Artificial Intelligence 255 (2018) 43–70

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Fixpoint semantics for active integrity constraints

Bart Bogaerts [a,*], Luís Cruz-Filipe [b]

[a] KU Leuven, Department of Computer Science, Celestijnenlaan 200A, Leuven, Belgium
[b] University of Southern Denmark, Department of Mathematics and Computer Science, Campusvej 55, Odense, Denmark

ARTICLE INFO

Article history:
Received 4 July 2017
Received in revised form 2 October 2017
Accepted 18 November 2017
Available online 23 November 2017

Keywords:
Active integrity constraints
Approximation fixpoint theory

ABSTRACT

Active integrity constraints (AICs) constitute a formalism to associate with a database not just the constraints it should adhere to, but also how to fix the database in case one or more of these constraints are violated. The intuitions regarding which repairs are "good" given such a description are closely related to intuitions that live in various areas of non-monotonic reasoning, such as logic programming and autoepistemic logic.

In this paper, we apply *approximation fixpoint theory*, an abstract, algebraic framework designed to unify semantics of non-monotonic logics, to the field of AICs. This results in a new family of semantics for AICs. We study properties of our new semantics and relationships to existing semantics. In particular, we argue that two of the newly defined semantics stand out. *Grounded repairs* have a simple definition that is purely based on semantic principles that semantics for AICs should adhere to. And, as we show, they coincide with the intended interpretation of AICs on many examples. The second semantics of interest is the AFT-well-founded semantics: it is a computationally cheap semantics that provides upper and lower bounds for many other classes of repairs.

1. Introduction

One of the key components of modern-day databases are integrity constraints: logical formulas that specify semantic relationships between the data being modeled that have to be satisfied at all times. When the database is changed (typically due to updating), it is necessary to check if its integrity constraints still hold; in the negative case, the database must be repaired.

The problem of database repair has been an important topic of research for more than thirty years [1]. There are two major problems when deciding how to repair an inconsistent database: *finding* possible repairs and *choosing* which one to apply. Indeed, there are typically several ways to fix an inconsistent database, and several criteria to choose the "best" one have been proposed over the years. Among the most widely accepted criteria are minimality of change [45,25] – change as little as possible – and the common-sense law of inertia (discussed in, e.g., [33]) – do not change anything unless there is a reason for the change.

A typical implementation of integrity constraints in database systems is by means of event–condition–action (ECA) rules [38,44], which specify update actions to be performed when a particular event (a trigger) occurs and specific conditions hold. ECA rules are widely used in practice, as they are simple to implement and their individual semantics is easy

* Corresponding author.
E-mail addresses: bart.bogaerts@cs.kuleuven.be (B. Bogaerts), lcfilipe@gmail.com (L. Cruz-Filipe).

https://doi.org/10.1016/j.artint.2017.11.003
0004-3702/

to understand. However, the lack of declarative semantics for ECA rules makes their interaction complex to analyze and their joint behavior hard to understand.

The formalism of *active integrity constraints (AICs)* [27] was inspired by a similar idea. AICs express database dependencies through logic programming-style rules that include update actions in their heads. They come with a set of declarative semantics that identifies several progressively more restricted classes of repairs, which can be used as criteria to select a preferred repair [13]. These repairs can be computed directly by means of tree algorithms [17], which have been implemented as a prototype [16].

Example 1.1. We motivate the use of AICs in practice by means of a simple example. Consider a company's database, including tables employee and dept (relating employees to the department where they work). In particular, each employee is assigned to a unique department; if an employee is listed as working in two different departments, then the database is inconsistent, and this inconsistency must be fixed by removing one of those entries.

We can write this requirement as the following AIC.

$$\forall x, y, z : \text{employee}(x), \text{dept}(x, y), \text{dept}(x, z), y \neq z \supset -\text{dept}(x, y)$$

The intended meaning of this rule is: if all the literals in the lefthandside (body) of the rule are true in some state of the database, for particular values of x, y and z, then the database is inconsistent, and this inconsistency can be solved by performing the action on the right.

Suppose that the database is

$$DB = \{\text{employee}(\text{john}), \text{dept}(\text{john}, \text{finance}), \text{dept}(\text{john}, \text{hr})\}.$$

This database is inconsistent, and applying our AIC with $x = \text{john}$, $y = \text{finance}$ and $z = \text{hr}$ gives us a possible fix consisting of the action "remove dept(john, finance)". Observe, however, that the instantiation $x = \text{john}$, $y = \text{hr}$ and $z = \text{finance}$ detects the same inconsistency, but proposes instead the fix "remove dept(john, hr)": in general, there can be several different ways to repair inconsistencies.

AICs may also interact with each other. Suppose that we add the constraint

$$\forall x, y, z : \text{supervisor}(x, y), \text{dept}(x, z), \neg\text{dept}(y, z) \supset +\text{dept}(y, z) \tag{1}$$

stating that employees can only supervise people from their own department, and that whenever this constraint is violated, the department of the supervisee needs to be updated (i.e., the supervisor table and the department of the supervisor are deemed correct). If the database is now

$$DB = \{\text{employee}(\text{john}), \text{employee}(\text{ann}), \text{dept}(\text{john}, \text{finance}), \text{dept}(\text{ann}, \text{hr}), \text{supervisor}(\text{ann}, \text{john})\}$$

then this AIC detects an inconsistency, and suggests that it be fixed by adding the entry dept(john, hr). The database is still inconsistent, though, since there are now two entries for John in the dept table; restoring inconsistency would also require removing the entry dept(john, finance).

An alternative repair of the integrity constraint that the supervisee and supervisor should belong to the same department would be to change the department information associated with ann. By using *active* integrity constraints, we discard this solution: rule (1) only allows to insert a new department for the supervisee. If we additionally also want to allow changing ann's department, we need an extra constraint. ▲

It is striking that many intuitions about what "good" repairs are, such as minimality of change, are similar to intuitions that surfaced in other domains of non-monotonic reasoning, such as logic programming [39] and default logic [34]. Still, it has been hard to find satisfying semantics for AICs. As shown by Cruz-Filipe et al. [17], the semantics of so-called *founded repairs* [12] unexpectedly fails to respect the common-sense law of inertia, while the more restricted semantics of justified repairs [13] forbids natural repairs in some cases. That work proposed the operational semantics of well-founded repairs, which however is not modular [14] and is therefore severely restricted in its practical applicability.

In this work, we begin by defining a new semantics for AICs that avoids these problems: *grounded repairs*. Grounded repairs are natural counterparts to existing semantics in various non-monotonic reasoning domains such as logic programming; we discuss how they relate to other semantics for AICs. We also argue that grounded repairs match our intuitions regarding AICs on a broad set of examples.

We then give a more abstract characterization of the different semantics for AICs by associating with each set of AICs η a semantic operator $\mathcal{T}_\eta$. This operator immediately induces several semantics:

(i) *weak repairs* are *fixpoints* of $\mathcal{T}_\eta$;
(ii) *repairs* are *minimal fixpoints* of $\mathcal{T}_\eta$;
(iii) *grounded repairs* are *grounded fixpoints* [7] of $\mathcal{T}_\eta$.

The first two semantics are pre-existing semantics for AICs that we recover in an operator-based fashion.

Next, we define a three-valued variant of $\mathcal{T}_\eta$. In the terminology of *approximation fixpoint theory (AFT)* [19] our three-valued operator is an *approximator* of the original semantic operator. Given such an approximator $\mathfrak{T}_\eta$, AFT induces a few more semantics:

(iv) the *Kripke–Kleene repair* is the *Kripke–Kleene fixpoint* of $\mathfrak{T}_\eta$;
(v) the *AFT-well-founded repair* is the *well-founded fixpoint* of $\mathfrak{T}_\eta$;
(vi) *(partial) stable repairs* are *(partial) stable fixpoints* of $\mathfrak{T}_\eta$;
(vii) *partial grounded repairs* are *partial grounded fixpoints* of $\mathfrak{T}_\eta$.

We again study properties of these new semantics and study how they compare to existing semantics. Furthermore, we argue that, from a practical point of view, the AFT-style well-founded semantics is very valuable. Indeed, we show that the AFT-well-founded repair can be computed in polynomial time, and that, on a broad set of practical examples, it corresponds to the intuitions underlying database repairs, providing natural upper and lower bounds on the set of acceptable repairs (formally: the AFT-style well-founded model approximates all justified, stable and grounded repairs).

All our semantics are defined within the framework of approximation fixpoint theory, a general algebraic framework for studying logics with a fixpoint semantics. This framework was initially developed by Denecker, Marek and Truszczyński, henceforth referred to as DMT [20], after identifying analogies in the semantics of logic programming [39], autoepistemic logic (AEL) [32] and default logic, hereafter abbreviated to DL [34]. The theory defines different types of fixpoints for what are called *approximating* operators, or *approximators*. In the context of logic programming, DMT [20] showed that Fitting's (three- or four-valued) immediate consequence operator is an approximator of the usual (two-valued) immediate consequence operator, and that the major semantics of logic programs coincide with the (equally named) different types of fixpoints of that approximator. They then identified approximators for both default and autoepistemic logic, showing that AFT induces all main semantics in these fields, as well as some new ones [21], thus unifying DL and AEL in a deep sense. More recently, Strass [36] showed that AFT can also be used to characterize the major semantics of Dung's argumentation frameworks [24] and abstract dialectical frameworks [11]. Other recent applications of AFT include: defining extensions of logic programming [2], defining new logics [10], integrating different formalisms [3], studying complexity [37], and studying modularity and predicate introduction in a uniform way [41–43].

As such, the contribution of this work goes beyond the definition of new semantics for AICs. By integrating active integrity constraints in AFT, we provide solid foundations for applying a rich algebraic theory to AICs. For instance, we can now directly apply existing results from AFT, such as modularity results and predicate introduction results to AICs. It remains to be researched how these related for instance to existing modularity results for AICs [14,15]. Furthermore, our work paves the way to applying AFT to revision programming, following the results from Caroprese and Truszczyński [13], and to AICs outside the database world, as generalized by Cruz-Filipe et al. [18].

The rest of this paper is structured as follows. In Section 2, we provide preliminaries related to active integrity constraints. In Section 3 we discuss the semantics of grounded repairs. While our definitions are motivated from approximation fixpoint theory, their semantics can also be given without this machinery, hence we start with a direct definition. Next, in Section 4, we give background on approximation fixpoint theory. In Section 5, we define a semantic operator for AICs and show that its grounded fixpoints indeed correspond to grounded repairs, as defined in Section 3. Next, we define an approximator of our operator in Section 6 and use it to derive more AFT-style semantics for AICs; we study how these semantics relate to existing semantics. Afterwards, in Section 7, we discuss the relationship between our newly defined semantics and the equally-named semantics for logic programming. In Section 8, we study complexity of various tasks related to our newly defined semantics. We conclude in Section 9.

Publication history The semantic operator for grounded AICs and the resulting semantics of grounded repairs were originally proposed by Cruz-Filipe [15]. The approximator for this operator and its properties were introduced by Bogaerts and Cruz-Filipe [6]. Our current work combines results from those conference papers and extends it with proofs, examples, and a detailed analysis of the connection between the approximation semantics for AICs and logic programming.

2. Preliminaries: active integrity constraints

In this section we summarize previous work on active integrity constraints (AICs), including results developed by Flesca et al. [27], Caroprese et al. [12], Caroprese and Truszczyński [13] and Cruz-Filipe et al. [17].

We assume a fixed set At of *atoms*. An *interpretation* or *database* is a subset of At. In the current paper, following, e.g., Cruz-Filipe et al. [17], we assume, At to be a finite set. This restriction is not essential for defining our semantics, or to any of the theorems we prove about them, with the exception of complexity results and comparison with existing semantics (that have only been defined in the finite case). A *literal* is an atom a or its negation $\neg a$. We say that $\neg a$ is the *dual* literal of a and vice versa, and denote the dual of a literal l by l^D. Propositional formulas are defined as usual: atoms are formulas, the negation of a formula is a formula, and the conjunction of formulas is a formula. The satisfaction relation between databases DB and formulas is defined as usual:

- $DB \models a$ if $a \in DB$,

- $DB \models \neg\varphi$ if $DB \not\models \varphi$,
- $DB \models \varphi \wedge \psi$ if $DB \models \varphi$ and $DB \models \psi$

for all atoms a and all formulas φ and ψ.

An *update action* α has the form $+a$ or $-a$ with $a \in At$. We call $+a$ and $-a$ *dual actions* and use α^D to denote the dual action of α. Intuitively, update actions represent changes to the database: $+a$ adds a, while $-a$ removes a. Formally, $+a$ transforms DB into $DB \cup \{a\}$, and $-a$ transforms DB into $DB \setminus \{a\}$. A set of update actions $\mathcal{U}$ is *consistent* if it does not contain both an action and its dual. A consistent set of update actions $\mathcal{U}$ acts on a database DB by executing all its actions simultaneously; we denote the result of this operation by $\mathcal{U}(DB)$. If α is an update action, we simply write $\alpha(DB)$ for the result of applying α to DB, i.e., for $\{\alpha\}(DB)$.

Literals and update actions are related by mappings lit and ua, where $\mathsf{lit}(+a) = a$, $\mathsf{lit}(-a) = \neg a$ and ua is the inverse of lit. These mappings naturally extend to sets of literals/actions.

Definition 2.1. An *active integrity constraint* (AIC) is a rule r of the form

$$l_1 \wedge \cdots \wedge l_n \supset \alpha_1 \mid \cdots \mid \alpha_k \tag{1}$$

such that $\mathsf{lit}(\alpha_i^D) \in \{l_1, \ldots, l_n\}$ for each i. We call $l_1 \wedge \cdots \wedge l_n$ the *body* of r, denoted $\mathsf{body}(r)$, and $\alpha_1 \mid \cdots \mid \alpha_k$ the head of r, denoted $\mathsf{head}(r)$.

The informal reading of the above rule is: "If each of the l_i holds in DB, then DB is inconsistent. It is allowed to repair this inconsistency by executing one or more of the α_i." The body of an AIC represents a constraint a database should adhere to and its head represents a set of atoms that are allowed to be changed in order to fix the constraint, in case it is violated. Intuitively, atoms should only be changed if there is some rule that allows it. Furthermore, the only actions that are able to repair the inconsistency detected by the body of an AIC are those corresponding to the duals of its literals [12], hence we restrict the actions allowed in the head to these. Contrary to the seminal work on AICs [27], we only consider propositional AICs, i.e., we do not allow first-order variables (note that we did use them in Example 1.1). However, the restrictions in that work (more precisely, range restrictedness) ensure that we can always reduce to the propositional case by means of *grounding*.

An AIC is called *normal* if $k = 1$. The *normalization* of an AIC of the form (1) is the set of AICs

$$\{l_1 \wedge \cdots \wedge l_n \supset \alpha_i \mid 1 \leq i \leq k\}.$$

It follows from the informal explanation above that we expect normalization to preserve semantics. Indeed, this is the case for most semantics of AICs – the notorious exception being the semantics of justified repairs [13], which also poses several other problems [17]. In the current paper, we assume that, unless explicitly mentioned otherwise, *all AICs are normal*. Extensions of the semantics we define for non-normal AICs can be obtained through normalization, if needed.

Definition 2.2. A set of update actions $\mathcal{U}$ is a *weak repair* for DB and a set η of AICs (shortly, for $\langle DB, \eta\rangle$) if:

- every action in $\mathcal{U}$ changes DB, and
- $\mathcal{U}(DB) \not\models \mathsf{body}(r)$ for each $r \in \eta$.

A $\subseteq$-minimal weak repair is called a *repair*.

(Weak) repairs do not take the head of AICs into account, and thus allow arbitrary changes to the database.

Example 2.3. Consider the restriction that "if a and b both hold, then c and d should also hold". In propositional logic, such a restriction can be expressed by the following formula:

$$(a \wedge b) \Rightarrow (c \wedge d),$$

or, equivalently by the two formulas

$$\neg(a \wedge b \wedge \neg c)$$
$$\neg(a \wedge b \wedge \neg d).$$

Now, the AIC formalism provides, besides the ability to express these constraints, also control over what should happen when one of them is violated. Assuming that in such a case, we only wish to modify a or b, the corresponding AICs are:

$$a \wedge b \wedge \neg c \supset -a \mid -b$$
$$a \wedge b \wedge \neg d \supset -a \mid -b$$

where the first one expresses that if a and b hold, but c does not (i.e., if the constraint is violated), we should remove one of the two literals a and b and the second constraint is similar for d. Despite the inclusion of explicit repair actions in the heads of these rules, the notions of weak repair and repair do not take them into account. Suppose a given database is $DB = \{a, b\}$. In this case, $\{-a\}$, $\{-b\}$ and $\{+c, +d\}$ are repairs; furthermore, sets such as $\{-a, -b\}$ or $\{-b, +c\}$ are weak repairs: in all cases, all actions change DB, and the result always negates at least one literal in the body of each of the rules.

Sets such as $\{-a, -c\}$ and $\{+a, +c, +d\}$ also solve the inconsistency, but they include actions that do not change the database, and therefore are not considered weak repairs. Sets such as $\{+a, -a\}$ are inconsistent, as it is not clear whether they state that the update of DB should include a or not.

Applying normalization yields the following set of AICs.

$$a \wedge b \wedge \neg c \supset -a$$
$$a \wedge b \wedge \neg d \supset -a$$
$$a \wedge b \wedge \neg c \supset -b$$
$$a \wedge b \wedge \neg d \supset -b$$

It is immediate to check that everything discussed above with respect to the original set of AICs also applies to its normalized counterpart, i.e., intuitively, these constraints represent the same knowledge and the sets of repairs and weak repairs remain unchanged. ▲

We now review several other semantics for AICs that have been defined with the intention to allow only changes explicitly allowed by one of the AICs: founded (weak) repairs [12], justified (weak) repairs [13], and well-founded (weak) repairs [17].

Definition 2.4 *([12]).* A set of update actions $\mathcal{U}$ is *founded* with respect to $\langle DB, \eta\rangle$ if, for each $\alpha \in \mathcal{U}$, there is a rule $r \in \eta$ with $\alpha \in \mathsf{head}(r)$ and such that $\mathcal{U}'(DB) \models \mathsf{body}(r)$, where $\mathcal{U}' = \mathcal{U} \setminus \{\alpha\}$. A *founded (weak) repair* is a (weak) repair that is founded.

The intuition behind this definition is that for each element in a "good" repair there should be a reason such that: if the element is removed, some constraint is violated and the removed element is an allowed fix.

Example 2.5. Consider again the database $DB = \{a, b\}$ together with the set η of normalized AICs from the previous example.

$$a \wedge b \wedge \neg c \supset -a \qquad (r_1)$$
$$a \wedge b \wedge \neg d \supset -a \qquad (r_2)$$
$$a \wedge b \wedge \neg c \supset -b \qquad (r_3)$$
$$a \wedge b \wedge \neg d \supset -b \qquad (r_4)$$

The set $\{-a\}$ is founded with respect to $\langle DB, \eta\rangle$: if its only action is removed, then rule r_1 is applicable, and $-a$ occurs in $\mathsf{head}(r_1)$. Likewise, the set $\{-b\}$ is also a founded repair for $\langle DB, \eta\rangle$.

On the other hand $\mathcal{U} = \{+c, +d\}$ is not founded. If we remove e.g. $+c$ from $\mathcal{U}$, obtaining $\mathcal{U}' = \{+d\}$, then $\mathcal{U}'(DB) = \{a, b, d\}$, and:

- $\mathcal{U}'(DB) \models \mathsf{body}(r_1)$, but $+c \notin \mathsf{head}(r_1)$, and likewise for r_3;
- $\mathcal{U}'(DB) \not\models \mathsf{body}(r_2)$ and $\mathcal{U}'(DB) \not\models \mathsf{body}(r_4)$,

and thus there is no support for $+c$ in $\mathcal{U}$. In this case, there is also no support for $+d$.

Caroprese and Truszczyński [13] discovered that there can be founded repairs exhibiting *circularity of support*. The following example is due to Cruz-Filipe et al. [17].

Example 2.6. Consider the following set of AICs η, expressing that a and b are equivalent (and if this is not the case, then they should both become false) and that c should be true whenever a or b is true.

$$a \wedge \neg b \supset -a \qquad (r_5)$$
$$\neg a \wedge b \supset -b \qquad (r_6)$$
$$a \wedge \neg c \supset +c \qquad (r_7)$$
$$b \wedge \neg c \supset +c \qquad (r_8)$$

Suppose that the database is $DB = \{a, b\}$. There are two repairs for $\langle DB, \eta\rangle$: $\mathcal{U}_1 = \{-a, -b\}$ and $\mathcal{U}_2 = \{+c\}$. Intuitively, the rules suggest that $\mathcal{U}_2$ should be the preferred repair, since it includes the action suggested by the only AIC that is not satisfied, and indeed $\mathcal{U}_2$ is founded (removing its only element yields $\emptyset$, and as we observed both r_7 and r_8 provide support for $+c$ given the state of DB).

However, $\mathcal{U}_1$ is also a founded repair. If we remove $-a$ from $\mathcal{U}_1$, we obtain $\mathcal{U}_1' = \{-b\}$, and $\mathcal{U}_1'(DB) = \{a\}$, where r_5 is applicable and $-a \in \mathsf{head}(r_5)$. Dually, if we remove $-b$ we obtain $\mathcal{U}_1'' = \{-a\}$, and $\mathcal{U}_1''(DB) = \{b\}$; now r_6 is applicable, and $-b \in \mathsf{head}(r_6)$. Thus both actions in $\mathcal{U}_1$ are founded, hence this is a founded repair.

The problem in this example is that the property of being a founded repair only excludes *individual* actions that are not supported by the remaining ones, rather than *sets* of actions with this characteristic. In order to avoid this unwanted characteristic, Caroprese and Truszczyński [13] proposed considering justified repairs, which we now define.[1]

Definition 2.7 *([13]).* Let $\mathcal{U}$ be a set of update actions and $\langle DB, \eta\rangle$ a database.

- The *no-effect actions* with respect to DB and $\mathcal{U}$, $\mathsf{neff}_{DB}(\mathcal{U})$, are the actions that change neither DB, nor $\mathcal{U}(DB)$.

$$\begin{aligned}\mathsf{neff}_{DB}(\mathcal{U}) &= \{+a \mid a \in DB \cap \mathcal{U}(DB)\} \cup \{-a \mid a \notin DB \cup \mathcal{U}(DB)\}\\ &= \{\alpha \mid \alpha(DB) = DB \wedge \alpha(\mathcal{U}(DB)) = \mathcal{U}(DB)\}\,.\end{aligned}$$

- The set of *non-updatable literals* of an AIC r, $\mathsf{nup}(r)$, contains all body literals of r that do not occur in the head of r.

$$\mathsf{nup}(r) = \mathsf{body}(r) \setminus \mathsf{lit}\left(\mathsf{head}(r)^D\right)\,.$$

- $\mathcal{U}$ is closed under η if for each $r \in \eta$, $\mathsf{ua}(\mathsf{nup}(r)) \subseteq \mathcal{U}$ implies $\mathsf{head}(r) \cap \mathcal{U} \neq \emptyset$.
- $\mathcal{U}$ is a *justified action set* if it is a minimal superset of $\mathsf{neff}_{DB}(\mathcal{U})$ closed under η.
- $\mathcal{U}$ is a *justified (weak) repair* if it is a (weak) repair and $\mathcal{U} \cup \mathsf{neff}_{DB}(\mathcal{U})$ is a justified action set.

Although the notion of closed set of actions does not take the database into account, its role in the definition of justified weak repairs is as part of the definition of justified action set – where all actions that do not change the database are included. In the normalized case that we considered, all justified weak repairs are minimal with respect to set inclusion, i.e., they are justified repairs.

Example 2.8. In the setting of Example 2.6, the sets of non-updateable literals are as follows.

$$\mathsf{nup}(r_5) = \{\neg b\}$$
$$\mathsf{nup}(r_6) = \{\neg a\}$$
$$\mathsf{nup}(r_7) = \{a\}$$
$$\mathsf{nup}(r_8) = \{b\}$$

The founded repair $\mathcal{U}_1$ is not justified. First, observe that $\mathsf{neff}(\mathcal{U}_1) = \{-c\}$ (assuming that a, b and c are the only atoms in the language). Consider $\mathcal{U}' = \emptyset \subseteq \mathcal{U}_1$; then $\mathcal{U}' \cup \mathsf{neff}(\mathcal{U}_1) = \{-c\}$, and $\mathsf{ua}(\mathsf{nup}(r)) \not\subseteq (\mathcal{U}' \cup \mathsf{neff}(\mathcal{U}_1))$ for every $r \in \eta$; therefore $\mathcal{U}' \cup \mathsf{neff}(\mathcal{U}_1)$ is a subset of $\mathcal{U}_1 \cup \mathsf{neff}(\mathcal{U}_1)$ containing $\mathsf{neff}(\mathcal{U}_1)$ that is (trivially) closed under η.

In contrast, the repair $\mathcal{U}_2$ is justified. In this case, we have $\mathsf{neff}(\mathcal{U}_2) = \{+a, +b\}$; the only proper subset of $\mathcal{U}_2$ is again $\mathcal{U}' = \emptyset$. Then both r_7 and r_8 satisfy $\mathsf{ua}(\mathsf{nup}(r)) \subseteq (\mathcal{U}' \cup \mathsf{neff}(\mathcal{U}_2)) = \{+a, +b\}$, and in both cases $\mathsf{head}(r) = \{+c\}$. Since $\mathsf{head}(r) \cap \{+a, +b\} = \emptyset$, we conclude that $\mathcal{U}' \cup \mathsf{neff}(\mathcal{U}_2)$ is not closed under η. ▲

The relation between founded and justified weak repairs was established by Caroprese and Truszczyński [13].

Lemma 2.9. *Let DB be a database, η be a set of AICs over DB and $\mathcal{U}$ be a set of update actions over DB. If $\mathcal{U}$ is a justified weak repair for $\langle DB, \eta\rangle$, then $\mathcal{U}$ is a founded weak repair for $\langle DB, \eta\rangle$.*

One of the main issues with the notion of justified repair is that it is too restrictive. Interestingly, this can already be seen by an example originally given by Caroprese and Truszczyński [13], which we reproduce below.

[1] Caroprese and Truszczyński [13] never formally define circularity of support, or discuss why justified repairs avoid it. We give an informal argument to sustain this claim in the discussion after Proposition 3.7 below.

Example 2.10. Consider the following set of AICs η.

$$a \wedge b \supset -a \tag{r_9}$$
$$a \wedge \neg b \supset -a \tag{r_{10}}$$
$$\neg a \wedge b \supset -b \tag{r_{11}}$$

Consider the same database $DB = \{a, b\}$ as before. This database is inconsistent (it does not satisfy r_9), and the only possible repair is $\mathcal{U} = \{-a, -b\}$. Furthermore, this repair is intuitively compatible with η, since rule r_9 requires us to remove a from DB, which triggers r_{11} and forces us also to remove b.

The repair $\mathcal{U}$ is founded, but for a different reason: if we remove $-a$ from $\mathcal{U}$, then r_{10} is applicable, and its head includes $-a$, while removing $-b$ from $\mathcal{U}$ makes r_{11} applicable, and its head includes $-b$. Caroprese and Truszczyński [13] considered this as another instance of circularity of support (see [17] for a discussion). The repair $\mathcal{U}$ is not justified: $\mathsf{neff}(\mathcal{U}) = \emptyset$, and taking $\mathcal{U}' = \emptyset$ we have that $\mathcal{U}' \cup \mathsf{neff}(\mathcal{U}) = \emptyset$ is again trivially closed under η (every rule has non-updateable literals in its body). ▲

A different attempt to resolve the problems of circularity posed by founded repairs, while avoiding the over-restrictiveness of justified repairs, was the introduction of well-founded repairs by Cruz-Filipe et al. [17] – a third kind of repairs, motivated by an operational approach directly inspired by the syntax of AICs.

Definition 2.11 *([17]).* A (weak) repair $\mathcal{U}$ for $\langle DB, \eta\rangle$ is *well-founded* if there exists a sequence of actions $\alpha_1, \ldots, \alpha_n$ such that $\mathcal{U} = \{\alpha_1, \ldots, \alpha_n\}$ and, for each $i \in \{1, \ldots, n\}$, there is a rule r_i such that $U_{i-1}(DB) \models \mathsf{body}(r_i)$ and $\alpha_i \in \mathsf{head}(r_i)$, where $U_{i-1} = \{\alpha_1, \ldots, \alpha_{i-1}\}$.

Example 2.12. In the setting of Example 2.6, the only well-founded repair for $\langle DB, \eta\rangle$ is $\{+c\}$, as r_7 and r_8 are the only rules applicable in DB.

Likewise, the repair $\mathcal{U}$ in Example 2.10 is well-founded, as it can be constructed by applying first r_9 (introducing $-a$) and afterwards r_{11}. ▲

However, well-founded repairs can also behave unexpectedly.

Example 2.13. Let η be the set of AICs containing

$$\neg a \supset +a \tag{r_{12}}$$
$$\neg a \wedge \neg b \supset +b \tag{r_{13}}$$
$$a \wedge \neg b \wedge \neg c \supset +c \tag{r_{14}}$$

and consider $DB = \emptyset$. There are two well-founded repairs for $\langle DB, \eta\rangle$: $\mathcal{U}_1 = \{+b, +a\}$, obtained by applying first r_{13} and then r_{12}, and $\mathcal{U}_2 = \{+a, +c\}$, obtained by applying first r_{12} and then r_{14}. It is arguable that $\mathcal{U}_2$ is preferable, as it is not reasonable to apply r_{13} when r_{12} is also applicable, since any action that solves the inconsistency detected by r_{12} also repairs r_{13}, but not conversely. However, the well-founded semantics for AICs cannot infer this restriction. ▲

In this example, it is interesting to note that $\mathcal{U}_2$ is a founded repair. Indeed, r_{12} supports $+a$ and this support is independent of b and c; furthermore r_{14} supports $+c$ when $+a$ is present. However, $\mathcal{U}_1$ is not founded, since the action $+b$ is not supported, as r_{13} is not applicable once a has been added to DB.

The examples above show that there exist both founded weak repairs that are not well-founded, and well-founded weak repairs that are not founded. These relations were established by Cruz-Filipe et al. [17], together with the connection to justified weak repairs.

Lemma 2.14. *Let DB be a database, η be a set of AICs over DB and $\mathcal{U}$ be a set of update actions over DB. If $\mathcal{U}$ is a justified weak repair for $\langle DB, \eta\rangle$, then $\mathcal{U}$ is a well-founded weak repair for $\langle DB, \eta\rangle$.*

We are also interested in the *shifting property*. Originally defined by Marek and Truszczynski [31] in the context of revision programming, this property was later transferred to active integrity constraints [13]. Intuitively, a semantics for AICs possesses the shifting property if uniformly replacing some literals with their duals preserves the semantics at hand, i.e., if the semantics treats truth and falsity of elements in the database symmetrically.

Definition 2.15. Let $S \subseteq At$ be a set of atoms and l a literal. The *shift of l* with respect to S is defined as

$$\mathsf{shift}_S(l) = \begin{cases} l & \text{if } l \notin S \\ l^D & \text{otherwise} \end{cases}$$

The shift function is extended to sets of literals, update actions and AICs in the straightforward manner.

Definition 2.16. We say that a semantics for AICs *has the shifting property* if: for all $\langle DB, \eta\rangle$ and all $S \subseteq At$, $\mathcal{U}$ is a repair of $\langle DB, \eta\rangle$ accepted by the semantics if and only if $\mathsf{shift}_S(\mathcal{U})$ is a repair of $\langle \mathsf{shift}_S(DB), \mathsf{shift}_S(\eta)\rangle$ accepted by the semantics.

If a semantics has the shifting property, then we can reduce any situation to the case $DB = \emptyset$ by taking $S = DB$. All semantics discussed in this section have the shifting property.

3. Grounded repairs

Founded, well-founded and justified repairs were all introduced with the purpose of characterizing a class of repairs whose actions are supported (there is a reason for having them in the set), without being self-supporting. I.e., they try to avoid certain forms of circularity of support. Sometimes, one also requires these repairs to be constructible "from the ground up", which was the motivation for defining well-founded repairs. However, all notions exhibit unsatisfactory examples: there exist founded repairs with circular support (see, e.g., Example 2.6), and repairs with no circular support that are not justified [17]. In this section, we introduce a new semantics, grounded repairs, aimed at directly tackling this issue.

Grounded repairs are motivated by Example 2.6, where we noticed that the definition of founded repairs does not manage to capture groups of self-supporting arguments. Indeed, there the repair $\mathcal{U}_1$ is founded. It consists of two actions, $-a$ and $-b$ such that whenever one of them is removed from $\mathcal{U}_1$, there is an AIC whose body is violated and whose head is the action in question. However, if we remove *both* of them simultaneously, we notice that no rule any longer applies. As such, we can conclude that these actions are "self-supporting": the only reason to have one of the two actions in the repair of our choice is because the other action is also in there. Our definition of grounded (weak) repair is aimed directly at avoiding this kind of situations.

Definition 3.1. A set of update actions $\mathcal{U}$ is *grounded* with respect to $\langle DB, \eta\rangle$ if, for each $\mathcal{V} \subsetneq \mathcal{U}$, there is a rule $r \in \eta$ such that $\mathcal{V}(DB) \models \mathsf{body}(r)$ and $\mathsf{head}(r) \in (\mathcal{U} \setminus \mathcal{V})$. A *grounded (weak) repair* is a (weak) repair that is grounded.

As can be seen, our definition of groundedness is a slight variant of the notion of foundedness: instead of only considering what happens if *one* action is dropped from a proposed set of update actions, we consider arbitrary removals. A first observation with respect to groundedness is that grounded weak repairs are always minimal, i.e., that each grounded weak repair is a repair.

Proposition 3.2. *All grounded weak repairs of $\langle DB, \eta\rangle$ are $\subseteq$-minimal, i.e., are repairs.*

Proof. Suppose $\mathcal{U}$ is a grounded weak repair and $\mathcal{U}$ is not minimal, i.e., that there exists a $\mathcal{V} \subsetneq \mathcal{U}$ that is also a weak repair. Since $\mathcal{U}$ is grounded, there must exist an AIC whose body is satisfied in $\mathcal{V}(DB)$, contradicting the fact that $\mathcal{V}$ is a weak repair. □

Thus, the notion of groundedness intrinsically embodies the principle of minimality of change, unlike other kinds of weak repairs previously defined.

Proposition 3.3. *Let DB be a database, η be a set of AICs over DB and $\mathcal{U}$ be a grounded repair for $\langle DB, \eta\rangle$. Then $\mathcal{U}$ is both founded and well-founded.*

Proof. Assume that $\mathcal{U}$ is a grounded repair for $\langle DB, \eta\rangle$. The fact that $\mathcal{U}$ is founded follows immediately from the definition of grounded repair, since for each action α, $\mathcal{V} = \mathcal{U} \setminus \{\alpha\}$ is a strict subset of $\mathcal{U}$. Hence, by groundedness of $\mathcal{U}$, there must be a rule r with $\mathsf{head}(r) \in \mathcal{U} \setminus \mathcal{V} = \{\alpha\}$, whose body is satisfied in $\mathcal{V}(DB)$. We find that $\mathcal{U}$ is founded indeed.

Now, we show how to construct a sequence of subsets of $\mathcal{U}$ that illustrates that $\mathcal{U}$ is well-founded. For this sequence, we start from $\mathcal{U}_0 = \emptyset$ and construct $\mathcal{U}_i = \mathcal{U}_{i-1} \cup \{u_i\}$ by picking a rule $r \in \langle DB, \eta\rangle$ with $\mathsf{head}(r) = u_i \in \mathcal{U}$ and $\mathcal{U}_{i-1}(DB) \models \mathsf{body}(r)$. Since $\mathcal{U}$ is grounded, if $\mathcal{U}_{i-1} \subsetneq \mathcal{U}$ then such a rule always exists, and by construction $\mathcal{U}_i \subseteq \mathcal{U}$. But $\mathcal{U}$ is finite, therefore this sequence converges towards $\mathcal{U}$, and thus $\mathcal{U}$ is a well-founded repair. □

However, the notion of grounded repair is strictly stronger than both of these: Example 2.6, presented earlier, also shows that some forms of circular justifications are avoided by grounded repairs.

Example 3.4 *(Example 2.6 continued).* The repair $\mathcal{U}_1 = \{-a, -b\}$ is a founded repair that is not grounded: taking $\mathcal{V} = \emptyset$, we notice that no AIC with $-a$ or $+a$ in the head has its body satisfied in $\mathcal{V}(DB)$. The more natural repair $\mathcal{U}_2 = \{+c\}$ is also founded, and it is immediate to verify that it is also grounded. ▲

Likewise, not all well-founded repairs are grounded.

Example 3.5 *(Example 2.13 continued).* Consider again η from Example 2.13, with $DB = \emptyset$. As shown earlier, the two well-founded repairs for $\langle DB, \eta\rangle$ are $\mathcal{U}_1 = \{+b, +a\}$ and $\mathcal{U}_2 = \{+a, +c\}$. We already observed that $\mathcal{U}_1$ is not founded, so it cannot be grounded; indeed, $\mathcal{V} = \{+a\}$ is a set of update actions such that no rule r with $\mathsf{head}(r) \in (\mathcal{U} \setminus \mathcal{V})$ has its body satisfied in $\mathcal{V}(DB)$.

We thus have that grounded repairs are always founded and well-founded; the next example shows that they do not correspond to the intersection of those classes.

Example 3.6. Consider the following set of AICs η.

$$\neg a, \neg b \supset +a \tag{r_{15}}$$
$$a, \neg b \supset +b \tag{r_{16}}$$
$$\neg a, b \supset -b \tag{r_{17}}$$
$$a, b, \neg c \supset +c \tag{r_{18}}$$
$$a, \neg b, c \supset +b \tag{r_{19}}$$
$$\neg a, b, c \supset +a \tag{r_{20}}$$

Let $DB = \emptyset$. Then $\mathcal{U} = \{+a, +b, +c\}$ is a repair for $\langle DB, \eta\rangle$: the first three constraints require a and b to be included in the database. They do not do this in a straightforward manner. Starting from a database in which b holds, but a does not, rule (r_{17}) first enforces removal of b. Next, rule (r_{15}) adds a and subsequently, rule (r_{16}) adds b. While this combination of constraints seems unnatural (no-one would use these to enforce inclusion of a and b in the database), it is possible that this type of constraints show up if they are written by different developers maintaining the database, having different concerns. The last three rules state that no 2-element subset of $\mathcal{U}$ can be a repair. Given rule (r_{16}), one might say that rule (r_{19}) is redundant. However, again the same argument holds: they *could* show up together in a database. In this example, $\mathcal{U}$ is founded (the last three rules also ensure that) and well-founded (starting with $\emptyset$, we are forced to apply rules r_{15}, r_{16} and r_{18}, in that order).

However, $\mathcal{U}$ is not grounded: if $\mathcal{V} = \{+b\}$, then $\mathcal{V} \subsetneq \mathcal{U}$, but there is no rule r with $\mathsf{head}(r) \in \{+a, +c\}$ whose body is satisfied in $\mathcal{V}(DB)$. ▲

In this situation, $\mathcal{U}$ might seem reasonable; however, observe that the support for its actions *is* circular: it is the three last rules that make $\mathcal{U}$ founded, and none of them is applicable to DB. Also note that $\mathcal{V}(DB) = \{b\}$ is a database for which the given set η behaves very awkwardly: the only applicable AIC tells us to remove b, while the only repair of $\mathcal{V}(DB)$ is actually $\{+a, +c\}$.

We do not feel that this example weakens the case for studying grounded repairs, though: the consensual approach to different notions of repair is that they express *preferences*. In this case, where $\langle DB, \eta\rangle$ admits no grounded repair, it is sensible to allow a repair in a larger class – and a repair that is both founded and well-founded is a good candidate. The discussion by Caroprese and Truszczyński [13, Section 8] already proposes such a "methodology": choose a repair from the most restrictive category (justified, founded, or any). We advocate a similar approach, but including grounded repairs among the possible choices.

We now investigate the relation between grounded and justified repairs, and find that all justified repairs are grounded, but not conversely – in line with our earlier claim that the notion of justified repair is too strong.

Proposition 3.7. *Let DB be a database, and let η be a set of normal AICs over DB. If $\mathcal{U}$ is a justified repair for $\langle DB, \eta\rangle$, then $\mathcal{U}$ is grounded.*

Proof. Let $\mathcal{U}$ be a justified repair for $\langle DB, \eta\rangle$ and take $\mathcal{V} \subsetneq \mathcal{U}$. Then $\mathcal{V} \cup \mathsf{neff}(\mathcal{U})$ is not closed under η, whence there is a rule $r \in \eta$ such that $\mathsf{ua}(\mathsf{nup}(r)) \subseteq \mathcal{V} \cup \mathsf{neff}(\mathcal{U})$ and $\mathsf{head}(r) \notin \mathcal{V} \cup \mathsf{neff}(\mathcal{U})$.

Since $\mathcal{V} \subseteq \mathcal{U}$, also $\mathsf{ua}(\mathsf{nup}(r)) \subseteq \mathcal{U} \cup \mathsf{neff}(\mathcal{U})$, whence $\mathsf{head}(r) \in \mathcal{U} \cup \mathsf{neff}(\mathcal{U})$ as $\mathcal{U}$ is closed under η. But $\mathsf{head}(r) \notin \mathcal{V} \cup \mathsf{neff}(\mathcal{U})$, so $\mathsf{head}(r) \in \mathcal{U} \setminus \mathcal{V}$.

We need to show that also $\mathcal{V} \models \mathsf{body}(r)$. On the one hand, $\mathsf{ua}(\mathsf{nup}(r)) \subseteq \mathcal{V} \cup \mathsf{neff}(\mathcal{U})$ implies that $\mathcal{V}(DB) \models \mathsf{nup}(r)$, as $\mathsf{neff}(\mathcal{U}) \subseteq \mathsf{neff}(\mathcal{V})$; on the other hand, from $\mathsf{head}(r) \in \mathcal{U}$ we know that $\mathsf{lit}(\mathsf{head}(r))^D \in DB$ (all actions in $\mathcal{U}$ change DB), whence $\mathcal{V}(DB) \models \mathsf{lit}(\mathsf{head}(r)^D)$ since $\mathsf{head}(r) \notin \mathcal{V}$. As r is normal, there are no more literals in $\mathsf{body}(r)$, so $\mathcal{V}(DB) \models \mathsf{body}(r)$. Hence, we have found a rule r such that $\mathcal{V} \models \mathsf{body}(r)$ and $\mathsf{head}(r) \in (\mathcal{U} \setminus \mathcal{V})$, thus showing that $\mathcal{U}$ is grounded. □

This proof does not use the hypothesis that $\mathcal{U}$ is a repair. This is due to the fact (already mentioned earlier) that we only consider *normal* AICs in this paper. By Theorem 4 of [13], all justified weak repairs are minimal when η consists of only normal AICs.

Recall Example 2.10, which was used by Cruz-Filipe et al. [17] to point out that justified repairs sometimes eliminate "natural" repairs. This example also shows that the notion of justified repair is stricter than that of grounded repair.

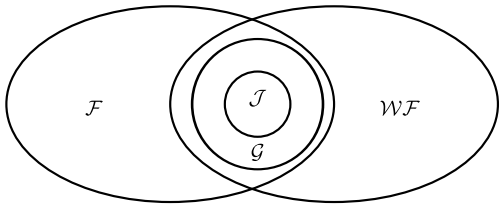


Fig. 1. Relative inclusions between the sets of founded ($\mathcal{F}$), well-founded ($\mathcal{WF}$), grounded ($\mathcal{G}$) and justified ($\mathcal{J}$) repairs. All inclusions are strict.

Example 3.8 *(Example 2.10 continued).* Although the repair $\mathcal{U} = \{-a, -b\}$ for $\langle DB, \eta\rangle$ is not justified, it is grounded: if $-a \in \mathcal{V} \subsetneq \mathcal{U}$, then there is a rule that derives $-b$; otherwise, there is a rule that derives $-a$. ▲

As discussed earlier, in this case the first rule clearly motivates the action $-a$, and the last rule then requires $-b$. This is in contrast to Example 2.6, where there was no clear reason to include either $-a$ or $-b$ in a repair. Hence grounded repairs avoid this type of unreasonable circularities, without being as restrictive as justified repairs.

We summarize the relations between the different types of repairs in Fig. 1.

4. Preliminaries: lattices, operators and approximation fixpoint theory

4.1. Lattices, operators and fixpoints

In this section we summarize the ideas, definitions and main results from approximation fixpoint theory (AFT) that we use in the remainder of the paper.

A *partially ordered set (poset)* $\langle L, \leq\rangle$ is a set L equipped with a partial order $\leq$, i.e., a reflexive, antisymmetric, transitive relation. As usual, we write $x < y$ as abbreviation for $x \leq y \wedge x \neq y$. If S is a subset of L, then x is an *upper bound* (a *lower bound*) of S if for every $s \in S$, it holds that $s \leq x$ ($x \leq s$, respectively). An element x is a *least upper bound* (a *greatest lower bound*) of S if it is an upper bound that is smaller than every other upper bound (a lower bound that is greater than every other lower bound, respectively). If S has a least upper bound (a greatest lower bound) we denote it $\text{lub}(S)$ ($\text{glb}(S)$, respectively). As is custom, we sometimes call a greatest lower bound a *meet*, and a least upper bound a *join* and use the related notations $\bigwedge S = \text{glb}(S)$, $x \wedge y = \text{glb}(\{x, y\})$, $\bigvee S = \text{lub}(S)$ and $x \vee y = \text{lub}(\{x, y\})$. We call $\langle L, \leq\rangle$ a *complete lattice* if every subset of L has a least upper bound and a greatest lower bound. A complete lattice has both a least element $\bot = \bigwedge L$ and a greatest element $\top = \bigvee L$.

A lattice L is *distributive* if $\wedge$ and $\vee$ distribute over each other, i.e., if $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ and $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ for all $x, y, z \in L$. A bounded lattice L is *complemented* if every element $x \in L$ has a complement: an element $\neg x \in L$ satisfying $x \wedge \neg x = \bot$ and $x \vee \neg x = \top$. A *Boolean lattice* is a distributive complemented lattice.

Since we apply our results to (finite) databases, for the sake of simplicity we assume L to be *finite* in this text. All presented results easily generalize to the infinite setting as well.

An operator $O : L \to L$ is *monotone* if $x \leq y$ implies that $O(x) \leq O(y)$. An element $x \in L$ is a *fixpoint* of O if $O(x) = x$. Every monotone operator O in a complete lattice has a least fixpoint, denoted $\text{lfp}(O)$, which is the limit (the least upper bound) of the increasing sequence $(x_i)_{i\in\mathbb{N}}$ defined by $x_0 = \bot$ and $x_{i+1} = O(x_i)$.

Bogaerts et al. [7] called a point $x \in L$ *grounded* for O if, for each $v \in L$ such that $O(v \wedge x) \leq v$, it holds that $x \leq v$. They called a point $x \in L$ *strictly grounded* if there does not exist a y such that $y < x$, and $O(y) \wedge x \leq y$. They explained the intuition underlying these concepts under the assumption that the elements of L are sets of "facts" of some kind and the $\leq$ relation is the subset relation between such sets: in this case, a point x is grounded if it contains only facts that are sanctioned by the operator O, in the sense that if we remove them from x, then the operator will add at least one of them again. Bogaerts [5] showed that for Boolean lattices the notions of groundedness and strict groundedness coincide. In this paper, all lattices of our application are Boolean, hence we use both notions interchangeably.

4.2. Approximation fixpoint theory

Given a lattice L, approximation fixpoint theory (AFT) [20] uses the bilattice L^2. We define two *projection* functions for pairs as usual: $(x, y)_1 = x$ and $(x, y)_2 = y$. Pairs $(x, y) \in L^2$ are used to approximate elements in the interval $[x, y] = \{z \mid x \leq z \wedge z \leq y\}$. We call $(x, y) \in L^2$ *consistent* if $x \leq y$, that is, if $[x, y]$ is non-empty, and use L^c to denote the set of consistent elements. Elements $(x, x) \in L^c$ are called *exact*; they constitute the embedding of L in L^2. We sometimes abuse notation and use the tuple (x, y) and the interval $[x, y]$ interchangeably. The *precision ordering* on L^2 is defined as $(x, y) \leq_p (u, v)$ if $x \leq u$ and $v \leq y$. In case (u, v) is consistent, this means that (x, y) approximates all elements approximated by (u, v), or in other words that $[u, v] \subseteq [x, y]$. If L is a complete lattice, then $\langle L^2, \leq_p\rangle$ is also a complete lattice.

AFT studies fixpoints of lattice operators $O : L \to L$ through operators approximating O. An operator $A : L^2 \to L^2$ is an *approximator* of O if it is $\leq_p$-monotone, and has the property that $A(x, x) = (O(x), O(x))$ for all x. Approximators are internal in L^c (i.e., map L^c into L^c). As usual, we often restrict our attention to *symmetric* approximators: approximators

A such that, for all x and y, $A(x,y)_1 = A(y,x)_2$. Denecker et al. [22] showed that the consistent fixpoints of interest (supported, stable, well-founded) are uniquely determined by an approximator's restriction to L^c, hence, sometimes we only define approximators on L^c.

AFT studies fixpoints of O using fixpoints of A.

- The *A-Kripke–Kleene fixpoint* is the $\leq_p$-least fixpoint of A, and it approximates all fixpoints of O.
- A *partial A-stable fixpoint* is a pair (x,y) such that $x = \text{lfp}(A(\cdot,y)_1)$ and $y = \text{lfp}(A(x,\cdot)_2)$, where $A(\cdot,y)_1$ denotes the operator $L \to L : z \mapsto A(z,y)_1$ and analogously for $A(x,\cdot)_2$.
- The *A-well-founded fixpoint* is the least precise (i.e., the $\leq_p$-minimal) partial A-stable fixpoint.
- An *A-stable fixpoint* of O is a fixpoint x of O such that (x,x) is a partial A-stable fixpoint. This is equivalent to the condition that $x = \text{lfp}(A(\cdot,x)_1)$.
- A *partial A-grounded fixpoint* is a consistent pair (x,y) such that for each $v \in L$, whenever $A(x \wedge v, y \wedge v)_2 \leq v$, also $y \leq v$.

All (partial A-)stable fixpoints are (partial A-)grounded fixpoints and the A-well-founded fixpoint is the least precise partial A-grounded fixpoint [8]. The A-Kripke–Kleene fixpoint of O can be constructed as the limit of any monotone induction of A. For the A-well-founded fixpoint, a similar constructive characterization has been worked out by Denecker and Vennekens [23]:

Definition 4.1. An *A-refinement* of (x,y) is a pair $(x',y') \in L^2$ satisfying one of the following two conditions:

(i) $(x,y) \leq_p (x',y') \leq_p A(x,y)$, or
(ii) $x' = x$ and $A(x,y')_2 \leq y' \leq y$.

An A-refinement is *strict* if $(x,y) \neq (x',y')$.

We call the first type (**i**) of refinements *application refinements* and the second type (**ii**) *unfoundedness refinements*. If (x',y') is an A-refinement of (x,y) and A is clear from the context, we often write $(x,y) \to (x',y')$.

Definition 4.2. A *well-founded induction* of A is a sequence $(x_i,y_i)_{i \leq n}$ with $n \in \mathbb{N}$ such that

- $(x_0,y_0) = (\bot,\top)$;
- (x_{i+1},y_{i+1}) is an A-refinement of (x_i,y_i), for all $i < n$.

A well-founded induction is *terminal* if its limit (x_n,y_n) has no strict A-refinements.

A well-founded induction is an algebraical generalization of the well-founded model construction defined by Van Gelder et al. [40]. The first type of refinement corresponds to making a partial structure more precise by applying Fitting's immediate consequence operator; the second type of refinement corresponds to making a structure more precise by eliminating an unfounded set. For a given approximator A, there are many different terminal well-founded inductions of A. Denecker and Vennekens [23] showed that they all have the same limit, which equals the A-well-founded fixpoint of O. Furthermore, if A is symmetric, then the A-well-founded fixpoint of O (in fact, every tuple in a well-founded induction of A) is consistent.

4.3. Logic programming and AFT

A (normal[2]) logic program $\mathcal{P}$ is a set of *rules* r of the form

$$h \leftarrow l_1 \wedge \cdots \wedge l_n \tag{2}$$

where h is an atom called the *head* of r, denoted $head(r)$, and each of the l_i is a literal; $l_1 \wedge \cdots \wedge l_n$ is called the *body* of r and denoted $body(r)$. A rule of the form (2) is called *simple* if none of the l_i equals $\neg h$ or h; a logic program is *simple* if it consists of only simple rules. The set of interpretations 2^{At} forms a lattice equipped with the order $\subseteq$. The truth value (**t** or **f**) of a propositional formula φ in a structure I, denoted φ^I is defined as usual. With a logic program $\mathcal{P}$, we associate an immediate consequence operator [39] $T_\mathcal{P}$ that maps a structure I to

$$T_\mathcal{P}(I) = \{p \mid \exists r \in \mathcal{P} : head(r) = p \wedge body(r)^I = \mathbf{t}\}.$$

Bogaerts et al. [7] called grounded fixpoints of $T_\mathcal{P}$ *grounded models of* $\mathcal{P}$.

[2] Sometimes, logic programs are defined with disjunctive rules. In the current paper, we only consider *normal* logic programs: logic programs where the head is a single atom. In some propositions, we explicate the fact our programs are normal to emphasize that these results are no longer guaranteed to hold in the more general case.

$A \wedge B$		B		
		t	**f**	**u**
A	**t**	**t**	**f**	**u**
	f	**f**	**f**	**f**
	u	**u**	**f**	**u**

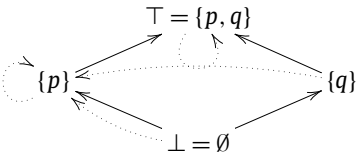
		$\neg A$
A	**t**	**f**
	f	**t**
	u	**u**

Fig. 2. The Kleene truth tables for conjunction and negation [29].

Example 4.3. Consider the following logic program $\mathcal{P}$:

$$\left\{ \begin{array}{l} p \\ q \leftarrow p \wedge q \end{array} \right\}$$

Its immediate consequence operator $T_{\mathcal{P}}$ is represented by the following graph:



$T_{\mathcal{P}}$ is a monotone operator with least fixpoint $\{p\}$. As such, $\{p\}$ is its only grounded fixpoint. To see that $\{p, q\}$ is not grounded, notice that, when taking $v = \{p\}$,

$$T_{\mathcal{P}}(v \cap \{p, q\}) = \{p\} \subseteq v,$$

but $\{p, q\} \not\subseteq v$. ▲

In the context of logic programming, elements of the approximation lattice $\left(2^{At}\right)^2$ are four-valued interpretations, pairs $\mathcal{I} = (I_1, I_2)$ of interpretations. The pair (I_1, I_2) approximates all interpretations I' with $I_1 \subseteq I' \subseteq I_2$. We often identify an interpretation I with the four-valued interpretation (I, I). We are mostly concerned with consistent (also called partial or three-valued) interpretations: tuples $\mathcal{I} = (I_1, I_2)$ with $I_1 \subseteq I_2$. For such an interpretation, the atoms in I_1 are *true* (**t**) in $\mathcal{I}$, the atoms in $I_2 \setminus I_1$ are *unknown* (**u**) in $\mathcal{I}$ and the other atoms are *false* (**f**) in $\mathcal{I}$. If $\mathcal{I}$ is a three-valued interpretation, and φ a formula, we write $\varphi^{\mathcal{I}}$ for the standard three-valued valuation based on the Kleene truth tables (see Fig. 2).

Several approximators have been defined for logic programs. The most common is Fitting's immediate consequence operator $\Psi_{\mathcal{P}}$ [26], a direct generalization of $T_{\mathcal{P}}$ to partial interpretations:

$$\Psi_{\mathcal{P}}(\mathcal{I})_1 = \{a \in \Sigma \mid body(r)^{\mathcal{I}} = \mathbf{t} \text{ for some rule } r \in \mathcal{P} \text{ with } head(r) = a\},$$
$$\Psi_{\mathcal{P}}(\mathcal{I})_2 = \{a \in \Sigma \mid body(r)^{\mathcal{I}} \neq \mathbf{f} \text{ for some rule } r \in \mathcal{P} \text{ with } head(r) = a\}.$$

Denecker et al. [20] showed that $\Psi_{\mathcal{P}}$ is an approximator of $T_{\mathcal{P}}$, that the well-founded fixpoint of $\Psi_{\mathcal{P}}$ is the well-founded model of $\mathcal{P}$ as defined by Van Gelder et al. and that $\Psi_{\mathcal{P}}$-stable fixpoints are exactly the stable models of $\mathcal{P}$ as defined by Gelfond and Lifschitz. In this case, the operator $\Psi_{\mathcal{P}}(\cdot, y)_1$ coincides with the immediate consequence operator of the Gelfond–Lifschitz reduct [28] of $\mathcal{P}$ with respect to the interpretation y.

Example 4.4. Consider the following logic program $\mathcal{P}$:

$$\left\{ \begin{array}{l} p \leftarrow \neg q \\ q \leftarrow \neg p \end{array} \right\}$$

It has two stable models, namely $\{p\}$ and $\{q\}$. Its well-founded model equals its Kripke–Kleene model and is $\mathcal{I} = (\emptyset, \{p, q\})$, i.e., the partial interpretation in which both p and q are unknown. To see that $\mathcal{I}$ is the Kripke–Kleene model, it suffices that it is the least element of the approximation lattice $\left(2^{At}\right)^2$ and that it is a fixpoint of $\Psi_{\mathcal{P}}$ (since in this interpretation, the value of all the bodies is **u**). To see that $\mathcal{I}$ is the well-founded model, we notice that there are no unfoundedness refinements of $\mathcal{I}$. Indeed, if for some $I \subseteq At$, it would hold that

$$\Psi_{\mathcal{P}}(\emptyset, I)_2 \leq I$$

then $q \in I$ (otherwise, p would be derived by the first rule) and $p \in I$ (for a symmetric argument). Hence, $I = \{p, q\}$ and there is no unfoundedness refinement.

To see that $\{p\}$ is a stable model, it suffices to note that $\Psi_{\mathcal{P}}(\emptyset, \{p\}) = (\{p\}, \{p\})$ and hence that $\{p\} = \operatorname{lfp} \Psi_{\mathcal{P}}(\cdot, \{p\})_1$ indeed. ▲

5. A semantic operator for AICs

In this section we show how a set of normal AICs induces an operator on a suitably defined lattice.
Given a fixed database DB, we are interested in the sets of update actions $\mathcal{U}$ such that:

(i) $\mathcal{U}$ is consistent and
(ii) each action in $\mathcal{U}$ modifies DB.

Note that the second condition here implies the first since it is not possible that both $+a$ and $-a$ modify DB. For each atom $a \in At$, we define

$$\mathrm{ch}\, a = \begin{cases} +a & \text{if } a \notin DB \\ -a & \text{otherwise.} \end{cases}$$

Let us furthermore denote the set of update actions that modify DB by $\mathcal{A}$. With this notation, $\mathcal{A} = \{\mathrm{ch}\, a \mid a \in At\}$ and the sets of update actions we are interested in are elements of $2^{\mathcal{A}}$. Note that ch and $\mathcal{A}$ are defined solely based on the initial database DB and are not dependent of, for instance, a given repair.

Following the principle of minimality of change [45,25], we also typically prefer smaller sets of updates over larger sets. Therefore, we are interested in the lattice $\langle 2^{\mathcal{A}}, \subseteq \rangle$, where smaller elements correspond to better repairs according to this principle.

The intuitive reading of an AIC r naturally suggests an operator over this lattice, defined as "if $\mathcal{U}(DB) \models \mathsf{body}(r)$, then add $\mathsf{head}(r)$ to $\mathcal{U}$" to obtain a new element of $2^{\mathcal{A}}$. However, this naive definition does not lead to an operator that is internal in $2^{\mathcal{A}}$, as illustrated for instance in the following example.

Example 5.1. Consider

$$\eta = \{\neg a \supset +a\}$$

and $DB = \{a\}$. In this case, $\mathcal{A} = \{-a\}$ and $2^{\mathcal{A}} = \{\emptyset, \{-a\}\}$. Now, taking $\mathcal{U} = \{-a\}$, the body of the only rule in η is satisfied. Naively adding its head to $\mathcal{U}$ results in the set $\{+a, -a\}$, which is not an element of $2^{\mathcal{A}}$. We expect a semantic operator to map $\mathcal{U}$ to $\emptyset$ since the rule in η indicates that the problems with $\mathcal{U}(DB)$ can be solved by adding a, i.e., by not changing DB at all. ▲

This kind of problems is inherent to the fact that AICs can have rules with dual heads and does not occur in other formalisms where AFT is applied, such as, e.g., logic programming[3]. Intuitively, the operator $\mathcal{T}_\eta$ we wish to define should satisfy the following properties:

- *(inertia)* Only change something in the input if there is a rule that warrants this change. This requirement consists itself of two parts:
  - Do not add anything to $\mathcal{U}$ unless there is a reason for it, i.e.,

$$\mathcal{T}_\eta(\mathcal{U}) \subseteq \mathcal{U} \cup \{\mathsf{head}(r) \mid r \in \eta \wedge \mathcal{U}(DB) \models \mathsf{body}(r)\}.$$

  - Do not remove anything from $\mathcal{U}$ unless there is a reason for it, i.e.,

$$\mathcal{U} \setminus \{\mathsf{head}(r)^D \mid r \in \eta \wedge \mathcal{U}(DB) \models \mathsf{body}(r)\} \subseteq \mathcal{T}_\eta(\mathcal{U}).$$

- *(cancellation)* If there is an action in $\mathcal{U}$ that is "canceled out" by some rule in η, then neither the action nor its dual are in the result (i.e., DB remains unchanged with respect to this action). Formally, if $\alpha \in \mathcal{U}$ and $\alpha^D \in \{\mathsf{head}(r) \mid r \in \eta \wedge \mathcal{U}(DB) \models \mathsf{body}(r)\}$, then $\alpha \notin \mathcal{T}_\eta(\mathcal{U}), \alpha^D \notin \mathcal{T}_\eta(\mathcal{U})$.
- *(completeness)* If there is an applicable rule whose body is satisfied, and whose head does not contradict $\mathcal{U}$, then the head is derived. Formally,

$$\{\mathsf{head}(r) \mid r \in \eta \wedge \mathcal{U}(DB) \models \mathsf{body}(r) \wedge \mathsf{head}(r)^D \notin \mathcal{U}\} \subseteq \mathcal{T}_\eta(\mathcal{U}).$$

It turns out that these three properties uniquely define an operator on $2^{\mathcal{A}}$. In order to give a constructive characterization of this operator, we introduce the following concept.

[3] Notice that for instance in default logic, rules with complementary literals in their head are allowed. However, such literals are treated as two independent statements and can be derived separately. In the current setting, an update action $+p$ can undo the effect of a previously derived action $-p$.

Definition 5.2. Let $\mathcal{U}_1$ and $\mathcal{U}_2$ be sets of update actions over a set of atoms At. The set $\mathcal{U}_1 \uplus \mathcal{U}_2$ is defined as

$$(\mathcal{U}_1 \uplus \mathcal{U}_2)\,(DB) = (\mathcal{U}_1 \cup \mathcal{U}_2) \setminus \{\alpha \mid \alpha, \alpha^D \in \mathcal{U}_1 \cup \mathcal{U}_2\}\,.$$

This operation models sequential composition of repairs in the following sense: given a database DB, if every action in $\mathcal{U}_1$ changes DB and every action in $\mathcal{U}_2$ changes $\mathcal{U}_1(DB)$, then $(\mathcal{U}_1 \uplus \mathcal{U}_2)(DB) = \mathcal{U}_2(\mathcal{U}_1(DB))$.

Observe that, while a set of AICs may include both a rule with head $+a$ and another with head $-a$, these rules are not simultaneously applicable, as the conjunction of their bodies is always unsatisfiable. However, it may be the case that applying one of them makes the other applicable (undoing the effect of the first one). In this case, the operation defined above guarantees that the database always reflects the *last* actions that were executed.

We note also that $\mathcal{U}_1 \uplus \mathcal{U}_2$ is consistent whenever $\mathcal{U}_1$ and $\mathcal{U}_2$ are.

Definition 5.3. Let DB be a database and η be a set of AICs over DB. The operator $\mathcal{T}_\eta^{DB} : 2^{\mathcal{A}} \to 2^{\mathcal{A}}$ is defined as follows:

$$\mathcal{T}_\eta^{DB}(\mathcal{U}) = \mathcal{U} \uplus \{\mathsf{head}(r) \mid r \in \eta \wedge \mathcal{U}(DB) \models \mathsf{body}(r)\}$$

In other words, $\mathcal{T}_\eta^{DB}(\mathcal{U})$ is obtained by updating $\mathcal{U}$ with the heads of all AICs whose bodies are satisfied by $\mathcal{U}(DB)$. To see that this operator is well-defined (i.e., that the result of applying it indeed yields a consistent set of actions), observe that the syntactic restrictions on AICs guarantee that the set $\{\mathsf{head}(r) \mid r \in \eta \wedge \mathcal{U}(DB) \models \mathsf{body}(r)\}$ is always consistent. Indeed, if both $+a$ and $-a$ are in this set, then there must be rules r_1 and r_2 such that $\neg a \in \mathsf{body}(r_1)$ and $a \in \mathsf{body}(r_2)$ with $\mathcal{U}(DB) \models \mathsf{body}(r_i)$ for $i = 1, 2$, which is impossible since it would mean that $\mathcal{U}(DB) \models a$ and $\mathcal{U}(DB) \models \neg a$. From this, it also follows that $\mathcal{T}_\eta(\mathcal{U})$ is always consistent. As before, when DB is clear from the context, we simply write $\mathcal{T}_\eta$ for $\mathcal{T}_\eta^{DB}$.

Proposition 5.4. *The operator $\mathcal{T}_\eta$ is the only operator on $2^{\mathcal{A}}$ that satisfies inertia, cancellation and completeness.*

Proof. It is easy to verify that $\mathcal{T}_\eta$ satisfies inertia, cancellation and completeness.

Now, assume $O : 2^{\mathcal{A}} \to 2^{\mathcal{A}}$ satisfies the three properties. Let α be any action in $\mathcal{A}$ and $\mathcal{U} \subseteq \mathcal{A}$. Let $\mathcal{V}$ denote $\{\mathsf{head}(r) \mid r \in \eta \wedge \mathcal{U}(DB) \models \mathsf{body}(r)\}$. Since, $\alpha \in \mathcal{A}$, we know that $\alpha^D \notin \mathcal{U}$. We show that $\alpha \in O(\mathcal{U})$ if and only if $\alpha \in \mathcal{T}_\eta(\mathcal{U}) = \mathcal{U} \uplus \mathcal{V}$. We distinguish 3 cases:

- If $\alpha \in \mathcal{V}$, then, since $\alpha^D \notin \mathcal{U}$, by completeness, $\alpha \in O(\mathcal{U})$. In this case, since $\alpha^D \notin \mathcal{U} \cup \mathcal{V}$, but $\alpha \in \mathcal{U} \cup \mathcal{V}$, it holds that $\alpha \in \mathcal{U} \uplus \mathcal{V} = \mathcal{T}_\eta(\mathcal{U})$.
- If $\alpha^D \in \mathcal{V}$, then there is a rule $r \in \eta$ such that $\alpha^D \in \mathsf{head}(r)$ and $\mathcal{U}(DB) \models \mathsf{body}(r)$. Since $\mathcal{U}(DB) \models \mathsf{body}(r)$, $\mathsf{lit}(\alpha)$ holds in $\mathcal{U}(DB)$. Since α changes DB, we know that $\mathsf{lit}(\alpha)$ does not hold in DB, hence it must be the case that $\alpha \in \mathcal{U}$. By cancellation, we then find that $\alpha \notin O(DB)$. Since $\alpha \in \mathcal{U}$ but $\alpha^D \in \mathcal{V}$, also $\alpha \notin \mathcal{U} \uplus \mathcal{V} = \mathcal{T}_\eta(\mathcal{U})$.
- If neither $\alpha \in \mathcal{V}$, nor $\alpha^D \in \mathcal{V}$, then by inertia, it must hold that $\alpha \in O(\mathcal{U})$ if and only if $\alpha \in \mathcal{U}$. In this case, it also holds that $\alpha \in \mathcal{T}_\eta(\mathcal{U}) = \mathcal{U} \uplus \mathcal{V}$ if and only if $\alpha \in \mathcal{U}$.

Hence, we have proven in each of the cases that $\alpha \in O(\mathcal{U})$ if and only if $\alpha \in \mathcal{T}_\eta(\mathcal{U})$ and the result follows. □

Example 5.5 *(Example 2.3 continued).* Consider again the set of AICs η from Example 2.3, where $DB = \{a, b\}$. Then $\mathcal{T}_\eta(\emptyset) = \{-a, -b\}$. Indeed, the bodies of all rules are satisfied; hence all heads are elements of $\mathcal{T}_\eta(\emptyset)$. ▲

Proposition 5.6. *Let DB be a database, η be a set of normal AICs over DB and $\mathcal{U}$ be a set of update actions. Then $\mathcal{U}$ is a weak repair for $\langle DB, \eta\rangle$ if and only if $\mathcal{U}$ is a fixpoint of $\mathcal{T}_\eta$.*

Proof. If $\mathcal{U}$ is a weak repair for $\langle DB, \eta\rangle$, then certainly, $\mathcal{U} \in 2^{\mathcal{A}}$. Also, $\mathcal{U}(DB) \not\models \mathsf{body}(r)$ for all $r \in \eta$, whence $\mathcal{T}_\eta(\mathcal{U}) = \mathcal{U}$. If $\mathcal{U}$ is not a weak repair for $\langle DB, \eta\rangle$, then $\mathcal{U}(DB) \models \mathsf{body}(r)$ for some $r \in \eta$, and $\mathcal{T}_\eta(\mathcal{U})$ differs from $\mathcal{U}$ by (at least) $\mathsf{head}(r)$. □

Example 5.7. In general, $\mathcal{T}_\eta$ does not need to have fixpoints (since there may be no database satisfying η). A simple (unrealistic) example is if η consists of the two rules

$$a \supset -a \qquad \text{and} \qquad \neg a \supset +a$$

where $\mathcal{T}_\eta(DB) \neq DB$ for any database DB. ▲

Proposition 5.8. *Let DB be a database, η be a set of normal AICs over DB and $\mathcal{U}$ be a set of update actions. Then $\mathcal{U}$ is a repair for $\langle DB, \eta\rangle$ if and only if $\mathcal{U}$ is a minimal fixpoint of $\mathcal{T}_\eta$.*

Proof. Follows directly from Proposition 5.6 and the definition of repair. □

Proposition 5.9. *Let DB be a database, η be a set of normal AICs over DB and $\mathcal{U} \in 2^{\mathcal{A}}$. Then $\mathcal{U}$ is founded with respect to $\langle DB, \eta\rangle$ if and only if, for all $\alpha \in \mathcal{U}$, it is the case that $\alpha \in \mathcal{T}_\eta(\mathcal{U} \setminus \{\alpha\})$.*

Proof. It follows from the definition of $\mathcal{T}_\eta$ that for each action $\alpha \in \mathcal{U}$, the following are equivalent:

(i) there is a rule $r \in \eta$ such that $(\mathcal{U} \setminus \{\alpha\})(DB) \models \mathsf{body}(r)$ and
(ii) $\alpha \in \mathcal{T}_\eta(\mathcal{U} \setminus \{\alpha\})$.

Now, $\mathcal{U}$ is founded if and only if **(i)** holds for all actions $\alpha \in \mathcal{U}$; this is indeed equivalent with the condition from this proposition, which is that **(ii)** holds for each $\alpha \in \mathcal{U}$. □

This result also gives some intuition regarding why founded repairs allow for circular dependencies: the definition of founded repair only checks that each individual action is supported by the remaining ones, but it still allows for dependency cycles.

Proposition 5.10. *Let DB be a database, η be a set of normal AICs over DB and $\mathcal{U}$ be a weak repair for $\langle DB, \eta\rangle$. Then $\mathcal{U}$ is well-founded if and only if there is an ordering $\alpha_1, \ldots, \alpha_n$ of the elements of $\mathcal{U}$ such that $\alpha_i \in \mathcal{T}_\eta(\{\alpha_1, \ldots, \alpha_{i-1}\})$ for each $i = 1, \ldots, n$.*

Proof. This is a direct consequence of the definitions of $\mathcal{T}_\eta$ and of well-founded repair. By definition, $\mathcal{U}$ is well-founded if and only if there exists a ordering $\alpha_1, \ldots, \alpha_n$ of the elements of $\mathcal{U}$ such that for each i, there is a rule $r_i \in \eta$ with $U_{i-1}(DB) \models \mathsf{body}(r_i)$ and $\alpha_i \in \mathsf{head}(r_i)$ (where U_i denotes $\{\alpha_1, \ldots, \alpha_i\}$). Now, the condition that there is a rule $r_i \in \eta$ with $U_{i-1}(DB) \models \mathsf{body}(r_i)$ and $\alpha_i \in \mathsf{head}(r_i)$ is equivalent with the condition that $\alpha_i \in \mathcal{T}_\eta(\mathcal{U}_{i-1})$, from which the result follows. □

Proposition 5.11. *Let DB be a database and η a set of AICs over DB. A set of update actions $\mathcal{U}$ is a grounded repair of $\langle DB, \eta\rangle$ if and only if $\mathcal{U}$ is a grounded fixpoint of $\mathcal{T}_\eta$.*

Proof. Recall that grounded and strictly grounded fixpoints of $\mathcal{T}_\eta$ coincide. We show that $\mathcal{U}$ is a grounded repair of $\langle DB, \eta\rangle$ iff $\mathcal{U}$ is a strictly grounded fixpoint of $\mathcal{T}_\eta$.

First suppose that $\mathcal{U}$ is not a strictly grounded fixpoint of $\mathcal{T}_\eta$. This means that there exists a set $\mathcal{V} \subsetneq \mathcal{U}$ such that $\mathcal{T}_\eta(\mathcal{V}) \cap \mathcal{U} \subseteq \mathcal{V}$. From the definition of $\mathcal{T}_\eta$ it follows immediately that, if r is a rule such that $\mathcal{V}(DB) \models \mathsf{body}(r)$, then $\mathsf{head}(r) \notin (\mathcal{U} \setminus \mathcal{V})$, whence $\mathcal{U}$ is not a grounded repair for $\langle DB, \eta\rangle$.

Conversely, assume that $\mathcal{U}$ is a strictly grounded fixpoint of $\mathcal{T}_\eta$ and let $\mathcal{V}$ be any subset of $\mathcal{U}$ such that there is no rule r for which $\mathcal{V}(DB) \models \mathsf{body}(r)$ and $\mathsf{head}(r) \in (\mathcal{U} \setminus \mathcal{V})$. From the definition of $\mathcal{T}_\eta$, it follows that $\mathcal{T}_\eta(\mathcal{V}) \cap (\mathcal{U} \setminus \mathcal{V}) = \emptyset$ and thus that $\mathcal{T}_\eta(\mathcal{V}) \cap \mathcal{U} \subseteq \mathcal{V}$. Since $\mathcal{U}$ is strictly grounded, it follows that $\mathcal{V} = \mathcal{U}$. □

The previous proposition illustrates that Proposition 3.7 is not a coincidence at all. Indeed, Bogaerts et al. [7] have already shown that all stable fixpoints of a given approximator are grounded, and Caroprese and Truszczyński [13, Theorem 6] showed that justified repairs are stable models of a given derived logic program. In the following sections, we explore this relationship further: first, we define an approximator for $\mathcal{T}_\eta$ and as such obtain also a notion of stable repair. Next, in Section 7, we study the relationship between logic programs and AICs in depth.

Since grounded repairs can be built from the ground up, this result also corroborates the informal claim that justified repairs avoid circularity of support, as stated by Caroprese and Truszczyński [13].

6. An approximator for AICs

In this section, we define an approximator for $\mathcal{T}_\eta$ and hence, obtain a set of AFT-based semantics for AICs, based on intuitions similar to those underlying groundedness and various semantics from non-monotonic reasoning.

A *partial action set* is a tuple $\mathfrak{U} = (\mathfrak{U}_c, \mathfrak{U}_p)$ where $\mathfrak{U}_c \in 2^{\mathcal{A}}$ and $\mathfrak{U}_p \in 2^{\mathcal{A}}$. A partial action set is an approximation of a set of update actions. It provides information on which update actions are *certainly* applied (the actions in $\mathfrak{U}_c$) and which actions are *possibly* applied (the actions in $\mathfrak{U}_p$). If $\mathfrak{U}_c \subseteq \mathfrak{U}_p$, we say that $\mathfrak{U}$ is *consistent*. We are mostly concerned with consistent partial action sets. If $\alpha \in \mathcal{A}$, the *value* of α in $\mathfrak{U}$ (denoted $\mathfrak{U}(\alpha)$) is:

- true, denoted **t**, if $\alpha \in \mathfrak{U}_c$ and $\alpha \in \mathfrak{U}_p$,
- false, denoted **f**, if $\alpha \notin \mathfrak{U}_c$ and $\alpha \notin \mathfrak{U}_p$,
- unknown, denoted **u**, if $\alpha \notin \mathfrak{U}_c$ and $\alpha \in \mathfrak{U}_p$,
- inconsistent, denoted **i**, if $\alpha \in \mathfrak{U}_c$ and $\alpha \notin \mathfrak{U}_p$.

Note that a partial action set is characterized completely by the mapping $\mathcal{A} \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}, \mathbf{i}\}$ it induces:

$$\mathfrak{U}_c = \{\alpha \in \mathcal{A} \mid \mathfrak{U}(\alpha) \in \{\mathbf{t}, \mathbf{i}\}\},$$

$$\mathfrak{U}_p = \{\alpha \in \mathcal{A} \mid \mathfrak{U}(\alpha) \in \{\mathbf{t}, \mathbf{u}\}\}.$$

We sometimes exploit this correspondence, by not explicating the partial action set itself, but the mapping, as this allows for more compact notation and more elegant definitions. It can be seen that $\mathfrak{U}$ is consistent if and only if $\mathfrak{U}(\alpha) \in \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ for all α. The intended interpretation of a consistent partial action set is thus that $\mathfrak{U}(a)$ is true iff a is certainly changed by $\mathfrak{U}$, it is false iff a is certainly not changed by $\mathfrak{U}$ and it is unknown if $\mathfrak{U}$ leaves it open whether or not a is changed. The set of all consistent partial action sets is denoted as P. As such, P is exactly the consistent part of the bilattice $(2^{\mathcal{A}})^2$. We call $\mathfrak{U}$ *two-valued* if $\mathfrak{U}_c = \mathfrak{U}_p$, i.e., if $\mathfrak{U}(\alpha) \in \{\mathbf{t}, \mathbf{f}\}$ for all α; in this case, we identify $\mathfrak{U}$ with the action set $\mathfrak{U}_c$. The truth order $\leq_t$ on truth values is defined by $\mathbf{f} \leq_t \mathbf{u} \leq_t \mathbf{t}$; $\mathbf{f} \leq_t \mathbf{i} \leq_t \mathbf{t}$. The inverse of a truth value is $\mathbf{f}^{-1} = \mathbf{t}, \mathbf{t}^{-1} = \mathbf{f}, \mathbf{u}^{-1} = \mathbf{u}, \mathbf{i}^{-1} = \mathbf{i}$.

A (consistent) partial database is a mapping $\mathfrak{DB} : At \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$. The intended reading is that $\mathfrak{DB}(a)$ is true if a is certainly in the database, $\mathfrak{DB}(a)$ is false if a is certainly not in the database and $\mathfrak{DB}(a)$ is unknown otherwise.

If $\mathfrak{U} \in P$ is a consistent partial action set and DB is a (regular) database, then we define $\mathfrak{U}(DB)$ to be the partial database such that

$$\mathfrak{U}(DB) : a \mapsto \begin{cases} DB(a) & \text{if } \mathfrak{U}(a) = \mathbf{f} \\ DB(a)^{-1} & \text{if } \mathfrak{U}(a) = \mathbf{t} \\ \mathbf{u} & \text{otherwise,} \end{cases}$$

where $DB(a) = \mathbf{t}$ if $a \in DB$ and $DB(a) = \mathbf{f}$ otherwise.

Definition 6.1. Given a partial database $\mathfrak{DB}$, a set of AICs η and an update action α, we define the *support* of α with respect to $\langle \mathfrak{DB}, \eta \rangle$ as

$$\mathsf{supp}_{\mathfrak{DB},\eta}(\alpha) = \max_{\leq_t} \{\mathsf{nup}(r)^{\mathfrak{DB}} \mid r \in \eta \wedge \mathsf{head}(r) = \alpha\},$$

where $\mathsf{nup}(r)^{\mathfrak{DB}}$ refers to the standard three-valued truth evaluation of the formula[4] $\mathsf{nup}(r)$ in the partial interpretation $\mathfrak{DB}$ based on Kleene's truth tables [29] (see Fig. 2).

Intuitively, this means that the support of an action α is the highest truth value of the (non-updateable part of the) body of a rule in η with α in the head.

Example 6.2. Consider $DB = \emptyset$ and the following set η:

$$\neg a \wedge b \supset +a$$
$$a \wedge c \wedge d \supset -a\,.$$

Consider $\mathfrak{U} = (\{+a, +b, +c\}, \{+a, +b, +c, +d\})$. Then $\mathfrak{U}(DB) = \{a \mapsto \mathbf{t}, b \mapsto \mathbf{t}, c \mapsto \mathbf{t}, d \mapsto \mathbf{u}\}$. In this case, $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(+a) = \mathbf{t}$ and $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(-a) = \mathbf{u}$. ▲

Definition 6.3. Given DB and η, we define an operator $\mathfrak{T}_{\langle DB,\eta\rangle} : P \to P$, such that for each $\mathfrak{U} \in P$ and each $\alpha \in \mathcal{A}$:

- If $\mathfrak{U}(\alpha) = \mathbf{f}$, then $\mathfrak{T}_{\langle DB,\eta\rangle}(\mathfrak{U})(\alpha) = \mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha)$.
- If $\mathfrak{U}(\alpha) = \mathbf{t}$, then $\mathfrak{T}_{\langle DB,\eta\rangle}(\mathfrak{U})(\alpha) = \mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha^D)^{-1}$.
- Otherwise (i.e., if $\mathfrak{U}(\alpha) = \mathbf{u}$):
  - if $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha) = \mathbf{t}$ and $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha^D) = \mathbf{f}$, then $\mathfrak{T}_{\langle DB,\eta\rangle}(\mathfrak{U})(\alpha) = \mathbf{t}$;
  - if $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha^D) = \mathbf{t}$ and $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha) = \mathbf{f}$, then $\mathfrak{T}_{\langle DB,\eta\rangle}(\mathfrak{U})(\alpha) = \mathbf{f}$;
  - otherwise, $\mathfrak{T}_{\langle DB,\eta\rangle}(\mathfrak{U})(\alpha) = \mathbf{u}$.

When DB is clear from the context, we write $\mathfrak{T}_\eta$ for $\mathfrak{T}_{\langle DB,\eta\rangle}$.

Definition 6.3 is motivated as follows. Assume $\mathfrak{U}$ is a partial update set containing some information on the intended update. In this case $\mathfrak{T}_\eta(\mathfrak{U})$ represents a revised update, using the AICs in η. In the case where $\mathfrak{U}(\alpha) = \mathbf{f}$, α is not an element of the (partial) update set at hand. The only way to add α to the update is if some rule supports α, which is captured by $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha)$. The case for $\mathfrak{U}(\alpha) = \mathbf{t}$ is completely symmetrical, in this case the only reason for removing α from the update at hand is there is support for its dual. In the last case, where $\mathfrak{U}(\alpha) = \mathbf{u}$, we have no information on whether α should or shouldn't be in the update yet. In this case, we can derive that α *must* be in the update if we already have support for α and we are sure that there is no support for it dual (for α^D). Similarly, we can derive that α *must not* be in the update if we have support for α^D and certainly not for α. In all other cases, we derive nothing about α being in the update or not.

[4] Technically, $\mathsf{nup}(r)$ is a set of literals; we identify it with the conjunction of those literals.

Since $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha)$ and $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha^D)^{-1}$ are consistent whenever $\mathfrak{U}$ is consistent, the above operator is indeed defined on P, i.e., $\mathfrak{T}_{\langle DB,\eta\rangle}(\mathfrak{U})$ is consistent whenever $\mathfrak{U}$ is.

Recall that Denecker et al. [22] showed that it suffices to define approximator on the consistent part of the bilattice. Using the fact that $P = (2^{\mathcal{A}})^c$, $\mathfrak{T}_\eta$ is a candidate approximator of $\mathcal{T}_\eta$. The next proposition shows that this is indeed the case.

Proposition 6.4. *$\mathfrak{T}_\eta$ is an approximator of $\mathcal{T}_\eta$.*

Proof. First, we show that $\mathfrak{T}_\eta$ is $\leq_p$-monotone. To see this, first note that Kleene-valuation is $\leq_p$-monotone. Hence, for each $\alpha \in \mathcal{A}$, also the functions that map $\mathfrak{U}$ to $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha)$ and $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha^D)$ are $\leq_p$-monotone. Now take some $\alpha \in \mathcal{A}$ and suppose $\mathfrak{U}' \geq_p \mathfrak{U}$. Then $\mathsf{supp}_{\mathfrak{U}'(DB),\eta}(\alpha) \geq_p \mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha)$ and $\mathsf{supp}_{\mathfrak{U}'(DB),\eta}(\alpha^D) \geq_p \mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha^D)$. We show that $\mathfrak{T}_\eta(\mathfrak{U}')(\alpha) \geq_p \mathfrak{T}_\eta(\mathfrak{U})(\alpha)$ by a case analysis on the definition of $\mathfrak{T}_\eta(\mathfrak{U})(\alpha)$.

- If $\mathfrak{U}(\alpha) = \mathbf{f}$, then $\mathfrak{U}'(\alpha) = \mathbf{f}$ since $\mathfrak{U}' \geq_p \mathfrak{U}$. In this case $\mathfrak{T}_\eta(\mathfrak{U}')(\alpha) = \mathsf{supp}_{\mathfrak{U}'(DB),\eta}(\alpha) \geq_p \mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha) = \mathfrak{T}_\eta(\mathfrak{U})(\alpha)$.
- The case where $\mathfrak{U}(\alpha) = \mathbf{t}$ is similar.
- Assume $\mathfrak{U}(\alpha) = \mathbf{u}$.
  - If $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha) = \mathbf{t}$ and $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha^D) = \mathbf{f}$, then $\mathfrak{T}_\eta(\mathfrak{U})(\alpha) = \mathbf{t}$ and also $\mathsf{supp}_{\mathfrak{U}'(DB),\eta}(\alpha) = \mathbf{t}$ and $\mathsf{supp}_{\mathfrak{U}'(DB),\eta}(\alpha) = \mathbf{f}$. Hence if $\mathfrak{U}'(\alpha) = \mathbf{u}$, this is trivially proven. In case $\mathfrak{U}'(\alpha) = \mathbf{t}$, $\mathfrak{T}_\eta(\mathfrak{U}')(\alpha) = \mathsf{supp}_{\mathfrak{U}'(DB),\eta}(\alpha^D)^{-1} = \mathbf{t}$ and in case $\mathfrak{U}'(\alpha) = \mathbf{f}$, $\mathfrak{T}_\eta(\mathfrak{U}')(a) = \mathsf{supp}_{\mathfrak{U}'(DB),\eta}(\alpha) = \mathbf{t}$. Hence, in all cases $\mathfrak{T}_\eta(\mathfrak{U})(\alpha) = \mathfrak{T}_\eta(\mathfrak{U}')(\alpha)$ and the claim follows.
  - The case where $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha^D) = \mathbf{t}$ and $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha) = \mathbf{f}$ is similar to the previous.
  - In all other cases, $\mathfrak{T}_\eta(\mathfrak{U})(\alpha) = \mathbf{u}$, hence $\mathfrak{T}_\eta(\mathfrak{U}')(\alpha) \geq_p \mathfrak{T}_\eta(\mathfrak{U})(\alpha)$ is trivially satisfied.

Secondly, we show that on two-valued update sets, $\mathcal{T}_\eta$ and $\mathfrak{T}_\eta$ coincide, i.e., that for all $\mathcal{U}$, $\mathcal{T}_\eta(\mathcal{U}) = \mathfrak{T}_\eta(\mathcal{U})$. Take any $\alpha \in \mathcal{A}$, we again prove this claim by a case analysis on the definition of $\mathfrak{T}_\eta(\mathcal{U})(\alpha)$.

- If $\mathcal{U}(\alpha) = \mathbf{f}$, then $\mathcal{T}_\eta(\mathfrak{U})(\alpha) = \mathbf{t}$ if and only if there is some rule $r \in \eta$ with $\mathcal{U}(DB) \models \mathsf{body}(r)$ and $\mathsf{head}(r) = \alpha$. Since $\mathsf{body}(r) = \mathsf{nup}(r) \wedge \mathsf{lit}(\alpha^D)$, we conclude that $\mathcal{T}_\eta(\mathcal{U})(\alpha)$ is true if and only if $\mathsf{supp}_{\mathcal{U}(DB),\eta}(\alpha) = \mathbf{t}$, if and only if $\mathfrak{T}_\eta(\mathcal{U})(\alpha) = \mathbf{t}$.
- The case for $\mathcal{U}(\alpha) = \mathbf{f}$ is similar.
- The case where $\mathcal{U}(\alpha) = \mathbf{u}$ cannot occur, since $\mathcal{U}$ is two-valued. □

Since $\mathfrak{T}_\eta$ is an approximator, it defines a family of semantics for AICs.

Definition 6.5. Let $\langle DB, \eta\rangle$ be a database.

- A *partial stable repair* of $\langle DB, \eta\rangle$ is a partial update set $\mathfrak{U}$ such that $\mathfrak{U}$ is a partial $\mathfrak{T}_\eta$-stable fixpoint. A *stable repair* is a partial stable repair that is two-valued.
- The *AFT-well-founded repair* of $\langle DB, \eta\rangle$ is the $\mathfrak{T}_\eta$-well-founded fixpoint (in general, this is a partial update set).
- The *Kripke–Kleene repair* of $\langle DB, \eta\rangle$ is the $\mathfrak{T}_\eta$-Kripke–Kleene fixpoint (in general, this is a partial update set).
- A *partial grounded repair* of $\langle DB, \eta\rangle$ is a partial update set $\mathfrak{U}$ such that $\mathfrak{U}$ is a partial $\mathfrak{T}_\eta$-grounded fixpoint Bogaerts et al. [8]. A *grounded repair* is a partial grounded repair that is two-valued.

The terminology in the above definition uses "repairs" for certain classes of fixpoints of a semantic operator. It follows easily that all two-valued update sets that are called "repair" in the above definition, indeed are repairs according to AIC terminology. This paper is the first work that studies partial (non-two-valued) repairs.

The well-founded semantics induced by AFT can in general differ from the existing well-founded semantics for AICs, as we show in Example 6.12. To distinguish the two, we use the term *AFT-well-founded semantics*.

It follows directly from Bogaerts et al. [8, Proposition 3.2] and Proposition 5.11 that grounded repairs as defined in Definition 6.5 coincide with Definition 3.1. All other classes of repairs are new.

We now illustrate these semantics by means of some examples.

Example 6.6. Consider the following set η of AICs:

$$\begin{aligned} \neg a &\supset +a \\ \neg a \wedge \neg b \wedge \neg c &\supset +c \\ a \wedge \neg b &\supset +b \\ a \wedge c \wedge b &\supset -b \end{aligned}$$

with $DB = \emptyset$. Now, the $\mathfrak{T}_\eta$-well-founded fixpoint can be computed as the limit of a well-founded induction. It starts at $\mathfrak{U}_0$ that maps

$$+a \mapsto \mathbf{u}, +b \mapsto \mathbf{u}, +c \mapsto \mathbf{u},$$

i.e., $\mathfrak{U}_0$ is[5] the partial action

$$(\emptyset, \{+a, +b, +c\}).$$

Here, we see that

$$\begin{aligned} &\mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(+a) = \mathbf{t}, && \mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(-a) = \mathbf{f} \\ &\mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(+b) = \mathbf{u}, && \mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(-b) = \mathbf{u} \\ &\mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(+c) = \mathbf{u}, && \mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(-c) = \mathbf{f} \end{aligned}$$

Hence,

$$\mathfrak{U}_1 = \mathfrak{T}_\eta(\mathfrak{U}_0) : +a \mapsto \mathbf{t}, +b \mapsto \mathbf{u}, +c \mapsto \mathbf{u}$$

is a refinement of $\mathfrak{U}_0$. Now, it can be verified that $\mathfrak{U}_1$ is a fixpoint of $\mathfrak{T}_\eta$. This is the Kripke–Kleene fixpoint. It is a partial repair set and provides the information that a must be added, but it is uncertain about b and c.

A well-founded induction continues by unfoundedness refinement. The partial update set

$$\mathfrak{U}_2 = +a \mapsto \mathbf{t}, +b \mapsto \mathbf{u}, +c \mapsto \mathbf{f}$$

is an unfoundedness refinement of $\mathfrak{U}_1$. This follows easily from the fact that

$$\mathfrak{U}_3 = \mathfrak{T}_\eta(\mathfrak{U}_2) = +a \mapsto \mathbf{t}, +b \mapsto \mathbf{t}, +c \mapsto \mathbf{f}.$$

Furthermore, $\mathfrak{U}_3$ is an application refinement of $\mathfrak{U}_2$. Since this is an exact point, it is the $\mathfrak{T}_\eta$-well-founded fixpoint of $\mathcal{T}_\eta$. It is clearly the intended repair in this example.

Note that in this example, unfoundedness refinements take care of minimizations of repairs. ▲

Example 6.7. Consider the following set η of AICs:

$$\begin{aligned} a \wedge \neg b &\supset +b \\ \neg a \wedge b &\supset +a \\ \neg a \wedge \neg b \wedge \neg c &\supset +c \end{aligned}$$

with $DB = \emptyset$. Intuitively, we expect $+c$ to be an element of "good" repairs, and (following the minimality of change principle), no other actions to be in "good" repairs.

Now, the $\mathfrak{T}_\eta$-well-founded fixpoint can be computed as the limit of a well-founded induction. It starts at

$$\mathfrak{U}_0 : +a \mapsto \mathbf{u}, +b \mapsto \mathbf{u}, +c \mapsto \mathbf{u}.$$

Here, we see that

$$\begin{aligned} &\mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(+a) = \mathbf{u}, && \mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(-a) = \mathbf{f} \\ &\mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(+b) = \mathbf{u}, && \mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(-b) = \mathbf{f} \\ &\mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(+c) = \mathbf{u}, && \mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(-c) = \mathbf{f} \end{aligned}$$

Hence, $\mathfrak{T}_\eta(\mathfrak{U}_0) = \mathfrak{U}_0$ and $\mathfrak{U}_0$ is the $\mathfrak{T}_\eta$-Kripke–Kleene fixpoint. A well-founded induction can continue with unfoundedness refinements. Indeed, consider

$$\mathfrak{U}_1 : +a \mapsto \mathbf{f}, +b \mapsto \mathbf{f}, +c \mapsto \mathbf{u}.$$

Since

$$\mathfrak{T}_\eta(\mathfrak{U}_1) = +a \mapsto \mathbf{f}, +b \mapsto \mathbf{f}, +c \mapsto \mathbf{t},$$

it holds that $\mathfrak{U}_1$ is an unfoundedness refinement of $\mathfrak{U}_0$. Finally, we can conclude that $\mathfrak{U}_2 := \mathfrak{T}_\eta(\mathfrak{U}_1)$ is the $\mathfrak{T}_\eta$-well-founded fixpoint. This corresponds to the intended repair. ▲

[5] Recall that for simplicity, we often characterize partial action sets by their induced truth function.

As can be expected, not every set of AICs has a two-valued well-founded repair. That would simply be too much to ask, as it would mean that for every set of AICs we can unambiguously identify a single repair. The following example illustrates that this is indeed not always the case. It also illustrates that, for this specific example, $\mathfrak{T}_\eta$-stable repairs provide a solution that corresponds to the intuitions.

Example 6.8. Consider the following set η of AICs:

$$\neg a \wedge \neg b \supset +a$$
$$\neg a \wedge \neg b \supset +b$$
$$a \wedge \neg c \supset +c$$

with $DB = \emptyset$. Intuitively, η has two "good" repairs. The first two rules state that a or b should be added in order to "fix" the violated constraint $\neg(\neg a \wedge \neg b)$. Depending on that choice, the last rule determines whether or not c should be repaired. The two intended repairs are thus $\{+a, +c\}$ and $\{+b\}$. Let us investigate what the different AFT-style semantics give in this case.

Consider

$$\mathfrak{U}_0 : +a \mapsto \mathbf{u}, +b \mapsto \mathbf{u}, +c \mapsto \mathbf{u}.$$

Here, it holds that

$$\mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(+a) = \mathbf{u}, \qquad \mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(-a) = \mathbf{f}$$
$$\mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(+b) = \mathbf{u}, \qquad \mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(-b) = \mathbf{f}$$
$$\mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(+c) = \mathbf{u}, \qquad \mathsf{supp}_{\mathfrak{U}_0(DB),\eta}(-c) = \mathbf{f}$$

Hence, $\mathfrak{T}_\eta(\mathfrak{U}_0) = \mathfrak{U}_0$ and $\mathfrak{U}_0$ is the $\mathfrak{T}_\eta$-Kripke–Kleene fixpoint. Furthermore, we claim that there are no unfoundedness refinements of $\mathfrak{U}_0$, and hence that $\mathfrak{U}_0$ is also the $\mathfrak{T}_\eta$-well-founded fixpoint. To see that our claim indeed holds, notice that any unfoundedness refinement of $\mathfrak{U}_0$ should consist of making a subset U of $\{+a, +b, +c\}$ false, in such a way that for each $\alpha \in U$, $\mathfrak{T}_\eta(\mathfrak{U}_0[U : \mathbf{f}])(\alpha) = \mathbf{f}$. Assume $+a \in U$. In order for $\mathfrak{T}_\eta(\mathfrak{U}_0[U : \mathbf{f}])(+a)$ to be false $+b$ must be $\mathbf{t}$ in $\mathfrak{U}_0[U : \mathbf{f}]$ (otherwise the body of the rule defining $+a$ is unknown or true). That is not possible, hence $+a \notin U$. From a similar argument, we find that $+b \notin U$ and $+c \notin U$. Trivially, the two intended repairs are more precise than the well-founded fixpoint.

Now, let us check whether $\mathcal{U} := \{+a, +c\}$ is a stable repair. For this, we need to verify if

$$\mathcal{U} = \mathrm{lfp}(\mathfrak{T}_\eta(\cdot, \mathcal{U})_1).$$

Define $\mathcal{U}_0 = \emptyset$. Then

$$(\mathcal{U}_0, \mathcal{U}) = +a \mapsto \mathbf{u}, +b \mapsto \mathbf{f}, +c \mapsto \mathbf{u}.$$

Hence

$$\mathfrak{T}_\eta(\mathcal{U}_0, \mathcal{U}) = +a \mapsto \mathbf{t}, +b \mapsto \mathbf{u}, +c \mapsto \mathbf{u}$$

and

$$\mathcal{U}_1 := \mathfrak{T}_\eta(\mathcal{U}_0, \mathcal{U})_1 = \{+a\}.$$

Similarly,

$$(\mathcal{U}_1, \mathcal{U}) = +a \mapsto \mathbf{t}, +b \mapsto \mathbf{f}, +c \mapsto \mathbf{u}.$$

Hence

$$\mathfrak{T}_\eta(\mathcal{U}_1, \mathcal{U}) = +a \mapsto \mathbf{t}, +b \mapsto \mathbf{f}, +c \mapsto \mathbf{t}$$

and

$$\mathcal{U}_2 := \mathfrak{T}_\eta(\mathcal{U}_1, \mathcal{U})_1 = \mathcal{U}.$$

Furthermore, $\mathfrak{T}_\eta(\mathcal{U}_2, \mathcal{U}) = \mathcal{U}_2 = \mathcal{U}$, hence we find that indeed, $\mathcal{U}$ is an $\mathfrak{T}_\eta$-stable fixpoint. The case for $\{+b\}$ is similar.

It can also be verified that there are no other $\mathfrak{T}_\eta$-stable fixpoints. Due to minimality, no other could contain $+b$, hence they must be subsets of $\{+a, +c\}$. But since $\mathcal{U}$ is a stable repair, it is already minimal, hence no strict subset of $\{+a, +c\}$ can be a stable repair. ▲

Hence, in the above example, $\mathfrak{T}_\eta$-stable repairs capture the intended semantics.

We omit examples of grounded fixpoints, as we already included some in Section 3.

Properties of AFT-style semantics We now show that all AFT-style semantics are nicely invariant under shifting.

Proposition 6.9. *$\mathfrak{T}_\eta$, and hence also $\mathcal{T}_\eta$, commutes with shifting, i.e., for each set $S \subseteq At$:*

$$\mathsf{shift}_S \circ \mathfrak{T}_{\langle DB,\eta\rangle} = \mathfrak{T}_{\langle \mathsf{shift}_S(DB),\mathsf{shift}_S(\eta)\rangle}$$

Proof. The clue to proving this proposition is the fact that for each action α,

$$\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha) = \mathsf{supp}_{\mathfrak{U}(\mathsf{shift}_S(DB)),\mathsf{shift}_S(\eta)}(\mathsf{shift}_S(\alpha)).$$

Then, the result easily follows from the fact that $\mathfrak{T}_\eta$ has been defined entirely based on the supp function. □

Corollary 6.10. *All AFT-style semantics for AICs have the shifting property.*

Proposition 6.11. *If the AFT-well-founded repair is two-valued, it is also well-founded (as defined by Cruz-Filipe et al. [17]).*

Proof. Let $(\mathfrak{U}_i)_{i\leq k}$ be a well-founded induction of $\mathfrak{T}_\eta$. For each i, define $\mathcal{U}_i$ as $\{\alpha \in \mathcal{A} \mid \mathfrak{U}_i(\alpha) = \mathbf{t}\}$. We will prove the following claim by induction on the length k of the well-founded induction (note that we restrict to finite well-founded inductions here, since we assume At to be finite).

> **Claim:** There exists a sequence $\alpha_1, \ldots, \alpha_n$ such that $\mathcal{U}_k = \{\alpha_1, \ldots, \alpha_n\}$ and, for each $i \in \{1, \ldots, n\}$, there is a rule r_i such that $\{\alpha_1, \ldots, \alpha_{i-1}\}(DB) \models \mathsf{body}(r_i)$ and $\alpha_i = \mathsf{head}(r_i)$.

From this claim, taking any terminal well-founded induction yields the desired result.

We now show the claim indeed holds. The claim is trivial for $k = 0$. Assume the claim holds for k, we show that it also holds for $k+1$. Thus assume $\alpha_1, \ldots, \alpha_n$ is a sequence with $\mathcal{U}_k = \{\alpha_1, \ldots, \alpha_n\}$ satisfying the above condition. If $\mathfrak{U}_{k+1}$ is an unfoundedness refinement of $\mathfrak{U}_k$, then $\mathcal{U}_{k+1} = \mathcal{U}_k$ and there is nothing to show. Hence, assume that $\mathfrak{U}_k \leq_p \mathfrak{U}_{k+1} \leq_p \mathfrak{T}_\eta(\mathfrak{U}_k)$. In this case, $\mathcal{U}_{k+1} = \mathcal{U}_k \cup \{\beta_j \mid 1 \leq j \leq m\}$ for some sequence of elements β_j such that $\mathfrak{T}_\eta(\mathfrak{U}_k)(\beta_j) = \mathbf{t}$. For each j, let $\mathcal{U}'_j$ denote $\{\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_j\}$. We claim that the sequence $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m$ still satisfies the condition in the claim. To see this, note that from the definition of $\mathfrak{T}_\eta$, it follows that for each $j \in \{1, \ldots, m\}$ there is a rule r_j with $\mathsf{nup}(r)^{\mathfrak{U}_k(DB)} = \mathbf{t}$ and $\mathsf{head}(r_j) = \beta_j$. Furthermore, for each j it holds that $\mathcal{U}'_{j-1} \geq_p \mathfrak{U}_k$, hence also $\mathsf{nup}(r)^{\mathcal{U}'_{j-1}} = \mathbf{t}$. Since $\beta_j \notin \mathcal{U}_{j-1}$, we also see that $\mathsf{body}(r_j)^{\mathcal{U}'_{j-1}} = \mathbf{t}$. Hence, the claim indeed holds for $k+1$ as well. □

Example 6.12. The converse of Proposition 6.11 does not hold, as illustrated by Example 2.13. The intuitive repair $\mathcal{U}_2 = \{+a, +c\}$ is the AFT-well-founded repair for $\langle DB, \eta\rangle$. However, there is another well-founded repair $\mathcal{U}_1 = \{+a, +b\}$. ▲

Proposition 6.13. *All $\mathfrak{T}_\eta$-stable repairs are justified.*

Proof. We can safely assume that $\mathcal{U}$ is a repair of DB (otherwise, it is not a $\mathfrak{T}_\eta$-stable repair).

Before starting the actual proof, we introduce a couple of auxiliary operators. For each set of update actions $\mathcal{V}$, let $\mathsf{ext}(\mathcal{V})$ denote $\mathcal{V} \cup \mathsf{neff}_{\mathcal{U}}(DB)$. Note that $\mathsf{ext}(\mathcal{V})$ is consistent whenever $\mathcal{V} \subseteq \mathcal{U}$. For each set of update actions $\mathcal{V}$, let $\mathsf{closure}(\mathcal{V})$ denote

$$\mathsf{closure}(\mathcal{V}) = \mathcal{V} \cup \{\mathsf{head}(r) \mid \mathsf{ua}(\mathsf{nup}(r)) \subseteq \mathcal{V}\}.$$

Now, we define the operator (on sets of update actions):

$$O : \mathcal{V} \mapsto \mathsf{closure}(\mathsf{ext}(\mathcal{V})) \setminus \mathsf{neff}_{\mathcal{U}}(DB).$$

Below, we prove the following claims:

Claim 1: $\mathcal{U}$ is a minimal prefixpoint of O if and only if $\mathcal{U}$ is a justified repair of $\langle DB, \eta\rangle$.
Claim 2: For any literal l, $\mathsf{ua}(l) \in \mathsf{ext}(DB)$ if and only if $l^{(\mathcal{V},\mathcal{U})(DB)} = \mathbf{t}$.
Claim 3: For any update action α, $\alpha \in \mathsf{closure}(\mathsf{ext}(\mathcal{V}))$ if and only if $\mathsf{supp}_{(\mathcal{V},\mathcal{U})(DB),\eta}(\alpha) = \mathbf{t}$.
Claim 4: For every set of update actions $\mathcal{V} \subseteq \mathcal{U}$:

$$\mathfrak{T}_\eta(\mathcal{V},\mathcal{U})_1 \subseteq O(\mathcal{V}).$$

Now, assume $\mathcal{U}$ is a $\mathfrak{T}_\eta$-stable repair. It is clear that O is a monotone operator, hence it has a unique minimal prefixpoint, which is also its least fixpoint. Since $\mathcal{U}$ is a repair, it is a fixpoint of O; we need to show that it is minimal. Assume $\mathcal{U}' \subseteq \mathcal{U}$ is a prefixpoint of O as well. This means $O(\mathcal{U}') \subseteq \mathcal{U}'$ and hence

$$\mathfrak{T}_\eta(\mathcal{U}', \mathcal{U})_1 \subseteq (O(\mathcal{U}')) \subseteq \mathcal{U}',$$

i.e., $\mathcal{U}'$ is a prefixpoint of $\mathfrak{T}_\eta(\cdot, \mathcal{U})_1$. Since $\mathcal{U}$ is an $\mathfrak{T}_\eta$-stable repair, $\mathcal{U}$ is the least prefixpoint of $\mathfrak{T}_\eta(\cdot, \mathcal{U})_1$ and thus $\mathcal{U} = \mathcal{U}'$. Hence, also $\mathcal{U} = \mathcal{U}'$, which we needed to prove. We conclude that, indeed, $\mathcal{U}$ is a justified repair.

Claim 1 Recall that $\mathcal{U}$ is a repair. The claim then follows immediately from the definition of justified repair. Prefixpoints of the closure operator are sets of update actions closed under η. Hence $\mathcal{U}$ is a minimal prefixpoint of O iff $\mathsf{ext}(\mathcal{U})$ is a minimal set of update actions that contains $\mathsf{neff}_{\mathcal{U}}(DB)$ and is closed under η.

Claim 2 Pick some literal l.

First assume that $\mathsf{ua}(l) \in \mathsf{ext}(\mathcal{V}) = \mathcal{V} \cup \mathsf{neff}_{\mathcal{U}}(DB)$. We show that $l^{(\mathcal{V},\mathcal{U})(DB)} = \mathbf{t}$. We consider two cases:

- If $\mathsf{ua}(l) \in \mathsf{neff}_{\mathcal{U}}(DB)$, then $l^{DB} = \mathbf{t}$ and $\mathcal{U}$ does not change the value of l. Thus, $l^{(\mathcal{V},\mathcal{U})(DB)} = l^{DB} = \mathbf{t}$.
- If $\mathsf{ua}(l) \notin \mathsf{neff}_{\mathcal{U}}(DB)$, then it must hold that $\mathsf{ua}(l) \in \mathcal{V} \subseteq \mathcal{U}$. Since $\mathsf{ua}(l) \in \mathcal{U}$ and every action in $\mathcal{U}$ changes DB, it must hold that $l^{DB} = \mathbf{f}$. Now, since $\mathsf{ua}(l) \in \mathcal{V}$, $(\mathcal{V}, \mathcal{U})$ changes the value of l and thus $l^{(\mathcal{V},\mathcal{U})(DB)} = (l^{DB})^{-1} = \mathbf{t}$ in this case.

For the other direction, assume that $l^{(\mathcal{V},\mathcal{U})(DB)} = \mathbf{t}$. We need to show that $\mathsf{ua}(l) \in \mathsf{ext}(\mathcal{V}) = \mathcal{V} \cup \mathsf{neff}_{\mathcal{U}}(DB)$. Assume that $\mathsf{ua}(l) \notin \mathsf{neff}_{\mathcal{U}}(DB)$, we show that $\mathsf{ua}(l) \in \mathcal{V}$. Since $l^{(\mathcal{V},\mathcal{U})(DB)} = \mathbf{t}$ and $\mathcal{U} \geq_p (\mathcal{V}, \mathcal{U})$, also $l^{\mathcal{U}} = \mathbf{t}$. Since $l \notin \mathsf{neff}_{\mathcal{U}}(DB)$, this means that $l^{DB} = \mathbf{f}$. Now, $l^{(\mathcal{V},\mathcal{U})(DB)} = \mathbf{t} = (l^{DB})^{-1}$. Hence it must hold that $(\mathcal{V}, \mathcal{U})$ changes the value of l. Since $l^{DB} = \mathbf{f}$, this means that $\mathsf{ua}(l) \in \mathcal{V}$, which is exactly what we needed to prove.

Claim 3 Let α be an update action. It holds that $\mathsf{supp}_{(\mathcal{V},\mathcal{U})(DB),\eta}(\alpha) = \mathbf{t}$ if and only if there is some rule r with $\mathsf{head}(r) = \alpha$ and $\mathsf{nup}(r)^{(\mathcal{V},\mathcal{U})(DB)} = \mathbf{t}$. This means that for each literal $l' \in \mathsf{nup}(r)$, $l'^{(\mathcal{V},\mathcal{U})(DB)} = \mathbf{t}$. From Claim 2 it then follows that this is equivalent with the condition that each literal $l' \in \mathsf{nup}(r)$ is an element of $\mathsf{ext}(\mathcal{V})$, i.e., with the condition that $\alpha \in \mathsf{closure}(\mathsf{ext}(\mathcal{V}))$.

Claim 4 Take $\alpha \in \mathcal{A}$ and a set $\mathcal{V}$ of update actions. It holds that $\alpha \in \mathfrak{T}_\eta(\mathcal{V}, \mathcal{U})_1$ if and only if one of the following holds:

(i) $(\mathcal{V}, \mathcal{U})(\alpha) = \mathbf{t}$ and $\mathsf{supp}_{(\mathcal{V},\mathcal{U})(DB),\eta}(\alpha^D) = \mathbf{f}$
(ii) $(\mathcal{V}, \mathcal{U})(\alpha) = \mathbf{f}$ and $\mathsf{supp}_{(\mathcal{V},\mathcal{U})(DB),\eta}(\alpha) = \mathbf{t}$
(iii) $(\mathcal{V}, \mathcal{U})(\alpha) = \mathbf{u}$, $\mathsf{supp}_{(\mathcal{V},\mathcal{U})(DB),\eta}(\alpha) = \mathbf{t}$, and $\mathsf{supp}_{(\mathcal{V},\mathcal{U})(DB),\eta}(\alpha^D) = \mathbf{f}$.

In the first case, $\alpha \in \mathcal{V}$, hence $\alpha \in \mathsf{ext}(\mathcal{V})$ and $\alpha \in \mathsf{closure}(\mathsf{ext}(\mathcal{V}))$. If $\mathsf{supp}_{(\mathcal{V},\mathcal{U})(DB),\eta}(\alpha) = \mathbf{t}$ (i.e., in cases ii and iii), then by Claim 3, $\alpha \in \mathsf{closure}(\mathsf{ext}(\mathcal{V}))$. Thus, in all cases, $\alpha \in \mathsf{closure}(\mathsf{ext}(\mathcal{V}))$. Now, since $\mathcal{U}$ is a repair, $(\mathcal{U}, \mathcal{U})$ is a fixpoint of $\mathfrak{T}_\eta$. Since $(\mathcal{V}, \mathcal{U}) \leq_p (\mathcal{U}, \mathcal{U})$, $\alpha \in \mathfrak{T}_\eta(\mathcal{V}, \mathcal{U})_1$ entails $\alpha \in \mathfrak{T}_\eta(\mathcal{U}, \mathcal{U})_1$, i.e., $\alpha \in \mathcal{U}$. Thus, $\alpha \notin \mathsf{neff}_{\mathcal{U}}(DB)$ and we find that $\alpha \in \mathsf{closure}(\mathsf{ext}(\mathcal{V})) \setminus \mathsf{neff}_{\mathcal{U}}(DB) = O(\mathcal{V})$, which we needed to prove. □

Example 6.14. The converse of Proposition 6.13 does not hold. Consider the following set η of AICs.

$$\neg a \supset +a$$
$$a \wedge \neg b \supset +b$$
$$a \wedge \neg b \supset -a$$

with $DB = \emptyset$. In this case $\{+a, +b\}$ is a justified repair of $\langle DB, \eta \rangle$, but not a stable repair. To see that it is not a stable repair, it suffices to note that

$$\mathfrak{T}_\eta(\emptyset, \{+a, +b\}) = (\emptyset, \{+a, +b\})$$

and hence

$$\mathrm{lfp}\, \mathfrak{T}_\eta(\cdot, \{+a, +b\})_1 = \emptyset \neq \{+a, +b\}.$$

To see that it is a justified repair, note that $\{+a, +b\}$ is the least set closed under η.

While the converse of Proposition 6.13 does not hold in general, for a broad class of active integrity constraints, it does hold. We first define this class and then prove that this is indeed the case.

Definition 6.15. A set of AICs η is called *unipolar* if there are no rules $r, r' \in \eta$ with $\mathsf{head}(r) = \mathsf{head}(r')^D$.

Unipolar AICs make sense in practice, for example if there are tables from which removing data is never an option.

Proposition 6.16. *If η is unipolar, then each justified repair of $\langle DB, \eta\rangle$ is $\mathfrak{T}_\eta$-stable.*

Proof. To prove this theorem, we show the following strengthening of **Claim 4** in the proof of Proposition 6.13.

Claim 4′: For every set of update actions $\mathcal{V} \subseteq \mathcal{U}$:

$$\mathfrak{T}_\eta(\mathcal{V}, \mathcal{U})_1 = O(\mathcal{V}).$$

It then follows easily that $\mathcal{U}$ is a minimal fixpoint of O iff $\mathcal{U}$ is a minimal fixpoint of $\mathfrak{T}_\eta(\cdot, \mathcal{U})_1$ and the result follows from **Claim 1** in the proof of Proposition 6.13 and the definition of $\mathfrak{T}_\eta$-stable fixpoint.

Claim 4′ One direction of this claim has been proven as **Claim 4** in the proof of Proposition 6.13; we show that the other inclusion also holds. Take $\alpha \in \mathcal{A}$ and a set $\mathcal{V} \subseteq \mathcal{U}$ of update actions such that $\alpha \in O(\mathcal{V})$; we show that $\alpha \in \mathfrak{T}_\eta(\mathcal{V}, \mathcal{U})_1$. Since $\alpha \in O(\mathcal{V})$, it holds that $\alpha \in \mathsf{closure}(\mathsf{ext}(\mathcal{V}))$ and thus (by **Claim 3**) that $\mathsf{supp}_{(\mathcal{V},\mathcal{U})(DB),\eta}(\alpha) = \mathbf{t}$. This means that there is at least one rule $r \in \eta$ with $\mathsf{head}(r) = \alpha$. Since η is unipolar, there can be no rules $r' \in \eta$ with $\mathsf{head}(r') = \alpha^D$ hence, it holds that $\mathsf{supp}_{(\mathcal{V},\mathcal{U})(DB),\eta}(a^D) = \mathbf{f}$. From the definition of $\mathfrak{T}_\eta$, we then find that $\mathfrak{T}_\eta(\mathcal{V}, \mathcal{U})(\alpha) = \mathbf{t}$ and thus that $\alpha \in \mathfrak{T}_\eta(\mathcal{V}, \mathcal{U})_1$, which we needed to prove. □

From Proposition 3.7, which states that all justified repairs are grounded, we easily find how justified repairs and the AFT-well-founded repair relate.

Corollary 6.17. *The AFT-well-founded repair and the Kripke–Kleene repair approximate all justified repairs.*

7. Relationship with logic programming

Caroprese and Truszczyński [13] defined a translation from logic programs to AICs as follows.

Definition 7.1. Let r be a normal logic programming rule,

$$a \leftarrow l_1 \wedge \cdots \wedge l_n.$$

We define the active integrity constraint $\mathsf{aic}(r)$ as

$$l_1 \wedge \ldots \wedge l_n \wedge \neg a \supset +a.$$

Furthermore, if $\mathcal{P}$ is a normal logic program, we define

$$\mathsf{aic}(\mathcal{P}) = \bigcup \{\mathsf{aic}(r) \mid r \in \mathcal{P}\}.$$

Caroprese and Truszczyński [13] showed that *for simple programs* $\mathcal{P}$, an interpretation I is a stable model of $\mathcal{P}$ if and only if I (viewed as an update set) is a justified repair of $\langle \mathsf{aic}(\mathcal{P}), \emptyset\rangle$. Since $\mathsf{aic}(\mathcal{P})$ is unipolar, from our earlier results (Propositions 6.13 and 6.16) it follows that this is also equivalent with the condition that I is a $\mathfrak{T}_{\mathsf{aic}(\mathcal{P})}$-stable repair. The same result is also a corollary of the following stronger theorem.

Theorem 7.2. *Let $\mathcal{P}$ be a simple normal logic program and $\mathcal{I}$ a partial interpretation. It holds that $\mathcal{I}$ is a partial stable model of $\mathcal{P}$ if and only if $\mathcal{I}$ is a partial stable repair of $\langle \mathsf{aic}(\mathcal{P}), \emptyset\rangle$.*

The proof of this theorem follows later, since it makes use of Theorem 7.6. Since the well-founded model of a logic program is the least precise partial stable model, and an analogous relationship holds in the setting of AICs, we immediately find the following corollary.

Corollary 7.3. *Let $\mathcal{P}$ be a normal logic program. The well-founded model of $\mathcal{P}$ coincides with the AFT-well-founded repair of $\langle \mathsf{aic}(\mathcal{P}), \emptyset\rangle$.*

While the operation aic preserves (partial) stable, well-founded fixpoints, it does not preserve grounded fixpoints or the Kripke–Kleene fixpoint (as the following two examples illustrate). In both cases, it is the intuition of *inertia*, present in AICs but not in logic programs, that is responsible for the difference.

Example 7.4. Consider the logic program

$$\mathcal{P}_g = \left\{ \begin{array}{ll} p & \leftarrow \neg q \\ q & \leftarrow p \end{array} \right\}.$$

Since $\mathcal{T}_{\mathcal{P}_g}$ has no fixpoints, $\mathcal{P}_g$ has no grounded models. In this case,

$$\mathsf{aic}(\mathcal{P}_g) = \left\{ \begin{array}{ll} \neg p \wedge \neg q & \supset +p \\ p \wedge \neg q & \supset +q \end{array} \right\}.$$

Now, $\langle \mathsf{aic}(\mathcal{P}_g), \emptyset \rangle$ has one grounded repair, namely $\{+p, +q\}$. ▲

Example 7.5. Consider the logic program

$$\mathcal{P}_{kk} = \{b \leftarrow a\}.$$

The Kripke–Kleene model of this program maps both a and b to **f**. The corresponding AIC

$$\mathsf{aic}(\mathcal{P}_{kk}) = \{\neg b \wedge a \supset +b\}$$

has a Kripke–Kleene repair that maps a and b to **u**, i.e., it is unknown if these need to be changed. ▲

It can be seen from the previous example that for AICs, the Kripke–Kleene semantics is very bad at deriving that something does *not* need to be changed. The well-founded semantics is much stronger with that respect. In a certain sense, one might say that the intuition of "inertia" underlying AICs lies at the foundation of this discrepancy. Now, in logic programming, the Kripke–Kleene semantics exhibits similar behavior. Consider for instance the empty logic program

$$\mathcal{P}_\emptyset = \{\}.$$

This program has a Kripke–Kleene model in which each atom is false, as expected. However, adding a trivial rule

$$p \leftarrow p$$

that "simulates" inertia results in a program with a Kripke–Kleene model in which p is unknown.

Inertia is also responsible for the discrepancy in Example 7.4. Intuitively, $\mathsf{aic}(\mathcal{P}_g)$ from that example corresponds more to the logic program

$$\mathcal{P}'_g = \left\{ \begin{array}{l} p \leftarrow \neg q \\ p \leftarrow p \\ q \leftarrow p \\ q \leftarrow q \end{array} \right\}.$$

Indeed $\mathsf{aic}(\mathcal{P}_g)$ states that if neither p nor q are present in the database, add p, and once we add it, by inertia, it stays unless there is a reason to remove it again (which there is not), and similarly for q. $\mathcal{P}'$ and $\mathcal{P}_g$ only differ from each other in the rules $p \leftarrow p$ and $q \leftarrow q$, which simulate inertia.

The above discussion provides intuitions on what a transformation that preserves all AFT semantics should look like. In the following theorem, this is formalized.

Theorem 7.6. *Suppose that $DB = \emptyset$ and that the only update actions in η are of the form $+a$. The mapping* ua *induces an isomorphism between the lattices 2^{At} and $2^{\mathcal{A}}$ and between $(2^{At})^c$ and $(2^{\mathcal{A}})^c$. Let* $\mathsf{lp}(\eta)$ *denote the following logic program:*

$$\begin{aligned} \mathsf{lp}(\eta) = &\{a \leftarrow \mathsf{nup}(r) \mid r \in \eta, \mathsf{head}(r) = +a\} \\ &\cup \{a \leftarrow a \mid a \in At\} \end{aligned}$$

then for each partial interpretation $\mathcal{I}$, $\mathfrak{T}_\eta(\mathsf{ua}(\mathcal{I})) = \mathsf{ua}(\Psi_{\mathsf{lp}(\eta)}(\mathcal{I}))$. Hence, all AFT semantics for $\langle DB, \eta \rangle$ coincide in this case with the equally-named semantics for the logic program $\mathsf{lp}(\eta)$.

Proof. By definition,

$$\begin{aligned} \Psi_{\mathsf{lp}(\eta)}(\mathcal{I})_1 &= \{a \in At \mid \mathsf{body}(r)^{\mathcal{I}} = \mathbf{t} \text{ for some } r \in \mathsf{lp}(\eta) \text{ with } \mathsf{head}(r) = a\} \\ &= \{a \in At \mid \mathsf{nup}(r)^{\mathcal{I}} = \mathbf{t} \text{ for some } r \in \eta \text{ with } \mathsf{head}(r) = +a\} \cup \{a \in At \mid a^{\mathcal{I}} = \mathbf{t}\} \\ &= \{\mathsf{lit}(\mathsf{head}(r)) \mid r \in \eta \wedge \mathsf{nup}(r)^{\mathcal{I}} = \mathbf{t}\} \cup \{a \in At \mid a^{\mathcal{I}} = \mathbf{t}\} \\ &= \{\mathsf{lit}(\alpha) \mid \mathsf{supp}_{\mathcal{I},\eta}(\alpha) = \mathbf{t}\} \cup \{a \in At \mid a^{\mathcal{I}} = \mathbf{t}\} \end{aligned}$$

Now, since, for each $\alpha \in \mathcal{A}$, it holds that $\mathsf{supp}_{\mathcal{I},\eta}(\alpha^D) = \mathbf{f}$ (since there are no rules with α^D in the head). From the definition of $\mathfrak{T}_\eta$, it then follows that

$$\mathfrak{T}_\eta(\mathsf{ua}(\mathcal{I}))_1 = \{\alpha \mid \mathsf{supp}_{\mathcal{I},\eta}(\alpha) = \mathbf{t} \text{ or } \alpha^{\mathsf{ua}(\mathcal{I})} = \mathbf{t}\},$$

i.e., that

$$\mathfrak{T}_\eta(\mathsf{ua}(\mathcal{I}))_1 = \mathsf{ua}(\Psi_{\mathsf{lp}(\eta)}(\mathcal{I})_1).$$

Similarly, we also find that

$$\mathfrak{T}_\eta(\mathsf{ua}(\mathcal{I}))_2 = \mathsf{ua}(\Psi_{\mathsf{lp}(\eta)}(\mathcal{I})_2)$$

and the result follows. □

Proof of Theorem 7.2. For each logic program $\mathcal{P}$, let $\mathsf{triv}(\mathcal{P})$ denote the logic program

$$\mathsf{triv}(\mathcal{P}) = \mathcal{P} \cup \{p \leftarrow p \mid p \in At\}.$$

It is well-known[6] that the partial stable models of $\mathcal{P}$ and those of $\mathsf{triv}(\mathcal{P})$ are the same. Now, the result easily follows from the facts that **(i)** if $\mathcal{P}$ is simple, then

$$\mathsf{lp}(\mathsf{aic}(\mathcal{P})) = \mathsf{triv}(\mathcal{P})$$

since aic adds a literal $\neg a$ to the body of each rule r with $\mathsf{head}(r) = a$ and lp removes such literal again (since $\mathcal{P}$ is simple, there was no such literal at the start) and additionally, adds the trivial rules triv adds, and **(ii)** that triv and lp preserve partial stable models. □

Example 7.7. Theorem 7.6 does not hold in general, not even for unipolar AICs. Consider for instance the following sets of AICs.

$$\eta_1 = \left\{ \begin{array}{l} a \supset -a \\ b \supset -b \end{array} \right\}$$

$$\eta_2 = \emptyset.$$

These two are not equivalent under all AFT semantics. For the first one, the KK-repair is $\{+a \mapsto \mathbf{f}, +b \mapsto \mathbf{f}\}$, while for the latter, the KK-repair is $\{+a \mapsto \mathbf{u}, +b \mapsto \mathbf{u}\}$. However, they have the same translation to logic programs (since there are no rules with positive actions in the head). ▲

The reason why the equivalence does not hold in the previous example is because rules with $-a$ in the head are ignored, while in the context of AICs such rules can make a semantic difference.

Limitations and related work In this Section, we studied mappings from sets of active integrity constraints to logic programs and back and how the semantics of the two formalisms relate. We briefly discuss the restrictions of this approach. In our translation of AICs to logic programs, namely in Theorem 7.6, we assume that $DB = \emptyset$ and that the only update actions that occur in heads of rules are positive, i.e., of the form $+a$. Since all our semantics satisfy the shifting property, the correspondence between the semantics still holds if DB is arbitrary but all actions in rule heads change DB. However, the condition that all actions in heads of rules change DB is essential, as illustrated by Example 7.7.

The translation *Rew* introduced by Caroprese et al. [12], on the other hand, is at first sight more expressive than ours, as it applies to AICs with arbitrary actions on their head. In order to deal also with removal actions (in the case of an empty database), the authors are required to consider a logic program with an extended signature that includes two copies of the database and two atoms for each possible update action. They can then prove a precise correspondence between stable models of logic programs and founded repairs for AICs – the only semantics that had been developed at that point in time.

However, their construction is very directly tailored to founded repairs, and it is not obvious that it can be adapted to AFT semantics. In order to capture inertia, the authors duplicate all database atoms, so that every model contains both a copy of the original database and a copy of the repaired database. Additional rules ensure that, in each model, the repaired database corresponds exactly to the result of applying the update actions also included in that model to the original database. It can easily be seen that this mapping does not preserve AFT semantics for AICs. For instance, this technique maps founded repairs (which are not always grounded) to stable models (which are always grounded), hence, it certainly does not preserve grounded repairs. By contrast, the correspondence stated in Theorem 7.6 holds for all different semantics considered in the current work.

[6] For completeness, we give the argument. Let I be an interpretation; from the definition of partial stable fixpoint, it suffices to show that $\mathsf{lfp}(\Psi_{\mathcal{P}}(\cdot, I)_1) = \mathsf{lfp}(\Psi_{\mathsf{triv}(\mathcal{P})}(\cdot, I)_1)$ and that $\mathsf{lfp}(\Psi_{\mathcal{P}}(I, \cdot)_2) = \mathsf{lfp}(\Psi_{\mathsf{triv}(\mathcal{P})}(I, \cdot)_2)$. Since we are working with symmetric operators, it suffices to prove the first of the two equalities. Now, it is easy to see that $\Psi_{\mathsf{triv}(\mathcal{P})}(\cdot, I)_1$ is the inflationary operator of $\Psi_{\mathcal{P}}(\cdot, I)_1$, i.e., that $\Psi_{\mathsf{triv}(\mathcal{P})}(\cdot, I)_1(J) = \Psi_{\mathcal{P}}(\cdot, I)_1(J) \cup J$. Hence, these two monotone operators have the same prefixpoints and thus also the same least prefixpoint, which equals their least prefixpoint.

Another limitation of our results is that for the reverse translation, we only preserve (partial) stable repairs and the AFT-well-founded repair (cf. Theorem 7.2). However, our discussion does highlight where the difference comes from, namely the intuition of inertia. That is, as shown in the proof of Theorem 7.2: for each simple normal logic program $\mathcal{P}$, the semantics of $\mathsf{triv}(\mathcal{P})$ and $\mathsf{aic}(\mathcal{P})$ coincide.

Furthermore, we restrict our attention to *normal* programs; Caroprese and Truszczyński [13] discuss this translation without requiring normality. However, as mentioned before, we see non-normal AICs as syntactic sugar for their normalization and hence have no need for non-normal logic programs either. The same applies to the translation *Rew* from Caroprese et al. [12]. The restriction that our programs are *simple* is rather a technical requirement that simplifies proofs. For several semantics, this is non-essential: for (partial) stable and well-founded semantics, each program can easily be translated into an equivalent simple program.

8. Complexity analysis

We begin this section by stating an observation about the complexity of computing $\mathfrak{T}_\eta$. All complexity results are in terms of the size of the database, $\langle DB, \eta\rangle$.

Proposition 8.1. *Given a partial action set $\mathfrak{U}$, $\mathfrak{T}_\eta(\mathfrak{U})$ is computable in polynomial time.*

Proof. The definition of $\mathfrak{T}_\eta$ only requires evaluating $\mathfrak{U}(\alpha)$, $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha)$ and $\mathsf{supp}_{\mathfrak{U}(DB),\eta}(\alpha^D)$ for each $\alpha \in \mathcal{A}$. In turn, the two last computations can be done in polynomial time: they require evaluating each literal in the body of each rule from η with head α or α^D and computing its truth value under the database updated by $\mathfrak{U}$. □

Proposition 8.2. *Let DB be a database and η be a set of normal AICs over DB. The problem of deciding whether there exists a grounded repair for $\langle DB, \eta\rangle$ is Σ_2^P-complete.*

Proof. *(Inclusion)* We need to show that we can decide the problem with a non-deterministic Turing machine with an NP oracle. Given a set of update actions $\mathcal{U}$, checking that it is a fixpoint of $\mathcal{T}_\eta$ can be done in polynomial time on the size of *DB* and η, as shown in Proposition 8.1; the NP-oracle can then answer whether there exists $\mathcal{U}' \subsetneq \mathcal{U}$ with $\mathcal{T}_\eta(\mathcal{U}') \cap \mathcal{U} \subseteq \mathcal{U}'$, thereby establishing whether $\mathcal{U}$ is grounded.

(Hardness) We show hardness directly by reducing another Σ_2^P-hard problem to deciding whether a particular database with a set of AICs has a grounded repair. Our proof is a straightforward adaptation of Bogaerts et al. [7, Theorem 5.7], which in turn is inspired by Denecker et al. [22, Theorem 6.12].

Given a DNF formula φ over propositional symbols $x_1, \ldots, x_m, y_1, \ldots, y_n$, and an interpretation I of the x_i, let φ_I denote the formula obtained by replacing each occurrence of x_i with either **t**, if $x_i \in I$, or **f**, otherwise. The problem to decide whether there exists an interpretation I of the x_i such that φ_I is a tautology is Σ_2^P hard. We now reduce this problem to our problem.

We consider the empty database *DB* over $At = \{x_i, x_i' \mid 1 \le i \le m\} \cup \{p, q, y_1, \ldots, y_n\}$, where we use x_i' to represent the negation of x_i. We write φ' for the formula obtained by uniformly replacing $\neg x_i$ with x_i' in φ. The set of AICs $\eta(\varphi)$ is defined as follows, where we assume $\varphi' = \varphi_1' \vee \ldots \vee \varphi_k'$ and each φ_i' is a conjunction of literals.

$$\neg x_i \wedge \neg x_i' \supset +x_i \qquad \neg x_i \wedge \neg x_i' \supset +x_i' \qquad \text{for } 1 \le i \le m \tag{r_{21}}$$

$$\varphi_i' \wedge \neg y_j \supset +y_j \qquad \varphi_i' \wedge \neg p \supset +p \qquad \text{for } 1 \le i \le k, 1 \le j \le n \tag{r_{22}}$$

$$\neg p, \neg q \supset +q \qquad \neg p, q \supset -q \tag{r_{23}}$$

The following properties hold about $\eta(\varphi)$.

a. Each weak repair of $\langle \eta(\varphi), \emptyset\rangle$ contains $+p$ (otherwise one of the rules r_{23} would apply).
b. No repair for $\langle \eta(\varphi), \emptyset\rangle$ contains $+q$ (due to minimality).
c. In each grounded repair for $\langle \eta(\varphi), \emptyset\rangle$, at least one of the φ_i' is satisfied (otherwise, removing $+p$ from that repair would result in a set of update actions where $+p$ is no longer derivable, contradicting groundedness).
d. Each grounded repair for $\langle \eta(\varphi), \emptyset\rangle$ contains all of the $+y_j$ (follows directly from the previous point).
e. Each grounded repair for $\langle \eta(\varphi), \emptyset\rangle$ contains for each i, exactly one of $+x_i$ and $+x_i'$ (it must contain at least one due to rules r_{21}; the previous points guarantee that rules r_{22} and r_{23} are satisfied regardless of the x_i and x_i' in each grounded repair, hence minimality implies that it can contain at most one of these two actions).

Given an interpretation I, we write $\breve{I}$ to denote the set $\{+x_i \mid x_i \in I\} \cup \{+x_i' \mid x_i \notin I\}$. From observations a–e above, it follows that all grounded fixpoints for $\langle \eta(\varphi), \emptyset\rangle$ must be of the form $\mathcal{U}_I = \breve{I} \cup \{p, y_1, \ldots, y_n\}$ for some interpretation I of the x_i. We now show that for each interpretation I, $\mathcal{U}_I$ is a grounded repair for $\langle \eta(\varphi), \emptyset\rangle$ iff φ_I is a tautology.

- First, assume that $\mathcal{U}_I$ is a grounded repair of $\langle \eta(\varphi), \emptyset \rangle$. If $J \subseteq \{y_1, \ldots, y_n\}$ is a falsifying assignment for φ_I, then $T_{\eta(\varphi)}(\check{I} \cup \{+y_i \mid y_i \in J\}) \cap \mathcal{U} = (\check{I} \cup \{+y_i \mid y_i \in J\} \cup \{+q\}) \cap \mathcal{U} = \check{I} \cup \{+y_i \mid y_i \in J\}$, contradicting the fact that $\mathcal{U}$ is a grounded repair for $\langle \eta(\varphi), \emptyset \rangle$. (Note that $\check{I} \cup \{+y_i \mid y_i \in J\}$ is always a strict subset of $\mathcal{U}$, since it does not contain p.) Therefore φ_I is a tautology.
- Suppose on the other hand that φ_I is a tautology. Let $\mathcal{V} \subseteq \mathcal{U}$ be such that $T_{\eta(\varphi)}(\mathcal{V}) \cap \mathcal{U} \subseteq \mathcal{V}$. If $+x_i \in (\mathcal{U} \setminus \mathcal{V})$ or $+x'_i \in (\mathcal{U} \setminus \mathcal{V})$ for some i, then $+x_i \in T_{\eta(\varphi)}(\mathcal{V})$ by rules r_{21}; therefore, $\check{I} \subseteq \mathcal{V}$. But φ_I is a tautology, hence if $+y_i \notin \mathcal{V}$, then the corresponding rule from r_{22} ensures that $+y_i \in T_{\eta(\varphi)}(\mathcal{V})$. Likewise, if $+p \notin \mathcal{V}$, then $+p \in T_{\eta(\varphi)}(\mathcal{V})$. We thus conclude that $\mathcal{V} = \mathcal{U}$, whence $\mathcal{U}$ is a grounded repair for $\langle \eta(\varphi), \emptyset \rangle$. □

Proposition 8.3. *The Kripke–Kleene repair for $\langle DB, \eta \rangle$ is computable in polynomial time.*

Proof. The Kripke–Kleene repair of $\langle DB, \eta \rangle$ can be computed by iterating $\mathfrak{T}_\eta$ until a fixpoint is reached. Since $\mathfrak{T}_\eta$ is monotonic, the maximum number of iterations is the size of At; since each iteration can be computed in polynomial time (Proposition 8.1), so can this fixpoint. □

Proposition 8.4. *The ATF-well-founded repair for $\langle DB, \eta \rangle$ is computable in polynomial time.*

The proof makes use of the following proposition.

Proposition 8.5 *([23]). Let A be an approximator of O and $(x, y) \in L^2$. Let S^x_A be the operator on L that maps every y' to $A(x, y')_2$. This operator is monotone. The smallest y' such that (x, y') is an unfoundedness refinement of (x, y) is given by $y' = \mathrm{lfp}(S^x_A)$.*

Proof of Proposition 8.4. To compute the $\mathfrak{T}_\eta$-well-founded fixpoint, we can construct a well-founded induction with only strict refinements. Since such a well-founded is $\leq_p$-increasing, it can consist of at most of $|\mathcal{A}| = |At|$ steps. Computing if there exists a strict application refinement of a given partial repair set $\mathfrak{U}$ can be done by computing $\mathfrak{T}_\eta(\mathfrak{U})$. Now, Proposition 8.5 shows that the most precise unfoundedness refinement can also be computed as the least fixpoint of a derived operator on $2^{\mathcal{A}}$. Such a fixpoint can again be computed in polynomial time. Hence, it follows that we can compute a terminal well-founded induction, and thus the well-founded fixpoint, in polynomial time. □

Proposition 8.6. *The task of checking if a database $\langle DB, \eta \rangle$ has a stable repair is NP-complete.*

Proof. *(Inclusion)* Given a candidate repair $\mathcal{U}$, checking that it is stable can be done in polynomial time, as it amounts to verifying that it is a repair (two-valued) and that it is a least fixpoint of the operators $\mathfrak{T}_\eta(\cdot, \mathcal{U})_1$ and $\mathfrak{T}_\eta(\mathcal{U}, \cdot)_2$. The latter can be done in polynomial time, as in the proof of Proposition 8.3.

(Hardness) For hardness, we again use the reduction from simple logic programs to AICs from Caroprese and Truszczyński [13] given in Definition 7.1. This operator preserves stable semantics by Theorem 7.2, and therefore allows us to compute stable models of simple logic programs by first translating them (in linear time) to sets of AICs. Since checking whether a logic program has a stable model is NP-complete [4,30], we conclude that checking whether a database has a stable repair must be NP-hard. Note that every logic program can be transformed in a simple logic program by removing the offending rules without changing its stable semantics. □

What we notice in this section is that complexity for inference tasks related to our semantics is always the same as the complexity of its counterpart in (normal) logic programming. This illustrates that the added expressivity (essentially, allowing AICs that are not unipolar) does not result in added complexity.

Contrary to the original work introducing AICs [27], our definitions do not include first-order quantifications. When allowing such a richer syntax, the results presented in this section can be re-used and constitute data-complexity results.

9. Conclusion

In this paper, we defined an approximator in the domain of active integrity constraints. The result is a family of semantics for AICs based on existing intuitions in various domains of non-monotonic reasoning. We studied properties of our induced semantics. In particular the AFT-well-founded semantics possesses desirable properties: it approximates all repairs of various families (stable, justified, grounded) and hence can be used for approximate skeptical query-answering with respect to any of these semantics. Furthermore, the AFT-well-founded repair can be computed in time polynomial in the size of the database.

Our study is far from finished. In the context of approximation fixpoint theory, *ultimate approximators* have been studied by Denecker et al. [22]. They showed that with each two-valued operator, we can associate a canonical approximator. The ultimate approximator induces another family of semantics for AICs. In other domains, e.g., in logic programming, semantics based on ultimate approximators have some very desirable properties, but in general come at the cost of a higher

computational complexity than their "standard" variants. It remains to be researched if the same holds in the context of AICs. In this paper, we showed that the class of justified repairs is situated in between the classes of stable and of grounded repairs. It is known from AFT that the class of ultimate stable fixpoints also falls in between the classes of stable fixpoints (for any approximator) and grounded fixpoints. Hence, an interesting research question would be to verify if justified repairs coincide with ultimate stable fixpoints in this domain, and if not, how they relate. Another topic with potential for interesting future work is the notion of inconsistency. Consider for instance the set of AICs $\{\neg a \supset +a, a \supset -a\}$; intuitively, we expect a semantic operator to derive an inconsistency from *any* partial action set; in standard AFT this is not possible. However, extensions of AFT that accommodate this have been defined [3]; it would be interesting to see how AICs fit in this general theory. Another AFT-based topic of interest could be to study what *safe inductions* [9] yield in the context of AICs and whether they can fix problems with the well-founded semantics. One last topic on which more extensive research might be needed is the domain of revision programming [31]. Caroprese and Truszczyński [13] showed structural correspondences between semantics for AICs and semantics for revision programs. Our paper now paves the way to applying AFT to revision programming as well.

Acknowledgements

Bart Bogaerts is a postdoctoral fellow of the Research Foundation – Flanders (FWO). Luís Cruz-Filipe was partially supported by the Danish Council for Independent Research Natural Sciences, grants DFF-1323-00247 and DFF-7014-00041.

References

[1] S. Abiteboul, Updates, a new frontier, in: M. Gyssens, J. Paredaens, D.V. Gucht (Eds.), Proceedings ICDT'88, 2nd International Conference on Database Theory, Bruges, Belgium, August 31–September 2, 1988, in: Lecture Notes in Computer Science, vol. 326, Springer, 1988, pp. 1–18.
[2] C. Antic, T. Eiter, M. Fink, Hex semantics via approximation fixpoint theory, in: P. Cabalar, T.C. Son (Eds.), Proceedings Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15–19, 2013, in: LNCS, vol. 8148, Springer, 2013, pp. 102–115.
[3] Y. Bi, J. You, Z. Feng, A generalization of approximation fixpoint theory and application, in: R. Kontchakov, M. Mugnier (Eds.), Proceedings Web Reasoning and Rule Systems – 8th International Conference, RR 2014, Athens, Greece, September 15–17, 2014, in: Lecture Notes in Computer Science, vol. 8741, Springer, 2014, pp. 45–59.
[4] N. Bidoit, C. Froidevaux, Negation by default and unstratifiable logic programs, Theor. Comput. Sci. 78 (1) (1991) 86–112, https://doi.org/10.1016/0304-3975(51)90004-7.
[5] B. Bogaerts, Groundedness in Logics with a Fixpoint Semantics, Ph.D. thesis, Department of Computer Science, KU Leuven, Jun. 2015, Marc Denecker (supervisor), Joost Vennekens, Jan Van den Bussche (cosupervisors), https://lirias.kuleuven.be/handle/123456789/496543.
[6] B. Bogaerts, L. Cruz-Filipe, Semantics for active integrity constraints using approximation fixpoint theory, in: [35], pp. 866–872, https://doi.org/10.24963/ijcai.2017/120.
[7] B. Bogaerts, J. Vennekens, M. Denecker, Grounded fixpoints and their applications in knowledge representation, Artif. Intell. 224 (2015) 51–71, https://doi.org/10.1016/j.artint.2015.03.006.
[8] B. Bogaerts, J. Vennekens, M. Denecker, Partial grounded fixpoints, in: Q. Yang, M. Wooldridge (Eds.), Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25–31, 2015, AAAI Press, 2015, pp. 2784–2790, http://ijcai.org/papers15/Abstracts/IJCAI15-394.html.
[9] B. Bogaerts, J. Vennekens, M. Denecker, Safe inductions: an algebraic study, in: [35], pp. 859–865, https://doi.org/10.24963/ijcai.2017/119.
[10] B. Bogaerts, J. Vennekens, M. Denecker, J. Van den Bussche, FO(C): a knowledge representation language of causality, Theory Pract. Log. Program. 14 (4–5-Online-Supplement) (2014) 60–69, https://lirias.kuleuven.be/handle/123456789/459436.
[11] G. Brewka, S. Woltran, Abstract dialectical frameworks, in: F. Lin, U. Sattler, M. Truszczyński (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9–13, 2010, AAAI Press, 2010, pp. 102–111, http://aaai.org/ocs/index.php/KR/KR2010/paper/view/1294.
[12] L. Caroprese, S. Greco, C. Sirangelo, E. Zumpano, Declarative semantics of production rules for integrity maintenance, in: S. Etalle, M. Truszczyński (Eds.), Proceedings Logic Programming, 22nd International Conference, ICLP 2006, Seattle, WA, USA, August 17–20, 2006, in: LNCS, vol. 4079, Springer, 2006, pp. 26–40.
[13] L. Caroprese, M. Truszczyński, Active integrity constraints and revision programming, Theory Pract. Log. Program. 11 (6) (2011) 905–952, https://doi.org/10.1017/S1471068410000475.
[14] L. Cruz-Filipe, Optimizing computation of repairs from active integrity constraints, in: C. Beierle, C. Meghini (Eds.), Proceedings Foundations of Information and Knowledge Systems – 8th International Symposium, FoIKS 2014, Bordeaux, France, March 3–7, 2014, in: Lecture Notes in Computer Science, vol. 8367, Springer, 2014, pp. 361–380.
[15] L. Cruz-Filipe, Grounded fixpoints and active integrity constraints, in: M. Carro, A. King, M. De Vos, N. Saeedloei (Eds.), Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16–21, 2016, New York City, USA, in: OASIcs, vol. 52, Schloss Dagstuhl, Nov. 2016, pp. 1–14, chap. 11.
[16] L. Cruz-Filipe, M. Franz, A. Hakhverdyan, M. Ludovico, I. Nunes, P. Schneider-Kamp, repAIrC: a tool for ensuring data consistency, in: A.L.N. Fred, J.L.G. Dietz, D. Aveiro, K. Liu, J. Filipe (Eds.), KMIS 2015 – Proceedings of the International Conference on Knowledge Management and Information Sharing, Part of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015), vol. 3, Lisbon, Portugal, November 12–14, 2015, SciTePress, 2015, pp. 17–26.
[17] L. Cruz-Filipe, G. Gaspar, P. Engrácia, I. Nunes, Computing repairs from active integrity constraints, in: Seventh International Symposium on Theoretical Aspects of Software Engineering, TASE 2013, Birmingham, UK, 1–3 July 2013, IEEE Computer Society, 2013, pp. 183–190.
[18] L. Cruz-Filipe, G. Gaspar, I. Nunes, P. Schneider-Kamp, Active integrity constraints for multi-context systems, in: E. Blomqvist, P. Ciancarini, F. Poggi, F. Vitali (Eds.), Proceedings Knowledge Engineering and Knowledge Management – 20th International Conference, EKAW 2016, Bologna, Italy, November 19–23, 2016, in: Lecture Notes in Computer Science, vol. 10024, 2016, pp. 98–112.
[19] M. Denecker, M. Bruynooghe, J. Vennekens, Approximation fixpoint theory and the semantics of logic and answers set programs, in: E. Erdem, J. Lee, Y. Lierler, D. Pearce (Eds.), Correct Reasoning, in: LNCS, vol. 7265, Springer, 2012, pp. 178–194.
[20] M. Denecker, V. Marek, M. Truszczyński, Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning, in: J. Minker (Ed.), Logic-Based Artificial Intelligence, in: The Springer International Series in Engineering and Computer Science, vol. 597, Springer, US, 2000, pp. 127–144.

[21] M. Denecker, V. Marek, M. Truszczyński, Uniform semantic treatment of default and autoepistemic logics, Artif. Intell. 143 (1) (2003) 79–122, https://doi.org/10.1016/S0004-3702(02)00293-X.
[22] M. Denecker, V. Marek, M. Truszczyński, Ultimate approximation and its application in nonmonotonic knowledge representation systems, Inf. Comput. 192 (1) (Jul. 2004) 84–121, https://lirias.kuleuven.be/handle/123456789/124562.
[23] M. Denecker, J. Vennekens, Well-founded semantics and the algebraic theory of non-monotone inductive definitions, in: C. Baral, G. Brewka, J.S. Schlipf (Eds.), LPNMR, in: Lecture Notes in Computer Science, vol. 4483, Springer, 2007, pp. 84–96.
[24] P.M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, Artif. Intell. 77 (2) (1995) 321–357, https://doi.org/10.1016/0004-3702(94)00041-X.
[25] T. Eiter, G. Gottlob, On the complexity of propositional knowledge base revision, updates, and counterfactuals, Artif. Intell. 57 (2–3) (1992) 227–270, https://doi.org/10.1016/0004-3702(92)90018-S.
[26] M. Fitting, Fixpoint semantics for logic programming — a survey, Theor. Comput. Sci. 278 (1–2) (2002) 25–51, https://doi.org/10.1016/S0304-3975(00)00330-3.
[27] S. Flesca, S. Greco, E. Zumpano, Active integrity constraints, in: E. Moggi, D.S. Warren (Eds.), Proceedings of the 6th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 24–26 August 2004, Verona, Italy, ACM, 2004, pp. 98–107.
[28] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R.A. Kowalski, K.A. Bowen (Eds.), ICLP/SLP, MIT Press, 1988, pp. 1070–1080, http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.6050.
[29] S.C. Kleene, On notation for ordinal numbers, J. Symb. Log. 3 (4) (1938) 150–155, http://www.jstor.org/stable/2267778.
[30] V. Marek, M. Truszczyński, Autoepistemic logic, J. ACM 38 (3) (1991) 588–619, https://doi.org/10.1145/116825.116836.
[31] V.W. Marek, M. Truszczynski, Revision programming, Theor. Comput. Sci. 190 (2) (1998) 241–277, https://doi.org/10.1016/S0304-3975(97)00092-3.
[32] R.C. Moore, Semantical considerations on nonmonotonic logic, Artif. Intell. 25 (1) (1985) 75–94, https://doi.org/10.1016/0004-3702(85)90042-6.
[33] T.C. Przymusinski, H. Turner, Update by means of inference rules, J. Log. Program. 30 (2) (1997) 125–143, https://doi.org/10.1016/S0743-1066(96)00091-X.
[34] R. Reiter, A logic for default reasoning, Artif. Intell. 13 (1–2) (1980) 81–132, https://doi.org/10.1016/0004-3702(80)90014-4.
[35] C. Sierra (Ed.), Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19–25, 2017, ijcai.org, 2017, http://www.ijcai.org/Proceedings/2017/.
[36] H. Strass, Approximating operators and semantics for abstract dialectical frameworks, Artif. Intell. 205 (2013) 39–70, https://doi.org/10.1016/j.artint.2013.09.004.
[37] H. Strass, J.P. Wallner, Analyzing the computational complexity of abstract dialectical frameworks via approximation fixpoint theory, in: C. Baral, G. De Giacomo, T. Eiter (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20–24, 2014, AAAI Press, 2014, pp. 101–110, http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/7917.
[38] E. Teniente, A. Olivé, Updating knowledge bases while maintaining their consistency, VLDB J. 4 (2) (1995) 193–241, http://www.vldb.org/journal/VLDBJ4/P193.pdf.
[39] M.H. van Emden, R.A. Kowalski, The semantics of predicate logic as a programming language, J. ACM 23 (4) (1976) 733–742, https://doi.org/10.1145/321978.321991.
[40] A. Van Gelder, K.A. Ross, J.S. Schlipf, The well-founded semantics for general logic programs, J. ACM 38 (3) (1991) 620–650, https://doi.org/10.1145/116825.116838.
[41] J. Vennekens, D. Gilis, M. Denecker, Splitting an operator: algebraic modularity results for logics with fixpoint semantics, ACM Trans. Comput. Log. 7 (4) (2006) 765–797, https://doi.org/10.1145/1182613.1189735.
[42] J. Vennekens, M. Mariën, J. Wittocx, M. Denecker, Predicate introduction for logics with a fixpoint semantics. Part I: logic programming, Fundam. Inform. 79 (1–2) (September 2007) 187–208, https://lirias.kuleuven.be/handle/123456789/266021.
[43] J. Vennekens, M. Mariën, J. Wittocx, M. Denecker, Predicate introduction for logics with a fixpoint semantics. Part II: autoepistemic logic, Fundam. Inform. 79 (1–2) (September 2007) 209–227, https://lirias.kuleuven.be/handle/123456789/146591.
[44] J. Widom, S. Ceri (Eds.), Active Database Systems: Triggers and Rules for Advanced Database Processing, Morgan Kaufmann, 1996.
[45] M. Winslett, Updating Logical Databases, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1990.