On Strong Equivalence Notions in Logic Programming and Abstract Argumentation

Giovanni Buraglio[1,*], Wolfgang Dvořák[1] and Stefan Woltran[1]

[1]TU Wien, Institute of Logic and Computation

Abstract

Strong equivalence between knowledge bases ensures the possibility of replacing one with the other without affecting reasoning outcomes, in any given context. This makes it a crucial property in nonmonotonic formalisms. In particular, the fields of logic programming and abstract argumentation provide primary examples in which this property has been subject to vast investigations. However, while (classes of) logic programs and abstract argumentation frameworks are known to be semantically equivalent in static settings, this alignment breaks in dynamic contexts due to differing notions of update. As a result, strong equivalence does not always carry over from one formalism to the other. In this paper, we carefully investigate this discrepancy and introduce a new notion of strong equivalence for logic programs. Our approach preserves strong equivalence under translation between certain classes of logic programs and both Dung-style and claim-augmented argumentation frameworks, thus restoring compatibility across these formalisms.

Keywords

Strong Equivalence, Logic Programming, Abstract Argumentation

1. Introduction

In the field of Knowledge Representation and Reasoning, the concept of equivalence between knowledge bases has been the subject of extensive studies. A primary motivation behind this research is the potential to exploit the equivalence between two knowledge bases to achieve a compact representation of the same information. Furthermore, the possibility to substitute a specific component of a given knowledge base $\mathcal{K}$ with an equivalent yet simplified alternative has been shown to ease the computational cost of reasoning over $\mathcal{K}$. While this advantageous behavior can be taken for granted in monotonic formalisms (e.g. classical logic), it is usually not the case in non-monotonic ones. For this, the notion of strong equivalence has been introduced for non-monotonic formalisms, to capture the idea of equivalence in a dynamic environment, under any possible update [1, 2, 3, 4, 5, 6].

In this paper, we consider two families of non-monotonic formalisms, namely logic programming and abstract argumentation. Logic programming is a declarative programming paradigm where a reasoning problem is specified by means of a so-called logic program (LP). This consists of a set of inference rules made of atoms, possibly preceded by a negation-as-failure operator. Negative literals are assumed to be true as long as their corresponding positive atom cannot be proven to hold. Abstract argumentation is a sub-field of symbolic Artificial Intelligence [7, 8] that offers formal approaches to represent and reason over situations where conflicting information is present. An argumentative scenario is specified by means of an abstract argumentation framework, which is a directed graph where nodes represent arguments and edges their relation. The starting point in the field is the seminal work of Dung [9], where abstract argumentation frameworks contain only a relation of attack among arguments. We will refer to these as Dung-style AFs.

23rd International Workshop on Nonmonotonic Reasoning, November 11–13, 2025, Melbourne, Australia
*Corresponding author.
giovanni.buraglio@tuwien.ac.at (G. Buraglio); wolfgang.dvorak@tuwien.ac.at (W. Dvořák); stefan.woltran@tuwien.ac.at (S. Woltran)
https://informatics.tuwien.ac.at/people/giovanni-buraglio (G. Buraglio); https://dbai.tuwien.ac.at/staff/dvorak/ (W. Dvořák); https://www.dbai.tuwien.ac.at/staff/woltran/ (S. Woltran)
0009-0004-9592-4739 (G. Buraglio); 0000-0002-2269-8193 (W. Dvořák); 0000-0003-1594-8972 (S. Woltran)

Several semantics have been proposed for both of these formalisms, with the common purpose of extracting solutions for the given program or argumentation framework. These are respectively sets of atoms that satisfy each rule in the program (called *answer-sets*), and sets of arguments that are able to defend themselves against possible counter-arguments (called *extensions*). For the purpose of this work, we restrict our attention to the stable model semantics for LPs [10] and the stable semantics for argumentation frameworks [9].

Previous work has shown a one-to-one mapping between Dung-style AFs and (a resticted class of) LPs under the stable model semantics [9, 11]. In particular, any problem can be either specified via an program P or an abstract argumentation framework F in such a way that the answer sets of P are identical to the stable extensions of F. Moreover, by taking into account a wider class of LPs it is possible to capture more expressive types of abstract argumentation frameworks than Dung-style AFs, such as claim-augmented argumentation frameworks [12]. Thus, equivalent LPs correspond to equivalent AFs.

However, such a semantic correspondence does not necessarily carry over to dynamic contexts, where strong equivalence is required. It is possible for two logic programs that are strongly equivalent to induce argumentation frameworks that do not share this property, due to the incongruous notions of update (or expansion) in the two realms. To acknowledge such mismatch, consider the following example:

Example 1. *Two people X and Y are suspects of a murder for which they do not have an alibi. However, by means of forensic analysis it has been established that the crime has been committed by one person alone. While questioning possible witnesses, the detective learns from witness Z that X and Y were in a bar far away from the crime scene at the time of the murder. Thus, he establishes that the case cannot be solved. We model the detective's knowledge via an LP and an AF as follows:*

F
$$\begin{tikzcd} x \arrow[r, bend left] & y \arrow[l, bend left] & a \arrow[l] \arrow[ll, bend right] \end{tikzcd}$$

$$P = \{x \leftarrow \mathtt{not}\, y, \mathtt{not}\, a.,\ y \leftarrow \mathtt{not}\, x, \mathtt{not}\, a.,\ a \leftarrow .\}$$

where a means "has an alibi" and x (resp. y) means "X (resp. Y) is the murder". After some time, however, a second witness shows up and testifies that Z was drunk the entire evening on the same night, thereby falsifying the alibi of the two suspects. This information update is captured via:

F'
$$\begin{tikzcd} a & d \arrow[l] \end{tikzcd}$$

$$P' = \{a \leftarrow \mathtt{not}\, d.,\ d \leftarrow .\}$$

where d means "Z was drunk".

In accordance with equivalence results between logic programs and argumentation frameworks, the two ways of modeling our knowledge base are consistent with each other when taken individually. However, this does not happen when they are combined. Incorporating the second pair of knowledge bases (F' and P') into the first one (resp. F and P) yields different result: in the case of F and F', their union returns the expected result in the form of two possible extensions $\{d, x\}$ and $\{d, y\}$. Indeed, as far as the detective knows, the witness Z was drunk and this is compatible with either X or Y being the murder. On the other hand, the union of the two logic programs P and P' yields an unexpected prediction: the alibi '$a \leftarrow$' is not overwritten by Z being drunk '$d \leftarrow$', leaving $\{a, d\}$ as the only possible solution.

The discrepancy illustrated above reveals that updating an existing logic program by simply adding rules may produce unexpected outcomes. Facts (e.g. '$a \leftarrow$' in P) cannot be overwritten by incoming information, while their corresponding arguments (e.g. $a \in F$) can. This behavior of logic programs is fundamentally in contrast with the way in which non-monotonicity is encoded in abstract argumentation: an argument can always be attacked by new ones.

In this work, we carefully analyze the relationship between logic programming and abstract argumentation, with a particular interest in dynamic contexts. As a first step, we recall and extend equivalence results for classes of logic programs and abstract argumentation frameworks. Subsequently, motivated by the mismatch above, we introduce a novel notion of update for a restricted class of LPs, called Rule Refinement, that resolves the issue by mimicking precisely the existing notion for Dung-style AFs. We further extend Rule Refinement to the wider class of atomic logic programs (where no positive literal occurs in the body). Within this class, Rule Refinement in LPs captures strong equivalence in claim-augmented AFs.

2. Preliminaries

Logic Programming We consider normal logic programs with negation-as-failure `not`. Such programs consist of rules r of the form '$n : c \leftarrow a_1, \dots, a_k, \texttt{not}\, b_1, \dots, \texttt{not}\, b_m$.' read as '$c$ if a_1 and ... and a_k and $\texttt{not}\, b_1$ and ... and $\texttt{not}\, b_m$'. Here, $n \in \mathbb{N}$ is the *identifier* of the rule r in the program; we refer to with $id(r) = n$. Further, a_i, b_i and c are ordinary atoms; $\mathcal{L}(P)$ is the set of all atoms occurring in P. The atoms a_i are called positive atoms and the atoms b_j are called negated atoms, respectively denoted by $pos(r) = \{a_1, \dots, a_k\}$ and $neg(r) = \{b_1, \dots, b_m\}$. We use $head(r) = c$ and $body(r) = \{a_1, \dots, a_k, \texttt{not}\, b_1, \dots, \texttt{not}\, b_m\}$ for the head and body of r. With a slight abuse of notation, we extend previous predicates pos, neg, $head$, $body$ to sets of rules, e.g. $head(R) = \{head(r) \mid r \in R\}$. Given a set of atoms S, we write $\texttt{not}\, S = \{\texttt{not}\, b \mid b \in S\}$. The indices k, m can be equal to zero (that is, rules can have an empty body or only positive or negative literals in the body). A rule r is called: *constraint* if $head(r) = \emptyset$; *fact* if $k = m = 0$; *atomic* if $k = 0$. A logic program P is *atomic* if all the rules in P are atomic.

The semantics of normal logic programs is defined in terms of answer sets, also called stable models. Whenever a program consists of positive atoms only, its (unique) stable model consists of the minimal set of atoms closed under the rules. If not, the procedure to extract answer sets from a program is performed in two steps: first a set of atoms is guessed as a candidate answer set; then, the program is modified accordingly, by propagating information relative to the candidate set. The result of this modification is a program with no negated atoms called *reduct*. Formally:

Definition 1. *Let P be a normal logic program and $S \subseteq \mathcal{L}(P)$ a set of atoms. The reduct of P w.r.t. S is the negation-free program P^S obtained from P by: (i) deleting all rules $r \in P$ with $\texttt{not}\, b_j$ in the body for some $b_j \in S$, (ii) deleting all negated atoms from the remaining rules. S is an answer set of P iff S is the minimal model of P^S. The collection of answer sets of a program P is $AS(P)$.*

Throughout the entire paper, we will focus on the class of atomic LPs. Whenever we write program we mean an atomic one. Moreover, we will restrict our attention to a sub-classes of atomic programs. In particular, we call *strict* (resp. *h-unique*) a logic program P where each atom $p \in \mathcal{L}(P)$ occurs at least (resp. at most) once in the head of a rule.

Abstract Argumentation We fix a infinite background set of arguments $\mathcal{U}$. A strict argumentation framework (strict AF[1]) is a directed graph $F = (A, R)$ where $A \subseteq \mathcal{U}$ is a finite set of arguments and $R \subseteq A \times A$ an attack relation between them. The union of any two strict AFs $F = (A, R)$ and $G = (A', R')$ is defined as $F \cup G = (A \cup A', R \cup R')$. For two arguments $a, b \in A$ we say that a attacks b if $(a, b) \in R$. Moreover, a set of arguments $E \subseteq A$ attacks b if $(a, b) \in R$ for some $a \in E$. Analogously, a attacks E if $(a, b) \in R$ for some $b \in E$ and for a set $E' \subseteq A$ we say that E' attacks E if E' attacks some $b \in E$. We use $E^+_R = \{a \in A \mid E \text{ attacks } a\}$ and $E^-_R = \{a \in A \mid a \text{ attacks } E\}$ to denote the set of arguments respectively attacked by and attacking E. Further, the range of E (with respect to R), denoted $E^\oplus_R$, is the set $E \cup E^+_R$. For a singleton $\{a\}$ we use a^+_R, a^-_R and $a^\oplus_R$.

E is conflict-free in F ($E \in cf(F)$) iff for no $a, b \in E$, $(a, b) \in R$. E defends an argument a if E attacks every argument attacking a. Based on these two notions, [9] introduced different argumentation semantics. Formally, these are functions $\sigma : F \mapsto \sigma(F) \subseteq 2^A$. This means that, for any given $F = (A, R)$, a semantics returns a set of subsets of A. These subsets are called σ-extensions. Here we consider only *stable* semantics (*stb* for short):

Definition 2. *Let $F = (A, R)$ be an AF and $E \in cf(F)$ a conflict-free set in F. We say that E is a stable extension of F ($E \in stb(F)$) iff $E^\oplus_R = A$.*

[1]We use this terminology to refer to the standard definition of Dung AFs, to emphasize the fact that the attack relation is restricted to arguments in A. In Section 3, we will relax this requirement.

In recent years, more expressive abstract formalisms have been proposed that extend Dung-style AFs. Among these, *claim-augmented argumentation frameworks* (or CAFs) add claims to the abstract representation [12, 13].

Definition 3. *Let $\mathcal{C}$ be a set (or universe) of claims. A claim-augmented argumentation framework (CAF) is a triple $\mathcal{F} = (A, R, \gamma)$ such that $F = (A, R)$ is a strict AF and $\gamma : A \mapsto \mathcal{C}$ is a function that assigns claims to arguments.*

For a set of arguments E, we use $\gamma(E) = \{\gamma(a) \mid a \in E\}$ to denote the associated set of claims. The main advantage of CAFs lies in the fact that they allow to represent situations where different arguments have the same claim. In this paper we focus on a particular type of CAFs, called *well-formed CAFs*, for which arguments with same claim attack the same arguments. A given CAF $\mathcal{F} = (A, R, \gamma)$ is well formed iff for any two arguments a, b, $\gamma(a) = \gamma(b)$ implies $a^+_R = b^+_R$. Stable semantics for a well-formed CAF $\mathcal{F} = (F, \gamma)$ can be defined as taking the claim-sets $\gamma(E)$ inherited from every stable extension E.

Definition 4. *Let $\mathcal{F} = (F, \gamma)$ be a well-formed CAF. We call a set S a stable (claim-)extension of $\mathcal{F}$ ($S \in stb_i(\mathcal{F})$) iff there is an $E \in stb(F)$ such that $S = \gamma(E)$.*

Strong Equivalence Strong equivalence notions for logic programs and argumentation frameworks have been presented to capture equivalence under any possible updates. In the case of LPs, updates consist of expanding the original program with a set of rules. Two LPs P and Q are said to be equivalent, denoted $P \equiv Q$ whenever $AS(P) = AS(Q)$. Further, P and Q are strongly equivalent, denoted $P \equiv_s Q$, iff, for any LP R, the programs $P \cup R$ and $Q \cup R$ are equivalent, i.e. $P \cup R \equiv Q \cup R$ [1]. Here, we consider specific classes of LPs and adjust the relative notion of strong equivalence accordingly.

Definition 5. *Given a class Π of LPs, P and Q in Π, we say that P and Q are strongly equivalent in Π, written $P \equiv^{\Pi}_s Q$, iff for every program R: either (1) $P \cup R \notin \Pi$ and $Q \cup R \notin \Pi$ or (2) $P \cup R \equiv Q \cup R$.*

More recently, the notion of strong equivalence has been studied for other non-monotonic formalisms, in the field of abstract argumentation [6]. Similarly to logic programs, two strict AFs F and G are equivalent under stable semantics, if $stb(F) = stb(G)$. Strong equivalence, denoted by $F \equiv_s G$, is satisfied whenever $stb(F \cup H) = stb(G \cup H)$ for any strict AF H. In this setting, strong equivalence admits a syntactic characterization in terms of so-called (semantics-dependent) kernels, obtained by syntactical modifications of the given frameworks. For stable semantics, such modification consists in the removal of out-going attacks from self-attacking arguments.

Definition 6. *Let $F = (A, R)$ be an strict AF. The stable kernel of F is $F^{SK} = (A, R^{SK})$ with $R^{SK} = R \setminus \{(a, b) \mid a \neq b, (a, a) \in R\}$.*

The definition of kernels is a key step in obtaining a characterization of strong equivalence.

Proposition 1 (Oikarinen and Woltran [6])**.** *For any two strict AFs F and G, $F \equiv_s G$ iff $F^{SK} = G^{SK}$.*

The idea of strong equivalence has been investigated for CAFs as well. In the literature only expansions among *compatible* CAFs are considered. Informally, two CAFs are compatible if and only if the arguments they share have the same claim. Moreover, AF kernels characterize strong equivalence for CAFs as well.

Proposition 2 (Baumann et al. [14])**.** *Let $\mathcal{F} = (F, \gamma)$ and $\mathcal{G} = (G, \gamma')$ be two compatible well-formed CAFs. Then $\mathcal{F} \equiv^{stb_i}_s \mathcal{G}$ iff $F \equiv^{stb}_s G$.*

This kernel characterization applies when $\mathcal{F}$ and $\mathcal{G}$ are well-formed, for any common update $\mathcal{H}$ (possibly not well-formed). In Section 5.2 we will follow a different approach, since CAFs translated from logic programs are always guaranteed to be well-formed and the requirement of compatibility for expansions is not natural for logic programs.

3. From Abstract Argumentation to Logic Programming and Back

In this section we introduce translations between increasingly larger classes of logic programs and increasingly more expressive abstract argumentation frameworks. In particular, we recall the fact that strict h-unique atomic programs correspond to strict AFs [11]. Further, we show that by relaxing the strictness requirement for LPs we end up in the broader class of (possibly non-strict) AFs. Finally, we consider the full class of atomic LPs and their correspondence to well-formed CAFs.

3.1. Logic Programs and Dung-style AFs

We begin with strict h-unique atomic programs and strict AFs. Any strict AF can be transformed into an logic program by generating a rule r for every argument such that the head of r is the argument's name and the (negative) body contains the names of each attacker.

Definition 7 (Caminada et al. [11]). *Let $F = (A, R)$ be an argumentation framework with $A = \{a_1, \ldots, a_n\}$. The corresponding logic program of F is:*

$$P_F = \{i : a_i \leftarrow \texttt{not}\, b_1, \ldots, \texttt{not}\, b_k. \mid a_i \in A \textit{ and } \{b_1, \ldots, b_k\} = a_i^-\}$$

Notice that by definition P_F is atomic, strict and h-unique. Consider the following example:

Example 2. *Consider the following strict AF $F = (A, R)$ depicted below with $A = \{a, b, c\}$ and $R = \{(a, a), (a, b), (b, a), (b, c)\}$ and its corresponding LP P_F:*

$$\begin{tikzcd} F & \boxed{a} \arrow[loop left] \arrow[r, bend left] & \boxed{b} \arrow[l, bend left] \arrow[r] & \boxed{c} \end{tikzcd}$$

$$P_F = \{1 : a \leftarrow \texttt{not}\, a, \texttt{not}\, b.,\ 2 : b \leftarrow \texttt{not}\, a.,\ 3 : c \leftarrow \texttt{not}\, b.\}$$

For the other direction, the transformation only takes strict h-unique programs as input. For each rule, the head atom is associated to an argument with the same name and each negated atom (called *vulnerability*) in the body is associated with an attacker of the argument corresponding to the rule-head.

Definition 8 (Caminada et al. [11]). *Let P be a strict h-unique program. The corresponding strict argumentation framework $F_P = (A_P, R_P)$ is defined by:*

- $A_P = \{a \mid a \in \mathit{head}(r),\ r \in P\}$,
- $R_P = \{(a, b) \mid b \in \mathit{head}(r),\ a \in \mathit{neg}(r),\ r \in P\}$.

It is easy to see that under translation, the program in Example 2 generates exactly the strict AF F. Indeed, the presented translation is a bijection.

Proposition 3 (Caminada et al. [11]). *For a strict AF F and its corresponding LP P_F, $F = F_{P_F}$.*

Moreover, the transformation preserves equivalence in both directions: the answer sets for a (strict h-unique) program correspond exactly to the stable extensions of its transformed strict AF.

Proposition 4 (Caminada et al. [11]). *For any strict AF F and h-unique atomic program P, $\mathit{stb}(\mathcal{F}) = AS(P_{\mathcal{F}})$ and $AS(P) = \mathit{stb}(\mathcal{F}_P)$.*

Given a certain program P, the attacks that are generated in F_P depends on two factors: the vulnerabilities and head-atoms. Caminada et al. [11] considered equivalence under transformation in a static setting, where strictness becomes a crucial aspect to enforce in order to define a one-to-one mapping. In a dynamic setting, on the other hand, the presence of vulnerabilities in P which do not occur as head-atoms has an impact on the semantic level: it opens for the possibility of new incoming attacks in F_P.

Example 3. *The two h-unique programs $P = \{a \leftarrow \texttt{not}\, b.\}$ and $P' = \{a \leftarrow .\}$ have the same unique answer set $\{a\}$. However, when they are simultaneously updated with $P'' = \{b \leftarrow .\}$, their answer sets are different: $AS(P \cup P'') = \{\{b\}\}$ and $AS(P' \cup P'') = \{\{a, b\}\}$.*

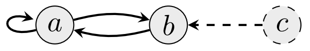
Motivated by these considerations, we relax the notion of strictness for logic program. That is, we consider programs with negative literals that never occur as head-atoms. Indeed, the previously shown translation would relate non-strict programs to strict AFs only at the cost of losing the one-to-one mapping: several non-strict LPs correspond to the same strict AF. In order to maintain an exact mapping, we also relax the definition of attack relation, allowing incoming attacks from elements in the universe $\mathcal{U}$ outside of A. We use *ungrounded attacks* to refer to these relations. Further, we simply call argumentation framework (AF) any framework with possibly ungrounded attacks.

Definition 9 (AF). *An AF $F = (A, R)$ is a directed graph where $A \subseteq \mathcal{U}$ is a (non-empty) set of arguments and $R \subseteq \mathcal{U} \times A$ is an conflict relation. We say that $R_{\downarrow A} = R \cap (A \times A)$ is the set of (proper) attacks, whereas $R \setminus R_{\downarrow A}$ is the set of ungrounded attacks.*

It is easy to see that the strictness notion of logic programs and argumentation frameworks are connected: every non-strict AF can be mapped to exactly one non-strict LP. Indeed, the isomorphism of Proposition 3 between strict h-unique atomic LPs and strict AFs can be lifted to possibly non-strict programs and AFs.

Remark 1. *For any AF F and corresponding logic program P_F, $F = F_{P_F}$. Conversely, for any h-unique atomic LP P and corresponding AF F, $P = P_{F_P}$.*

Example 4. *Let $F = (A, R)$ be the AF below and P_F its corresponding LP. The ungrounded attack $(c, b) \in R \setminus R_{\downarrow A}$ is represented by the vulnerability $c \in neg(P_F)$.*

F 

$P_F = \{1 : a \leftarrow \texttt{not}\, a, \texttt{not}\, b., \ 2 : b \leftarrow \texttt{not}\, a, \texttt{not}\, c.\}$

The notions of defense, conflict-freeness as well as stable semantics for AFs are defined over the attack relation $R_{\downarrow A}$ and are therefore identical to those introduced for strict AFs.

Definition 10. *Let $F = (A, R)$ be an AF. A set $E \subseteq A$ is conflict-free in F iff for no $a, b \in A$, $(a, b) \in R_{\downarrow A}$. Further, E is a stable extension of F iff $E^{\oplus}_{R_{\downarrow A}} = A$.*

Notice that the notion of stable-kernel trivially characterizes strong equivalence between AFs. This follows straightforwardly from the fact that we fixed $R \subseteq \mathcal{U} \times A$. Therefore, no ungrounded attack generates from a self-attacking argument. Moreover, ungrounded attacks have no influence on the evaluation of an AF.

Proposition 5. *For an AF $F = (A, R)$ and its strict correspondent $F_{\downarrow A} = (A, R_{\downarrow A})$, $stb(F) = stb(F_{\downarrow A})$.*

For h-unique programs we observe a similar behavior. We can identify a set $S \subseteq neg(P)$ of *ungrounded vulnerabilities* that make a program P non-strict. Formally, the set of ungrounded vulnerabilities for P is $UV(P) = \{\texttt{not}\, b \mid b \in neg(P) \setminus head(P)\}$. By removing such vulnerabilities, we obtain the strict program $P_{\downarrow head}$, defined as $P_{\downarrow head} = \{id(r) : head(r) \leftarrow body(r) \setminus UV(P). \mid r \in P\}$. Notice that transforming a non-strict atomic program into the corresponding strict one yields equivalent answer-sets, and vice-versa. Ungrounded vulnerabilities have no semantic impact, similarly to ungrounded attacks. Thus, answer-sets are preserved when restricting to strict programs.

Proposition 6. *For an atomic program P and its strict correspondent $P_{\downarrow head}$, $AS(P) = AS(P_{\downarrow head})$.*

To show that semantic equivalence can be lifted to AFs and h-unique programs, we introduce the following lemma.

Lemma 1. *For any h-unique atomic logic program P, $F_{P_{\downarrow head}} = (F_P)_{\downarrow A_P}$. For any AF F, $P_{F_{\downarrow A}} = (P_F)_{\downarrow head}$.*

Proposition 7. *For any h-unique atomic program P and the corresponding AF F_P, it holds that $AS(P) = stb(F_P)$. Conversely, for any AF F and its corresponding program P_F, we have that $stb(F) = AS(P_F)$.*

Proof. From propositions and lemmas above, it can be easily constructed the following chain of equivalences: $AS(P) = AS(P_{\downarrow head}) = stb(F_{P_{\downarrow head}}) = stb((F_P)_{\downarrow A_P}) = stb(F_P)$. In a similar way, it is easily proven that $stb(F) = stb(F_{\downarrow A}) = AS(P_{F_{\downarrow A}}) = AS((P_F)_{\downarrow head}) = AS(P_F)$. □

3.2. Logic Programs and well-formed CAFs

We can now move one step forward and focus on the wider class of atomic LPs and their relation to well-formed CAFs. Within this class of frameworks, attacks are not arbitrary pairs of arguments, but depends on the attacker's claim. A translation has been introduced for *strict* atomic LPs from and to well-formed CAFs. In our setting, we are able to provide a slightly more general translation that considers possibly non-strict programs. To do so, we present a definition of well-formed CAFs more suitable for a dynamic setting.

Definition 11. *A well-formed CAF is a triple $\mathcal{F} = (A, R^{\mathcal{C}}, \gamma)$ where A is a finite set of arguments, $R^{\mathcal{C}} \subseteq \mathcal{C} \times A$ is a set of claim-attacks with $\mathcal{C}$ the universe of claims and $\gamma : A \mapsto \mathcal{C}$ the claim function.*

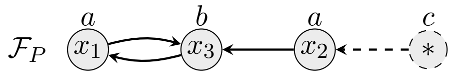
One can easily retrieve the original definition of CAF by fixing $R = \{(x_i, x_j) \mid x_i, x_j \in A, \gamma(x_i) = c \text{ and } (c, x_j) \in R^{\mathcal{C}}\}$. As a result, we can utilize the definition of stable semantics for well-formed CAFs by means of this translation. By appealing to the universe of claims $\mathcal{C}$, we have a formulation that encompasses claim attacks from claims which do not label any argument, i.e. ungrounded attacks. Any atomic LP exactly corresponds to some well-formed CAF by taking an argument for each rule, a set of claims for each atom occurring as the head of some rule and attacks defined from claims to arguments.

Definition 12. *Given an atomic program P, we obtain the well-formed CAF $\mathcal{F}_P = (A_P, R^{\mathcal{C}}_P, \gamma_P)$ via:*

- $A_P = \{x_i \mid id(r) = i, r \in P\}$,
- $R^{\mathcal{C}}_P = \{(c, x_i) \mid c \in neg(r), id(r) = i\}$,
- $\gamma_P(x_i) = head(r)$ *with* $id(r) = i$.

Example 5. *Consider the following atomic LP P below and its corresponding well-formed CAF. Notice that the claim-attack (c, x_2) is realized by an ungrounded attack.*

$$P = \{1 : a \leftarrow \texttt{not}\, b.,\ 2 : a \leftarrow \texttt{not}\, c.,\ 3 : b \leftarrow \texttt{not}\, a.\}$$



From a well-formed CAF, each argument x_i corresponds to a rule r with $id(r) = i$ and head $\gamma(x_i)$. Further, for each claim $c \in \mathcal{C}$ attacking the argument x_i, $\texttt{not}\, c$ appears in the body of the rule.

Definition 13. *Let $\mathcal{F} = (A, R^{\mathcal{C}}, \gamma)$ be a well-formed CAF with $A = \{x_1, \ldots, x_n\}$. The corresponding LP is $P_{\mathcal{F}} = \{i : \gamma(x_i) \leftarrow \texttt{not}\, b_1, \ldots \texttt{not}\, b_k. \mid x_i \in A, \{b_1, \ldots, b_k\} = \gamma(x_i^-)\}$.*

Indeed, the possibility of having different arguments with the same claim in $\mathcal{F}$ could make $P_{\mathcal{F}}$ a non h-unique program. As for AFs, also CAFs are isomorphic to atomic LPs.

Proposition 8 (König et al. [15]). *Let $\mathcal{F}$ be a CAF and $P_{\mathcal{F}}$ its corresponding LP. Then, $\mathcal{F} = \mathcal{F}_{P_{\mathcal{F}}}$.*

For well-formed CAFs and LPs, equivalence is preserved under translation in a static setting.

Proposition 9 (König et al. [15]). *For any well-formed CAF $\mathcal{F}$ and atomic program P, $stb(\mathcal{F}) = AS(P_{\mathcal{F}})$ and $AS(P) = stb(\mathcal{F}_P)$.*

In their original formulation, the propositions above consider only strict well-formed CAFs and atomic programs. However, they can be carefully adapted to our context based on the our relaxed notion of well-formed CAF.

3.3. Equivalence from Static to Dynamic Setting

Until now, we have considered semantic equivalence within a static setting and provided translations that preserve it. We point out that such a semantic correspondence does not carry over to dynamic scenarios, already within the smaller class of strict AFs. Two strongly equivalent LPs may have corresponding AFs that are not strongly equivalent.

Proposition 10. *For two LPs P and Q it holds that $P \equiv_s Q$ does not imply $F_P \equiv_s F_Q$.*

Example 6. *Take two logic programs as follows:*

$P = \{a \leftarrow \texttt{not}\, b, \texttt{not}\, c.,\ b \leftarrow \texttt{not}\, a, \texttt{not}\, c.,\ c \leftarrow .\}$ $\quad Q = \{a \leftarrow \texttt{not}\, c.,\ b \leftarrow \texttt{not}\, a, \texttt{not}\, c.,\ c \leftarrow .\}$

Since both P and Q contain the fact $c \leftarrow$ and $c \in neg(r)$ for every other rule in both programs, c occurs in any answer-set of $P \cup R$ and $Q \cup R$ for any R. Hence, they are strongly equivalent. The associated strict AFs are:

$$\begin{tikzcd} F_P & a \arrow[r, bend left] & b \arrow[l, bend left] & c \arrow[l] \arrow[ll, bend right] \end{tikzcd} \qquad \begin{tikzcd} F_Q & a \arrow[r] & b & c \arrow[l] \arrow[ll, bend right] \end{tikzcd}$$

Both F_Q and F_P have no self-attacking arguments. Hence, F_Q and F_P coincide to their own kernels. Thus, since they are different, they are not strongly equivalent.

4. Strong Equivalence under Rule Refinement

As previously shown, equivalence among LPs and AFs does not carry over to strong equivalence: due to the incongruous definitions of update in the two realms, their evaluation changes when moving from a static to a dynamic setting. To solve this mismatch we look at LPs through the lenses of abstract argumentation. From the LP perspective, adding attackers on the corresponding AF means adding vulnerabilities to the rule whose head identifies the attacked argument. We call this operation *rule refinement*, defined as follows.

Definition 14 (rule refinement)**.** *Let r and r' be two arbitrary LP rules. We use*

$$\texttt{refine}(r, r') := \{id(r) : head(r) \leftarrow body(r) \cup body(r').\}$$

to express that r is refined by means of r'.

In the remainder of the section, we introduce a novel notion of strong equivalence for the class of (h-unique) atomic programs based on the rule refinement operator.

4.1. Rule Refinement for h-unique LPs

We first consider the class of h-unique atomic programs. Within this class, talking about rules or their respective head-atoms is identical. Consequently, we can omit the explicit identifiers from rules and use the head-atoms instead. We can now introduce a novel notion of LP update, that we call Rule Refinement (RR, for brevity). Updating a program P with a rule r' via RR consists of two possible operations: in the case where $head(r') \notin head(P)$, the update is a simple set-union; otherwise, the rule-body of $r \in P$ for which $head(r) = head(r')$ is merged with that of r'.

Definition 15 ($\uplus$-update)**.** *Let P be an h-unique atomic program and r a rule in P. We define the $\uplus$-update (or simply update) of P by means of a new atomic rule r' as:*

$$P \uplus r' = \begin{cases} P \cup \{r'\} & \text{if } head(r') \notin head(P) \\ P \setminus \{r\} \cup \texttt{refine}(r, r') & \text{otherwise} \end{cases}$$

Example 7. *The $\uplus$-update of $P = \{a \leftarrow \texttt{not}\, b.,\ b \leftarrow \texttt{not}\, a.,\ c \leftarrow .\}$ via the rule $r = c \leftarrow \texttt{not}\, d.$ is: $P \uplus r = \{a \leftarrow \texttt{not}\, b.,\ b \leftarrow \texttt{not}\, a.,\ c \leftarrow \texttt{not}\, d.\}$.*

Towards the generalization of $\uplus$ between sets of rules, we notice that the consecutive applications of $\uplus$ preserve the result independently of the order in which they are executed.

Lemma 2. *For an atomic program P and two rules r_1 and r_2 it holds that $(P \uplus r_1) \uplus r_2 = (P \uplus r_2) \uplus r_1$.*

Updating an h-unique program P with another program $Q = \{r_1, \ldots, r_n\}$ results in $P \uplus Q = P \uplus r_1 \uplus \cdots \uplus r_n$. Moreover, $P \uplus Q$ is guaranteed to be atomic and h-unique.

Proposition 11. *Let P be an h-unique atomic program. For any atomic program Q, updating P with Q under Rule Refinement results in $P \uplus Q$ being a h-unique and atomic.*

We are now able to introduce RR strong equivalence in the class of h-unique atomic programs Ξ.

Definition 16. *Two h-unique atomic programs P and Q are strongly equivalent under Rule Refinement in Ξ, written $P \equiv_r^{\Xi} Q$, iff for any program R: either (1) $P \uplus R \notin \Xi$ and $Q \uplus R \notin \Xi$ or (2) $AS(P \uplus R) = AS(Q \uplus R)$.*

By the definition of RR-update, whenever $R \in \Xi$, condition (1) is never satisfied. Moreover, strongly equivalent programs under standard expansions are not guaranteed to be RR strong equivalent in Ξ.

Proposition 12. *For any two h-unique atomic programs P and Q:*

- *$P \equiv_s^{\Xi} Q$ does not imply $P \equiv_r^{\Xi} Q$;*
- *$P \equiv_s Q$ does not imply $P \equiv_r^{\Xi} Q$.*

Example 8. *Consider the programs P and Q as follows:*

$P = \{a \leftarrow \mathtt{not}\, b, \mathtt{not}\, c.,\ b \leftarrow \mathtt{not}\, a, \mathtt{not}\, c.,\ c \leftarrow .\}$ $\quad Q = \{a \leftarrow \mathtt{not}\, b, \mathtt{not}\, c.,\ b \leftarrow \mathtt{not}\, c.,\ c \leftarrow .\}$

Let R be an (atomic) program. Since $c \leftarrow$ is contained in both $P \cup R$ and $Q \cup R$, the atom c occurs in every answer-set of both programs. Thus, all the other rules in P and Q do not fire, making P and Q strongly equivalent, i.e. $P \equiv_s Q$ and $P \equiv_s^{\Xi} Q$. However, for the program $R' = \{c \leftarrow \mathtt{not}\, d., d \leftarrow .\}$, we derive: (1) $P \uplus R' \in \Xi$ and $Q \cup R' \in \Xi$ as well as (2) $AS(P \uplus R') \neq AS(Q \uplus R')$ since $\{a, d\}$ is an answer-set of the former, but not of the latter. Thus, $P \not\equiv_r^{\Xi} Q$.

4.2. Rule Refinement for LPs

In this section we consider a notion of strong equivalence under rule refinement for the whole class Λ of atomic LPs by dropping the requirement of h-uniqueness. We deal with (possibly non-strict) atomic programs, in which several rules with the same head might occur. Here, the information provided by rule identifiers becomes relevant when considering possible updates: identifiers allow to distinguish among updates that involve rules already contained in P or new ones. Indeed, updating a program with a new rule r' may result in the addition or refinement, irrespective of the head of r'. We then introduce a different notion of update.

Definition 17 ($\boxplus$-update)**.** *Let P be an atomic program and r a rule in P. We define the $\boxplus$-update (or simply update) of P by means of a new rule r' as:*

$$P \boxplus r' = \begin{cases} P \cup \{r'\} & \textit{if } id(r') \notin id(P) \\ P \setminus \{r\} \cup \mathtt{refine}(r, r') & \textit{otherwise} \end{cases}$$

Identifiers guide the update either towards the addition of the new rule or the refinement of an old rule with the same identifier.

Example 9. *The $\boxplus$-update of $P = \{1 : a \leftarrow \mathtt{not}\, b.,\ 2 : b \leftarrow \mathtt{not}\, a.,\ 3 : b \leftarrow .\}$ by means of the rule $r = 3 : a \leftarrow \mathtt{not}\, c.$ is: $P \boxplus r = \{1 : a \leftarrow \mathtt{not}\, b.,\ 2 : b \leftarrow \mathtt{not}\, a.,\ 3 : b \leftarrow \mathtt{not}\, c.\}$.*

As before, to generalize the definition of $\boxplus$-updates between sets of rules, we note that associativity is preserved.

Lemma 3. *For an atomic program P and two rules r_1 and r_2 it holds that $(P \boxplus r_1) \boxplus r_2 = (P \boxplus r_2) \boxplus r_1$.*

Moreover, $P \boxplus Q$ is atomic whenever P and Q are.

Proposition 13. *For any two atomic programs P and Q, the updated program $P \boxplus Q$ is atomic.*

We are now able to define strong equivalence between atomic programs under rule refinement.

Definition 18. *Two (atomic) programs P and Q are strongly equivalent under Rule Refinement in a class Π, written $P \equiv_{r+}^{\Pi} Q$, iff for any program R: either (1) $P \uplus R \notin \Pi$ and $Q \uplus R \notin \Pi$ or (2) $AS(P \uplus R) = AS(Q \uplus R)$.*

In the following we will consider $\Pi = \Lambda$. Then, in case P and Q are atomic, condition (1) is false iff $R \in \Lambda$, i.e., it is sufficient to check condition (2) for $R \in \Lambda$. For atomic programs, we inherit results from Proposition 12, i.e. $P \equiv_{s}^{\Lambda} Q$ does not imply $P \equiv_{r+}^{\Lambda} Q$ (see Example 8). Further this notion faithfully generalizes RR strong equivalence for h-unique programs.

Proposition 14. *For the class of h-unique atomic programs Ξ and $P, Q \in \Xi$, it holds that $P \equiv_{r+}^{\Xi} Q$ implies $P \equiv_{r}^{\Xi} Q$.*

5. Rule Refinement Captures Strong Equivalence in Abstract Argumentation

In the present section, we show that the notion of strong equivalence under Rule Refinement introduced for logic programs matches the one for abstract argumentation frameworks. We first consider h-unique LPs and AF strong equivalence. In this setting, a syntactic characterization of RR-strong equivalence can be reached via the notion of ASP kernel. Further, we see that our result generalizes to the full class of atomic programs and well-formed CAFs.

5.1. AFs and h-unique Logic Programs

Inspired by the syntactic characterization of strong equivalence for strict AFs [6], we introduce the notion of *kernel* for logic programs. Similarly to its AF counterpart, the ASP kernel of a program is obtained by deleting dispensable vulnerabilities from its rules. Every atom that is in conflict with itself, i.e. it occurs in the head as well as in the negative body of a rule, gets removed from every other rule-body. We call such rules loops and define $loop(P) = \{r \in P \mid head(r) \in neg(r)\}$ for an LP P.

Definition 19. *Let $loopH(Q) = \{\texttt{not}\, head(r) \mid r \in loop(Q)\}$. The kernel of a h-unique program P is:*

$$P^K = \{head(r) \leftarrow body(r) \setminus loopH(P \setminus \{r\}). \mid r \in P\}.$$

Example 10. *In what follows, we represent a program P and its kernel P^K.*

$$P = \{a \leftarrow \texttt{not}\, a, \texttt{not}\, b.,\ b \leftarrow \texttt{not}\, a, \texttt{not}\, c.,\ c \leftarrow \texttt{not}\, c, \texttt{not}\, d.,\ d \leftarrow \texttt{not}\, a, \texttt{not}\, c.\} \qquad P^K = \{a \leftarrow \texttt{not}\, a, \texttt{not}\, b.,\ b \leftarrow .,\ c \leftarrow \texttt{not}\, c, \texttt{not}\, d.,\ d \leftarrow .\}$$

As a sanity check, we show that our notions of LP and AF kernel carry over under transformation: the order in which we apply the translation and construct the kernel does not matter.

Proposition 15. *For a h-unique atomic program P, it holds that $(F_P)^{SK} = F_{P^K}$. Analogously for an AF F, it holds that $(P_F)^K = P_{F^{SK}}$.*

Combining two programs P and Q through Rule Refinement and transforming the resulting program into an AF yields the same framework as taking the AF union of the transformed programs.

Lemma 4. *For two h-unique atomic programs P and Q , it holds that $F_{P \uplus Q} = F_P \cup F_Q$. Vice-versa, for two AFs F and G, it holds that $P_{F \cup G} = P_F \uplus P_G$.*

As illustrated for AFs, we show that ASP kernels are semantically equivalent to the original instance.

Proposition 16. *For any h-unique atomic program P it holds that $AS(P) = AS(P^K)$.*

Further, two programs P and Q maintain the same kernel under any common update.

Proposition 17. *Let P and Q be two h-unique atomic programs with $P^K = Q^K$. Then for any h-unique atomic program R, it holds that $(P \uplus R)^K = (Q \uplus R)^K$.*

Finally, we can prove that RR strong equivalence for LPs characterizes strong equivalence for AFs.

Theorem 1. *Let P and Q be two h-unique atomic programs, then the following conditions are equivalent:*

1. *P and Q have the same kernel, i.e. $P_K = Q_K$.*
2. *The AFs corresponding to P and Q have the same stable-kernel, i.e. $(F_P)^{SK} = (F_Q)^{SK}$.*
3. *The AFs corresponding to P and Q are strongly equivalent for stable semantics, i.e. $F_P \equiv_s F_Q$.*
4. *P and Q are strongly equivalent under RR, i.e. $P \equiv_r^\Xi Q$.*

Proof. We initially show the equivalence from 1. to 4. and then the other way around. From $P_K = Q_K$ we derive $F_{P^K} = F_{Q^K}$ via Definition 8. Further, via Proposition 15 we obtain $(F_P)^{SK} = (F_Q)^{SK}$. This is equivalent to $F_P \equiv_s F_Q$ from the characterization of strong equivalence for AFs. From the definition of strong equivalence, we derive that for any AF H, *stb*$(F_P \cup H) =$ *stb*$(F_Q \cup H)$ and $AS(P_{F_P \cup H} = AS(P_{F_Q \cup H})$ by Proposition 4. Now, given Lemma 4, we can rewrite the previous equivalence as $AS(P_{F_P} \uplus P_H) = AS(P_{F_Q} \uplus P_H)$. Since the transformation between AFs and h-unique atomic programs is an isomorphism (Remark 1), it holds that $AS(P \uplus P_H) = AS(Q \uplus P_H)$. Therefore, for any program $H' = P_H$ associated to the AF H we get $AS(P \uplus H') = AS(Q \uplus H')$. Since H is an AF, P_H is guaranteed to be h-unique. Further, since P and Q are also h-unique by hypothesis, we derive $P \uplus H' \in \Xi$ and $Q \uplus H' \in \Xi$, concluding $P \equiv_r^\Xi Q$.

We now prove the other direction. By definition, $P \equiv_r Q$ entails $AS(P \uplus R) = AS(Q \uplus R)$ for any h-unique atomic program R. Thus under translation we get $F_{P \uplus R}$ and $F_{Q \uplus R}$ s.t. *stb*$(F_{P \uplus R}) =$ *stb*$(F_{Q \uplus R})$. Then, via Lemma 4, we rewrite the previous equivalence as *stb*$(F_P \cup F_R) =$ *stb*$(F_Q \cup F_R)$ for any program R. Let us call H the unique AF corresponding to each F_R, we write *stb*$(F_P \cup H) =$ *stb*$(F_Q \cup H)$ for any H. Thus, $F_P \equiv_s F_Q$, by definition of strong equivalence. From Proposition 1 the two frameworks have the same stable kernel: $(F_P)^{SK} = (F_Q)^{SK}$. We transform back to programs, obtaining $P_{(F_P)^{SK}} = P_{(F_Q)^{SK}}$ and $(P_{F_P})^K = (P_{F_Q})^K$ via Proposition 15. Finally, Remark 1 brings us to $P^K = Q^K$. □

In light of Propositions 5 and 6, the result above immediately applies to the smaller class of strict AFs.

Remark 2. *For two strict h-unique atomic programs P and Q, $P \equiv_r^\Xi Q$ if and only if $F_P \equiv_s F_Q$ where F_P and F_Q are strict AFs.*

5.2. Atomic Programs and Well-Formed CAFs

We first show that the ASP-kernel from Definition 19 is not helpful to characterize strong equivalence under Rule Refinement for atomic programs.

Example 11. *Take two programs P and Q with $Q = P^K$.*

$P = \{1 : a \leftarrow \texttt{not}\, b.,\ 2 : b \leftarrow .,\ 3 : b \leftarrow \texttt{not}\, b.\}$ $\qquad$ $Q = \{1 : a \leftarrow .,\ 2 : b \leftarrow .,\ 3 : b \leftarrow \texttt{not}\, b.\}$

It is easy to see that $AS(P) \neq AS(P^K)$. Hence, ASP-kernels do not characterize strong equivalence for atomic programs. Thus, we provide a direct characterization of RR strong equivalence for atomic programs. Later we will connect this characterization with strong equivalence for well-formed CAFs. We show that two programs are RR strongly equivalent if their rules' identifiers, heads and bodies pairwise coincide, with the possible exception of loop-rules. In fact, two programs are strongly equivalent under RR irrespective of whether the head-atoms of corresponding loop-rules coincide.

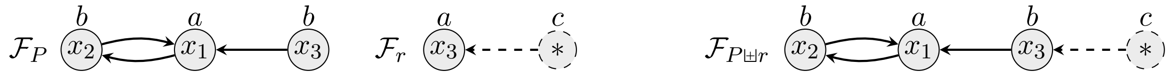
Lemma 5. *Let P be a program and $r \in loop(P)$ a loop-rule in P. Then, r gets removed when computing P^S for any $S \in AS(P)$.*

Theorem 2. *Two atomic programs P and Q are strongly equivalent under Rule Refinement, denoted $P \equiv_{r+}^\Lambda Q$, iff it holds that:*

1. *$id(r) = id(r')$ implies $body(r) = body(r')$ for all rules $r \in P$ and $r' \in Q$;*
2. *$id(r) = id(r')$ implies $head(r) = head(r')$ for all rules $r \notin loop(P)$ and $r' \notin loop(Q)$.*

As before, we analyze how this new notion of update is understood in terms of CAFs. On the one hand, simply adding a rule to a program P amounts to augmenting $\mathcal{F}_P$ with an argument and possibly new attacks. On the other hand, refining a rule corresponds to adding claim-attacks to $\mathcal{F}_P$.

Example 12. *Consider the CAFs $\mathcal{F}_P$, $\mathcal{F}_r$ and $\mathcal{F}_{P \uplus r}$ corresponding to P, r and $P \uplus r$ from Example 9:*



We next introduce our update operator for well-formed CAFs that will mimic $\uplus$-update.

Definition 20. *Given any two well-formed CAFs $\mathcal{F}_1 = (A_1, R_1^{\mathcal{C}}, \gamma_1)$ and $\mathcal{F}_2 = (A_2, R_2^{\mathcal{C}}, \gamma_2)$, we say that $\mathcal{F}_1 \cup \mathcal{F}_2 = (A_1 \cup A_2, R_1^{\mathcal{C}} \cup R_2^{\mathcal{C}}, \gamma_1 \vec{\times} \gamma_2)$ is the update of $\mathcal{F}_1$ via $\mathcal{F}_2$ where:*

$$\gamma_1 \vec{\times} \gamma_2(a) = \begin{cases} \gamma_1(a) & \textit{if } a \in A_1 \\ \gamma_2(a) & \textit{if } a \in A_2 \setminus A_1 \end{cases}$$

Updating an atomic program with an atomic rule might result in a violation of compatibility in the corresponding CAFs. For instance, in Example 12 the argument x_3 is labelled with two different claims in $\mathcal{F}_P$ and $\mathcal{F}_r$. For such incompatible expansions, we operate a choice between the claim functions, prioritizing γ_1. This ensures that whenever two CAFs are incompatible with respect to some arguments, we choose the original claim as it is done with the rule-head of any rule r that is refined via an $\uplus$-update. We can now define strong equivalence between well-formed CAFs.

Definition 21. *Given any two well-formed CAFs $\mathcal{F}$ and $\mathcal{G}$, we say that $\mathcal{F} \equiv_s \mathcal{G}$ if and only if $\sigma(\mathcal{F} \cup \mathcal{H}) = \sigma(\mathcal{G} \cup \mathcal{H})$ for any well-formed CAF $\mathcal{H}$.*

This notion differs from the original in two respects: (1) it does not restricts to compatible updates only and (2) it requires updates to be well-formed. This notion of strong equivalence for CAFs corresponds to strong equivalence under rule refinement for atomic LPs. To show this, we use of the following lemma.

Lemma 6. *Given any two atomic programs P and Q, it holds that $\mathcal{F}_{P \uplus Q} = \mathcal{F}_P \cup \mathcal{F}_Q$. Conversely, given two well-formed CAFs $\mathcal{F}$ and $\mathcal{H}$, it holds that $P_{\mathcal{F} \cup \mathcal{H}} = P_{\mathcal{F}} \uplus P_{\mathcal{H}}$.*

Theorem 3. *Let P and Q two atomic logic programs, then the following conditions are equivalent:*

- *The CAFs corresponding to P and Q are strongly equivalent for stable semantics, i.e. $\mathcal{F}_P \equiv_s \mathcal{F}_Q$.*
- *P and Q are strongly equivalent under Rule Refinement, i.e. $P \equiv_{r+}^{\Lambda} Q$.*

Proof. We prove the two statements from top to bottom and viceversa. Assume $\mathcal{F}_P \equiv_s \mathcal{F}_Q$. By definition, for any well-formed CAF $\mathcal{H}$, *stb*$(\mathcal{F}_P \cup \mathcal{H}) =$ *stb*$(\mathcal{F}_Q \cup \mathcal{H})$. From Proposition 9, we can rewrite the previous as $AS(P_{\mathcal{F}_P \cup \mathcal{H}}) = AS(P_{\mathcal{F}_Q \cup \mathcal{H}})$. Thanks to Lemma 6, we derive $AS(P_{\mathcal{F}_P} \uplus P_{\mathcal{H}}) = AS(P_{\mathcal{F}_Q} \uplus P_{\mathcal{H}})$. We transform this via the isomorphism in Proposition 8 into $AS(P \uplus P_{\mathcal{H}}) = AS(Q \uplus P_{\mathcal{H}})$. Thus, for any $H' = P_H$, we get $P \uplus H' = Q \uplus H'$. Since $\mathcal{H}$ is a well-formed CAF, P_H is guaranteed to be an atomic program. Further, since P and Q are also atomic by hypothesis, we derive $P \uplus H' \in \Lambda$ and $Q \uplus H' \in \Lambda$. Finally, we conclude $P \equiv_{r+}^{\Lambda} Q$. Assume now $P \equiv_{r+}^{\Lambda} Q$. Hence, either (i) $AS(P \uplus R) = AS(Q \uplus R)$ for any program R or (ii) $P \uplus R \notin \Lambda$ and $Q \uplus R \notin \Lambda$. If (ii) is the case, then R is not atomic. If (i) is the case, then we can translate the corresponding programs, deriving *stb*$(\mathcal{F}_{P \uplus R}) =$ *stb*$(\mathcal{F}_{Q \uplus R})$ by Proposition 9. Through Lemma 6, we then obtain *stb*$(\mathcal{F}_P \cup \mathcal{F}_R) =$ *stb*$(\mathcal{F}_Q \cup \mathcal{F}_R)$ for any well-formed CAF $\mathcal{F}_R$. We fix the well-formed CAF $\mathcal{H} = \mathcal{F}_R$ and rewrite the previous equivalence as *stb*$(\mathcal{F}_P \cup \mathcal{H}) =$ *stb*$(\mathcal{F}_Q \cup \mathcal{H})$, which by definition means $\mathcal{F}_P \equiv_s \mathcal{F}_Q$. $\square$

From Theorems 2 & 3 we obtain a characterization of strong equivalence on well-formed CAFs.

Corollary 1. *For two well-formed CAFs $\mathcal{F} = (A_{\mathcal{F}}, R_{\mathcal{F}}^{\mathcal{C}}, \gamma_{\mathcal{F}})$ and $\mathcal{G} = (A_{\mathcal{G}}, R_{\mathcal{G}}^{\mathcal{C}}, \gamma_{\mathcal{G}})$, we have $\mathcal{F} \equiv_s \mathcal{G}$ iff*

1. *$A_{\mathcal{F}} = A_{\mathcal{G}}$ and $R_{\mathcal{F}}^{\mathcal{C}} = R_{\mathcal{G}}^{\mathcal{C}}$, and*
2. *for each $x_i \in A$, either $\gamma_{\mathcal{F}}(x_i) = \gamma_{\mathcal{G}}(x_i)$ or $(\gamma_{\mathcal{F}}(x_i), x_i) \in R_{\mathcal{F}}^{\mathcal{C}} \wedge (\gamma_{\mathcal{G}}(x_i), x_i) \in R_{\mathcal{F}}^{\mathcal{C}}$ holds.*

Finally, having a characterization for strong equivalence under rule refinement, we now prove that it generalizes the standard notion of strong equivalence for atomic programs.

Proposition 18. *For any two atomic programs P and Q, $P \equiv_{r+}^{\Lambda} Q$ implies $P \equiv_s Q$.*

6. Related Work

The notion of strong equivalence has been extensively investigated in both logic programming and argumentation. For logic programs, strong equivalence has been characterized using SE-models [1], which offer a semantic basis for comparing programs under arbitrary expansions. An SE-model of a program P is a pair (X, Y) of sets of atoms such that $X \subseteq Y$, $Y \models P$ and $X \models P^Y$. Two logic programs are strongly equivalent iff their SE-models coincide. Building on this, Delgrande et al. [16] defined belief revision operators for LPs and established representation theorems for various program classes, connecting strong equivalence with stability after revisions. Studying the relation between revision and update operators in ASP [17] and our notion of rule refinement is subject of future investigations.

Besides aforementioned works for strong equivalence in Dung-style AFs and CAFs, more fine-grained notions of strong equivalence have been introduced, see e.g. [18], and the concept of strong equivalence has been extended to argumentation frameworks with collective attacks, abstract dialectical frameworks as well as structured argumentation [19, 20, 21]. The work of Baumann and Strass [22] shares with our research the motivation of bridging the gap between notions of strong equivalence across different formalisms. Indeed, they provide a general semantic framework for strong equivalence that is formally independent of specific formalism, and capable of subsuming both logic programs and Dung-style AFs. In contrast, by remaining on the syntactic level, our work is closer to original characterizations of strong equivalence and addresses the mismatch in a direct way.

7. Conclusion and Future Work

In this paper, we analyzed the correspondence between certain classes of logic programs and abstract argumentation frameworks. In the static setting, we extended existing semantic equivalence results from strict to non-strict programs and AFs, by appealing to the concept of ungrounded attack. Further, we adapted the definition of well-formed CAFs to align precisely with (possibly non-strict) atomic programs. We then examined equivalence in dynamic contexts, demonstrating how the correspondence between strongly equivalent programs and argumentation frameworks gets lost in translation. We identified the source of such misalignment in the diverging representations of expansions. To overcome the issue, we proposed a new notion of update for (h-unique) atomic programs, called rule refinement. Based on this, we then defined the novel notion of strong equivalence under Rule Refinement for (h-unique) atomic programs and investigated its relations with the standard one. We have proven that RR strong equivalence for (strict) h-unique programs characterizes strong equivalence for (strict) AFs. Finally, we considered the class of atomic programs. In this setting, we provided a direct characterization for RR strong equivalence and showed that it captures strong equivalence between well-formed CAFs. In future research, we aim at providing a characterization of RR strong equivalence for the entire class of normal logic programs, and connect this to possibly more expressive type of argumentation frameworks. We anticipate that this could be achieved by encoding positive dependencies among atoms via some notion of support in the corresponding framework, as for Bipolar AFs [23]. Further, we aim to study the possibility to characterize RR strong equivalence via an equivalence notion in an intermediate logic, analogously to the approach employed for the standard notion and the logic of Here-and-There [1].

Acknowledgments

This work has been supported by the European Union's Horizon 2020 research and innovation programme (under grant agreement 101034440).

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] V. Lifschitz, D. Pearce, A. Valverde, Strongly equivalent logic programs, ACM Trans. Comput. Log. 2 (2001) 526–541. doi:10.1145/383779.383783.

[2] H. Turner, Strong equivalence for logic programs and default theories (made easy), in: LPNMR 2001, LNCS 2173, pp. 81–92. doi:10.1007/3-540-45402-0_6.

[3] P. Cabalar, A three-valued characterization for strong equivalence of logic programs, in: AAAI 2002, pp. 106–111.

[4] T. Eiter, M. Fink, S. Woltran, Semantical characterizations and complexity of equivalences in answer set programming, ACM Trans. Comput. Log. 8 (2007). doi:10.1145/1243996.1244000.

[5] M. Truszczynski, Strong and uniform equivalence of nonmonotonic theories - an algebraic approach, Ann. Math. Artif. Intell. 48 (2006) 245–265. doi:10.1007/S10472-007-9049-2.

[6] E. Oikarinen, S. Woltran, Characterizing strong equivalence for argumentation frameworks, Artif. Intell. 175 (2011) 1985–2009.

[7] I. Rahwan, G. R. Simari, Argumentation in Artificial Intelligence, Springer, 2009.

[8] P. Baroni, D. Gabbay, M. Giacomin, L. van der Torre (Eds.), Handbook of Formal Argumentation, College Publications, 2018.

[9] P. M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, Artif. Intell. 77 (1995) 321–358.

[10] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: ICLP, 1988, pp. 1070–1080.

[11] M. Caminada, S. Sá, J. Alcântara, W. Dvořák, On the equivalence between logic programming semantics and argumentation semantics, Int. J. Approx. Reasoning 58 (2015) 87–111. doi:10.1016/j.ijar.2014.12.004.

[12] W. Dvořák, S. Woltran, Complexity of abstract argumentation under a claim-centric view, Artif. Intell. 285 (2020) 103290. doi:10.1016/j.artint.2020.103290.

[13] W. Dvořák, A. Rapberger, S. Woltran, On the different types of collective attacks in abstract argumentation: Equivalence results for SETAFs, Journal of Logic and Computation 30 (2020) 1063–1107. doi:10.1093/logcom/exaa033.

[14] R. Baumann, A. Rapberger, M. Ulbricht, Equivalence in argumentation frameworks with a claim-centric view: Classical results with novel ingredients, J. Artif. Intell. Res. 77 (2023) 891–948. doi:10.1613/JAIR.1.14625.

[15] M. König, A. Rapberger, M. Ulbricht, Just a matter of perspective – intertranslating expressive argumentation formalisms, in: Proceedings of COMMA 2022, volume 353 of *FAIA*, IOS Press, 2022, pp. 212–223. doi:10.3233/FAIA220154.

[16] J. P. Delgrande, P. Peppas, S. Woltran, Agm-style belief revision of logic programs under answer set semantics, in: LPNMR 2013, LNCS 8148, pp. 264–276. doi:10.1007/978-3-642-40564-8_27.

[17] J. Leite, M. Slota, A brief history of updates of answer-set programs, Theory Pract. Log. Program. 23 (2023) 57–110. doi:10.1017/S1471068422000060.

[18] R. Baumann, Normal and strong expansion equivalence for argumentation frameworks, Artif. Intell. 193 (2012) 18–44. doi:10.1016/j.artint.2012.08.004.

[19] W. Dvořák, A. Rapberger, S. Woltran, Strong equivalence for argumentation frameworks with collective attacks, in: KI 2019, LNCS 11793, pp. 131–145.

[20] J. Heyninck, G. Kern-Isberner, T. Rienstra, K. Skiba, M. Thimm, Revision, defeasible conditionals and non-monotonic inference for abstract dialectical frameworks, Artif. Intell. 317 (2023) 103876. doi:10.1016/J.ARTINT.2023.103876.

[21] A. Rapberger, M. Ulbricht, On dynamics in structured argumentation formalisms, J. Artif. Intell. Res. 77 (2023) 563–643. doi:10.1613/JAIR.1.14481.

[22] R. Baumann, H. Strass, An abstract, logical approach to characterizing strong equivalence in non-monotonic knowledge representation formalisms, Artif. Intell. 305 (2022) 103680.

[23] C. Cayrol, M.-C. Lagasquie-Schiex, On the acceptability of arguments in bipolar argumentation frameworks, in: ECSQARU 2005), LNCS 3571, pp. 378–389.