Fundamenta Informaticae XXI (2001) 1–15
DOI 10.3233/FI-2012-0000
IOS Press

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 at the object level, and different semantics of AFs are uniformly characterized by stable models of transformed programs. We show how transformed programs can be used for solving AF problems such as query-answering, enforcement of arguments, and agreement of different AFs. The results exploit new connection between AF and logic programming, and enables one to realize AF on top of existing answer set solvers.

Keywords: argumentation framework, answer set programming, transformation.

1. Introduction

Logic programming (LP) and argumentation framework (AF) are two different languages for representing knowledge, while close connections in their semantics have been revealed by several researchers [18]. Dung [5] introduces a transformation from LP to AF and shows that stable models (resp. the well-founded model) of a logic program correspond to stable extensions (resp. the grounded extension) of a

Address for correspondence: Chiaki Sakama, Wakayama University, 930 Sakaedani, Wakayama 640-8510, Japan

transformed argumentation framework. The results are later enhanced by connections between 3-valued stable models of LP and complete extensions of AF [20], and regular models of LP and preferred extensions of AF [4]. On the other side, Dung [5] introduces a converse transformation from AF to LP, and shows that stable extensions (resp. the grounded extension) of an argumentation framework are computed as stable models (resp. the well-founded model) of a transformed logic program. The results are also extended to relating preferred, complete, or other extensions of AF to their corresponding semantics of transformed logic programs [8, 9, 15, 20, 19].

In his transformation from AF to LP, Dung uses LP as a meta-interpreter for computing arguments. The idea is inherited by studies [8, 9, 19] which use answer set programming (ASP) [10, 11, 13] as a meta-interpreter for processing AF given as input. An important difference is that Dung characterizes different semantics of AF in terms of different semantics of LP, while those studies [8, 9, 19] characterize different semantics of AF in terms of the answer set semantics of LP. Such characterization is important in the sense that it enables one to use existing ASP solvers for computing different semantics of AF. On the other hand, encoding AF semantics in meta-interpretative LP results in quite complicated programs, especially for problems under preferred semantics that are located at the second level of the polynomial hierarchy, and the resulting program are hardly accessible for non-experts in ASP [8].

In this paper, we introduce transformations from AF to LP such that different semantics of AF (i.e., complete, stable, grounded, preferred) are characterized by the answer set semantics of LP in a simple and uniform manner. Different from [8, 9, 19], we do not take the meta-interpretative approach, but translate an argumentation framework into a logic program at the object level. This viewpoint is similar to a translation given in [4, 20] where arguments and attack relations are encoded by rules of logic programs. The translation given in [4, 20], however, maps different semantics of AF into different semantics of LP, so that it does not realize computing different semantics of AF in terms of ASP. Among the meta-interpretative ASP approaches, [8, 9] compute extension-based semantics of AF, while [19] computes labelling-based semantics of AF. It is known that there is a one-to-one correspondence between extension-based semantics and labelling-based semantics of AF [3]. Unlike extension-based semantics which computes accepted arguments, labelling-based semantics not only computes accepted arguments but distinguishes rejected arguments from undecided arguments. In this paper, we compute labelling-based semantics of AF in terms of ASP.

We introduce transformations that are used for computing different semantics of AF in terms of ASP, and we show that transformed programs can be used for solving several AF problems on top of ASP. The rest of this paper is organized as follows. In Section 2 we review basic notions of argumentation frameworks and answer set programming. Section 3 introduces transformations from AF to LP which translate complete semantics, stable semantics, grounded semantics and preferred semantics into answer sets of transformed logic programs. Section 4 uses transformed programs for realizing query-answering, enforcement, and agreement in AF. Section 5 discusses related issues and Section 6 summarizes the paper.

2. Preliminaries

2.1. Argumentation Framework

This section reviews formal argumentation frameworks which are in [3, 5].

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 $att \subseteq Ar \times Ar$. An argument a *attacks* an argument 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 directed arcs from a to b exist 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 $a \in Ar$, an argument a is *accepted* (resp. *rejected* or *undecided*) in $\mathcal{L}$ (written as $\mathtt{in}(a)$ (resp. $\mathtt{out}(a)$ or $\mathtt{und}(a)$)). We call $\mathtt{in}(a)$, $\mathtt{out}(a)$ and $\mathtt{und}(a)$ *labelled arguments*. A set S of labelled arguments for AF under $\mathcal{L}$ is defined as $S = \{\, \ell(x) \mid \mathcal{L}(x) = \ell$ for $x \in Ar \,\}$ where ℓ is either $\mathtt{in}$, $\mathtt{out}$ or $\mathtt{und}$.

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 under complete labelling if and only if none of its attackers is $\mathtt{in}$, and at least one of its attackers is undecided.

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

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

The grounded or preferred labelling is also characterized using $\mathtt{out}$ and $\mathtt{und}$ as follows [3].

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

There is a one-to-one correspondence between the set $\mathtt{in}(\mathcal{L})$ with a complete (resp. stable, grounded, preferred) labelling $\mathcal{L}$ of AF and a *complete* (resp. *stable*, *grounded*, *preferred*) *extension* of AF [3].

[1]A preferred labelling $\mathcal{L}$ does not imply the minimality of $\mathtt{und}(\mathcal{L})$, while it is implied by a *semi-stable labelling* [3].

2.2. Answer Set Programming

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

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

where each a_i is a ground atom. **not** is *negation as failure* (NAF) and $\mathbf{not}\, a$ is called an *NAF-literal*. The left-hand side of the rule is the *head*, and the right-hand side is the *body*. For each rule r of the above form, $\mathit{head}(r)$, $\mathit{body}^+(r)$, and $\mathit{body}^-(r)$ 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 $\mathit{head}(r) = \emptyset$; and r is a *(disjunctive) fact* if $\mathit{body}^+(r) = \mathit{body}^-(r) = \emptyset$. 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 $|\,\mathit{head}(r)\,| \leq 1$ for every rule r in P. A program P is called a *positive program* if $\mathit{body}^-(r) = \emptyset$ for every rule r in P. A positive program P is called a *definite program* if $|\,\mathit{head}(r)\,| = 1$ for every rule r in P.

The semantics of a program is defined by the *stable model semantics* (or *answer set semantics*) [10, 11]. Let B be the *Herbrand base* of a program. An interpretation $I \subseteq B$ *satisfies* a rule r of the form (1) if $\mathit{body}^+(r) \subseteq I$ and $\mathit{body}^-(r) \cap I = \emptyset$ imply $\mathit{head}(r) \cap I \neq \emptyset$. In particular, I satisfies a constraint r such that $\mathit{head}(r) = \emptyset$ if $\mathit{body}^+(r) \setminus I \neq \emptyset$ or $\mathit{body}^-(r) \cap I \neq \emptyset$. 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 wrt I): $P^I = \{\, a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m \mid (a_1 \vee \cdots \vee a_l \;\leftarrow\; a_{l+1}\,, \ldots,\, a_m,\, \mathbf{not}\, a_{m+1}\,, \ldots,\, \mathbf{not}\, a_n) \in P$ 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 general. A program is *consistent* if it has at least one stable model; otherwise, the program is *inconsistent*. Representing knowledge by logic programs under the stable model semantics is called *stable model programming* [13], or more popularly, *answer set programming* (ASP). In this paper, we use the terms logic programming and answer set programming interchangeably.

3. Transforming AF to LP

Given an argumentation framework $AF = (Ar, att)$, the set B of labelled arguments is constructed as

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

We view Ar as a set of constants and consider B as the Herbrand base on which a logic program is constructed. We first introduce a set of rules which will be used in what follows.

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

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

The rule (2) states that an argument x is labelled $\mathtt{in}$ if every attacker $y_1, \ldots, y_k$ of x is labelled $\mathtt{out}$. The rule (3) states that an argument x is labelled $\mathtt{out}$ if there is an attacker y which is labelled $\mathtt{in}$. The constraint (4) states that every $\mathtt{in}$-labelled argument x has no attacker y which is not labelled $\mathtt{out}$. The constraint (5) states that every $\mathtt{out}$-labelled argument x has at least one attacker y_i $(1 \le i \le k)$ which is labelled $\mathtt{in}$. (4) and (5) represent the *admissibility* condition of [3]. 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 $\mathtt{in}$. 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 $\mathtt{out}$.

3.1. Complete Semantics

We first consider representing the complete semantics of AF by ASP.

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:

$$\begin{aligned}\Pi^C_{AF} \;=\; \Gamma_{AF} \;&\cup\; \{\, \mathtt{in}(x) \vee \mathtt{out}(x) \vee \mathtt{und}(x) \leftarrow \;\mid\; x \in Ar \,\} && (6)\\ &\cup\; \{ \leftarrow \mathtt{in}(x),\, \mathtt{out}(x), \;\; \leftarrow \mathtt{in}(x),\, \mathtt{und}(x), \;\; \leftarrow \mathtt{out}(x),\, \mathtt{und}(x) \;\mid\; x \in Ar \,\}. && (7)\end{aligned}$$

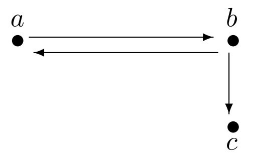
The rules of Γ_{AF} represent the necessary and sufficient condition of $\mathtt{in}$ or $\mathtt{out}$ labellings under the complete labelling of Definition 2.3. In addition, the disjunctive fact (6) states that every argument is labelled by either $\mathtt{in}$, $\mathtt{out}$ or $\mathtt{und}$, and the constraints (7) state that each argument cannot take two different labellings.[2] The transformed program encodes the complete labelling of AF.

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

Proof:
The result follows from the definition of complete labelling and the rules of Π^C_{AF}. □

Example 3.1. Suppose $AF = (\{a, b, c\}, \{(a, b), (b, a), (b, c)\})$. Then Π^C_{AF} consists of the rules:

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



Π^C_{AF} has three stable models:

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

which are equivalent to three sets of labelled arguments under the complete semantics of AF.

[2]The program Π^C_{AF} is also represented by a normal program. We will argue the issue in Section 5.1.

3.2. Stable Semantics

We next consider representing the stable semantics of AF by ASP.

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.

$$\begin{aligned} \Pi^S_{AF} \;=\; \Gamma_{AF} \;\; & \cup \;\; \{\, \mathtt{in}(x) \vee \mathtt{out}(x) \leftarrow \;\mid\; x \in Ar\,\} && (8)\\ & \cup \;\; \{\leftarrow \mathtt{in}(x),\, \mathtt{out}(x) \;\mid\; x \in Ar\,\}. && (9)\end{aligned}$$

In contrast to Π^C_{AF}, the program Π^S_{AF} introduces disjunctive facts (8) and constraints (9). This is because every argument in stable labelling is either in or out (but not both).

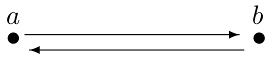
Theorem 3.2. *Let* $AF = (Ar, att)$ *be an argumentation framework and* Π^S_{AF} *an associated AF-program under the stable semantics. Then the sets of labelled arguments under the stable semantics of AF coincide with the stable models of* Π^S_{AF}.

Proof:
A stable labelling is a complete labelling $\mathcal{L}$ such that $\mathtt{und}(\mathcal{L}) = \emptyset$. 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:

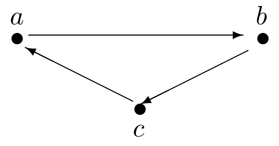
$\mathtt{in}(a) \leftarrow \mathtt{out}(b),\quad \mathtt{in}(b) \leftarrow \mathtt{out}(a),\quad \mathtt{out}(a) \leftarrow \mathtt{in}(b),$
$\mathtt{out}(b) \leftarrow \mathtt{in}(a),\quad \leftarrow \mathtt{in}(a),\, \mathbf{not}\, \mathtt{out}(b),\quad \leftarrow \mathtt{in}(b),\, \mathbf{not}\, \mathtt{out}(a),$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



$\Pi^S_{AF_1}$ has two stable models $\{\, \mathtt{in}(a),\, \mathtt{out}(b)\,\}$ and $\{\, \mathtt{out}(a),\, \mathtt{in}(b)\,\}$ which are equivalent to two sets of labelled arguments under the stable semantics.

Next suppose $AF_2 = (\{a, b, c\}, \{(a, b), (b, c), (c, a)\})$. Then $\Pi^S_{AF_2}$ consists of rules:

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



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

3.3. Grounded Semantics

To represent the grounded semantics of AF, we define the next program.

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.

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

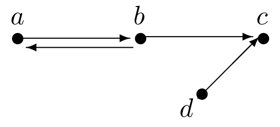
Unlike Π^C_{AF} or Π^S_{AF}, Π^G_{AF} does not have disjunctive facts. Instead, it has a rule (10) which states that an argument x is labelled und if neither $\mathtt{in}(x)$ nor $\mathtt{out}(x)$ is derived in Γ_{AF}.

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

Proof:
Let M be the set of labelled arguments under the grounded semantics of AF. Since M is also a set of labelled arguments under 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 Γ^M_{AF} (a reduct of Γ_{AF} wrt M) is a definite 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}. The converse is shown in a similar way. □

Example 3.3. Suppose $AF = (\{a, b, c, d\}, \{(a, b), (b, a), (b, c), (d, c)\})$. Then Π^G_{AF} consists of rules:

$\mathtt{in}(a) \leftarrow \mathtt{out}(b)$, $\mathtt{in}(b) \leftarrow \mathtt{out}(a)$, $\mathtt{in}(c) \leftarrow \mathtt{out}(b), \mathtt{out}(d)$, $\mathtt{in}(d) \leftarrow$,
$\mathtt{out}(a) \leftarrow \mathtt{in}(b)$, $\mathtt{out}(b) \leftarrow \mathtt{in}(a)$, $\mathtt{out}(c) \leftarrow \mathtt{in}(b)$,
$\mathtt{out}(c) \leftarrow \mathtt{in}(d)$, $\leftarrow \mathtt{in}(a), \mathbf{not}\,\mathtt{out}(b)$, $\leftarrow \mathtt{in}(b), \mathbf{not}\,\mathtt{out}(a)$,
$\leftarrow \mathtt{in}(c), \mathbf{not}\,\mathtt{out}(b)$, $\leftarrow \mathtt{in}(c), \mathbf{not}\,\mathtt{out}(d)$, $\leftarrow \mathtt{out}(a), \mathbf{not}\,\mathtt{in}(b)$,
$\leftarrow \mathtt{out}(b), \mathbf{not}\,\mathtt{in}(a)$, $\leftarrow \mathtt{out}(c), \mathbf{not}\,\mathtt{in}(b), \mathbf{not}\,\mathtt{in}(d)$, $\leftarrow \mathtt{out}(d)$,
$\mathtt{und}(x) \leftarrow \mathbf{not}\,\mathtt{in}(x), \mathbf{not}\,\mathtt{out}(x)$ (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 is equivalent to the set of labelled arguments under the grounded semantics of AF.

3.4. Preferred Semantics

To represent preferred semantics of AF, we extend the Herbrand base as follows. Given an argumentation framework $AF = (Ar, att)$, let

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

In this section, we construct a logic program over B'.

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.

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

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 IN-labelling under the preferred semantics if it is labelled in under the stable labelling; while the rule (12) (called OUT-rule) means that an argument x has an OUT-labelling under the preferred semantics if it is labelled out under the stable labelling. On the other hand, the rule (13) (called UND-rule) means that an argument x has an UND-labelling under the preferred semantics if it does not have a consistent stable labelling (i.e., in and out are assigned at the same time). Π^P_{AF} introduces these IN-OUT-UND rules to Π^S_{AF} instead of constraints (9) of Definition 3.3.

Theorem 3.5. *Let $AF = (Ar, att)$ be an argumentation framework and Π^P_{AF} an associated AF-program under the preferred semantics. Then there is a one-to-one correspondence between the sets of labelled arguments under the preferred labelling of AF and the stable models of Π^P_{AF}.*

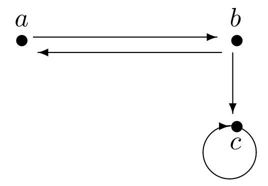
Proof:
Suppose that $\mathcal{L}$ is a preferred labelling of AF. Then $\mathcal{L}$ is either a stable labelling or not. (i) If $\mathcal{L}$ is a stable labelling, the set of labelled arguments under the stable labelling $\mathcal{L}$ of AF coincide with stable models of Π^S_{AF} (Theorem 3.2). Those stable models of Π^S_{AF} do not simultaneously contain both $\mathtt{in}(x)$ and $\mathtt{out}(x)$ for any $x \in Ar$. Then replacing the constraint $\leftarrow \mathtt{in}(x), \mathtt{out}(x)$ of Π^S_{AF} with rules (11) and (12), there is a one-to-one correspondence between stable models of Π^S_{AF} and stable models of $\Pi^P_{AF} \setminus \{(13)\}$. That is, M is a stable model of Π^S_{AF} iff $N = M \cup \{\mathtt{IN}(x) \mid \mathtt{in}(x) \in M\} \cup \{\mathtt{OUT}(x) \mid \mathtt{out}(x) \in M\}$ is a stable model of $\Pi^P_{AF} \setminus \{(13)\}$. Hence, the result holds. (ii) Else if $\mathcal{L}$ is not a stable labelling, there is some argument $x \in Ar$ which cannot be labelled by one of in and out in a way that satisfies the condition of Definition 2.3. In this case, assuming $\mathcal{L}(x) = \mathtt{in}$ leads to $\mathcal{L}(x) = \mathtt{out}$ and vice versa, thereby becomes $\mathcal{L}(x) = \mathtt{und}$. The situation is represented by the UND-rule (13) which produces $\mathtt{UND}(x)$ for such arguments. Consequently, $\mathcal{L}(x) = \mathtt{in}$ (resp. $\mathcal{L}(x) = \mathtt{out}$ or $\mathcal{L}(x) = \mathtt{und}$) under a preferred labelling $\mathcal{L}$ iff there is a stable model M of Π^P_{AF} such that $\mathtt{IN}(x) \in M$ (resp. $\mathtt{OUT}(x) \in M$ or $\mathtt{UND}(x) \in M$). □

Corollary 3.6. $AF = (Ar, att)$ has a stable labelling iff Π^P_{AF} has a stable model M such that $\mathtt{UND}(x) \notin M$ 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:

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

$\mathtt{IN}(x) \leftarrow \mathtt{in}(x),\ \textbf{not}\ \mathtt{out}(x)$ (where $x \in \{a,b,c\}$),
$\mathtt{OUT}(x) \leftarrow \textbf{not}\ \mathtt{in}(x),\ \mathtt{out}(x)$ (where $x \in \{a,b,c\}$),
$\mathtt{UND}(x) \leftarrow \mathtt{in}(x),\ \mathtt{out}(x)$ (where $x \in \{a,b,c\}$).



$\Pi^P_{AF_1}$ has two stable models:

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

Then two sets $\{\, \mathtt{OUT}(a),\ \mathtt{IN}(b),\ \mathtt{OUT}(c) \,\}$ and $\{\, \mathtt{IN}(a),\ \mathtt{OUT}(b),\ \mathtt{UND}(c) \,\}$ correspond to two sets of labelled arguments under the preferred semantics (of which the first one also corresponds to the stable semantics).

4. Applications

In Section 3 we provide encodings of different argumentation semantics using stable models. This enables us to use ASP techniques for solving various problems in AF. In this section, we provide some applications of AF programs under the complete semantics. The methods are directly applicable to AF programs under the stable, grounded or preferred semantics.

4.1. Query Answering

To see whether an argument x is acceptable in some extension of an AF, one can use a *dispute tree* in which an argument x is put in the root of the tree and every branch from the root to leafs is a dispute [14]. Using an AF program under the complete semantics, the question whether an argument x is accepted or not in a complete extension of AF is checked as follows.

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

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

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

Theorem 4.1 uses a standard query-answering technique of ASP. In this way, we can use existing ASP solvers such as [12, 17] for checking credulous or skeptical entailment of an argument in AF.

4.2. Enforcement

The enforcement problem is to check whether it is possible to modify a given AF in such a way that a desired set of arguments becomes a subset of an extension by adding new arguments which may interact with old ones [2]. To encode the problem, we introduce the universal argumentation framework. The *universal argumentation framework* (UAF) is an argumentation framework (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. With this setting, an argumentation framework is defined as a pair $AF = (Ar, att)$ where Ar is a finite subset of U and $att = att_U \cap (Ar \times Ar)$ [16]. We say that AF is a sub-AF of the UAF. For the set U of all arguments, the set of all labelled arguments is defined as $B_U = \{\, \mathtt{in}(x),\ \mathtt{out}(x),\ \mathtt{und}(x) \mid x \in U \,\}$. The *enforcement problem* is defined as follows.

Definition 4.1. (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' such that (i) $AF' = (Ar', att')$ where $Ar \subseteq Ar' \subseteq U$ and $att' = att_U \cap (Ar' \times Ar')$, and (ii) AF' has a complete labelling $\mathcal{L}$ such that $\mathcal{L}(x) = \ell$ for any $\ell(x) \in E$, then AF *satisfies* the enforcement E under the complete semantics.

We introduce an AF program for the enforcement problem.

Definition 4.2. (AF program for enforcement)
Let $AF = (Ar, att)$ be a sub-AF of $UAF = (U, att_U)$. Then an *AF-program for enforcement under the complete semantics* $\varepsilon\Pi^C_{AF}$ is defined as

$$\varepsilon\Pi^C_{AF} = \Pi^C_{UAF} \setminus \{\, \mathtt{in}(x) \leftarrow,\quad \leftarrow \mathtt{out}(x) \mid x \in U \setminus Ar \,\}. \tag{14}$$

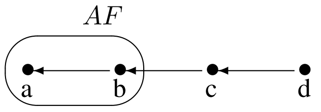
By definition, $\varepsilon\Pi^C_{AF}$ is obtained from Π^C_{UAF} by removing every fact $\mathtt{in}(x) \leftarrow$ and every constraint $\leftarrow \mathtt{out}(x)$ for $x \in U \setminus Ar$. These facts and constraints appear if an argument $x \in U \setminus Ar$ has no attacker (by (2) and (5)). The program Π^C_{UAF} represents arguments and attack relations over $UAF = (U, att_U)$, while AF does not necessarily require all arguments in $U \setminus Ar$ and attack relations $att_U \setminus att$ to satisfy a given enforcement. Hence, the fact $\mathtt{in}(x) \leftarrow$ and the constraint $\leftarrow \mathtt{out}(x)$ for $x \in U \setminus Ar$ are removed as they may not be used. Using the program, the enforcement problem is computed in ASP as follows.

Theorem 4.2. *Let $AF = (Ar, att)$ be an argumentation framework and $\varepsilon\Pi^C_{AF}$ a 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.*

The result of Theorem 4.2 is analogous to the result of Theorem 4.1. The difference is that to make $\ell(x)$ true, any argument that is not in Ar but in $U \setminus Ar$ is introduced with an appropriate labelling.

Example 4.1. Let $UAF = (\{a, b, c, d\}, \{(d, c), (c, b), (b, a)\})$ and $AF = (\{a, b\}, \{(b, a)\})$. Then AF has the complete labelling $\{\, \mathtt{out}(a),\ \mathtt{in}(b) \,\}$. The program $\varepsilon\Pi^C_{AF}$ consists of rules:

$$\begin{aligned}
&\mathtt{in}(a) \leftarrow \mathtt{out}(b), \quad \mathtt{in}(b) \leftarrow \mathtt{out}(c), \quad \mathtt{in}(c) \leftarrow \mathtt{out}(d),\\
&\mathtt{out}(a) \leftarrow \mathtt{in}(b), \quad \mathtt{out}(b) \leftarrow \mathtt{in}(c), \quad \mathtt{out}(c) \leftarrow \mathtt{in}(d),\\
&\leftarrow \mathtt{in}(a),\ \mathbf{not}\,\mathtt{out}(b), \quad \leftarrow \mathtt{in}(b),\ \mathbf{not}\,\mathtt{out}(c), \quad \leftarrow \mathtt{in}(c),\ \mathbf{not}\,\mathtt{out}(d),\\
&\leftarrow \mathtt{out}(a),\ \mathbf{not}\,\mathtt{in}(b), \quad \leftarrow \mathtt{out}(b),\ \mathbf{not}\,\mathtt{in}(c), \quad \leftarrow \mathtt{out}(c),\ \mathbf{not}\,\mathtt{in}(d),\\
&\mathtt{in}(x) \vee \mathtt{out}(x) \vee \mathtt{und}(x) \leftarrow \quad (\text{where } x \in \{a, b, c, d\}),\\
&\leftarrow \mathtt{in}(x),\ \mathtt{out}(x), \quad \leftarrow \mathtt{in}(x),\ \mathtt{und}(x), \quad \leftarrow \mathtt{out}(x),\ \mathtt{und}(x) \ \ (\text{where } x \in \{a, b, c, d\}).
\end{aligned}$$



Given the enforcement set $E = \{\mathtt{in}(a)\}$, the program $\varepsilon\Pi^C_{AF} \cup \{\, \leftarrow \mathbf{not}\ \mathtt{in}(a) \,\}$ has the stable model $\{\, \mathtt{in}(a),\ \mathtt{out}(b),\ \mathtt{in}(c),\ \mathtt{out}(d) \,\}$. Then AF satisfies the enforcement E. That is, to enforce $\mathtt{in}(a)$,

AF is modified by introducing the new argument c and the attack relation (c, b). On the other hand, let $AF' = (\{a, b, d\}, \{(b, a)\})$. Then the fact $\mathtt{in}(d) \leftarrow$ and the constraint $\leftarrow \mathtt{out}(d)$ is in $\varepsilon\Pi^C_{AF'}$ and $\varepsilon\Pi^C_{AF'} \cup \{ \leftarrow \mathbf{not}\, \mathtt{in}(a) \}$ 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 $\{ \mathtt{out}(a), \mathtt{in}(b), \mathtt{out}(c), \mathtt{in}(d) \}$.

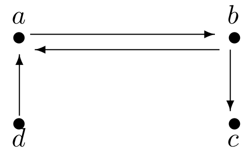
4.3. Agreement

Argumentation frameworks are used in negotiation [1] and debate [16]. 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 4.3. (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 set S of labelled arguments under a complete labelling and AF_2 has a set T of labelled arguments under a complete labelling 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 4.2. 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 sets of labelled arguments under complete labelling: $\{ \mathtt{in}(a), \mathtt{out}(b), \mathtt{in}(c) \}$, $\{ \mathtt{out}(a), \mathtt{in}(b), \mathtt{out}(c) \}$, $\{ \mathtt{und}(a), \mathtt{und}(b), \mathtt{und}(c) \}$; while AF_2 has the set of labelled arguments under 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 associated 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 an answer set S. In this case, AF_1 and AF_2 agree on each argument x such that* $\mathtt{agree}(x) \in S$.

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

5. Discussion

5.1. Reduction to Normal Programs

In Section 3 we introduce four different transformations from AF to LP. 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 three constraints (7) with three rules:

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

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 two rules:

$$\mathtt{in}(x) \leftarrow \mathbf{not}\,\mathtt{out}(x) \quad \text{and} \quad \mathtt{out}(x) \leftarrow \mathbf{not}\,\mathtt{in}(x).$$

Thus, Π^C_{AF}, Π^S_{AF} and Π^G_{AF} can be represented by normal programs. On the other hand, the program Π^P_{AF} *cannot* be transformed to a semantically equivalent normal program in polynomial time. This is because in Π^P_{AF} two atoms $\mathtt{in}(x)$ and $\mathtt{out}(x)$ may hold at the same time. In this case, $\mathtt{UND}(x)$ holds by definition. Then the above mentioned replacement in Π^S_{AF} is not applied to Π^P_{AF}. 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 [6, 7], 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.

5.2. Related Work

Connections between argumentation frameworks and logic programming have been investigated by several researchers. Dung [5] 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 $AGU = \{\, attack(x, y) \leftarrow \mid (x, y) \in att \,\}$ and $APU = \{\, defeat(x) \leftarrow attack(y, x), acc(y),$ $acc(x) \leftarrow \mathbf{not}\, defeat(x) \,\}$ where $acc(x)$ stands for "an argument x is acceptable" and $defeat(x)$ for "an argument x is defeated". For each extension E of AF, put $m(E) = AGU \cup \{\, acc(x) \mid x \in E\} \cup \{\, defeat(y) \mid y$ is attacked by some $x \in E \,\}$. He then shows the following results: (i) E is a stable extension of AF iff $m(E)$ is a stable model of P_{AF}; (ii) E is a grounded extension of AF iff $m(E) \cup \{\, \mathbf{not}\; defeat(a) \mid a \in E \,\}$ is the well-founded model of P_{AF}; (iii) The well-founded model and Fitting's model of P_{AF} coincide. Our representation of AF 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 a grounded extension of AF. This is different from our encoding in which arguments and their attack relations are represented as object-level rules in a program. Secondly, in Dung's encoding different semantics of AF correspond to different semantics of a logic program. By contrast, in our encoding, different semantics of AF are all characterized by stable models of a transformed program. Thirdly, Dung encodes extension-based semantics of AF, while we encode labelling-based semantics of AF. 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.

Egly *et al.* [9] introduce ASP encodings for AF. Their encoding is in line with Dung's meta-interpretative approach. Given $AF = (Ar, att)$, they define the program:

$$\begin{aligned}\pi_s \;=\;& \{\, arg(a) \mid a \in Ar \,\} \cup \{\, \mathit{defeat}(a,b) \;\mid\; (a,b) \in att \,\}\\ & \cup \{\, in(x) \leftarrow \mathbf{not}\ out(x), arg(x), \quad out(x) \leftarrow \mathbf{not}\ in(x), arg(x),\\ & \quad\; \leftarrow in(x), in(y), \mathit{defeat}(x,y)\}\\ & \cup \{\, \mathit{defeated}(x) \leftarrow in(y), \mathit{defeat}(y,x), \quad \leftarrow out(x), \mathbf{not}\ \mathit{defeated}(x)\,\}.\end{aligned}$$

Then they show that there is a one-to-one correspondence between stable extensions of AF and stable models of π_s. They provide similar encodings of complete extensions, grounded extensions, and preferred extensions. Different from Dung's encodings, Egly *et al.* characterize different semantics of AF in terms of stable models of a program. In the meta-interpretative approach, an instance of AF is given as an input to a single meta-logic program under a particular argumentation semantics. On the other hand, an ASP encodings becomes complicated for semantics such as the preferred semantics. The complication comes from the fact that a program has to encode tests for checking subset-maximality of admissible sets. To ease the problem, Dvořák *et al.* [8] use a built-in function for computing subset minimization.

Wakaki *et al.* [19] also introduce a meta-interpretative computation of AF semantics in ASP. They represent different labelling-based semantics of AF (complete, stable, grounded, preferred and semi-stable) by answer sets of a transformed program. For instance, complete labellings of $AF = (Ar, att)$ is computed by answer sets of the following program:

$$\begin{aligned}\pi_c \;=\;& \{\, arg(a) \mid a \in Ar \,\} \cup \{\, att(a,b) \;\mid\; (a,b) \in att \,\}\\ & \cup \{\, in(x) \leftarrow arg(x), \mathbf{not}\, ng(x), \quad ng(x) \leftarrow in(y), att(y,x),\\ & \quad ng(x) \leftarrow undec(y), att(y,x), \quad out(x) \leftarrow in(y), att(y,x),\\ & \quad undec(x) \leftarrow arg(x), \mathbf{not}\, in(x), \mathbf{not}\, out(x)\,\}.\end{aligned}$$

Stable labellings are computed by introducing the constraint $\leftarrow undec(x)$ to π_c. These transformations are similar to our Π^C_{AF} and Π^S_{AF}. To compute grounded/preferred labellings of AF, on the other hand, they first compute answer sets of π_c, i.e. computing every complete labelling as candidates, then they check whether those candidates satisfy the condition of minimal/maximal `in` labellings for grounded/preferred semantics. To this end, they introduce a meta-logic program to which answer sets of π_c are given as an input. As such, their transformation is not done in polynomial time for grounded and preferred semantics.

Nieves *et al.* [15] provide an encoding of preferred extensions of AF into a logic program. Using the atom $d(x)$ meaning "an argument x is defeated", they characterize the preferred semantics of AF in terms of stable models of positive (disjunctive) program. Given $AF = (Ar, att)$, they define its associated program as $\Gamma_{AF} = \bigcup_{a \in Ar} \Gamma(a)$ where

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

Informally speaking, the first part says that an argument a is defeated when any one of its adversaries is not defeated (i.e., $d(a) \vee d(b)$ is read as $d(a) \leftarrow \neg d(b)$). The second part says that an argument a is defeated when all the arguments that defend a are defeated. Then they show that there is a one-to-one correspondence between preferred extensions of AF and stable models of Γ_{AF}. Different from our transformation, $\Gamma(a)$ considers not only attackers but also defenders (i.e., the argument c defends a in the second part). Nieves $et\,al.$'s transformation is simple in the sense that it uses only one predicate d. On the other hand, they only provide encodings of preferred extensions and it is unclear whether similar encodings are possible for other semantics.

Wu $et\,al.$ [20] introduce a translation from AF to 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} [20]. 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} [4]. Wu $et\,al.$'s result 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 AF to different semantics of logic programs. By contrast, we characterize different semantics of AF by a single semantics—2-valued stable model semantics of logic programs. We do not study encoding semi-stable labelling in this paper.

6. Conclusion

We introduced methods of representing argumentation frameworks in terms of logic programs. The proposed transformations are simple and encode different AF semantics by stable models of LP in a uniform manner. This enables one to use existing answer set solvers for computing argumentation semantics and solving various problems of AF. Moreover, several techniques developed in LP 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, optimization (or partial evaluation) of AFs is viewed as optimization of AF-programs, update of AFs is realized by updates of AF-programs, etc. Once optimization or update is performed on an AF-program, the new AF-program is easily converted to a corresponding argumentation framework. In this way, the result of this study implicates 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 will argue possibilities of importing LP techniques into AF and investigate characterizing other semantics (such as semi-stable models) of AF in ASP.

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] Baumann, R. and Brewka, G.: Expanding argumentation frameworks: enforcing and monotonicity results. In: *Proc. 3rd COMMA, Frontiers in AI and Applications* 216, IOS Press, 2010, 75–86.

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

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] Gelfond, M. and Lifschitz, V.: The stable model semantics for logic programming. In: *Proc. 5th International Conference and Symposium on Logic Programming*, MIT Press, 1988, 1070–1080.

[11] Gelfond, M. and Lifschitz, V.: Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9(3/4), 1991, 365–385.

[12] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F.: The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic* 7(3), 2006, 1–57.

[13] Marek, V. W. and Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: *The Logic Programming Paradigm – A 25 Year Perspective* (K. R. Apt, V. M. Marek, M. Truszczynski, and D. S. Warren, Eds.), Springer, 1999, 375–398.

[14] 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.

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

[16] Sakama, C.: Dishonest arguments in debate games, In: *Proc. 4th International Conference on Computational Models of Argument. Frontiers in AI and Applications* 245, IOS Press, 2012, 177–184.

[17] Simons, P., Niemelä, I. and Soininen, T.: Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1/2), 2002, 181–234.

[18] 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.

[19] 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.

[20] 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.