Detecting Errors in Numeric Attributes

Grace Fan[1] Wenfei Fan[2] Floris Geerts[3]

[1] Conestoga High School, USA
[2] University of Edinburgh, UK
[3] University of Antwerp, Belgium

Abstract. To detect errors in numeric data, this paper proposes *numeric functional dependencies* (NFDs), a class of dependencies that allow us to specify arithmetic relationships among numeric attributes. We show that NFDs subsume conditional functional dependencies (CFDs); hence, we can catch data inconsistencies, numeric or not, in a uniform logic framework by using NFDs as data quality rules. Better still, NFDs do not increase the complexity of reasoning about data quality rules. We show that the satisfiability and implication problems for NFDs remain NP-complete and coNP-complete, respectively, the same as their counterparts for CFDs. Moreover, NFDs can be implemented in SQL and hence, error detection can be readily supported by DBMS. In addition, we show that NFDs and CFDs can be extended across multiple tables, without increasing the complexity of static analyses and error detection.

1 Introduction

One of the central problems with real-life data is data consistency. Indeed, data in the real world is often dirty, with errors, conflicts and discrepancies. Inconsistent data inflicts a daunting cost. For example, it costs US businesses 600 billion dollars annually [5], and errors in medical data may have disastrous consequences such as death [18]. The scale of the problem is even worse in big data, and it poses one of the most pressing challenges to big data management.

To detect data inconsistencies, a number of dependency formalisms have been studied, such as denial constraints [3], conditional functional dependencies (CFDs) [7] and conditional inclusion dependencies (CINDs) [14]. Using these dependencies as data quality rules, several systems have been developed for detecting errors and repairing real-life data (see [6] for a recent survey).

These dependencies, however, fall short of effectively catching errors in numeric data, *e.g.,* integer and real numbers. Numeric values are widely found in, *e.g.,* medical, scientific and financial data, and should by no means be overlooked.

Example 1. (1) A relation r_1 is shown in Fig. 1(a), in which each tuple specifies a composer with his name, year of birth (YoB), year of death (YoD) and origin (country, town). The data in r_1 is, however, inconsistent: (a) Bonn is a city in Germany, not in Belgium (tuple t_2), and (b) it is normally expected that no one lives more than 120 years, *i.e.,* $\mathsf{YoD} - \mathsf{YoB} \leq 120$, in contrast to tuple t_1.

The error in tuple t_2 can be detected by a CFD ψ_1: town = "Bonn" $\rightarrow$ country = "Germany", which asserts that if the town is Bonn then the country is Germany. When ψ_1 is used as a data quality rule, the inconsistency in t_2 emerges

	name	YoB	YoD	town	country
t_1:	Wolfgang Amadeus Mozart	1756	1891	Salzburg	Austria
t_2:	Ludwig van Beethoven	1770	1827	Bonn	Belgium

(a) A composer relation r_1

	SS#	name	cno	hw	tests	lab	proj
t_3:	14311	Joe Lee	C_1	5%	35%	40%	35%
t_4:	14311	Joe Lee	C_2	15%	55%	0%	30%

(b) A report relation r_2

	cno	start	end	day
s_1:	C_1	9am	11am	Tue
s_2:	C_2	10am	11am	Tue

(c) Course relation r_2'

	CC#	name	street	city	zip	when	where	amnt
t_5:	610000253253775	Mark Smith	Main St.	Edi	EH8 9LE	21:00/7/7/2013	EDI	£350
t_6:	610000253253775	Mark Smith	Main St.	Edi	EH8 9LE	22:00/7/7/2013	NYC	£500

(d) A transaction relation r_3

	Hid	Pid	relationship	sex	age	status
t_7:	237654	1	reference	M	35	married
t_8:	237654	2	child	F	24	single

(e) A census relation r_4

Fig. 1. Example relations

as a violation of ψ_1. However, to detect the inconsistency in t_1, it requires an arithmetic operation $\mathsf{YoD} - \mathsf{YoB}$ on the numeric attributes YoB and YoD, which is, unfortunately, not supported by denial constraints, CFDs or CINDs.

(2) Relation r_2 of Fig. 1(b) shows the academic report of a high-school student, one tuple for each course taken, with the distribution of the score into homework (hw), tests, lab and projects (proj) (ignore Fig. 1(c) for now). Obviously the sum of these elements should be 100%. However, for tuple t_3, its total percentage is 115%. To detect this error, arithmetic operations are again needed.

(3) Relation r_3 of Fig. 1(d) is a sample of data from a bank. A tuple in r_3 specifies a transaction record of a credit card: the card number (CC#), card holder in the UK (name, street, city, zip), when and where the card was physically used, and the amount charged to the card (amnt). Note that tuples t_5 and t_6 indicate a possible *fraud*. Indeed, Edinburgh (Edi) is 5 hours ahead of New York city (NYC), and flights from Edi to NYC take 7 hours. Hence, we have the constraint $t_6[\mathsf{when}] - t_5[\mathsf{when}] \geq 2$, where $t_5[\mathsf{when}]$ and $t_6[\mathsf{when}]$ are the local time in Edi and NYC, respectively. That is, there is no way for one to use a card in Edinburgh at 21:00, and use the same card again in New York at 22:00 on the same day. The fraud, however, cannot be detected by the dependencies mentioned earlier.

(4) Relation r_4 of Fig. 1(e) is a piece of census data from [10]. Each household is identified by an Hid, and within each household, Pid is a key for persons. There is a *reference* person for each Hid, and every other person in the household specifies a relationship with the reference, *e.g.,* child, spouse, along with age, sex and marital status. One rule for census data is that a parent must be at least 12 years older than a child [10]. However, $t_7[\mathsf{age}] - t_8[\mathsf{age}] = 11$, where t_8 is a child of t_7. Unfortunately, the dependencies mentioned earlier are unable to detect this. □

This example tells us that to clean real-life data, we need new dependencies to detect errors in numeric attributes. Moreover, as the dependencies will be incorporated into existing data cleaning systems that support, *e.g.,* CFDs, the extension with the new dependencies should not incur substantial extra costs.

Contributions. This paper studies new dependencies in response to the need.

(1) We propose *numeric functional dependencies* (NFDs, Section 2). NFDs are defined in a QBE-like syntax [16], and support arithmetic operations. We show that CFDs are a special form of NFDs. Hence, NFDs can serve as data quality rules for detecting inconsistencies commonly found in practice, numeric or not.

(2) We show that NFDs do not increase the complexity of the static analyses of data quality rules (Section 3). Indeed, the classical problems – the satisfiability and implication problems – are NP-complete and coNP-complete for NFDs, respectively, the same as their counterparts for CFDs [7].

(3) We show that error detection with NFDs can be built on top of relational DBMS without requiring any additional functionality (Section 4). Indeed, for each NFD φ, an SQL query Q_φ can be automatically generated such that when Q_φ is evaluated on a dataset D, $Q_\varphi(D)$ returns all and only those tuples in D that violate φ, *i.e.,* data inconsistencies, in low polynomial time (PTIME).

(4) Finally, we present an extension of NFDs and CFDs (Section 5). While NFDs and CFDs are defined on a single table, we show that they can be naturally extended to span across multiple tables, in a QBE-like syntax. Better still, the extended NFDs do not make our lives harder: they retain the same complexity bounds of the static analyses and error detection as their NFD counterparts.

We contend that NFDs are a natural extension of CFDs. They can be readily employed by data cleaning systems that already support CFDs [6], and extend the capabilities of those system to detect numeric errors in real-life data.

Related work. A variety of dependencies have been studied as data quality rules for detecting data inconsistencies, from traditional functional and inclusion dependencies [2] to CFDs, CINDs and denial constraints [6]. Several data quality systems based on CFDs are already in place, and have proven effective in various applications. However, such dependencies cannot express arithmetic relationships and hence, do not effectively catch inconsistencies in numeric data.

The need for detecting numeric errors has long been recognized [4, 9–11, 13, 17]. Metric functional dependencies [13] and sequential dependencies [11] extend functional dependencies by supporting (numeric) metrics and intervals on ordered data, respectively. However, they do not support arithmetic operations and cannot capture the inconsistencies of Example 1. A class of powerful aggregation constraints was proposed in [17], defined in terms of aggregate functions (*e.g.,* max, min, sum, avg, count). Using these constraints as data quality rules, data repairing and consistent query answering were studied in [4]. It is, however, too expensive to use aggregation constraints: it is undecidable even to decide whether a given set of aggregation constraints is satisfiable. There has also been

work on repairing numeric data using constraints defined in terms of aggregate functions [9] and disjunctive logic programming [10]. These constraints are far more complicated than data quality rules that are already employed by data cleaning systems such as CFDs. The complexity of their static analyses (satisfiability and implication) is not yet known and is suspected high. In contrast to the previous work, NFDs aim to strike a balance between the complexity of their reasoning and the expressive power needed for detecting numeric inconsistencies commonly found in practice. We want NFDs to be seamlessly incorporated into data quality systems that already use CFDs, without incurring much extra cost.

It is known that the satisfiability and implication analyses of CFDs are NP-complete and coNP-complete, respectively [7]. Moreover, for each CFD ψ, two SQL queries can be automatically generated to detect all violations of ψ in a dataset [7]. We will show that NFDs retain the same complexity and property.

2 Numeric Functional Dependencies

Below we first define NFDs. We then show that CFDs are a special case of NFDs.

Numeric functional dependencies. NFDs are defined on instances of a single relation schema $R(A_1, \ldots, A_n)$. Each A_i is an attribute, with domain $\mathsf{dom}(A_i)$.

A *numeric functional dependency* φ (NFD) defined on R is a pair of tables:

(1) a *pattern table* T_p of schema R has two tuples p_1 and p_2; for $j \in [1, 2]$ and $i \in [1, n]$, $p_j[A_i]$ is a constant in $\mathsf{dom}(A_i)$, a variable $x_{(i,j)}$ or wildcard '_'; and

(2) a *condition table* T_c with a single *condition tuple* of the form e op z, where
- e is either a variable or a *linear* arithmetic expression;
- op is one of the built-in predicates $=, \neq, <, \leq, >, \geq$; and
- z is either a constant c or a variable y in T_p.

Here an *arithmetic expression* is built up from terms of numeric constants c or variables x in T_p with a numeric domain, by closing them under arithmetic operators $+, -, \times, \div$ and $|\cdot|$ (for absolute value). Note that variables $x_{(i,j)}$ and $x_{(l,s)}$ that appear in pattern tuples p_1 and p_2 of T_p may be identical, asserting condition $p_j[A_i] = p_s[A_l]$. When $p_j[A_i]$ is a constant c, it denotes condition $p_j[A_i] = c$. If p_1 and p_2 are identical, T_p consists of a single tuple p_1 only.

Example 2. (1) An NFD $\varphi_1 = (T_{P1}, T_{C1})$ is defined on the composer relation of Fig. 1(a). As shown in Figures 2(a) and 2(b), T_{P1} consists of a single pattern tuple t_{p1} with $t_{p1}[\mathsf{YoB}] = x$ and $t_{p1}[\mathsf{YoD}] = y$, and T_{P1} consists of a single condition. It is to ensure that for any composer tuple t, $t[\mathsf{YoD}] - t[\mathsf{YoB}] \leq 120$.

(2) Figures 2(c) and 2(d) define an NFD $\varphi_2 = (T_{P2}, T_{C2})$ on the academic report relation of Fig. 1(b). It states that for any report tuple t, the sum $t[\mathsf{hw}]+t[\mathsf{tests}]+t[\mathsf{lab}]+t[\mathsf{proj}]$ of the distribution of the marks should be equal to 100%.

(3) Another NFD $\varphi_3 = (T_{P3}, T_{C3})$ is given in Figures 2(e) and 2(f), defined on the transaction data of Fig. 1(d). Note that pattern table T_{P3} consists of two tuples. The NFD states that for any two transaction records of the same credit card (specified by $p_1[\mathsf{CC\#}] = x_c$ and $p_2[\mathsf{CC\#}] = x_c$), if it was used in Edi and NYC,

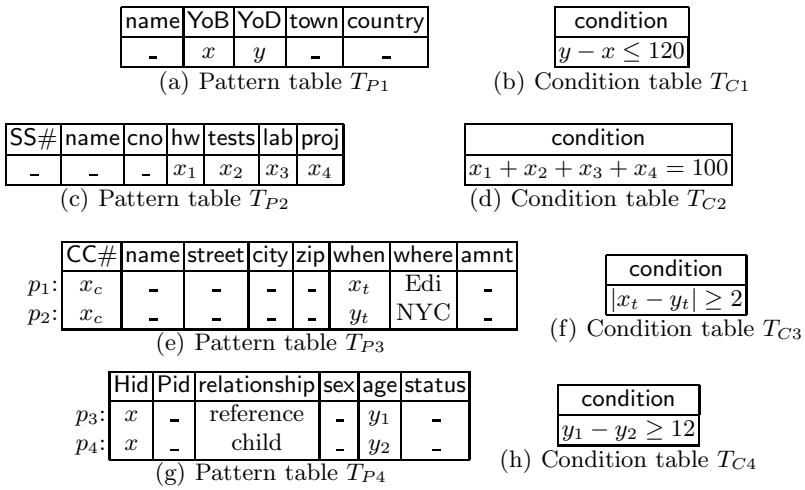

Fig. 2. Example NFDs

then the two transactions had to be at least two-hour apart. Note that constants "Edi" and "NYC" are used to specify a pattern, along the same lines as CFDs [7]. In addition, variable x_c is used to enforce condition $p_1[\mathsf{CC\#}] = p_2[\mathsf{CC\#}]$.

(4) Finally, an NFDs $\varphi_4 = (T_{P4}, T_{C4})$ is given in Figures 2(g) and 2(h), defined on the census relation of Fig. 1(e). It assures that for any two persons in the same household ($p_3[\mathsf{hid}]) = x$ and $p_4[\mathsf{hid}] = x$), if one is the "reference" and the other is his/her "child", then the parent must be at least 12-years older than the child. Again, constants "reference" and "child" specify patterns. □

Semantics. To give a formal semantics of NFDs, we use an operator $\trianglerighteq$ defined on constants, variables and '_': $\eta_1 \trianglerighteq \eta_2$ is interpreted as the truth value true if η_2 is '_', and it is an equality predicate $\eta_1 = \eta_2$ otherwise. The operator $\trianglerighteq$ naturally extends to tuples and produces true or a *conjunction of equality atoms*, *e.g.,* (Main St, EDI, NYC) $\trianglerighteq$ (_, x, y) yields 'Edi' = $x \wedge$ 'NYC' = y.

Given $\trianglerighteq$, an NFD $\varphi = (T_P, T_C)$ in the QBE-like syntax [16] given above can be rewritten into an equivalent first-order logic (FO) sentence as follows. Assume that T_p has two pattern tuples p_1 and p_2, T_c is e op z, and that all the variables appearing in T_P are $x_1, \ldots, x_m$. Then φ can be written as the FO sentence:

$$\varphi = \forall t_1, t_2,\ x_1, \ldots, x_m\big((t_1 \trianglerighteq p_1 \wedge t_2 \trianglerighteq p_2) \rightarrow e\ \mathsf{op}\ z\big).$$

As remarked earlier, $t_1 \trianglerighteq p_1$ and $t_2 \trianglerighteq p_2$ yield two conjunctions ω_1 and ω_2, possibly equal to true. The variables x_i are specified in $t_j \trianglerighteq p_j$ and used in e and z.

An instance D of relation schema R *satisfies* the NFD φ, denoted by $D \models \varphi$, if for *all* tuples t_1 and t_2, if t_1 and t_2 satisfy $\omega_1 \wedge \omega_2$ following the standard semantics of first-order logic, then the condition e op z is also satisfied. We say that D *satisfies* a set Σ of NFDs, denoted by $D \models \Sigma$, if for all $\varphi \in \Sigma$, $D \models \varphi$.

Intuitively, if tuples t_1 and t_2 match pattern tuples p_1 and p_2, respectively, then the predicate e op z defined with arithmetic operations in e and the comparison operation op has to be satisfied. Observe that pattern tableau T_P supports

patterns of semantically related values such as "$t[\mathsf{where}]$ = "Edi" like CFDs [7]. Moreover, they also enforce equality by using variables, *e.g.,* $t_1[\mathsf{CC\#}] = x_c$ and $t_2[\mathsf{CC\#}] = x_c$ entail that $t_1[\mathsf{CC\#}] = t_2[\mathsf{CC\#}]$. In contrast to traditional FDs that are defined on all tuples in D (see, *e.g.,* [2]), the NFD φ is applicable only to the *subset* of tuples in D that match patterns p_1 and p_2. Note that when T_P consists of a single tuple p only, it is equivalent to two identical patterns $p_1 = p_2 = p$, and hence the semantics given above is also well-defined in this case.

Example 3. Consider NFD $\varphi_1 = (T_{P1}, T_{C1})$ given in Figures 2(a) and 2(b). It is interpreted as $\forall\, t, x, y\ \big(t[\mathsf{YoB}] = x \wedge t[\mathsf{YoD}] = y) \rightarrow (y - x \leq 120)\big)$.

The NFD $\varphi_3 = (T_{P3}, T_{C3})$ given in Figures 2(e) and 2(f) is interpreted as

$$\forall\, t_1, t_2, x_c, x_t, y_t\ \big(t_1[\mathsf{CC\#}] = x_c \wedge t_2[\mathsf{CC\#}] = x_c \wedge t_1[\mathsf{where}] = \text{"Edi"}$$
$$\wedge\, t_2[\mathsf{where}] = \text{"NYC"} \wedge t_1[\mathsf{when}] = x_t \wedge t_2[\mathsf{when}] = y_t) \rightarrow (|y_t - x_t| \geq 2)\big).$$

which is in turn equivalent to $\forall\, t_1, t_2\big(t_1[\mathsf{CC\#}] = t_2[\mathsf{CC\#}] \wedge t_1[\mathsf{where}] = \text{"Edi"} \wedge t_2[\mathsf{where}] = \text{"NYC"} \rightarrow (|t_2[\mathsf{when}] - t_1[\mathsf{when}]| \geq 2)\big)$. The semantics of φ_2 and φ_3 given in Example 2 can be similarly interpreted in first-order logic. □

Taking NFDs and CFDs together. Recall that CFDs on schema R can be written in a normal form $\psi = (X \rightarrow A,\ t_p)$ [6], where (a) X is a set of attributes of R, A is a single attribute of R, and (b) t_p is a *pattern tuple* with attributes in X and A such that for each $B \in X \cup \{A\}$, $t_p[B]$ is either a constant in $\mathsf{dom}(B)$ or a wildcard '_'. An instance D of R satisfies the CFD ψ, denoted by $D \models \psi$, if for any two tuples $t_1, t_2 \in D$, when $t_1[X] \trianglerighteq t_p[X]$ and $t_2[X] \trianglerighteq t_p[X]$, then $t_1[A] = t_2[A] \trianglerighteq t_p[A]$ (here CFDs restrict "$\trianglerighteq$" to constants and '_' only).

One can readily verify that ψ is equivalent to an NFD φ of the following form.

(1) If $t_p[A]$ is a constant c, then the NFD $\varphi = (T_p, T_c)$, where T_p consists of a single pattern tuple p such that (a) $p[A]$ is a distinct variable, and (b) for all the other attributes B of R, $p[B] = t_p[B]$ if $B \in X$, and $p[B]$ = '_' otherwise; and (c) T_c is $x_A = c$. Such a CFD is referred to as a *constant* CFD in [6].

(2) If $t_p[A]$ is wildcard '_', then $\varphi = (T_p, T_c)$, where T_p consists of two pattern tuples p_1 and p_2, such that (a) $p_1[A]$ and $p_2[A]$ are distinct variables x_1 and x_2, respectively; (b) for all the other attributes B of R, $p_1[B]$ and $p_2[B]$ are defined in the same way as for constant CFDs; and (c) T_c is $x_1 = x_2$. Such CFDs are called *variable* CFDs [6], which subsume traditional functional dependencies [2].

Example 4. The constraint ψ_1 given in Example 1 is a constant CFD, and can be expressed as an equivalent NFD $\varphi_0 = (T_{P0}, T_{C0})$, where

$T_{P0} =$

name	YoB	YoD	town	country
_	_	_	Bonn	x

$T_{C0} =$

condition
x = Germany

□

Based on the discussions above, one can readily verify the following.

Proposition 1: CFDs of [7] are a special cases of NFDs. □

This tells us that NFDs provide us with a uniform logic formalism to express data quality rules for detecting data inconsistencies, numeric or not. In other words, NFDs are capable of capturing all errors that CFDs can catch, and moreover, errors in numeric attributes that CFDs are not able to detect.

3 Reasoning about NFDs

NFDs are strictly more expressive than CFDs. Indeed, Proposition 1 tells us that NFDs subsume CFDs; moreover, NFDs can specify arithmetic relationships among numeric attributes such as those in φ_1–φ_4, which CFDs cannot express. Despite the increase in expressive power, we next show that NFDs do not increase the complexity of reasoning about data quality rules. More specifically, we study two classical problems that are associated with any dependency class C.

Consider a set Σ of dependencies in C defined on a relation schema R. To use Σ as data quality rules, we have to answer the following questions.

- The *satisfiability problem* for C is to decide, given a set Σ of dependencies in C, whether there exists a nonempty instance D of R such that $D \models \Sigma$.
- The *implication problem* for C is to determine, given Σ and another dependency φ in C that is also defined on R, whether Σ *implies* φ, denoted by $\Sigma \models \varphi$, *i.e.,* for each instance D of R, if $D \models \Sigma$, then $D \models \varphi$.

The satisfiability analysis checks whether Σ has conflicts, *i.e.,* whether the data quality rules in Σ are "dirty" themselves. It help us find out what goes wrong in the rules. The implication analysis allows us to optimize our rules by eliminating redundant ones: if $\Sigma \models \varphi$, then it suffices to use Σ instead of $\Sigma \cup \{\varphi\}$.

When C is the class of traditional functional dependencies, any finite subset Σ of C is satisfiable, and its implication problem is in linear-time [2]. When it comes to NFDs, however, these problems are no longer simple.

To understand where the complication arises, we consider a special form of NFDs, denoted by AFDs, in which conditions have the form of e op c, where e is a linear arithmetic expressions defined on numeric attributes, and c is a constant. One can show that AFDs cannot express CFDs (not even constant CFDs $(X \to A, t_p)$ when A is a non-numeric attribute), and that CFDs cannot express AFDs. We show below that there exists a set of AFDs that is not satisfiable, even when all the attributes in R have an infinite domain. In contrast, for a set of CFDs to be unsatisfiable, the CFDs must be defined on some attributes with a finite domain (see [7] for more details).

Example 5. Consider a relation schema $R(A, B)$, where A and B have an infinite integer domain. Let Σ be a set consisting of two AFDs $\phi_1 = (T_P, T'_{C1})$ and $\phi_2 = (T_P, T'_{C2})$, where T_P consists of a single tuple (x, y), the condition of T'_{C1} is $x - y = 0$, and T'_{C2} is $y - x > 0$. Then there exists no nonempty instance D of R such that $D \models \Sigma$. Indeed, for any instance D of R, if there exists a tuple $t \in D$, then ϕ_1 requires $t[A] = t[B]$ whereas ϕ_2 asks for $t[A] \neq t[B]$. □

The good news is that the extra expressive power introduced by NFDs over CFDs does not make our lives harder: their static analyses have the same complexity as their counterparts for CFDs [7], and as well as for AFDs.

Theorem 2. *For NFDs, (1) the satisfiability problem is* NP*-complete, and (2) the implication problem is* coNP*-complete. For AFDs, (3) the satisfiability and implication problems remain* NP*-complete and* coNP*-complete, respectively.* □

Proof Sketch. (1) NFDs. The lower bounds follow from their counterparts for CFDs [7], since CFDs are a special case of NFDs. The upper-bound proofs are far more involved. To see the complications, observe that if arbitrary arithmetic expressions were allowed in NFDs, the satisfiability problem would be undecidable, which could be verified by reduction from undecidable Diophantine equations [12]. That is why we restrict arithmetic expressions in NFDs to be linear.

For the satisfiability problem, we first show that NFDs have a small model property: given a set Σ of NFDs defined on a schema R, if there exists a nonempty instance of R that satisfies Σ, then there exists an instance D_0 of R such that $D_0 \models \Sigma$, D_0 consists of a single tuple t_0, and moreover, it suffices to consider the values of the attributes of t_0 from a finite domain determined by the domains of the attributes and those constants in Σ. The proof of the small model property is nontrivial and needs to distinguish integers and non-integers. Based on the small model property, one can develop an NP algorithm for satisfiability checking as follows: (a) guess a tuple t_0 with values drawn from the finite domain, and (b) check whether $\{t_0\} \models \Sigma$. The algorithm is in NP since step (b) is in PTIME.

Similarly, the implication problem for NFDs is shown in coNP by establishing a small model property, where D_0 consists of two tuples.

(b) AFDs. The upper bounds follow from their counterparts for NFDs, since AFDs are a special case of NFDs. The lower bound for the satisfiability problem is verified by reduction from the linear integer programming problem (LIP), which is NP-complete (cf. [15]). The latter problem is to determine whether a set of linear inequalities has an integer solution. Similarly, the implication problem for AFDs is verified coNP-complete by reduction from the complement of LIP. □

4 Validating NFDs in SQL

Recall that we introduce NFDs to detect data inconsistencies. We next present an SQL technique for relational DBMS to detect inconsistencies as violation of NFDs.

Consider a set Σ of NFDs defined on a relation schema R, an instance D of R, and an NFD $\varphi = (T_p, T_c)$ in Σ, where T_p consists of two pattern tuples p_1 and p_2, and T_c is a condition e op z (see Section 2 for the definition of NFDs). We say that a tuple $t \in D$ is a *violation* of φ if there exists a tuple $t' \in D$ such that $\{t, t'\} \not\models \varphi$, *i.e.,* either $t \trianglerighteq p_1$ and $t' \trianglerighteq p_2$ or $t' \trianglerighteq p_1$ and $t \trianglerighteq p_2$, but $\neg(e \text{ op } z)$. Here we use $\neg(e \text{ op } z)$ to denote that the condition e op z is not satisfied. For instance, if op is '=', then $\neg(e \text{ op } z)$ is $e \neq z$, and if op is '$\leq$', then $\neg(e \text{ op } z)$ is $e > z$. Note that violations of φ are defined on single tuples.

The *error detection problem* can then be stated as follows. Given Σ and D, it is to find the set of all tuples in D that are violations of some NFD in Σ, denoted by $\mathsf{vio}(\Sigma, D)$; *i.e.,* $\mathsf{vio}(\Sigma, D) = \{t \in D \mid \exists \varphi \in \Sigma, t \text{ is a violation of } \varphi\}$.

The main result of this section is as follows.

Proposition 3: (1) For any NFD φ defined on a schema R, there exists an SQL query Q_φ such that for any instance D of R, $Q_\varphi(D)$ returns all violations of φ in D. (2) For any set Σ of NFDs defined on R and any instance D of R, $\mathsf{vio}(\Sigma, D)$

is computable in at most $O(\|\Sigma\||D|^2)$ time, where $\|\Sigma\|$ is the cardinality of Σ (*i.e.,* the number of NFDs in Σ), and $|D|$ is the size of D. □

To prove Proposition 3, we show how Q_φ is (automatically) generated from φ. Assume that $\varphi = (T_p, T_c)$, $T_p = \{p_1, p_2\}$, and T_c is condition e op z. Then Q_φ is:

select t
from $R\ t,\ R\ t'$
where $(\Omega_1(t, t', p_1, p_2)$ **and** $\mathsf{C}_1(t, t', e, z))$ **or** $(\Omega_2(t, t', p_1, p_2)$ **and** $\mathsf{C}_2(t, t', e, z))$

Here $\Omega_1(t, t', p_1, p_2)$ encodes $t \trianglerighteq p_1$ and $t' \trianglerighteq p_2$ in SQL, as a conjunction of *terms* such that for each attribute A of R, (1) $t[A] = c$ is a term if $p_1[A]$ is a constant c; similarly for $t'[A] = c$; and (2) $t[A] = t'[A]$ is a term if $p_1[A]$ and $p_2[A]$ are the same variable x_A. The conjunct $\mathsf{C}_1(t, t', e, z)$ encodes $\neg(e$ op $z)$ as described above, by substituting attributes of t and t' for variables in e or z; more specifically, each variable x occurring in e or z is replaced with $t[A]$ if $p_1[A] = x$ (resp. with $t'[A]$ if $p_2[A] = x$). This is possible since SQL supports arithmetic operations (see, *e.g.,* [1]). Along the same lines, $\Omega_2(t, t', p_1, p_2)$ encodes $t \trianglerighteq p_2$ and $t' \trianglerighteq p_1$ in SQL, and $\mathsf{C}_2(t, t', e, z)$ encodes $\neg(e$ op $z)$ accordingly.

Example 6. The NFD φ_1 given in Example 2 can be implemented in SQL as Q_1:

select t
from composer t
where t[YoD] $-$ t[YoB] > 120

Here we need a single variable t to range over composer tuples in this simple SQL query, since T_{P1} has a single pattern tuple. Similarly, φ_4 can be validated by Q_4:

select t
from census t, census t',
where t[Hid]=t'[Hid] **and**
(t[relationship]="reference" **and** t'[relationship] ="child" **and** t[age] $-$ t'[age] < 12) **or**
(t[relationship] = "child" **and** t'[relationship] = "reference" **and** t'[age] $-$ t[age] < 12)

In contrast to Q_1, the SQL query Q_4 uses two variables t and t' to range over census tuples, since T_{P4} consists of two pattern tuples. Similarly, one can get SQL queries to validate the NFDs φ_2 and φ_3 given in Example 2.

As another example, recall the NFD φ_0 of Example 4 for expressing the CFD ψ_1; this NFD can be validated by using the following SQL query Q_0:

select t
from composer t
where t[town] = "Bonn" **and** t[country] $\neq$ "Germany" □

One can readily verify that given any instance D of R, it takes $O(|D|^2)$ time to evaluate $Q_\varphi(D)$ in the worst case, to compute all violations of the NFD φ in D (note that $|\varphi|$ is determined by the arity of R). From this it follows that to validate a set Σ of NFDs, it takes at most $O(\|\Sigma\||D|^2)$ time. The cost can be substantially reduced via, *e.g.,* indexing. This completes the proof of Proposition 3.

Remark. Proposition 3 tells us that one can support inconsistency detection based on NFDs directly on top of commercial DBMS, without requiring any additional functionality. We conclude this section with the following remarks.

(1) To detect violations of a CFD ψ in a database, an SQL method has been presented in [7], which requires two SQL queries in general. In contrast, we show that a single SQL query Q_φ suffices to validate an NFD φ, although NFDs are more expressive than CFDs. Nonetheless, the SQL queries in [7] are in $O(|\psi||D|)$ time, where $|\psi|$ denotes the size of ψ, whereas the SQL query Q_φ takes $O(|D|^2)$ time (although the cost can be reduced as mentioned above).

(2) After we have seen the static analyses and validation of NFDs, we now justify the definition of NFDs (Section 2). One might be tempted to extend NFDs by allowing (a) non-linear arithmetic expressions in T_c, or (b) an unbounded number of pattern tuples in T_p instead of at most two. However, either extension would lead to substantial increase in the complexity. Indeed, (a) non-linear arithmetic expressions in conditions would make the satisfiability and implication analyses of NFDs undecidable, as shown in the proof of Theorem 2; and (b) an unbounded number of pattern tuples would require exponential time to validate an NFD φ; more specifically, it would take $O(|D|^n)$ time to validate φ in the worst case, where n is the number of pattern tuples of φ. As remarked earlier, we want to strike a balance between the expressive power and the complexity of NFDs. The current definition of NFDs, on one hand, suffices to catch (numeric) errors commonly found in the real world, and on the other hand, extends CFDs without increasing the complexity of the static analyses of data quality rules.

5 Extending NFDs and CFDs across Multiple Tables

Both NFDs and CFDs are defined on a single relation schema R. We next show that they can be readily extended to detect data inconsistencies across multiple tables, without increasing the complexity of their validation and reasoning.

To illustrate the need for such an extension, consider the following example.

Example 7. Consider the report relation r_2 of Fig. 1(b) and course relation r_2' of Fig. 1(c). Suppose that the inconsistency in tuple t_1 of r_2 is fixed by the NFD φ_2 of Example 2. Then each of the relations r_2 and r_2', when taken separately, seems consistent. However, when r_2 and r_2' are put together, inconsistencies emerge: course $\mathsf{C_1}$ specified by tuple s_1 of r_2' and course $\mathsf{C_2}$ given by tuple s_2 overlap with each other for an hour; this accounts for *a conflict* when some student takes both courses, which is witnessed by tuples t_3 and t_4 in relation r_2. □

Such a conflict cannot be directly captured by NFDs or CFDs that are defined on a single table. Moreover, it cannot be detected by CINDs [14] although CINDs are defined on two tables. To catch this, one may want to compute the natural join r_2'' of r_2 and r_2' on attribute cno and then define an NFD on r_2''. This is doable, but costly: to detect such inconsistencies one has to compute a number of joins.

Extended NFDs. This motivates us to extend NFDs across multiple tables, so that we can directly catch data inconsistencies between these tables.

Consider a relational schema $\mathcal{R}$, which is a collection $(R_1, \ldots, R_m)$ of relation schemas. Let $k \geq 2$ be a predefined natural number (a constant). We define an *extended* NFD φ on relational schema $\mathcal{R}$ to be a pair of $(\overline{T_p}, T_c)$, where

(1) $\overline{T_p}$ is a set of k pattern tables defined on k relation schemas of $\mathcal{R}$; each T_p of $\overline{T_p}$ is a pattern table of a schema R of $\mathcal{R}$, consisting of two pattern tuples p_1 and p_2 defined with constants, variables and wildcard '_' as before; and

(2) T_c is the *condition table* of φ with a tuple that is either (a) e op z as before, or (b) a *Boolean expression* e defined with terms of comparison predicates $(=, \neq, \leq, <, \geq, >)$ on constants and variables, by closing them under $\wedge, \vee$ and $\neg$. Here e and z may include variables from any pattern table T_p of $\overline{T_p}$.

The semantics of extended NFDs is defined along the same lines as NFDs.

Example 8. We define an extended NFDs φ_e across report and course relations:

SS#	name	cno	hw	tests	lab	proj
x	_	y_1	_	_	_	_
x	_	y_2	_	_	_	_

T_{Pr}

cno	start	end	day
y_1	z_1	w_1	d
y_2	z_2	w_2	d

T_{Pc}

condition
$(z_1 > w_2) \vee (z_2 > w_1)$

T_C

It asserts that for any two courses y_1 and y_2, (a) if there exists a student x taking both (pattern table T_{Pr}), and (b) if the two courses are on the same day d (table T_{Pc}), then they do not overlap (condition T_C). To ensure this, we also use an NFD to assert that for any course tuple s, $s[\mathsf{start}] < s[\mathsf{end}]$ (omitted). □

Validating and reasoning about extended NFDs. Extended NFDs do not increase the complexity of validation and static analyses of data quality rules. Along the same lines as the argument of Section 4, one can verify the following.

Corollary 4: (1) For any extended NFD φ defined on a relational schema $\mathcal{R}$, there exists an SQL query Q_φ such that for any instance $\mathcal{D}$ of $\mathcal{R}$, $Q_\varphi(\mathcal{D})$ finds all violations of φ in $\mathcal{D}$. (2) For any set Σ of extended NFDs on $\mathcal{R}$ and instance $\mathcal{D}$ of $\mathcal{R}$, vio$(\Sigma, \mathcal{D})$ can be computed in $O(|\Sigma||\mathcal{D}|^2)$ time, where $|\Sigma|$ is the size of Σ. □

We should remark that Corollary 4 would no longer hold if one would allow either arbitrary number of pattern tables (*i.e.,* without the constant bound k) or unbounded number of tuples in a pattern table in extended NFDs.

Extending the proof of Theorem 2, one can verify the following corollary, in which *extended* AFDs refer to the subclass of extended NFDs in which the conditions are linear arithmetic expressions e op c for some constant c.

Corollary 5: For extended NFDs and for extended AFDs, (1) the satisfiability problem is NP-complete, and (2) the implication problem is coNP-complete. □

6 Conclusion

We have proposed NFDs and shown the following. (1) NFDs extend CFDs [7] and are capable of detecting inconsistencies in numeric attributes. (2) Despite the increased expressive power, NFDs do not increase the complexity of the satisfiability and implication analyses of data quality rules. (3) Better still, NFDs allow us to detect errors in low PTIME by using existing relational DBMS, by means of automatically generated SQL queries. (4) In addition, NFDs (and hence CFDs) can be extended to catch inconsistencies across different tables,

without incurring substantial extra overhead. In light of these, we suggest to use NFDs to detect errors, numeric or not, in a uniform logic framework.

Several topics are targeted for future work. (1) It is known that CFDs are finitely axiomatizable [7]: there exists a finite set of axioms for the implication analysis of CFDs. The finite axiomatizability of NFDs remains to be investigated. (2) To make practical use of NFDs, algorithms need to be developed for automatically discovering NFDs from (possibly dirty) data, along the same lines as discovery algorithms for CFDs (*e.g.,* [8]). (3) To simplify the discussion, we prove Proposition 3 by using a separate SQL query for each NFD in a given set Σ of NFDs (Section 4). It is possible to find a fixed number of SQL queries for error detection, regardless of the cardinality of Σ, along the same lines as CFDs [7]. (4) Finally, data repairing algorithms based on NFDs should be in place to fix errors detected by NFDs (see [6] for data repairing based on CFDs).

Acknowledgments. Wenfei Fan is supported in part by NSFC 61133002, 973 Program 2012CB316200, Guangdong Innovative Research Team Program 2011D005 and Shenzhen Peacock Program 1105100030834361, China.

References

1. MySQL. *http://dev.mysql.com/doc/refman/5.0/en/func-op-summary-ref.html*.
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
3. M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. *TPLP*, 3(4-5), 2003.
4. L. E. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. Complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Inf. Syst.*, 33(4-5):407–434, 2008.
5. W. W. Eckerson. Data Quality and the Bottom Line: Achieving Business Success through a Commitment to High Quality Data. In *The Data Warehousing Institute*, 2002.
6. W. Fan and F. Geerts. *Foundations of Data Quality Management*. Morgan & Claypool Publishers, 2012.
7. W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(1), 2008.
8. W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *TKDE*, 23(5):683–698, 2011.
9. S. Flesca, F. Furfaro, and F. Parisi. Querying and repairing inconsistent numerical databases. *TODS*, 35(2), 2010.
10. E. Franconi, A. L. Palma, N. Leone, S. Perri, and F. Scarcello. Census data repair: a challenging application of disjunctive logic programming. In *LPAR*, 2001.
11. L. Golab, H. J. Karloff, F. Korn, A. Saha, and D. Srivastava. Sequential dependencies. *PVLDB*, 21(1), 2009.
12. J. P. Jones. Undecidable Diophantine equations. *Bull. Amer. Math. Soc.*, 3(2):859–862, 1980.
13. N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian. Metric functional dependencies. In *ICDE*, 2009.
14. S. Ma, W. Fan, and L. Bravo. Extending inclusion dependencies with conditions. *TCS*, 515:64–95, 2014.
15. C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
16. R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill Higher Education, 2000.
17. K. A. Rossa, D. Srivastava, P. J. Stuckeyc, and S. Sudarshan. Foundations of aggregation constraints. *TCS*, 193(1-2):149179, 1998.
18. The New York Times. Articles about Jesica Santillan. *http://topics.nytimes.com/topics/reference/timestopics/people/s/jesic_santillan/index.html*.