Decker, H. (2013). Measure-Based Inconsistency-Tolerant Maintenance of Database Integrity. Lecture Notes in Computer Science. 7693:149-173. doi:10.1007/978-3-642-36008-4_7

Measure-based Inconsistency-tolerant Maintenance of Database Integrity

Hendrik Decker

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Spain

Note:
This is a revised version of a paper with the same title, published in a collection of revised papers that were selected from the 5th International Workshop on Semantics in Data and Knowledge Bases (SDKB), Zürich, Switzerland, July 3, 2011, pp. 149–173. The copyright owner of the original paper is Springer-Verlag Berlin Heidelberg. Consult that paper at the following reference:

Hendrik Decker:
Measure-based Inconsistency-tolerant Maintenance of Database Integrity.
In *Semantics in Data and Knowledge Bases* (*SDKB 2011*), 5th International Workshop, Revised Selected Papers. Proceedings edited by Klaus-Dieter Schewe and Bernhard Thalheim, published in Springer Lecture Notes in Computer Science, vol. 7693, 2013.

Measure-based Inconsistency-tolerant Maintenance of Database Integrity

Hendrik Decker *

Instituto Tecnológico de Informática, Valencia, Spain

Abstract. To maintain integrity, constraint violations should be prevented or repaired. However, it may not be feasible to avoid inconsistency, or to repair all violations at once. Based on an abstract concept of violation measures, updates and repairs can be checked for keeping inconsistency bounded, such that integrity violations are guaranteed to never get out of control. This measure-based approach goes beyond conventional methods that are not meant to be applied in the presence of inconsistency. It also generalizes recently introduced concepts of inconsistency-tolerant integrity maintenance.

1 Introduction

To some extent, the intended semantics of a database can be modeled by integrity constraints (in short, constraints). Such constraints are declared by formal sentences that express what should or should not hold in each state of the database. Semantic consistency, a.k.a. integrity, then corresponds to constraint satisfaction and inconsistency to constraint violation. Satisfaction means that each constraint is satisfied, i.e., evaluates to *true* in the given database state, and violation means that some constraint is violated, i.e., evaluates to *false*.

The problem studied in this paper is the *maintenance* of integrity. In particular, we focus on *checking* the preservation of integrity satisfaction across updates, and on *repairing* integrity violations. Checking means to prevent integrity violations that could be induced by updates, repairing means to update the database such that integrity violations are eliminated.

Solutions for integrity maintenance have been discussed in many research papers and state-of-the-art inventories (see [53] for a fairly recent survey). Rather than proposing new methods for integrity maintenance, we present generic formalizations of approaches to integrity checking and repairing that subsume most existing solutions. In particular, we generalize the formalization in [27], which has rebutted the theoretical point of view that inconsistency in databases is intolerable. In order to achieve conceptual genericity and inconsistency tolerance, we quantify the lack of integrity satisfaction by violation measures. They enable to monitor and reason about integrity in the presence of inconsistency.

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

In Section 2, we outline the foundations of the paper. In Section 3, we define a concept of violation measures for quantifying the amount of inconsistency in databases. In Section 4, we formalize a measure-based inconsistency-tolerant approach to integrity checking. In Section 5, we describe the use of violation measures and inconsistency-tolerant integrity checking for obtaining partial repairs that curtail constraint violations while tolerating extant inconsistencies. The ease with which the theorems in Sections 3 – 5 are obtained mainly is due to the strength of the abstractions in the definitions from which they follow. In Section 6, we address related work, including our own. In Section 7, we conclude.

2 Background and Framework

In 2.1, we outline a broad background of issues related to integrity maintenance, in order to facilitate the placement of this paper into the wide spectrum of work on database integrity. In 2.2, we formalize the framework of the remainder.

2.1 Background

Integrity constraints and their maintenance are of crucial importance, not only for the preservation of the semantic correctness of the data across state changes, but already for the design and the implementation of database schemas. It is indeed of utmost importance that, first of all, requirements engineering (RE) and conceptual modeling (CM) are done well; otherwise, a systematic maintenance of integrity may be a lost cause from the start. Similarly, a careful database schema design (SD) is indispensable for having a chance of effective integrity maintenance at all. Also schema evolution (SE) should be conscious of constraints to be maintained or changed, since they are meant to evolve consistently with the schema. Moreover, an integrity-aware design of database transactions (transaction design, TD) can help to prevent constraint violations.

A lot of work on RE and CM that is related to integrity constraints can be found in the literature by authors such as Borgida, Chen, Jarke, Mylopoulos, Olivé, Thalheim and many others. Plenty of material on SD can be found in all textbooks on the foundations of databases. In SD, constraints are often called 'dependencies'. Dependency theory is a subfield of SD that deals with the prevention of constraint violations, called 'update anomalies', by various normal forms of schemas. These normal forms are obtained by enforcing various kinds of dependencies. Authors such as Beeri, Fagin, Schewe, Vardi and many others have designated and pushed the limits of dependency theory. By comparison, SE and TD have received less attention, but interesting proposals have been made in [15] [6] and others, for SE, and [2] [57] and others, for TD.

In this paper, we shall not be concerned with RE, CM, SD, SE and TD. We assume that each database is an instance of a syntactically well-defined schema, to which an *integrity theory*, i.e., a finite set of declarative integrity constraints, is associated. Beyond the syntactic confinements given by the schema description language, we do not insist that schemas conform to any prescribed or desirable

normal form. Properties that serve to obtain a schema that complies with any semantic requirement are supposed to be expressed declaratively by suitable integrity constraints. A careful integrity-aware design of schema alterations and transactions is welcome, but not compulsory for the purpose of this paper.

In a setting as sketched in the preceding paragraph, integrity can be maintained in two complementary ways: by *checking* the preservation of constraint satisfaction upon updates (i.e., updates that would violate integrity are filtered out), or by *repairing* constraint violations.

We further assume that integrity checking is done by a software module that is independent of applications, transactions, triggers and stored procedures defined by the schema designer or the user. (This assumption does not exclude the implementation of integrity checking by triggers or stored procedures by the manufacturer of the DBMS or the provider of some application-independent middleware, rather than by the schema designer or the user.) The particular approach to integrity checking by such a module often is called a *method* (for integrity checking). A module that embodies a method can either be built into the core of the DBMS (as it is the case for checking standard constructs such as primary and foreign key constraints), or be situated on top of the DBMS, as part of some middleware that interfaces users and applications with the database. Similarly, repairing is supposed to be done methodically: a module that is independent of applications, transactions, schemas and users is supposed to generate update candidates that, when executed, would eliminate integrity violations.

2.2 The Formal Framework

In 2.2.1, we outline some basic preliminaries. In 2.2.2 and 2.2.3, we recapitulate the notions of 'cases' from [27], and 'causes' from [20], respectively. Cases are instances of constraints that are useful for three objectives: simplified integrity checking, quantifying constraint violations and tolerating inconsistency. Causes are stored data that are responsible for the violation of constraints, and are of similar use as cases. Unless specified otherwise, we use notations and terminology that are common for *datalog* [1, 30] and first-order predicate logic [32].

2.2.1 Databases, Completions, Updates, Constraints

Let us assume a universal language $\mathcal{L}$ for expressing the domain of discourse of each database. Let $\mathcal{H}_\mathcal{L}$ denote the Herbrand base of $\mathcal{L}$, and $\mathcal{L}^c$ the set of constant terms in $\mathcal{L}$, which we may represent, w.l.o.g, by natural numbers.

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 constant terms or variables. A *literal* is either of the form A or $\sim A$, where A is an atom; $\sim$ represents negation.

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 in databases.

A *database* is a finite set of database clauses. As usual, we assume that, for each database D, the set of predicates of facts in D and the set of predicates of the head of rules in D are disjoint.

The well-known *completion* of D be denoted by *comp*(D), which essentially consists of the if-and-only-if completions (in short, completions) of all predicates in $\mathcal{L}$. [14]. For a predicate p in $\mathcal{L}$, let p_D denote the completion of p in D.

Definition 1. Let D be a database, p a predicate in $\mathcal{L}$, n the arity of p, $x_1, \ldots, x_n$ the $\forall$-quantified variables in p_D and θ a substitution of $x_1, \ldots, x_n$. For $A = p(x_1, \ldots, x_n)\theta$, the *completion* of A in D is obtained by applying θ to p_D and is denoted by A_D. Further, let $\underline{comp}(D) = \{A_D \mid A \in \mathcal{H}_\mathcal{L}\}$, and *if*($D$) and *only-if*($D$) be obtained by replacing $\leftrightarrow$ in each $A_D \in \underline{comp}(D)$ by $\leftarrow$ and, resp., $\rightarrow$. Finally, let *iff*(D) = *if*(D) $\cup$ *only-if*(D). The usual equality axioms of *comp*(D) that interpret $=$ as identity 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 ',' also denotes conjunction in the body of rules and denials. Otherwise, conjunction is denoted by $\wedge$. 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 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 R is *true* in D^U. View updating is a well-known special kind of satisfying update requests. In Section 5, 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*, 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 [17], 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, 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* [17].

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).

2.2.2 Cases

For each constraint I, a *case* of I is an instance of I obtained by substituting the variables in I with terms in $\mathcal{L}$. This definition of cases is simpler than a more encompassing one in [27], where cases have been defined for constraints in a more general syntax. A *ground case* of I is a case of I obtained by a substitution of all variables in I with ground terms.

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 use of cases for simplified integrity checking is illustrated in Example 1.

Example 1. A constraint in a database D which requires that each person's *ID* be unique, by asserting that no two persons with the same identifier x may have different attributes y_1, y_2, is represented by $I = \leftarrow p(x, y_1),\ p(x, y_2),\ y_1 \neq y_2$. For the insertion of a record about a person, e.g., $p(1111, jill)$, typical methods for simplified integrity checking do not evaluate I in its full generality, but just the relevant case $\leftarrow p(1111, jill),\ p(1111, y_2),\ jill \neq y_2$. Actually, also the case $\leftarrow p(1111, y_1),\ p(1111, jill),\ y_1 \neq jill$ is relevant, but it is logically equivalent to the previous one and thus can be ignored.

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 Sections 4 and 5.

2.2.3 Causes

As in [20], 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. Causes generalize the notion of 'resource set' in [52]. In Section 3, causes are used for measuring inconsistency, and in Sections 4 and 5 for measure-based inconsistency-tolerant integrity maintenance.

Definition 2. Let D be a database and $I = \leftarrow B$ an integrity constraint such that $D(\exists B) = true$. 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.

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 2.

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 $D = \{p(x) \leftarrow q(1, x);\ q(2, y) \leftarrow r(y);\ r(1)\}$. The only cause of $\sim p(2)$ in D is $\{p(2) \rightarrow q(1, 2);\ \sim q(1, 2)\}$.

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

e) 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 $\{p \rightarrow \exists x\, q(x, x)\} \cup \{q(i, i) \rightarrow r(i) \wedge s(i)) \mid i \in \mathcal{L}^c\} \cup \{\sim r(2);\ \sim s(1)\}$ and, for each $j > 2$ in $\mathcal{L}^c$, either $\sim r(j)$ or $\sim s(j)$, and nothing else.

f) Let $D = \{p \leftarrow \sim q;\ q \leftarrow \sim r;\ q \leftarrow \sim s\}$. The two causes of $\sim p$ in D are $\{q \leftarrow \sim r;\ p \rightarrow \sim q;\ \sim r\}$ and $\{q \leftarrow \sim s;\ p \rightarrow \sim q;\ \sim s\}$.

g) Let $D = \{p \leftarrow q;\ p \leftarrow \sim q\}$, $D' = \{p \leftarrow q;\ p \leftarrow \sim q;\ q\}$ and $I = \leftarrow p$. Clearly, D is a cause of the violation of I in D and in D'. Another cause of p in D is $\{p \leftarrow \sim q;\ \sim q\}$. Another cause of p in D' is $\{p \leftarrow q;\ q\}$.

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

i) 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.

Note that causes are not compositional, as shown by Example 2*i*, 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. However, it can be shown that E is a cause of the violation 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 $E' \subsetneq E$.

The following definition of *vioCau* is analogous to the definition of *vioCas* in 2.2.2. While *vioCas* pinpoints inconsistency by focusing on violated cases, *vioCau* as defined below localizes inconsistency by focusing on the data that cause integrity violation.

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

3 Violation Measures

Violation measures are a special kind of inconsistency measures [41]. Violation measures are geared to gauge the amount of integrity violation in databases, e.g., by sizing cases or causes of constraint violations. In 3.1, we conceptualize our approach to violation measures. In 3.2, we define this concept formally and give several examples. In 3.4, we discuss the desirability of some properties that are commonly associated to measures. In Sections 4 and 5, violation measures are used for characterizing inconsistency-tolerant integrity maintenance.

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 smallest element 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.

Thus, it can be checked if an update U does not increase the amount of inconsistency, or at least if U does not trespass a certain threshold of inconsistency or if the increase of inconsistency brought about by U is negligible. In any case, extant inconsistency is tolerated.

In classical measure theory [7], 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 *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 $\mathbb{M}$, 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 has a partial order $\preccurlyeq$ that is reflexive, antisymmetric and transitive, and an addition with neutral element o that is, at a time, the smallest element of $\preccurlyeq$. The purpose of $\nu(D, IC)$ is to size the amount of inconsistency in (D, IC).

3.2 Formalizing Violation Measures

Definitions 3 and 4 below specialize the classical concepts of metric spaces and measures [7], for databases and integrity violations. Yet, in a sense, these definitions also generalize the traditional concepts, since they allow both numerical and non-numerical quantifications and comparisons of measured items. 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 $m \prec m'$ denote that $m \preccurlyeq m'$ 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.

Now, we define measures with metric spaces such as those in Example 3.

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

In the following subsection, we are going to give examples of violation measures with metric spaces such as those in Example 3.

3.3 Examples of Violation Measures

Example 4. A coarse violation measure β is defined by $\beta(D, IC) = D(IC)$. Its range is 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 $v, v' \in \{true, false\}$. 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. We are going to meet β again in 4.1.

Example 5. The measures ι and $|\iota|$ are characterized by comparing and, resp., counting the set of violated constraints in the database. They are defined 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 the case-based measures in Example 5, also cause-based measures can be defined, by the equations $\kappa(D, IC) = vioCau(IC, D)$ and $|\kappa|(D, IC) = |\kappa(D, IC)|$, with the metric spaces $(2^{iff(D)}, \subseteq, \ominus, \cup, \emptyset)$ and, resp., again $(\mathbb{N}_0^+, \leq, |-|, +, 0)$. Specific differences between case- and cause-based measures are addressed in [21].

Other measures are discussed in [26], among them two variants of an inconsistency measure in [40], based on quasi-classical models [8]. Essentially, both 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 $(2^{\mathcal{H}^*_{\mathcal{L}}}, \subseteq, \ominus, \cup, \emptyset)$ where $\mathcal{H}^*_{\mathcal{L}} = \mathcal{H}_{\mathcal{L}} \cup \{\sim A \,|\, A \in \mathcal{H}_{\mathcal{L}}\}$, and, resp., $(\mathbb{N}_0^+, \leq, |-|, +, 0)$.

Some more violation measures are going to be identified in 3.4.1 and 4.2.

3.4 Properties of Violation Measures

Note that, as opposed to classical measure theory and previous work on inconsistency measures (to be addressed in Section 6), Definition 4 does not require any axiomatic property of measures, such as definiteness, additivity or monotonicity. These usually are required for each classical measure μ, as already mentioned in 3.1. We are going to look at such properties, and argue that definiteness is not cogent, and both additivity and monotonicity do not hold in many databases.

In 3.4.1, we discuss the standard axiom of definiteness of measures, including some weakenings thereof. In 3.4.2, we show that the standard axiom of additivity of measures is invalid for violation measures. In 3.4.3, we dismiss the standard axiom of monotonicity of measures for violation measures in databases with non-monotonic negation, and propose some valuable variants.

3.4.1 Definiteness

For classical measures μ, 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

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

for each pair (D, IC).

A property corresponding to (12) is postulated for inconsistency measures in [43] [35] (in [43], (12) is called 'consistency'). 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 $\beta,\ \iota,\ |\iota|,\ \zeta,\ |\zeta|,\ \kappa,\ |\kappa|$ in 3.3 fulfills (12).

So, 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, 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 [61] [59].

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 [59], 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 some databases without integrity constraints. That value can then be interpreted as a warning that there is a non-negligible likelihood of inconsistency, although no constraints have been imposed, be it out of neglect, or for trading off consistency for performance, or for any other reason.

So, in the end, only the rather bland property $\nu(\emptyset,\emptyset)=0$ remains as a weakening of (12) that should be 'de rigueur' for violation measures.

3.4.2 Additivity

For classical 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 classical 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, $D(IC)=\mathit{true}$ and $D'(IC')=\mathit{true}$, thus $|\zeta|(D,IC)+|\zeta|(D',IC')=0$, but $|\zeta|(D\cup D',\ IC\cup IC')=1$.

Yet, it can be shown that (15) holds for each of the measures β, ι, $|\iota|$, ζ, $|\zeta|$, κ, $|\kappa|$ in 3.3 if (D,IC) and (D',IC') do not share any predicate.

3.4.3 Monotonicity

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

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

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

A property corresponding to (16) is postulated for inconsistency measures in [43] [35]. 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 definite databases D,D' and definite integrity theories IC,IC', each of the measures β, ι, $|\iota|$, ζ, $|\zeta|$, κ, $|\kappa|$ in 3.3 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 $|\zeta|(D', IC') = 0$.

A variant of (16), with same conclusion but different premise, that holds also for non-definite databases and integrity theories, requires that the measured amount of inconsistency in databases that violate integrity is never lower than the measured inconsistency in databases that satisfy integrity. Formally, for each pair of pairs (D, IC), (D', IC'),

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

is asked to hold. It is easy to show the following result.

Theorem 3. Each of the measures $\beta,\ \iota,\ |\iota|,\ \zeta,\ |\zeta|,\ \kappa,\ |\kappa|$ in 3.3 fulfills (17).

A property that is slightly stronger than (17) has been postulated in [26]. It is obtained by replacing $\preccurlyeq$ in (17) by $\prec$. It also holds for all measures in 3.3. Yet, similar to (12), it does not hold for measures ζ' and σ, as defined in 3.4.1, while (17) does hold for those measures.

The following weakening of (16) has been postulated in [22]. It requires that, for each D, the values of ν grow monotonically with growing integrity theories.

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

It is easy to show the following result.

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

Interestingly, (18) may not hold for measures that calculate the ratio of conflicting and conflict-free atoms in (D, IC), such as the measure in [40], as mentioned in 3.3, since an increase of (D, IC) by non-conflicting atoms, i.e., by consistent knowledge, decreases the ratio of inconsistency.

4 Integrity Checking

Due to a possibly complex quantification of constraints, integrity checking tends to be unbearably expensive, unless some simplification method is used [13]. Simplification theory traditionally requires *total integrity*, i.e., that, for each update U, the state D to be updated by U must satisfy all constraints. Then, integrity checking can focus on those cases of constraints that are relevant, i.e., possibly affected by the update, and ignore all others, since they are going to remain satisfied in the state D^U, reached by the update.

Example 10. Suppose that, in Example 1, there is no other constraint with an occurrence of p as the predicate of some non-negated literal, nor with an occurrence of a predicate the definition of which recurs on p. Then, it suffices to evaluate the simplification $\leftarrow p(1111, y_2, z_2),\ \mathit{jill} \neq y_2$ of the relevant case $\leftarrow p(1111, \mathit{jill}),\ p(1111, y_2),\ \mathit{jill} \neq y_2$, obtained from I by dropping the conjunct $p(1111, \mathit{jill})$, which is known to be *true* in D^U. Each other case of I and each other constraint without the mentioned occurrences can be ignored.

Thus, if integrity is totally satisfied in D, and all relevant constraints remain satisfied when U is committed, then D^U also satisfies integrity totally.

Often, however, total integrity is nothing but wishful thinking: the accumulation of integrity violations in databases is commonplace, since consistency is not always taken care of sufficiently. That may be due to many different possible reasons. Some typical ones are: plain neglect (e.g., integrity checking had been switched off for bulk updates or reloading a backup, but not switched on again afterwards), or efficiency considerations (e.g., integrity maintenance is skipped in favour of performance), or the heterogenity of data or schemas to be integrated (e.g., during the ETL process of data warehousing, or for federating hitherto disparate databases), or architectural impediments (e.g., poor integrity support in distributed databases), or other circumstances (e.g., altered constraints are not checked against legacy data, or locally consistent data fail to comply with global constraints in distributed databases, etc).

Since a total avoidance of inconsistency often is impractical or unfeasible, an inconsistency-tolerant approach to integrity maintenance is needed. As we are going to see, that can be achieved by using violation measures. In fact, even in the presence of persisting inconsistency, the use of such measures can prevent the increase of inconsistency across updates. Moreover, violations measures allow to control that the amount of inconsistency never exceeds given thresholds.

In 4.1, we define and illustrate measure-based inconsistency-tolerant integrity checking. In 4.2, we show how inconsistency can be confined by assigning weights to violated cases of constraints, which goes beyond the measures seen so far. In 4.2, we also show how to generalize measure-based inconsistency-tolerant integrity checking by allowing for certain increases of inconsistency that are bounded by some thresholds.

4.1 Measure-based Inconsistency-tolerant Integrity Checking

To motivate measure-based ITIC, let us look again at Example 10. As we have seen there, only a single case is evaluated for checking U, no matter if other cases of the same or of other constraints are violated in D or not. Hence, that check tolerates any extant integrity violation. It also guarantees that all consistent parts of the database remain consistent, i.e., that U does not increase the set of violated cases of I, nor induces any other violation in D^U. It also guarantees that U does not introduce any new cause of integrity violation. Thus, that check behaves as if it used any of the measures ι, $|\iota|$, ζ, $|\zeta|$, κ or $|\kappa|$.

Definition 5, below, subsumes each method with such a behaviour, i.e., 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 ν, the range of which is structured by a partial order $\preccurlyeq$, 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. Definition 5 also generalizes ITIC as defined in [27], since the latter is equivalent to Definition 5 for $\nu = \zeta$.

In [27], we have shown that the total integrity requirement is dispensable for most TIC approaches. Similar to corresponding proofs in [25, 27], it can be shown that not all, but most TIC methods, including built-in integrity checks in common DBMSs, are ν-based, for each $\nu \in \{\iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|\}$. Moreover, the following results are easily shown by applying the definitions.

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

4.2 Weighted ITIC and Thresholds

Example 11, below, illustrates how the measures $|\iota|$ and $|\zeta|$ that count violated constraints or cases 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 is known from deontic logic, where 'soft' constraints that *ought to* be satisfied are distinguished from 'hard' constraints that *must* be satisfied [55].

Example 11. Let *mr*, *lr* and *hr* be predicates that model a minor, a low and, resp., a high risk. Further, $I_1 = \leftarrow mr(x)$, $I_2 = \leftarrow lr(x)$, $I_3 = \leftarrow hr(x)$ be two soft and one hard constraint, for protecting against *m*inor, *l*ow and, resp., *h*igh *r*isks, where *mr*, *lr* and *hr* are defined by the clauses $mr(x) \leftarrow p(x,y)$, $x = 3$, $lr(x) \leftarrow p(y,z)$, $x = y + z$, $x > th$, $z \geq y$ 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 D. For each $\nu \in \{\iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|\}$, no ν-based method would accept the update $U = \{delete\ p(8,3),\ insert\ p(3,8)\}$, although the high risk provoked by $p(8,3)$ is diminished to a minor and a low risk produced by $p(3,8)$. However, measures that assign suitable weights to the cases of I_1, I_2 and I_3 can avoid that problem. For instance, consider the measure ω that counts the numbers n_i of violated cases of I_i $(i = 1,2,3)$, and assigns $n_1 + n_2 + fn_3$ to $(D, \{I_1, I_2, I_3\})$, where f is a weighting factor such that $f \geq 3$. Clearly, $\omega(D^U, \{I_1, I_2, I_3\}) < \omega(D, \{I_1, I_2, I_3\}$, hence each ω-based method accepts U.

Instead of modeling thresholds in constraints, as in Example 11, it is also possible to include thresholds in measures. For instance, let ν be a measure the range of which is structured by a distance δ, and methods $\mathcal{M}$ be defined by replacing the consequent of (19) and the antecedent of (20) in Definition 5 by

$$\nu(D^U, IC) \preccurlyeq \nu(D, IC) \quad or \quad (\delta((D, IC), (D^U, IC)) \preccurlyeq th \quad and \quad \nu(D^U, IC) \preccurlyeq th')$$

where th, th' are thresholds (that, in general, may be parametrizable terms). Clearly, th limits the increment of inconsistency that may be induced by any update, while th' is an absolute upper bound of permissible inconsistency. Note that, if $\mathcal{M}$ would not check that th' is not trespassed, then inconsistency may accumulate over time beyond tolerability, by repeated increments of inconsistency, each of which does not exceed th but which may eventually surpass th'.

5 Repairs

Roughly, repairing a database means to compute and execute an update in order to eliminate integrity violation. The latter either is already manifest in the database, or it would come into existence if some update would be committed.

For instance, if a constraint I (or some case C of I) is already violated in a database D, then a repair, i.e., an update U is called for such that I (or C) is no longer violated in D^U. Else, if I would become violated by committing some update U_R, the purpose of which is to satisfy an update request R, then an update U of D is called for such that U satisfies R and neither induces any violation of I that would be caused by U_R, nor any other violation in D^U that did not exist in D. Thus, U can be seen as a repair of D^{U_R}.

Hence, each repair can be identified with some update that either eliminates an extant integrity violation or satisfies an update request while preserving integrity. In the literature, the updated database itself is often also called a 'repair'.

In 5.1, we distinguish between partial and total repairs, as well as between repairs that do or do not preserve integrity. In 5.2, we recapitulate the concept of integrity-preserving update methods. In 5.3, we outline how such methods use ITIC for computing total and partial integrity-preserving repairs.

5.1 Partial Repairs that Preserve Integrity

In general, repairing is complex [12], and can be too costly or even unfeasible, e.g., if inconsistencies are hidden or unknown. Yet, it may still be possible to curtail inconsistency by not repairing *all*, but only *some* violations.

The definition below distinguishes between total repairs, which eliminate all inconsistencies, and partial repairs, which repair only a fragment of the database. Obviously, partial repairs tolerate inconsistency, since some constraints may remain violated.

Definition 6. (*Repair*) [27]
Let D be a database, IC an integrity theory and S a subset of $Cas(IC)$ such that $D(S) = false$. An update U is called a *repair* of (D, S) if $D^U(S) = true$. If $D^U(IC) = false$, U is also called a *partial repair* of (D, IC). Otherwise, if $D^U(IC) = true$, U is called a *total repair* of (D, IC). For a measure ν, we say that U *preserves integrity w.r.t.* ν if $\nu(D^U, IC) \preccurlyeq \nu(D, IC)$.

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 [12] or can be imagined (see also related remarks in Section 6). Note that Definition 6 does not involve any notion of minimality. However, Example 12 features subset-minimal repairs.

Example 12. 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 $\leftarrow q(c,c)$. There are exactly two minimal total repairs of IC in D, viz. $\{delete\, q(c,c);\ \ delete\, p(b,b,c);\ \ delete\, p(c,b,c)\}$ and $\{delete\, q(c,c);\ insert\, q(b,c);\ delete\, p(c,b,c)\}$. Each of $U_1 = \{delete\, p(b,b,c)\}$ and $U_2 = \{insert\, q(b,c)\}$ is a minimal repair of $\{\leftarrow p(b,b,c), \sim q(b,c)\}$ in D and a partial repair of IC in D. Both tolerate the persistence of the violation of $\leftarrow q(c,c)$. Similarly, $U_3 = \{delete\, q(c,c)\}$ is a minimal repair of $\{\leftarrow q(c,c)\}$ in D and a partial repair of IC, which tolerates the violation of $\leftarrow p(b,b,c), \sim q(b,c)$.

W.r.t. each $\nu \in \{\iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|\}$, each total repair trivially preserves integrity, (e.g., those in Example 12), since no violations remain after total repairs. Unfortunately, however, partial repairs may not preserve integrity w.r.t. any $\nu \in \{\iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|\}$, i.e., they may induce the violation of some constraint that is not in the repaired set, as illustrated by the following example.

Example 13. Consider again D and IC in Example 12. As opposed to U_1 and U_2, U_3 induces the violation of a case in the updated state that is satisfied before the update. That case is $\leftarrow p(c,b,c), \sim q(c,c)$. It is satisfied in D but not in D^{U_3}. Thus, the non-minimal 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 a violation that did not exist before the partial repair. For each $\nu \in \{\iota, |\iota|, \zeta, |\zeta|, \kappa, |\kappa|\}$ and each $i \in \{1, 2, 4\}$, U_i clearly preserves integrity, since all cases in $\mathsf{SatCas}(D, IC)$ remain satisfied in D^{U_i} and no new cause of the violation of IC in D is induced by U_i. Note that U_4 is a minimal integrity-preserving repair of $\{\leftarrow q(x,x)\}$, but not a mere minimal repair of $\{\leftarrow q(x,x)\}$, since the minimal repair U_3 of $\{\leftarrow q(x,x)\}$ is a proper subset of U_4. However, U_4 is preferable to U_3 since U_4 preserves integrity, while U_3 does not, as seen above.

The enlargement of U_3 to U_4, i.e., deleting also $p(c, b, c)$, fortunately does not induce any similar side effect as produced by deleting $q(c, c)$ alone. In general, stepwise repairs such as the one from U_3 to U_4 may possibly continue indefinitely, since each iteration may cause some other violation(s). The termination of such iterations is unpredictable, in general, as known from repairing by triggers [10]. However, side effects of updates can be avoided by checking if a given repair preserves integrity, with any convenient measure-based method, as expressed in the following result, which is an immediate consequence of Definitions 5 and 6.

Theorem 6. For each triple (D, IC, U), each measure ν and each ν-based method $\mathcal{M}$, U preserves integrity w.r.t. ν if $\mathcal{M}(D, IC, U) = ok$.

In general, the only-if version of Theorem 6 does not hold. A counter-example is provided by each method that is incomplete for measure-based integrity checking, in the sense of Definition 5 (e.g., the methods in [49, 59] have been shown to be incomplete for ζ-based integrity checking in [27]). However, it is easy to see that the only-if version of Theorem 6 does hold for methods that are complete for measure-based integrity checking. For instance, the well-known method in [56] is complete for ζ-based integrity checking, as shown in [27]).

Thus, Theorem 6 guarantees that, for each partial repair U, each measure-based method can be used to check if U preserves integrity, and each complete ν-based method is a procedure for deciding if U preserves integrity or not.

5.2 Integrity-preserving Update Methods

Update methods are algorithms that take as input an update request and compute candidate updates for satisfying the request as their output. Such a method is said to be *integrity-preserving* if each of its computed updates preserves integrity. Integrity-preserving update methods can be used to compute partial repairs that are integrity-preserving w.r.t. any measure ν, as shown in [27] for the special case of $\nu = \zeta$. Theorem 7 below generalizes that result.

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 $D^{U_i}(R) = true$ $(1 \leq i \leq n)$.

Note that an update method as defined above is impartial with regard to possible integrity violation that may be induced by any of the U_i. As opposed to that, Definition 8, below, takes such undesirable side effects into account.

To avoid that updates induce new integrity violations, many update methods in the literature (e.g., [18, 37, 44]) postulate the total satisfaction of all constraints in the state before the update, in analogy to the total integrity premise of traditional integrity checking, as mentioned in 4. However, for the class of update methods defined below, that postulate is as superfluous for satisfying update requests as it has been for integrity checking.

Definition 8. (*Integrity-preserving Update Method*)
Let ν be a measure. An update method $\mathcal{UM}$ is *integrity-preserving w.r.t.* ν if each update computed by $\mathcal{UM}$ preserves integrity w.r.t. ν.

For an update request R and a database D, several integrity-preserving 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 integrity checking 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 7, below, follows from Definition 8 and Theorem 6.

Theorem 7. For each measure ν, each update method that uses ν-based ITIC to check its computed candidate updates is integrity-preserving w.r.t. ν.

Theorem 7 serves to identify several known update methods as integrity-preserving, since they use inconsistency-tolerant integrity checking. Among them are the update methods described in [18] and [37, 38]. Several other known update methods are abductive e.g., [44, 45, 29]. They interleave the two phases as addressed above. Most of them are also integrity-preserving, as has been shown in [27] for the update method in [44].

The triviality of Theorem 7 should not be depreciated. Example 14 shows what can go wrong if an update method that is not integrity-preserving is used.

Example 14.
Let $D = \{q(x) \leftarrow r(x), s(x);\quad p(a,a)\}$, $IC = \{\leftarrow p(x,x);\quad \leftarrow p(a,y),\ q(y)\}$ and R the view update request to insert $q(a)$. 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 $\leftarrow p(a,y),\ q(y)$, integrity checking methods that are not inconsistency-tolerant (e.g., those in [39, 46]) may be mislead to 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 inconsistency-tolerant method rejects U and computes the update $U' = U \cup \{\mathit{delete}\, p(a,a)\}$ for satisfying R. Clearly, U' preserves integrity. Note that, incidentally, U' even removes the violated case $\leftarrow p(a,a)$.

The reduction of inconsistency as observed in Example 14 is not accidental. In fact, as long as *ITIC* is applied for each update, the number of violated cases is not only prevented from increasing, but also is likely to decrease over time, since each update, be it accidentally or on purpose, may repair part or all of the extant inconsistencies. An extended study of this feature is reported in [27].

5.3 Computing Integrity-preserving Repairs

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

Example 15. 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. A repair of (D, S) (which is total if $S = IC$) can be computed by each update method, simply by running the update request $\sim vio_S$, where vio_S be defined by the clauses $vio_S \leftarrow B_i$ $(1 \leq i \leq n)$.

Now, we recall from Section 5 that partial repairs may not preserve integrity. That problem is solved by the following corollary of Theorems 6 and 7. It says that the integrity preservation of partial repairs can be checked by measure-based ITIC (part *a*), and that integrity-preserving repairs can be computed by integrity-preserving update methods (part *b*).

Corollary

a) For each tuple (D, IC), each partial repair U of IC in D, each measure ν and each ν-based method $\mathcal{M}$ such that $\mathcal{M}(D, IC, U) = ok$, U preserves integrity w.r.t. ν.

b) For each measure ν and each partial repair U computed by an integrity-preserving update method that uses a ν-based integrity checking method, U preserves integrity w.r.t. ν.

So far, we have said nothing about computing any measure. 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 by a study of computing causes in [20].

However, violation measures may not need to be computed explicitly. For instance, instead of computing $\nu(D, IC)$ and $\nu(D^U, IC)$ entirely, it may suffice to compute an approximation of the difference $\delta(\nu(D, IC), \nu(D^U, IC))$, as many TIC methods do, for $\nu = \zeta$. As attested by such methods, checking an approximation of 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, 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'.

6 Related Work

To a large extent, this paper is a synopsis of previous work in [27, 22, 23, 26]. In [27], the emphasis is on ITIC, but without generalizing it to measure-based integrity maintenance. That generalization is done, to some extent, in [26], and further abstracted in [22], but only for ITIC, not for repairing. Measure-based repairing is the theme of [23]. Causes, as recapitulated in 2.2.3, are the basis of several measures addressed in that paper. Originally, they had been developed in [20], for computing answers that have integrity. That topic is different from integrity maintenance, but related. We shall come back on the relationship of query answering and integrity maintenance toward the end of this section.

Some of the related work of other authors has already been addressed in previous sections. There is plenty more work on inconsistency measures (sometimes also called 'measures of contradiction', 'quality metrics', 'coherence metrics', etc), integrity checking and repairing in the literature.

A non-comprehensive survey of measuring inconsistency is presented in [41]. Interesting work not yet cited nor addressed in [41] includes [54] [42] [60] [51] [48]. The main differences between this paper and previous work on inconsistency measures are, firstly, that the latter use non-standard logics such as paraconsistent, multivalued, annotated, probabilistic or possibilistic calculi, while our work exclusively relies on standard datalog and a theory of measures based on standard mathematics. Secondly, violation measures are applicable also in non-monotonic databases, where consistency is not compact [16], whereas, to the best of this author's knowledge, other inconsistency measures in the literature do not deal with that.

For instance, a frequently adopted approach to deal with inconsistent sets of data, including their measurement, is to distinguish maximally consistent or minimally inconsistent subsets, as done, e.g., in [47] [50] [42]. Unfortunately, this approach is bound to fail in non-monotonic databases, since, for instance, subsets (D', IC) of consistent pairs (D, IC) can be inconsistent (e.g., $D' = \{p\}$, $D = D \cup \{q\}$, $IC = \{\leftarrow p, \sim q\}$). A conservative way out of that could be to consider subsets of *iff*(D) that are maximally consistent or minimally inconsistent with IC, which remains to be investigated further.

Previous work on ITIC has culminated in [27]. Apart from our own previous work on measure-based ITIC [25, 26, 22], this author could not find anything quite similar in the literature.

No survey seems to exist yet for repairing. However, [10] may be consulted for repairs that satisfy update requests, and [62] for repairing manifest constraint violations. Repair-like cleaning of inconsistencies for data warehousing and schema matching is surveyed in [58].

As already mentioned in 5.1, repairs commonly are required to be minimal, i.e., the existence of some sort of measure used for deciding minimality is assumed. For instance, a cost model that, in essence, measures the affordability of repairs is proposed in [9]. A distance-based repairing is studied in [5]. However, inconsistency measures are usually not considered for repairing.

Conventionally, concepts of repair in the literature (e.g., [4], [36], [33]) only deal with total repairs. To the best of the author's knowledge, partial repairs have never been addressed elsewhere, except in [27, 23]. In [34], null values and a 3-valued database semantics are used to "summarize" total repairs. However, there is no notion of integrity-preserving updates for partial repairs by other authors, since integrity preservation is a trivial issue for total repairs.

In [31], several shortcomings of integrity maintenance are identified, and in particular the need of inconsistency tolerance. As a solution, facilities for explaining violations to the user who may intervene in the repair process are proposed, however without a systematic treatment of inconsistency tolerance. A concept of explanations based on causes is proposed in [24].

The application of our definitions and results is not compromised by limitations imposed by the syntax of integrity constraints, while various syntactical restrictions of constraints are typical in the literature on integrity maintenance.

There are two fields related to integrity maintenance that have not been mentioned in 2.1. One is the use of integrity constraints for query answering, in particular for *semantic query optimization* (abbr. SQO) [11] and *consistent query answering* (abbr. CQA) [4]. *ITIC* for SQO has been studied in [19].

CQA defines an answer to be consistent in (D, IC) if it is *true* in each minimal repair of IC in D. CQA depends on the chosen notion of minimality, of which Definition 5 is steered clear. In an experimental study [27], CQA and standard query answering in databases maintained by ζ-based ITIC have been compared, with favourable results for ITIC. CQA usually is not implemented by computing each repair, but by techniques of SQO or disjunctive logic programming. It should be interesting to revise CQA in terms of partial instead of total repairs, since, in general, not all violated constraints are relevant with regard to the given query.

The other remaining field related to integrity maintenance is *repair checking*, i.e., algorithms for deciding if a given update is a repair or not. Analogous to similar definitions in [12, 3], the problem of *inconsistency-tolerant repair checking* can be defined as the check if a given update is an integrity-preserving repair. Thus, part a of the Corollay in 5.3 entails that each measure-based integrity checking method implements inconsistency-tolerant repair checking.

7 Conclusion

In theory, an automated maintenance of declaratively stated integrity constraints can be achieved by either preventing their violation, i.e., by checking updates, or eliminating their violation, i.e., by repairing the database. In practice, however, integrity violation cannot always be prevented, and a total elimination of all violations often is not achievable. Thus, integrity maintenance must be inconsistency-tolerant. For the prevention of constraint violations, inconsistency tolerance means that integrity checking needs to wave the total integrity requirement, which insists that each committed database state satisfies all constraints without exception. Similarly, for the elimination of constraint violations, inconsistency tolerance means that repairs may be partial instead of total.

In this paper, we have generalized the concept of inconsistency-tolerant integrity checking and repairing in [27]. 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.

Ongoing work includes an application of the concept of measure-based inconsistency tolerance for computing answers that have integrity in databases with violated constraints, i.e., a generalization of [20], and the use of measure-based ITIC for concurrent transactions in distributed and replicated databases, i.e., an amplification of [28].

Acknowledgement

I'd like to acknowledge earlier collaborations and discussions with Davide Martinenghi, which formed the basis of elaborating the material presented in this paper. Also, I'd like to acknowledge helpful comments provided by the reviewers.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, 1995.
2. Abiteboul, S., Vianu, V.: A transaction-based approach to relational database specification. JACM 36(4):758-789, 1989.
3. Afrati, F., Kolaitis, P.: Repair checking in inconsistent databases: algorithms and complexity. In *12th ICDT*, ACM Press, 2009, pp. 31-41.
4. Arenas, M., Bertossi, L., Chomicki, J.: Consistent query answers in inconsistent databases. In *PODS'99*. ACM Press, 1999, pp. 68-79.
5. Arieli, O., Denecker, M., Bruynooghe, M.: Distance semantics for database repair. Ann. Math. Artif. Intell. 50:389415, Springer, 2007.
6. Arni-Bloch, N., Ralyté, J., Léonard, M.: Service-driven Information Systems Evolution: Handling Integrity Constraints Consistency. In A. Persson, J. Stirna (Eds.): PoEM 2009, Springer LNBIP vol. 39, pp. 191-206, 2009.
7. Bauer, H.: Maß- und Integrationstheorie, 2. Auflage. De Gruyter, 1992.
8. Besnard, Ph., Hunter, A.: Quasi-classical logic: Non-trivializable classical reasoning from inconsistent information. Symbolic and Quantitative Approaches to Uncertainty, Springer LNCS vol. 946, pp. 44-51, 1995.
9. Bohanon, P., Fan, W., Flaster, M., Rastogi, R.: A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. Proc. SIGMOD 2005, pp. 143-154. ACM Press, 2005.
10. 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.
11. Chakravarthy, U., Grant, J., Minker, J.: Logic-based Approach to Semantic Query Optimization. Transactions on Database Systems 15(2):162-207. ACM Press, 1990.
12. Chomicki, J.: Consistent query answering: Five easy pieces. *Proc. 11th ICDT*, pp. 1-17. Springer LNCS vol. 4353, 2007.
13. Christiansen, H., Martinenghi, D.: On simplification of database integrity constraints. *Fundamenta Informaticae* 71(4):371-417, 2006.
14. Clark, K.: Negation as Failure. In Gallaire, H., Minker, J. (Eds.): *Logic and Data Bases*, pp. 293-322. Plenum Press, 1978.
15. Curino, C., Moon, H., Deutsch, A., Zaniolo, C.: Update Rewriting and Integrity Constraint Maintenance in a Schema Evolution Support System: PRISM++. PVLDB, vol. 4, pp. 117-128, 2010.
16. Dawson, J.: The compactness of first-order logic: From Gödel to Lindström. History and Philosophy of Logic 14(1):15-37. Taylor & Francis, 1993.
17. Decker, H.: The Range Form of Databases and Queries or: How to Avoid Floundering. Proc. 5th ÖGAI, pp. 114-123. Informatik-Fachberichte vol. 208, Springer, 1989.
18. Decker, H.; Drawing Updates From Derivations. *Proc. 3rd ICDT*, pp. 437-451. Springer LNCS vol. 470, 1990.

19. Decker, H.: Extending Inconsistency-Tolerant Integrity Checking by Semantic Query Optimization. Proc. 19th DEXA, pp 89-96. Springer LNCS vol. 5181, 2008.
20. Decker, H.: Answers that have integrity. In Klaus-Dieter Schewe, Bernhard Thalheim (Eds.): Semantics in Data and Knowledge Bases - 4th International Workshop SDKB, pp. 54-72. Springer LNCS vol. 6834, 2011.
21. Decker, H.: Causes of the Violation of Integrity Constraints for Supporting the Quality of Databases. Proc. 12th ICCSA, pp. 283–292. Springer LNCS vol. 6786, 2011.
22. Decker, H.: Inconsistency-tolerant Integrity Checking based on Inconsistency Metrics. Proc. KES, Part II, pp. 548-558, Springer LNCS vol. 6882, 2011.
23. Decker, H.: Partial Repairs that Preserve Inconsistency. Proc. 15th ADBIS, pp. 389-400. Springer LNCS vol. 6909, 2011.
24. Decker, H.: Consistent Explanations of Answers to Queries in Inconsistent Knowledge Bases. In Roth-Berghofer, T., Tintarev, N. and Leake, D. (Eds.), Explanation-aware Computing, Proc. IJCAI-11 Workshop ExaCt 2011, pp. 71-80. Available at http://exact2011.workshop.hm/index.php.
25. Decker, H., Martinenghi, D.: Classifying integrity checking methods with regard to inconsistency tolerance. *Proc. PPDP'08*, pp. 195-204. ACM Press, 2008.
26. Decker, H., Martinenghi, D.: Modeling, Measuring and Monitoring the Quality of Information. *Proc. 28th ER Workshops*, pp. 212-221. Springer LNCS vol. 5833, 2009.
27. Decker, H., Martinenghi, D.: Inconsistency-tolerant Integrity Checking. IEEE TKDE 23(2):218-234, 2011.
28. Decker, H., Muñoz-Escoí, F. D.: Revisiting and Improving a Result on Integrity Preservation by Concurrent Transactions. *Proc. OTM Workshops 2010*, 297-306. Springer LNCS vol. 6428, 2010.
29. Dung, P., Kowalski, R., Toni, F.: Dialectic Proof Procedures for Assumption-based Admissible Argumentation. *Artificial Intelligence* 170(2):114-159, 2006.
30. Ebbinghaus, H.-D., Flum, J.: Finite Model Theory, 2nd edition. Springer, 2006.
31. Embury, S., Brandt, S., Robinson, J., Sutherland, I., Bisby, F., Gray, A., Jones, A., White, R.: Adapting integrity enforcement techniques for data reconciliation. Information Systems 26:657-689. Pergamon Press, 2001.
32. Enderton, H.: A Mathematical Introduction to Logic, 2nd edition. Academic Press, 2001.
33. Eiter, T., Fink, M., Greco, G., Lembo, D.: Repair localization for query answering from inconsistent databases. *ACM TODS* 33(2), article 10, 2008.
34. Furfaro, F., Greco, S., Molinaro, C.: A three-valued semantics for querying and repairing inconsistent databases. *Ann. Math. Artif. Intell.* 51(2-4):167-193, 2007.
35. Grant, J., Hunter, A.: Measuring the Good and the Bad in Inconsistent Information. *Proc. 22nd IJCAI*, 2632-2637, 2011.
36. Greco, G., Greco, S., Zumpano, E.: A logical framework for querying and repairing inconsistent databases. *IEEE TKDE* 15(6):1389-1408, 2003.
37. Guessoum, A., Lloyd, J.: Updating knowledge bases. *New Generation Computing* 8(1):71-89, 1990.
38. Guessoum, A., Lloyd, J.: Updating knowledge bases II. *New Generation Computing* 10(1):73-100, 1991.
39. Gupta, A., Sagiv, Y., Ullman, J., Widom, J.: Constraint checking with partial information. Proc. PODS'94, pp. 45-55. ACM Press, 1994.
40. Hunter, A.: Measuring Inconsistency in Knowledge via Quasi-Classical Models. Proc. 18th AAAI & 14th IAAI, 6873, 2002.

41. Hunter, A., Konieczny, S: "Approaches to measuring inconsistent information". In *Inconsistency Tolerance*, pp 191236, Springer LNCS vol. 3300, 2005.
42. Hunter, A., Konieczny, S.: Measuring inconsistency through minimal inconsistent sets, in: Brewka, G., Lang, J. (Eds.), Principles of Knowledge Representation and Reasoning (Proc. 11th KR), pp 358-366. AAAI Press, 2008.
43. Hunter, A., Konieczny, S.: On the measure of conflicts: Shapley Inconsistency Values. Artificial Intelligence vol. 174, pp. 10071026. Elsevier, 2010.
44. Kakas, A., Mancarella, P.: Database updates through abduction. In Proc. 16th VLDB, pp. 650-661. Morgan Kaufmann, 1990,
45. Kakas, A., Kowalski, R., Toni, F.: The role of Abduction in Logic Programming. In Gabbay, D., Hogger, Ch., Robinson, J. A. (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 5, pp. 235-324. Oxford University Press, 1998.
46. Lee, S. Y. Ling, T. W.: Further improvements on integrity constraint checking for stratifiable deductive databases. Proc. VLDB'96, pp. 495-505. Morgan Kaufmann, 1996.
47. Lehrer, K.: Relevant Deduction and Minimally Inconsistent Sets. Journal of Philosophy 3(2,3):153-165, 1973.
48. Mu, K., Liu, W., Jin, Z., Bell, D.: A Syntax-based Approach to Measuring the Degree of Inconsistency for Belief Bases. J. Approx. Reasoning 52(7):978-999, 2011.
49. Lloyd, J., Sonenberg, L., Topor, R.: Integrity constraint checking in stratified databases. *J. Logic Programming* 4(4):331-343, 1987.
50. Lozinskii, E.: Resolving contradictions: A plausible semantics for inconsistent systems. J. Automated Reasoning, 12(1):131, 1994.
51. Ma, Y., Qi, G., Hitzler, P.: Computing inconsistency measure based on paraconsistent semantics. J. Logic Computation 21(6):1257-1281. Oxford University Press, 2011.
52. Martinenghi, D., Christiansen, H.: Transaction management with integrity checking. Proc. 16th DEXA, pp. 606-615. Springer LNCS vol. 3588, 2005.
53. Martinenghi, D., Christiansen, H., Decker, H.: Integrity Checking and Maintenance in Relational and Deductive Databases and Beyond. In Z. Ma (Ed.): Intelligent Databases: Technologies and Applications, pp. 238-285. IGI Global, 2006.
54. Martinez, V., Pugliese, A., Simari, G., Subrahmanian, V. S., Prade, H.: How Dirty Is Your Relational Database? An Axiomatic Approach. Proc. 9th ECSQARU, pp. 103-114. Springer LNCS 4724, 2007.
55. Meyer, J., Wieringa, R. (Eds.): Deontic Logic in Computer Science. Wiley, 1994.
56. Nicolas, J.M.: Logic for improving integrity checking in relational data bases. *Acta Informatica* 18, 227-253, 1982.
57. Plexousakis, D., Mylopoulos, J.: Accommodating Integrity Constraints During Database Design. Proc. 5th EDBT, pp. 497-513. Springer LNCS 1057, 1996.
58. Rahm, E., Do, H.: Data Cleaning: Problems and Current Approaches. Data Engineering Bulletin 23(4):3-13. IEEE CS, 2000.
59. Sadri, F., Kowalski, R.: A theorem-proving approach to database integrity. In Minker, J. (Ed.), Foundations of Deductive Databases and Logic Programming, pp. 313-362. Morgan Kaufmann, 1988.
60. Thimm, M.: Measuring Inconsistency in Probabilistic Knowledge Bases. Proc. 25th UAI, pp. 530-537. AUAI Press, 2009.
61. Vardi, M.: On the integrity of databases with incomplete information. Proc. 5th PODS, pp. 252-266. ACM Press, 1986.
62. Wijsen, J.: Database repairing using updates. *ACM Trans. Database Syst.* 30(3):722-768,2005.