Semantics for Active Integrity Constraints Using Approximation Fixpoint Theory*

Bart Bogaerts† and **Luís Cruz-Filipe**‡
† KU Leuven, Department of Computer Science
Celestijnenlaan 200A, Leuven, Belgium
‡ University of Southern Denmark, Department of Mathematics and Computer Science
Campusvej 55, Odense, Denmark
bart.bogaerts@cs.kuleuven.be, lcfilipe@gmail.com

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.

In this paper, we apply *approximation fixpoint theory*, an algebraic framework that unifies semantics of non-monotonic logics, to the field of AICs. This results in a new family of semantics for AICs, of which we study semantics and relationships to existing semantics. We argue that the AFT-well-founded semantics has some desirable properties.

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 by updating), it is necessary to check if its integrity constraints still hold; in the negative case, it must be repaired.

Database repairs have been an important topic of research for many years [Abiteboul, 1988]. 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 [Winslett, 1990; Eiter and Gottlob, 1992] – change as little as possible – and the common-sense law of inertia [Przymusinski and Turner, 1997] – do not change anything without a reason to do it.

A typical implementation of integrity constraints in database systems is by means of event-condition-action rules [Teniente and Olivé, 1995; Widom and Ceri, 1996], which specify update actions to be performed when a particular event (a trigger) occurs and specific conditions hold. However, these rules do not have a declarative semantics, making their joint behavior hard to understand.

The formalism of *active integrity constraints (AICs)* [Flesca *et al.*, 2004] 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 [Caroprese and Truszczyński, 2011] that identifies several progressively more restricted classes of repairs, which can be used as criteria to select a preferred repair. These repairs can be computed by means of tree algorithms[Cruz-Filipe *et al.*, 2013; Cruz-Filipe *et al.*, 2015].

It is striking that many intuitions about "good" repairs, such as minimality of change, are similar to intuitions that surfaced in other domains of non-monotonic reasoning, such as logic programming [van Emden and Kowalski, 1976] and default logic [Reiter, 1980]. Still, it has been hard to find satisfying semantics for AICs. As shown by Cruz-Filipe *et al.* [2013], the semantics of founded repairs [Caroprese *et al.*, 2006] unexpectedly fails to respect the common-sense law of inertia, while the more restricted semantics of justified repairs [Caroprese and Truszczyński, 2011] forbids natural repairs. That work proposed the operational semantics of well-founded repairs, which however is not modular [Cruz-Filipe, 2014] and therefore restricted in its practical applicability.

In this paper, we define a new class of semantics for AICs that are natural counterparts of existing semantics in various non-monotonic reasoning domains. To be precise, we define semantics for AICs based on *approximation fixpoint theory (AFT)* [Denecker *et al.*, 2000], an abstract algebraic framework that unifies semantics of logic programming, default logic, autoepistemic logic [Moore, 1985], abstract argumentation frameworks [Dung, 1995] and abstract dialectical frameworks [Brewka and Woltran, 2010], as shown by Denecker, Marek and Truszczyński [2000; 2003] and Strass [2013]. In order to do so, we continue the work of Cruz-Filipe [2016], who defined a semantic operator for AICs and used it to study the grounded fixpoint semantics [Bogaerts *et al.*, 2015a] of AICs. In this paper, we define an approximating operator of Cruz-Filipe's semantic operator. From this operator, approximation fixpoint theory immediately provides us with **(i)** a supported fixpoint semantics, **(ii)** a well-founded

*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, grant DFF-1323-00247.

fixpoint semantics, **(iii)** a (partial) stable fixpoint semantics, **(iv)** a Kripke-Kleene fixpoint semantics, and **(v)** a (partial) grounded fixpoint semantics. We study properties of these new semantics and study how they compare to existing semantics. We argue that the AFT-style well-founded semantics is valuable. Indeed, we show that this semantics can be computed in polynomial time, and that, on 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).

Besides defining a new class of interesting semantics, our work provides solid foundations for transferring work from the non-monotonic reasoning domain to AICs. For instance, we can now directly apply existing results from AFT, such as modularity results [Vennekens *et al.*, 2006; Bogaerts *et al.*, 2016] or predicate introduction results [Vennekens *et al.*, 2007a; 2007b]. Following Caroprese and Truszczyński [2011], our work also paves the way to applying AFT to revision programming.

2 Preliminaries

Active Integrity Constraints We assume a fixed set At of *atoms*. An *interpretation* or *database* is a subset of At. A *literal* is an atom a or its negation $\neg a$. We say $\neg a$ and a are dual literals and denote the dual of a literal l by l^D. The satisfaction relation between databases DB and literals is defined as usual: $DB \models a$ if $a \in DB$ and $DB \models \neg a$ if $DB \not\models a$.

An *update action* 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. Update actions represent changes to the database: $+a$ transforms DB to $DB \cup \{a\}$ and $-a$ transforms DB to $DB \setminus \{a\}$. A set of update actions $\mathcal{U}$ is *consistent* if it does not contain 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 $\mathcal{U}(DB)$. Following Cruz-Filipe [2016], we define an operation $\biguplus$ on consistent sets of update actions:

$$\mathcal{U}_1 \biguplus \mathcal{U}_2 = (\mathcal{U}_1 \cup \{\alpha \in \mathcal{U}_2 | \alpha^D \notin \mathcal{U}_1\}) \setminus \{\alpha \in \mathcal{U}_1 | \alpha^D \in \mathcal{U}_2\}.$$

This operation has the property that: if every action in $\mathcal{U}_1$ changes DB and every action in $\mathcal{U}_2$ changes $\mathcal{U}_1(DB)$, then

$$\left(\mathcal{U}_1 \biguplus \mathcal{U}_2\right)(DB) = \mathcal{U}_2(\mathcal{U}_1(DB)).$$

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. An allowed fix is to execute one or more of the α_i." A set of AICs represents constraints a database should adhere to and, in addition, which atoms can be changed in order to fix it. Intuitively, atoms can only be changed if there is some rule that allows it.

An AIC is *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. This is the case for most semantics of AICs (the notorious exception being the semantics of justified repairs [Caroprese and Truszczyński, 2011], which poses several other problems [Cruz-Filipe *et al.*, 2013]). In the current paper, we assume *all AICs are normal*. Extensions of the semantics we define for non-normal AICs can then 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. We now review semantics for AICs that have been defined with the intention to allow only changes allowed by one of the AICs.

Definition 2.3 ([Caroprese *et al.*, 2006])**.** 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.

Definition 2.4 ([Cruz-Filipe *et al.*, 2013])**.** 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 $U_{i-1} = \{\alpha_1, \ldots, \alpha_{i-1}\}$.

Definition 2.5 ([Caroprese and Truszczyński, 2011])**.** 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 $\mathcal{U}(DB)$.
- 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.
- $\mathcal{U}$ 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.

Marek and Truszczynski [1998] defined in the context of revision programming *the shifting property*; this property was later transferred to active integrity constraints [Caroprese and

Truszczyński, 2011]. Intuitively, a semantics for AICs possesses the shifting property if uniformly replacing some literals with their duals preserves the semantics at hand.

Definition 2.6. 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 2.7. 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 $S = DB$.

Lattices, Operators and Fixpoints 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*, respectively a *lower bound* of S if for every $s \in S$, it holds that $s \leq x$, respectively $x \leq s$. An element x is a *least upper bound* (respectively *greatest lower bound* of S) if it is an upper bound that is smaller than every other upper bound (resp. a lower bound that is greater than every other lower bound). If S has a least upper bound (resp. a greatest lower bound) we denote it $\mathrm{lub}(S)$ (resp. $\mathrm{glb}(S)$). As is custom, we sometimes call a greatest lower bound a *meet*, and a least upper bound a *join* and use the related notations $\bigwedge S = \mathrm{glb}(S)$, $x \wedge y = \mathrm{glb}(\{x, y\})$, $\bigvee S = \mathrm{lub}(S)$ and $x \vee y = \mathrm{lub}(\{x, y\})$. 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 $\bot$ and a greatest element $\top$.

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 $O(x) = x$. Every monotone operator O in a complete lattice has a least fixpoint, denoted $\mathrm{lfp}(O)$, 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.* [2015a] 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 $x \leq v$. Later, they generalized this notion to *partial* grounded fixpoints [Bogaerts *et al.*, 2015b]. 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.

Approximation Fixpoint Theory Given a lattice L, approximation fixpoint theory (AFT) [Denecker *et al.*, 2000] 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.* [2004] 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 $x = \mathrm{lfp}(A(\cdot, y)_1)$ and $y = \mathrm{lfp}(A(x, \cdot)_2)$, where $A(\cdot, y)_1$ denotes the operator $L \to L : x \mapsto A(x, y)_1$ and analogously for $A(x, \cdot)_2$.
- The *A-well-founded fixpoint* is the least precise 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 $x = \mathrm{lfp}(A(\cdot, x)_1)$.

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 [2007]:

Definition 2.8. 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 2.9. 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.* [1991]. 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. All terminal well-founded inductions of A have the A-well-founded fixpoint as limit [Denecker and Vennekens, 2007].

When we introduce semantics for AIC based on AFT in the next section, we provide examples of the different constructions considered here.

3 AFT-style Semantics for AICs

A Semantic Operator for AICs Recall that in this paper, we only consider normal AICs.

Given a fixed database DB, the sets of update actions $\mathcal{U}$ that are of interest to us are those such that **(i)** $\mathcal{U}$ is consistent and **(ii)** each action in $\mathcal{U}$ modifies DB. As argued by Cruz-Filipe [2016], the set of such sets is isomorphic to 2^{At}. Hence, from now on, we identify such a set with a subset of At (the atoms whose value is changed by $\mathcal{U}$). If $a \in At$ and DB is a database, we define $\mathrm{ch}\, a$ to be the update action $+a$ if $a \notin DB$ and $-a$ if $a \in DB$. Thus, in the above identification, a set of atoms $\overline{\mathcal{U}} \subseteq At$ is identified with the set

$$\mathcal{U} = \{\mathrm{ch}\, a \mid a \in \overline{\mathcal{U}}\}$$

of update actions. Usually, we omit the bar, and simply write $\mathcal{U}$ for the subset of At as well.

Following the principle of minimality of change [Winslett, 1990; Eiter and Gottlob, 1992], we prefer smaller sets of updates. The lattice we are interested in is thus $\langle 2^{At}, \subseteq\rangle$, where smaller elements correspond to "better" repairs.

Now consider a set of AICs η. Cruz-Filipe associated with such a set a semantic operator $\mathcal{T}_\eta : 2^{At} \to 2^{At}$ such that

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

Cruz-Filipe argued that a semantics for AICs based on grounded fixpoints of $\mathcal{T}_\eta$ coincides with the intuitions on a large set of examples and that it solves problems with several previously existing semantics. In the following paragraph, we define an approximator for $\mathcal{T}_\eta$ and hence, obtain a set of AFT-based semantics for AICs, based on similar intuitions.

An Approximator for $\mathcal{T}_\eta$ A *partial action set* is a mapping $\mathfrak{U} : At \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$. A partial action set $\mathfrak{U}$ is *two-valued* if $\mathfrak{U}(At) \subseteq \{\mathbf{t}, \mathbf{f}\}$; in this case, we identify $\mathfrak{U}$ with the action set $\{\mathrm{ch}\, a \mid \mathfrak{U}(a) = \mathbf{t}\}$. The intended reading of such a mapping is that $\mathfrak{U}(a)$ is true if a is changed by $\mathfrak{U}$, it is false if a is not changed by $\mathfrak{U}$ and it is unknown if $\mathfrak{U}$ leaves it open whether or not a is changed. Alternatively, a partial action set is identified with an element of $(2^{At})^c$ (as standard). The set of all partial action sets is denoted P. The truth order $\leq_t$ on three-valued truth values is defined by $\mathbf{f} \leq_t \mathbf{u} \leq_t \mathbf{t}$. The inverse of a truth value is $\mathbf{f}^{-1} = \mathbf{t}, \mathbf{t}^{-1} = \mathbf{f}, \mathbf{u}^{-1} = \mathbf{u}$.

A 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 in the database, $\mathfrak{DB}(a)$ is false if a is not in the database and $\mathfrak{DB}(a)$ is unknown otherwise.

If $\mathfrak{U}$ is a 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 3.1. 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[1] $\mathsf{nup}(r)$ in the partial interpretation $\mathfrak{DB}$ based on Kleene's truth tables [Kleene, 1938].

Using this notion, we define two additional values.

Definition 3.2. If $a \in At$, $\mathfrak{U}$ is a partial set of update actions and $\langle DB, \eta\rangle$ as before, we define

$$\mathsf{supp}^{ch}_{DB,\eta,\mathfrak{U}}(a) = \mathsf{supp}_{\mathfrak{U}(DB),\eta}(\mathrm{ch}\, a)$$
$$\mathsf{supp}^{keep}_{DB,\eta,\mathfrak{U}}(a) = \mathsf{supp}_{\mathfrak{U}(DB),\eta}((\mathrm{ch}\, a)^D)$$

As before, we often drop DB and η from the notation if they are clear from the context. Intuitively, $\mathsf{supp}^{ch}_{\mathfrak{U}}(a)$ is true if there is support for *changing* a after updating the database with $\mathfrak{U}$, i.e., if at least one rule r with $\mathsf{head}(r) = \mathrm{ch}\, a$ has $\mathsf{nup}(r)^{\mathfrak{U}(DB)} = \mathbf{t}$; $\mathsf{supp}^{ch}_{\mathfrak{U}}(a)$ is unknown if it is not true and there is at least one rule r with $\mathsf{head}(r) = \mathrm{ch}\, a$ that has $\mathsf{nup}(r)^{\mathfrak{U}(DB)} = \mathbf{u}$; otherwise $\mathsf{supp}^{ch}_{\mathfrak{U}}(a)$ is false. Similarly, $\mathsf{supp}^{keep}_{\mathfrak{U}}(a)$ expresses the support for *keeping* a as it is in the database (hence the name choice).

Example 3.3. Consider $DB = \emptyset$ and the following set η:

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

Consider $\mathfrak{U} = \{a \mapsto \mathbf{t}, b \mapsto \mathbf{t}, c \mapsto \mathbf{t}, d \mapsto \mathbf{u}\}$. Then $\mathsf{supp}^{ch}_{DB,\eta,\mathfrak{U}}(a) = \mathbf{t}$ and $\mathsf{supp}^{keep}_{DB,\eta,\mathfrak{U}}(a) = \mathbf{u}$. ▲

Definition 3.4. Given DB and η, we define an operator $\mathfrak{T}_{\langle DB,\eta\rangle} : P \to P$, as follows:

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

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

[1] Technically, $\mathsf{nup}(r)$ is a set of literal; we identify it with its conjunction here.

Definition 3.4 is motivated as follows. Assume $\mathfrak{U}$ is a partial update set containing 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 $\mathfrak{U}(a) = \mathbf{f}$, a is not an element of the (partial) update set at hand. The only way to add a to the update is if some rule supports changing a, which is captured by supp^{ch}. The case for $\mathfrak{U}(a) = \mathbf{t}$ is completely symmetrical, in this case the only reason for removing a from the update at hand is if there is some rule that supports keeping a. In the last case, where $\mathfrak{U}(a) = \mathbf{u}$, we have no information what the update does to a yet. In this case, we can derive that a *must* be in the update if we already have support for $\mathrm{ch}\, a$ and we are sure that there is no support for keeping a (for $\mathrm{ch}\, a^D$). Similarly, we can derive that a *must not* be in the update if we have support for $\mathrm{ch}\, a^D$ but not for $\mathrm{ch}\, a$. In all other cases, we derive nothing about a being in the update or not.

Proposition 3.5. *$\mathfrak{T}_\eta$ is an approximator of $\mathcal{T}_\eta$.*

Proof. (Sketch) The first step is to show that $\mathfrak{T}_\eta$ is $\leq_p$-monotone. Since Kleene-valuation is $\leq_p$-monotone, the functions that map $\mathfrak{U}$ to $\mathsf{supp}^{ch}_{\mathfrak{U}}(a)$ and $\mathsf{supp}^{keep}_{\mathfrak{U}}(a)$ for each $a \in At$ are also $\leq_p$-monotone. Monotonicity of $\mathfrak{T}_\eta$ then follows from case analysis on the definition of $\mathfrak{T}_\eta(\mathfrak{U})(a)$.

The second step is to show that $\mathcal{T}_\eta$ and $\mathfrak{T}_\eta$ coincide on two-valued sets, which again can be established by case analysis on the definition of $\mathfrak{T}_\eta(\mathcal{U})(a)$. □

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

Definition 3.6. 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.*, 2015b]. 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 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 3.13. To distinguish the two, we use the term *AFT-well-founded semantics*.

Grounded repairs were defined previously by Cruz-Filipe [2016]. All other classes of repairs defined in Definition 3.6 are newly introduced semantics by the current paper. We now illustrate these semantics by means of some examples.

Example 3.7. Consider the following set η of AICs:

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

with $DB = \emptyset$. The $\mathfrak{T}_\eta$-well-founded fixpoint can be computed as the limit of the following well-founded induction:

$$\mathfrak{U}_0 : a \mapsto \mathbf{u}, b \mapsto \mathbf{u}, c \mapsto \mathbf{u}.$$
$$\mathfrak{U}_1 = \mathfrak{T}_\eta(\mathfrak{U}_0) : a \mapsto \mathbf{t}, b \mapsto \mathbf{u}, c \mapsto \mathbf{u}$$
$$\mathfrak{U}_2 : a \mapsto \mathbf{t}, b \mapsto \mathbf{u}, c \mapsto \mathbf{f}$$
$$\mathfrak{U}_3 = \mathfrak{T}_\eta(\mathfrak{U}_2) : a \mapsto \mathbf{t}, b \mapsto \mathbf{t}, c \mapsto \mathbf{f},$$

where $\mathfrak{U}_2$ is an unfounded refinement of $\mathfrak{U}_1$ and all other refinements are application refinements. Note that unfoundedness refinements serve to minimize repairs. ▲

Example 3.8. Consider the following set η of AICs:

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

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.

The $\mathfrak{T}_\eta$-well-founded fixpoint can be computed as the limit of the following well-founded induction:

$$\mathfrak{U}_0 : a \mapsto \mathbf{u}, b \mapsto \mathbf{u}, c \mapsto \mathbf{u}.$$
$$\mathfrak{U}_1 : a \mapsto \mathbf{f}, b \mapsto \mathbf{f}, c \mapsto \mathbf{u}.$$
$$\mathfrak{U}_2 = \mathfrak{T}_\eta(\mathfrak{U}_1) : a \mapsto \mathbf{f}, b \mapsto \mathbf{f}, c \mapsto \mathbf{t}.$$

Here, the first refinement is an unfoundedness refinement. In can be verified that $\mathfrak{U}_0$ is a fixpoint of $\mathfrak{T}_\eta$ and hence is the Kripke-Kleene repair. Again, the AFT-well-founded repair is two-valued and corresponds to the intuitions. ▲

As can be expected, not every set of AICs has a two-valued well-founded repair. That would simply be too much to ask. 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 3.9. Consider the following set η of AICs:

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

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 $\{+b\}$. In can be verified that these two intended repairs are indeed stable and that the AFT-well-founded repair maps all atoms to $\mathbf{u}$ (hence it approximates the two intended repairs). ▲

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

Proposition 3.10. *$\mathfrak{T}_\eta$, and hence also $\mathcal{T}_\eta$, is invariant under shifting, i.e., for each set $S \subseteq At$:*

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

Corollary 3.11. *All AFT-style semantics for AICs have the shifting property.*

Proposition 3.12. *If the AFT-well-founded repair is two-valued, it is also well-founded (as defined by Cruz-Filipe* et al. *[2013]).*

Example 3.13. The converse of Proposition 3.12 does not hold. Consider for instance the following set η of AICs (based on Example 18 of Cruz-Filipe [2016])

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

and $DB = \emptyset$. In this case, the AFT-well-founded repair is $\mathcal{U}_{AFT} : a \mapsto \mathbf{t}, b \mapsto \mathbf{f}, c \mapsto \mathbf{t}$. This is clearly the intended repair. Indeed, the first rule dictates that a needs to be changed (here: added) no matter what. The second rule then becomes void (there is never a reason to add b) and the last rule dictates that also c must be added. However, $a \mapsto \mathbf{t}, b \mapsto \mathbf{t}, c \mapsto \mathbf{f}$ is also a well-founded repair, obtained by first applying the second rule and then the first one. ▲

Proposition 3.14. *All $\mathfrak{T}_\eta$-stable repairs are justified.*

Example 3.15. The converse of Proposition 3.14 does not hold. Consider the following set η of AICs.

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

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, recall that we identify a partial action set with an element of $(2^{At})^c$, e.g., $\{a \mapsto \mathbf{u}, b \mapsto \mathbf{u}\}$ is identified with $(\emptyset, \{a, b\})$. It now 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 3.14 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 3.16. 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 3.17. *If η is unipolar, then each justified repair of $\langle DB, \eta\rangle$ is $\mathfrak{T}_\eta$-stable.*

Using Lemma 19 of Cruz-Filipe [2016], which states that all justified repairs are grounded, we easily find how justified repairs and the AFT-well-founded repair relate.

Corollary 3.18. *The AFT-well-founded repair and the Kripke-Kleene repair approximate all justified repairs.*

4 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 4.1. *$\mathfrak{T}_\eta$ is computable in polynomial time.*

As a consequence, we get the following complexity results.

Proposition 4.2. *The Kripke-Kleene repair for $\langle DB, \eta\rangle$ is computable in polynomial time.*

Proposition 4.3. *The ATF-well-founded repair for $\langle DB, \eta\rangle$ is computable in polynomial time.*

Proposition 4.4. *The task of checking if a database $\langle DB, \eta\rangle$ has a stable repair is NP-complete.*

For each of our semantics, complexity is the same as the complexity of its counterpart in (normal) logic programming. This illustrates that the added expressivity (allowing AICs that are not unipolar) does not result in added complexity.

5 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.* [2004]. 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 are inconsistencies. 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 [Bi *et al.*, 2014]; 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* [Bogaerts *et al.*, 2017] 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 [Marek and Truszczynski, 1998]. Caroprese and Truszczyński [2011] 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.

References

[Abiteboul, 1988] Serge Abiteboul. Updates, a new frontier. In Marc Gyssens, Jan Paredaens, and Dirk van Gucht, editors, *ICDT*, volume 326 of *LNCS*, pages 1–18. Springer, 1988.

[Bi *et al.*, 2014] Yi Bi, Jia-Huai You, and Zhiyong Feng. A generalization of approximation fixpoint theory and application. In *Proceedings of RR*, pages 45–59, 2014.

[Bogaerts *et al.*, 2015a] Bart Bogaerts, Joost Vennekens, and Marc Denecker. Grounded fixpoints and their applications in knowledge representation. *AIJ*, 224:51–71, 2015.

[Bogaerts *et al.*, 2015b] Bart Bogaerts, Joost Vennekens, and Marc Denecker. Partial grounded fixpoints. In *Proceedings of IJCAI*, pages 2784–2790, 2015.

[Bogaerts *et al.*, 2016] Bart Bogaerts, Joost Vennekens, and Marc Denecker. On well-founded set-inductions and locally monotone operators. *ACM Trans. Comput. Logic*, 17(4):27:1–27:32, September 2016.

[Bogaerts *et al.*, 2017] Bart Bogaerts, Joost Vennekens, and Marc Denecker. Safe inductions: An algebraic study. In *Proceedings of IJCAI*, 2017. (to appear).

[Brewka and Woltran, 2010] Gerhard Brewka and Stefan Woltran. Abstract dialectical frameworks. In *Proceedings of KR*, pages 102–111, 2010.

[Caroprese and Truszczyński, 2011] Luciano Caroprese and Mirosław Truszczyński. Active integrity constraints and revision programming. *TPLP*, 11(6):905–952, 2011.

[Caroprese *et al.*, 2006] Luciano Caroprese, Sergio Greco, Cristina Sirangelo, and Ester Zumpano. Declarative semantics of production rules for integrity maintenance. In *Proceedings of ICLP*, pages 26–40, 2006.

[Cruz-Filipe *et al.*, 2013] Luís Cruz-Filipe, Graça Gaspar, Patrícia Engrácia, and Isabel Nunes. Computing repairs from active integrity constraints. In *TASE 2013*, pages 183–190. IEEE Computer Society, 2013.

[Cruz-Filipe *et al.*, 2015] Luís Cruz-Filipe, Michael Franz, Artavazd Hakhverdyan, Marta Ludovico, Isabel Nunes, and Peter Schneider-Kamp. repAIrC: A tool for ensuring data consistency by means of active integrity constraints. In *Proceedings of KMIS*, volume 3, pages 17–26, 2015.

[Cruz-Filipe, 2014] Luís Cruz-Filipe. Optimizing computation of repairs from active integrity constraints. In *Proceedings of FoIKS*, pages 361–380, 2014.

[Cruz-Filipe, 2016] Luís Cruz-Filipe. Grounded fixpoints and active integrity constraints. In *Technical communications of ICLP*, pages 11.1–11.14, 2016.

[Denecker and Vennekens, 2007] Marc Denecker and Joost Vennekens. Well-founded semantics and the algebraic theory of non-monotone inductive definitions. In *Proceedings of LPNMR*, pages 84–96, 2007.

[Denecker *et al.*, 2000] Marc Denecker, Victor Marek, and Mirosław Truszczyński. Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In *Logic-Based Artificial Intelligence, Springer*, volume 597, pages 127–144, 2000.

[Denecker *et al.*, 2003] Marc Denecker, Victor Marek, and Mirosław Truszczyński. Uniform semantic treatment of default and autoepistemic logics. *AIJ*, 143(1):79–122, 2003.

[Denecker *et al.*, 2004] Marc Denecker, Victor Marek, and Mirosław Truszczyński. Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Information and Computation*, 192(1):84–121, July 2004.

[Dung, 1995] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *AI*, 77(2):321 – 357, 1995.

[Eiter and Gottlob, 1992] Thomas Eiter and Georg Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *AIJ*, 57(2-3):227–270, 1992.

[Flesca *et al.*, 2004] Sergio Flesca, Sergio Greco, and Ester Zumpano. Active integrity constraints. In *Proceedings of SIGPLAN*, pages 98–107, 2004.

[Kleene, 1938] Stephen Cole Kleene. On notation for ordinal numbers. *The Journal of Symbolic Logic*, 3(4):150–155, 1938.

[Marek and Truszczynski, 1998] V. Wiktor Marek and Miroslaw Truszczynski. Revision programming. *Theor. Comput. Sci.*, 190(2):241–277, 1998.

[Moore, 1985] Robert C. Moore. Semantical considerations on nonmonotonic logic. *AIJ*, 25(1):75–94, 1985.

[Przymusinski and Turner, 1997] Teodor C. Przymusinski and Hudson Turner. Update by means of inference rules. *Journal of Logic Programming*, 30(2):125–143, 1997.

[Reiter, 1980] Raymond Reiter. A logic for default reasoning. *AIJ*, 13(1-2):81–132, 1980.

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

[Teniente and Olivé, 1995] Ernest Teniente and Antoni Olivé. Updating knowledge bases while maintaining their consistency. *VLDB Journal*, 4(2):193–241, 1995.

[van Emden and Kowalski, 1976] Maarten H. van Emden and Robert A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, 1976.

[Van Gelder *et al.*, 1991] Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, 1991.

[Vennekens *et al.*, 2006] Joost Vennekens, David Gilis, and Marc Denecker. Splitting an operator: Algebraic modularity results for logics with fixpoint semantics. *ACM Trans. Comput. Log.*, 7(4):765–797, 2006.

[Vennekens *et al.*, 2007a] Joost Vennekens, Maarten Mariën, Johan Wittocx, and Marc Denecker. Predicate introduction for logics with a fixpoint semantics. Part I: Logic programming. *Fundamenta Informaticae*, 79(1-2):187–208, September 2007.

[Vennekens *et al.*, 2007b] Joost Vennekens, Maarten Mariën, Johan Wittocx, and Marc Denecker. Predicate introduction for logics with a fixpoint semantics. Part II: Autoepistemic logic. *Fundamenta Informaticae*, 79(1-2):209–227, September 2007.

[Widom and Ceri, 1996] Jennifer Widom and Stefano Ceri, editors. *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann, 1996.

[Winslett, 1990] Marianne Winslett. *Updating Logical Databases*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.