Computing Repairs from Active Integrity Constraints

Luís Cruz-Filipe
Patrícia Engrácia
Escola Superior Náutica Infante D. Henrique
Paço d'Arcos, Portugal
CMAF, Lisboa, Portugal

Graça Gaspar
Isabel Nunes
Faculdade de Ciências, Universidade de Lisboa
Lisboa, Portugal
LabMag, Lisboa, Portugal

Abstract—**Repairing an inconsistent knowledge base is a well-known problem for which several solutions have been proposed and implemented in the past. In this paper, we start by looking at databases with active integrity constraints – consistency requirements that also indicate how the database should be updated when they are not met – as introduced by Caroprese et al. We show that the different kinds of repairs considered by those authors can be effectively computed by searching for leaves of specific kinds of trees. Although these computations are in general not very efficient (deciding the existence of a repair for a given database with active integrity constraints is NP-complete), on average the algorithms we present make significant reductions on the number of nodes in the search tree. Finally, these algorithms also give an operational characterization of different kinds of repairs that can be used when we extend the concept of active integrity constraints to the more general setting of knowledge bases.**

I. Introduction

Database dependencies have been since long a main tool in the fields of relational and deductive databases [1], [3], used to express integrity constraints on databases. They formalize relationships between data in the database that need to be satisfied so that the database conforms to its intended meaning.

Whenever an integrity constraint is violated, the database must be repaired in order to regain consistency. Repairing a database consists in inserting or deleting tuples (update actions), so that the resulting database satisfies all of the integrity constraints. In most cases, there are many different candidate sets of update actions that can be used to repair a database, leading to different revised consistent databases. Being able to restrict the set of database "repairs" to those considered the most "adequate", in some sense, is therefore an important task in order to automate maintenance of the consistency of databases.

Minimality of change – not containing unnecessary actions – is commonly accepted as a very important characteristic of a repair [6], [10], however it is not enough to narrow down the set of possible repairs sufficiently. It is also important that the database designer may be able to express, for each given integrity constraint, his preferred repair action(s).

Active integrity constraints (denoted hereafter by AICs), first proposed in [7], are special forms of production rules that encode both an integrity constraint and preferred update actions to be performed whenever the former is violated. In [4] the authors study AICs and define a declarative semantics for them, defining the class of founded repairs. That work is extended in [5], and, as a result of the effort to clarify the relation between the semantics of AICs and revision programming [8], the more restricted class of *justified* repairs is defined. Informally, justified repairs are the repairs that are the most strongly grounded in the given database and the given set of AICs, that is, those considered most "adequate" in the sense that they result strictly from combinations of the preferences expressed by the database designer for each of the integrity constraints and from the principle of minimal change.

The most common approach to processing integrity constraints in database management systems is to use active rules (a kind of event-condition-action rules, or ECAs [9]), for which rule processing algorithms have been proposed and a procedural semantics has been defined. However, in opposition to AICs, their lack of declarative semantics makes it difficult to understand the behaviour of multiple ECAs acting together and to evaluate, in a principled way, the rule-processing algorithms.

Our work starts from the definitions of founded and justified repairs, given by Caroprese at al. [5], and shows how those repairs can be effectively computed by searching for leaves of specific kinds of trees, thus giving an operational characterization of those kinds of repairs. Although these computations are in general not very efficient (deciding the existence of a repair for a given database with active integrity constraints is NP-complete), on average the algorithms we present make significant reductions on the number of nodes in the search tree.

To extend active integrity constraints and the computations of repairs from the context of databases to the more general context of knowledge bases, one has to take into account that the violation of integrity constraints can originate not just from data explicitly represented, but also from data derived from rules or axioms in the knowledge base. Therefore, we introduce the notion of *generalized active integrity constraint*, which relaxes the relation between the two parts of an active integrity constraint – the integrity constraint part and the update action part – by taking into account the structure of the knowledge base itself. We then discuss how the various kinds of trees that we have defined for computing repairs of databases can (or cannot) be generalized to knowledge bases to capture the most adequate repairs in that more general context.

The remainder of the paper is organized as follows: Section II summarizes the fundamental notions underlying AICs; Section III shows how repairs can be straightforwardly computed by a tree-generating algorithm; Section IV restricts the construction of descendant nodes in the previous algorithm to obtain trees that correspond to the notions of founded and justified repair. This correspondence is not precise – the trees compute some extra (non-founded or non-justified) weak repairs – but this is to be expected, since our algorithm runs in non-deterministic polynomial time, whereas the problem of existence of founded or justified repairs is Σ_P^2-complete. Section V generalizes AICs to a more encompassing setting of knowledge bases; finally, Section VI summarizes the achievements described in this paper.

II. Background

Constraint checking has been an area of active research since long, with the most effort being put in detecting inconsistencies, preventing violation of constraints, and answering queries over inconsistent databases in a meaningful way. Among the work in this field, a substancial amount deals with repairing databases that do not satisfy the desired constraints.

In this paper we focus on a particular approach: that of *active integrity constraints* (AICs, for short), as proposed in [5]. For space limitations, we will not discuss other approaches to the problem of constraint checking that are not directly relevant for this work. In the last section, we discuss how AICs can be applied in the more general setting of knowledge bases. Throughout the whole presentation, we strictly follow the notation introduced in [5]; in particular, all definitions in this section are taken from that article.

In the framework of AICs, databases are considered to be subsets of some finite set $\mathcal{A}t$ of propositional atoms, and integrity constraints are clauses of the form $L_1, \ldots, L_m \supset \bot$, where L_i are all literals in the propositional language generated by $\mathcal{A}t$, with intended meaning that the conjunction of all L_is must not hold.

Any database $\mathcal{I}$, being a subset of $\mathcal{A}t$, can be regarded as a propositional interpretation. $\mathcal{I}$ *entails* literal L, $\mathcal{I} \models L$, if L is a and $a \in \mathcal{I}$ or L is not a and $a \notin \mathcal{I}$. $\mathcal{I}$ *satisfies* integrity constraint r , $\mathcal{I} \models r$, if $\mathcal{I} \not\models L$ for some L in r. Moreover, $\mathcal{I}$ satisfies a set η of integrity constraints, $\mathcal{I} \models \eta$, if it satisfies each integrity constraint in η.

Whenever $\mathcal{I}$ does not satisfy a set η of integrity constraints, $\mathcal{I}$ may be updated by inserting and deleting atoms, through *update actions* of the form $+a$ and $-a$, where $a \in \mathcal{A}t$, stating that a is to be inserted in or deleted from $\mathcal{A}t$, respectively. As expected, the result of updating $\mathcal{I}$ by means of a set $\mathcal{U}$ of update actions is the database $\mathcal{I} \circ \mathcal{U} = (\mathcal{I} \cup \{a \mid +a \in \mathcal{U}\}) \setminus \{a \mid -a \in \mathcal{U}\}$. A set of update actions is *consistent* if it does not contain both $+a$ and $-a$, for any $a \in \mathcal{A}t$.

Given a database $\mathcal{I}$ and a set η of integrity constraints, the problem of database repair is to determine how to update $\mathcal{I}$ so that it satisfies η.

Definition 1 *Let $\mathcal{I}$ be a database and η a set of integrity constraints. A* weak repair *for $\langle \mathcal{I}, \eta \rangle$ is a consistent set $\mathcal{U}$ of update actions such that:*

- $(\{+a \mid a \in \mathcal{I}\} \cup \{-a \mid a \in \mathcal{A}t \setminus \mathcal{I}\}) \cap \mathcal{U} = \emptyset$ *($\mathcal{U}$ consists of "essential" update actions only);*
- $\mathcal{I} \circ \mathcal{U} \models \eta$ *(constraint enforcement).*

A repair *for $\langle \mathcal{I}, \eta \rangle$ is a weak repair $\mathcal{U}$ such that, for every $\mathcal{U}' \subseteq \mathcal{U}$, if $\mathcal{I} \circ \mathcal{U}' \models \eta$, then $\mathcal{U}' = \mathcal{U}$ (minimality of change).*

The problem of existence of a weak repair is NP-complete [5], and so is the one of existence of a repair.[1]

The formalism of AICs was designed to address the problem of guiding the process of selection of a repair. The literal corresponding to an action α, $\mathsf{lit}(\alpha)$, is a if $\alpha = +a$ and not a if $\alpha = -a$, while the update action corresponding to a literal a, $\mathsf{ua}(a)$, is $+a$ if $L = a$ and $-a$ if $L = \text{not } a$. The *dual* of a is not a, and conversely. Denoting the dual of L by L^D, an *active integrity constraint* is an expression r of the form $L_1, \ldots, L_m \supset \alpha_1 \mid \ldots \mid \alpha_k$ where the L_i (in the *body* of r, $\mathsf{body}\,(r)$) are literals and the α_j (in the *head* of r, $\mathsf{head}\,(r)$) are update actions, and

$$\left\{\mathsf{lit}(\alpha_1)^D, \ldots, \mathsf{lit}(\alpha_k)^D\right\} \subseteq \{L_1, \ldots, L_m\} \ .$$

The set $\mathsf{lit}(\mathsf{head}\,(r))^D$ contains the *updatable* literals of r. The *non-updatable* literals of r form the set $\mathsf{nup}(r) = \mathsf{body}\,(r) \setminus \mathsf{lit}\,(\mathsf{head}\,(r))^D$.

The head of a rule is meant to be given a disjunctive interpretation; therefore, $\mathcal{I} \models r$ if $\mathcal{I} \models \mathsf{lit}(\alpha)$ for some α in the head of r whenever $\mathcal{I} \models \mathsf{nup}(r)$.[2]

In order to distinguish repairs that are "supported" by a given set of AICs, in the sense that they somehow comply with the update actions in the rules, Caroprese et al. introduced the notion of founded repairs.

Definition 2 *Let $\mathcal{I}$ be a database, η a set of active integrity constraints and $\mathcal{U}$ a consistent set of update actions.*

- *An update action α is* founded *w.r.t. $\langle \mathcal{I}, \eta \rangle$ and $\mathcal{U}$ if there is $r \in \eta$ such that $\alpha \in \mathsf{head}\,(r)$, $\mathcal{I} \circ \mathcal{U} \models \mathsf{nup}(r)$, and $\mathcal{I} \circ \mathcal{U} \models \mathsf{lit}(\beta)^D$ for every $\beta \in \mathsf{head}\,(r) \setminus \{\alpha\}$.*
- *$\mathcal{U}$ is* founded *w.r.t. $\langle \mathcal{I}, \eta \rangle$ if all of its elements are founded w.r.t. $\langle \mathcal{I}, \eta \rangle$ and $\mathcal{U}$.*
- *$\mathcal{U}$ is a* founded (weak) repair *for $\langle \mathcal{I}, \eta \rangle$ if $\mathcal{U}$ is a* (weak) repair *for $\langle \mathcal{I}, \eta \rangle$ and $\mathcal{U}$ is* founded *w.r.t. $\langle \mathcal{I}, \eta \rangle$.*

We will use the following equivalent characterization of founded update action: α is founded w.r.t. $\langle \mathcal{I}, \eta \rangle$ and $\mathcal{U}$ if there is $r \in \eta$ such that $\alpha \in \mathsf{head}\,(r)$ and $\mathcal{I} \circ \mathcal{U} \models L$ for every $L \in \mathsf{body}\,(r) \setminus \{\mathsf{lit}(\alpha)\}$.

Notice that founded repairs are not minimal founded weak repairs, but rather founded weak repairs that are minimal

[1]Due to space limitations, we defer the presentation of examples of these concepts to the next section.

[2]Because of the syntactic restrictions of AICs, this is actually equivalent to saying that $\mathcal{I} \not\models L$ for some $L \in \mathsf{body}\,(r)$; but the form presented above reflects the "operational" view of AICs: if the non-updatable atoms in the body of a rule hold, then some action must be taken to guarantee that the integrity constraint is satisfied.

among *all* repairs. An example of a minimal founded weak repair that is not a founded repair can be found in [5].

While the problem of existence of a weak founded repair is again NP-complete, the problem of existence of founded repairs has higher complexity – it is Σ_P^2-complete.

Situations may arise in which the support for the actions in a founded repair is circular – an action α is supported by a rule r_α whose body holds because of another action β, in turn supported by a rule r_β whose body holds because of α.

In order to disallow these circular dependencies, [5] introduced justified repairs for a set of AICs. This definition requires some auxiliary notions.

A set $\mathcal{U}$ of update actions is *closed* under r if $\mathsf{nup}(r) \subseteq \mathsf{lit}(\mathcal{U})$ implies $\mathsf{head}\,(r) \cap \mathcal{U} \neq \emptyset$. $\mathcal{U}$ is closed under a set η of AICs if it is closed under every r in η. Note that, in particular, every founded weak repair for $\langle \mathcal{I}, \eta \rangle$ is closed under η.

An update action $+a$ (respectively, $-a$) is a *no-effect* action w.r.t. $(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$ if $a \in \mathcal{I} \cap (\mathcal{I} \circ \mathcal{U})$ (respectively $a \notin \mathcal{I} \cup (\mathcal{I} \circ \mathcal{U})$) – intuitively, the action does not change $\mathcal{I}$. The set of all no-effect actions with respect to $(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$ is denoted by $\mathsf{ne}\,(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$.

A set $\mathcal{U}$ of update actions is then a justified action set if it is precisely the set of update actions forced or "justified" by the set of AICs and the no-effect actions with respect to the database in its two versions, the pre-repair and the post-repair [5]. The formal definition follows.

Definition 3 *Let $\mathcal{I}$ be a database and η a set of AICs. A consistent set $\mathcal{U}$ of update actions is a* justified action set *for $\langle \mathcal{I}, \eta \rangle$ if $\mathcal{U}$ is a minimal set of update actions containing $\mathsf{ne}\,(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$ and closed under η.*

If $\mathcal{U}$ is a justified action set for $\langle \mathcal{I}, \eta \rangle$, then $\mathcal{U} \setminus \mathsf{ne}\,(\mathcal{I}, \mathcal{I} \circ \mathcal{U})$ is a justified weak repair for $\langle \mathcal{I}, \eta \rangle$.

In particular, it is shown in [5] that justified repairs are always founded. However, the formal definition of justified repairs is not very intuitive. Part of the purpose of this paper is to give an operational characterization of justified repairs and show that it coincides with the above formal definition.

Existence of justified weak repairs or justified repairs for $\langle \mathcal{I}, \eta \rangle$ is again a Σ_P^2-complete problem. However, [5] also introduces *normalized* active integrity constraints: these are AICs whose head contains at most one action. The authors show that, when η is a set of normalized AICs, the problem of existence of justified (weak) repairs for $\langle \mathcal{I}, \eta \rangle$ becomes NP-complete.

III. Finding repairs of inconsistent databases

In this section we present an algorithm for computing repairs of a database $\mathcal{I}$ not satisfying a set of integrity constraints η. The idea is simple: we look at each integrity constraint that is not being satisfied and attempt to fix the database.

At this stage, we are not concerned specifically with active integrity constraints (although all the work in this section applies to them). Therefore, if r is not satisfied, then negating *any* of the literals in $\mathsf{body}\,(r)$ is a possibility of repair. We therefore build a tree as follows.

Definition 4 *Let $\mathcal{I}$ be a database and η be a finite*[3] *set of (active) integrity constraints. The* repair tree *of $\langle \mathcal{I}, \eta \rangle$ is constructed as follows.*

- *Each node is a set of repair actions.*
- *A node is* consistent *if it does not contain both an action and its dual.*
- *Each edge is labeled with a rule.*
- *The root of the tree is the empty set $\emptyset$.*
- *For each consistent node n and rule r, if $\mathcal{I} \circ n \not\models r$ then for each $L \in \mathsf{body}\,(r)$ the set $n' = n \cup \left\{\mathsf{ua}(L)^D\right\}$ is a child of n, with the edge from n to n' labeled by r.*

In order to illustrate this construction, we discuss some examples.

Example 1 *Consider the following set η of active integrity constraints.*

$$r_1 : a, b \supset -a \mid -b \qquad r_2 : a, \mathsf{not}\ b \supset -a$$
$$r_3 : \mathsf{not}\ a, b \supset -b$$

Given the database $\mathcal{I} = \{a, b\}$, the repair tree for $\langle \mathcal{I}, \eta \rangle$ is the following.

$$\begin{tikzcd}
 & & \emptyset \arrow[dl, "r_1"'] \arrow[d, "r_1"] & \\
 & \{-a\} \arrow[dl, "r_3"'] \arrow[d, "r_3"] & \{-b\} \arrow[d, "r_2"'] \arrow[dr, "r_2"] & \\
\{-a, +a\} & \{-a, -b\} & \{-b, -a\} & \{-b, +b\} \\
\times & & & \times
\end{tikzcd}$$

Observe that both consistent leaves contain a repair (in fact, the only one) for $\langle \mathcal{I}, \eta \rangle$.

Example 2 *Consider the following set η of active integrity constraints.*

$$r_1 : a \supset -a \qquad r_2 : a, b \supset -b$$

Given the database $\mathcal{I} = \{a, b\}$, the repair tree for $\langle \mathcal{I}, \eta \rangle$ is the following.

$$\begin{tikzcd}
 & \emptyset \arrow[dl, "r_1"'] \arrow[d, "r_2"] \arrow[dr, "r_2"] & \\
\{-a\} & \{-a\} & \{-b\} \arrow[d, "r_1"] \\
 & & \{-b, -a\}
\end{tikzcd}$$

In this case there are three leaves, all consistent. The leaves $\{-a\}$ correspond to the only repair for $\langle \mathcal{I}, \eta \rangle$, while $\{-a, -b\}$ is a weak repair for the same pair.

[3]From a practical perspective, it does not make sense to assume that either $\mathcal{I}$ or η is infinite; still, finiteness of η is relevant for some theoretical properties we discuss. We will assume without stating explicitly that this is the case.

We now prove some simple results about repair trees.

Lemma 1 *The repair tree for $\langle \mathcal{I}, \eta \rangle$ is finite.*

Proof. First note that the number of rules not satisfied at each node is always finite. Furthermore, each rule has a finite number of literals in its body. Therefore, the number of descendants of any node is finite.

If there is an edge from n to n' labeled by r, then (by construction of n') rule r is satisfied in n' and in any of its descendants. Since the number of rules is finite, this means that the depth of the tree is also finite.

Therefore the repair tree for $\langle \mathcal{I}, \eta \rangle$ is finite. $\square$

Note that the number of nodes in this tree can be enormous. If η has k rules each with exactly m literals, and all rules are independent (in the sense that fixing one does not affect any of the others), then all leaves of the tree lie at depth k and nodes at height h have $m \times (k-h)$ descendants, giving a total number of nodes of magnitude $O\left(m^k k!\right)$. We can improve this substantially, however, by identifying nodes corresponding to the same set, thereby reducing the number of nodes in the tree to 2^p, where p is the number of different literals occurring in the bodies of rules in η. In either case, the total number of nodes does not depend on $\mathcal{I}$, as was already noted in [5].

A more interesting result is the following.

Lemma 2 *Every consistent leaf of the repair tree for $\langle \mathcal{I}, \eta \rangle$ is a weak repair for $\langle \mathcal{I}, \eta \rangle$.*

Proof. By construction, if there is a rule in η that is not satisfied in node n, then n is not a leaf. Therefore, leaves satisfy all rules in η, and hence correspond to weak repairs for $\langle \mathcal{I}, \eta \rangle$. $\square$

In particular, if the set of constraints η is inconsistent, then the repair tree for $\langle \mathcal{I}, \eta \rangle$ has no consistent leaves.

Note that an inconsistent node can never produce consistent leaves. Therefore, from this point onwards we will omit inconsistent nodes from the repair tree. Consistency of a generated node is decidable in linear time (on the number of repair actions in the node), so this can only improve the complexity of all algorithms discussed below.

The reason for calling this tree a repair tree is more profound: *every* repair for $\langle \mathcal{I}, \eta \rangle$ corresponds to a leaf in the repair tree; so by constructing the repair tree for $\langle \mathcal{I}, \eta \rangle$ we can immediately decide whether there are repairs for $\langle \mathcal{I}, \eta \rangle$, and determine who they are.

Theorem 1 *Let $\mathcal{U}$ be a repair for $\langle \mathcal{I}, \eta \rangle$. Then there is a branch of the repair tree for $\langle \mathcal{I}, \eta \rangle$ that ends with leaf $\mathcal{U}$.*

Proof. Note that $\mathcal{U}$ is finite. We show that there is a branch of the tree whose nodes are $\emptyset = \mathcal{U}_0, \mathcal{U}_1, \ldots, \mathcal{U}_n = \mathcal{U}$, i.e. $\mathcal{U}_{i+1}$ is obtained from $\mathcal{U}_i$ by adding an action in $\mathcal{U} \setminus \mathcal{U}_i$.

The only issue is showing that we can always find $\mathcal{U}_{i+1}$ as described, for $i < n$. Since $\mathcal{U}$ is a repair (and not a weak repair), $\mathcal{U}_i$ cannot be a repair; therefore, there are some rules which are not satisfied in $\mathcal{I} \circ \mathcal{U}_i$. Let r be one of these rules; if the body of r does not contain literals fixed by update actions in $\mathcal{U} \setminus \mathcal{U}_i$, then $\mathcal{I} \circ \mathcal{U}$ does not satisfy r, which is absurd since $\mathcal{U}$ is a repair. Therefore, such a $\mathcal{U}_{i+1}$ exists. $\square$

This result also provides an alternative proof of the already known fact that deciding whether there is a (weak) repair for $\langle \mathcal{I}, \eta \rangle$ can be solved in non-deterministic polynomial time. Note that by finding all leaves of the repair tree we can also identify all repairs for $\langle \mathcal{I}, \eta \rangle$ – namely, they are the leaves that are not a superset of any other leaf.

IV. Active integrity constraints and justified repairs

As the authors of [5] discuss, the purpose of introducing active integrity constraints (AICs) is to allow one to express preferences on how the database should be repaired when the integrity constraints are not met. The construction presented in the previous section does not take these preferences into account, as it ignores the head of rules.

To deal with these preferences, the authors introduced the notion of founded repair: a repair that is "compatible" with the heads of the AICs. Intuitively, one should be able to compute founded repairs by looking at the AICs currently being violated and following the suggestions in their heads. However, even this does not work, as the following example shows.

Example 3 *Consider the database $\mathcal{I} = \{a, b\}$ together with the following set η of active integrity constraints.*

$$r_1 : a, \mathsf{not}\ b \supset -a \qquad r_3 : a, \mathsf{not}\ c \supset +c$$
$$r_2 : \mathsf{not}\ a, b \supset -b \qquad r_4 : b, \mathsf{not}\ c \supset +c$$

Following the heads of the rules that are not satisfied, one is led to the set $\{+c\}$, which is a founded repair for $\langle \mathcal{I}, \eta \rangle$. However, the set $\{-a, -b\}$ is also *a founded repair for the same pair.*

The issue is – again – that, in this set, the action $-a$ is supported by rule r_1 because of the presence of $-b$, which in turn is supported by rule r_2 because of the presence of $-a$. One might argue that $\{+c\}$ is a better repair: indeed, following the authors' original route, it can be seen that $\{+c\}$ is a *justified* repair for $\langle \mathcal{I}, \eta \rangle$, while $\{-a, -b\}$ is not.

Simply restricting the construction of the repair tree to reflect the preferences expressed by the heads of active integrity constraints, however, is not enough to obtain only justified repairs. In some situations, we may still obtain founded repairs that are not justified, as the following example shows.

Example 4 *Consider the following variation of Example 1, where η is*

$$r_1' : a, b \supset -a \qquad r_2 : a, \mathsf{not}\ b \supset -a \qquad r_3 : \mathsf{not}\ a, b \supset -b$$

Constructing a variation of the repair tree of $\langle\mathcal{I},\eta\rangle$ containing only the edges that connect nodes differing by an action on the head of the rule labeling them, one obtains the following tree.

$$\begin{tikzcd}
\emptyset \arrow[d, "r_1'"] \\
\{-a\} \arrow[d, "r_3"] \\
\{-a,-b\}
\end{tikzcd}$$

Notice that we also omitted edges leading to inconsistent nodes, as discussed earlier. The set $\{-a,-b\}$ is a founded repair for $\langle\mathcal{I},\eta\rangle$, but, as discussed in [5], it is not justified.

The intuition behind justified repairs seems to be that one should go one step further: the rule leading to the introduction of an action in a would-be repair should be the same rule supporting that action in the final repair. This turns out to be too restrictive, however, as one can easily find examples where justified repairs exist but they would not be found by this approach – Example 1 is such an example. Since the definition of justified repair does not directly refer to support for actions, but to justified action sets, which in turn look at the non-updatable literals in the rules' bodies, we use a technique from [2] to keep track not only of the actions introduced at each step, but also of the non-updatable assumptions made when introducing the actions. The justified repair tree for $\langle\mathcal{I},\eta\rangle$ is then defined as follows.

Definition 5 *Let $\mathcal{I}$ be a database and η be a set of active integrity constraints. The* justified repair tree *of $\langle\mathcal{I},\eta\rangle$ is constructed as follows.*

- *Each node n is a pair of sets of repair actions $\mathcal{U}_n,\mathcal{J}_n$.*
- *Each edge is labeled with a rule.*
- *The root of the tree is $\emptyset,\emptyset$.*
- *For each node n and rule r, if $\mathcal{I}\circ\mathcal{U}_n\not\models r$, then for each $\alpha\in\mathsf{head}\,(r)$ there is a descendant n' of n, with the edge from n to n' labeled by r, where:*
  - $\mathcal{U}_{n'}=\mathcal{U}_n\cup\{\alpha\}$;
  - $\mathcal{J}_{n'}=(\mathcal{J}_n\cup\{\mathsf{ua}(\mathsf{nup}(r))\})\setminus\mathcal{U}_n$.
  - *If $\mathcal{U}_{n'}$ is inconsistent, then n' is removed.*
  - *If $\mathcal{U}_{n'}\cap(\mathcal{J}_{n'})^D\neq\emptyset$, then n' is removed.*

Note that we do not add actions in $\mathcal{U}_n$ to $\mathcal{J}_n$. The reason for this is that the definition of justified action set only takes into account non-updatable atoms on which the action set has no effect.

Reconsider the previous example.

Example 5 *The justified repair tree for $\langle\mathcal{I},\eta\rangle$ as in Example 4 is the following.*

$$\begin{tikzcd}
\emptyset,\emptyset \arrow[d, "r_1'"] \\
\{-a\},\{+b\} \arrow[d, "r_3"] \\
\{-a,\underline{-b}\},\{\underline{+b}\} \\[-20pt]
\times
\end{tikzcd}$$

Now this tree has no valid leaves, as desired.

On the other hand, in Example 1 all repairs are justified. Indeed, the justified repair tree for $\langle\mathcal{I},\eta\rangle$ in that situation is the following.

$$\begin{tikzcd}
 & \emptyset,\emptyset \arrow[dl, "r_1"'] \arrow[dr, "r_1"] & \\
\{-a\},\emptyset \arrow[d, "r_3"] & & \{-b\},\emptyset \arrow[d, "r_2"] \\
\{-a,-b\},\emptyset & & \{-b,-a\},\emptyset
\end{tikzcd}$$

Note that, in the applications of rules r_2 and r_3, their sets of non-updatable literals ($\{\mathsf{not}\ b\}$ and $\{\mathsf{not}\ a\}$, respectively) correspond to actions already in the update part of the node to which they were applied.

In fact, we can show that every justified repair of $\langle\mathcal{I},\eta\rangle$ is computed by the justified repair tree for $\langle\mathcal{I},\eta\rangle$.

Theorem 2 *Let $\mathcal{U}$ be a justified repair for $\langle\mathcal{I},\eta\rangle$. Then there is a leaf n in the justified repair tree for $\langle\mathcal{I},\eta\rangle$ such that $\mathcal{U}=\mathcal{U}_n$.*

Proof. Suppose $\mathcal{U}$ is a justified repair for $\langle\mathcal{I},\eta\rangle$ and let $\mathcal{J}$ be

$$\{\mathsf{nup}(r)\cap\mathsf{lit}(\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U}))\mid\mathsf{ua}(\mathsf{nup}(r))\subseteq\mathcal{U}\cup\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})\}$$

The idea is as follows: we will use $\mathcal{J}$ to "guide" the construction of the branch of the tree leading to $\mathcal{U}$; the choice of $\mathcal{J}$ is directly connected to the definition of justified action set. Note that, by construction, $\mathcal{U}$ is disjoint from the actions dual to the literals in $\mathcal{J}$.

Clearly the root node satisfies $\mathcal{U}_{\mathsf{root}}=\emptyset\subseteq\mathcal{U}$ and $\mathcal{J}_{\mathsf{root}}=\emptyset\subseteq\mathcal{J}$. We show that whenever $\mathcal{U}_n\subseteq\mathcal{U}$ and $\mathcal{J}_n\subseteq\mathcal{J}$ the tree has a descendant n' of n such that $\mathcal{U}_{n'}\subseteq\mathcal{U}$ and $\mathcal{J}_{n'}\subseteq\mathcal{J}$.

Let n be a node of the justified repair tree for $\langle\mathcal{I},\eta\rangle$ such that $\mathcal{U}_n\subseteq\mathcal{U}$ and $\mathcal{J}_n\subseteq\mathcal{J}$. If $\mathcal{U}_n=\mathcal{U}$, then we are done. Otherwise, n must have descendants, since $\mathcal{U}_n$ cannot be a repair (it is a proper subset of $\mathcal{U}$, which is a repair). Suppose $\mathcal{I}\circ\mathcal{U}_n\not\models r$ and assume that $\mathsf{nup}(r)\setminus\mathsf{lit}(\mathcal{U})\not\subseteq\mathcal{J}$. We show that $\mathcal{U}_n\cup\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})$ is closed for r.

Observe that

$$\mathsf{ua}(\mathsf{nup}(r))=\underbrace{(\mathsf{ua}(\mathsf{nup}(r))\cap\mathcal{U}_n)}_{\subseteq\mathcal{U}_n\subseteq\mathcal{U}}\cup\underbrace{(\mathsf{ua}(\mathsf{nup}(r))\setminus\mathcal{U}_n)}_{\subseteq\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ U_n)}\ .$$

Since $\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U}_n)=\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})\cup(\mathcal{U}\setminus\mathcal{U}_n)^D$, actions in $\mathsf{ua}(\mathsf{nup}(r))$ coming from the second of the above sets must be in either $\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})$ or in $(\mathcal{U}\setminus\mathcal{U}_n)^D$. If the former were the case for every action, then $\mathsf{ua}(\mathsf{nup}(r))\subseteq\mathcal{U}\cup\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})$, whence $\mathsf{nup}(r)\cap\mathsf{lit}(\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U}))=\mathsf{nup}(r)\cap\mathsf{lit}\,(\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ U_n))\subseteq\mathcal{J}$, which contradicts our hypothesis.

Therefore, there is an action in $\mathsf{ua}(\mathsf{nup}(r))\cap(\mathcal{U}\setminus\mathcal{U}_n)^D$, whence $\mathsf{ua}(\mathsf{nup}(r))\not\subseteq\mathcal{U}_n\cup\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})$ and therefore $\mathcal{U}_n\cup\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})$ is closed for r.

But if $\mathcal{U}_n\cup\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})$ is closed for every rule applicable in node n, then $\mathcal{U}_n\cup\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})$ is closed for η: a rule is not applicable in node n if either one of its non-updatable literals is contradicted in $\mathcal{I}\circ\mathcal{U}_n$ or $\mathcal{U}_n$ contains an action in the head of r. This cannot be the case, since $\mathcal{U}$ is a justified repair for $\langle\mathcal{I},\eta\rangle$. Therefore, there is a rule r applicable in node n and such that $\mathsf{nup}(r)\setminus\mathsf{lit}(\mathcal{U})\subseteq\mathcal{J}$; this leads to a node further down in the tree still satisfying the required conditions. $\square$

Interestingly, in the previous proof it does not suffice to show that there is a branch leading to $\mathcal{U}$. The following example shows that there may be invalid paths to the justified repair that are pruned; the usage of $\mathcal{J}$ is therefore essential to guarantee the existence of a path to a leaf that is not pruned.

Example 6 *Consider the database $\mathcal{I}=\{a,b\}$ with the following set η of active integrity constraints.*

$$r_1 : a, b \supset -a \qquad r_3 : b, c \supset -b$$
$$r_2 : a, c \supset -a \qquad r_4 : -c \supset +c$$

In this case, there is one justified repair for $\langle\mathcal{I},\eta\rangle$: $\mathcal{U}=\{-a,-b,+c\}$. The justified repair tree for $\langle\mathcal{I},\eta\rangle$ is the following.

$$\begin{tikzcd}
 & \emptyset,\emptyset \arrow[dl, "r_1"'] \arrow[ddd, "r_4"] & \\
\{-a\},\{+b\} \arrow[d, "r_4"'] & & \\
\{-a,+c\},\{+b\} \arrow[d, "r_3"'] & & \\
\underset{\times}{\{-a,+c,\underline{-b}\},\{\underline{+b}\}} & \{+c\},\emptyset \arrow[dl, "r_1"'] \arrow[d, "r_2"] \arrow[dr, "r_3"] & \\
\{+c,-a\},\{+b\} \arrow[d, "r_3"'] & \{+c,-a\},\emptyset \arrow[d, "r_3"'] & \{+c,-b\},\emptyset \arrow[d, "r_2"'] \\
\underset{\times}{\{+c,-a,\underline{-b}\},\{\underline{+b}\}} & \{+c,-a,-b\},\emptyset & \{+c,-b,-a\},\emptyset
\end{tikzcd}$$

There are four leaves in this tree, but only two are not pruned, pointing out that the support for the actions is provided by r_2, r_3 and r_4, but not by r_1.

As was shown in [5], the problem of deciding whether there exists a justified repair for $\langle\mathcal{I},\eta\rangle$ is Σ^2_P-complete. This means that we expect the justified repair tree for $\langle\mathcal{I},\eta\rangle$ to contain leaves that do not correspond to justified repairs (or even to justified weak repairs): since the tree can be constructed in polynomial time, a non-deterministic Turing machine would be able to decide existence of justified repairs in polynomial time, which would imply that $NP=\Sigma^2_P$.

Example 7 *Let $\mathcal{I}$ be the empty database and take η to be the following set of AICs.*

$$r_1 : \mathsf{not}\ a, b \supset +a$$
$$r_2 : a, \mathsf{not}\ b \supset +b$$
$$r_3 : \mathsf{not}\ a, \mathsf{not}\ b, \mathsf{not}\ c \supset +a \mid +b \mid +c$$
$$r_4 : \mathsf{not}\ c \supset +c$$

The justified repair tree for $\langle\mathcal{I},\eta\rangle$, omitting inconsistent nodes from the construction, is the following.

$$\begin{tikzcd}
 & & \emptyset,\emptyset \arrow[dll, "r_3"'] \arrow[dl, "r_3"] \arrow[d, "r_3"'] \arrow[dr, "r_4"] & \\
\{+c\},\emptyset & \{+a\},\emptyset \arrow[dl, "r_2"'] \arrow[d, "r_4"] & \{+b\},\emptyset \arrow[d, "r_1"'] \arrow[dr, "r_4"] & \{+c\},\emptyset \\
\{+a,+b\},\emptyset \arrow[d, "r_4"'] & \{+a,+c\},\emptyset \arrow[d, "r_2"'] & \{+b,+a\},\emptyset \arrow[d, "r_4"'] & \{+b,+c\},\emptyset \arrow[d, "r_1"'] \\
\{+a,+b,+c\},\emptyset & \{+a,+c,+b\},\emptyset & \{+b,+a,+c\},\emptyset & \{+b,+c,+a\},\emptyset
\end{tikzcd}$$

All of the leaves correspond to valid nodes. The set $\mathcal{U}=\{+c\}$ is a justified repair for $\langle\mathcal{I},\eta\rangle$ (actually, the only one), but $\mathcal{U}'=\{+a,+b,+c\}$ is a weak repair for $\langle\mathcal{I},\eta\rangle$ that is not justified: the set $\{+c\}$ is closed under η and contains the no-effect actions of $\mathcal{U}'$.

In the case of normalized AICs, where heads of rules may contain at most one action, the problem of existence of justified repairs is NP-complete. In this case, the leaves of the justified repair tree for $\langle\mathcal{I},\eta\rangle$ are always justified weak repairs.

Lemma 3 *Let $\mathcal{I}$ be a database and η be a set of normalized AICs. Then every leaf of the justified repair tree for $\langle\mathcal{I},\eta\rangle$ contains a founded weak repair.*

Proof. Since the first set of every leaf of the justified repair tree for $\langle\mathcal{I},\eta\rangle$ corresponds to a leaf of the repair tree for $\langle\mathcal{I},\eta\rangle$, by Lemma 2 it is a weak repair for $\langle\mathcal{I},\eta\rangle$. We only need to show that it is founded.

Let α be an action in the update set of that leaf. The rule r labeling the edge from n to n' in the step where α was introduced provides support for α. Indeed, $\alpha\in\mathsf{head}\,(r)$ by construction; furthermore, since every literal $a\neq\mathsf{lit}(\alpha)$ in $\mathsf{body}\,(r)$ is not updatable (α is the only action in the head of r), either $\mathsf{ua}(a)\in\mathcal{U}_n$ or $\mathsf{ua}(a)\in\mathcal{J}_{n'}$. In the first case, $\mathsf{ua}(a)\in\mathcal{U}$, since $\mathcal{U}_n\subseteq\mathcal{U}$. In the second case, $\mathsf{ua}(a)\in\mathcal{J}_{\mathsf{leaf}}$, where leaf is the leaf in question; since this leaf was not pruned, this means that $a\in\mathcal{I}$ and that $\mathsf{ua}(a)^D\notin\mathcal{U}$. In either case, $\mathcal{I}\circ\mathcal{U}\models a$. Therefore r supports α in $\mathcal{U}$. $\square$

Theorem 3 *In the conditions of Lemma 3, every leaf of the justified repair tree for $\langle\mathcal{I},\eta\rangle$ contains a justified repair for $\langle\mathcal{I},\eta\rangle$.*

Proof. By Lemma 3, we already know that every leaf of the justified repair tree for $\langle\mathcal{I},\eta\rangle$ is founded for $\langle\mathcal{I},\eta\rangle$. Let $\mathcal{U}$ be the weak founded repair in a leaf and assume that $\mathcal{U}$ is not a justified repair for $\langle\mathcal{I},\eta\rangle$; since founded weak repairs are always closed, this means that there is $\mathcal{U}' \subsetneq \mathcal{U}$ such that $\mathcal{U}' \cup \mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})$ is closed under η.

Let $\alpha \in \mathcal{U}\setminus\mathcal{U}'$. For every rule r containing α in its head and such that $\mathcal{I}\circ(\mathcal{U}\setminus\{\alpha\})$ satisfies every literal in the body of r except for $\mathsf{lit}(\alpha)^D$, necessarily $\mathsf{ua}(\mathsf{nup}(r)) \not\subseteq \mathcal{U}'\cup\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})$: if $\mathsf{ua}(\mathsf{nup}(r)) \subseteq \mathcal{U}'\cup\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})$, then there is an action $\beta \in \mathsf{head}\,(r)$ such that $\beta \in \mathcal{U}'\cup\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})$, which is absurd: β cannot be α (since $\alpha \notin \mathcal{U}'$ and $\alpha \notin \mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})$), but then $\mathcal{I}\circ(\mathcal{U}\setminus\{\alpha\}) \models \mathsf{lit}(\beta)^D$. Note also that there is at least one such rule, since $\mathcal{U}$ is founded. Therefore there exists $\beta \in \mathsf{ua}(\mathsf{nup}(r))$ such that $\beta \notin \mathcal{U}'\cup\mathsf{ne}\,(\mathcal{I},\mathcal{I}\circ\mathcal{U})$, hence $\beta \in \mathcal{U}\setminus\mathcal{U}'$. In other words: for every $\alpha \in \mathcal{U}\setminus\mathcal{U}'$, if a rule supports α in $\mathcal{U}$ then α is not applicable in $\mathcal{U}'$.

Now consider the branch from the root to this particular leaf. $\mathcal{U}_{\mathsf{root}} = \emptyset \subseteq \mathcal{U}'$. At each stage, either we introduce an action in $\mathcal{U}'$ or an action in $\mathcal{U}\setminus\mathcal{U}'$. In the first case, we maintain the invariant $\mathcal{U}_n \subseteq \mathcal{U}'$; in the second case, we introduce an action α by applying a rule that does not support α in $\mathcal{U}$. But the proof of Lemma 3 shows that the rule introducing an action is always supported in leaves that are not pruned, which is a contradiction. Therefore, $\mathcal{U}\setminus\mathcal{U}' = \emptyset$, whence $\mathcal{U}$ is a justified weak repair. But if η is normalized then justified weak repairs are necessarily repairs (Theorem 4 in [5]), so $\mathcal{U}$ is a justified repair. □

In short: the justified repair tree for $\langle\mathcal{I},\eta\rangle$ contains all justified repairs and, if η is normalized, it only contains justified repairs. We now present two examples showing that, if η is not normalized, (1) there may be weak justified repairs that are not in the justified repair tree for $\langle\mathcal{I},\eta\rangle$; and (2) there may be weak justified repairs in the justified repair tree for $\langle\mathcal{I},\eta\rangle$, so the tree does not contain only repairs.

Example 8 *Let $\mathcal{I}$ and η be, respectively, the empty database and the following set of active integrity constraints.*

$$r_1 : \mathsf{not}\ a, b \supset +a \mid -b \qquad r_2 : a, \mathsf{not}\ b \supset -a \mid +b$$

The justified repair tree for $\langle\mathcal{I},\eta\rangle$ is empty, since the empty set (at the root) is trivially a (justified) repair. However, the set $\{+a,+b\}$ is also a weak justified repair for $\langle\mathcal{I},\eta\rangle$, which is not in the tree.

Example 9 *Consider the following variation of Example 7: $\mathcal{I}$ is again the empty database, but the rules in η are the following.*

$$r_1 : \mathsf{not}\ a, b, c \supset +a \mid -b$$
$$r_2 : a, \mathsf{not}\ b, c \supset -a \mid +b$$
$$r_3 : \mathsf{not}\ a, \mathsf{not}\ b, \mathsf{not}\ c \supset +c \mid +a \mid +b$$
$$r_4 : \mathsf{not}\ c \supset +c$$

Now the only justified repair is still $\mathcal{U} = \{+c\}$, but the weak repair $\mathcal{U}' = \{+a,+b,+c\}$ is also justified. The justified repair tree for $\langle\mathcal{I},\eta\rangle$, omitting inconsistent nodes from the construction, is the following.

$$\begin{tikzcd}
 & & \emptyset,\emptyset \arrow[dll, "r_3"'] \arrow[dl, "r_3"] \arrow[d, "r_3"'] \arrow[dr, "r_4"] & \\
\{+c\},\emptyset & \{+a\},\emptyset \arrow[d, "r_4"'] & \{+b\},\emptyset \arrow[d, "r_4"'] & \{+c\},\emptyset \\
 & \{+a,+c\},\emptyset \arrow[d, "r_2"'] & \{+b,+c\},\emptyset \arrow[d, "r_1"'] & \\
 & \{+a,+c,+b\},\emptyset & \{+b,+c,+a\},\emptyset &
\end{tikzcd}$$

At this stage, we have presented direct algorithms to compute repairs and justified repairs for inconsistent databases with (active) integrity constraints. Since the general problem of finding these repairs is at least NP-complete, in the worst case our algorithms are asymptotically equivalent to the techniques presented in [4] (namely, translating the context to production rules and computing a stable model of these).

However, the general case is not the worst case. We feel that the presentation and discussion of these algorithms serves several purposes. First, these algorithms operate directly on the database and the integrity constraints, requiring significantly less overhead in their execution. Second, they provide insight into the definitions of founded and justified repair, which are not very intuitive in the first place. Third, as we will show in the next section, they allow a refinement of the notion of founded support which can be readily adapted to a more general notion of active integrity constraint.

V. AICs OUTSIDE THE DATABASE WORLD

Due to their syntactic nature, active integrity constraints are restricted to the setting of database-like knowledge bases. We now show how, using the operational characterizations of repairs presented in the previous sections, we can generalize AICs and repairs to more general contexts.

Before proceeding, we take a new look at Examples 3 and 4, which are both examples of what the authors of [5] call "circularity of support", and thus are both rejected by the notion of justified repair. We feel differently about these two situations. While Example 3 contains a very counterintuitive repair $\{-a,-b\}$, which is hard to defend (none of the constraints violated at any stage suggests that either a or b should be removed), Example 4 has a rule clearly suggesting that $-a$ should be applied, although that rule is not the one providing support for $-a$ in the final repair.

The notion of *well-founded repair tree* for $\langle\mathcal{I},\eta\rangle$ captures this distinction: this is a tree built like the justified repair tree for $\langle\mathcal{I},\eta\rangle$, but omitting the sets $\mathcal{J}_n$ (and therefore including more branches). Note that the well-founded repair tree for $\langle\mathcal{I},\eta\rangle$ contains all justified repairs for $\langle\mathcal{I},\eta\rangle$, as well as some founded (weak) repairs that are not justified, but will not contain circularities of support where none of the actions is motivated by violated AICs – see Example 4. Furthermore, this is the tree that is suitable for generalizations outside the database world.

Let $\mathcal{KB}$ be a knowledge base over some logic, e.g. a set of first-order formulas or a description logic knowledge base. A *generalized active integrity constraint* (gAIC) over $\mathcal{KB}$ is a rule r of the form $\varphi \supset \alpha$ satisfying the following conditions.

- The body of the rule, φ, is a decidable condition over the logic underlying $\mathcal{KB}$, i.e. there is an algorithm to decide whether $\Gamma \models \varphi$ for any knowledge base Γ.
- The action α is such that $\Gamma \circ \alpha \models \neg\varphi$ for every Γ, where $\Gamma \circ \alpha$ is the result of updating Γ with the action α.[4]

As a simple, yet relevant, example, we can take $\mathcal{KB}$ to be a description logic knowledge base and allow α to be $\pm C(t)$ or $\pm R(t_1, t_2)$, representing the addition or removal of an instance of a concept or role to the knowledge base. However, the relationship between the left- and right-handside of the gAICs for $\mathcal{KB}$ can take into account the structure of the knowledge base itself. For example, if $\mathcal{KB}$ contains the axiom $C \sqsubseteq D$ for concepts C and D, then $\neg D(t) \supset +C(t)$ would be a valid gAIC.

Given $\mathcal{KB}$ and a set η of gAICs over $\mathcal{KB}$, we can define the well-founded repair tree for $\langle \mathcal{KB}, \eta \rangle$ in a similar way as for AICs over databases: each node is a set of actions; for each node, each rule whose body is not satisfied after updating $\mathcal{KB}$ with the node's actions generates a descendant obtained by adding the action in the head of the rule to the current node. Consistent leaves (where logic consistency is required) are guaranteed to repair the knowledge base, in the sense that all gAICs are satisfied, and we define minimal repairs occurring in the tree to be the *well-founded repairs* of $\langle \mathcal{KB}, \eta \rangle$. This straightforwardly generalizes the earlier definitions: if $\mathcal{KB}$ is a database, then AICs over $\mathcal{KB}$ are gAICs over that database, and the well-founded repair tree for $\langle \mathcal{KB}, \eta \rangle$ coincides with the one defined earlier.

Also note that both the repair tree and the justified repair tree are not readily generalizable. The size of the repair tree now depends on the size of $\mathcal{KB}$ and not of η, since there is no syntactic relationship between conditions and actions allowed in rules, which makes it unusable in practice. As for justified repair trees, their definition is directly related to the specific syntactic structure of AICs: the notion of non-updatable atom in the body of a rule has no counterpart in this more general setting. However, the well-founded repair tree is the "right" generalization since it captures the most adequate notion of repair, as argued above.

VI. Conclusions and Future Work

We introduced an operational semantics for active integrity constraints, showing how repairs and justified repairs for a database $\mathcal{I}$ with a set of AICs η can, under the right conditions, be computed by a search in an adequately constructed tree.

[4]Note that it does not suffice to guarantee this condition for $\mathcal{KB}$, since α must be able to repair φ even after $\mathcal{KB}$ has been updated.

The complexity of the algorithms we present is the best that can be hoped, since existence of the different kinds of repairs is an NP- or Σ_P^2-complete problem. In the first case – finding repairs or justified repairs for a normalized η –, the trees exactly compute the desired kinds of (weak) repairs (possibly requiring an inclusion test, which does not affect the overall complexity); in the second case – searching for justified repairs with possibly non-normalized AICs –, the trees can compute all repairs, but one must still verify whether they are justified (which is again an NP-complete problem). Our algorithms also provide more intuition on the different semantics of repairs, since they follow the original idea behind AICs: that the actions in their heads should "guide" the search for repairs. Also note that these algorithms are suitable for parallel computation, since each branch is independent of the remaining ones; and all the well-known search techniques for trees can be applied especially if one only wishes to find *one* viable repair.

Furthermore, while attempting to build similar trees to compute founded repairs, we were driven to the notion of well-founded repair tree, which provides a more fine-grained characterization of founded, but non-justified repairs, distinguishing essential circularity of support from support that is indeed circular when one considers the final repair, but can be motivated by the heads of violated active integrity constraints.

The notion of well-founded repair tree also allowed us to define analogues of active integrity constraints in a more general setting of knowledge bases, namely description logics where reasoning techniques are available, and give operational semantics for these generalized AICs. We intend to pursue this idea by implementing these algorithms on top of existing ontologies.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[2] G. Antoniou, C.V. Damasio, B. Grosof, I. Horrocks, M. Kifer, J. Maluszynski, and P.F. Patel-Schneider. Combining rules and ontologies: A survey. Technical Report IST506779/Linköping/I3-D3/D/PU/a1, Linköping University, 2004. available at http://rewerse.net/publications/.

[3] C. Beeri and M.Y. Vardi. The implication problem for data dependencies. In *Proceedings of the 8th Colloquium on Automata, Languages and Programming*, pages 73–85, London, UK, 1981. Springer-Verlag.

[4] L. Caroprese, S. Greco, C. Sirangelo, and E. Zumpano. Declarative semantics of production rules for integrity maintenance. In S. Etalle and M. Truszczynski, editors, *ICLP*, volume 4079 of *Lecture Notes in Computer Science*, pages 26–40. Springer, 2006.

[5] L. Caroprese and M. Truszczyński. Active integrity constraints and revision programming. *Theory Pract. Log. Program.*, 11(6):905–952, November 2011.

[6] J. Chomicki. Consistent query answering: Five easy pieces. In T. Schwentick and D. Suciu, editors, *ICDT*, volume 4353 of *LNCS*, pages 1–17. Springer, 2007.

[7] S. Flesca, S. Greco, and E. Zumpano. Active integrity constraints. In E. Moggi and D. Scott Warren, editors, *PPDP*, pages 98–107. ACM, 2004.

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

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

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