New Measures for Maintaining the Quality of Databases

Hendrik Decker*

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Spain

Abstract. Integrity constraints are a means to model the quality of databases. Measures that size the amount of constraint violations are a means to monitor and maintain the quality of databases. We present and discuss new violation measures that refine and go beyond previous inconsistency measures. They serve to check updates for integrity preservation and to repair violations in an inconsistency-tolerant manner.

1 Introduction

The quality of databases can be identified with the satisfaction of the integrity constraints imposed on the data, and the lack of quality with their violation. In [8,11], we have seen how integrity constrains can be used to model, measure and monitor the quality of information stored in databases.

Building on that idea, we have shown in [10] how to identify, compute and measure cases and causes of integrity violations in order to control the quality of data in terms of their consistency. In [8,11,10], we have defined several measure-based methods which check updates for integrity preservation in an inconsistency-tolerant manner. Inconsistency-tolerant integrity checking (abbr. ITIC) means that only updates that do not increase a measured amount of inconsistency are acceptable, no matter if any constraint violation may already exist before the execution of a given update.

In this paper, we take the idea of using constraints and the measurement of their violation for controlling the quality and integrity of databases two steps further. Firstly, we present several new measures of integrity violation that enable a refined assessment of data quality. Secondly, we show how such measures can be used not only for ITIC, but also for inconsistency-tolerant integrity repair.

In Section 2, we define the formal framework of the remainder. In Section 3, we refine the axiomatization of violation measures developed in [8,11] and define several new measures that go beyond those defined in [10]. In Section 4, we show how the new violation measures can be used to maintain database integrity, by checking updates and repairing inconsistencies. In Section 5, we conclude.

* Partially supported by FEDER and the Spanish grants TIN2009-14460-C03 and TIN2010-17139.

B. Murgante et al. (Eds.): ICCSA 2012, Part IV, LNCS 7336, pp. 170–185, 2012.

2 The Framework

In 2.1, we outline some basic preliminaries. In 2.2 we recapitulate the notion of 'cases' from [8,11]. In 2.3, we extend the notion of 'causes' from [10]. Cases are instances of constraints that are useful for three objectives: simplified integrity checking, quantifying constraint violations and tolerating inconsistency. Causes are the data that are responsible for the violation of constraints, and are of similar use as cases. We use notations and terminology that are known from *datalog* [16] and first-order predicate logic.

2.1 Databases, Completions, Updates, Constraints

An *atom* is an expression of the form $p(t_1, ..., t_n)$, where p is a predicate of arity n ($n \geq 0$); the t_i are either constants or variables. A *literal* is either an atom A or a negated atom $\sim A$. A *database clause* is a universally closed formula of the form $A \leftarrow B$, where the *head* A is an atom and the *body* B is a possibly empty conjunction of literals. If B is empty, A is called a *fact*. If B is not empty, $A \leftarrow B$ is called a *rule*. As is well-known, rules are useful for defining view predicates, as well as for enabling deductive and abductive reasoning capabilities in databases. A *database* is a finite set of database clauses. A datbase is *definite* if there is no negated atom in the body of any of its clauses.

Let us assume a universal Herbrand base $\mathcal{H}$ and a universal set $\mathcal{N}$ of constants, represented w.l.o.g. by natural numbers, that underlie each database.

Let *comp*(D) denote the well-known completion of D [6]. It essentially consists of the if-and-only-if completions (in short, completions) of all predicates in the language of D. For a predicate p, let p_D denote the completion of p in D.

Definition 1. Let D be a database, p a predicate, n the arity of p, $x_1, \ldots, x_n$ the $\forall$-quantified variables in p_D, and θ a substitution of $x_1, \ldots, x_n$.

a) For $A = p(x_1, \ldots, x_n)\theta$, the *completion* of A in D, be obtained by applying θ to p_D, and be denoted by A_D.

b) Let $\underline{comp}(D) = \{A_D \mid A \in \mathcal{H}\}$.

c) Let *if*(D) and *only-if*(D) be obtained by replacing $\leftrightarrow$ in each $A_D \in \underline{comp}(D)$ by $\leftarrow$ and, resp., $\rightarrow$.

d) Let *iff*$(D) =$ *if*$(D) \cup$ *only-if*(D). Let the usual equality axioms of *comp*(D) be associated by default also to *iff*(D).

Clearly, *if*(D) is equivalent to the set of all ground instances of clauses in D. Moreover, *comp*(D), $\underline{comp}(D)$ and *iff*(D) clearly have the same logical consequences. However, the characterization of causes in 2.2.3 by subsets of *iff*(D) is more precise than it could be if subsets of *comp*(D) were used instead.

We may use ';' instead of ',' to delimit elements of sets since ',' is also used to denote conjunction in the body of clauses. Symbols $\models$, $\Rightarrow$ and $\Leftrightarrow$ denote logical consequence (i.e., truth in all Herbrand models), meta-implication and, resp., meta-equivalence. By overloading, we use $=$ as identity predicate, assignment in substitutions, or meta-level equality; $\neq$ is the negation of $=$.

An *update* is a finite set of database clauses to be inserted or deleted. For an update U of a database D, we denote the database in which all inserts in U are added to D and all deletes in U are removed from D, by D^U. An *update request* in a database D is a sentence R that is requested to become *true* by updating D. An update U *satisfies* an update request R in D if $D^U(R) = true$. View updating is a well-known special kind of satisfying update requests. From now on, repairs are treated as updates, and repairing as satisfying specific update requests.

An *integrity constraint* (in short, *constraint*) is a sentence which can always be represented by a *denial clause*, i.e., a universally closed formula of the form $\leftarrow B$, where the body B is a conjunction of literals that asserts what *should not* hold in any state of the database. If the original specification of a constraint by a sentence I expresses what *should* hold, then a denial form of I can be obtained by an equivalence-preserving re-writing of $\leftarrow \sim I$ as proposed, e.g., in [7], that results in a denial the predicates of which are defined by clauses to be added to the database. An *integrity theory* is a finite set of constraints.

From now on, let the symbols D, IC, I, U and adornments thereof always stand for a database, an integrity theory, a constraint and, resp., an update, each of which is assumed, as usual, to be range-restricted [7].

For each sentence F, and in particular for each integrity constraint, we write $D(F) = true$ (resp., $D(F) = false$) if F evaluates to *true* (resp., *false*) in D. Similarly, we write $D(IC) = true$ (resp., $D(IC) = false$) if each constraint in IC is satisfied in D (resp., at least one constraint in IC is violated in D). Let $vioCon(D, IC)$ denote the set of violated constraints in D.

2.2 Cases

For each constraint I, a *case* of I is an instance of I obtained by substituting the variables in I with (not necessarily ground) terms of the underlying language. This definition of cases is a simpler version of a more encompassing variant in [12], where cases have been defined for the purpose of inconsistency-tolerant integrity checking of constraints in a more general syntax.

Reasoning with cases of I instead of I itself lowers the cost of integrity maintenance, since, the more variables in I are instantiated with ground values, the easier the evaluation of the so-obtained case tends to be. Also, to know which particular cases of a constraint are violated may be useful for repairing, since it turns out to be easier, in general, to identify and eliminate the causes of integrity violation if the violated cases are made explicit.

Let $Cas(IC)$ denote the set of all ground cases of each $I \in IC$. Further, let $vioCon(D, IC) = \{I \mid I \in IC,\ D(I) = false\}$, i.e., the set of all constraints in IC that are violated in D, and $vioCas(D, IC) = \{C \mid C \in Cas(IC),\ D(C) = false\}$, i.e., the set of all violated ground cases of IC in D.

The usefulness of cases for simplified integrity checking is well-known and well documented, e.g. in [5]. The use of $vioCon(D, IC)$ and $vioCas(D, IC)$ for measuring the inconsistency of (D, IC) is addressed in Section 3, their use for inconsistency-tolerant integrity maintenance in Section 4.

2.3 Causes

As in [10], we are going to define a 'cause' of the violation of a constraint $I = \leftarrow B$ in a database D as a minimal explanation of why I is violated in D, i.e., why the existential closure $\exists B$ of B is *true* in D. However, the definition of causes below is much more general than its predecessor in [10]. The latter applied only to definite databases and integrity theories without negation in the body of denials. In this paper, that restriction is dropped. A significant consequence of this generalization is that the logic to be used for reasoning with cases and causes of constraints must comply with the non-monotonicity of negation in databases.

In Section 3, causes are used for measuring inconsistency, and in Section 4 for measure-based inconsistency-tolerant integrity maintenance.

Definition 2. Let D be a database and $I = \leftarrow B$ an integrity constraint such that $D(\exists B) = \mathit{true}$. A subset E of *iff*(D) is called a *cause of the violation of* I *in* D if $E \models \exists B$, and for each proper subset E' of E, $E' \not\models \exists B$.

We also say that E is a *cause of* $\exists B$ *in* D if E is a cause of the violation of $\leftarrow B$ in D. Moreover, we say that, for an integrity theory IC, E is a *cause of the violation of* IC *in* D if E is a cause of the violation of a denial form of the conjunction of all constraints in IC.

Clearly, E is a cause of the violation of each denial form of the conjunction of all $I \in IC$ if and only if E is a cause of the violation of some $I \in IC$ and there is no cause E' of any constraint in IC such that $E' \subsetneq E$.

For easy reading, we represent elements of *only-if*(D) in a simplified form, if possible, in the subsequent examples of causes. Simplifications are obtained by replacing ground equations with their truth values and by common equivalence-preserving rewritings for the composition of subformulas with *true* or *false*.

Example 1.

a) Let $D = \{p \leftarrow q, \sim r;\ q\}$.
The only cause of the violation of $\leftarrow p$ in D is $D \cup \{\sim r\}$.

b) Let $D = \{p(x) \leftarrow q(x), r(x);\ q(1);\ q(2);\ r(2);\ s(1);\ s(2)\}$. The only cause of the violation of $\leftarrow s(x), \sim p(x)$ in D is $\{s(1),\ p(1) \rightarrow q(1) \wedge r(1),\ \sim r(1)\}$.

c) Let $D = \{p \leftarrow q(1,x);\ q(2,y) \leftarrow r(y);\ r(1)\}$.
The only cause of $\sim p$ in D is $\{p \rightarrow \exists x\, q(1,x)\} \cup \{\sim q(1,i) \mid i \in \mathcal{N}\}$.

d) Let $D = \{p \leftarrow q(x,x);\ q(x,y) \leftarrow r(x), s(y);\ r(1);\ s(2)\}$. Each cause of $\sim p$ in D contains $\{p \rightarrow \exists x\, q(x,x)\} \cup \{q(i,i) \rightarrow r(i) \wedge s(i)) \mid i \in \mathcal{N}\} \cup \{\sim r(2), \sim s(1)\}$ and, for each $j > 2$ in $\mathcal{N}$, either $\sim r(j)$ or $\sim s(j)$, and nothing else.

e) Let $D = \{p(x) \leftarrow r(x);\ r(1)\}$ and $I = \exists x(r(x) \wedge \sim p(x))$. A denial form of I is $\leftarrow \mathit{vio}$, where *vio* is defined by $\{\mathit{vio} \leftarrow \sim q;\ q \leftarrow r(x), \sim p(x)\}$, where q is a fresh 0-ary predicate. Thus, the causes of the violation of I in D are the causes of *vio* in $D' = D \cup \{\mathit{vio} \leftarrow \sim q;\ q \leftarrow r(x), \sim p(x)\}$. Thus, for each $\mathcal{K} \subseteq \mathcal{N}$ such that $1 \in \mathcal{K}$, $\{\mathit{vio} \leftarrow \sim q\} \cup \{p(i) \leftarrow r(i) \mid i \in \mathcal{K}\} \cup \{q \rightarrow \exists x(r(x) \wedge \sim p(x))\} \cup \{\sim r(i) \mid i \notin \mathcal{K}\}$ is a cause of *vio* in D'.

The following example shows that causes are not compositional, i.e., the causes of the violation of an integrity theory IC are not necessarily the union of the causes of the violation of the constraints in IC.

Example 2. Let $D = \{r(1,1);\, s(1)\}$, $I_1 = \leftarrow r(x,x)$, $I_2 = \leftarrow r(x,y), s(y)$ and $IC = \{I_1, I_2\}$. The only cause of the violation of IC in D is $\{r(1,1)\}$, which is a proper subset of the single cause D of the violation of I_2 in D.

Let $vioCau(D, IC)$ be the set of all causes of the violation of IC in D.

Clearly, $vioCau(D, IC)$ is analogous to *vioCas* as defined in 2.2. While *vioCas* locates inconsistency by focusing on violated constraints, *vioCau* localizes inconsistency on the data that are responsible for integrity violations.

3 Violation Measures

Violation measures are a special kind of inconsistency measures [15]. Their purpose is to size the amount of integrity violation in databases. In 3.1, we semi-formally sketch our approach of violation measures. In 3.2, we define this concept formally. In 3.3, we first recapitulate some measures already defined in [8,11,10], and then introduce and discuss some new ones. In 3.4, we discuss some properties that are commonly associated to measures and investigate to which extent they apply to violation measures.

3.1 Conceptualizing Violation Measures

In 3.2, we are going to define an abstract concept of violation measures as a mapping from pairs (D, IC) to a set $\mathbb{M}$ that is structured by a partial order $\preccurlyeq$ with an infimum o, a distance δ and an addition $\oplus$ with neutral element o.

The partial order $\preccurlyeq$ allows to compare the amount of inconsistency in two pairs of databases and integrity theories, and in particular in consecutive states (D, IC) and (D^U, IC). With the distance δ, the difference, i.e., the increase or decrease of inconsistency between D and D^U, can be sized. The addition $\oplus$ allows to state a standard metric property for δ, and o is, at a time, the smallest element of $\preccurlyeq$ and the neutral element of $\oplus$.

Thus, there are various measurable criteria according to which an update U can be checked while tolerating extant inconsistencies, e.g., if U does not increase the amount of inconsistency, or if D^U does not trespass a certain threshold of inconsistency, or if any increase of inconsistency brought about by U is negligible.

In traditional measure theory [1], a measure μ maps elements of a measure space $\mathbb{S}$ (typically, a set of sets) to a metric space $(\mathbb{M}, \preccurlyeq, \delta)$ (typically, $\mathbb{M} = \mathbb{R}_0^+$, i.e., the non-negative real numbers, often with an additional greatest element ∞, $\preccurlyeq = \leq$, and $\delta = |-|$, i.e., the absolute difference). For $S \in \mathbb{S}$, $\mu(S)$ usually tells how 'big' S is. Standard properties are that μ is *positive definite*, i.e., $\mu(S) = 0 \Leftrightarrow S = \emptyset$, μ is *additive*, i.e., $\mu(S \cup S') = \mu(S) + \mu(S')$, for disjoint sets $S, S' \in \mathbb{S}$, and μ is *monotone*, i.e., if $S \subseteq S'$, then $\mu(S) \leq \mu(S')$. The distance δ maps $\mathbb{M} \times \mathbb{M}$ to itself, for determining the difference between measured entities.

Similarly, for assessing inconsistency in databases, a violation measure ν as defined in 3.2 maps pairs (D, IC) to a metric space that, like $\mathbb{R}_0^+$, has a partial order and an addition with neutral element. The purpose of $\nu(D, IC)$ is to size the amount of inconsistency in (D, IC), e.g., by counting or comparing sets of cases or causes of constraint violations.

3.2 Formalizing Violation Measures

Definitions 3 and 4 below specialize the traditional concepts of metric spaces and measures, since they are made up for databases and integrity violations. Yet, in a sense, these definitions also generalize the traditional concepts, since they allow to size and compare different amounts of inconsistency without necessarily quantifying them numerically. For example, with $\mathbb{M} = 2^{Cas(IC)}$ (powerset of $Cas(IC)$ as defined in 2.2.2), $\preccurlyeq = \subseteq$ (subset), $\delta = \ominus$ (symmetric set difference), $\oplus = \cup$ (set union) and $o = \emptyset$ (empty set), it is possible to measure the inconsistency of (D, IC) by sizing $vioCas(D, IC)$.

Definition 3. A structure $(\mathbb{M}, \preccurlyeq, \delta, \oplus, o)$ is called a *metric space for integrity violation* (in short, a *metric space*) if $(\mathbb{M}, \oplus)$ is a commutative semi-group with neutral element o, $\preccurlyeq$ is a partial order on $\mathbb{M}$ with infimum o, and δ is a distance on $\mathbb{M}$. More precisely, for each $m, m', m'' \in \mathbb{M}$, the following properties (1)–(4) hold for $\preccurlyeq$, (5)–(8) for $\oplus$, and (9)–(11) for δ.

$$m \preccurlyeq m \quad (reflexivity) \tag{1}$$

$$m \preccurlyeq m',\ m' \preccurlyeq m \ \Rightarrow\ m = m' \quad (antisymmetry) \tag{2}$$

$$m \preccurlyeq m',\ m' \preccurlyeq m'' \ \Rightarrow\ m \preccurlyeq m'' \quad (transitivity) \tag{3}$$

$$o \preccurlyeq m \quad (infimum) \tag{4}$$

$$m \oplus (m' \oplus m'') = (m \oplus m') \oplus m'' \quad (associativity) \tag{5}$$

$$m \oplus m' = m' \oplus m \quad (commutativity) \tag{6}$$

$$m \oplus o = m \quad (neutrality) \tag{7}$$

$$m \preccurlyeq m \oplus m' \quad (\oplus\text{-}monotonicity) \tag{8}$$

$$\delta(m, m') = \delta(m', m) \quad (symmetry) \tag{9}$$

$$\delta(m, m) = o \quad (identity) \tag{10}$$

$$\delta(m, m') \preccurlyeq \delta(m, m'') \oplus \delta(m'', m') \quad (triangle\ inequality) \tag{11}$$

Let $m \prec m'$ ($m \succ m'$) denote that $m \preccurlyeq m'$ (resp., $m' \preccurlyeq m$) and $m \neq m'$.

Example 3. $(\mathbb{N}_0, \leq, |-|, +, 0)$ is a metric space for integrity violation, where $\mathbb{N}_0$ is the set of non-negative integers. In this space, $vioCon(D, IC)$, $vioCas(D, IC)$ or $vioCau(D, IC)$ can be counted and compared. As already indicated, these three sets may also be sized and compared in the metric spaces $(2^X, \subseteq, \ominus, \cup, \emptyset)$, where X stands for IC, $Cas(IC)$ or $\mathit{iff}(D)$, respectively.

The following definition generically characterizes violation measures, whose ranges are metric spaces such as those in Example 3.

Definition 4. A *violation measure* (in short, a *measure*) is a function ν that maps pairs (D, IC) to a metric space $(\mathbb{M}, \preccurlyeq, \delta, \oplus, o)$ for integrity violation.

3.3 New Violation Measures

We continue with examples of violation measures that are going to accompany us throughout the remainder of the paper. Preliminary versions of some of them have already been presented in [8,11,10]. However, the axiomatization of those previous versions is more shallow than in 3.2. Also the study of various properties of violation measures in 3.4 is very scant in the cited predecessors.

Example 4. A coarse measure β is defined by $\beta(D, IC) = D(IC)$ with the binary metric space $(\{true, false\}, \preccurlyeq, \tau, \wedge, true)$, where $\preccurlyeq$ and τ are defined by stipulating $true \prec false$ (i.e., satisfaction means lower inconsistency than violation), and, resp., $\tau(v, v') = true$ if $v = v'$, else $\tau(v, v') = false$, for $v, v' \in \{true, false\}$, i.e., the value of $\tau(v, v')$ is the truth value of the logical equivalence $v \leftrightarrow v'$. Clearly, β and its metric space reflect the classical logic distinction that a set of formulas is either consistent or inconsistent, without any further differentiation of different degrees of inconsistency. The meaning of τ is that each consistent pair (D, IC) is equally good, and each inconsistent pair (D, IC) is equally bad.

Example 5. Two measures ι and $|\iota|$ that are quite straightforward are characterized by comparing and, resp., counting the set of violated constraints in the integrity theory of the database schema. The formal definition of these measures is given by the equations $\iota(D, IC) = vioCon(IC, D)$ and $|\iota|(D, IC) = |\iota(D, IC)|$, where $|\,.\,|$ is the cardinality operator, with metric spaces $(2^{IC}, \subseteq, \ominus, \cup, \emptyset)$ and, resp., $(\mathbb{N}_0^+, \leq, |-|, +, 0)$.

Example 6. Two measures that are more fine-grained than those in Example 5 are given by $\zeta(D,IC) = vioCas(IC,D)$ and $|\zeta|(D,IC) = |\zeta(D, IC)|$, with metric spaces $(2^{Cas(IC)}, \subseteq, \ominus, \cup, \emptyset)$ and, resp., $(\mathbb{N}_0^+, \leq, |-|, +, 0)$.

Example 7. Similar to cases, cause-based measures can be defined by the equations $\kappa(D, IC) = vioCau(IC, D)$ and $|\kappa|(D, IC) = |\kappa(D, IC)|$, with metric spaces $(2^{iff(D)}, \subseteq, \ominus, \cup, \emptyset)$ and, resp., again $(\mathbb{N}_0^+, \leq, |-|+, 0)$. Specific differences between measures based on cases (as in Example 6) and measures based on causes (as in this example) have been discussed in [10].

Other violation measures are discussed in [8,11] among them two variants of an inconsistency measure in [14] that are applicable to databases with integrity constraints, based on quasi-classical models [2]. Essentially, both violation measures size the set of conflicting atoms in (D, IC), i.e., atoms A such that both A and $\sim A$ are *true* in the minimal quasi-classical model of $D \cup IC$. Hence, their metric spaces are $(2^{\mathcal{H}^*}, \subseteq, \ominus, \cup, \emptyset)$ where $\mathcal{H}^*$ is the union of $\mathcal{H}$ and $\{\sim A \mid A \in \mathcal{H}\}$, and, resp., $(\mathbb{N}_0^+, \leq, |-|+, 0)$.

Some more new measures are going to be identified in 3.4.1 and 4.1.

3.4 Properties of Violation Measures

As opposed to classical measure theory and previous work on inconsistency measures, Definition 4 does not require any axiomatic property of measures, such as positive definiteness, additivity or monotony. These usually are required for each traditional measure μ, as already mentioned in 3.1. We are going to look at such properties, and argue that positive definiteness is not cogent, and both additivity and monotony are invalid for many databases.

In 3.4.1, we discuss the standard axiom of positive definiteness of measures, including some weakenings thereof. In 3.4.2, we show the standard axiom of additivity of measures is invalid for violation measures. In 3.4.3, also the standard axiom of monotonicity of measures is dismissed for violation measures, and some valuable weakenings thereof are proposed.

3.4.1 Positive Definiteness

For traditional measures μ, positive definiteness means that $\mu(S) = 0$ if and only if $S = \emptyset$, for each $S \in \mathbb{S}$. For violation measures ν, that takes the form

$$\nu(D, IC) = o \;\Leftrightarrow\; D(IC) = true \quad (positive\ definiteness) \tag{12}$$

for each pair (D, IC).

A property corresponding to (12) is postulated for inconsistency measures in [13]. However, we are going to argue that (12) is not cogent for violation measures, and that even two possible weakenings of (12) are not persuasive enough as sine-qua-non requirements.

At first, (12) may seem to be most plausible as an axiom for any reasonable inconsistency measure, since it assigns the lowest possible inconsistency value o precisely to those databases that totally satisfy all of their constraints. In fact, it is easy to show the following result.

Theorem 1. Each of the measures $\beta, \iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|$ fulfills (12).

Thus, in particular $|\zeta|$, which counts the number of violated ground cases, complies with (12). Now, let the measure ζ' be defined by the following modification of $|\zeta|$: $\zeta'(D, IC) = 0$ if $|\zeta|(D, IC) \in \{0,1\}$ else $\zeta'(D, IC) = |\zeta|(D, IC)$. Thus, ζ' considers each inconsistency that consists of just a single violated ground case as insignificant. Hence, ζ' does not obey (12) but can be, depending on the application, a very reasonable violation measure that tolerates negligible amounts of inconsistency.

Even the weakening

$$D(IC) = true \;\Rightarrow\; \nu(D, IC) = o \tag{13}$$

of (12) is not a cogent requirement for all reasonable violation measures, as witnessed by the measure σ, as defined below. It takes a differentiated stance with regard to integrity satisfaction and violation, by distinguishing between satisfaction, satisfiability and violation of constraints, similar to [19] [17].

The measure σ be defined by incrementing a count of 'problematic' ground cases of constraints by 1 for each ground case that is satisfiable but not a theorem

of the completion of the given database, and by 2 for each ground case that is violated. Hence, by the definitions of integrity satisfaction and violation in [17], there are pairs (D, IC) such that IC is satisfied in D but $\sigma(D, IC) > 0$.

Another measure ϵ that does not respect (13) can be imagined as follows, for databases with constraints of the form $I = \leftarrow p(x), x > th,$ where $p(x)$ is a relation defined by some aggregation of values in the database, meaning that I is violated if $p(x)$ holds for some x that trespasses a certain threshold *th*. Now, suppose that ϵ assigns a minimal non-zero value to (D, IC) whenever I is still satisfied in D but $D(p(th)) = true$, so as to indicate that I is at risk of becoming violated. Hence, there are pairs (D, IC) such that $\nu = \epsilon$ contradicts (13).

Also the requirement

$$\nu(D, \emptyset) \;=\; o \tag{14}$$

which weakens (13) even further, is not indispensable, although analogons of (14) are standard in the literature on classical measures and inconsistency measures. In fact, it is easy to imagine a measure that assigns a minimal non-zero value of inconsistency to a database without integrity constraints. That value can then be interpreted as a warning that there is a non-negligible likelihood of inconsistency in a database where no constraints are imposed, be it out of neglect, or for trading off consistency for efficiency, or due to any other reason.

So, in the end, only the rather bland property $\nu(\emptyset, \emptyset) = \mathrm{o}$ remains as a weakening of (12) that should be required from violation measures.

3.4.2 Additivity

For traditional measures μ, additivity means $\mu(S \cup S') = \mu(S) + \mu(S'),$ for each pair of disjoint sets $S, S' \in \mathbb{S}$. For violation measures ν, additivity takes the form

$$\nu(D \cup D',\ IC \cup IC') \;=\; \nu(D, IC) \oplus \nu(D', IC') \quad (\textit{additivity}) \tag{15}$$

for each (D, IC), (D', IC') such that D and D' as well as IC and IC' are disjoint.

Additivity is standard for traditional measures. However, (15) is invalid for violation measures, as shown by the following example.

Example 8. Let $D = \{p\}$, $IC = \emptyset$, $D' = \emptyset$, $IC' = \{\leftarrow p\}$. Clearly, $D(IC) = true$ and $D'(IC') = true,$ thus $|\zeta|(D, IC) + |\zeta|(D', IC') = 0,$ but $|\zeta|(D \cup D',\ IC \cup IC') = 1$.

3.4.3 Monotony

For traditional measures μ, monotony means $S \subseteq S' \Rightarrow \mu(S) \preccurlyeq \mu(S')$, for each pair of sets $S, S' \in \mathbb{S}$. For violation measures ν, monotony takes the form

$$\mathcal{D} \subseteq D',\ IC \subseteq IC' \quad \Rightarrow \quad \nu(D, IC) \preccurlyeq \nu(D', IC') \quad (\nu\textit{-monotonicity}) \tag{16}$$

for each pair of pairs (D, IC), (D', IC').

A property corresponding to (16) is postulated for inconsistency measures in [13]. For *definite* databases and integrity theories (i.e., the bodies of clauses do not contain any negative literal), it is easy to show the following result.

Theorem 2. For each pair of definite databases D, D' and each pair of definite integrity theories IC, IC', each of the measures $\beta, \iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|$ fulfills (16).

However, due to the non-monotonicity of negation in the body of clauses, (16) is not valid for non-definite databases or non-definite integrity theories, as shown by Example 9, in which the foreign key constraint $\forall x(q(x,y) \rightarrow \exists z\, s(x,z))$ on the x-column of q referencing the x-column of s is rewritten into denial form (we ignore the primary key constraint on the x-column of s since it is not relevant).

Example 9. Let $D = \{p(x) \leftarrow q(x,y), \sim r(x);\ r(x) \leftarrow s(x,z);\ q(1,2);\ s(2,1)\}$ and $IC = \{\leftarrow p(x)\}$. Clearly, $D(IC) = \mathit{false}$ and $|\zeta|(D, IC) = 1$. For $D' = D \cup \{s(1,1)\}$ and $IC' = IC$, we have $D'(IC') = \mathit{true}$, hence $|\zeta|(D', IC') = 0$.

Now, we are going to look at two weakenings of (16) that hold also for non-definite databases and integrity theories. In fact, (16) is already a weakening of (15), since it is easily shown that (15) and (8) entail (16). Hence, any valid weakening of (16) can also be understood as a valid weakening of (15).

The first weakening requires that the inconsistency in databases that violate integrity is never lower than the inconsistency in databases that satisfy integrity. Formally, for each pair of pairs (D, IC), (D, IC'), the following property is asked to hold.

$$D(IC) = \mathit{true},\ D'(IC') = \mathit{false} \ \Rightarrow\ \mu(D, IC) \preccurlyeq \mu(D', IC') \tag{17}$$

It is easy to show the following result.

Theorem 3. Each of the measures $\beta, \iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|$ fulfills (17).

A property that is slightly stronger than (17) has been postulated in [8,11]. It is obtained by replacing $\preccurlyeq$ in (17) by $\prec$. It also holds for all measures from Subsection 3.3. Yet, similar to (12), it does not hold for measures ζ', σ and ϵ, as defined in 3.4.1, while (17) does hold for those measures.

The second weakening of (16) has been postulated in [10]. It requires that, for each D, the values of ν grow monotonically with growing integrity theories.

$$IC \subseteq IC' \ \Rightarrow\ \nu(D, IC) \preccurlyeq \nu(D,\ IC') \tag{18}$$

Since (18) weakens (16), the following result is entailed by Theorem 3 for β, $\iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|$, and can be shown also for ζ', σ, ϵ.

Theorem 4. Each of the measures $\beta, \iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|, \zeta', \sigma, \epsilon$ fulfills (18).

4 Integrity Maintenance

To maintain integrity, constraint violations should be prevented or repaired. For prevention, a common approach is to check if updated preserve integrity. For repairing, methods described, e.g., in [20] may be used. However, it may be impractical or unfeasible to avoid inconsistency, or to repair all violated constraints at once. Thus, an inconsistency-tolerant approach to integrity maintenance is

needed. As we are going to see, that can be achieved by violation measures. In fact, even in the presence of persisting inconsistency, the use of measures can prevent the increase of inconsistency across updates. Measures also are useful for controling that the amount of inconsistency never exceeds given thresholds.

In 4.1, we revisit measure-based inconsistency-tolerant integrity checking (abbr. ITIC). Also, we show how inconsistency can be confined by assigning weights to violated cases of constraints, which goes beyond the measures seen so far. Moreover, we show how to generalize measure-based ITIC by allowing for certain increases of inconsistency that are bounded by some thresholds. In 4.2, we outline how measure-based inconsistency-tolerant integrity checking can be used also for making repairing inconsistency-tolerant.

4.1 Measure-Based Inconsistency-Tolerant Integrity Checking

Definition 5, below, characterizes integrity checking methods that may accept updates if there is no increase of inconsistency, no matter if there is any extant constraint violation or not. It abstractly captures measure-based ITIC methods as black boxes, of which nothing but their i/o interface is observable. More precisely, each method $\mathcal{M}$ is described as a mapping from triples (D, IC, U) to $\{ok, ko\}$. Intuitively, *ok* means that U does not increase the amount of measured inconsistency, and *ko* that it may.

Definition 5. (*Inconsistency-tolerant Integrity Checking*, abbr. *ITIC*)
An *integrity checking method* maps triples (D, IC, U) to $\{ok, ko\}$. For a measure $(\nu, \preccurlyeq)$, a method $\mathcal{M}$ is called *sound* (*complete*) *for* ν*-based ITIC* if, for each (D, IC, U), (19) (resp., (20)) holds.

$$\mathcal{M}(D, IC, U) = ok \;\Rightarrow\; \nu(D^U, IC) \preccurlyeq \nu(D, IC) \tag{19}$$

$$\nu(D^U, IC) \preccurlyeq \nu(D, IC) \;\Rightarrow\; \mathcal{M}(D, IC, U) = ok \tag{20}$$

Each $\mathcal{M}$ that is sound for ν-based ITIC is also called a ν-based method.

Intuitively, (19) says: $\mathcal{M}$ is sound if, whenever it outputs *ok*, the amount of violation of IC in D as measured by ν is not increased by U. Conversely, (20) says: $\mathcal{M}$ is complete if it outputs *ok* whenever the update U that is checked by $\mathcal{M}$ does not increase the amount of integrity violation.

As opposed to ITIC, traditional integrity checking (abbr. TIC) imposes the *total integrity* requirement. That is, TIC additionally requires $D(IC) = true$ in the premises of (19) and (20). The measure used in TIC is β (cf. Example 4). Since ITIC is defined not just for β but for any violation measure ν, and since TIC is not applicable if $D(IC) = false$, while ITIC is, Definition 5 generalizes TIC. Moreover, the definition of ITIC in [12] is equivalent to Definition 5 for $\nu = \zeta$. Hence, the latter also generalizes ITIC as defined in [12].

In [12], we have shown that the total integrity requirement is dispensable for most TIC approaches. Similar to corresponding proofs in [12], it can be shown that not all, but most TIC methods, including built-in integrity checks in common DBMSs, are ν-based, for each $\nu \in \{\iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|\}$. The following results are easily shown by applying the definitions.

Theorem 5. If a method $\mathcal{M}$ is ν-based, then it is $|\nu|$-based, for each $\nu \in \{\iota, \zeta, \kappa\}$. If $\mathcal{M}$ is κ-based, then it is ζ-based. If $\mathcal{M}$ is ζ-based, then it is ι-based. The converse of none of these implications holds.

Example 10, below, illustrates how the measures $|\iota|, |\zeta|, |\kappa|$ that count violated constraints, cases or causes thereof can be generalized by assigning weight factors to the counted entities. Such weights are useful for modeling application-specific degrees of violated integrity. A simple variant of such an assignment comes into effect whenever 'soft' constraints that *ought to* be satisfied are distinguished from 'hard' constraints that *must* be satisfied.

Example 10. Let *lr* and *hr* be two predicates that model a low, resp., high risk. Further, $I_1 = \leftarrow lr(x)$, $I_2 = \leftarrow hr(x)$, be a soft, resp., hard constraint for protecting against low and, resp., high risks, where *lr* and *hr* are defined by $lr(x) \leftarrow p(y,z)$, $x = y + z$, $x > th$, $z \geq y$ and $hr(x) \leftarrow p(y,z)$, $x = y + z$, $x > th$, $y > z$, resp., where *th* is a threshold value that should not be exceeded. and $p(8,3)$ be the only cause of integrity violation in some database D. Now, for each $\nu \in \{\iota, \zeta, \kappa\}$, no ν-based method would accept the update $U = \{\mathit{delete}\ p(8,3), \mathit{insert}\ p(3,8)\}$, although the high risk provoked by $p(8,3)$ is diminished to the low risk produced by $p(3,8)$. However, measures that assign weights to cases of I_2 that are higher than those of I_1 can avoid that problem. For instance, consider the measure ω that counts the numbers n_1 and n_2 of violated cases of I_1 and, resp., I_2 in D, and assigns $f_1 n_1 + f_2 n_2$ to $(D, \{I_1, I_2\})$, where $0 < f_1 < f_2$. Clearly, each ω-based method will accept U. In fact, for $\nu \in \{|\iota|, |\zeta|, |\kappa|\}$, also each ν-based method would accept U, but it is easy to imagine a slightly more elaborated update U' such that ν-based methods would not accept U' but ω-based methods would.

4.2 Repairs

Roughly, repairing means to compute and execute an update in order to eliminate integrity violation. Thus, each repair can be identified with an update.

In 4.2.1, we formalize repairs and illustrate them by examples. In 4.2.2, we outline how to compute repairs.

4.2.1 Formalizing Repairs

In [12], we have distinguished *total* and *partial* repairs. The former eliminate all inconsistencies, the latter only some. Partial repairs tolerate inconsistency, since violated constraints may persist, as illustrated by Example 11.

Example 11. Let $D = \{p(a,b,c), p(b,b,c), p(c,b,c), q(a,c), q(c,b), q(c,c)\}$ and $IC = \{\leftarrow p(x,y,z), \sim q(x,z);\ \leftarrow q(x,x)\}$. Clearly, the violated cases of IC in D are $\leftarrow p(b,b,c), \sim q(b,c)$ and $\leftarrow q(c,c)$. Each of the updates $U_1 = \{\mathit{insert}\ q(b,c)\}$ and $U_2 = \{\mathit{delete}\ p(b,b,c)\}$ is a partial repair of (D, IC), since both fix the violation of $\{\leftarrow p(b,b,c), \sim q(b,c)\}$ in D. Similarly, $U_3 = \{\mathit{delete}\ q(c,c)\}$ is a partial repair that fixes the violation of $\{\leftarrow q(c,c)\}$ in D.

Sadly, partial repairs may cause new violations, as shown in Example 12.

Example 12. Consider again Example 11. As opposed to U_1 and U_2, U_3 causes a new violation: $\leftarrow p(c,b,c), \sim q(c,c)$ is satisfied in D but not in D^{U_3}. Thus, the partial repair $U_4 = \{delete\, q(c,c);\ delete\, p(c,b,c)\}$ is needed to eliminate the violation of $\leftarrow q(c,c)$ in D without causing any violation that did not exist before executing the partial repair.

Definition 6, below, generalizes the definition of partial repairs by requiring that each repair must decrease the measured amount of integrity violation.

Definition 6. (*Repair*) Let D be a database, IC an integrity theory such that $D(IC) = false$, ν a violation measure and U an update.

a) U is said to *preserve integrity wrt.* ν if $\nu(D^U, IC) \preccurlyeq \nu(D, IC)$ holds.

b) For a proper subset S of $Cas(IC)$ such that $D(S) = false$ and $D^U(S) = true$, U is called a *partial repair* of (D, IC).

c) U is called a ν*-based repair* of (D, IC) if $\nu(D^U, IC) \prec \nu(D, IC)$ holds. If, additionally, $D^U(IC) = false$, U is also called a ν*-based patch* of (D, IC). Else, if $D^U(IC) = true$, U is called a *total repair* of (D, IC).

Definition 6 could be sightly modified by replacing all occurrence of conditions $D^U(IC) = false$ and $D^U(IC) = true$ by $\nu(D, IC) \succ o$ and $\nu(D, IC) = o$, respectively. For each $\nu \in \{\iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|\}$, that replacement yields a definition that is equivalent to Definition 6. Moreover, it is easy to show the following.

Theorem 6. For each pair (D, IC) and each $\nu \in \{\iota, |\iota|, \zeta, |\zeta|\}$, each ν-based patch of (D, IC) is a partial repair of (D, IC).

Note that the converse of Theorem 6 does not hold, as seen in Example 12. Theorem 6 also does not hold for $\nu \in \{\kappa, |\kappa|\}$, since the violation of some case C may have n causes, $n > 0$, in some database D, and a repair U may just eliminate one of the causes that violate C. Then, for $\nu \in \{\kappa, |\kappa|\}$, $\nu(D^U, IC) \prec \nu(D, IC)$, i.e., U a ν-based patch but not a partial repair of (D, IC) since $vioCas(D, IC) = vioCas(D^U, IC)$, hence $D(S) = D^U(S) = false$.

In the literature, repairs usually are required to be total and, in some sense, minimal. Mostly, subset-minimality is opted for, but several other notions of minimality exist [4] [18]. Note that Definition 6 does not involve any notion of minimality. However, each repair in Example 11 is subset-minimal.

Unpleasant side effects of repairs such as U_3 can be avoided by checking if a given partial repair is a patch with any convenient measure-based method, as expressed in the following result. It follows from Definitions 5 and 6.

Theorem 7. For each tuple (D, IC), each partial repair U of (D, IC), each measure ν and each ν-based method $\mathcal{M}$, U is a ν-based patch if $\mathcal{M}(D, IC, U) = ok$.

4.2.2 Computing Repairs

Repairs can be computed by update methods, defined as follows.

Definition 7. An *update method* is an algorithm that, for each database D and each update request R, computes candidate updates $U_1, \ldots, U_n$ $(n \geq 0)$ such that $D^{U_i}(R) = \mathit{true}$ $(1 \leq i \leq n)$. For a measure ν, an update method $\mathcal{UM}$ is *integrity-preserving wrt.* ν if each U_i computed by $\mathcal{UM}$ preserves integrity wrt. ν.

Integrity-preserving update methods can be used to compute patches and repairs wrt. any measure ν, as shown in [12] for the special case of $\nu = \zeta$. Theorem 8 below generalizes that result.

For an update request R in a database D, several update methods in the literature work in two phases. First, a candidate update U such that $D^U(R) = \mathit{true}$ is computed. Then, U is checked for integrity preservation by some TIC method. If that check is positive, U is accepted. Else, U is rejected and another candidate update, if any, is computed and checked. Hence, Theorem 8, below, follows from Definition 7 and Theorem 7.

Theorem 8. For each measure ν, each update method that uses ν-based ITIC to check its computed candidate updates is integrity-preserving wrt. ν.

Example 13 shows what can go wrong if an update method that is not integrity-preserving is used.

Example 13. Let $D = \{q(x) \leftarrow r(x), s(x);\ \ p(a,a)\}$, R the view update request to insert $q(a)$, and $IC = \{\leftarrow p(x,x);\ \leftarrow p(a,y), q(y)\}$. To satisfy R, most update methods compute the candidate update $U = \{\mathit{insert}\ r(a);\ \mathit{insert}\ s(a)\}$. To check if U preserves integrity, most methods compute the simplification $\leftarrow p(a,a)$ of the second constraint in IC. For avoiding a possibly expensive disk access for evaluating the simplified case $\leftarrow p(a,a)$ of $\leftarrow p(a,y), q(y)$, TIC methods that are not inconsistency-tolerant may use the invalid premise that $D(IC) = \mathit{true}$, by reasoning as follows. The constraint $\leftarrow p(x,x)$ in IC is not affected by U and subsumes $\leftarrow p(a,a)$; hence, IC remains satisfied in D^U. Thus, such methods wrongly conclude that U preserves integrity, since the case $\leftarrow p(a,y), q(y)$ is satisfied in D but violated in D^U. By contrast, each ITIC method rejects U, so that $U' = U \cup \{\mathit{delete}\, p(a,a)\}$ can be computed for satisfying R. Clearly, U' preserves integrity, and even removes the violated case $\leftarrow p(a,a)$.

The following example illustrates a general approach of how patches and total repairs can be computed by update methods off the shelve.

Example 14. Let $S = \{\leftarrow B_1, \ldots, \leftarrow B_n\}$ $(n \geq 0)$ be a set of cases of constraints in an integrity theory IC of a database D. An integrity-preserving repair of (D, S) (which is total if $S = IC$) can be computed by each integrity-preserving update method, simply by running the update request $\sim \mathit{vio}_S$, where vio_S be defined by the clauses $\mathit{vio}_S \leftarrow B_i$ $(1 \leq i \leq n)$.

So far, we have said nothing about computing any measure that may be used in integrity-preserving update methods. In fact, computing measures $\iota, |\iota|, \zeta, |\zeta|$

corresponds to the cost of searching SLDNF trees rooted at constraint denials, which can be exceedingly costly. The same correspondence holds for computing κ and $|\kappa|$ in databases and integrity theories without negation in the body of clauses. If negation may occur, the cost can even be higher, as evidenced in [9].

Fortunately, none of these measures needs to be computed explicitly. Instead of computing $\nu(D^U, IC)$ and $\nu(D^U, IC)$ entirely, it suffices to compute a superset approximation of the increment $\delta(\nu(D, IC), \nu(D^U, IC))$, as many TIC methods do, for $\nu = \zeta$. As attested by such methods, approximating the increment of inconsistency in consecutive states is significantly less costly than checking the inconsistency of entire databases. Moreover, for two integrity-preserving partial repair candidates U, U' of IC in D that do not repair the same set of violations, U is preferable to U' if $\delta(\nu(D, IC), \nu(D^{U'}, IC)) \prec \delta(\nu(D, IC), \nu(D^U, IC))$, since U eliminates more inconsistency from D than U'.

5 Conclusion

In theory, database quality can be achieved by either preventing or eliminating the violation of integrity constraints. In practice, however, integrity violation cannot always be prevented, and a total elimination of all violations often is infeasible. Thus, integrity maintenance must be inconsistency-tolerant.

In this paper, we have generalized the concept of inconsistency-tolerant integrity checking and repairing in [12]. We have axiomatized measures that determine the amount of violation in given databases with associated integrity theories. Using such measures, each update can be checked and accepted if it does not increase the measured violation. Similarly, each repair is acceptable if it decreases the measured violation.

Future work includes an application of the concept of measure-based inconsistency tolerance for computing answers that have integrity in databases with violated constraints, and the use of measure-based ITIC for concurrent transactions in distributed and replicated databases.

References

1. Bauer, H.: Maß- und Integrationstheorie, 2nd edn. De Gruyter (1992)
2. Besnard, P., Hunter, A.: Quasi-Classical Logic: Non-trivializable Classical Reasoning from Inconsistent Information. In: Froidevaux, C., Kohlas, J. (eds.) ECSQARU 1995. LNCS, vol. 946, pp. 44–51. Springer, Heidelberg (1995)
3. Ceri, S., Cochrane, R., Widom, J.: Practical Applications of Triggers and Constraints: Success and Lingering Issues. In: Proc. 26th VLDB, pp. 254–262. Morgan Kaufmann (2000)
4. 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)
5. Christiansen, H., Martinenghi, D.: On simplification of database integrity constraints. Fundam. Inform. 71(4), 371–417 (2006)
6. Clark, K.: Negation as Failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 293–322. Plenum Press (1978)

7. Decker, H.: The Range Form of Databases and Queries or: How to Avoid Floundering. In: Proc. 5th ÖGAI. Informatik-Fachberichte, vol. 208, pp. 114–123. Springer (1989)
8. Decker, H.: Quantifying the Quality of Stored Data by Measuring their Integrity. In: Proc. DIWT 2009, Workshop SMM, pp. 823–828. IEEE (2009)
9. Decker, H.: Answers That Have Integrity. In: Schewe, K.-D., Thalheim, B. (eds.) SDKB 2010. LNCS, vol. 6834, pp. 54–72. Springer, Heidelberg (2011)
10. 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)
11. 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)
12. Decker, H., Martinenghi, D.: Inconsistency-tolerant Integrity Checking. TKDE 23(2), 218–234 (2011)
13. Grant, J., Hunter, A.: Measuring the Good and the Bad in Inconsistent Information. In: Proc. 22nd IJCAI, pp. 2632–2637 (2011)
14. Hunter, A.: Measuring Inconsistency in Knowledge via Quasi-Classical Models. In: Proc. 18th AAAI & 14th IAAI, pp. 68–73 (2002)
15. 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)
16. Ramakrishnan, R., Gehrke, J.: Database Management Systems. McGraw-Hill (2003)
17. Sadri, F., Kowalski, R.: A theorem-proving approach to database integrity. In: Foundations of Deductive Databases and Logic Programming, pp. 313–362. Morgan Kaufmann (1988)
18. ten Cate, B., Fontaine, G., Kolaitis, P.: On the Data Complexity of Consistent Query Answering. To appear in Proc. 15th ICDT. LNCS, Springer (2012)
19. Vardi, M.: On the integrity of databases with incomplete information. In: Proc. 5th PODS, pp. 252–266. ACM Press (1986)
20. Wijsen, J.: Database repairing using updates. ACM Trans. Database Syst. 30(3), 722–768 (2005)