Measure-Based Repair Checking by Integrity Checking

Hendrik Decker[1(✉)] and Sanjay Misra[2]

[1] PMS, Inst. f. Informatik, LMU, München, Germany
hdecker@pms.ifi.lmu.de
[2] Covenant University, Canaanland, Ota, Nigeria

Abstract. Quality damage in databases can be measured by seizing extant inconsistency, i.e., by quantifying the amount of violation of integrity constraints. A repair is an update that reduces inconsistency and hence improves data quality. Repair checking finds out if a given update is a repair or not. Repair checking can be done by checking if the undo of the update increases the amount of integrity or not. To do so, sound measure-based integrity checking methods can be used. To do so well, the used methods should also be complete. Repair checking by integrity checking is an attractive alternative to conventional repair checking approaches. However, the completeness of measure-based integrity checking may be a problem, in general. We build on concepts, techniques and results as presented in the first author's previous work.

1 Introduction

Repair checking (abbr. *RCh*) is the problem to find out if a given update repairs violations of database integrity [1,5,6,20]. Conventional RCh evaluates the integrity of each constraint brute-force. Alternatively, we propose to compute RCh by measure-based integrity checking (abbr. *ICh*) [8,11,12].

The main conceptual difference between brute-force and measure-based RCh: the latter is inconsistency-tolerant, i.e., it accepts reductions of integrity violations that may not totally eliminate inconsistency. As opposed to that, brute-force RCh disqualifies each update that does not yield total consistency. The main technical difference: measure-based RCh can be implemented by simplified ICh [7], while brute-force RCh evaluates all constraints brute-force. The main practical difference: If implemented by simplified ICh which exploits the incrementality of updates, measure-based RCh is more efficient.

The two main pillars of our approach are inconsistency-tolerant ICh [9,15] and inconsistency measures [4,16,17,19]. Both have been combined to measure-based ICh in [8,14], which we now propose to use for RCh. Obviously, RCh is related to repairing, i.e., the problem to eliminate inconsistency from a database D such that consistent parts of D are preserved [10,22]. Repairing may be in need of RCh, as broached in [6,20]. Yet, repairs can be processed as goals to be solved, e.g., as view update requests to be satisfied, as done in [12,15].

O. Gervasi et al. (Eds.): ICCSA 2016, Part V, LNCS 9790, pp. 530–543, 2016.
DOI: 10.1007/978-3-319-42092-9_40

By combining integrity-preserving update methods with measure-based ICh, inconsistency-tolerant repairing can be achieved, as described in [10].

In fact, those methods usually do not check if repairs are minimal, as required by consistent query answering (CQA) [2,5], because most of the repairs computed by the mentioned methods are minimal already. On the other hand, if an arbitrarily given update needs to be checked to be or not to be a repair, then the satisfaction of some minimality criterion is indispensable.

Our main contributions: a measure-based inconsistency-tolerant generalization of repairs, of their minimality, and of RCh, as well as an implementation of RCh by measure-based ICh. Proofs of all results are available in the full paper.

2 Key Concepts

As in [7–14], we use common database research terminology and formalisms. See [10] also for definitions of cases and causes of integrity violations, inconsistency measures, measures $\iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|$ based on cases or causes, and measure-based inconsistency-tolerant integrity checking methods. Let D, IC, U, μ always stand for a database, an integrity theory, an update, and, resp., an inconsistency measure. We may use ";" for delimiting clauses in sets, instead of ",".

In Subsect. 2.1, we define measure-based repairs. In Subsect. 2.2, we characterize repair checking of updates U as a function, which is decomposed into checking if U reduces inconsistency, and if it does, then checking if U is also minimal, in some sense.

2.1 Repairs

Measure-based repairs generalize total repairs: they do not insist that all inconsistency is eliminated. Thus, a measure-based repair is an update U of a database that is inconsistent with its constraints, such that the updated database becomes less inconsistent, and there is no subset of U that could achieve the same or a larger amount of inconsistency reduction. Below, we define total and measure-based repairs as inconsistency reductions that are minimal.

Definition 1 *(total and measure-based repair).*

(a) U is a *total inconsistency reduction* of (D, IC) if $\mu(D^U, IC) = 0$.
(b) U is a *total repair* of (D, IC) if U is a total inconsistency reduction of (D, IC), and there is no proper subset U' of U with that property.
(c) U is a *μ-based inconsistency reduction* of (D, IC) if $\mu(D^U, IC) < \mu(D, IC)$.
(d) U is a *μ-based repair* of (D, IC) if U is a μ-based inconsistency reduction, and there is no proper subset U' of U such that $\mu(D^{U'}, IC) \leq \mu(D^U, IC)$.

Clearly, total repairs do not depend on any inconsistency measure, as opposed to inconsistency reductions and measure-based repairs. In the literature, total repairs are defined without recurring explicitly on any measure, by requiring that each constraint in IC is satisfied in D^U. However, minimality needs to be

measured, and indeed, there are several alternative, non-equivalent definitions of the minimality of total repairs. Yet, subset minimality as in Definition 1a is the one most often referred to.

Obviously, each total and each μ-based repair is a total and, resp., μ-based inconsistency reduction, but not vice-versa.

The following example features an update U that satisfies $\mu(D^U, IC) < \mu(D, IC)$ as in Definition 1c, but not the minimality condition of 1*d*. Typically, updates of that kind contain elements that do not contribute to the reduction of inconsistency. Example 1 also features an update of D that is a measure-based repair of (D, IC).

Example 1. Let $D = \{p, q, r\}$, $IC = \{\leftarrow q\}$, $U = \{delete\ q,\ insert\ s\}$. It is easy to verify that, for each measure $\mu \in \{\iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|\}$, $\mu(D^U, IC) < \mu(D, IC)$ holds, i.e., U is a μ-based inconsistency reduction. However, U is not a μ-based repair of (D, IC), since its subset $U' = \{delete\ q\}$ clearly is a μ-based repair of (D, IC) that yields the same amount of inconsistency reduction in a minimal way, since $\mu(D^{U'}, IC) = \mu(D^U, IC)$ holds.

In fact, the minimality condition of measure-based repairs in Definition 1d should not be weakened so as to simply require that there is no proper subset U' of U such that $\mu(D^{U'}, IC) < \mu(D, IC)$, as illustrated by Example 2.

Example 2. Let $\mu \in \{\iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|\}$. Further, let $D = \{p, q, r, s\}$, $IC = \{\leftarrow q, \leftarrow r, \leftarrow s\}$, $U = \{delete\ r,\ delete\ s\}$, and $U' = \{delete\ r\}$. Clearly, U and U' are μ-based repairs such that $D^U = \{p, q\}$ and $D^{U'} = \{p, q, s\}$. Moreover, U' is a proper subset of U and $\mu(D^{U'}, IC) < \mu(D, IC)$. However, also $\mu(D^U, IC) < \mu(D^{U'}, IC)$ holds, i.e., U reduces inconsistency more than U', i.e., U' is not preferable to U.

The following corollaries of Definition 1 interrelate measure-based repairs, total repairs and inconsistency reductions.

Corollary 1. For each measure μ and each inconsistent pair (D, IC), each total repair of (D, IC) is a μ-based repair.

Corollary 2. If U is a total inconsistency reduction of (D, IC), then some subset of U is a total repair of (D, IC).

Corollary 3. Each singleton update U such that $\mu(D^U, IC) < \mu(D, IC)$ is a μ-based repair of (D, IC).

Corollary 3 devises an iteration of updates for iterated repairs by singleton updates. For example, let $\{I_1, I_2, \ldots, I_n\}$ be the set of violated basic cases in (D, IC). Then, for $\mu \in \{\zeta, |\zeta|\}$, a total elimination of inconsistency in (D, IC) can be achieved by iteratively applying singleton μ-based repairs $U_1, U_2, \ldots, U_n$ such that, for $U_0 = \emptyset$, U_i eliminates the violation of I_i in D^{Ui-1}, $1 \leq i \leq n$, until all violations in (D, IC) are gone in D^{Un}. If some U_i also eliminates the violation of I_j $(j > i)$, then $U_j = \emptyset$.

Example 3a and b illustrate that, for two measures μ, μ', a μ-based repair U is not necessarily a μ'-based repair, since μ and μ' may measure the effect of U differently.

Example 3. Let $D = \{p, q, r\}$ and $IC = \{\leftarrow q, \leftarrow s\}$.

(a) Let $U_a = \{$*delete* q, *insert* $s\}$. Clearly, U_a is a μ-based repair of (D, IC) for each μ that assigns a higher weight of inconsistency to the violation of $\leftarrow q$ than to the violation of $\leftarrow s$. However, U_a is not a $|\iota|$-based repair of (D, IC) ($|\iota|$ counts the violated constraints in IC), since $|\iota|(D^{Ua}, IC) = |\iota|(D, IC) = 1$.
(b) Let $U_b = \{$*insert* $o\}$, and μ be the measure that counts the facts in D that contribute to some integrity violation and then divides that count by the cardinality of D. Since $\mu(D^{Ub}, IC) = 1/4$ and $\mu(D, IC) = 1/3$, U_b is a μ-based repair, by Corollary 3. However, for each $\mu' \in \{\iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|\}$, U_b is clearly not a μ'-based repair of (D, IC), since $\mu'(D, IC) = \mu'(D^{Ub}, IC) = 1$.

2.2 Repair Checking

We distinguish between *total* and *measure-based* repair checking. The total repair checking problem is to find out if an update U is a total repair of (D, IC). Hence, each total repair checking method can be described as a function rc_t that maps triples (D, IC, U) to $\{yes, no\}$. Similarly, the μ-based repair checking problem is to find out if U is a μ-based repair of (D, IC). Hence, each μ-based repair checking method can be described as a function rc_m that maps triples (D, IC, U) to $\{yes, no\}$.

By Definition 1, repair checking of an update U proceeds in two phases. First, to check if U reduces inconsistency totally ($1a$) or if U is an inconsistency reduction ($1c$). We call this phase the *inconsistency reduction check*. If U has passed the inconsistency reduction check, the second phase of repair checking is to check if U is minimal, in the sense of Definition 1b and d, respectively. We call this phase the *minimality check*. The soundness and completeness of repair checking methods is defined as follows.

Definition 2 *(soundness and completeness of measure-based repair checking methods).* Let μ be an inconsistency measure, and rc a function that maps triples (D, IC, U) to $\{\text{yes}, \text{no}\}$. rc *is* called a sound, resp., complete, μ-based repair checking method if (*), resp., (**) holds, for each triple (D, IC, U).

(*) $rc(D, IC, U) = yes \Rightarrow U$ is a μ-based repair
(**) U is a μ-based repair $\Rightarrow rc(D, IC, U) = yes$

In words, rc is sound if its output $rc(D, IC, U) = yes$ correctly identifies U as a μ-based repair of (D, IC), and complete if each μ-based repair U of (D, IC) is checked correctly by rc. Soundness and completeness of total repair checking is defined analogously: ignore μ and replace each occurrence of "μ-based" by "total" in Definition 3.

Corollary 4, below, is going to be useful for valuating results in Sect. 3.

Corollary 4.

(a) If U is not a μ-based repair of (D, IC), then each sound μ-based repair checking method rc outputs $ir(D, IC, U) = \text{no}$.
(b) If a complete repair checking method rc outputs $rc(D, IC, U) = \text{no}$, then U is not a μ-based repair of (D, IC).

According to the first of the two phases of repair checking as identified above, the following definition characterizes sound and complete measure-based inconsistency reduction checking. (Analogously, total integrity reduction checking could be defined.)

Definition 3 *(measure-based inconsistency reduction checking).* Let μ be an inconsistency measure, and ir a function that maps triples (D, IC, U) to $\{yes, no\}$. ir is called a *sound,* resp., *complete,* μ-based inconsistency reduction checking method if (*), resp., (**) holds, for each triple (D, IC, U).

$(*)$ $ir(D, IC, U) = yes \;\Rightarrow\; \mu(D^U, IC) < \mu(D, IC)$

$(**)$ $\mu(D^U, IC) < \mu(D, IC) \;\Rightarrow\; ir(D, IC, U) = yes$

In words, ir is sound if $ir(D, IC, U) = yes$ correctly indicates that U reduces the inconsistency of (D, IC) measured by μ, and complete if each U that reduces inconsistency is checked correctly by rc.

Next, we define the soundness and completeness of the second phase of measure-based repair checking, viz. measure-based minimality checking, according to Definition 1d.

Definition 4 *(measure-based minimality checking).*

Let μ be an inconsistency measure, and mc a function that maps triples (D, IC, U) to $\{yes, no\}$. mc is called a *sound,* resp., *complete,* μ-based minimality checking method if (*), resp., (**) holds, for each triple (D, IC, U) such that U is an μ-based inconsistency reduction.

$(*)$ $mc(D, IC, U) = yes \;\Rightarrow\; for\ each\ U' \subsetneq U,\ \mu(D^{U'}, IC) \not\leq \mu(D^U, IC)$

$(**)$ $for\ each\ U' \subsetneq U,\ \mu(D^{U'}, IC) \not\leq \mu(D^U, IC) \;\Rightarrow\; mc(D, IC, U) = yes$

In words, mc is sound if $mc(D, IC, U) = yes$ correctly indicates that U is a minimal inconsistency reduction, according to Definition 1d, and mc is complete if the minimality of each μ-based repair of (D, IC) is checked correctly by rc.

The following result is a straightforward consequence of Definitions 3, 4 and 5. It effectively says that repair checking can be realized by inconsistency reduction checking, followed by minimality checking, if needed. (Minimality checking is of course not needed if the output of the inconsistency reduction check is negative.)

Theorem 1. Let μ be an inconsistency measure, ir a sound (resp., complete) μ-based inconsistency reduction method, mc a sound (resp., complete) μ-based minimality checking method, and rc a function that maps triples (D, IC, U) to $\{yes, no\}$. rc is a sound (resp., complete) μ-based repair checking method if it is defined by the following equivalence.

$rc(D, IC, U) = yes \;\Leftrightarrow\; ir(D, IC, U) = yes \text{ and } mc(D, IC, U) = yes$

In Sect. 3, we are going to see how each of the two phases of measure-based repair checking can be implemented by measure-based integrity checking.

3 The Main Results

In Subsect. 3.1, we show that inconsistency reduction can be computed by integrity checking. In Subsect. 3.2, we show that also minimality can be verified by integrity checking. This leads to the main result in Subsect. 3.3, that repair checking can be computed by integrity checking.

3.1 Inconsistency Reduction Checking by Integrity Checking

We are going to show that inconsistency reduction checking, i.e. $ir(D, IC, U)$, can be computed by $ic(D^U, IC, \overline{U})$, where $\overline{U}$ is the update that undoes U, as formalized below. We denote consecutive updates U, U' of D and then D^U by $D^{UU'}$. Hence, $D^{U\overline{U}} = D$.

Definition 5. Let $\overline{U}$ denote the *undo* of U: for each element of the form *insert* X or *delete* Y in U, $\overline{U}$ contains *delete* X or, resp., *insert* Y, and nothing else.

The following lemmata serve to identify a way to check inconsistency reduction by a complete integrity checking method. Lemma 1 says that complete measure-based inconsistency reduction checking of an update can be realized by measure-based integrity checking of its undo. Lemma 2 says that soundness of inconsistency reduction checking can be obtained by an additional integrity check of the update itself. Lemma 3 says that sound and complete inconsistency reduction checking is realizable by checking the undo for integrity preservation alone, in case the used integrity checking method is complete and the range of the measure on which it is based is totally ordered.

Checking inconsistency reduction by integrity checking is more efficient than by evaluating each constraint in IC brute-force against D^U, inasmuch as integrity checking simplifies the constraints to be checked by number and complexity [7].

Lemma 1. Let ic be a sound μ-based integrity checking method, and ir a function that maps triples (D, IC, U) to $\{yes, no\}$. ir is a complete μ-based inconsistency reduction checking method if ir is defined by the following equivalence.

$$ir(D, IC, U) = yes \;\Leftrightarrow\; ic(D^U, IC, \overline{U}) = no$$

An immediate consequence of Lemma 1 is that each update that does not reduce inconsistency is detected by checking the undo for integrity preservation with a complete integrity checking method.

The following lemma states that U reduces inconsistency if and only if the output of checking the integrity of $\overline{U}$ is negative and the output of checking the integrity of U is positive; if the outcome for $\overline{U}$ was positive, U is not a repair, by Lemma 1.

Lemma 2. Let ic be a sound and complete μ-based integrity checking method, and ir a function that maps triples (D, IC, U) to $\{yes, no\}$. ir is a sound and complete μ-based inconsistency reduction checking method if ir is defined by the following equivalence.

$$ir(D, IC, U) = yes \;\Leftrightarrow\; ic(Di^U, IC, \overline{U}) = no \text{ and } ic(D, IC, U) = yes$$

Lemma 3, below, sharpens the equivalence of Lemma 2: it states that, if the range of the measure μ is totally ordered, then $ir(D, IC, U)$ can be computed by applying any sound and complete μ-based integrity checking to $(D^U, IC, \overline{U})$.

Lemma 3. Let μ be an inconsistency measure with a totally ordered range, ic a sound and complete μ-based integrity checking method, and ir a function that maps triples (D, IC, U) to $\{yes, no\}$. Then, ir is a sound and complete μ-based inconsistency reduction checking method if ir is defined by the following equivalence.

$$ir(D, IC, U) = yes \;\Leftrightarrow\; ic(D^U, IC, \overline{U}) = no$$

The following example shows that incomplete integrity checking is deficient for inconsistency reduction checking, i.e., Lemmata 2 and 3 cannot be weakened by dropping the completeness requirement of integrity checking.

Example 4. Let $D = \{p \leftarrow q;\ p \leftarrow r;\ q\}$, $IC = \{\leftarrow p\}$, $U = \{insert\ r\}$, ic be the well-known integrity checking method in [18], and inconsistency be measured by ι. It is shown in [15] that ic is a sound but incomplete ι-based integrity checking method. It is easy to see that $ic(D^U, IC, \overline{U}) = no$, although we have that $\iota(D^{U\overline{U}}, IC) = \iota(D, IC) = \iota(D^U, IC)$, i.e., U is not a repair of (D, IC).

The next example illustrates that Lemma 3 cannot be weakened by waiving the total order requirement of the range of μ.

Example 5. The range of κ (the measure that maps (D, IC) to the set of causes of violations of IC in D) is partially but not totally ordered by the subset-or-equal relationship. Now, let ic be Decker's extension of Nicolas' well-known method to deductive databases. ic is a sound and complete κ-based method for definite propositional databases and constraints. For $D = \{p \leftarrow q;\ r \leftarrow s;\ q\}$, $IC = \{\leftarrow p,\ \leftarrow r\}$ and $U = \{delete\, q,\ insert\, s\}$, it is easy to see that $\overline{U} = \{insert\ q,\ delete\ s\}$, $\kappa(D, IC) = \{q\}$, $D^U = \{p \leftarrow q; r \leftarrow s; s\}$, $\kappa(D^U, IC) = \{s\}$, and $ic(D, IC, U) = ic(D^U, IC, \overline{U}) = no$.

Hence, by Lemma 2, $ic(D, IC, U) = no$ correctly indicates that U is not a κ-based repair of (D, IC). Yet, Lemma 3 cannot be used to obtain that result, since the range of κ is not totally ordered. Indeed, waiving that requirement in Lemma 3 would lead to the wrong conclusion that U was an κ-based repair. If, instead of κ, $|\kappa|$ would be used, for which ic is sound but incomplete, we'd have $|\kappa|(D, IC) = |\kappa|(D^U, IC) = 1$. By Lemma 1, we could define two complete inconsistency reduction methods, based on κ or, resp., $|\kappa|$, via the output of ic, both of which however would not be sound, since they both would wrongly indicate that U was a repair of (D, IC).

3.2 Minimality Checking by Integrity Checking

By Lemmata 4 and 5, we show that also the minimality check of measure-based repairs can be accomplished by integrity checking.

Lemma 4. For each inconsistency measure μ, each sound μ-based integrity checking method ic, each database D, each integrity theory IC, and each μ-based inconsistency reduction U of (D, IC), the following holds.

(a) U is not a μ-based repair of (D, IC) if there is $U' \subsetneq U$ such that $ic(D^U, IC, \overline{U}'') = yes$, where $U'' = U \setminus U'$. If, additionally, ic is complete, then
(b) U is not a μ-based repair of (D, IC) iff there is $U' \subsetneq U$ such that $ic(D^U, IC, \overline{U}'') = yes$, where $U'' = U \setminus U'$.

The contrapositive of Lemma 4 is the following corollary, which provides a way to check the minimality of a measure-based inconsistency reduction by measure-based integrity checking, without the necessity of computing measures.

Corollary 5. For each inconsistency measure μ, each sound μ-based integrity checking method ic, each database D, each integrity theory IC, and each μ-based inconsistency reduction U of (D, IC), the following holds.

(a) U is a μ-based repair of (D, IC) only if, for each $U' \subsetneq U$, $ic(D^U, IC, \overline{U}'') =$ no, where $U'' = U \setminus U'$. If, additionally, ic is complete, then
(b) *U is a μ-based repair of (D, IC) if and only if, for each $U' \subsetneq U$, $ic(D^U, IC, \overline{U}'') =$ no, where $U'' = U \setminus U'$.*

Corollary 5 enables us to define sound and complete minimality checking methods on the basis of complete and, resp., sound integrity checking methods, as follows.

Lemma 5. Let μ be an inconsistency measure, ic a sound μ-based integrity checking method, and mc a function that maps triples (D, IC, U) to $\{yes, no\}$. mc is a complete μ-based minimality checking method if mc is defined by the following equivalence.

$$mc(D, IC, U) = yes \;\Leftrightarrow\; \text{for each } U' \subsetneq U,\ ic(D^U, IC, \overline{U}'') = no, \text{ where } U'' = U \setminus U'.$$

If, additionally, ic is a complete μ-based integrity checking method, then mc also is a sound minimality checking method.

3.3 The Main Theorems

Lemmata 1, 2, 3 and 5 provide handles for unfolding Theorem 1 into the main results of this paper, as stated in Theorems 2, 3 and 4, below. Essentially, each of these results states that measure-based repair checking can be done by integrity checking, without having to compute the measures. Theorem 2 is the weakest

of the three, in that it only devises a way to see that a given update U is not a repair (cf. Corollary 3). By requiring that the used integrity checking method is not just sound but also complete, each of Theorems 2 and 3 devises a sound and complete way to check if U is or is not a repair. The difference between the two results is that Theorem 3 additionally requires that the range of the used inconsistency measure is totally ordered, but in turn requires one run of integrity checking less than Theorem 2.

Theorem 2. Let ic be a sound μ-based integrity checking method, and rc a function that maps triples (D, IC, U) to $\{yes, no\}$. rc is a complete μ-based repair checking method if rc is defined by the following quivalence.

$$rc(D, IC, U) = yes \;\Leftrightarrow\; ic(D^U, IC, \overline{U}) = no, \text{ and for each } U' \subsetneq U, \\ ic(D^U, IC, \overline{U}'') = no, \text{ where } U'' = U \setminus U'.$$

Theorem 3. Let μ be an inconsistency measure, ic a sound and complete μ-based integrity checking method, and rc a function that maps triples (D, IC, U) to $\{yes, no\}$. rc is a sound and complete μ-based repair checking method if rc is defined by the following equivalence.

$$ir(D, IC, U) = yes \;\Leftrightarrow\; ic(D^U, IC, \overline{U}) = no,\ ic(D, IC, U) = yes, \text{ and for each} \\ U' \subsetneq U,\ ic(D^U, IC, \overline{U}'') = no, \text{ where } U'' = U \setminus U'.$$

Theorem 4. Let μ be an inconsistency measure with a totally ordered range, ic a sound and complete μ-based integrity checking method, and rc a function that maps triples (D, IC, U) to $\{yes, no\}$. rc is a sound and complete μ-based repair checking method if rc is defined by the following equivalence.

$$rc(D, IC, U) = yes \;\Leftrightarrow ic(D^U, IC, \overline{U}) = no, \text{ and for each } U' \subsetneq U, \\ ic(D^U, IC, \overline{U}'') = no, \text{ where } U'' = U \setminus U'.$$

4 Computing Repair Checking

In Subsect. 4.1, we outline how to compute total repair checking. In Subsects. 4.1 and 4.3, we do the same for measure-based repair checking. The approach in Subsect. 4.1 is "naive"; it computes inconsistency measures. The one in Subsect. 4.3 uses measure-based integrity checking; we call it "repair checking by (measure-based) integrity checking". For comparing the three, note that the approaches in Subsects. 4.1 and 4.3 are inconsistency-tolerant, and therefore much more realistic than total repair checking. Moreover, we are going to see that measure-based repair checking tends to be far less costly than both naive and total repair checking.

4.1 Computing Total Repair Checking

After describing the computation of total repair checking, we assess its cost. Then, we describe how total repairs can be computed from a given inconsistency reduction.

4.1.1 Two Phases for Computing Total Repair Checking

By Definitions 1a and b, total repair checking may verify or falsify, for triples (D, IC, U), that (D^U, IC) is consistent, and that U is minimal, in two phases.

Phase 1 (*inconsistency reduction check*):
Check if U is a total inconsistency reduction of (D, IC), by querying each $I \in IC$ against D^U. If some I is not satisfied in D^U, then U is not a total inconsistency reduction and hence not a total repair of (D, IC). If each $I \in IC$ is satisfied in D^U, then U is a total inconsistency reduction, hence proceed to Phase 2.

Phase 2 (*minimality check*):
For each $U' \subsetneq U$, check if U' is a total inconsistency reduction of (D, IC), by querying each $I \in IC$ against $D^{U'}$. If each I is satisfied in $D^{U'}$, then U' is a total inconsistency reduction of (D, IC). Hence, U is not a total repair of (D, IC). If some I is not satisfied in $D^{U'}$, then U' is not a total inconsistency reduction of (D, IC). If no proper subset U' of U is an inconsistency reduction of (D, IC), then U is a total repair of (D, IC).

4.1.2 The Cost of Total Repair Checking

We now assess the cost of computing Phases 1 and 2. Let n be the cardinality of U and $m = 2^n$. Thus, there are m subsets $U_1, \ldots, U_m$ of U; one of them empty. Hence, easily up to $m-1$ brute-force integrity checks of (D^{Ui}, IC) ($i = 1, \ldots, m\text{-}1$) are needed for deciding if U (or any of its proper non-empty subsets) is a total repair of (D, IC). If k is the cardinality of IC, then that amounts to $k \times (m-1)$ evaluations of arbitrarily complex constraints in IC.

4.1.3 A Third Phase for Computing Total Repairs

If, in Phase 1, U turns out to be an inconsistency reduction of (D, IC), and in Phase 2, U turns out to not be a total repair of (D, IC), then, by Corollary 2, at least one total repair of (D, IC) can be found in a third phase, as follows.

Phase 3 *(identify total repairs)*:
Let $\mathcal{U} = \{U' \mid U$ is inconsistency reduction of (D, IC) detected in Phase 2, $U' \subsetneq U\}$. The subset-minimal elements in $\mathcal{U}$ are total repairs of (D, IC).

Phases 1–3 can also be presented uniformly, as follows: For each $U' \subseteq U$, check if U' is a total inconsistency reduction of (D, IC), by querying each I in IC against $D^{U'}$. U' is a total inconsistency reduction of (D, IC) if and only if each I is satisfied in $D^{U'}$. Let $\mathcal{U}$ be the set of subsets of U that are a total inconsistency reduction of (D, IC). If $\mathcal{U}$ is empty, neither U nor any of its subsets is a total repair of (D, IC). Otherwise, each subset-minimal set in $\mathcal{U}$ is a total repair of (D, IC).

4.2 Computing Naive Repair Checking

In this subsection, we first describe a naive way of computing total repair checking. Then, we assess the cost of that computation. Last, we describe how measure-based repairs can be computed from a given inconsistency reduction.

4.2.1 Two Phases for Computing Naive Repair Checking

According to Definition 1d, measure-based repair checking can be implemented naively in two phases: inconsistency reduction checking and minimality checking.

Phase 1: To check if U is a μ-based inconsistency reduction of (D, IC), i.e., to check $\mu(D^U, IC) < \mu(D, IC)$, the values of $\mu(D, IC)$ and $\mu(D^U, IC)$ can be computed and then compared. By that comparison, U is or is not an inconsistency reduction.

Phase 2: To check if a μ-based inconsistency reduction U of (D, IC) is minimal, the measure $\mu(D^{U'}, IC)$ of each proper non-empty subset U' of U has to be computed and compared to $\mu(D, IC)$, as already computed in Phase 1. If $\mu(D^{U'}, IC) \leq \mu(D^U, IC)$ does not hold for any such U', then U is a μ-based repair of (D, IC). If, for some such U', $\mu(D^{U'}, IC) \leq \mu(D^U, IC)$ holds, then (U' is an μ-based inconsistency reduction of (D, IC), and) U is not a μ-based repair of (D, IC).

The part of the description of Phase 2 between $(\ldots)$ does not contribute to the decision if U is a total repair of (D, IC) or not, but is of interest in the context of an additional third phase, as addressed in Subsubsect. 4.2.3.

4.2.2 The Cost of Naive Measure-Based Repair Checking

Clearly, $\mu(D^{U'}, IC)$ has to be computed for each subset U' of U: for subsets $\{\}$ and U in Phase 1, and for proper non-empty subsets in Phase 2. For $|U| = n$, that amounts to the computation of 2^n measures, one for each subset U' of U. The cost of the computation of measures obviously depends on the definition of μ. To compute $\mu(D, IC)$ for any of the measures $\iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|$ involves the evaluation of each I in IC against D, for $\zeta, |\zeta|, \kappa, |\kappa|$ also an analysis of the proof tree, for determining the violated cases or the causes of integrity violation. Thus, the order of magnitude of evaluating constraints for naive measure-based repair checking is about the same as for total repair checking. However, depending on the specific measure to be computed, the total cost of computing naive repair checking may easily be higher than that of total repair checking.

4.2.3 A Third Phase for Computing Measure-Based Repairs

If U turns out in Phase 1 to be an inconsistency reduction and in Phase 2 to be not an μ-based repair of (D, IC), then, analogously to Subsubsect. 4.2.3, at least one such repair can be identified, as in the following third phase.

Phase 3 *(identify total repairs)*: Let $\mathcal{U} = \{U' \mid U$ is inconsistency reduction of (D, IC) detected in Phase 2, $U' \subsetneq U\}$. The subset-minimal elements in $\mathcal{U}$ are total repairs of (D, IC).

4.3 Computing Repair Checking by Integrity Checking

A computation of measure-based repair checking is suggested by Theorems 2, 3 and 4. According to Theorem 2, the output $rc(D, IC, U) = no$, obtained by a

measure-based integrity checking method that is only sound but not complete, indicates that U is not a μ-based repair. For U to be a μ-based repair, the output $rc(D, IC, U) = yes$, computed according to Theorem 2, is only necessary, but not sufficient. Necessary and sufficient conditions to identify updates as repairs are given by Theorems 3 and 4, which require that a sound and complete μ-based integrity checking method is used.

We are going to assess measure-based repair checking according to Theorem 3, again by the two phases of inconsistency reduction and minimality checking. Note, however, that the cost of measure-based repair checking by Theorem 4 is lower than as by Theorem 3, since the totally ordered range of μ, as required in Theorem 4, enables a less costly inconsistency reduction checking phase.

Phase 1: To check if U is a μ-based inconsistency reduction of (D, IC), compute $ic(D^U, IC, \overline{U})$ and $ic(D, IC, U)$ according to the equivalence in Lemma 2. Thus, at most 2 runs of ic are needed.

Phase 2: To check if U is minimal according to Definition 1*d*, it suffices, by Lemma 5, to check if $ic(D^U, IC, \overline{U}'') = no$, for each $U' \subsetneq U$ such that $U' \neq \emptyset$, where $U'' = U \setminus U'$. Thus, if n is the size of U, at most $2^n - 2$ runs of ic are needed.

So, for Phases 1 and 2, maximally 2^n runs of ic are needed. Thus, the actual cost of repair checking by integrity checking depends on ic. If a sound and complete method for simplified integrity checking is available, then running such a method tends to be much less costly than brute-force integrity checking, as employed for total repair checking and for naive measure-based repair checking.

Recall from Subsect. 4.1 that the cost of total repair checking was $k \times (m-1)$ unsimplified constraint evaluations, where $m = 2^n$ and k is the cardinality of IC. For ease of comparison, suppose that all constraints in IC are expressed by a single constraint formula I (the conjunction of all constraints). Then, for total repair checking, we'd have in the order of m-1 evaluations of I against D^{Ui} $(1 \leq i \leq m-1)$, where U_i is one of the non-empty subsets of U. Compared to that, $m-2$ runs of a simplified form of I for simplified repair checking obviously tends to be significantly more advantageous.

5 Related Work

Related work on integrity checking [7,8,11,12,14,15,18], inconsistency measuring [4,8,11,12,14,16,17,19] and repairing [6,9–12,20,22] has been duly referenced already. Maybe more relevant is the work on brute-force repair checking [1,3,5,6,20,21], as already addressed in previous sections, particularly in Sect. 4.

Work on conventional repairing and repair checking has always focused on complexity issues in relation with CQA. As opposed to that, our alternative to CQA in [13], is not hung up in the complexity nexus between CQA, repairing and repair checking.

The discussion of complexity issues related to CQA focuses grosso modo on certain classes of constraints, e.g., those expressed as conjunctive queries over flat

relational databases. The related complexity classes are amorphously treated, e.g., conjunctive queries have polynomial-time complexity. However, such an undifferentiated treatment misses the point of where cost issues of integrity checking may hurt most.

For example, in a relational database with very large extensions of p, q, r, the evaluation of the conjunctive constraint $I = \leftarrow p(x, y), q(y, z), r(x, y, z)$ may be considered prohibitively costly, although I falls into a relatively "harmless" complexity class. However, for repair candidate U that consists of the insertion of $r(a, b, c)$, our approach to repair checking does not have to evaluate I brute-force. Rather, for most integrity checking methods that are amenable to our way of repair checking, it suffices to evaluate the simplified instance $\leftarrow p(a, b), q(b, c), r(a, b, c)$, by simple look-ups. This shows that our proposal is significantly different from related work on repair checking.

6 Conclusion

Deploying inconsistency-measure-based ICh, we have elaborated a non-standard approach to RCh, that compares favorably to conventional brute-force RCh.

Measure-based RCh does not need to compute measures, nor does it have to know which constraints are violated, nor why or how they are violated. Nor does brute-force RCh. However, the latter does check each constraint brute-force, and gives up upon the least bit of remaining consistency impairment. As opposed to that, measure-based RCh is inconsistency-tolerant, permits measured comparisons of the quality increase of repair alternatives, and is less costly since its integrity checks are simplified. In general, however, sound repair checking by integrity checking requires a complete measure-based integrity checking method. Although most known measure-based integrity checking methods are incomplete, there are significant classes of databases, integrity theories, updates and inconsistency measures for which the completeness of integrity checking can be guaranteed, or relaxed while preserving the soundness of measure-based repair checking. Ongoing work is concerned with identifying such classes.

Acknowledgement. John Grant had provided valuable comment on early drafts of this work.

References

1. Afrati, F., Kolaitis, P.: Repair checking in inconsistent databases: algorithms and complexity. In: 12th ICDT, pp. 31–41. ACM Press (2009)
2. Arenas, M., Bertossi, L., Chomicki, J.: Consistent query answers in inconsistent databases. In: Proceedings of 18th PoDS, pp. 68–79. ACM (1999)
3. Arming, S., Pichler, B., Sallinger, E.: Combined complexity of repair checking and consistent query answering. In: 19th International Conference on Database Theory (ICDT), vol. 48, pp. 21:1–21:18. LIPIcs (2016)

4. Besnard, P.: Revisiting postulates for inconsistency measures. In: Fermé, E., Leite, J. (eds.) JELIA 2014. LNCS, vol. 8761, pp. 383–396. Springer, Heidelberg (2014)
5. Chomicki, J.: Consistent query answering: five easy pieces. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 1–17. Springer, Heidelberg (2006)
6. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Inf. Comput. **197**(1/2), 90–121 (2005)
7. Christiansen, H., Martinenghi, D.: On simplification of database integrity constraints. Fundam. Inf. **71**(4), 371–417 (2006). IOS Press
8. Decker, H.: Quantifying the quality of stored data by measuring their integrity. In: Proceedings of DIWT 2009, Workshop SMM, pp. 823–828. IEEE (2009)
9. Decker, H.: Causes of the violation of integrity constraints for supporting the quality of databases. In: Murgante, B., Gervasi, O., Iglesias, A., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2011, Part V. LNCS, vol. 6786, pp. 283–292. Springer, Heidelberg (2011)
10. Decker, H.: Partial repairs that tolerate inconsistency. In: Eder, J., Bielikova, M., Tjoa, A.M. (eds.) ADBIS 2011. LNCS, vol. 6909, pp. 389–400. Springer, Heidelberg (2011)
11. Decker, H.: New measures for maintaining the quality of databases. In: Murgante, B., Gervasi, O., Misra, S., Nedjah, N., Rocha, A.M.A.C., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2012, Part IV. LNCS, vol. 7336, pp. 170–185. Springer, Heidelberg (2012)
12. Decker, H.: Measure-based inconsistency-tolerant maintenance of database integrity. In: Schewe, K.-D., Thalheim, B. (eds.) SDKB 2013. LNCS, vol. 7693, pp. 149–173. Springer, Heidelberg (2013)
13. Decker, H.: Answers that have quality. In: Murgante, B., Misra, S., Carlini, M., Torre, C.M., Nguyen, H.-Q., Taniar, D., Apduhan, B.O., Gervasi, O. (eds.) ICCSA 2013, Part II. LNCS, vol. 7972, pp. 543–558. Springer, Heidelberg (2013)
14. Decker, H., Martinenghi, D.: Modeling, measuring and monitoring the quality of information. In: Heuser, C.A., Pernul, G. (eds.) ER 2009. LNCS, vol. 5833, pp. 212–221. Springer, Heidelberg (2009)
15. Decker, H., Martinenghi, D.: Inconsistency-tolerant Integrity Checking. IEEE Trans. Knowl. Data Eng. **23**(2), 218–234 (2011)
16. Grant, J.: Classifications for inconsistent theories. Notre Dame J. Form. Log. **19**(3), 435–444 (1978)
17. Hunter, A., Konieczny, S.: Approaches to measuring inconsistent information. In: Bertossi, L., Hunter, A., Schaub, T. (eds.) Inconsistency Tolerance. LNCS, vol. 3300, pp. 191–236. Springer, Heidelberg (2005)
18. Lloyd, J., Sonenberg, E., Topor, R.: Integrity constraint checking in stratified databases. J. Log. Program. **4**(4), 331–343 (1987)
19. Mu, K., Liu, W., Jin, Z., Bell, D.: A syntax-based approach to measuring the degree of inconsistency for belief bases. IJAR **52**, 978–999 (2011). Elsevier
20. Staworko, S.: Declarative inconsistency handling in relational and semi-structured databases. Ph.D. thesis, May 2007
21. ten Cate, B., Fontaine, G., Kolaitis, P.: On the data complexity of consistent query answering. Theory Comput. Syst. **57**(4), 843–891 (2015)
22. Wijsen, J.: Database repairing using updates. ACM Trans. Database Syst. **30**(3), 722–768 (2005)