Pi^C_{AF_1} \cup \gamma\Pi^C_{AF_2}$ has a stable model $M \cup N'$ where $N' = \{\, \ell'(x) \mid \ell(x) \in N$ and $\ell \in \{\mathtt{in}, \mathtt{out}, \mathtt{und}\}\,\}$. Then $(*)$ holds iff $\Pi^C_{AF_1} \cup \gamma\Pi^C_{AF_2}$ has a stable model T such that $\emptyset \subset \{\ell(x), \ell'(x)\} \subseteq T$ for some $x \in U$. In this case, $\Pi^C_{AF_1} \cup \gamma\Pi^C_{AF_2} \cup \Phi$ has a stable model S in which $\mathtt{agree}(x) \in S$ for any $x \in U$ such that $\ell(x) \in M$ iff $\ell(x) \in N$. □

Theorem 4.3 can be extended to agreement among more than two agents.

4.4 Equivalence

Two argumentation frameworks are *equivalent* if they have the same extensions (or labellings) under the designated semantics. Whether two argumentation frameworks are equivalent or not depends on the choice of semantics.

Example 4.4 Consider $AF_1 = (\{a, b\}, \{(a, a), (a, b), (b, a)\})$ and $AF_2 = (\{a, b\}, \{(a, a), (b, a)\})$.

AF_1 has the two complete labellings: $\{\mathtt{out}(a),\, \mathtt{in}(b)\}$ and $\{\mathtt{und}(a), \mathtt{und}(b)\}$, of which $\{\mathtt{out}(a),\, \mathtt{in}(b)\}$ is the stable labelling and $\{\mathtt{und}(a), \mathtt{und}(b)\}$ is the grounded labelling. On the other hand, AF_2 has the single complete labelling: $\{\mathtt{out}(a),\, \mathtt{in}(b)\}$, which is also the stable labelling and the grounded labelling. Thus, AF_1 and AF_2 are equivalent under the stable semantics, while they are different under the complete semantics and the grounded semantics.

Equivalence of two AFs is known by checking the equivalence of their AF-programs. In the above example, Γ_{AF_1} and Γ_{AF_2} become

$$
\begin{array}{ll}
\Gamma_{AF_1}: & \mathtt{in}(a) \leftarrow \mathtt{out}(a), \mathtt{out}(b), \quad \mathtt{in}(b) \leftarrow \mathtt{out}(a), \\
& \mathtt{out}(a) \leftarrow \mathtt{in}(a), \quad \mathtt{out}(a) \leftarrow \mathtt{in}(b), \quad \mathtt{out}(b) \leftarrow \mathtt{in}(a), \\
& \leftarrow \mathtt{in}(a), \mathbf{not}\, \mathtt{out}(a), \quad \leftarrow \mathtt{in}(a), \mathbf{not}\, \mathtt{out}(b), \quad \leftarrow \mathtt{in}(b), \mathbf{not}\, \mathtt{out}(a), \\
& \leftarrow \mathtt{out}(a), \mathbf{not}\, \mathtt{in}(a), \mathbf{not}\, \mathtt{in}(b), \quad \leftarrow \mathtt{out}(b), \mathbf{not}\, \mathtt{in}(a). \\
\Gamma_{AF_2}: & \mathtt{in}(a) \leftarrow \mathtt{out}(a), \mathtt{out}(b), \quad \mathtt{in}(b) \leftarrow, \\
& \mathtt{out}(a) \leftarrow \mathtt{in}(a), \quad \mathtt{out}(a) \leftarrow \mathtt{in}(b), \\
& \leftarrow \mathtt{in}(a), \mathbf{not}\, \mathtt{out}(a), \quad \leftarrow \mathtt{in}(a), \mathbf{not}\, \mathtt{out}(b), \\
& \leftarrow \mathtt{out}(a), \mathbf{not}\, \mathtt{in}(a), \mathbf{not}\, \mathtt{in}(b), \quad \leftarrow \mathtt{out}(b).
\end{array}
$$

The AF-program under the complete semantics $\Pi^C_{AF_1} = \Gamma_{AF_1} \cup \{\, \mathtt{in}(x) \vee \mathtt{out}(x) \vee \mathtt{und}(x) \leftarrow \mid x \in \{a,b\}\} \cup \{\leftarrow \mathtt{in}(x), \mathtt{out}(x) \mid x \in \{a,b\}\}$ has two stable models $\{\mathtt{out}(a), \mathtt{in}(b)\}$ and $\{\mathtt{und}(a), \mathtt{und}(b)\}$, while the AF-program under the complete semantics $\Pi^C_{AF_2}$ has the single stable model $\{\mathtt{out}(a), \mathtt{in}(b)\}$. The AF-program under the stable semantics $\Pi^S_{AF_1} = \Gamma_{AF_1} \cup \{\mathtt{in}(x) \vee \mathtt{out}(x) \leftarrow \mid x \in \{a,b\}\} \cup \{\leftarrow \mathtt{in}(x), \mathtt{out}(x) \mid x \in \{a,b\}\}$ has the single stable model $\{\mathtt{out}(a), \mathtt{in}(b)\}$ and the AF-program under the stable semantics $\Pi^S_{AF_2}$ has the same stable model. The AF-program under the grounded semantics $\Pi^G_{AF_1} = \Gamma_{AF_1} \cup \{\mathtt{und}(x) \leftarrow \mathbf{not}\, \mathtt{in}(x), \mathbf{not}\, \mathtt{out}(x) \mid x \in \{a,b\}\}$ has the single stable model $\{\mathtt{und}(a), \mathtt{und}(b)\}$, while the AF-program under the grounded semantics $\Pi^G_{AF_2}$ has the stable model $\{\mathtt{out}(a), \mathtt{in}(b)\}$.

Since $\Pi^S_{AF_1}$ and $\Pi^S_{AF_2}$ have the same stable model, the corresponding argumentation frameworks AF_1 and AF_2 are equivalent under the stable semantics. We write $\Pi_1 \equiv \Pi_2$ if Π_1 and Π_2 have the same stable models.

Theorem 4.4 *Two argumentation frameworks AF_1 and AF_2 are equivalent under the complete (resp. stable, grounded, preferred) semantics iff $\Pi^C_{AF_1} \equiv \Pi^C_{AF_2}$ (resp. $\Pi^S_{AF_1} \equiv \Pi^S_{AF_2}$, $\Pi^G_{AF_1} \equiv \Pi^G_{AF_2}$, $\Pi^P_{AF_1} \equiv \Pi^P_{AF_2}$).*

Proof: The results follow from Theorems 3.1, 3.2, 3.4 and 3.6. □

Strong equivalence is also used for comparing different AFs [27]. Two argumentation frameworks $AF_1 = (Ar_1, att_1)$ and $AF_2 = (Ar_2, att_2)$ are *strongly equivalent* (under some semantics $\mathcal{L}$) if $AF_1 \sqcup AF = (Ar_1 \cup Ar, att_1 \cup att)$ and $AF_2 \sqcup AF = (Ar_2 \cup Ar, att_2 \cup att)$ are equivalent (under $\mathcal{L}$) for any sub-AF $AF = (Ar, att)$ of UAF. The problem is translated into ASP such that AF_1 and AF_2 are strongly equivalent iff $\Pi^X_{AF_1 \sqcup AF} \equiv \Pi^X_{AF_2 \sqcup AF}$ holds for any $AF \sqsubseteq UAF$ where X is either C, S, G or P. In this way, equivalence issues in AFs are translated into those in ASP. Testing equivalence of two AFs is done by first translating them into corresponding AF-programs, then using existing techniques of testing equivalence of logic programs [26].

4.5 Encoding ABA

Assumption-based argumentation (ABA) [5, 15] is a form of argumentation and its relationship to abstract argumentation frameworks has been studied in the literature [9, 14, 32]. In this section, we provide a method of representing ABA in answer set programming.

Definition 4.5 (argument [15]) Given a deductive system $(\mathbf{L}, \mathcal{R})$ where $\mathbf{L}$ is a logical language and $\mathcal{R}$ is a set of *inference rules* in this language, and a set of *assumptions* $\mathcal{A} \subseteq \mathbf{L}$, an *argument* for a conclusion $c \in \mathbf{L}$ supported by $S \subseteq \mathcal{A}$ is a finite tree with nodes labelled by formulas in $\mathbf{L}$ or by the special symbol $\top$ such that:

- the root is labelled by 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 $b \in \mathbf{L}$ 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 $b_1, \ldots, b_m$ ($b_i \in \mathbf{L}$ for $i = 1, \ldots, m$) respectively.
- S is the set of all assumptions labelling the leaves.

An argument for c supported by S is written as $S \vdash c$ or $A : S \vdash c$ where A is the *name* used for referring the argument.

Definition 4.6 (ABA framework [15]) An *ABA framework* is a tuple $(\mathbf{L}, \mathcal{R}, \mathcal{A}, {}^{-})$ where

- $(\mathbf{L}, \mathcal{R})$ is a deductive system.
- $\mathcal{A} \subseteq \mathbf{L}$ is a (non-empty) set of assumptions.
- ${}^{-}$ is a total mapping from $\mathcal{A}$ into $\mathbf{L}$, where $\bar{\alpha}$ is the *contrary* of α.
- A set of assumptions $S \subseteq \mathcal{A}$ *attacks* $\alpha \in \mathcal{A}$ if there exists an argument $T \vdash \bar{\alpha}$ such that $T \subseteq S$.

An ABA framework is *flat* if assumptions only occur on the right of the arrow in inference rules of $\mathcal{R}$.

In this section, we consider only flat ABA frameworks and an ABA means a flat ABA hereafter. Let $\mathcal{LA} : \mathcal{A} \rightarrow \{IN, OUT, UND\}$ be a total function (called *assumption labelling*). Define $IN(\mathcal{LA}) = \{\, \alpha \in \mathcal{A} \mid \mathcal{LA}(\alpha) = IN \,\}$, $OUT(\mathcal{LA}) = \{\, \alpha \in \mathcal{A} \mid \mathcal{LA}(\alpha) = OUT \,\}$ and $UND(\mathcal{LA}) = \{\, \alpha \in \mathcal{A} \mid \mathcal{LA}(\alpha) = UND \,\}$.

Schulz and Toni [32] introduce a complete labelling semantics for ABA as follows.

Definition 4.7 (complete assumption labelling [32]) An assumption labelling $\mathcal{LA}$ is a *complete assumption labelling* iff for each assumption $\alpha \in \mathcal{A}$ it holds that:

- if $\mathcal{LA}(\alpha) = IN$ then each set of assumptions attacking α contains some β such that $\mathcal{LA}(\beta) = OUT$.
- if $\mathcal{LA}(\alpha) = OUT$ then there exists a set of assumptions S attacking α such that $S \subseteq IN(\mathcal{LA})$.
- if $\mathcal{LA}(\alpha) = UND$ then each set of assumptions attacking α contains some β such that $\mathcal{LA}(\beta) \neq IN$ and there is a set of assumptions S attacking α such that $S \cap OUT(\mathcal{LA}) = \emptyset$.

An ABA framework $(\mathbf{L}, \mathcal{R}, \mathcal{A}, {}^-)$ can be mapped onto an associated (abstract) argumentation framework (Ar_{ABA}, att_{ABA}) [14] where

- Ar_{ABA} is the set of all constructible arguments $S \vdash c$ in $(\mathbf{L}, \mathcal{R}, \mathcal{A}, {}^-)$;
- $(S_1 \vdash c_1,\ S_2 \vdash c_2) \in att_{ABA}$ iff c_1 is the contrary of some $\alpha \in S_2$.

In what follows, labelling $\mathcal{L}$ of arguments in an (abstract) argumentation framework is also called *argument labelling* in order to distinguish it from assumption labelling. Schulz and Toni [32] show that there is a one-to-one correspondence between complete assumption labellings of an ABA framework and complete argument labellings of its associated argumentation framework.

Proposition 4.5 *([32]) Let $\mathcal{LA}$ be an assumption labelling of an ABA framework $(\mathbf{L}, \mathcal{R}, \mathcal{A}, {}^-)$. Then $\mathcal{LA}$ is a complete assumption labelling of $(\mathbf{L}, \mathcal{R}, \mathcal{A}, {}^-)$ iff the argument labelling $\mathcal{L}$ with*

- $\mathtt{in}(\mathcal{L}) = \{\, (S \vdash c) \in Ar_{ABA} \mid S \subseteq IN(\mathcal{LA}) \,\}$,
- $\mathtt{out}(\mathcal{L}) = \{\, (S \vdash c) \in Ar_{ABA} \mid \exists \alpha \in S \ \textit{ such that } \ \alpha \in OUT(\mathcal{LA}) \}$,
- $\mathtt{und}(\mathcal{L}) = \{\, (S \vdash c) \in Ar_{ABA} \mid \exists \alpha \in S \ \textit{ such that } \ \alpha \in UND(\mathcal{LA}) \textit{ and } S \cap OUT(\mathcal{LA}) = \emptyset \,\}$

is a complete argument labelling of (Ar_{ABA}, att_{ABA}).

Example 4.5 *([32])* Consider the ABA framework $(\mathbf{L}, \mathcal{R}, \mathcal{A}, {}^-)$ where $\mathbf{L} = \{a, b, c, \alpha, \beta, \gamma\}$, $\mathcal{R} = \{\, a \leftarrow \alpha,\ a \leftarrow \beta,\ c \leftarrow \beta,\ b \leftarrow \gamma \,\}$, $\mathcal{A} = \{\alpha, \beta, \gamma\}$, $\bar{\alpha} = a$, $\bar{\beta} = b$ and $\bar{\gamma} = c$. The ABA has the three complete assumption labellings:

$$
\begin{aligned}
&IN(\mathcal{LA}_1) = \emptyset, \quad OUT(\mathcal{LA}_1) = \emptyset, \quad UND(\mathcal{LA}_1) = \{\alpha, \beta, \gamma\},\\
&IN(\mathcal{LA}_2) = \{\gamma\}, \quad OUT(\mathcal{LA}_2) = \{\beta\}, \quad UND(\mathcal{LA}_2) = \{\alpha\},\\
&IN(\mathcal{LA}_3) = \{\beta\}, \quad OUT(\mathcal{LA}_3) = \{\alpha, \gamma\}, \quad UND(\mathcal{LA}_3) = \emptyset.
\end{aligned}
$$

The ABA framework $(\mathbf{L}, \mathcal{R}, \mathcal{A}, {}^-)$ is transformed to the argumentation framework (Ar_{ABA}, att_{ABA}) such that

$$
\begin{aligned}
Ar_{ABA} = \{\, &A_1 : \{\alpha\} \vdash \alpha, \quad A_2 : \{\beta\} \vdash \beta, \quad A_3 : \{\gamma\} \vdash \gamma, \quad A_4 : \{\alpha\} \vdash a, \quad A_5 : \{\beta\} \vdash a,\\
&A_6 : \{\beta\} \vdash c, \quad A_7 : \{\gamma\} \vdash b \,\},\\
att_{ABA} = \{&(A_4, A_1), (A_4, A_4), (A_5, A_1), (A_5, A_4), (A_6, A_3), (A_6, A_7), (A_7, A_2), (A_7, A_5), (A_7, A_6)\}.
\end{aligned}
$$

(Ar_{ABA}, att_{ABA}) has the three complete argument labellings which correspond to the three complete assumption labellings.

$$\begin{array}{l}
\mathtt{in}(\mathcal{L}_1) = \emptyset,\ \ \mathtt{out}(\mathcal{L}_1) = \emptyset,\ \ \mathtt{und}(\mathcal{L}_1) = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\},\\
\mathtt{in}(\mathcal{L}_2) = \{A_3, A_7\},\ \ \mathtt{out}(\mathcal{L}_2) = \{A_2, A_5, A_6\},\ \ \mathtt{und}(\mathcal{L}_2) = \{A_1, A_4\},\\
\mathtt{in}(\mathcal{L}_3) = \{A_2, A_5, A_6\},\ \ \mathtt{out}(\mathcal{L}_3) = \{A_1, A_3, A_4, A_7\},\ \ \mathtt{und}(\mathcal{L}_3) = \emptyset.
\end{array}$$

An ABA framework is represented in a logic program by combining the transformation from ABA to AFs and the transformation from AFs to LPs (under the complete semantics).

Definition 4.8 (transforming ABA to LP) Let $(\mathbf{L}, \mathcal{R}, \mathcal{A}, {}^-)$ be an ABA framework and $AF_{ABA} = (Ar_{ABA}, att_{ABA})$ its associated AF. Then $\Pi^C_{AF_{ABA}}$ is called an ABA-program under the complete semantics.

Theorem 4.6 *Let $(\mathbf{L}, \mathcal{R}, \mathcal{A}, {}^-)$ be an ABA framework and $\Pi^C_{AF_{ABA}}$ its associated ABA-program under the complete semantics. Then $\mathcal{LA}$ is a complete assumption labelling of $(\mathbf{L}, \mathcal{R}, \mathcal{A}, {}^-)$ iff M is a stable model of $\Pi^C_{AF_{ABA}}$ such that*

- $\mathtt{in}(A) \in M$ *iff* $(A : S \vdash c) \in Ar_{ABA}$ *and* $S \subseteq IN(\mathcal{LA})$,
- $\mathtt{out}(A) \in M$ *iff* $(A : S \vdash c) \in Ar_{ABA}$ *and* $\exists \alpha \in S$ *such that* $\alpha \in OUT(\mathcal{LA})$,
- $\mathtt{und}(A) \in M$ *iff* $(A : S \vdash c) \in Ar_{ABA}$ *and* $\exists \alpha \in S$ *such that* $\alpha \in UND(\mathcal{LA})$ *and* $S \cap OUT(\mathcal{LA}) = \emptyset$.

Proof: The result holds by Proposition 4.5 and Theorem 3.1. □

Example 4.6 The ABA framework of Example 4.5 is transformed to $\Pi^C_{AF_{ABA}}$ such that

$$\begin{array}{l}
\mathtt{in}(A_1) \leftarrow \mathtt{out}(A_4), \mathtt{out}(A_5), \quad \mathtt{in}(A_2) \leftarrow \mathtt{out}(A_7), \quad \mathtt{in}(A_3) \leftarrow \mathtt{out}(A_6),\\
\mathtt{in}(A_4) \leftarrow \mathtt{out}(A_4), \mathtt{out}(A_5), \quad \mathtt{in}(A_5) \leftarrow \mathtt{out}(A_7), \quad \mathtt{in}(A_6) \leftarrow \mathtt{out}(A_7),\\
\mathtt{in}(A_7) \leftarrow \mathtt{out}(A_6), \quad \mathtt{out}(A_1) \leftarrow \mathtt{in}(A_4), \quad \mathtt{out}(A_1) \leftarrow \mathtt{in}(A_5), \quad \mathtt{out}(A_2) \leftarrow \mathtt{in}(A_7),\\
\mathtt{out}(A_3) \leftarrow \mathtt{in}(A_6), \quad \mathtt{out}(A_4) \leftarrow \mathtt{in}(A_4), \quad \mathtt{out}(A_4) \leftarrow \mathtt{in}(A_5), \quad \mathtt{out}(A_5) \leftarrow \mathtt{in}(A_7),\\
\mathtt{out}(A_6) \leftarrow \mathtt{in}(A_7), \quad \mathtt{out}(A_7) \leftarrow \mathtt{in}(A_6),\\
\leftarrow \mathtt{in}(A_1), \mathbf{not}\ \mathtt{out}(A_4), \quad \leftarrow \mathtt{in}(A_1), \mathbf{not}\ \mathtt{out}(A_5), \quad \leftarrow \mathtt{in}(A_2), \mathbf{not}\ \mathtt{out}(A_7),\\
\leftarrow \mathtt{in}(A_3), \mathbf{not}\ \mathtt{out}(A_6), \quad \leftarrow \mathtt{in}(A_4), \mathbf{not}\ \mathtt{out}(A_4), \quad \leftarrow \mathtt{in}(A_4), \mathbf{not}\ \mathtt{out}(A_5),\\
\leftarrow \mathtt{in}(A_5), \mathbf{not}\ \mathtt{out}(A_7), \quad \leftarrow \mathtt{in}(A_6), \mathbf{not}\ \mathtt{out}(A_7), \quad \leftarrow \mathtt{in}(A_7), \mathbf{not}\ \mathtt{out}(A_6),\\
\leftarrow \mathtt{out}(A_1), \mathbf{not}\ \mathtt{in}(A_4), \mathbf{not}\ \mathtt{in}(A_5), \quad \leftarrow \mathtt{out}(A_2), \mathbf{not}\ \mathtt{in}(A_7), \quad \leftarrow \mathtt{out}(A_3), \mathbf{not}\ \mathtt{in}(A_6),\\
\leftarrow \mathtt{out}(A_4), \mathbf{not}\ \mathtt{in}(A_4), \mathbf{not}\ \mathtt{in}(A_5), \quad \leftarrow \mathtt{out}(A_5), \mathbf{not}\ \mathtt{in}(A_7), \quad \leftarrow \mathtt{out}(A_6), \mathbf{not}\ \mathtt{in}(A_7),\\
\leftarrow \mathtt{out}(A_7), \mathbf{not}\ \mathtt{in}(A_6),\\
\mathtt{in}(A_i) \vee \mathtt{out}(A_i) \vee \mathtt{und}(A_i) \leftarrow, \qquad \leftarrow \mathtt{in}(A_i), \mathtt{out}(A_i) \quad \text{for } i = 1, \ldots, 7
\end{array}$$

$\Pi^C_{AF_{ABA}}$ has the three stable models

$$\{\, \mathtt{und}(A_1), \mathtt{und}(A_2), \mathtt{und}(A_3), \mathtt{und}(A_4), \mathtt{und}(A_5), \mathtt{und}(A_6), \mathtt{und}(A_7)\,\},$$
$$\{\, \mathtt{in}(A_3), \mathtt{in}(A_7), \mathtt{out}(A_2), \mathtt{out}(A_5), \mathtt{out}(A_6), \mathtt{und}(A_1), \mathtt{und}(A_4)\,\},$$
$$\{\, \mathtt{in}(A_2), \mathtt{in}(A_5), \mathtt{in}(A_6), \mathtt{out}(A_1), \mathtt{out}(A_3), \mathtt{out}(A_4), \mathtt{out}(A_7)\,\},$$

which correspond to the three complete assumption labellings of the ABA framework.

Assumption labellings are also defined under the grounded, stable or preferred semantics of ABA [10]. Schulz [33] reports connections between assumption labellings and argument labellings under the grounded, stable or preferred semantics of ABA. Using the results, the transformation from ABA to LPs in this section is applied to the grounded, stable or preferred semantics of ABA as well.

5 Comparison with ASP-based Argumentation Systems

There are several studies that are concerned with encoding argumentation frameworks into ASP. Egly *et al*. [19] represent extension-based semantics of AFs (complete, stable, grounded, preferred) by ASP, which is later modified and extended by Gaggl *et al*. [20] to the preferred, semi-stable and stage semantics of AFs. Wakaki *et al*. [36] represent labelling-based semantics of AFs (complete, stable, grounded, preferred and semi-stable) by ASP. In this section, we compare our encodings of the complete, stable, grounded, and preferred semantics of AFs with those presented in [19, 20, 36].

Given $AF = (Ar, att)$, define $\mathcal{F}_{AF} = \{\, arg(x) \mid x \in Ar \,\} \cup \{\, att(x,y) \mid (x,y) \in att \,\}$.

5.1 Egly, Gaggl and Woltran

Egly *et al*. [19] compute complete extensions of $AF = (Ar, att)$ by answer sets of the program:

$$\pi^C_E = \mathcal{F}_{AF} \cup \pi_{adm} \cup \{\, \leftarrow out(x), \mathbf{not}\ undefended(x)\,\}$$

where

$$\begin{aligned}\pi_{adm} \;=\; \{\, & in(x) \leftarrow arg(x), \mathbf{not}\ out(x), \quad out(x) \leftarrow arg(x), \mathbf{not}\ in(x),\\ & \leftarrow in(x), in(y), att(x,y), \quad defeated(x) \leftarrow in(y), att(y,x),\\ & undefended(x) \leftarrow att(y,x), \mathbf{not}\ defeated(y), \quad \leftarrow in(x), undefended(x)\,\}.\end{aligned}$$

They compute stable extensions of $AF = (Ar, att)$ by answer sets of the program:

$$\begin{aligned}\pi^S_E \;=\; \mathcal{F}_{AF} \cup \{\, & in(x) \leftarrow arg(x), \mathbf{not}\ out(x), \quad out(x) \leftarrow arg(x), \mathbf{not}\ in(x),\\ & \leftarrow in(x), in(y), att(x,y), \quad defeated(x) \leftarrow in(y), att(y,x),\\ & \leftarrow out(x), \mathbf{not}\ defeated(x)\,\}.\end{aligned}$$

Next, the grounded extension of $AF = (Ar, att)$ is computed by the answer set of the program:

$$\pi_E^G = \mathcal{F}_{AF} \cup \pi_< \cup \pi_{def} \cup \{ in(x) \leftarrow defended(x) \}$$

where $\pi_<$ is defined as:

$$\begin{aligned}\pi_< \;=\; \{\, & lt(x,y) \leftarrow arg(x), arg(y), x < y, \\ & nsucc(x,z) \leftarrow lt(x,y), lt(y,z), \quad succ(x,y) \leftarrow lt(x,y), \mathbf{not}\ nsucc(x,y), \\ & ninf(y) \leftarrow lt(x,y), \quad inf(x) \leftarrow arg(x), \mathbf{not}\ ninf(x), \\ & nsup(x) \leftarrow lt(x,y), \quad sup(x) \leftarrow arg(x), \mathbf{not}\ nsup(x) \,\}\end{aligned}$$

where $<$ is a total order defined over the arguments in Ar. π_{def} is defined as:

$$\begin{aligned}\pi_{def} \;=\; \{\, & defended\_upto(x,y) \leftarrow inf(y), arg(x), \mathbf{not}\ att(y,x), \\ & defended\_upto(x,y) \leftarrow inf(y), in(z), att(z,y), att(y,x), \\ & defended\_upto(x,y) \leftarrow succ(z,y), defended\_upto(x,z), \mathbf{not}\ att(y,x), \\ & defended\_upto(x,y) \leftarrow succ(z,y), defended\_upto(x,z), in(v), att(v,y), att(y,x), \\ & defended(x) \leftarrow sup(y), defended\_upto(x,y) \,\}.\end{aligned}$$

Finally, preferred extensions of $AF = (Ar, att)$ are computed by answer sets of the program:

$$\pi_E^P = \mathcal{F}_{AF} \cup \pi_{adm} \cup \pi_< \cup \pi_p$$

where

$$\begin{aligned}\pi_p \;=\; \{\, & eq\_upto(x) \leftarrow inf(x), in(x), inN(x), \\ & eq\_upto(x) \leftarrow inf(x), out(x), outN(x), \\ & eq\_upto(x) \leftarrow succ(y,x), in(x), inN(x), eq\_upto(y), \\ & eq\_upto(x) \leftarrow succ(y,x), out(x), outN(x), eq\_upto(y), \\ & eq \leftarrow sup(x), eq\_upto(x), \\ & undefeated\_upto(x,y) \leftarrow inf(y), outN(x), outN(y), \\ & undefeated\_upto(x,y) \leftarrow inf(y), outN(x), \mathbf{not}\ att(y,x), \\ & undefeated\_upto(x,y) \leftarrow succ(z,y), undefeated\_upto(x,z), outN(y), \\ & undefeated\_upto(x,y) \leftarrow succ(z,y), undefeated\_upto(x,z), \mathbf{not}\ att(y,x), \\ & undefeated(x) \leftarrow sup(y), undefeated\_upto(x,y), \\ & inN(x) \vee outN(x) \leftarrow out(x), \quad inN(x) \leftarrow in(x), \\ & spoil \leftarrow eq, \quad spoil \leftarrow inN(x), inN(y), att(x,y), \\ & spoil \leftarrow inN(x), outN(y), att(y,x), undefeated(y), \\ & inN(x) \leftarrow spoil, arg(x), \quad outN(x) \leftarrow spoil, arg(x), \quad \leftarrow \mathbf{not}\ spoil \,\}.\end{aligned}$$

The encoding for preferred extensions is later simplified by Gaggl *et al*. [20]. They compute preferred extensions of $AF = (Ar, att)$ by answer sets of the program:

$$\pi_G^P = \mathcal{F}_{AF} \cup \pi_{adm} \cup \pi_g$$

where

$$\begin{aligned}\pi_g \;=\; \{\; & nontrivial \leftarrow out(x),\\ & witness(x) : out(x) \leftarrow nontrivial,\\ & spoil \;\vee\; witness(z) : att(z,y) \leftarrow witness(x), att(y,x),\\ & spoil \leftarrow witness(x), witness(y), att(x,y),\\ & spoil \leftarrow in(x), witness(y), att(x,y),\\ & witness(x) \leftarrow spoil, arg(x), \quad \leftarrow \mathbf{not}\; spoil, nontrivial\;\}.\end{aligned}$$

In the above, a *conditional literal* "$p(x) : q(x)$" represents a head of disjunctions of atoms $p(a)$ where $q(a)$ is true.

5.2 Wakaki and Nitta

Wakaki *et al*. [36] compute complete labellings of $AF = (Ar, att)$ by answer sets of the program:

$$\begin{aligned}\pi^C_W \;=\; \mathcal{F}_{AF} \;\cup\; \{\; & in(x) \leftarrow arg(x), \mathbf{not}\, ng(x), \quad ng(x) \leftarrow in(y), att(y,x),\\ & ng(x) \leftarrow undec(y), att(y,x), \quad out(x) \leftarrow in(y), att(y,x),\\ & undec(x) \leftarrow arg(x), \mathbf{not}\, in(x), \mathbf{not}\, out(x)\;\}.\end{aligned}$$

They compute stable labellings of $AF = (Ar, att)$ by answer sets of the program:

$$\pi^S_W = \pi^C_W \;\cup\; \{\leftarrow undec(x)\}.$$

Next, the grounded labelling of $AF = (Ar, att)$ is computed by the answer set of the program:

$$\pi^G_W = \pi^C_W \;\cup\; \Gamma \;\cup\; \Xi_G.$$

To define Γ and Ξ_G, we need additional notions. Define $\mathcal{I} = \{\, in(x) \mid x \in Ar\}$, $\mathcal{U} = \{\, undec(x) \mid x \in Ar\}$ and $\mathcal{C} = \{L_t \mid L_t$ is the term expressing an atom $L \in \mathcal{I} \cup \mathcal{U}\}$. Let $as(\Pi)$ be the set of answer sets of a program Π and ξ the cardinality of $as(\Pi)$, i.e., $\xi = |as(\Pi)|$. Define a bijective function $\psi : as(\Pi) \to \{1, \ldots, \xi\}$. Then $\psi(S) = j$ $(1 \le j \le \xi)$ for each $S \in as(\Pi)$. With this setting Γ is defined as:

$$\begin{aligned}\Gamma \;=\; & \{\, m_1(L_t) \leftarrow L \mid L \in \mathcal{I} \cup \mathcal{U} \text{ and } L_t \in \mathcal{C}\,\}\\ & \cup \{\, m_2(L_t, j) \leftarrow, \quad cno(j) \leftarrow \mid \psi(S) = j \;(1 \le j \le \xi) \text{ for } S \in as(\pi^C_W)\\ & \qquad\qquad \text{and } L_t \in \mathcal{C} \text{ for } L \in S \cap (\mathcal{I} \cup \mathcal{U})\,\}\\ & \cup \{\, i(L_t) \leftarrow \mid L \in \mathcal{I} \text{ and } L_t \in \mathcal{C}\,\}\\ & \cup \{\, u(L_t) \leftarrow \mid L \in \mathcal{U} \text{ and } L_t \in \mathcal{C}\,\}.\end{aligned}$$

Ξ_G is defined as:

$$\begin{aligned}\Xi_G \;=\; \{\; & c(y) \leftarrow cno(y), m_1(x), i(x), \mathbf{not}\, m_2(x,y),\\ & d(y) \leftarrow m_2(x,y), i(x), \mathbf{not}\, m_1(x),\\ & \leftarrow c(x), \mathbf{not}\, d(x)\;\}.\end{aligned}$$

Finally, preferred labellings of $AF = (Ar, att)$ are computed by answer sets of the program:

$$\pi^P_W = \pi^C_W \ \cup \ \Gamma \ \cup \ \Xi_P$$

where Ξ_P is defined as:

$$\begin{array}{rcl} \Xi_P & = & \{\, c(y) \leftarrow cno(y), m_1(x), i(x), \mathbf{not}\, m_2(x,y), \\ & & \quad d(y) \leftarrow m_2(x,y), i(x), \mathbf{not}\, m_1(x), \\ & & \quad \leftarrow d(x), \mathbf{not}\, c(x)\,\}. \end{array}$$

5.3 Comparison

Comparing our encodings with [19, 20, 36], we can observe the following facts.

- Our encoding produces a ground logic program that directly represents arguments and attack relations in an individual AF. The program Γ_{AF} reflects the structure of an individual AF, while additional rules in each AF-program (disjunctive facts, constraints, etc) are independent of individual AFs. Every encoding is done in polynomial time. On the other hand, encodings of [19, 20, 36] produce a set $\mathcal{F}_{AF}$ of ground facts that represents an individual AF, while rules for computing AF semantics are given as meta-rules that are independent of individual AFs.

- For the complete and the stable semantics, our encodings are similar to those of Wakaki $et\,al.$, while our encodings introduce no auxiliary atoms, i.e. the resulting stable models are identical to the sets of labelled arguments under the complete, stable and grounded semantics. Transformations for the grounded and the preferred semantics by Wakaki $et\,al.$ require computation of answer sets of π^C_W, hence those encodings are not done in polynomial time. The encodings by Egly $et\,al.$ and Gaggl $et\,al.$ also introduce auxiliary atoms.

- Our encoding is more compact than other encodings. This is particularly the case in the grounded and the preferred semantics. For the preferred semantics, the encodings of Egly $et\,al.$ and Wakaki $et\,al.$ "naively" maximize admissible extensions. On the other hand, the encoding of Gaggl $et\,al.$ and ours make implicit use of the minimality inherent to the answer set semantics. The encoding by Gaggl $et\,al.$ uses conditional literals that are not used in our encodings.

- Query answering, agreement, equivalence testing and ABA encoding in Section 4 would also be done using encodings by [19, 20, 36]. On the other hand, it is not straightforward to realize enforcement in Section 4.2 by [19, 20, 36].

The fact that our encoding is more compact than existing ones does not necessarily imply runtime efficiency, which is to be verified by empirical evaluation. On the other hand, the fact that our encoding provides a simple representation of AFs in ASP is considered an advantage from the viewpoint of knowledge representation and declarative problem solving.

6 Discussion

6.1 Reduction to Normal Programs

In Section 3 we introduce four different transformations from AFs to LPs. Of which, Π^C_{AF} (AF program under the complete semantics), Π^S_{AF} (AF program under the stable semantics), and Π^P_{AF} (AF program under the preferred semantics) are programs that contain both disjunction and NAF-literals. By contrast, Π^G_{AF} (AF program under the grounded semantics) is a normal program that contains NAF-literals but no disjunction. The program Π^C_{AF} is transformed to a semantically equivalent normal program by replacing the disjunctive fact (6) and the constraint (7) with the following three rules [4]:

$$\begin{aligned}
\texttt{in}(x) &\leftarrow \textbf{not}\ \texttt{out}(x),\ \textbf{not}\ \texttt{und}(x),\\
\texttt{out}(x) &\leftarrow \textbf{not}\ \texttt{in}(x),\ \textbf{not}\ \texttt{und}(x),\\
\texttt{und}(x) &\leftarrow \textbf{not}\ \texttt{in}(x),\ \textbf{not}\ \texttt{out}(x).
\end{aligned}$$

Likewise, the program Π^S_{AF} is transformed to a semantically equivalent normal program by replacing the disjunctive fact (8) and the constraint (9) with the following two rules:

$$\begin{aligned}
\texttt{in}(x) &\leftarrow \textbf{not}\ \texttt{out}(x),\\
\texttt{out}(x) &\leftarrow \textbf{not}\ \texttt{in}(x).
\end{aligned}$$

Thus, Π^C_{AF}, Π^S_{AF} and Π^G_{AF} can be represented by normal programs. In particular, Π^G_{AF} is a class of *stratified programs* with constraints. This class of programs is tractable [31] and thus matches the complexity of the grounded semantics [16, 17]. On the other hand, the program Π^P_{AF} *cannot* be transformed to a semantically equivalent normal program in polynomial time in general. This is because in Π^P_{AF} two atoms $\texttt{in}(x)$ and $\texttt{out}(x)$ may hold at the same time, so the disjunctive fact (8) is not replaced by the above mentioned normal rules. Generally, a disjunctive program can be transformed to a semantically equivalent to normal program if it is *head-cycle-free*.[3] Π^P_{AF} is not head-cycle-free in general because two atoms $\texttt{in}(c)$ and $\texttt{out}(c)$ have a cycle through the disjunctive fact $\texttt{in}(c) \vee \texttt{out}(c) \leftarrow$ in Example 3.4. Thus, Π^P_{AF} is in the class of logic programs which are more expressive and computationally expensive than others unless the polynomial hierarchy collapses. This is consistent with the complexity results of argumentation frameworks [16, 17], namely that deciding whether an argument is in every extension of an AF is coNP-complete for the stable semantics, while it is Π^P_2-complete for the preferred semantics.

6.2 Commonsense Reasoning

Answer set programming is used for reasoning with commonsense as default inference. Representing argumentation in logic programs enables us to combine argumentative

[3] A disjunctive program P is *head-cycle-free* if the dependency graph of P contains no directed cycle that goes through two different atoms in the head of the same disjunctive rule in P [4].

reasoning and commonsense reasoning. Consider, for instance, an argument between a driver and a policeman. The driver claims that he did not break the speed limit while the policeman claims that there is evidence that he did. The driver will receive a fine only if the outcome of this argument is that the driver did indeed break the speed limit. Let $AF = (Ar, att)$ be an argumentation framework that represents the discussion between the driver and the policeman. We denote the main argument in AF (i.e., the claim that the driver was speeding) by *speeding*. Then the situation can be represented by the logic program:

$$\Pi^G_{AF} \cup \{\, fined \leftarrow \mathtt{in}(speeding), \mathbf{not}\ \neg fined, \quad \neg fined \leftarrow \mathbf{not}\ \mathtt{in}(speeding)\,\}$$

where $speeding \in Ar$. The first rule represents that if the argument *speeding* is accepted in AF under the grounded semantics, then the driver is normally fined. The second rule represents that the driver is not fined unless the acceptance of speeding is proved. Here the grounded semantics is selected under the policy that "one is innocent unless proven guilty". Generally, argumentative reasoning and default reasoning are combined as $\Pi^X_{AF} \cup \Pi$ where X is either C, S, G or P. Π^X_{AF} encodes argumentation under a designated semantics. Π encodes a problem for solving that can refer justification states of arguments of AF in the body of rules in Π.

Further, AF-programs under different semantics can be combined for solving a problem. For instance, suppose a couple who argue a place to visit for a honeymoon. They have to choose one country to visit, while they want to visit several cities in the country as much as possible. If the couple do not reach an agreement on a country to visit, they will give up the travel and save money instead. Suppose that $AF_1 = (Ar_1, att_1)$ represents argumentation for country to visit, while $AF_2 = (Ar_2, att_2)$ represents argumentation for cities to visit (where $Ar_1 \cap Ar_2 = \emptyset$). In this setting, the grounded semantics of AF_1 is considered for deciding a country to visit, while the preferred semantics of AF_2 is considered for deciding cities to visit. The problem can be represented as

$$\begin{aligned}\Pi^G_{AF_1} \cup \Pi^P_{AF_2} \cup \Pi \cup \{\, & visit(country) \leftarrow \mathtt{in}(country),\\ & save\_money \leftarrow \mathbf{not}\ \mathtt{in}(country),\\ & visit(city) \leftarrow visit(country), located(city, country), \mathtt{in}(city)\ \mid\\ & country \in Ar_1 \text{ and } city \in Ar_2\}\end{aligned}$$

where Π is the background knowledge specifying $located(city, country)$. The program has stable models that represent candidate cities to visit in a single country if an argument $country$ is accepted in AF_1. Prakken [28] introduces a framework that combines skeptical epistemic reasoning based on the grounded semantics and credulous practical reasoning based on the preferred semantics. The above example shows that AF-programs could provide yet another method for combining skeptical and credulous reasoning in AFs that is realized in terms of ASP.

6.3 Related Work

Connections between argumentation frameworks and logic programming have been investigated by several researchers. Dung [13] first provides a transformation from

a logic program to an argumentation framework. He shows that logic programming semantics are characterized by extension based argumentation semantics in different ways. He also represents an argumentation framework in a logic program. Given an argumentation framework $AF = (Ar, att)$, Dung defines the logic program $P_{AF} = AGU \cup APU$ where

$$AGU = \{\, attack(x, y) \leftarrow |\ (x, y) \in att \,\},$$
$$APU = \{\, att(x) \leftarrow attack(y, x), acc(y), \quad acc(x) \leftarrow \mathbf{not}\ att(x) \,\}$$

where $acc(x)$ stands for "an argument x is acceptable" and $att(x)$ for "an argument x is defeated". For each extension E of AF, put

$$m(E) = AGU \cup \{\, acc(x) \mid x \in E\} \cup \{\, att(y) \mid y \text{ is attacked by some } x \in E \,\}.$$

He then shows that (i) E is a stable extension of AF iff $m(E)$ is a stable model of P_{AF}, and (ii) E is the grounded extension of AF iff $m(E) \cup \{\, \mathbf{not}\ att(a) \mid a \in E \,\}$ is the well-founded model [35] of P_{AF}. Our representation of AFs in logic programs is different from Dung's encoding in three ways. First, Dung captures a logic program as a meta-interpreter for argumentation systems. That is, an AF is given as input to a logic program, then the program produces a stable model or the well-founded model that characterizes a stable extension or the grounded extension of an AF. This is different from our encoding in which an individual AF is translated into a logic program that represents arguments and their attack relations at the object level. Secondly, in Dung's encoding different semantics of an AF correspond to different semantics of a logic program. By contrast, in our encoding, different semantics of an AF are all characterized by stable models of a transformed program. Thirdly, Dung encodes extension-based semantics of AFs, while we encode labelling-based semantics of AFs. In labelling-based semantics, rejected arguments and undecided arguments are distinguished by two labellings `out` and `und`, while extension-based semantics does not distinguish them.

Dung's meta-interpretative approach has been later extended by several researchers. Egly *et al.* [19] introduce ASP encodings for different AF semantics. Different from Dung's encodings, Egly *et al.* characterize complete extensions, stable extensions, grounded extensions, and preferred extensions of AFs in terms of stable models of logic programs. Gaggl *et al.* [20] extend the work and provide encodings for semi-stable and stage extensions of AFs. Wakaki *et al.* [36] represent different labelling-based semantics of AF (complete, stable, grounded, preferred and semi-stable) by answer sets of a transformed program. These studies use meta-interpretative encodings, that is, an instance of AFs is given as an input to a single meta-logic program under a particular argumentation semantics. As shown in Section 5, the meta-interpretative approach generally requires rather cumbersome encoding techniques and it becomes complicated for semantics such as the grounded semantics and the preferred semantics. The complication comes from the fact that a program has to encode tests for checking subset-minimality (or maximality) of admissible sets. To ease the problem, Dvořák *et al.* [18] use a built-in function for computing subset minimization, and Gaggl *et al.* [20] use "conditional disjunctions" in ASP.

Carballido *et al.* [12] provide a logic programming encoding for stable extensions

of $AF = (Ar, att)$ such that

$$\Psi_{AF} = \bigcup_{a \in Ar} \{\Psi(a) \cup \{acc(a) \leftarrow \neg d(a)\}\}$$

where

$$\Psi(a) = \{ \bigcup_{b:(b,a)\in att} \{d(a) \leftarrow \neg d(b)\}\} \cup \{ \bigcup_{b:(b,a)\in att} \{d(a) \leftarrow \bigwedge_{c:(c,b)\in att} d(c)\}\}.$$

The first rule of $\Psi(a)$ says that an argument a is defeated when any one of its attackers is not defeated. The second rule says that an argument a is defeated when all the arguments that defend a are defeated. They show that there is a one-to-one correspondence between stable extensions of AF and stable models of Ψ_{AF}. They also characterize the grounded extension of an AF in terms of the well-founded model of Ψ_{AF}. Nieves *et al*. [25] also show a one-to-one correspondence between preferred extensions of AF and stable models of the program which is obtained from Ψ_{AF} by replacing the rules $d(a) \leftarrow \neg d(b)$ in $\Psi(a)$ with $d(a) \vee d(b)$. Different from our transformation, $\Psi(a)$ considers not only attackers but also defenders (i.e., the argument c defends a in the second part). The above transformation is simple in the sense that it uses only one predicate d meaning "an argument x is defeated", while they do not provide ASP encodings for complete or grounded extensions of AFs.

Wu *et al*. [38] introduce a translation from AFs into logic programs. Given $AF = (Ar, att)$, its associated logic program is defined as

$$P_{AF} = \{\, a \leftarrow \mathbf{not}\, b_1, \ldots, \mathbf{not}\, b_n \mid a \in Ar \text{ and } a^- = \{b_1, \ldots, b_n\}\, (n \geq 0)\,\}.$$

It is shown that there is a one-to-one correspondence between complete labellings of AF and 3-valued stable models of P_{AF} [38]. The result is later extended to the correspondences between stable (resp. grounded, preferred, semi-stable) labellings of AF and stable (resp. well-founded, regular, L-stable) models of P_{AF} [8]. The result of Wu *et al*. is similar to ours in the sense that they map arguments and attack relations into rules of a logic program at the object level. On the other hand, they relate different semantics of AFs to different semantics of logic programs. By contrast, we characterize different semantics of AFs by a single semantics—(2-valued) stable model semantics of logic programs. We do not study encoding semi-stable labellings in this paper.

Caminada and Schulz [10] provide a transformation from flat ABA frameworks to logic programs. Given an ABA framework $ABA = (\mathbf{L}, \mathcal{R}, \mathcal{A}, {}^-)$, an associated logic program P_{ABA} is defined as

$$\begin{aligned} P_{ABA} = \{\, & x \leftarrow y_1, \ldots, y_m, \mathbf{not}\, z_1, \ldots, \mathbf{not}\, z_n \mid \\ & x \leftarrow y_1, \ldots, y_m, \zeta_1, \ldots, \zeta_n \in \mathcal{R} \text{ and } \bar{\zeta_i} = z_i \text{ for } i = 1, \ldots, n\,\}. \end{aligned}$$

Then they show that 3-valued stable (resp. well-founded, regular, 2-valued stable, ideal) models of P_{ABA} correspond to complete (resp. grounded, preferred, stable, ideal) assumption labellings of the ABA framework ABA. As such, they relate different semantics of ABA to different semantics of LPs. In this paper, we characterize complete

Table 1: Transformations from AFs to LPs

reference	representation	semantics	transformation
Dung [13]	meta-interpretative	extension	stable ext. $\rightarrow$ stable model (1-to-1) grounded ext. $\rightarrow$ well-founded model (1-to-1)
Nieves *et al.* [25]	object level	extension	preferred ext. $\rightarrow$ stable model (1-to-1)
Carballido *et al.* [12]	object level	extension	stable ext. $\rightarrow$ stable model (1-to-1) grounded ext. $\rightarrow$ well-founded model (1-to-1)
Wu *et al.* [38]	object level	labelling	complete labelling $\rightarrow$ 3-valued stable model (1-to-1)
Wakaki *et al.* [36]	meta-interpretative	labelling	complete/stable/grounded/preferred/semi-stable lab. $\rightarrow$ stable model (many-to-one)
Egly *et al.* [19]	meta-interpretative	extension	complete/stable/grounded/preferred ext. $\rightarrow$ stable model (many-to-one)
Dvořák *et al.* [18]	meta-interpretative	extension	grounded/preferred/semi-stable/stage ext. $\rightarrow$ stable model (many-to-one)
Gaggl *et al.* [20]	meta-interpretative	extension	preferred/semi-stable/stage ext. $\rightarrow$ stable model (many-to-one)
Caminada *et al.* [8]	object level	labelling	stable/grounded/preferred/semi-stable lab. $\rightarrow$ stable/well-founded/regular/L-stable model (1-to-1)
Our current study	object level	labelling	complete/stable/grounded/preferred lab. $\rightarrow$ stable model (many-to-one)

assumption labellings of ABA in terms of stable models of LPs. The result is extended to encode different semantics of ABA in stable models of LPs using the result of [33].

We summarize comparisons with related studies on the transformation from AFs to LPs in Table 1. From the representation viewpoint, in the meta-interpretative approach, individual AFs are given as input to a single metalogic program that produces selected models characterizing input AF semantics. In the object level approach, individual AFs are transformed to corresponding logic programs whose selected models characterize input AF semantics. From the semantic viewpoint, in the extension-based approach, extensions of an AF are characterized by selected models of a transformed logic program. In the labelling-based approach, labellings of an AF are characterized by selected models of a transformed logic program. In the transformational viewpoint, in one-to-one mapping, different semantics of an AF are characterized by different semantics of a transformed LP. In many-to-one mapping, different semantics of an AF are characterized by a single semantics of a transformed LP. By the table, it is observed that our current study realizes many-to-one transformations from AFs to LPs at the object level under the labelling semantics, which has not been studied in the literature.

7 Conclusion

We introduced methods of representing argumentation frameworks in terms of logic programs and showed their applications. The proposed transformations encode different AF semantics by stable models of LPs in a simple and uniform manner. This enables one to use existing answer set solvers for computing argumentation semantics and solving various problems of AFs. Moreover, several techniques developed in LPs are directly applied to transformed AF-programs. For instance, the equivalence issue of AFs is converted to the equivalence issue of the transformed AF programs, opti-

mization of AFs is viewed as optimization of AF-programs, revision of AFs is realized by revision of AF-programs, etc. In this way, the result of this study implies potential use of rich LP techniques in AF problems, and contributes to strengthen the relationship between formal argumentation and logic programming. In future study, we plan to implement ASP encodings of AFs and solve AF problems using ASP solvers. We will also argue possibilities of importing LP techniques into AFs and investigate ASP encodings for other semantics of AFs (such as semi-stable labellings).

Acknowledgments

We thank anonymous referees for useful comments.

References

[1] Amgoud, L. and Vesic, S.: A formal analysis of the role of argumentation in negotiation dialogues. *Journal of Logic and Computation* 22, 2012, 957–978.

[2] Baroni, P., Caminada, M. and Giacomin, M.: An introduction to argumentation semantics. *The Knowledge Engineering Review* 26, 2011, 365–410.

[3] Baumann, R. and Brewka, G.: Expanding argumentation frameworks: enforcing and monotonicity results. In: *Proc. 3rd Int'l Conf. Computational Models of Argument. Frontiers in AI and Applications* 216, IOS Press, 2010, 75–86.

[4] Ben-Eliyahu, R. and Dechter, R.: Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence* 12, 1994, 53–87.

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

[6] Brewka, G., Eiter, T. and Truszczyński, M.: Answer set programming at a glance. *Communications of the ACM* 54, 2011, 92–103.

[7] Caminada, M. and Gabbay, D.: A logical account of formal argumentation. *Studia Logica* 93, 2009, 109–145.

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

[9] Caminada, M., Sá, S., Alcântara, J. and Dvořák, W.: On the difference between assumption-based argumentation and abstract argumentation. In: *Proc. 25th Benelux Conf. Artificial Intelligence*, 2013, 25–32.

[10] Caminada, M. and Schulz, C.: On the equivalence between assumption-based argumentation and logic programming. In: *Proc. 1st Int'l Workshop on Argumentation and Logic Programming*, Cork, Ireland, 2015.

[11] Caminada, M. and Sakama, C.: On the issue of argumentation and informedness. *New Frontiers in Artificial Intelligence* (H. Otake *et al.*, Eds.), LNAI, vol. 10091, Springer, 2016.

[12] Carballido, J. L., Nieves, J. C. and Osorio, M.: Inferring preferred extensions by pstable semantics. *Inteligencia Artificial* 41, 2009, 38–53.

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

[14] Dung, P. M., Mancarella, P., and Toni, F.: Computing ideal sceptical argumentation. *Artificial Intelligence* 171, 2007, 642–674.

[15] Dung P. M., Kowalski, R. A. and Toni, F.: Assumption-based argumentation. In: *Argumentation in Artificial Intelligence* (I. Rahwan and G. R.. Simari, Eds.), Springer, 2009, 199–218.

[16] Dunne, P. E. and Wooldridge, M.: Complexity of abstract argumentation. In: *Argumentation in Artificial Intelligence* (I. Rahwan and G. R.. Simari, Eds.), Springer, 2009, 85–104.

[17] Dvořák, W. and Woltran, S.: On the intertranslatability of argumentation semantics. *J. Artificial Intelligence Research* 41, 2011, 445–475.

[18] Dvořák, W., Gaggl, S. A., Wallner, J. P. and Woltran, S. Making use of advances in answer-set programming for abstract argumentation systems. In: *Proc. 19th Int'l Conf. Applications of Declarative Programming and Knowledge Management, Revised Selected Papers*, LNAI, vol. 7773, Springer, 2013, 114–133.

[19] Egly, U., Gaggl, S. A. and Woltran, S.: Answer-set programming encodings for argumentation frameworks. *Argument and Computation* 1, 2010, 147–177.

[20] Gaggl, S. A., Manthey, N., Ronca, A., Wallner, J. P. and Woltran, S.: Improved answer-set programming encodings for abstract argumentation. *Theory and Practice of Logic Programming* 15, 2015, 434–448.

[21] Gelfond, M. and Lifschitz, V.: The stable model semantics for logic programming. In: *Proc. 5th Int'l Conf. and Symp. Logic Programming*, MIT Press, 1988, 1070–1080.

[22] Gelfond, M. and Lifschitz, V.: Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9, 1991, 365–385.

[23] Marquis, S. C, Konieczny, S., Mailly, J.-G. and Marquis, P.: Extension enforcement in abstract argumentation as an optimization problem. In: *Proc. 24th Int'l Joint Conf. Artificial Intelligence*, 2015, 2876–2882.

[24] Modgil, S. and Caminada, M.: Proof theories and algorithms for abstract argumentation framework. In: *Argumentation in Artificial Intelligence* (I. Rahwan and G. R.. Simari, Eds.), Springer, 2009, 105–129.

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

[26] Oikarinen, E. and Janhunen, T.: Verifying the equivalence of logic programs in the disjunctive case. In: *Proc. 7th Int'l Conf. Logic Programming and Nonmonotonic Reasoning*, LNAI, vol. 2923, Springer, 2004, 180–193.

[27] Oikarinen, E. and Woltran, S.: Characterizing strong equivalence for argumentation frameworks. *Artificial Intelligence* 175, 2011, 1985–2009.

[28] Prakken, H.: Combining sceptical epistemic reasoning with credulous practical reasoning (corrected version). Revised version of the paper originally published in: *Proc. 1st Int'l Conf. Computational Models of Argument. Frontiers in AI and Applications* 144, IOS Press, 2006, 311–322.

[29] Przymusinski, T. C.: The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae* 13, 1990, 445–463.

[30] Sakama, C.: Dishonest arguments in debate games. In: *Proc. 4th Int'l Conf. Computational Models of Argument. Frontiers in AI and Applications* 245, IOS Press, 2012, 177–184.

[31] Schlipf, J.: Complexity and undecidability results for logic programming. *Annals of Mathematics and Artificial Intelligence* 15, 1995, 257–288.

[32] Schulz, C. and Toni, F.: Complete assumption labellings. In: *Proc. 5th Int'l Conf. Computational Models of Argument. Frontiers in AI and Applications* 266, IOS Press, 2014, 405–412.

[33] Schulz, C.: Assumption labellings. Technical Report, Imperial College, London, UK, 2015.

[34] Toni, F. and Sergot, M.: Argumentation and answer set programming. In: *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays in Honor of Michael Gelfond* (M. Balduccini and T. C. Son, Eds.), LNCS, vol. 6565, Springer, 2011, 164–180.

[35] Van Gelder, A., Ross, K. and Schlipf, J. S.: The well-founded semantics for general logic programs. *J. ACM* 38, 1991, 620–650.

[36] Wakaki, T. and Nitta, K.: Computing argumentation semantics in answer set programming. *New Frontiers in Artificial Intelligence* (H. Hattori *et al.*, Eds.), LNAI, vol. 5447, Springer, 2009, 254–269.

[37] Wallner, J. P., Niskanen, A. and Jarvisalo, M.: Complexity results and algorithms for extension enforcement in abstract argumentation. In: *Proc. 30th AAAI Conf. Artificial Intelligence*, 2016, 1088–1094.

[38] Wu, Y., Caminada, M. and Gabbay, D. M.: Complete extensions in argumentation coincide with 3-valued stable models in logic programming. *Studia Logica* 93, 2009, 383–403.

[39] You, J.-H. and Yuan, L.-Y.: A three-valued semantics for deductive databases and logic programs. *J. Computer and System Science* 49, 1994, 334–361.